Trabalho Prático 2

Gilson Urbano Ferreira Dias - 2015430959
Gustabo Braga - 2012076429
guf@ufmg.br, gbrabosa@ufmg.br

23 de outubro de 2018

1 Introdução

O objetivo deste trabalho é implementar um par de programas que operem no modelo cliente-servidor e exercitem tanto a transmissão unidirecional quanto a comunicação do tipo requisição-resposta sobre o protocolo UDP. A implementação será realizada utilizando a biblioteca de sockets do Unix (Linux).

O cliente deve se conectar ao servidor, enviar um string com o nome do arquivo desejado, receber o arquivo por meio de um buffer de cada vez e salvar os dados no disco à medida que eles chegam. O servidor por sua vez deve operar de forma complementar.

Neste trabalho será utilizado o protocolo UDP. Como nesse protocolo não há apresentação entre as entidades remetente e destinatária da camada de transporte antes de enviar um segmento, e é realizada apenas a melhor tentativa para entregar o segmento ao hospedeiro receptor, pacotes de dados podem ser perdidos. Esse problema pode ser corrigido se a confiabilidade for embutida na própria aplicação, por exemplo, adicionando mecanismos de reconhecimento e de retransmissão. O protocolo pare e espere (stop-and-wait) foi implementado garantindo a transferência confiável de dados, uma característica necessária em aplicações de transferência de arquivos [6].

2 Implementação

Os pacotes transmitidos são compostos por duas partes; cabeçalho e corpo, para o cabeçalho estão disponíveis 15 bytes que informam um número de identificação (ID), o corpo tem o tamanho do buffer escolhido por parâmetro composto. A variável "ID"é incrementada se o recebimento de um pacote de dados transmitido for confirmado pelo cliente. Quando o cliente recebe um pacote, ele envia uma respostas de reconhecimento positivo (ACK), confirmando o incremento do valor do ID para, $ID = ID + 1$ e o servidor tenta realizar a próxima transmissão se houver disponibilidade de um pacote de dados. Já quando um pacote é perdido, o cliente envia um reconhecimento negativo (NACK), e o ID não é incrementado.

O protocolo stop-and-wait baseia-se nas mensagens de reconhecimento e números de identificação para garantir a transferência confiável de dados. Toda vez que um pacote é enviado, o servidor aguarda pela mensagens de reconhecimento do cliente para então realizar uma nova tentativa de transmissão. Quando a mensagem de reconhecimento NACK é recebida, o servidor terá que realizar uma nova tentativa de envio do pacote de

dados perdido durante a transmissão anterior. Quando a mensagem de reconhecimento for um ACK, o servidor entende que o pacote enviado foi recebido corretamente pelo cliente, e segue enviado os pacotes na ordem, garantindo que o cliente receba uma cópia do arquivo.

3 Metodologia

Diversos testes foram realizados de forma a verificar o correto funcionamento do par de programas cliente e servidor que foram implementados na linguagem C, e para coletar os dados de performance que serão analisados posteriormente. Os testes foram realizados em no laptop Thinkpad T440 com processador Intel Core i5 Vpro 4300U rodando a 2.90 GHz, 8 Gb de memória RAM, e sistema operacional Elementary OS que é baseado no Ubuntu 16.04 LTS.

Para a execução dos testes foi utilizado um arquivo de texto com 10.233.868 bytes (10.23 MB), o objetivo principal é verificar como o desempenho do par de programas varia quando diferentes tamanhos de buffer são utilizados. Os programas foram colocados em pastas diferentes para garantir que a transferência do arquivo realmente ocorreu. Em alto nível o funcionamento dos programas segue os seguintes passos, o servidor deve ser iniciado e aguardar que o o cliente se conecte, enviando uma string com o nome do arquivo desejado, para que possa receber o arquivo um buffer de cada vez e salvar os dados no disco à medida que eles chegam. Quando não houver mais bytes para ler, o cliente fecha a conexão e o arquivo. O servidor por sua vez deve operar de forma complementar.

