Documentação TP3
Sockets UDP, janela deslizante

André Henriques Rodrigues [2016057941]
Luciano Otoni Milen [2012079754]

1- Introdução

Neste trabalho prático deve-se implementar dois programas que estabelecem uma conexão entre si via UDP. O servidor cria uma porta disponível que será acessada pelo cliente, que requisita um arquivo a ser transferido pelo servidor, onde há um tamanho de buffer especificado. A transferência deve ser realizada utilizando o método da janela deslizante, onde é assegurado que nenhuma informação será perdida e que os pacotes serão transmitidos na ordem correta. Este protocolo permite, através da janela deslizante, que o servidor transmita mais de um pacote antes de receber um ACK, o que aumenta a eficiência da transmissão. O termo "janela" no transmissor representa o limite lógico do total de pacotes que serão reconhecidos (ACK) pelo cliente[1]. Em casos de *timeout*, por exemplo, o servidor retransmite os pacotes enviados enquanto aguarda um ACK do cliente.

2- Metodologia

Os experimentos foram realizados da seguinte forma:

a) Foi criado um arquivo teste que contém um texto qualquer;
b) O servidor é iniciado em uma porta aleatória (6001, por exemplo), um tamanho_de_buffer, como 100, e um tamanho de janela, como 8;
c) O cliente é iniciado com o endereço do host , como localhost por exemplo. Os parâmetros de chamada são hostname , porta , nome_do_arquivo (teste , no caso), tamanho_de_buffer (100) e tamanho_da_janela (8). A conexão então é estabelecida;
d) O servidor recebe o nome_do_arquivo, confirma com um ACK e o abre no diretório em modo de leitura;
e) O cliente recebe o ACK do servidor e aguarda a transmissão dos *buffers* particionados do arquivo, enquanto houverem *bytes* a serem transmitidos;
f) O servidor envia os pacotes conforme couberem no tamanho_da_janela especificado. Se cabem 8 pacotes, por exemplo, ele envia os 8 e então espera o ACK do cliente. Caso haja um *timeout* ou perda de pacotes, o servidor reenvia os pacotes que foram transmitidos naquela janela.

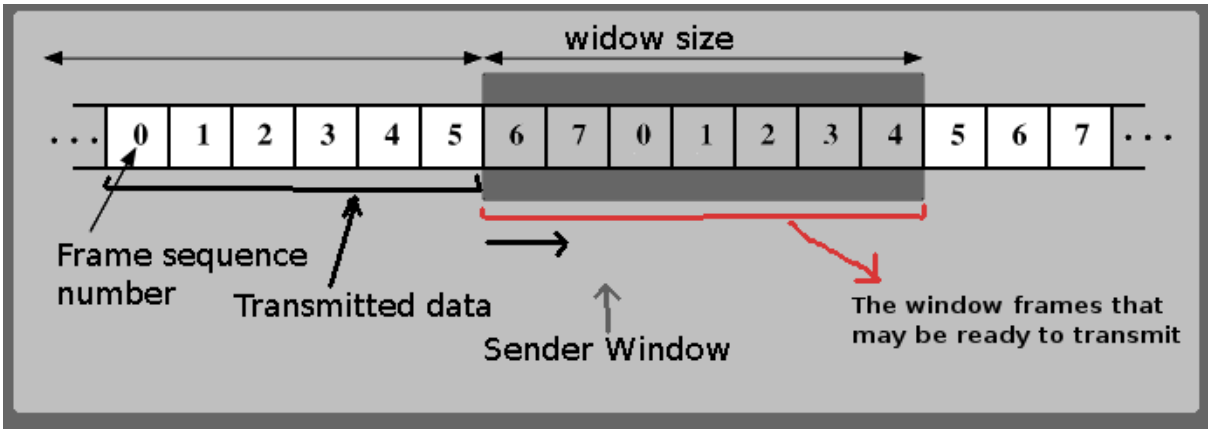
g) Conforme os *buffers* chegam o cliente confirma o recebimento dos pacotes enviados na janela com um ACK e o número do ID do pacote recebido;
h) A conexão é encerrada quando o servidor envia uma confirmação do tipo FINAL e o cliente confirma um ACK, fechando o *socket*;
i) As estatísticas são impressas na tela e o arquivo de cópia salvo no cliente. É importante notar que o tratamento de falha no recebimento e envio dos pacotes é feito, conforme será descrito a seguir.

