Trabalho Prático 1:
Interface de sockets e medição de desempenho

Pedro Lopes Miranda Junior

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

plmj@dcc.ufmg.br

Resumo. *O objetivo deste trabalho é (1) exercitar a interface de programação com sockets (em python), (2) se familiarizar com o ambiente virtualizado mininet para emulação de sistemas em rede e (3) medir e analisar o desempenho de aplicações simples neste ambiente. Para tanto, as seguintes etapas estão envolvidas:*

1. *Instalação do mininet e configuração de uma topologia.*
2. *Implementação/execução de programas de teste no mininet e medição de desempenho.*
3. *Análise de resultados e escrita do relatório.*

1. INTRODUÇÃO

Da internet a um sistema de comunicação entre uma estação aqui na terra e um robô em Marte, redes de computadores é algo essencial em tempos onde a iformação é dada em tempo real.
Desse modo, as redes mais simples são compostas geralmente pela comunicação de um cliente com um servidor. O cliente é o responsável pelo envio das mensagens. Por sua vez o servidor recebe as mensagens dos clientes podendo ou não retornar uma resposta.

A comunicação efetiva entre esses pares depende, entre outras coisas da largura da banda de comunicação, da distancia entre esses pares, do tamanho e quantidade das mensagens enviadas e do método utilizado para envio.

A largura de banda ...
A distância entre o cliente e o servidor ...
Já a quantidade e tamanho das mensagens envidas ...
Por último o método ou protocólo de envio influencia diretamente ...

Há contudo modelo mais complexos de redes que serão abordados mais adiante na disciplina, porém para uma iniciação aos conceitos de redes de computadores foi requisitado o teste de dois pares cliente/servidor conforme descritos abaixo:

Par 1: O cliente envia um certo número de mensagens para o servidor, sem paradas entre cada envio. Ao final, ele espera uma mensagem de um byte de volta do servidor.

Par 2: Cada vez que o cliente enviar uma mensagem, ele deverá esperar uma resposta de um byte do ser- vidor. O cliente termina depois de fazer esse processo um certo número de vezes.

2. SOLUÇÃO PROPOSTA

Para solucionar os problemas propostos foram feitos dois pares de cliente/servidor utilizando a linguagem *Python* e a biblioteca *socket* para fazer a conexão do cliente com o servidor. Outras bibliotecas usadas foram as *sys, random* e a *timeit*.
A primeira foi utilizada para recuperar os parâmetros passados na chamda do script. Já a segunda para criar as sequencias aleatórias de mensagens. A última para medir o tempo de execução da função *client* em cada um dos pares.
Os demais detalhes de implementação serão tratados na subseção a seguir

2.1. Par 1

Para o par 1 foi necessário criar um cliente que enviava mensagens intermitentes esperando ao fim de todas elas uma resposta de 1 byte do servidor, e as premissas usadas em cada item do par estão descritas abaixo:

2.1.1. Cliente

O cliente recebe como parâmetro o número de mensagens e o tamanho de cada uma delas, cria as mensagens aleatórias que servirão de envio para o servidor e as envia, uma a uma, esperando ao fim uma resposta de um byte. Para o funcionamento correto do par o cliente envia ao fim das mensagens um sinal de fechamento da conexão apenas para envio, ficando de guarda quanto a resposta final do servidor, para só então fechar completamente a conexão.

A criação das mensagens foi feita separadamente do código do cliente para não interferir na análise de tempo de envio das mensagens do cliente para o servidor. Para essa medição foi utulizada a função *Timer* da biblioteca *timeit* e será evidenciada na análise experimental.

2.1.2. Servidor

Para esse par o servidor recebia ininterruptamente mensagens do cliente pareado com ele e só fechava a conexão ao receber do cliente a instrução de que não receberia mais mensagem alguma. O servidor então enviava uma mensagem de 1 Byte como resposta para o cliente e se encerrava.

2.2. Par 2

Já para o par 2 foi necessário implmentar um cliente que enviava mensagens ao servidor, que respondia a cada uma recebida com uma nova mensagem de 1 Byte. Após enviar todas as mensagen o cliente se encerrava, desconectando-se do servidor.

2.2.1. Cliente

O cliente recebe como parâmetro o número de mensagens e o tamanho de cada uma delas, cria as mensagens aleatórias que servirão de envio para o servidor e as envia,

uma a uma, esperando uma resposta de um byte para cada mensagem enviada. Para o funcionamento correto do par o cliente envia ao fim da última mensagens um sinal de fechamento da conexão, avisando assim ao servidor o fim do envio.

A criação das mensagens foi feita separadamente do código do cliente para não interferir na análise de tempo de envio das mensagens do cliente para o servidor. Para essa medição foi utulizada a função *Timer* da biblioteca *timeit* e será mais detalhada na análise experimental.

2.2.2. Servidor

Nesse caso o servidor iniciava a conexão com seu cliente e a cada nova mensagem enviava uma mensagem de 1 byte como resposta. Após o recebimento de todas as mensagens o mesmo esperava a mensagem de fechamento de conexão vindo do cliente. Note que em momento algum o servidor sabe quantas mensagens o cliente enviará e o mesmo só encerra a conexão quando recebe a mensagem do servidor de conexão fechada.

3. AVALIAÇÃO EXPERIMENTAL

Na avaliação experimental era necessário analisar a variação do tempo de envio das mensagens segundo a alteração da topologia da rede e para isso foi utilizado máquina virtual do mininet. Essa máquina virtual emula elementos de rede como switches e hosts, podendo inclusive alterar elementos como o tempo de comunicação entre eles ou a largura da banda. Ela calcula o tempo de execução de uma função específica chamando-a diversas vezes e tirando a média do tempo de execução da mesma. Nesse primeiro momento não há variação das mensagens, então essa média é a de envio de uma ou mais mensagens específicas. Além disso o procedimento citado é repetido com o envio de outras mensagens, deixando a média ainda mais confiável. Para o teste cada mensagem foi enviada 10 vezes e foram criadas 4 mensagens diferentes para envio ao servidor.

3.1. Resultados

3.2. Análises

4. CONCLUSÃO