MC833 Programação de Redes de Computadores

Projeto 2 - Socket UDP

Carlos Robson ra135257

Introdução

Neste trabalho foi implementado um servidor e um cliente que se comunicam através de sockets UDP (User Datagram Protocol). Este é um protocolo utilizado para a transmissão não segura de dados entre dois computadores.

Diferente do TCP, o UDP não é um protocolo confiável, pois não implementa retransmissão de pacotes perdidos, controle de fluxo e controle de congestão e não necessita estabelecer uma conexão antes do envio de dados.

Essa comunicação não é concorrente por natureza, pois o serviço UDP usa um mesmo buffer para todos os clientes. Para tornar esse serviço concorrente é preciso implementar um fluxo para tratar isso usando SELECT.

Sistema

O trabalho simula um sistema acadêmico, que possui uma base de dados com informações de várias disciplinas e permita a realização de algumas operações sobre ela, como consultas e escritas.A operação de escrita de comentário só é permitida pelo professor. Para executar o sistema, deve ser compilado com o gcc, aconselhando nomear os executáveis como server e client. As operações estão listadas abaixo com seus devidos comandos e casos de uso:

1 - Listar todos os códigos de disciplinas com seus respectivos títulos;
    - Executa ./client hostname
    - Escolher tipo de usuário(aluno ou professor)
    - Digitar ID da função: 1
    - Recebe lista dos títulos e os tempos requisição

2 - Listar todas as informações de todas as disciplinas;

- Executa ./client hostname
- Escolher tipo de usuário(aluno ou professor)
- Digitar ID da função: 2
- Recebe lista das informações e os tempos requisição

3 - Dado o código de uma disciplina, retornar a ementa;

- Executa ./client hostname
- Escolher tipo de usuário (aluno ou professor)
- Digitar ID da função: 3
- Digitar Codigo da Disciplina: MC102 (por exemplo)
- Recebe ementa e os tempos requisição

4 - Dado o código de uma disciplina, retornar todas as informações desta disciplina;

- Executa ./client hostname
- Escolher tipo de usuário (aluno ou professor)
- Digitar ID da função: 5
- Digitar Codigo da Disciplina: MC102 (por exemplo)
- Recebe informações e os tempos requisição

5 - Dado o código de uma disciplina, retornar o texto de comentário sobre a próxima aula

- Executa ./client hostname
- Escolher tipo de usuário (aluno ou professor)
- Digitar ID da função: 5
- Digitar Codigo da Disciplina: MC102 (por exemplo)
- Recebe o comentário e os tempos requisição

6 - Ecrever um texto de comentário sobre a próxima aula de uma disciplina

- Executa ./client hostname
- Escolher tipo de usuário (apenas usuário professor);
- Digitar ID da função: 6
- Digitar Codigo da Disciplina: MC102 (por exemplo)
- Digita comentário
- Recebe os tempos requisição

Armazenamento e estruturas de dados do servidor

Os dados do servidor são armazenados diretamente em um arquivo dataAccess.c, com lista de estruturas definidas em base.h. Isso permite controlar o acesso aos dados mais fácilmente.
Os campos guardados de cada disciplina são: id, title, program, schedule, commentary.

Detalhes da Implementação

Para tratar problemas com tamanho de buffer e permissão foi usado a estrutura abaixo:

```
struct headerMsg{
        int userType;                   //0: Student; 1: Professor
         int functionName;              // Request Function Id
         int sizePayload;               //Total Bytes of Payload
         char disciplineId[10];
        char payload[777];
};
```

Para verificar se os dados cabem no buffer tanto na leitura como na escrita, é feito uma checagem no size payload para ver a necessidade de repetir a função. Com o usertype é feito o controle de permissão para o comando 5.

Durante a implementação foi focada na conexão entre os servidor e cliente usando o padrão a biblioteca <sys/socket.h>, procurando tratar erros de conexão, leitura e escrita. Ao invés de um banco de dados relacional, foi criada uma base de dados simples.

Nessa implementação foi usado servidor e cliente UDP concorrente rodando sobre IP versão 4 (IPv4). Utilizando a biblioteca para importar as funções socket(), bind(), sendto(), recvfrom().

Estes métodos foram um pouco diferentes do TCP pois como não tinha uma conexão estabelecida, para usar o sendto() e recvfrom() em conjunto, após usar um precisou armazenar as informações do host que estava se comunicando para fazer utilizar o outro.

Para simplificar o script que rodava os vários pedidos,os comando foram usados como inteiros. Ao rodar um cliente, o usuário deve inserir a comando de 1 a 6, dependendo do comando é solicitado os parâmetros da requisição que serão enviados ao servidor.

Para cada requisição que o usuário faz é repetida 30 vezes, para pegar os dados do tempo que a requisição demora. Tais tempos são subtraídos do tempo de execução do servidor, a fim de avaliar unicamente os tempo de transporte das requisições. Os testes foram feitos em ambiente LAN, sendo o servidor conectado via Wifi e o client de forma cabeada.

