UNIVERSIDADE FEDERAL DE MINAS GERAIS

Estudante: Fernanda Almeida Duarte

Matrícula: 2014144189

Disciplina: Redes de Computadores

Trabalho Prático 2:

sockets UDP, stop-and-wait

1. Introdução

Este trabalho prático consiste na implementação de um software de comunicação cliente-servidor sobre o protocolo UDP de tal forma que o cliente solicita ao servidor um arquivo e esse retorna o conteúdo do arquivo, que o cliente cria uma cópia localmente. Para tal são utilizados sockets para permitir a comunicação cliente-servidor e, para criar um protocolo confiável sobre o UDP, a técnica stop-and-wait. Para realizar medições de desempenho é medido o tempo desde a solicitação do envio do arquivo até o término de seu recebimento.

2. Implementação

Para a implementação deste trabalho prático foi utilizada a biblioteca *tp_socket* para auxiliar no uso dos sockets. Funcionalidades como criação e configuração de socket, envio e recebimento de dados foram construídas com auxílio dessa biblioteca.

O cabeçalho do protocolo foi construído como bater do *buffer*, de forma que cada mensagem tenha um id seguido de uma barra (/) e então os dados a serem enviados, como demonstrado abaixo:

1/Lorem ipsum dolor sit amet, consectetur adipiscing elit

Para permitir tal implementação até mesmo com buffers de tamanho menor que o necessário para o id e o caractere / cada buffer enviado na realidade possui *tam_buffer* +11 bytes, sendo que *tam_buffer* é o tamanho do buffer informado pela linha de comando. Esses 11 bytes adicionados correspondem aos 10 necessários para representar inteiros de 1 até INT_MAX em uma string e o byte restante para o caractere separador /.

Para melhor compreensão da lógica utilizada segue passo a passo de uma linha de execução resumida da solução implementada para o *clienteFTP* e *servidorFTP* complementar:

clienteFTP:

1. envia nome do arquivo para servidor
2. recebe primeiro buffer
3. separa o buffer recebido para obter o id da mensagem e os dados
4. verifica o id recebido
    a. se id é igual ao esperado - no caso id=1:
        i. grava dados recebidos em um arquivo
        ii. envia ACK para servidor
        iii. aguarda segundo buffer
        iv. recebe segundo buffer
        v. separa o buffer recebido para obter o id da mensagem e os dados
        vi. se id é igual a 2 grava dados no arquivo
        vii. envia ACK para servidor
    b. se id é diferente ao esperado: envia NACK para o servidor
        i. aguarda recebimento do buffer de id 1

E então prossegue a execução até receber uma mensagem que possui apenas um caractere “0”, o qual indica o fim da transmissão e fim da conexão.

servidorFTP

1. recebe nome do arquivo
2. lê arquivo e cria o primeiro buffer (concatenando a substring do id “1/” aos dados)
3. envia primeiro buffer
4. aguarda recebimento de ACK ou NACK
    a. caso receba ACK:
        i. lê arquivo e cria o segundo buffer (concatenando a substring do id “2/” aos dados)

ii. envia segundo buffer buffer
iii. aguarda recebimento de ACK ou NACK

b. caso receba NACK
i. envia novamente a última mensagem enviada
ii. aguarda recebimento de ACK ou NACK

Além do recebimento de uma mensagem fora da ordem há outra forma do clienteFTP enviar um NACK para o servidorFTP: caso ocorra uma temporização. Para implementação das temporizações foi utilizada a função *setsockopt*, que permite a configuração de um tempo máximo pelo qual o socket aguarda uma mensagem. Dessa forma quando ocorre uma temporização a função *tp_recvfrom* retorna -1 e um NACK solicitando reenvio da próxima mensagem esperada é enviado. O ACK consiste numa mensagem que o cliente envia para o servidor contendo o caractere "0" e um NACK, "1",

Abaixo segue código utilizado para configurar a temporização em 1 segundo:

```
struct timeval tv;
tv.tv_sec = 1;
tv.tv_usec = 0;
if (setsockopt(clientefd, SOL_SOCKET, SO_RCVTIMEO,&tv,sizeof(tv)) < 0)
 {
    perror("Error");
}
```

Dessa forma foi implementada a técnica *stop-and-wait*, através de cabeçalhos com o id e envio de ACKs e NACKs.

3. Metodologia

Os experimentos foram realizados em uma única máquina, notebook com as seguintes configurações:

- Sistema operacional: Ubuntu 16.04 LTS 64-bit;
- Memória RAM: 4GB DDR3 L;
- Processador: Intel Core i3-5015U (2.1GHz, 3MB L3 Cache).

