Bruno Pires Naschpitz

Jorge Ricardo Jaú Junior

Vinicius Bastos Bittencourt

**Construindo um simulador do protocolo TCP**

Trabalho realizado para a disciplina Avaliação e Desempenho, do curso de Ciência da Computação na Universidade Federal do Rio de Janeiro.

Professor:

Paulo Henrique de Aguiar Rodrigues

Rio de Janeiro

Julho de 2013

1. **Introdução**

Este trabalho tem como objetivo consolidar os conhecimentos adquiridos na disciplina de Avaliação e Desempenho através do desenvolvimento de um programa que simulará o comportamento do TCP. Intervalos de confiança, geração de amostras, processos Poisson e distribuições de probabilidade estão entre os conceitos aprendidos na cadeira que iremos utilizar aqui.

As especificações desse cenário são as seguintes:

* **n** pares de transmissores e receptores TCP;
* Servidores transmissores possuem uma taxa de transmissão de **1 Gbps**;
* Há um gargalo, que é o roteador, cuja saída possui taxa de **10 Mbps**;
* Pacotes possuem tamanho máximo de **1500 bytes (12000 bits)**;
* Dois valores possíveis para o tempo de propagação entre servidor e roteador: **100ms** (grupo 1) e **50ms** (grupo 2);
* Tempo de propagação entre roteador e estações é **desprezível**;
* Tempo de propagação dos ACKs podendo ser de **100ms (grupo 1)** ou **50ms (grupo 2)**;
* Tráfego de fundo com taxa média **de 5 Mbps** (pacotes de **1500 bytes** chegando com distribuição geométrica de tamanho médio **10** e com as rajadas chegando segundo processo Poisson com intervalo médio de **24 ms**.
* Armazenamento de até **40** pacotes do tamanho máximo no roteador, incluindo o pacote em serviço no tempo atual.

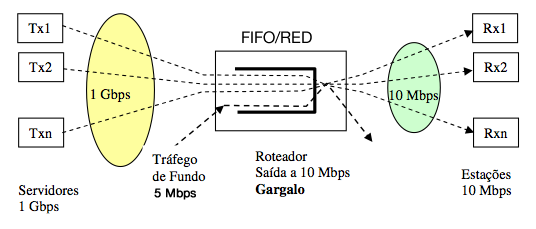


Figura 1. Modelo a ser simulado

O simulador que construímos funciona tanto com a política **FIFO** quanto com a política **RED** para tratar do gargalo.

Nós estamos interessados em avaliar o comportamento do roteador nos cenários descritos acima, entretanto, esses valores são variáveis de entradas que podem ser alteradas conforme for necessário se desejarmos simular para outras condições.

Iremos avaliar o comportamento de FIFO e RED em diversos cenários, onde compararemos sessões onde há um único transmissor do grupo 1 com sessões onde há uma única do grupo 2; 10 sessões em paralelo onde metade é de cada grupo; e 100 sessões todas de um mesmo grupo. Estamos interessados em calcular a vazão média dessas sessões.

O TCP é um dos protocolos mais usados em redes. Observar seu comportamento em diferentes cenário não é apenas um trabalho de Avaliação e Desempenho mas também um bom aprendizado para Teleprocessamento e Redes, apesar de alguns de nós já terem feito o curso.

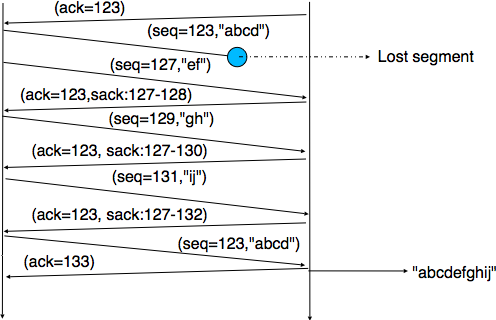


Figura 2. Exemplo de funcionamento do Selective ACK

Optamos por usar a linguagem Java para o desenvolvimento deste simulador. O motivo por essa escolha se dá ao fato desta linguagem já possuir estruturas de dados bem definidas, incluindo filas com prioridades, que são necessárias para a criação da lista de eventos.

Embora linguagem procedurais como C sejam mais rápidas em questão de tempo de execução, acreditamos que a orientação a objeto facilita muito quando precisamos criar um modelo.

