UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Redes de Computadores

TRABALHO PRÁTICO 2

Sockets UDP, janela deslizante

Guilherme Saulo Alves

07 de Junho de 2015

Belo Horizonte – MG

1 Introdução

O objetivo deste trabalho prático consiste em 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. Este protocolo utiliza a forma de pacotes para enviar mensagens e não garante a confiabilidade do canal de comunicação. Portanto, a implementação deve lidar com possíveis erros na transmissão de pacotes entre as entidades. Uma forma de permitir a entrega confiável e ordenada dos pacotes é por meio do protocolo de janela deslizante. A implementação utilizou a biblioteca de sockets do Unix.

2 Implementação

De forma resumida, os programas cliente e servidor funcionam da seguinte forma: o cliente envia o nome do arquivo desejado ao servidor. O servidor recebe o nome do arquivo e começa enviar os pacotes que contém os dados do arquivo ao cliente. O cliente começa a armazenar os dados dos pacotes em disco a medida que eles chegam. Quando não houver mais pacotes a ser transferidos, o cliente fecha a conexão e imprime os dados de execução.

2.1 Cabeçalho do Protocolo

Segue abaixo a definição dos campos do cabeçalho dos pacotes que são enviados dentro do pacote UDP:

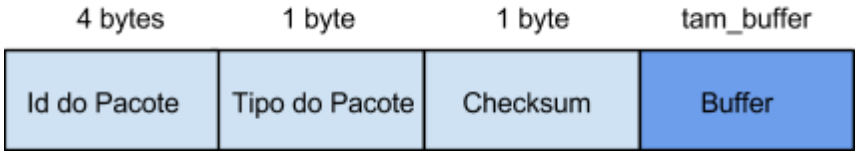

Figura 1 - Pacote do protocolo

Existem quatro tipos de pacotes para o protocolo implementado: pacotes de dados com o cabeçalho “TIPO_DADOS”, pacotes de confirmações com o cabeçalho “TIPO_ACK”, pacotes de termino de transmissão com o cabeçalho “TIPO_FINAL”. Logo, adotou-se um formato único de pacote composto da seguinte forma:

Bytes 1 à 4: Contém a identificação do pacote;
Byte 5: Contém o tipo do pacote;
Byte 6: Contém o *checksum*;
*Byte 7 à tam_buffe*r: contém a mensagem do arquivo.

2.2 Janela deslizante

No trabalho, foi implementado o protocolo de janela deslizante *Go-back-n.* Ele é uma forma de permitir a confiabilidade da rede que não é fornecida pelo protocolo UDP.

2.2.1 Funcionamento

O protocolo cria uma janela de tamanho fixo.

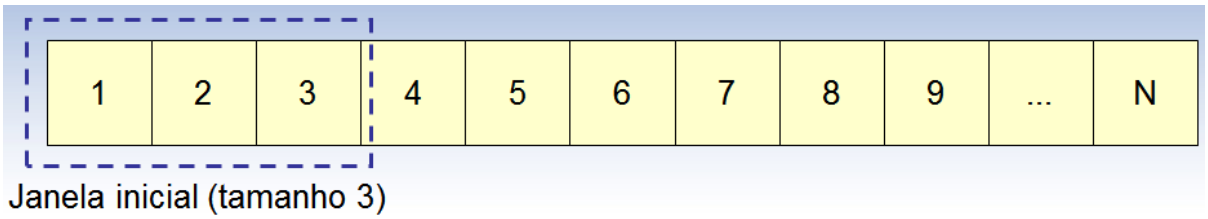

Figura 2 - Janela com tamanho fixo

Em seguida, todos pacotes que se encontram nesta janela são enviados antes de receber uma confirmação (um pacote é dito não-confirmado se foi enviado e nenhum ACK retornou). O máximo de confirmações é o tamanho da janela. Quando o remetente recebe um ACK para o primeiro pacote da janela, a janela é deslizada e o próximo pacote é enviado. A janela continua deslizando de acordo com a chegada das confirmações.