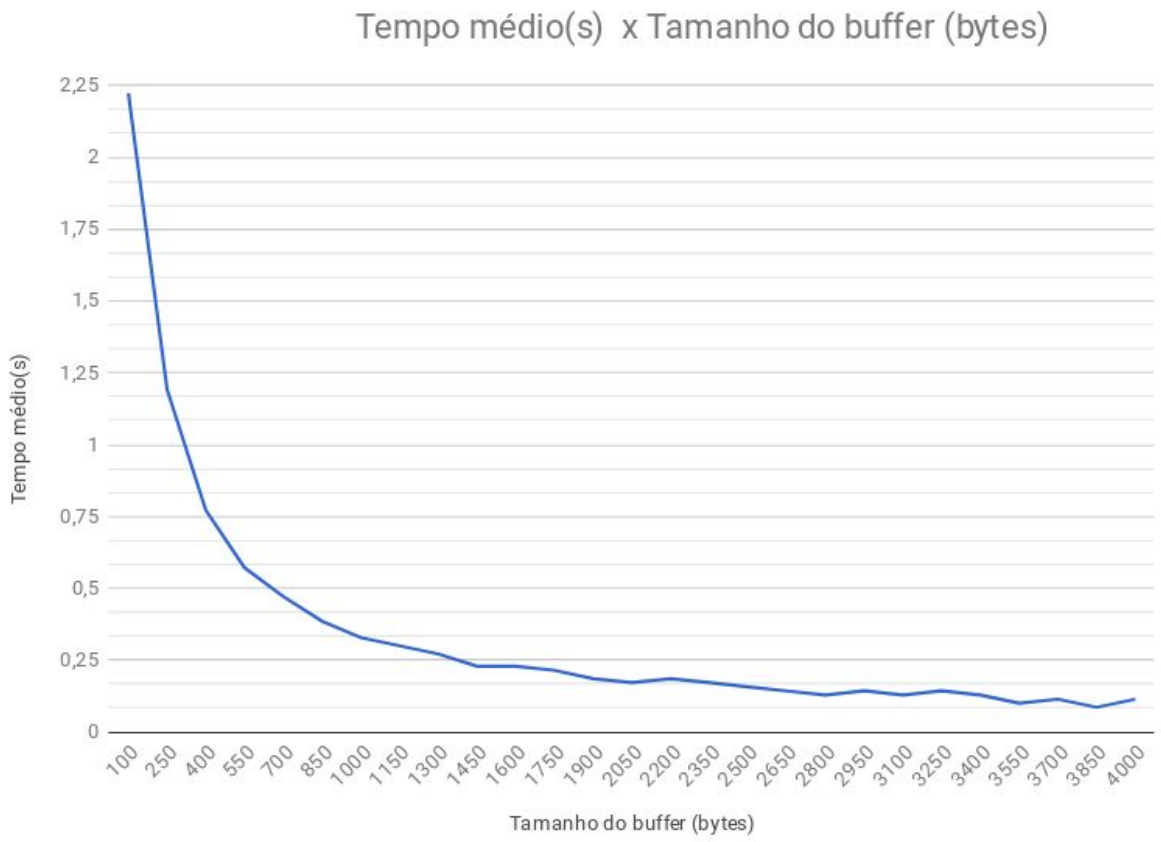
Para medição de tempo foi utilizada a função *gettimeofday*. A quantidade de bytes do arquivo que o cliente recebeu corretamente do servidor foi calculada a partir da soma do tamanho da área de dados dos buffers recebidos. Cada bateria de teste consistiu no envio de um arquivo de 3MB 100 vezes para cada tamanho do buffer - que foi de 100 a 4000 bytes, variando de 200 em 200 bytes. Foi construído um loop para cada bateria, que rodou as 100 vezes necessárias.

