Trabalho Prático 2 - Redes de Computadores

Prof. João Guilherme M. Menezes

Alexandre Paiva, Bárbara Ferreira, Jéssica Rodrigues

22 de Outubro de 2018

1 Introdução

Este trabalho tem por objetivo, realizar e analisar a comunicação e transmissão de mensagens no modelo cliente-servidor.
Essa comunicação foi realizada por meio de protocolo UDP (User Datagram Protocol) e implementada fazendo uso da biblioteca de sockets.
O protocolo UDP manda informações a um destinatário, sem se preocupar se elas foram recebidas devidamente — em caso de erros, ocorre o envio do próximo pacote programado pelo sistema, e os anteriores são perdidos. Embora esse método de funcionamento potencialize a ocorrência de erros, ele garante uma comunicação rápida entre dois computadores.
O problema da tarefa, então também consistiu em garantir a confiabilidade na entrega dos pacotes.
Neste trabalho também foi feita a medição do desempenho do envio dos arquivos , conforme a quantidade de dados enviados e tamanho do buffer para transferência.

2 Implementação

Para garantir a entrega confiável dos dados protocolo UDP foi utilizado implementando o stop-and-wait. Um remetente stop-and-wait (servidor enviando os arquivos) envia um quadro por vez e espera que o destinatario (cliente) responda que foi recebido com sucesso. Fizemos umas pequenas alteracões no código tp_socket.c e tp_socket.h. A primeira delas foi garantir que a variável "struct sockaddr_in local_ddr"fosse global para poder ser utilizada no servidor e o cliente. Da forma que foi implementado originalmente os valores do socket ficavam guardados internamente e não estavamos conseguindo acessa-los. A segunda alteração foi realizada para facilitar o teste

na mesma maquina, como não é possivel fazer o bind do socket do cliente quando o mesmo já está sendo utilizado pelo servidor, adicionamos uma flag que determina se o socket a ser criado será de servidor ou cliente

```
int tp_socket(unsigned short port, bool isServer)
```

O "datagrama"na nossa implementação é uma string. Para indicar o "tipo", usamos a primeira posição: 0 indica fim da transferencia, 1 é o "ACK", 2 é arquivo aberto com sucesso e 3 erro na abertura do arquivo. Os códigos 2 e 3 são enviados do servidor para o cliente. Já os códigos 0 e 1 são utilizados por ambos os hospedeiros.

O servidor fica esperando por uma mensagem de um cliente com o nome do arquivo. Ao receber uma string com este nome, ele envia um datagrama para o cliente com um cabeçalho indicando sucesso ou erro na leitura do arquivo. O cliente então envia uma confirmação com cabeçalho 1 e o número de bytes do arquivo que já foram recebidos. O servidor, de acordo com esse número, decide se deve enviar o mesmo datagrama ou continuar com a leitura do arquivo, no caso de perda depacote. Em ambos os casos os datagramas são enviados novamente caso receba timeout. Assim que o servidor termina a leitura, envia um cabeçalho 0 e espera por uma última confirmação do cliente.

3 Metodologia

Para realização do trabalho, usamos máquinas, todas com sistema operacional Ubuntu/Linux.
Medimos o desempenho do nosso programa utilizando arquivos de até XXXMB variando o tamanho do buffer. Para criar estes arquivos utilizamos o seguinte comando para criar, por exemplo, um arquivo de 10MB:

```
base64 /dev/urandom | head -c 10000000 > file.txt
```

Para realizar a contagem do tempo utilizamos a função gettimeofday e foram feitas diferentes execuções do programa para diferentes tamanhos de arquivo e buffer. Os resultados podem ser encontrados na seção abaixo.

4 Resultados

Foram realizados diversos testes afim de analisar o desempenho da implementação, segue abaixo os resultados.

