UNIVERSIDADE ESTADUAL DE LONDRINA

GABRIEL ÂNGELO P. G. SABAUDO
GUILHERME HENRIQUE GONÇALVES SILVA

Ferramenta para teste de rede utilizando socket

LONDRINA - PR
2022

Menu inicial - protocolo TCP

Executando o código do protocolo TCP ele aparecerá um menu para o usuário decidir entre receber dados de transmissão (opção 1), enviar dados de transmissão (opção 2) e encerrar a execução do código (opção 3).

```
def menu():
    print("Escolha uma opção\n")
    print("1 - Receber dados de transmissão")
    print("2 - Enviar dados de transmissão")
    print("3 - Sair da aplicação")


def main():
    while True:
        menu()
        op = int(input())

        if op == 1:
            receber()
        elif op == 2:
            enviar()
        elif op == 3:
            print("Saindo do app. . .")
            return
```

Escolhendo a opção 1 de receber dados de transmissão ele irá começar a executar a função de receber dados de transmissão.

Receber dados

A execução da função começa com a declarações de variais, como a declaração do PACKAGE_SIZE, IP e Port.

```
PACKAGE_SIZE = 4096
IP = input('IP: ')
PORT = int(input('Porta: '))
```

Logo em sequência realiza a criação do soquete é realizada e faz a conexão no servidor com o endereço passado pelo usuário.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Cria o socket
sock.connect((IP ,PORT)) # Realiza a conexão com o host
```

Inicia o tempo de contagem da transmissão e declara e inicializa duas variáveis, package_count = 0 e print_count = 1.

```
starttime = datetime.datetime.now() # Inicia o tempo de contagem da transmissão
package_count = 0 # Inicializa package_count igual a zero
print_count = 1 # Inicializa print_count igual a um
```

Após realizado o código acima, começa a ser feito o teste de download do cliente. Entrando em um while true, enquanto houver pacotes para enviar, incrementa a contagem package_count e os recebe (a função recv é usada para ler dados de entrada em soquetes) e finaliza o tempo de contagem da transmissão e realiza a variação de tempo entre as variáveis endtime e starttime e salva na variável delta.

```
while True:
    data = sock.recv(PACKAGE_SIZE)
    package_count = package_count + 1 # Incrementa a contagem +1

    endtime = datetime.datetime.now() # Finaliza o tempo de contagem da transmissão
    delta = endtime - starttime # Variação de tempo entre as variáveis endtime e starttime
```

Ainda dentro do while true, haverá uma verificação se existe conteúdo na variável data caso ela tenha conteúdo esse será deletado, com isso, entrará em um if que faz apenas uma animação de pontos sendo carregados para que o usuário tenha conhecimento que o teste está sendo processado e após sair do if ele encontrará um continue e voltar para o início do while true e realizará esse loop por 20 segundos. Após passar os 20 segundos haverá um função sock.close que encerra o soquete associado.

```
while True:
    data = sock.recv(PACKAGE_SIZE)
    package_count = package_count + 1 # Incrementa a contagem +1

    endtime = datetime.datetime.now() # Finaliza o tempo de contagem da transmissão
    delta = endtime - starttime # Variação de tempo entre as variáveis endtime e starttime

    if data:
        del data

        if(delta.seconds >= print_count):
            print_count = print_count + 1
            print('.', end='', flush=True)

        continue

    print('\n')
    sock.close()
```

E por fim, haverá a realização dos cálculos dos pacotes recebidos, bytes transferidos, velocidade média do download e o tempo de execução, por fim, mostrará os resultados obtidos e encontrará um break para terminar o laço de repetição do while true.

```
endtime = datetime.datetime.now()
delta = endtime - starttime
bytes_transferred = round(package_count * PACKAGE_SIZE / 1024 / 1024, 2)
speed = round(bytes_transferred * 8 / delta.seconds, 2)

print('## RESULTADO ##')
print(f'Pacotes Recebidos: {package_count}')
print(f'Bytes Transferidos: {bytes_transferred} MB')
print(f'Velocidade Média: {speed} Mbps')
print(f'Tempo: {delta.seconds} segundos\n')
break
```

Depois de ter finalizado o download começa a parte do upload. O começo do upload e bem parecido com o do download. A criação do soquete é realizada e faz a conexão no servidor com o endereço passado pelo usuário.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Cria o socket
sock.connect((IP ,PORT)) # Realiza a conexão com o host
```

Inicia o tempo de contagem da transmissão e declara e inicializa três variáveis, package_count = 0, print_count = 1 e package = b'x' * PACKAGE_SIZE * 4.

```
starttime = datetime.datetime.now() # Inicia o tempo de contagem da transmissão

package_count = 0 # Inicializa package_count igual a zero
print_count = 1 # Inicializa print_count igual a um
package = b'x' * PACKAGE_SIZE * 4
```