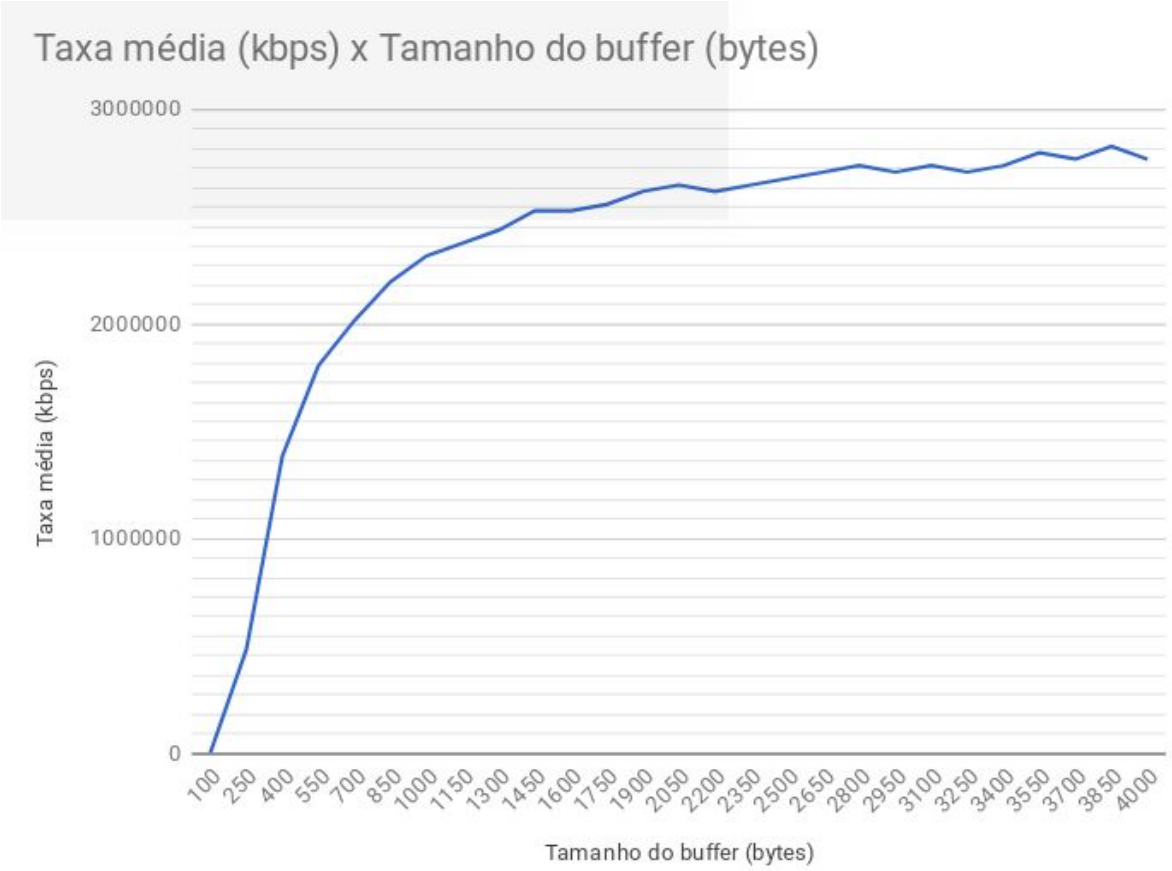
Para análise dos resultados foi calculado o tempo médio de cada bateria de testes e montada uma tabela e gráficos que se encontram na seção *Resultados*. Como a taxa é dada em kbps a mesma foi medida dividindo o número de bytes recebidos com sucesso dividido pelo tempo (segundos apenas, desconsiderando os milisegundos) gasto.

4. Resultados

Tamanho do buffer (bytes)	Tempo médio(s)	Taxa média (kbps)
100	2,223403	1580,29
250	1,1921870	484.140,30
400	0,7719280	1.386.259,48
550	0,5717990	1.807.248,43
700	0,4717340	2.017.742,90
850	0,3859640	2.198.166,74
1000	0,3287840	2.318.449,30
1150	0,3001940	2.378.590,58
1300	0,2716040	2.438.731,85
1450	0,2287190	2.528.943,77
1600	0,2287190	2.528.943,77
1750	0,2144250	2.559.014,41
1900	0,1858350	2.619.155,69
2050	0,1715400	2.649.226,33
2200	0,1858350	2.619.155,69
2350	0,1715400	2.649.226,33
2500	0,1572450	2.679.296,97
2650	0,1429500	2.709.367,61
2800	0,1286550	2.739.438,25
2950	0,1429500	2.709.367,61

3100	0,1286550	2.739.438,25
3250	0,1429500	2.709.367,61
3400	0,1286550	2.739.438,25
3550	0,1000650	2.799.579,53
3700	0,1143600	2.769.508,89
3850	0,0857700	2.829.650,16
4000	0,1143600	2.769.508,89





5. Análise

A partir da tabela e gráficos acima nota-se que quanto maior o buffer menor o tempo de envio do arquivo. Esse resultado encontra-se com o esperado pois à medida que o buffer cresce mais dados podem de ser enviados de uma só vez e menos mensagens são necessárias para enviar o arquivo inteiro.

O resultado interessante são os picos encontrados ao longo do gráfico. Por exemplo, com o buffer de 3550 bytes o tempo médio foi de 0,1000650 segundo e com 3700 foi de 0,1143600 segundo. O esperado seria seguir a tendência geral do gráfico, com o tempo médio sempre diminuindo com o aumento do buffer.

No gráfico de taxa média x tamanho do buffer é também é possível observar comportamento equivalente. É de se esperar que a taxa aumente com o aumento do buffer (e diminuição do tempo). Porém há alguns vales na curva, demonstrando diminuição da taxa média em alguns momentos enquanto o buffer aumentava.

Considerando que estamos utilizando UDP nesta aplicação possivelmente tais picos identificados devem-se a perdas de pacotes (seja pacotes fora de ordem ou temporizações

ocorridas) e a necessidade de reenvio dos mesmos, fazendo com que o tempo gasto seja maior do que o esperado.

6. Conclusão

A partir deste trabalho prático foi possível fixar mais o funcionamento do UDP e como construir um protocolo em cima de outro a fim de garantir a entrega em ordem das mensagens. O principal desafio consistiu na lógica necessária para garantir a entrega das mensagens da forma desejada. As variações encontradas nos gráficos são interessantes pois fazem pensar sobre como uma ferramenta que usamos no dia a dia - a internet - consegue funcionar de forma rápida e eficiente mesmo quando são utilizados protocolos como o UDP, que em uma aplicação simples como a desenvolvida aqui já demonstra variações no tempo de envio de dados.