Gráficos e análise de resultados

Foram realizados testes para cada caso de uso. Os tempos médios de comunicação de cada caso, bem como desvio padrão e intervalo de confiança constam na tabela abaixo:

Tabela 1 : Resultado dos testes de caso

	Caso 1	Caso 2	Caso 3	Caso 4	Caso 5	Caso 6
Média	3042	2693.5	2738.5	2692	2737.5	2690.5
Desvio	1050.2	12856.3	8983.3	20669.8	493.1	8549.5
Confiança	3417.8	7294.1	5953.1	10088.6	2914.9	5749.9

Como vemos, em geral os valores da média são próximos, mas os desvios padrões podem ser bem grandes, isso ocorre principalmente devido aos valores de tempo da primeira requisição que em geral é bem maior. Esses valores maiores são ocasionados devido a erros na rede, principalmente na wifi onde o servidor estava funcionando.

No gráfico a seguir que representa o caso 4, onde há maior desvio, podemos ver como ocorrem os tempos dos testes:

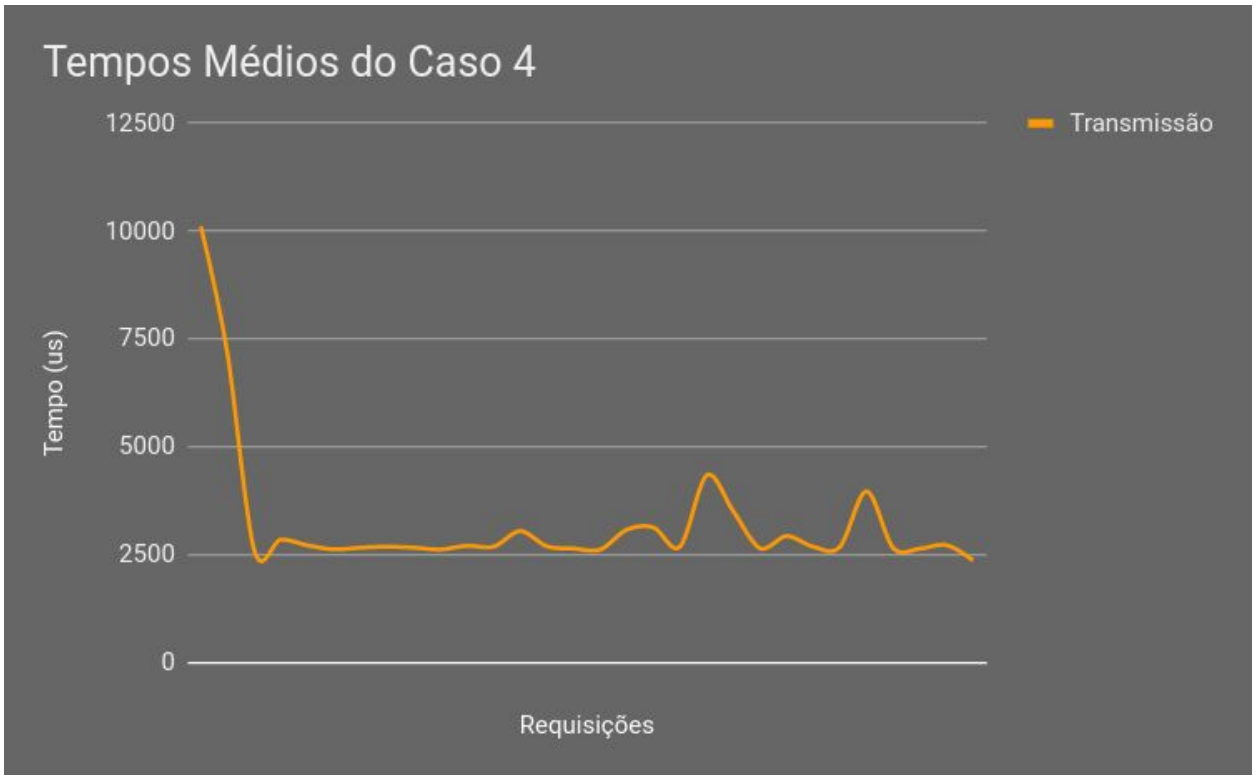
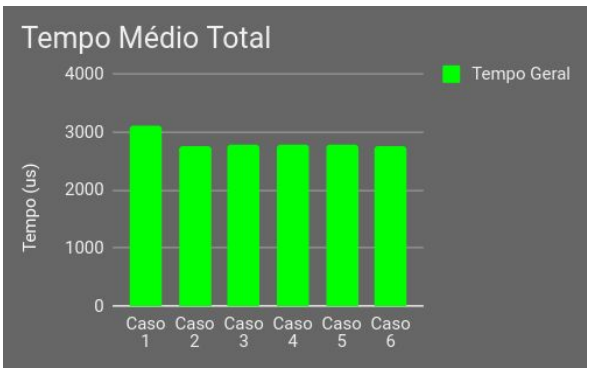


Gráfico 1: Tempos de transmissão para o caso 4

Como podemos ver tem um tempo inicial para o primeiro dentre os 30 testes feitos, isso se deve ao envio dos dados do diagrama que não tem uma conexão inicial,então deve descobrir o caminho até o destinatário. Os demais experimento segue um padrão normal.