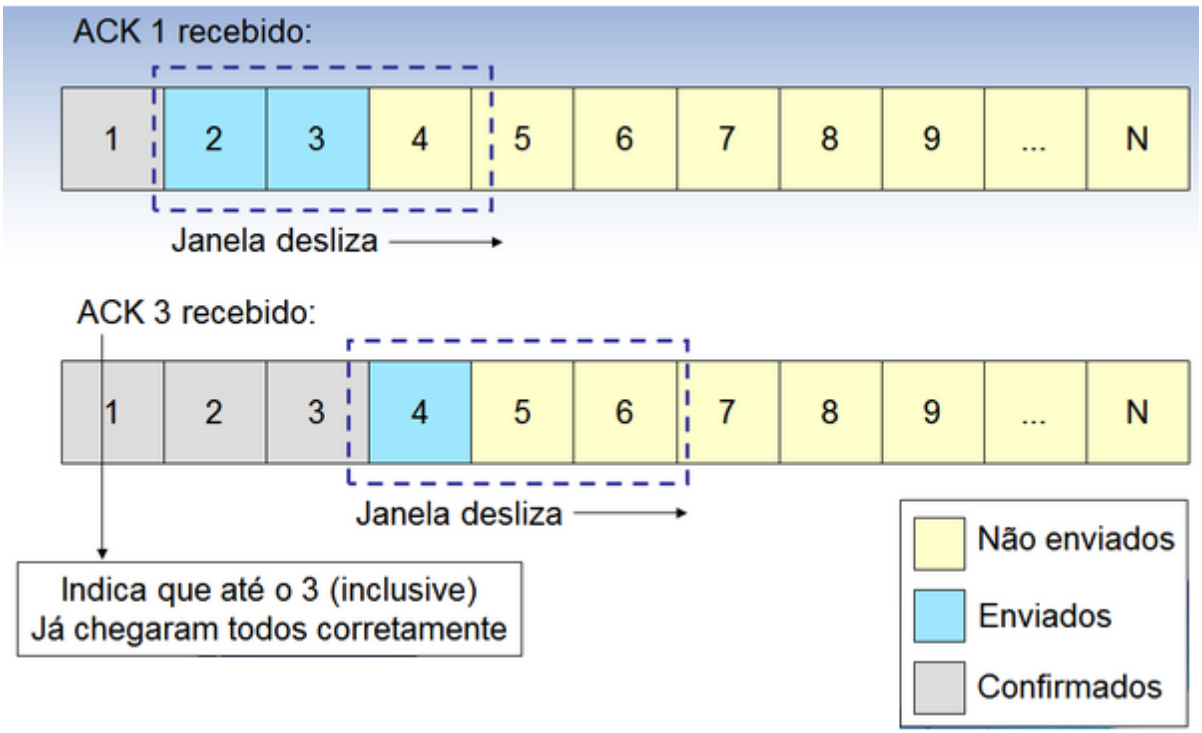

Figura 3 - Janela desliza

2.3 Estrátegia para lidar com erros

A técnica utilizada para detecção de erros é baseada na temporização e na checagem do *checksum* pelo cliente.

2.3.1 Erro de Transmissão

Ao considerar uma temporização de um segundo (*timeout de 1s*) para os pacotes, caso não chegue a confirmação de um pacote esperado, ele é enviado novamente. Para a temporização, foi utilizado uma técnica associada ao recv da biblioteca sockets.

Caso o servidor envie *n* pacotes do tamanho da janela e não receba nenhuma confirmação, ele reenvia todos os pacotes da janela, pois não se sabe qual pacote não foi recebido pelo cliente.

Caso o servidor receba uma confirmação de uma pacote não esperado, levamos em consideração que os pacotes anteriores foram recebidos corretamente e a janela desliza.

Caso o cliente receba uma pacote não seja o esperado, significa que o servidor não recebeu o ACK daquele pacote. Logo, reenviamos a confirmação do pacote quando ele chega novamente e descartamos este pacote.

Caso um pacote danificado chega ao cliente, ele deve ser descartado e enviado novamente. A técnica do *checksum* foi adotada para tratar danificação de pacotes.

2.3.2 Checksum

Dado um pacote, o seu checksum é anexado ao seu cabeçalho antes de ser enviado. Quando o pacote chega no cliente, é feito o *checksum* novamente e os resultados são comparados. Caso os valores sejam diferentes, significa que o pacote foi danificado e ele deve ser retransmitido. O cliente não envia confirmação e temporizador do pacote tem seu tempo esgotado. Assim, o servidor retransmite o pacote quando o temporizador exceda seu tempo.