Durante o desenvolvimento do trabalho, usamos nomes grandes tanto para as variáveis quanto para as funções, de maneira que podemos identificar o que essa variável ou função faz apenas de bater o olho no nome. Isso não prejudica absolutamente nada no desempenho do programa e nos ajuda a nos organizar. Já que podemos dar nomes longos e intuitivos, por que não fazer isso? (dica aprendida na disciplina de *Desenvolvimento Ágil*)

Importamos para nosso projeto **bibliotecas matemáticas de domínio público** disponibilizadas na Internet pela **Universidade de Montréal** *(dica dada por colegas de outro grupo)*. Com essas bibliotecas podemos obter rapidamente através de uma chamada a uma função, valores da Tabela Normal e da *t-Student*, o que facilita a parte do processo que utiliza Intervalos de Confiança. Como tais pacotes são datados de 1999, é normal que o Java indique através de *Warnings* durante a compilação que existem métodos depreciados, mas isso é apenas uma questão de versão e não atrapalhará o funcionamento do simulador.

A biblioteca matemática nativa do Java (pacote *Math*) nos permite usar raiz quadrada, potência, arredondamento (útil para geração de números aleatórios) dentre outros.

O pacote *java.util.random* nos forneceu bons recursos para a geração de números aleatórios, sendo necessário apenas passar para a forma de código a forma das inversas das distribuições acumuladas (CDFs) da exponencial e da geométrica. O *random* do Java gera automaticamente um número do tipo *double* uniformemente distribuído entre 0 e 1, ou seja, o u0 que usaremos para a geração na distribuição desejada. E as inversas necessárias estão disponíveis na apostila do curso, capítulo *Geração de Distribuições*.

O simulador pode ser disparado ao executar o projeto em qualquer ambiente de desenvolvimento Java. Não desenvolvemos interface gráfica para esse simulador pois, além de ser um trabalho demorado, prejudica a performance do simulador em tempo de execução. Entretanto, fizemos o console interagir com o usuário, pedindo explicitamente o valor de cada variável de entrada, que também é digitada usando o console.

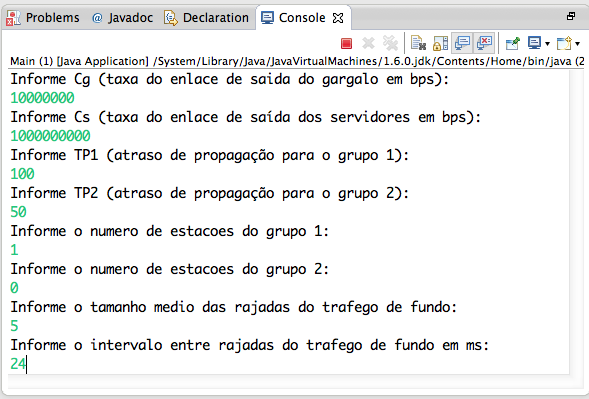


Figura 3. Exemplo de Entrada

Optei por colocar a saída do programa em um arquivo de formato CSV (comma separated values), pois arquivos desse tipo podem ser lidos pelo Microsoft Excel, que gera gráficos. Assim fica mais prático observar o comportamento do sistema.

Java é uma linguagem que roda sobre uma máquina virtual, de maneira que para executar esse simulador, é necessário ter o Java Runtime Environment instalado em seu computador. O mesmo pode ser obtido em <http://www.java.com/pt_BR/download/>.

Utilizamos o ambiente de desenvolvimento integrado Eclipse, na versão Eclipse IDE for Java Developers, que pode ser obtida em <http://www.eclipse.org/downloads/>. O Eclipse foi o ambiente no qual nos foi introduzido o Java e todos nós deste grupo o utilizamos a partir de então, desde o segundo período do curso de Ciência da Computação.

Para analisar os resultados, utilizamos o método **replicativo**. Embora gaste mais tempo de execução do que o modo regenerativo, é mais prático de implementar, bastando ter cuidado com a estimativa da fase transiente e definir sementes distantes.

Os resultados apresentados nesse relatório foram obtidos enquanto executados em um computador Apple MacBook Pro com processador Intel Core i7 quad core, 2,4GHz (Turbo Boost até 3,4 GHz) e Memória de 8GB e 1600 MHz.