Buffer de 100 bytes					
MB	**3.000.000**	**50.000.000**	**100.000.000**	**500.000.000**	**1.000.000.000**
Tempo (s)	0	5	11	72	145
Vazão (kbps)	0	10.000	9.091	6.944	6.897
Quantidade de mensagens	30.000	500.000	1.000.000	5.000.000	10.000.000
MB	**3.000.000**	**50.000.000**	**100.000.000**	**500.000.000**	**1.000.000.000**
Tempo (s)	0	5	11	58	135
Vazão (kbps)	0	10.000	9.091	8.621	7.407
Quantidade de mensagens	30.000	500.000	1.000.000	5.000.000	10.000.000
MB	**3.000.000**	**50.000.000**	**100.000.000**	**500.000.000**	**1.000.000.000**
Tempo (s)	0	5	11	59	122
Vazão (kbps)	0	10.000	9.091	8.475	8.197
Quantidade de mensagens	30.000	500.000	1.000.000	5.000.000	10.000.000

Figura 1: Alguns dos valores encontrados utilizando buffer de 100 bytes

Buffer de 100 bytes (média)					
MB	**3.000.000**	**50.000.000**	**100.000.000**	**500.000.000**	**1.000.000.000**
Tempo médio (s)	0	5,33	11	63,00	134,00
Vazão média (kbps)	0	9.381	9.091	7.937	7.463
Quantidade de mensagens	30.000	500.000	1.000.000	5.000.000	10.000.000
Desvio padrão	0,00	0,00	0,00	7,81	11,53

Figura 2: Média dos valores encontrados utilizando buffer de 100 bytes

Buffer de 1000 bytes					
MB	**3.000.000**	**50.000.000**	**100.000.000**	**500.000.000**	**1.000.000.000**
Tempo (s)	0	1	2	10	18
Vazão (kbps)	0	50.000	50.000	50.000	55.555,56
Quantidade de mensagens	3.000	50.000	100.000	500.000	1.000.000
MB	**3.000.000**	**50.000.000**	**100.000.000**	**500.000.000**	**1.000.000.000**
Tempo (s)	0	1	2	10	16
Vazão (kbps)	0	50.000,00	50.000,00	50.000,00	62.500,00
Quantidade de mensagens	3.000	50.000	100.000	500.000	1.000.000
MB	**3.000.000**	**50.000.000**	**100.000.000**	**500.000.000**	**1.000.000.000**
Tempo (s)	0	1	2	12	17
Vazão (kbps)	0	50.000,00	50.000,00	41.666,67	58.823,53
Quantidade de mensagens	3.000	50.000	100.000	500.000	1.000.000

Figura 3: Alguns dos valores encontrados utilizando buffer de 1000 bytes

Buffer de 1000 bytes (média)					
MB	**3.000.000**	**50.000.000**	**100.000.000**	**500.000.000**	**1.000.000.000**
Tempo médio (s)	0	1	2	10,66	17,00
Vazão média (kbps)	0	50.000	50.000	46.904	58.824
Quantidade de mensagens	3.000	50.000	100.000	500.000	1.000.000
Desvio padrão	0,00	0,00	0,00	1,15	1,00