Após realizado o código acima começa a ser feito o teste de upload do cliente. Entrando em um while true, enquanto há pacotes para enviar, incrementa a contagem e os envia (a função send deve enviar uma mensagem somente quando o socket estiver conectado) e finaliza o tempo de contagem da transmissão e realiza a variação de tempo entre as variáveis endtime e starttime e salva na variável delta.

```
while True:
    sock.send(package)
    package_count = package_count + 1

    endtime = datetime.datetime.now()  # Finaliza o tempo de contagem da transmissão
    delta = endtime - starttime  # Variação de tempo entre as variáveis endtime e starttime
```

Com isso, entrará em um if que faz apenas uma animação de pontos sendo carregados para que o usuário tenha conhecimento que o teste está sendo processado e após sair do if ele entrará em um segundo if verificando se passou os 20 segundos de execução, caso ainda não tenha passado os 20 segundos ele voltará para o início do while true, mas caso, tenha chegado nos 20 segundos entrará dentro desse segundo if e dará um break para sair do while true, haverá um função sock.close que encerra o soquete associado.

```
while True:
    sock.send(package)
    package_count = package_count + 1

    endtime = datetime.datetime.now()  # Finaliza o tempo de contagem da transmissão
    delta = endtime - starttime  # Variação de tempo entre as variáveis endtime e starttime

    if(delta.seconds >= print_count):
        print_count = print_count + 1
        print('.', end='', flush=True)

    if(delta.seconds >= 20):
        break

print('\n')
sock.close()
```

E por fim, haverá a realização dos cálculos dos pacotes recebidos, bytes transferidos, velocidade média do download e o tempo de execução, por fim, mostrará os resultados obtidos. E com isso, finaliza a função receber.

```
endtime = datetime.datetime.now()
delta = endtime - starttime
bytes_transferred = round(package_count * PACKAGE_SIZE / 1024 / 1024, 2)
speed = round(bytes_transferred * 8 / delta.seconds, 2)

print('## RESULTADO ##')
print(f'Pacotes Enviados: {package_count}')
print(f'Bytes Transferidos: {bytes_transferred} MB')
print(f'Velocidade Média: {speed} Mbps')
print(f'Tempo: {delta.seconds} segundos\n')
```

Escolhendo a opção 2 de enviar dados de transmissão ele irá começar a executar a função de enviar dados de transmissão.

Enviar Dados

A execução da função começa com a declarações de variais, como a declaração do PACKAGE_SIZE, IP, Port e package.

```
PACKAGE_SIZE = 4096
IP = input('IP: ')
PORT = int(input('Porta: '))

package = b'x' * PACKAGE_SIZE * 4
```

Com isso, começa a ser realizado o teste de download do cliente. Logo em sequência realizará a criação do soquete e linka o server de acordo com o endereço e porta oferecidos (a função bind liga um nome local exclusivo ao socket com o descritor socket.). E com isso, o socket começa a ouvir o endereço fornecido (a função listen deixa um socket pronto para aceitar conexões).

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind((IP,PORT))
sock.listen(0)
```

Prosseguindo, a função sock.accept aceita uma conexão. O soquete deve estar vinculado a um endereço e escutar conexões. O valor de retorno é um par `(socket, address)` onde socket é um novo objeto de soquete utilizável para enviar e receber dados na conexão, e address é o endereço vinculado ao soquete na outra extremidade da conexão. E Inicia o tempo de contagem da transmissão.

```
client_sock, client_addr = sock.accept()

starttime = datetime.datetime.now()
```

Entrando em um while true, enquanto há pacotes para enviar, incrementa a contagem e os envia (a função send deve enviar uma mensagem somente quando o socket estiver conectado) e finaliza o tempo de contagem da transmissão e realiza a variação de tempo entre as variáveis endtime e starttime e salva na variável delta.

```
while True:
    client_sock.send(package)

    endtime = datetime.datetime.now()
    delta = endtime - starttime
```

Logo em seguida, ainda dentro do while true haverá um if verificando se passou os 20 segundos de execução, caso ainda não tenha passado os 20 segundos ele voltará para o início do while true, mas caso já tenha chegado nos 20 segundos entrará dentro de um if e haverá um função client_sock.close que encerará o soquete associado e encontrará um break para sair do while true e finaliza o tempo de contagem da transmissão.

