Universidade Federal de Minas Gerais
Rede de Computadores

Arthur Phillip D. Silva
Gabriel Almeida de Jesus

Trabalho Prático 1

Introdução

O trabalho consiste em criar um servidor e um cliente que sejam capazes de transmitir e armazenar logs. E para isso devemos implementar os conceitos de janela deslizante, confirmações de mensagem e verificação de erros, que foram vistos em sala, além dos conceitos que foram vistos e usados no TP0, como encode/decode, funcionamento de um servidor (No caso desse TP foi usado UDP), etc.

Desafios

Nosso primeiro desafio surgiu quando tentamos implementar a criação do pacote a ser enviado, os detalhes de empacotamento e desempacotamento apresentam um certo nível de dificuldade de compreensão do que está sendo gerado e como essa informação trafega na rede, foi necessário compreender como a abstração dos dados originais se comportavam e como a informação podia ser concatenada de forma a formar um pacote único para concluir a tarefa conforme a especificação do enunciado.

Outro desafio foi lidar com as janelas deslizantes, a complexidade se deve ao comportamento não linear que ocorrem em alguns casos, por exemplo, quando uma mensagem é recebida e gravada pelo servidor, mas a mensagem de confirmação vai truncada para o cliente, a janela do cliente não pode andar, e o servidor já a pagou mensagem da janela e se não for feito um tratamento para confirmação de mensagens antigas, pode ocorrer um deadlock.

A concorrência foi um terceiro ponto a ser vencido, o servidor conseguia tratar os vários clientes de forma linear, por fazer um cálculo relativamente rápido de análise do pacote e envio da confirmação consegue tratar alguns clientes de de forma serial. O cliente porém, deve mandar e receber os pacotes paralelamente, já que uma parte tenta enviar todos os pacotes que a janela de confirmação consegue guardar e outra deve ficar parada aguardando o recebimento da confirmação. O primeiro popula a janela enquanto o segundo a esvazia, um problema de concorrência de escrita que resolvemos com o uso de lock, que trava a estrutura impedindo acesso simultâneo.

Ter que saber lidar com os diversos casos que podem ocorrer, e implementar isso se torna um desafio interessante e também é uma boa oportunidade de treinar o que foi visto em sala.

Comportamento no Caso Anômalo

Verificar se o hash do pacote está correto:
Se sim :
Verificar se o id da mensagem é menor que o máximo da da janela
Se sim:
Calcular o hash do pacote com a probabilidade de Erro
Enviar pacote de confirmação

Verificar se a mensagem já foi gravada
Se não:
Verificar se a mensagem já está na janela
Se não:
Verifica se a janela está cheia
Se não:
Inserir a mensagem na janela do cliente

Senão:
Verifica se a mensagem já foi gravada
Se sim:
Calcular hash do pacote com a probabilidade de Erro
Envia confirmação

Esse é um pseudocódigo do nosso programa que determina como essa anomalia é tratada. Podemos ver nos terceiro e quarto condicionais que é verificado se a mensagem já foi gravada e se ela está na janela, e os próximos passos só acontecem caso a mensagem não esteja na janela, portanto, na anomalia em que mesmo número de sequência e hashs corretos são passados, o nosso servidor interpretaria que a mensagem já foi recebida e por tanto não a reescreveria. No caso das mensagens já gravadas, uma confirmação é enviada novamente mesmo que o hash esteja errado, pois o servidor interpreta que o cliente não recebeu a confirmação, mas ele já recebeu pelo menos um hash correto dessa mensagem.

