Projeto 2 - MC833 - Programação em Rede de Computadores

Prof. Dr. Edmundo Roberto Mauro Madeira

Eduardo Moreira Freitas de Souza - RA: 166779

1. Introdução

O objetivo deste projeto foi o desenvolvimento de uma aplicação cliente-servidor, com comunicação pela rede utilizando o protocolo UDP, a fim de comparar com uma implementação concorrente de uma outra aplicação cliente-servidor utilizando o protocolo TCP.

O servidor armazena um banco de dados com entradas geradas aleatoriamente, que representam pessoas com as seguintes informações:

- Email;
- Senha de acesso;
- Nome completo;
- Residência;
- Formação Acadêmica;
- Habilidades;
- Experiência;

2. Servidor UDP

O servidor foi feito na linguagem C, com acesso ao banco de dados MongoDB[1] através da biblioteca MongoDB C Driver[2]. Assim que recebe um pacote de um cliente, armazena as informações dele para que possa enviar a resposta. Em seguida, realiza a consulta no banco de dados, retornando todas as informações da pessoa requisitada pelo seu email, caso exista e esteja armazenada neste.

Como no protocolo UDP não é necessário que haja uma conexão formal entre o cliente e o servidor, o cliente inicia a comunicação enviando uma mensagem para o servidor contendo apenas o email para obter as informações necessárias e espera uma resposta, encerrando a comunicação naquela estância do cliente quando a recebe. Caso haja múltiplas requisições simultâneas, trata elas iterativamente por ordem de chegada.

3. Servidor TCP

O servidor foi feito na linguagem C, com acesso ao banco de dados MongoDB[1] através da biblioteca MongoDB C Driver[2]. Assim que recebe uma requisição de um cliente para conectar, executa um fork() - para lidar de forma exclusiva e concorrente, em relação a outras possíveis conexões. Conectado, espera um pacote do cliente, contendo o email desejado e, em seguida, realiza a consulta no banco de dados, retornando todas as informações da pessoa requisitada pelo seu email, caso exista e esteja armazenada neste.

O servidor, em Linux, utiliza de *file descriptors* de *sockets* para realizar a comunicação com o cliente e vice-e-versa, encapsulando funções prontas de envio de

mensagens para as camadas inferiores à de Aplicação, utilizando-se de TCP/IPV4 ou IPV6, quando necessário.

4. Estrutura de dados armazenados no servidor

A estrutura de dados do MongoDB foi montada através de um docker-compose, que popula o banco com um arquivo .json, no seguinte formato:

```
[
     {
     'Nome Completo' : { 'nome' : 'Lucas', 'sobrenome' : 'Teixeira
Amaral' },
     'Email' : 'lucas.amaral@globo.com',
     'senha' : 'plaintext1',
     'Residencia' : 'Pereiras',
     'Formacao Academica' : 'Psicologia',
     'Habilidades' : 'Parapsicologia',
     'Experiencia' : [
          {
               'Local' : 'Sefit',
               'Cargo' : 'Estagiario'
          },
          {
               'Local' : 'Bolotus',
               'Cargo' : 'Psicologo'
          }
     ]
     },
     …
]
```

O cliente recebe um objeto json, fruto da *query* requisitada pela sua operação, como:

```
Escreva o email que deseja saber: lucas.amaral@globo.com
{ "Email" : "lucas.amaral@globo.com", "senha" : "plaintext1",
"Residencia" : "Pereiras", "Formacao Academica" : "Psicologia",
"Habilidades" : "Parapsicologia", "Experiencia" : [ { "Local" :
"Sefit", "Cargo" : "Estagiario" }, { "Local" : "Bolotus", "Cargo" :
"Psicologo" } ] }
```

Quanto a implementação da mensagem a ser enviada, esta é reunida em um único buffer de tamanho elástico, que logo em seguida é concatenada com a string que representa o tamanho da mensagem a ser enviada, em bytes, para ser tratada mais facilmente pelo lado do cliente, como o exemplo abaixo:

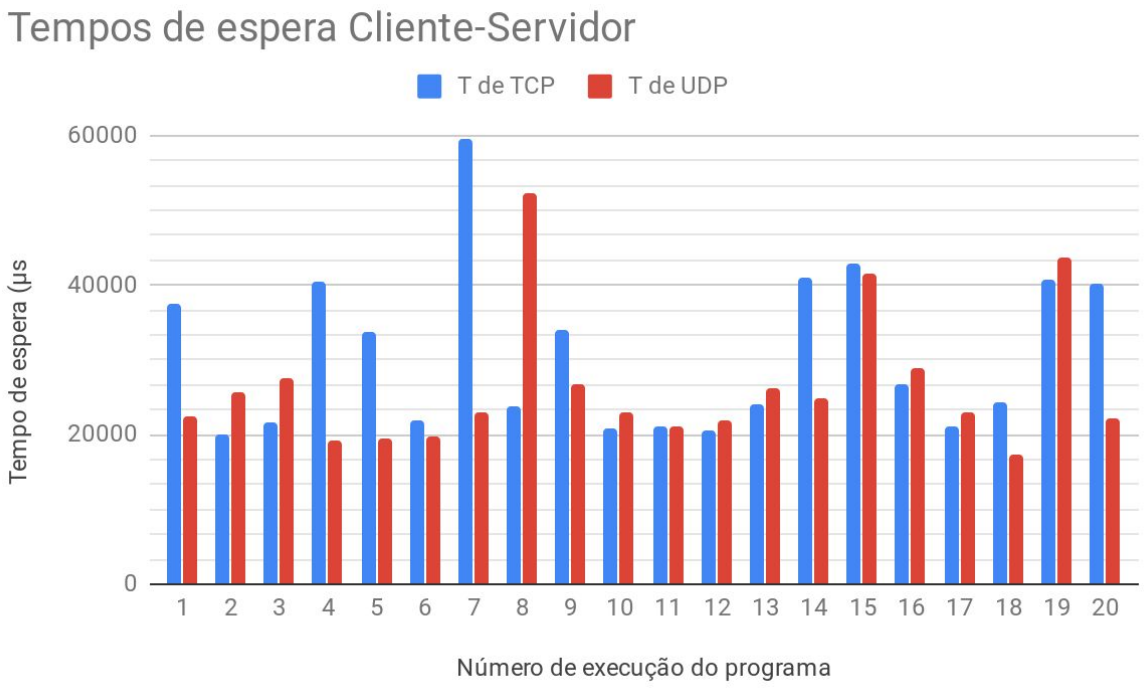
```
“35\nEssa eh uma mensagem de exemplo!”
```