O programa foi elaborado e testado em duas máquinas: uma delas era Ubuntu GNOME Linux 17.04 Intel® Core TM i7-3610QM CPU @ 2.30GHz × 8 com 8gb de memória RAM, em uma rede banda larga wireless de 15MB/s de download (2MB/s de upload) e a outra rodando Linux Mint 19 Intel® Core TM i5-5210QM CPU @ 1.90GHz × 4 com 4gb de memória RAM. As medições foram feitas utilizando o *gettimeofday* para calcular o tempo decorrido na transferência. O teste principal foi executado 15 vezes e diversos testes menores foram feitos durante o desenvolvimento dos programas. O teste principal é composto por um arquivo sem formato específico contendo um texto Lorem Ipsum gerado aleatoriamente com 1 mil *bytes*.

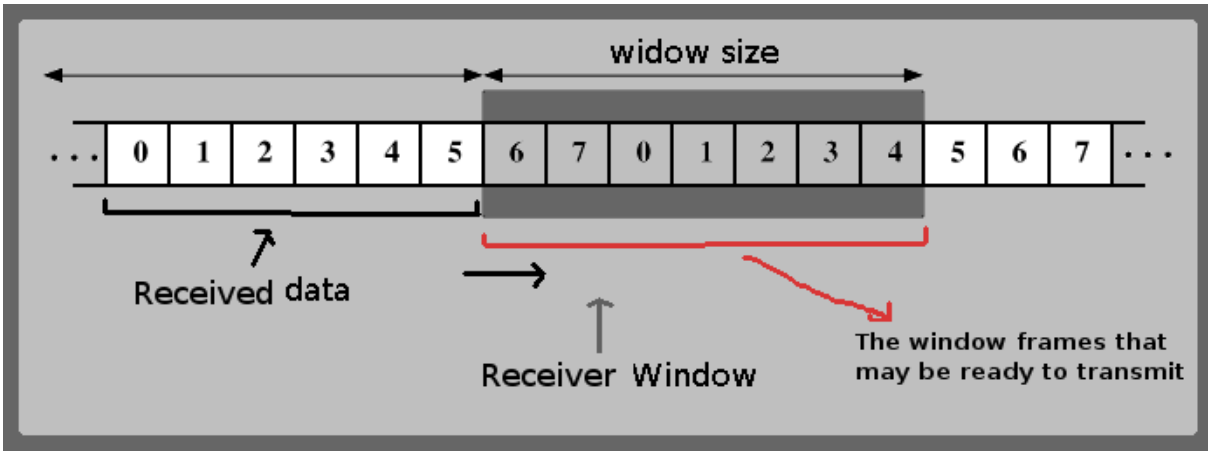
Uma parte importante é a formatação dos pacotes. O esquema segue a tabela a seguir:

PACKET (BYTES)			
PACKET_ID	PACKET_TYPE	CHECKSUM	PACKET_DATA
4	1	1	buffSize

O código está todo documentado, com o passo a passo do que é feito na execução. Para rodá-lo, basta utilizar o comando *make* no Linux para compilar os arquivos e executá-los na ordem servidor -> cliente, passando informações como hostname , porta, tamanho do buffer, arquivo da transferência e tamanho da janela. Após a transferência ser completada, um arquivo com o mesmo nome adicionado de "_" é gerado.



Exemplo de comunicação via janela deslizante do lado do servidor. [3]



Exemplo de comunicação via janela deslizante do lado do cliente. [3]

A ideia de "deslizante" vem da própria confirmação de recebimento dos pacotes. O servidor agrupa diversos pacotes em uma janela. Considerando uma janela de tamanho 6, como na imagem acima:

1. O servidor inicia o *socket* e aloca memória para a janela. O cliente conecta no mesmo *socket*, onde é feita a transação.
2. O servidor agrupa os 6 primeiros pacotes em uma janela e a envia para o cliente.
3. O cliente recebe os pacotes e confirma o último pacote com um ACK, ou seja, todos os pacotes foram recebidos.
4. A função de deslocamento da janela para agrupar os próximos pacotes é a seguinte:
5. `win_slide = (win_slide + 1) % window_size`
6. Então, a janela é preenchida com novos pacotes a serem enviados ao cliente