Gráficos de Desempenho

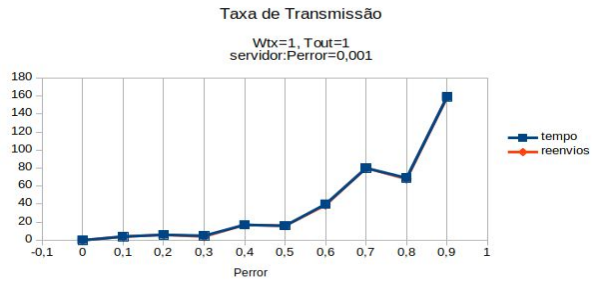

fig.1

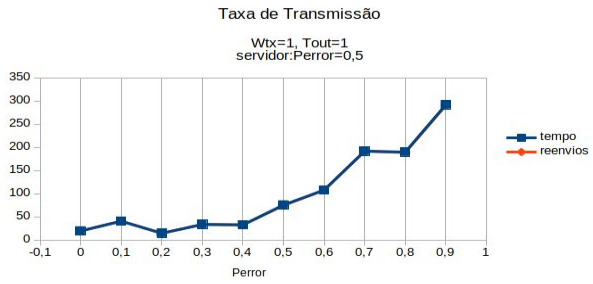

fig.2

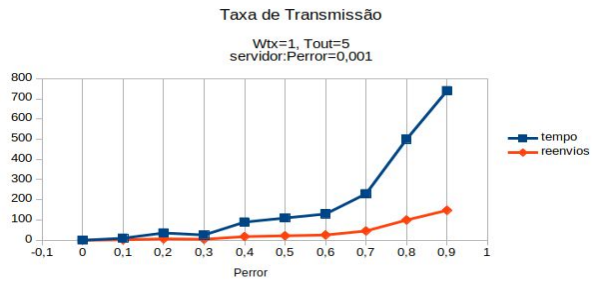

fig.3

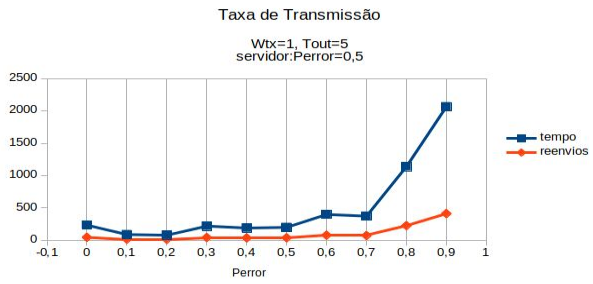

fig.4

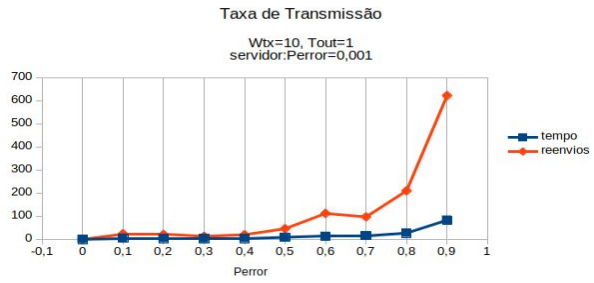

fig.5

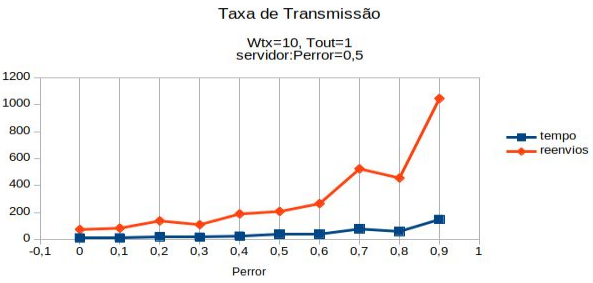

fig.6

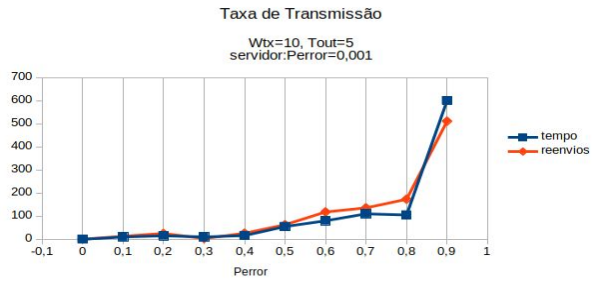

fig.7

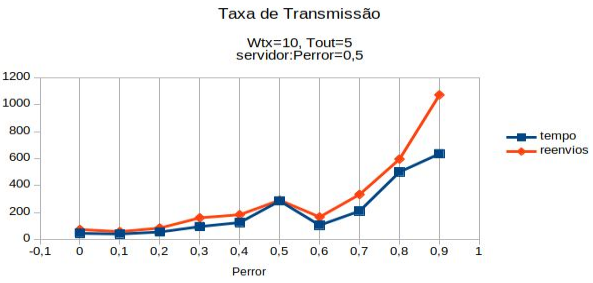

fig.8

Para realizar o teste utilizamos um script que realizava várias chamadas clientes sequencialmente. Gradativamente aumentou-se a probabilidade de erro (0,0-0,9), o tamanho da janela (1-10) e o *time out* (1-5) e para cada chamada gravamos o tempo total de execução, o número de reenvios e *time out* total e por mensagem, e os erros de hash no envio e recebimento dos pacotes.

Quando a janela tem tamanho 1 o sistema funciona como um *stop-and-wait,* pois cada pacote só pode ser enviado quando se recebe a confirmação do anterior, nesse caso o tempo gasto está diretamente relacionado ao *time out* de reenvio multiplicado ao número de erros de envio, como podemos ver nas figuras 1 a 4. Quando aumenta-se o tamanho da janela o número de reenvios é quase sempre maior que o tempo, pois o cliente pode enviar várias mensagens ao mesmo tempo, mas mesmo nesses casos, apenas quando o *time out* é muito alto o tempo total se assemelha a quantidade de reenvios (figuras 5 a 8).

Aumentar o erro do servidor muda a escala do gráfico, mas não sua forma, o que pode ser notado comparando os gráficos da direita com os da esquerda.

Instruções de Compilação

Segue o mesmo padrão da documentação.

Comando para rodar cliente
python client.py <arquivo> <IP>:<port> <Wtx> <Tout> <Perror>

Comando para rodar servidor
python server.py <arquivo> <port> <Wrx> <Perror>

Conclusão

O trabalho foi de grande importância no entendimentos do funcionamento das janelas deslizantes, pois, ao implementar essa função e lidar com os desafios encontrados em implementá-la, fixamos melhor como é o funcionamento do método e compreendemos seus pequenos detalhes, que são de grande importância para determinar se tudo vai funcionar de forma correta, mesmo se pacotes são perdidos durante as trocas de mensagens. Outro avanço na maturidade do conhecimento em Redes, veio na implementação do envio e recebimento dos pacotes, entender o que devia ser enviado e como a informação deveria ser enviada de forma que pudesse ser separada e compreendida do outro lado, foi um passo importante na definição de como o trabalho deveria ser feito. Esses foram apenas dois exemplos dos avanços que tivemos que desenvolver para concluir o trabalho, mas além desses muitas outras coisas tiveram que ser superadas, como por a parte relativa a concorrência (Descrita em “Desafios”.)

Quanto ao desempenho, na parte dos gráficos podemos ver como evitar erros é importante, seja evitar erros usando um algoritmo mais eficiente, ou um meio de comunicação mais confiável, pois, nos gráficos podemos ver como o aumento dos erros interfere na comunicação entre cliente e servidor.