Experimento: Lab 2 - UDP Pinger

Neste experimento, você aprenderá os fundamentos da programação de *socket* para UDP em Python. Você aprenderá como enviar e receber pacotes *datagram* usando *sockets UDP* e também como definir o tempo limite adequado de um *socket*. Ao longo do experimento, você se familiarizará com uma aplicação de Ping e sua utilidade no cálculo de estatísticas, como a taxa de perda de pacotes.

Inicialmente você estudará um simples servidor de ping na Internet, escrito em Python, e implementará um cliente correspondente. A funcionalidade fornecida por esses programas é semelhante aos programas de ping padrão disponíveis em sistemas operacionais modernos. No entanto, esses programas usam o protocolo UDP, que é mais simples, em vez do protocolo ICMP (Internet Control Message Protocol) padrão, para se comunicarem. O protocolo de ping permite que uma máquina cliente envie um pacote de dados para uma máquina remota e que a máquina remota devolva o dado para o cliente sem modificações (ação conhecida como `echo`). Entre outros usos, o protocolo de ping permite que os hosts determinem o tempo de ida e volta (`round-trip time`) para outras máquinas.

Você receberá o código completo do servidor de ping abaixo. Sua tarefa será escrever o cliente de ping.

Código do Servidor

O código a seguir implementa completamente um servidor de ping. Você precisa compilar e executar este código antes de rodar seu programa cliente. Não é necessário modificar este código.

Neste código do servidor, 30% dos pacotes do cliente são simulados para serem perdidos. Você deve estudar este código cuidadosamente, pois ele o ajudará a escrever o cliente de ping.

```
# UDPPingerServer.py

import random
from socket import *

# Criar um socket UDP
serverSocket = socket(AF_INET, SOCK_DGRAM)

# Atribuir endereço IP e número da porta ao socket
serverSocket.bind(('', 12000))

while True:
    # Gerar número aleatório entre 0 e 10
    rand = random.randint(0, 10)

    # Receber o pacote do cliente junto com o endereço de origem
    message, address = serverSocket.recvfrom(1024)

    # Colocar a mensagem recebida em maiúsculas
    message = message.upper()

    # Se o número rand for menor que 4, consideramos que o pacote foi perdido e não
respondemos
    if rand < 4:
        continue
```

```
    # Caso contrário, o servidor responde
    serverSocket.sendto(message, address)
```

O servidor fica em um loop infinito, escutando por pacotes UDP recebidos. Quando um pacote é recebido, e se um número aleatório for maior ou igual a 4, o servidor simplesmente capitaliza (converte os dados recebidos para letras maiúsculas) os dados encapsulados e os envia de volta ao cliente.

Perda de Pacotes

O UDP fornece às aplicações um serviço de transporte não confiável. As mensagens podem ser perdidas na rede devido a estouros de fila em roteadores, falhas de hardware ou outras razões. Como a perda de pacotes é rara ou até inexistente em redes típicas de campus, o servidor neste experimento injeta perda artificial para simular os efeitos de perda de pacotes na rede.

Código do Cliente

Você precisa implementar o seguinte programa cliente:

- 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 essa razão, o cliente não pode esperar indefinidamente por uma resposta de um ping.
- O cliente deve aguardar 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.

Especificamente, o programa cliente deve:

1. Enviar a mensagem de ping usando UDP (Nota: Diferente do TCP, você não precisa estabelecer uma conexão, 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, para cada pacote, se o servidor responder.
4. Caso contrário, imprimir `Request timed out` (*Requisição expirou*).

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). Depois de depurar completamente o seu código, veja como sua aplicação se comunica na rede, com o servidor e cliente de ping rodando em máquinas diferentes.

Formato da Mensagem

As mensagens de ping neste experimento são formatadas de maneira simples. A mensagem do cliente é uma linha, consistindo de caracteres ASCII no seguinte formato:

```
Ping sequence_number time
```

Onde `sequence_number` começa em 1 e avança até 10 para cada mensagem de ping enviada pelo cliente, e `time` é o momento em que o cliente envia a mensagem.

O que apresentar

Você deve apresentar o código completo do cliente e mostrar que o programa de ping funciona conforme requerido.

Esta atividade vale nota. A implementação básica resulta em nota máxima 6,0.

Exercícios Opcionais (Nota Extra)

1. Atualmente, o programa calcula o tempo de ida e volta para cada pacote e imprime individualmente. Modifique isso para corresponder à forma como o programa padrão de ping funciona. Você precisará reportar os RTTs mínimo, máximo e médio ao final de todos os pings do cliente. Além disso, calcule a taxa de perda de pacotes (em porcentagem).
2. Outra aplicação semelhante ao UDP Ping seria o UDP Heartbeat. O Heartbeat pode ser usado para verificar se uma aplicação está em execução e relatar a perda de pacotes unidirecional. O cliente envia um número de sequência e a hora atual no pacote UDP para o servidor, que escuta pelo Heartbeat (ou seja, pelos pacotes UDP) do cliente. Ao receber os pacotes, o servidor calcula a diferença de tempo e relata quaisquer pacotes perdidos. Se os pacotes de Heartbeat estiverem ausentes por um determinado período, podemos assumir que a aplicação cliente parou. Implemente o UDP Heartbeat (tanto cliente quanto servidor). Você precisará modificar o `UDPPingerServer.py` e seu cliente de ping UDP.