Os parâmetros do par de programas e a ordem que devem ser inseridos para o perfeito funcionamento devem seguir os exemplos a seguir. O programa servidorFTP recebe como parâmetros o número da Porta e o tamanho do buffer em bytes, por exemplo, Porta 8080 e buffer de 256 bytes.

```
1 urbanogilson@urbanogilson:~/servidorFTP$ ./servidorFTP 8080 256
```

No programa clienteFTP são necessários os parâmetros host, número da Porta do servidor, nome do arquivo que se deseja recuperar do servidor e o tamanho do buffer em bytes, por exemplo, host 127.0.0.1 IP loopback utilizado na realização dos testes na máquina atual, Porta 8080 do servidor, arquivo a ser recuperado file.txt e o buffer com tamanho de 256 bytes.

```
1 urbanogilson@urbanogilson:~/clienteFTP$ ./clienteFTP 127.0.0.1 8080 file.txt 256
```

Medições de tempo foram realizadas variando o tamanho da mensagens em 2^i bytes, com $6 \leq i \leq 12$. Para cada tamanho de mensagem (buffer $= 2^i$ bytes), foram realizados 5 execuções diferentes do programa utilizando a função gettimeofday da biblioteca sys/-time.h para obter o tempo de execução, os resultados estão dispostos na Tabela 1. Na Tabela 2 foram realizadas as médias do tempos obtidos nos testes, medidas de dispersão dos tempos e vazão da comunicação.

4 Resultados

Os resultados dos tempos (em segundos) de execução foram obtidos utilizando a estratégia de realizar diferentes execuções do programa para cada tamanho de buffer. Os valores foram tabulados e podem ser vistos na Tabela 1.

Tabela 1: Tempo de transmissão para um arquivo de 10.23 MB.

Tam. Buffer	**Tempo 1**	**Tempo 2**	**Tempo 3**	**Tempo 4**	**Tempo 5**
64	5,257610	5,238230	5,472875	5,150758	5,205772
128	2,742809	2,623602	2,664381	2,563583	2,579929
256	1,276804	1,457267	1,276666	1,305771	1,369879
512	0,780680	0,860295	0,731135	0,710031	0,680235
1024	0,426994	0,366490	0,367926	0,366203	0,365172
2048	0,233712	0,236923	0,303356	0,232403	0,217534
4096	0,135008	0,125547	0,212651	0,123357	0,135086

Utilizando os dados acima foram realizados algumas medidas, como a média dos tempos para cada tamanho de buffer em bytes, medidas de dispersão como: desvio padrão e variância dos tempos amostrados e uma medida da vazão (KB/s) da comunicação que é o número de kilobytes enviados dividido pelo tempo medido no cliente. É possível notar um decréscimo no tempo necessário para a transferência do aquivo de acordo o tamanho do buffer aumenta, um gráfico Figura 1 foi criado para verificar essa conclusão visualmente.

Tabela 2: Análise do tempos de transmissão para um arquivo de 10.23 MB.

Tam. Buffer	**Tempo $\bar{t}$**	**Desvio Padrão σ**	**Variância σ^2**	**Vazão $\overline{v}$**
64	5,265049	0,123027	0,015136	1943,74
128	2,634861	0,072023	0,005187	3884,03
256	1,337277	0,077110	0,005946	7652,76
512	0,752475	0,070527	0,004974	13600,27
1024	0,378557	0,027095	0,000734	27033,89
2048	0,244786	0,033582	0,001128	41807,48
4096	0,146330	0,037459	0,001403	69937,01

Visualmente também fica fácil notar um resultado que olhando apenas a Tabela 2 é difícil de perceber, com o tamanho do buffer crescendo exponencialmente o tempo decresce exponencialmente.

