Documentação TP1

Gabriel Soares
Mariane Fernandes

4 de Novembro de 2018

1 Introdução

O objetivo deste trabalho é desenvolver uma biblioteca para criação, transmissão e armazenamento de logs. Ela deve permitir que um dispositivo cliente gere mensagens de log e as envie para um servidor. A biblioteca deve realizar a codificação e transmissão confiável de mensagens seguindo os protocolos de janela deslizante e confirmação seletiva. Além disso, a comunicação deve ser feita através de um soquete UDP.

2 Mensagens

O formato das mensagens transmitidas foi definido na especificação do trabalho da seguinte forma:

- Pacotes enviados pelo cliente::
  1. Número de sequência da mensagem em 8 bytes e sem sinal
  2. *Timestamp* com o instante de tempo em que a mensagem de log foi lida do arquivo de entrada. Ele é transmitido como dois inteiros de 8 e 4 bytes, sem sinal, onde o primeiro representa a quantidade de segundos desde a data de referência do período corrente, e o segundo inteiro a quantidade de nanosegundos desde o início do segundo atual
  3. Tamanho da mensagem como um inteiro de 2 bytes, sem sinal
  4. Mensagem enviada, que deve conter apenas caracteres ASCII e um tamanho máximo de 2^{14} bytes
  5. Código de verificação em forma de um hash MD5, 16 bytes, calculado sobre os 4 campos anteriores
- Confirmação, ou ACK, do servidor:
  1. Número de sequência da mensagem sendo confirmado em 8 bytes
  2. O *timestamp* original da mensagem sendo confirmada, em 8 e 4 bytes.
  3. Código de verificação em forma de um hash MD5, 16 bytes, calculado sobre os 2 campos anteriores

3 Código

O código da biblioteca foi dividido em 4 classes: cliente, servidor, pacote e *buffer*. Dentre elas, apenas as classes cliente e servidor contêm o método *main* para serem executadas.

3.1 Pacote

Esta classe contém os atributos exigidos para construir o pacote que será transportado pela rede. Ela também inclui dois atributos adicionais, um que indica se o pacote já foi enviado alguma vez e outra que indica se o pacote já foi confirmado.

A classe também contém 3 métodos: um responsável por verificar se o hash passado como parâmetro é igual ao hash do pacote, outro que gera o pacote para a rede de acordo com o formato definido na especificação e o último que gera um pacote a partir dos dados recebidos em formato de rede.

3.2 Buffer

Esta classe fornece a estrutura necessária para simular uma janela deslizante. Os dados são armazenados em formato de fila circular, sendo assim, a fila contém espaço caso o primeiro lugar esteja vago. A classe também contém um método de inserção ordenada, que é utilizado pela classe servidor, e outro para verificar se algum número está dentro da janela deslizante.

3.3 Cliente

Esta é a classe responsável por enviar as mensagens para o servidor. Ela deve ser executada através do comando: *python3 cliente.py arquivo IP:port Wtx Tout Perror* onde os parâmetros representam, respectivamente, o nome do arquivo onde estão as mensagens, o ip e porto do servidor, o tamanho da janela de transmissão, a duração do temporizador e a probabilidade de enviar um pacote com erro.

A execução do cliente é dividida em duas *threads* principais. A primeira *thread* é responsável por ler as mensagens que estão armazenadas no arquivo de entrada. Ela lê uma linha do arquivo, cria o pacote correspondente e o armazena em uma fila de espera do cliente.

A segunda *thread* é responsável por fazer o gerenciamento dos envios. Ela dispara uma *thread*, que é a responsável por escutar o servidor e confirmar os ACK's, e executa um *loop* enquanto existir algum pacote dentro da janela deslizante. Dentro deste *loop*, a *thread* preenche a janela deslizante com os pacotes da fila de espera, dispara uma nova *thread* para cada pacote ainda não enviado e retira da janela os pacotes que já foram confirmados. Ao final do *loop*, a *thread* notifica o ouvinte que a transmissão terminou.

A *thread* disparada para cada pacote cuida da transmissão do mesmo. Ela faz o envio do pacote e dorme pelo tempo do temporizador, esse processo se repete até que o pacote pela qual é responsável seja confirmado.

3.4 Servidor

Esta é a classe responsável por receber os pacotes enviados pelos clientes. Ela deve ser executada da através do comando: *python3 servidor.py arquivo port Wrx Perror*, onde os parâmetros representam, respectivamente, o nome do arquivo de saida, o porto do servidor, o tamanho da janela deslizante e a probabilidade de enviar uma pacote com erro.

A execução do servidor é dividida em duas *threads* principais. A primeira *thread* é responsável por escrever as mensagens da fila de espera no arquivo de saída do servidor, enquanto a outra é responsável por receber os pacotes.

Ao receber um pacote, caso o md5 esteja correto, o servidor dispara uma *thread* para tratar o pacote. A nova *thread* recebe a janela deslizante - que é criada ao receber o primeiro pacote de um novo cliente e armazenada em um dicionário-, o endereço do cliente e o pacote recebido. Ela verifica se o pacote pode ser confirmado e, em casos positivos, adiciona o pacote na janela deslizante, gera e envia o ACK para o cliente. Como o md5 já foi verificado, a *thread* insere o pacote apenas verificando se ele está dentro dos limites da janela, sendo assim, ela não verifica se a mensagem é repetida ou se md5 é igual a de outro pacote.

Antes de enviar a confirmação, a *thread* movimenta a janela deslizante do cliente tirando os pacotes da janela e os colocando na fila de espera para que possam ser escritos no arquivo de saída, como somente o primeiro pacote é tirado, o servidor sempre escreve os pacotes na ordem correta.