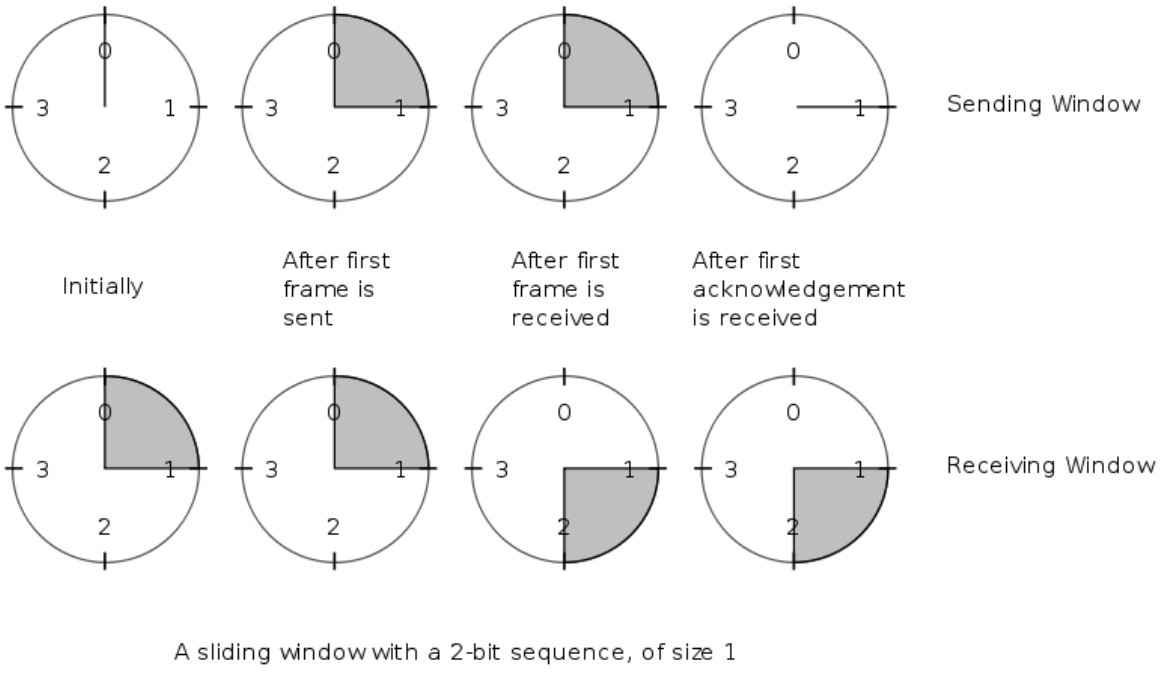


Ilustração da janela deslizante em funcionamento[1]

Um importante detalhe é o que acontece nas falhas de transmissão. Se forem enviados 6 pacotes e não chegue nenhum ACK no servidor, os 6 pacotes são retransmitidos. Isto se deve ao fato de que o servidor não tem controle se algum dos 6 pacotes (exceto o último) foi recebido corretamente, pois o ACK vem no pacote final.

Caso o servidor envie um pacote que não era o próximo aguardado pelo cliente, este último entende que o servidor não recebeu o ACK que o cliente enviou. Desta forma, o cliente envia um ACK para o servidor com o identificador do último pacote recebido corretamente, para que a retransmissão seja feita.

O *checksum* é feito para garantir a integridade do pacote. Caso o valor calculado seja diferente do esperado, o pacote deve ser re-transmitido, como é sinalizado pelo cliente.

Os tratamentos de erro são feitos na função `getBuff(char *buffer, int buffSize)`. São 3 casos possíveis:

1. O cliente recebe um pacote que já tinha sido recebido anteriormente, ou seja, o ACK do pacote não chegou no servidor. Neste caso, o cliente retransmite o ACK para o servidor para que a transferência possa continuar.
2. O pacote recebido é do tipo *FINAL_TYPE*. O cliente responde com um pacote do mesmo tipo para encerrar a conexão.
3. O caso esperado, o pacote veio corretamente. O cliente retorna um pacote com ACK para o servidor, incrementando a quantidade de ACKS enviados e o número do próximo *packID* a ser recebido, além de escrever o *buffer* no arquivo de saída.

Uma breve descrição da estrutura do código: uma biblioteca `common.h` junta as funções de uso comum do servidor e cliente. As funções específicas de cada um estão em seus respectivos arquivos. Além disso, todas importam a `tp_socket.h` fornecida pela especificação do trabalho prático.

As funções relacionadas à comunicação via *socket* são feitas através dos arquivos disponibilizados na especificação do trabalho. A função `timer(unsigned int sec)` é responsável pela temporização na transferência de arquivos. Utiliza a função `setsockopt()`[2] que seta no *socket* o valor máximo de segundos para tolerar até emitir um *timeout*.

