



## Redes de Computadores – 2019.2

Prof.: Carlos Bruno

### Trabalho de Prática com Sockets

#### Objetivo

O objetivo do trabalho é praticar a implementação de Sockets TCP e UDP. O trabalho deve ser desenvolvido em Python.

#### Questão 1 – UDP “Pinger”

Nesta questão, você aprenderá as noções básicas de programação de soquete para UDP em Python. Você aprenderá como enviar e receber pacotes datagramas usando soquetes UDP e também como definir um tempo limite (“timeout”) de soquete apropriado.

Ao longo do trabalho, você ganhará familiaridade com a aplicação Ping e sua utilidade no cálculo de estatísticas, como a taxa de perda de pacotes.

Primeiro você estudará um servidor de Ping simples escrito em Python e implementará um cliente correspondente. A funcionalidade fornecida pelo cliente e servidor em Python é semelhante à fornecida pelos programas de Ping disponíveis nos sistemas operacionais modernos. No entanto, eles (cliente e servidor) usam um protocolo mais simples, o UDP, em vez do protocolo ICMP (Internet Control Message Protocol) para se comunicar entre si. O protocolo ping permite que uma máquina cliente envie um pacote de dados para uma máquina remota e faça com que a máquina remota retorne os dados inalterados de volta ao cliente (uma ação chamada de “eco”). Entre outros usos, o protocolo ping permite que os hosts determinem tempos de ida e volta (Round-Trip Time ou RTT) para outras máquinas.

O código do servidor de ping está disponível logo abaixo. Sua tarefa será analisar cuidadosamente as linhas de código e implementar o cliente ping com base no código do servidor.

#### **Código do Servidor**

O código a seguir implementa um servidor de ping. **Não o modifique.** Nesse código, o servidor simula 30% de perda dos pacotes que chegam. Você deve estudar este código cuidadosamente, pois ele irá ajudá-lo a escrever seu cliente ping.

```
# UDPPingerServer.py
# precisaremos do módulo random para gerar perdas de pacotes aleatórias
import random
from socket import *

# Cria um socket UDO
# Note o uso de SOCK_DGRAM para pacotes UDP
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Atribui um endereço IP e um número de porta ao socket
serverSocket.bind(('', 12000))

while True:
    # Gera um número aleatório de 0 a 10
    rand = random.randint(0, 10)
    # Recebe do cliente o pacote junto com seu endereço de destino
    message, address = serverSocket.recvfrom(1024)
    # Escreve a mensagem em letras maiúsculas
    message = message.upper()
    # Se rand < 4, consideramos que o pacote foi perdido
    if rand < 4:
        continue
    # Caso contrário, o servidor responde
    serverSocket.sendto(message, address)
```

## Perda de Pacotes

O UDP fornece às aplicações um serviço de transporte não confiável. As mensagens podem se perder na rede devido a sobrecargas na fila do roteador, a algum hardware defeituoso ou outros motivos. Já que a perda de pacotes pode ser rara ou até mesmo inexistente em redes LANs, o servidor em questão injeta perda artificial para simular seus efeitos.

## Código do Cliente

Implementar o seguinte programa. O cliente deve enviar 10 pings para o servidor. Como o UDP é um protocolo não confiável, um pacote enviado do cliente para o servidor pode ser perdido na rede ou vice-versa. Por esse motivo, o cliente não pode aguardar indefinidamente uma resposta a uma mensagem de ping. Você deve fazer com que o cliente espere até *um segundo* por uma resposta; Se nenhuma resposta for recebida dentro de um segundo, seu programa cliente deve assumir que o pacote foi perdido durante a transmissão pela rede. Você precisará procurar a documentação do Python para descobrir como definir o valor de tempo limite (timeout) em um soquete de datagrama.

Especificamente, seu programa cliente deve:

(1) enviar uma mensagem de ping usando UDP (Nota: Ao contrário do TCP, você não precisa estabelecer uma conexão primeiro, já que o UDP é um protocolo sem conexão.)

- (2) imprimir a mensagem de resposta do servidor, se houver
- (3) calcular e imprimir o tempo de ida e volta (RTT), em segundos, de cada pacote, se o servidor responder
- (4) caso contrário, imprima "Solicitação expirada"

Durante o desenvolvimento, você deve executar o *UDPPingerServer.py* em sua máquina e testar seu cliente enviando pacotes para o "localhost" (ou, 127.0.0.1).