Figura 4: Média dos valores encontrados utilizando buffer de 1000 bytes

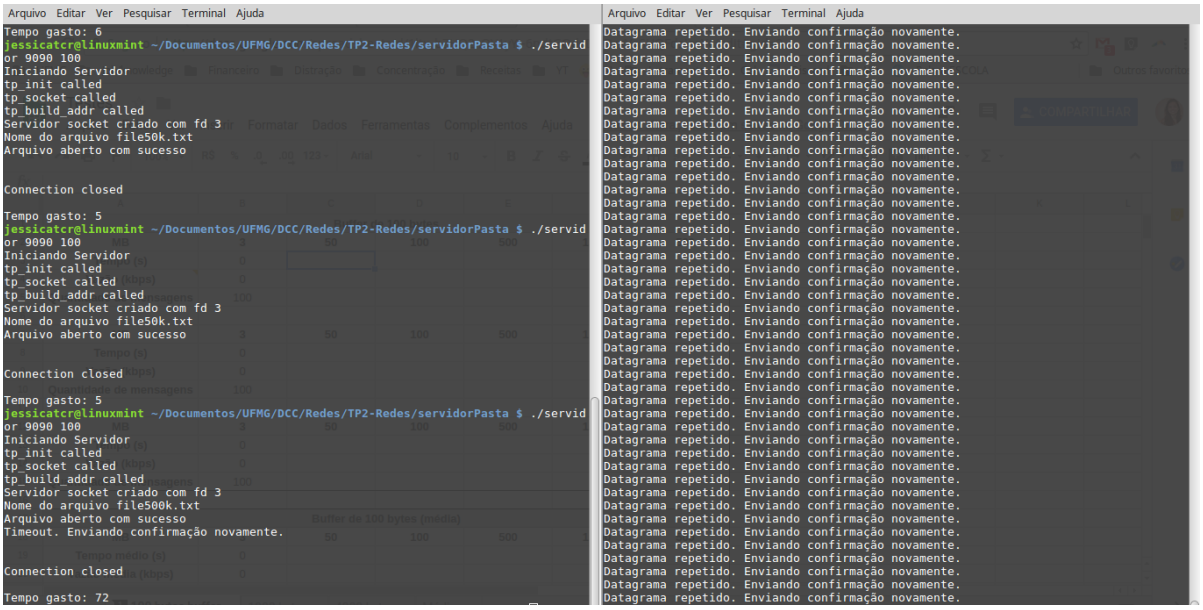

Figura 5: Evento que ocorreu Time Out

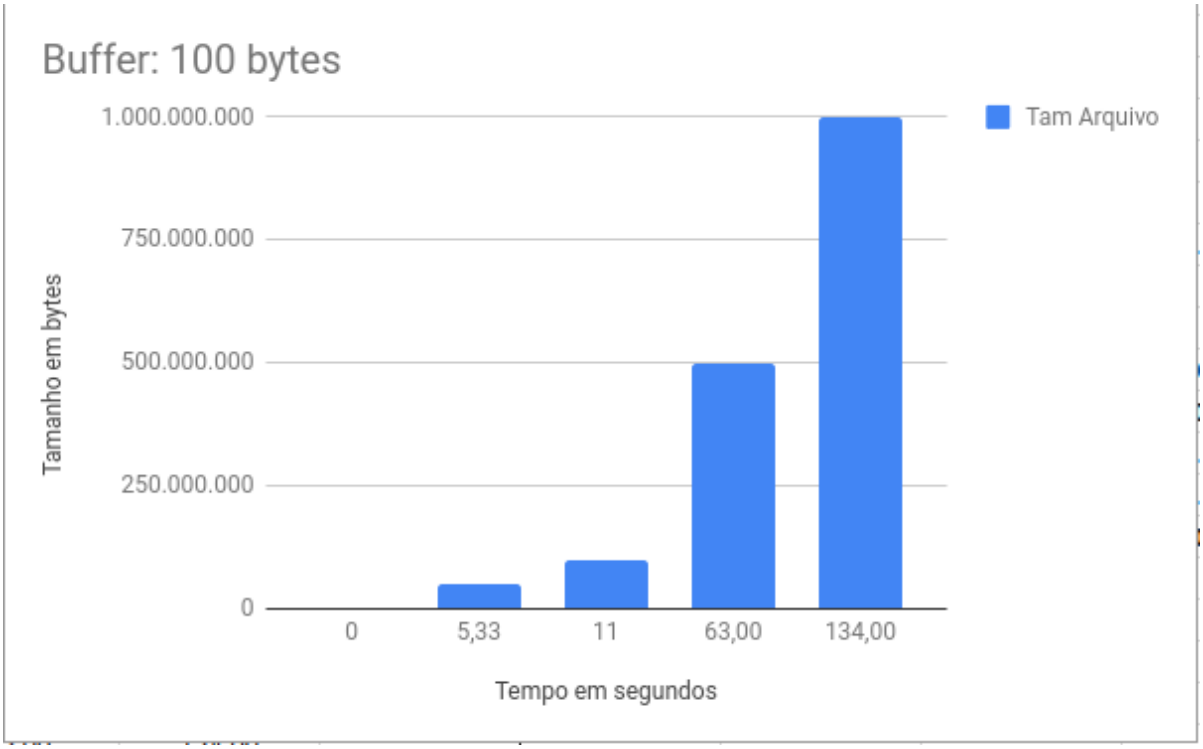

Figura 6: Buffer: 100bytes

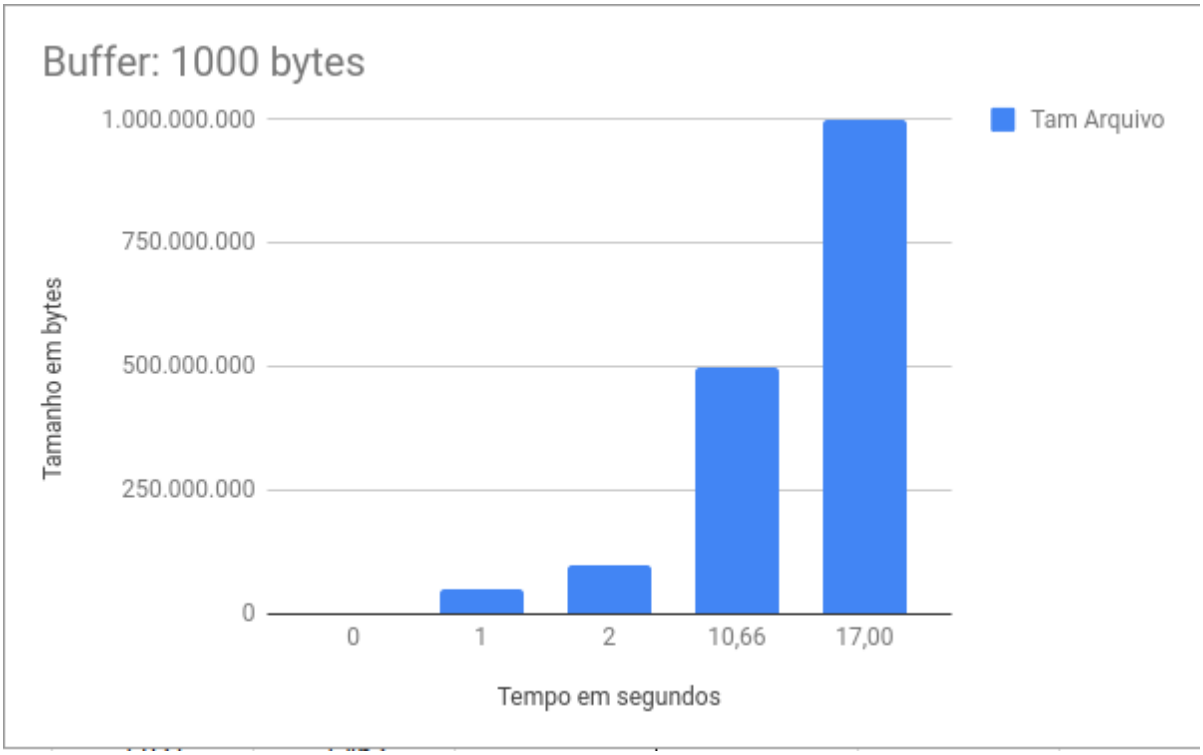

Figura 7: Buffer: 1000bytes

5 Análise

Pelos gráficos e tabelas pode-se perceber que o problema foi resolvido de forma satisfatória. O tempo de transferência é sempre proporcional ao tamanho do arquivo, com alguma variação, mas que indica que ao menos o programa não se comporta de forma exponencial.
Mas algo interessante de perceber que é talvez a execução do programa em si seja mais responsável pelo tempo que a taxa de transferência em si, mesmo considerando que tanto servidor quanto cliente tenham sido testados apenas na mesma máquina. Podemos perceber que o tempo de execução é menor para os mesmos arquivos quanto maior é o tamanho do buffer.
Outro ponto interessante é que observamos Time Out apenas em arquivos maiores, como por exemplo os arquivos de 500MB e 1GB que foram utilizados para testar o nosso trabalho.

6 Conclusão

Realizar esse trabalho nos fez entender melhor como a estrutura da comunicação cliente-servidor funciona sobre o protocolo UDP, aprendida em sala de aula.
Fizemos muitas pesquisas para entender do que se tratava o problema, por exemplo, quanto a temporização e isso foi bom, pois entendemos melhor sobre os conceitos e algoritmos.
Tivemos algumas dificuldade, no que tange a implementação. A principal foi em relação a adaptação inicial, fazendo uso do código de auxílio, tivemos que fazer algumas alterações, como foi explicado na seção de implementação.
Foi um trabalho muito proveitoso, e interessante. Fizemos uso de aprendizados do trabalho prático 1 e com a realização deste tivemos mais conhecimento agregado, maior do que parecia de início.

Referências

- Programando um protocolo utilizando Sockets. Disponível em: `https://blog.pantuza.com/artigos/programando-um-protocolo-utilizando-sockets`
- Programming udp sockets in C on Linux. Disponível em: `https://www.binarytides.com/programming-udp-sockets-c-linux/`
- UDP Server-Client implementation in C. Disponível em: `https://www.geeksforgeeks.org/udp-server-client-implementation-c/`

- UDP Programming in C. Disponível em: `https://www.youtube.com/watch?v=Emuw71lozdA`
- Stop-and-Wait Protocol. Disponível em: `https://seattle.poly.edu/wiki/EducationalAssignments/StopAndWait`