Os tempos médio de comunicação de cada caso podem ser comparados no gráficos a seguir:



Gráficos 2 e 3: Comparação entre todo processamento e só a transmissão;

Por esses gráficos percebemos como o tempo de processamento das requisições é mínimo e que o mais importante é mensurar o tempo de transmissão, que se mostrou bem semelhante em todos os casos, sendo o primeiro os mais demorado. Esse comportamento é bastante curioso, já que o primeiro caso deveria passar menos dados que o segundo. Acredita-se que isso se deve ao estabelecimento da rede de comunicação já que os cliente e servidores estavam sem subredes diferentes.

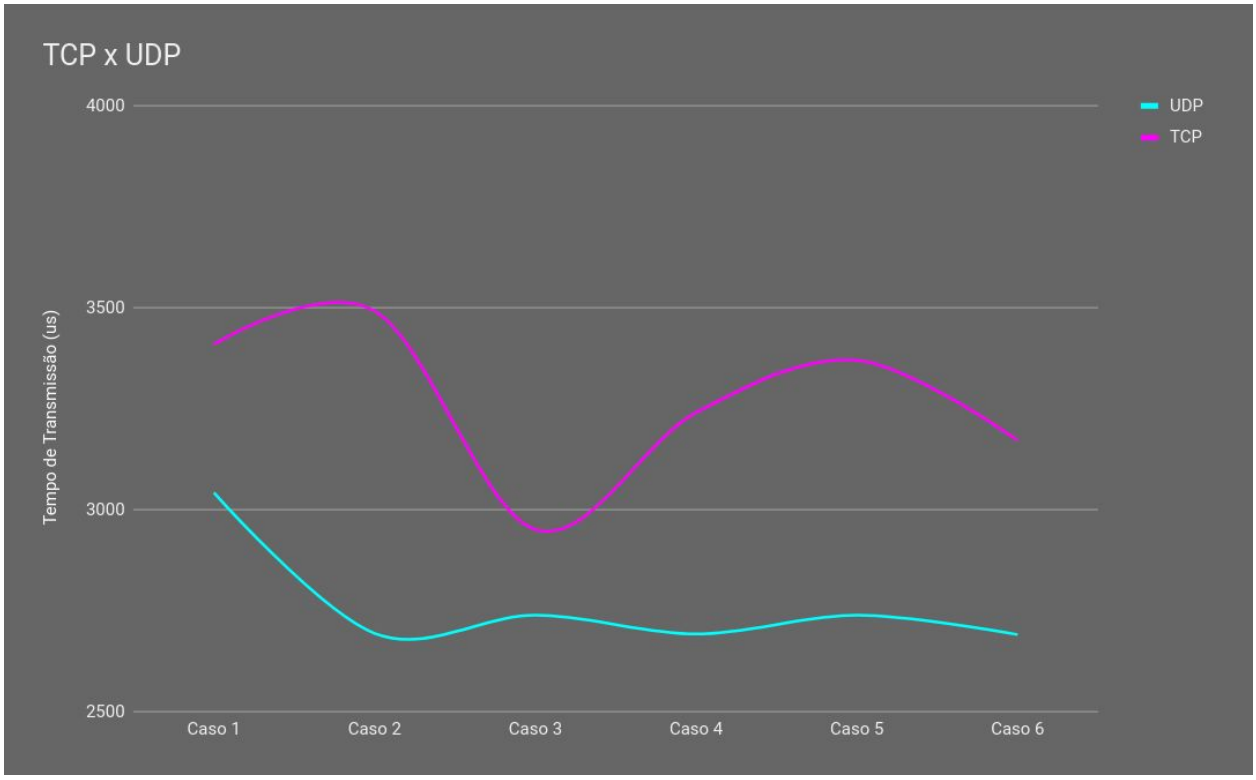


Gráfico 4: Comparação entre os tempos de transmissão do TCP x UDP

Conclusão

Analisando os dados obtidos,podemos perceber que o UDP, em geral é mais rápido, pois como não implementa entrega segura, evita muitos processamentos durante o

roteamento pela rede. Mas devido a isso, aumenta a complexidade de código para tratar alguns problemas gerados, como verificar o endereço de remetente no cliente, a implementação de concorrência, sincronização de entrega de pacotes.
O TCP parece ter tempos mais variáveis no decorrer dos experimentos, já o UDP teve esse começo com tempo maior que os demais e vai se equalizando nas demais execuções.
Esse tipo de comunicação parece ser ideal para programas que precisam de rapidez e que aceitam algumas perdas de pacotes , como é o caso de transmissão de stream de vídeo.

Referencias

https://linux.die.net/man

Beej's Guide to Network Programming

STEVENS R. W., FENNER B., RUDOFF A. M.: UNIX Network Programming The Sockets Networking APIs, Vol. 1, Third Edition, Person Education, 2004