3 Metodologias

O ambiente de desenvolvimento do código foi via compilador GNU GCC Compiler via linha de comando no Sistema Operacional Ubuntu Linux 13.04. Para gerar os programas executáveis do servidor e do cliente, utiliza-se o comando *make* no terminal do Linux. Os programas gerados possuem nome servidor e cliente e deve aceitar os argumentos descritos da especificação do trabalho. Para testar o trabalho, foi realizado testes apenas com o servidor rodando localmente para testar a comunicação consigo mesmas. Portanto o

endereço IP a ser passado como parâmetro deve ser localhost ou 127.0.0.1 para protocolos IPv4. Como o nome dos arquivos finais terão o mesmo nome e ambos estarão no mesmo diretório, isso gerava conflitos. Logo, o nome do arquivo do cliente será nome_arquivo_(cópia).txt, para diferenciar do arquivo original.

3.1 Medições de desempenho

Foi computado o tempo total gasto, a quantidade total de bytes transmitidos e a velocidade de transmissão. O tempo gasto foi medido utilizando a função gettimeofday e os bytes transmitidos foram medidos com o retorno da função tp_sendto e tp_recvfrom. De posse do tempo gasto e do total de bytes transmitidos, o cliente faz o cálculo da velocidade da transmissão e exibe essas informações no terminal. Além disso, é computado a quantidade de confirmações recebidos e enviadas, pacotes enviados, recebidos e descartados e a quantidade de reenvios. Estes últimos são mostrados no terminal no fim da execução do programa,

3.2 Testes

Cada teste foi executado cinco vezes e os valores apresentados abaixo são a média dos resultados dos testes.

4 Resultados

Alguns testes foram realizados com o programa de forma a verificar o seu funcionamento. Os testes foram realizados em um Atom CPU D525, com 2 Gb de memória. Abaixo seguem os resultados em uma tabela que contém o *thoughput* dos testes realizados no mesmo computador com tamanhos de mensagens e janelas diferentes para um buffer de 10bytes.

Tam. Janela x **Tam. Mensagem**	100 bytes	1000 bytes	4000 bytes
2	15470,02	37437,84	42879,37
4	14376,20	46176,50	44992,89
8	34107,46	50065,18	46285,07
16	34036,18	52742,57	45670,48
32	40225,03	47741,73	48973,40

Exemplo saída dos programas desconsiderando os erros (para mensagem de 100 bytes):

Servidor	Cliente
Reenvios: 0 Pacotes enviados: 11 Pacotes recebidos: 1 Pacotes descartados: 0 ACKs enviados: 0 ACKs recebidos: 11	Reenvios: 0 Pacotes enviados: 1 Pacotes recebidos: 11 ACKs enviados: 11 ACKs recebidos: 0

5 Análise

Foi observado que os resultados obtidos atenderam a espectativas nos casos sem erros. Um fato destacado é que devido os programas estarem na mesma maquina, o processamento necessário para enviar a mensagem (tamanho da janela) é o que mais influencia no *thoughput*. Além disso, percebe-se que a velocidade de transmissão nos testes feitos localmente ficaram bem mais alta que o padrão. Isso acontece porque ambos programas estão na mesma maquina. O desempenho é maximo no momento em que o tamanho da janela consegue guardar a mensagem inteira.

6 Conclusão

Com esse trabalho foi possível implementar um par de programas que operem no modelo cliente-servidor e exercitar a comunicação do tipo requisição-resposta e a transmissão unidirecional sobre o protocolo UDP. Além disso, foi possível praticar a programação com a biblioteca de sockets do Unix para observar como funciona o tratamento de erros do protocolo UDP.

Devido ao tempo tempo corrido de final de semestre e algumas dificuldades de entendimento do protocolo de janela deslizante, não foi possível realizar testes que simulam erros no canal de transmissão. Além disso, a verificação do *checksum* e a temporização não foram implementados completamente. A teoria pode parecer simples, porém a implementação possui vários detalhes de funcionamento que a torna complexa de se codificar.

No mais, o desenvolvimento desse trabalho foi muito importante para consolidar a matéria vista em sala de aula e colocar em pratica a teoria por trás das redes de computadores.