1. **Teste de Correção**

Já era sabido antes mesmo de começar a programar que nossos resultados deveriam ser algo próximo da figura abaixo:

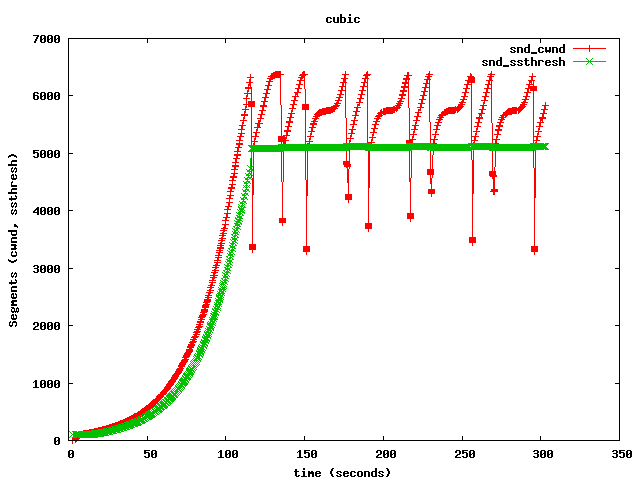


Figura 5. Exemplo de comportamento da janela deslizante do TCP segundo a Linux Foundation

Desenvolvemos nosso trabalho então sabendo que se o gráfico plotado dos resultados fugisse muito disso, algo estaria errado.

1. **Estimativa da fase transiente**

Os dados coletados nos permitem plotar gráficos da janela deslizante (cwnd). Podemos observar a variação da média dessa janela ao longo do tempo pelo gráfico e, quando o gráfico se estabilizar, não variando muito em torno de um determinado valor, podemos assumir que a fase transiente terminou e o sistema está estacionário. Se durante várias rodadas com sementes distintas esse comportamento ocorrer de forma muito parecida, então a fase transiente está bem estimada.

1. **Listagem documentada do programa**

A documentação do programa está diretamente no código fonte. Comentamos todas classes, métodos e variáveis, informando qual a utilidade de cada uma e como ela deve ser usada, de maneira que o código ficasse legível mesmo para quem não está familiarizado com a linguagem Java.

O pacote do projeto que corresponde ao simulador está em ad.simulador. Os demais pacotes são as bibliotecas de apoio citadas na Introdução, que já foram obtidas prontas na Internet.

1. **Resultados e comentários**
2. **Conclusão**

Desenvolvendo esse trabalho, aprendemos bastante sobre o protocolo TCP. Duas pessoas desse grupo (Vinícius e Bruno) já haviam cursado Teleprocessamento e Redes e o terceiro elemento (Jorge) ainda não.

Aprendemos em maiores detalhes do funcionamento da janela deslizante (*cwnd*), que no curso de Redes apenas havíamos feito exemplos no papel mas nunca havíamos implementado um modelo funcional para observar o tratamento dos pacotes e o deslocamento da janela conforme os ACKs chegam.

Entendemos que o TCP é o melhor protocolo da camada de transporte para usar na maioria de nossas tarefas cotidianas, já que é um protocolo que garante a entrega ordenada e confiável dos pacotes, além do controle de fluxo promovido pelos ACKs. Em outros protocolos, como por exemplo o UDP, não existe essa confiabilidade.

Também não conhecíamos a política RED, foi um conhecimento novo que esse trabalho nos agregou. Achamos mais interessante e vantajoso do que a FIFO, já que evita a exaustão do buffer descartando pacotes antecipadamente.

Com o RED, as chegadas de rajadas funcionam melhor com o buffer, isto é, o buffer acaba ficando com espaço melhor gerenciado para armazenar essas rajadas. É uma política mais justa, que acaba penalizando mais as sessões de alto tráfego em benefício das de pouco tráfego, utilizando o tamanho médio das filas e os limites inferior e superior para definir a probabilidade de um pacote ser descartado.

Entretanto, se as filas tiverem taxas de utilização muito fracas, o comportamento da RED e da FIFO ficam bem similares.

Esse comportamento do TCP de estar sempre aumentando a janela de congestionamento lentamente e depois reduzindo à metade bruscamente gera um comportamento parecido com dentes de serra, se visualizado graficamente.

Gostaríamos de ter tido tempo para implementar a WFQ, mas devido a complexidade do trabalho e às outras tarefas de outras disciplinas que tivemos, deixamos esse item com prioridade baixa em nossa lista de tarefas e acabou que não foi possível implementar.

**Referências Bibliográficas:**

1. RODRIGUES, P.H.A. Avaliação e Desempenho – Apostila. 2013.
2. KUROSE; ROSS. Computer Networking: A Top-Down Approach, 6a Edição.
3. Documentação do Java SE 1.6