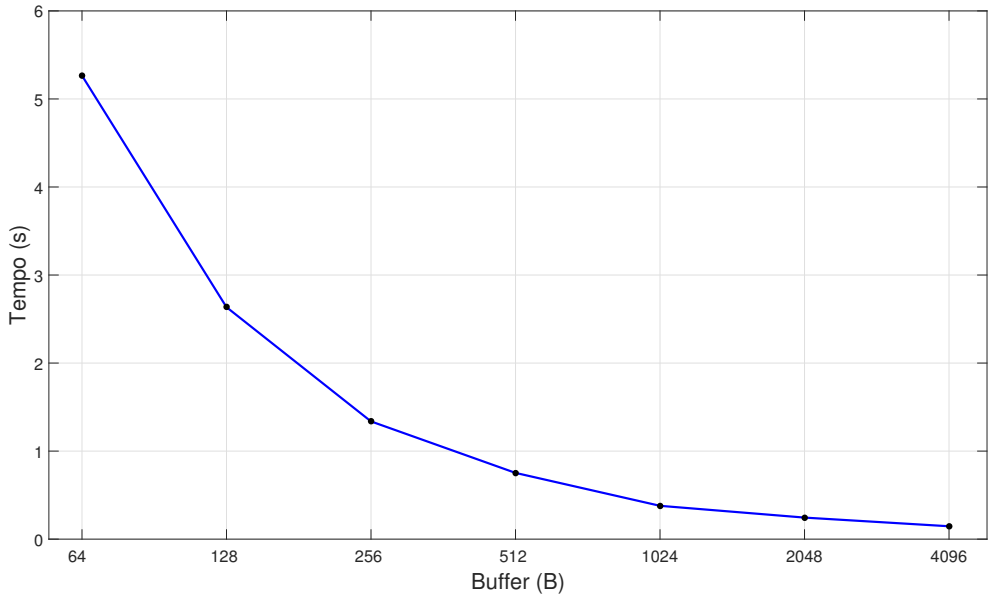

Figura 1: Relação Buffer x Tempo na transmissão de um arquivo com 10.23 MB.

5 Análise

A Tabela 2 mostra que o desvio padrão ficou baixo mesmo utilizando apenas 5 amostras de tempo. Com exceção do teste com o buffer igual a 64 bytes, que teve $\sigma = 0,123027$, todos os tamanhos de buffer tiveram o desvio padrão menor do que $0,1$.

O resultado da Figura 1 já era esperado, pois um tamanho maior do buffer diminuiu o número de mensagens trocadas entre trocadas entre o cliente e o servidor, implicando em um decréscimo no tempo de transmissão.

Apesar de existir um risco de pacotes serem perdidos pela utilização do protocolo UDP, a implementação do protocolo stop-and-wait tornou a transmissão confiável. Em todos os testes os arquivos chegaram íntegros no cliente.

Porém, o preço a se pagar pela utilização do protocolo stop-and-wait foi o tempo. Ao compararmos com os resultados obtidos no primeiro trabalho, que foi implementado utilizando o protocolo TCP, ao transferir o mesmo arquivo, a diferença de tempo foi muito grande. Na tabela 3 usamos uma das tomadas de tempo de ambos trabalhos para realizarmos a comparação. É fácil notar uma enorme diferença.

Tabela 3: Comparação entre o protocolo UDP com Stop-and-wait e protocolo TCP.

UDP		**TCP**	
Tam. Buffer	**Tempo 1**	**Tam. Buffer**	**Tempo 1**
64	5,257610	**64**	0,192399
128	2,742809	**128**	0,111296
256	1,276804	**256**	0,059986
512	0,780680	**512**	0,043763
1024	0,426994	**1024**	0,033564
2048	0,233712	**2048**	0,027506
4096	0,135008	**4096**	0,020911

6 Conclusão

O trabalho complementou o que estudamos em sala de aula. Ao contrário do que ocorre com o protocolo TCP, o protocolo UDP, normalmente preza pela velocidade e não pela integridade dos dados. Neste trabalho foi possível comparar a todo momento os dois tipos de protocolo, pois implementamos trocas de mensagens de reconhecimento ACK e NACK, algo que ocorre de forma muito similar no protocolo TCP. Com isso foi garantida a transferência confiável de dados, uma característica necessária em aplicações de transferência de arquivos, em uma aplicação utilizando o protocolo UDP.

Por fim o trabalho pode se concluído com os resultados esperados. Por exemplo, a transmissão do arquivo utilizando o protocolo UDP ocorreu sem perdas de informação, isso deve-se ao fato da implementação correta do protocolo stop-and-wait, além disso, e o tempo de transmissão diminui quando o tamanho do buffer aumenta.

Referências

[1] arpa/inet.h. `http://pubs.opengroup.org/onlinepubs/009696899/basedefs/arpa/inet.h.html`. Accessado em: 15-09-2018.

[2] netinet/in.h. `http://pubs.opengroup.org/onlinepubs/000095399/basedefs/netinet/in.h.html`. Accessado em: 15-09-2018.

[3] sys/socket.h. `http://pubs.opengroup.org/onlinepubs/7908799/xns/syssocket.h.html`. Accessado em: 15-09-2018.

[4] sys/time.h. `http://pubs.opengroup.org/onlinepubs/7908799/xsh/systime.h.html`. Accessado em: 15-09-2018.

[5] sys/types.h. `http://pubs.opengroup.org/onlinepubs/009696699/basedefs/sys/types.h.html`. Accessado em: 15-09-2018.

[6] James F Kurose and Keith W Ross. *Redes de computadores e a Internet - uma abordagem top-down*. Pearson Education do Brasil, 6 edition, 2013.