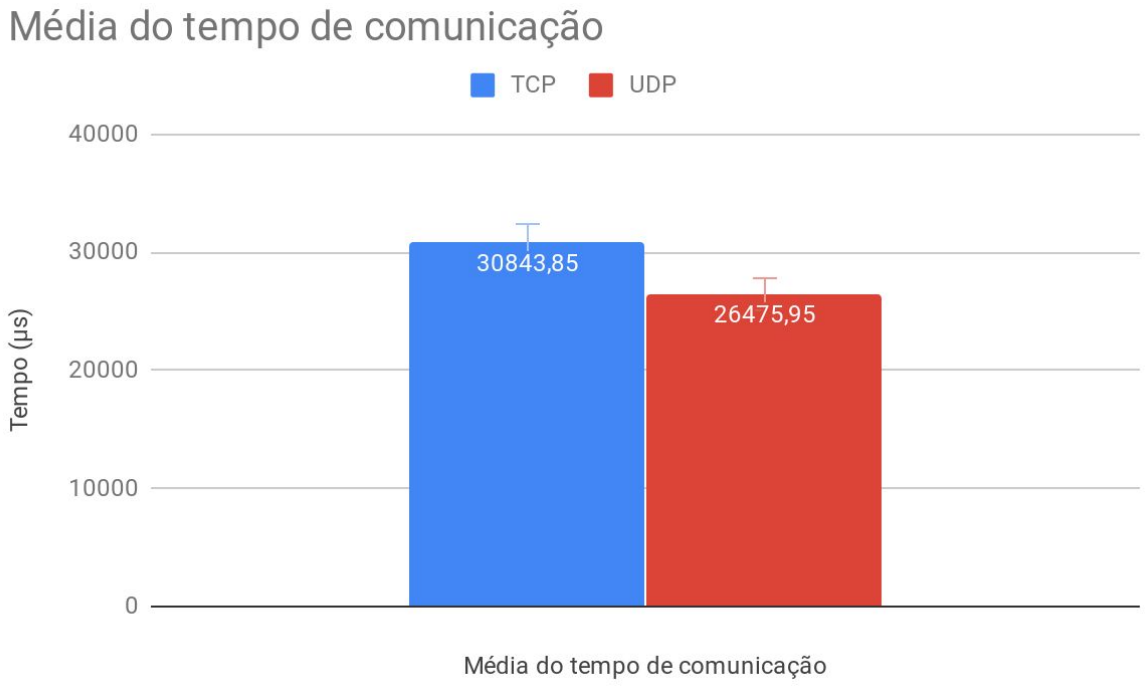
Apesar do buffer ser elástico, convencionou-se que o tamanho máximo da mensagem a ser enviada por pacote foi de 1024 bytes, ou seja, se uma mensagem possui 2048 bytes, esta é dividida em duas chamadas da função *sendto()*.

5. Tempo

O tempo foi coletado tanto por parte do servidor quanto por parte do cliente, em etapas distintas do fluxo do programa.

No servidor, o primeiro tempo salvo é logo após que recebe a mensagem, e o segundo, logo quando envia a mensagem - ou seja, o tempo de processamento dos dados para o envio. No cliente, o tempo é contabilizado entre enviar a mensagem e receber uma resposta, ou seja, o tempo de espera do processamento e do envio do pacote - em ambos os sentidos. Abaixo segue o tempo de cada execução da consulta, tanto em TCP quanto em UDP e as médias dos tempos em microsegundos das operações no servidor e no cliente, com máquinas em redes diferentes:





6. Confiabilidade

Apesar do protocolo UDP não garantir o envio nem o recebimento dos pacotes por ambos os lados, este não apresentou perda de mensagens em nenhum momento durante o desenvolvimento da aplicação. Porém é recomendável implementar uma função de *timeout*, que, dado a falta de resposta após um determinado tempo depois do envio da mensagem, a envia novamente, visto que esta deve ter sido perdida entre o cliente e o servidor.

7. Conclusão

O desenvolvimento da aplicação cliente-servidor utilizando o protocolo UDP tornou possível a comparação deste com a mesma aplicação utilizando o protocolo TCP. Como é possível deduzir pelos gráficos, o protocolo UDP tem suas mensagens enviadas um pouco mais rápido do que comparando com o TCP. Entretanto, durante a decisão de qual deve ser utilizado, deve se levar em consideração a confiabilidade de cada um: em uma aplicação como aqui desenvolvida, onde as mensagens recebidas e enviadas devem chegar de forma integral no cliente e no servidor, o protocolo TCP garantirá isso, diferente do UDP, que seria necessário um tratamento para perdas de parte de pacotes a parte do protocolo já implementado pelo sistema operacional.

8. Bibliografia

[1] https://www.mongodb.com/
[2] http://mongoc.org/
[3] https://beej.us/