3- Resultados

Seguem abaixo informações gráficas a respeito do desempenho dos programas. A tabela identifica alguns resultados obtidos nos testes realizados. Cada experimento foi executado 10 vezes e o valor mostrado representa a média obtida. O ponto de interesse é variar o tamanho da janela, mensagem e *buffer*.

Experimento	Tamanho Janela	Tamanho Mensagem (B)	Tamanho Buffer (B)	Número de Pacotes	Tempo Decorrido (s)	*Throughtput* (kbps)
1	300	1000	100	12	0.000391	2560059.94
2	1100	1000	600	3	0.000231	4332815.59
3	4000	1000	1600	2	0.000132	8239582.02
4	300	4000	100	42	0.000776	8411665.14
5	1100	4000	600	8	0.000480	14337088.55
6	4000	4000	1600	4	0.000366	10965304.20
7	300	100000	100	1005	0.033561	2989452.82
8	1100	100000	600	169	0.002952	33988395.61
9	4000	100000	1600	64	0.001811	55398937.07

Experimento 1: No caso base, com a mensagem de tamanho 1000, a execução é ágil. Entretanto, nos experimentos seguintes, o aumento da janela acelerou bastante a transferência dos arquivos.
Experimento 2: Conforme dito no experimento 1, a janela aumentou quase 3x e o tempo diminuiu consideravelmente.
Experimento 3: Com um buffer muito grande e uma janela imensa, a transferência foi praticamente instantânea.
Experimento 4: Nos experimentos seguintes onde a mensagem possui 4000 Bytes o tempo de transferência foi bem maior.
Experimento 5: O experimento 5 acompanhou as expectativas, conforme esperado no experimento 2. A janela cresceu bastante de tamanho, mas os pacotes acompanharam tal crescimento.
Experimento 6: Este é um caso curioso, onde a janela tem o mesmo tamanho da mensagem. Entretanto, o tamanho do buffer anda exige 4 pacotes. Se tivéssemos um tamanho de buffer igual ao da janela e da mensagem, a transferência seria como o caso do *stop-and-wait*.
Experimento 7: Este foi o experimento mais lento de todos, ainda que tenha ocorrido em menos de 0.5 segundos.
Experimento 8 e 9: Este caso foi realmente interessante. O tempo do experimento 9, que conta com mais que o triplo do tamanho da janela e de *buffer*, foi muito inferior ao experimento 8.

Os resultados foram de certa forma surpreendentes. A variação no tamanho da janela e do *buffer* surtiu efeitos esperados, mas a janela faz muita diferença na velocidade de transmissão. É notável o aumento de desempenho se compararmos com o *stop-and-wait*, que é em si o caso mais trivial da janela deslizante (tamanho 1). Arquivos imensos, de +30mb, travam o lado do cliente, sem explicação alguma. Para arquivos até de 1mb o código funciona perfeitamente.

5- Conclusão

Neste trabalho foi possível entender como funciona o protocolo UDP. Sua implementação requer bem mais empenho que o TCP, feito no primeiro exercício. A troca de pacotes deve sempre ter a informação do tipo do pacote, seja ele ACK, DATA_TYPE ou FINAL_TYPE. É a única forma de garantir que a troca de mensagens seja feita na ordem correta.

Em relação ao TP2, a janela deslizante é sem dúvida mais interessante que o *stop-and-wait*, pois permite que mais pacotes sejam enviados antes de esperar a resposta do cliente. Garante mais agilidade no processo, principalmente se considerarmos que as redes hoje são de alta velocidade, logo é possível enviar mais pacotes de uma vez só.

É um protocolo difícil de implementar. Devem ser feitos tratamentos de erro, como o ACK não chegar a tempo do próximo envio de um pacote pelo servidor. O re-envio de pacotes é feito mais frequentemente, já que a janela possibilita ainda que mais velocidade, mais chance de erro, já que mais pacotes são enviados simultaneamente. O arquivo chega no cliente com todos seus dados.

6- Referências

[1] https://en.wikipedia.org/wiki/Sliding_window_protocol

[2] http://pubs.opengroup.org/onlinepubs/009695399/functions/setsockopt.html

[3] http://wikistack.com/what-is-tcp-sliding-window-protocol/

Slides dos professores Marcos e Loureiro

https://linux.die.net/man/7/socket

https://stackoverflow.com