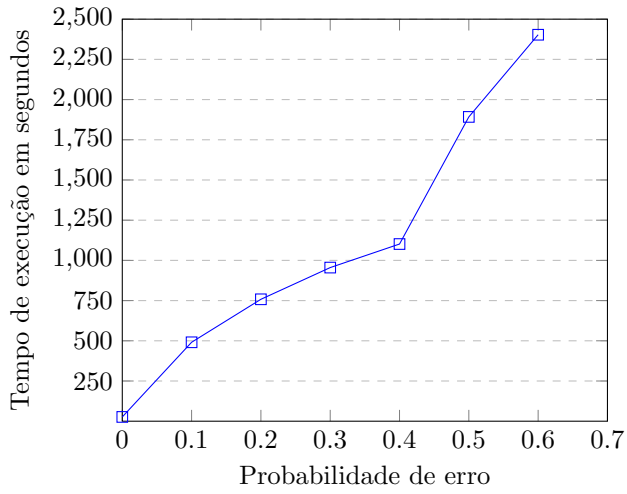
O servidor pode ser finalizado pelo comando de interrupção de teclado *crlt-c*, entretanto o processo pode demorar um pouco devido a um *timeout* para escrever no arquivo de saída. Devido a isso, o arquivo de saída só será criado de forma correta caso o servidor termine corretamente.

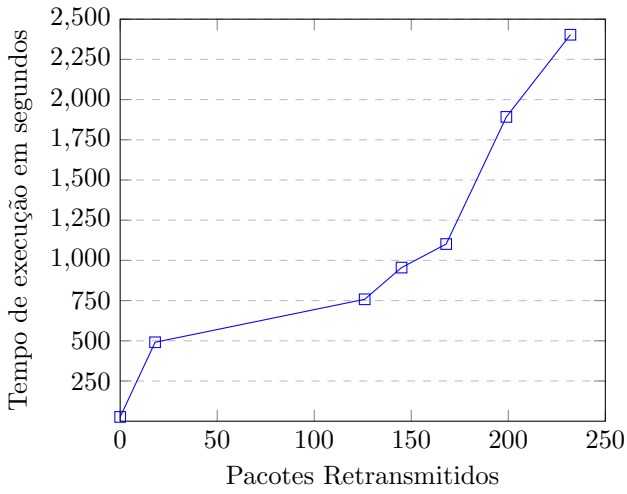
4 Análise de Desempenho

Os testes foram feitos em um i5 1.6GHz, com uma memória de 6 GB DDR3 L e rodando Ubuntu 18.04.1 LTS. Os arquivos de mensagens foram gerados aleatoriamente respeitando o tamanho máximo de entrada.

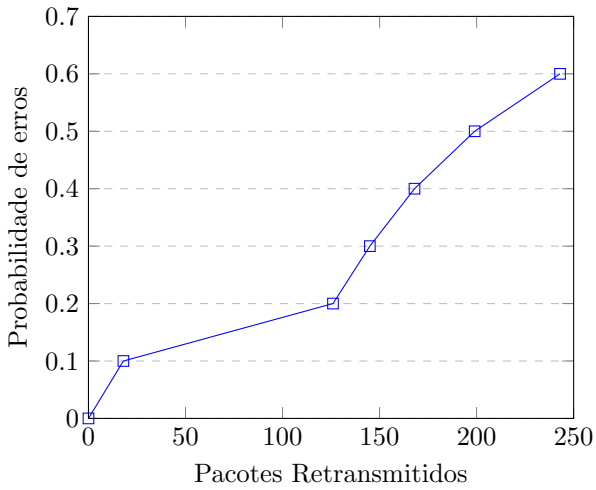
O testes foram feitos alterando apenas a probabilidade de erro do cliente, o arquivo de mensagens continha 100 linhas de no máximo 100 caracteres. O cliente tinha uma janela deslizante de 5 pacotes e 40 segundos de temporizador, enquanto o servidor tinha uma janela deslizante de 50 pacotes e 0% de chances de enviar uma confirmação errada. Vale ressaltar que a espera ocupada do cliente, necessária para manejar a janela deslizante e dar inicio as *threads* de transmissão, altera o consumo de CPU e isso afeta o desempenho de forma geral.

Os resultados mostram que retransmissões de pacote pioram consideravelmente os tempos de execução, mesmo considerando a melhora do desempenho após a repetição dos teste. Além disso, foi possível notar a retransmissão de pacotes já confirmados pelo servidor e também alguma demora entre a chamada da função de envio e a real transmissão dos pacotes. A melhor hipótese para ambas as constatações está ligada ao escalonamento das *threads*, como não existe nenhuma ordem de prioridade, os pacotes podem ser retransmitidos antes do cliente terminar o processo de confirmação dos mesmos. A *thread* que escuta o servidor também pode utilizar o soquete em sequência mesmo sem ter recebido algo, fato que atrasa o envio dos pacotes.





A respeito da relação entre probabilidade de erro no envio de um pacote e a quantidade de retransmissões de pacotes, os resultados estão dentro do esperado.



5 Conclusão

Nesse trabalho, foi possível ver o funcionamento das técnicas de transmissão sem conexão apresentadas em sala de aula. Durante o desenvolvimento, um dos maiores desafios foi descobrir como funcionava as bibliotecas para enviar e receber os pacotes pela rede. Outro problema enfrentado foi realizar a integração entre cliente e servidor, apesar de estarem com a lógica correta, os teste não funcionaram nas primeiras tentativas devido a pequenos detalhes de implementação e a erros na utilização das bibliotecas de redes. Também existiu alguma dificuldade de sincronizar o acesso à janela deslizante, como solução, decidimos trabalhar com uma fila de espera auxiliar. Dessa forma, como a estrutura já apresenta recursos de acesso concorrente, não foi necessário fazer a sincronização diretamente na janela deslizante.