### **Formato da Mensagem**

As mensagens de ping impressas no cliente devem ser formatadas de maneira simples: uma linha, consistindo em caracteres ASCII no seguinte formato:

*Ping número\_de\_sequência hora*

o *número\_de\_sequência* começa em 1 e progride até 10 para cada mensagem de ping enviada pelo cliente, e a hora é a hora em que o cliente envia a mensagem.

### **O que entregar?**

Todo o código fonte do cliente mais as imagens do programa em funcionamento. Além de ser apresentado ao professor.

### **Questão 2 – Web Server**

Nesta questão, você aprenderá os fundamentos da programação de soquetes para conexões TCP em Python: como criar um soquete, vinculá-lo a um endereço e porta específicos, bem como enviar e receber um pacote HTTP. Você também aprenderá algumas noções básicas de formato do cabeçalho HTTP.

- a) Você desenvolverá um servidor Web que manipula apenas uma solicitação HTTP por vez. Seu servidor Web deve aceitar solicitações HTTP, obter páginas solicitadas pelo cliente, e também deve criar mensagens de resposta HTTP, constituída do arquivo solicitado precedido pelas linhas de cabeçalho e, então, enviar a resposta ao cliente. Se o arquivo solicitado não estiver presente no servidor, o servidor deverá enviar uma mensagem HTTP "404 Not Found" de volta ao cliente.
- b) Implemente um servidor multithread capaz de atender a várias solicitações simultaneamente. Usando *threading*, primeiro crie um *thread* principal no qual o servidor atende clientes em uma porta fixa. Quando ele recebe uma solicitação de conexão TCP de um cliente, ele configura a conexão TCP através de outra porta e atender à solicitação do cliente em um *thread* separado.

Haverá uma conexão TCP separada em um *thread* separado para cada par de solicitação/resposta.

- c) Em vez de usar um navegador, escreva seu próprio cliente HTTP para testar seu servidor. Seu cliente se conectará ao servidor usando uma conexão TCP, enviará uma solicitação HTTP ao servidor e exibirá a resposta do servidor como uma saída. Você pode assumir que a solicitação HTTP enviada é um método GET. O cliente deve usar argumentos de linha de comando, especificando o endereço IP ou o nome do host do servidor, a porta na qual o servidor está atendendo e o caminho no qual o objeto solicitado é armazenado no servidor. A seguir está um formato de comando de entrada para executar o cliente.

```
client.py server_host server_port nome_arquivo
```

## Código

Abaixo, você encontrará o código “esqueleto” do servidor Web. Você deve completá-lo. Os locais em que você precisa inserir código são marcados com `#codigo_inicio` e `#codigo_fim`. Cada local pode exigir uma ou mais linhas.

## O que entregar?

Você entregará o código do servidor completo junto com as capturas de tela do navegador do cliente, verificando se você realmente recebe o conteúdo do arquivo HTML do servidor.

## Esqueleto Python para o Web Server

```
#import socket module
from socket import *
import sys # para terminar o programa

serverSocket = socket(AF_INET, SOCK_STREAM)

#Prepara o socket servidor
#codigo_inicio
#codigo_fim

while True:
    #Estabelece a conexão
    print('Ready to serve...')
    connectionSocket, addr = #codigo_inicio #codigo_fim
    try:
        message = #codigo_inicio #codigo_fim
        filename = message.split()[1]
```

```

f = open(filename[1:])
outputdata = #codigo_inicio      #codigo_fim
#Envia um linha de cabeçalho HTTP para o socket
#codigo_inicio
#codigo_fim
#Envia o conteúdo do arquivo solicitado ao cliente
for i in range(0, len(outputdata)):
connectionSocket.send(outputdata[i].encode())
connectionSocket.send("\r\n".encode())
connectionSocket.close()
except IOError:
    #Envia uma mensagem de resposta "File not Found"
    #codigo_inicio
    #codigo_fim
    #Fecha o socket cliente
    #codigo_inicio
    #codigo_fim
serverSocket.close()
sys.exit()#Termina o programa depois de enviar os dados

```

## Pontuação

30% da AP2

## Instruções para a entrega do trabalho

- i) Criar pasta cujo nome é composto pela matrícula da dupla (e.g., 222333\_444555);
- ii) Dentro da pasta criada em i), criar uma pasta para cada questão (Q1 e Q2)
- iii) Adicionar o código desenvolvido (arquivos .py) e as capturas de tela na pasta correspondente à questão;
- iv) Compactar a pasta criada em i) usando o programa ZIP
- v) Enviar o arquivo .ZIP pelo SIPPA (Trabalho T1)

**Data da entrega: até 12 de Novembro de 2019**