```
while True:
    client_sock.send(package)

    endtime = datetime.datetime.now()
    delta = endtime - starttime

    if(delta.seconds >= 20):
        client_sock.close()
        break

endtime = datetime.datetime.now()
```

Depois de finalizado o download começa a parte do upload. O começo do upload e bem parecido com o do download. O socket começa a ouvir o endereço fornecido (a função listen deixa um socket pronto para aceitar conexões). Prosseguindo, a função sock.accept aceita uma conexão. O soquete deve estar vinculado a um endereço e escutar conexões. O valor de retorno é um par `(socket, address)` onde socket é um novo objeto de soquete utilizável para enviar e receber dados na conexão, e address é o endereço vinculado ao soquete na outra extremidade da conexão. E Inicia o tempo de contagem da transmissão.

```
sock.listen(0)

client_sock, client_addr = sock.accept()

starttime = datetime.datetime.now()
```

Entrando em um while true, enquanto há pacotes para enviar, incrementa a contagem e os recebe (a função recv é usada para ler dados de entrada em soquetes). Haverá uma verificação se existe conteúdo na variável data, caso ela tenha conteúdo esse será deletado, com isso, encontrará um continue e voltar para o início do while true e realizará esse loop até que a variável data esteja vazia. A partir do momento que ela esteja vazia haverá um função client_sock.close que encerará o soquete associado.

```
while True:
    data = client_sock.recv(PACKAGE_SIZE)
    if data:
        del data
        continue

    client_sock.close()
```

Logo em seguida, ainda dentro do while true será finaliza o tempo de contagem da transmissão e encontrará um break para sair do laço do while true, por fim, haverá um função sock.close que encerará o soquete associado. Com isso, finalizando a função enviar.

```
while True:
    data = client_sock.recv(PACKAGE_SIZE)
    if data:
        del data
        continue

    client_sock.close()

    endtime = datetime.datetime.now()
    print(endtime, end=' ')
    print(f'{client_addr[0]}:{client_addr[1]} desconectado\n')
    break

sock.close()
```

Menu inicial protocolo - UDP

O programa do protocolo UDP começa primeiramente pelo menu de opções, onde o usuário da ferramenta pode escolher entre enviar os pacotes, ou receber os pacotes, a fim de testar a conexão entre os 2 hosts.

```
def menu():
    print("Escolha uma opção\n")
    print("1 - Receber dados de transmissão")
    print("2 - Enviar dados de transmissão")
    print("3 - Sair da aplicação")


def main():
    while True:
        menu()
        op = int(input())

        if op == 1:
            receber()
        elif op == 2:
            enviar()
        elif op == 3:
            print("Saindo do app. . .")
            return


main()
```

Enviar pacotes

O programa começa primeiramente perguntando as informações necessárias para o funcionamento da ferramenta como ip, porta e tamanho do pacote. Depois é feita a inicialização das variáveis de header e tamanho do payload.

```
HOST = input('IP: ')
PORT = int(input('PORTA: '))
PACKAGE_SIZE = int(input('TAMANHO DO PACOTE: '))

HEADER_SIZE = 8
PAYLOAD_SIZE = PACKAGE_SIZE - HEADER_SIZE
```

Em seguida é criado o socket de protocolo UDP e então é feito o link entre host e porta usando socket.bind(). Depois, é feita uma verificação para saber se o outro host conseguiu se conectar ou não.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((HOST,PORT))

print(f'\nVinculado em {HOST}:{PORT}\n')

data, client_addr = sock.recvfrom(PACKAGE_SIZE)
if data == b'CONNECT!':
    sock.sendto(b'CONNECTED!', client_addr)
else:
    print('Server nao se conectou com o cliente!')
    exit(1)
```

Teste de download

É feito um timeout de 3 segundos para garantir que os dados possam chegar até o outro host, para assim continuar o programa. Algumas variáveis são setadas com um valor inicial, e o tempo de transmissão é iniciado.

```
sock.settimeout(3)

starttime = datetime.datetime.now()
package_id = 1
count_timeout = 0
payload = b'x' * PAYLOAD_SIZE
```

O header e payload são criados, e o programa então começa a enviar os dados. O tempo máximo de envio de 20 é definido, e a cada vez que a variável chega nos 20 segundos, é feita uma verificação de se caso não tiver tido a confirmação em um determinado tempo, o pacote é reenviado em até no máximo 5 vezes.

```
while True:
    header = bytes('{:0>8}'.format(format(package_id, 'X')), 'utf-8')      #Faz um header com caracteres com um identificador único para cada pacote, em hexadecimal
    package_id += 1
    package = b''.join([header, payload])

    sock.sendto(package, client_addr)

    endtime = datetime.datetime.now()
    delta = endtime - starttime

    if(delta.seconds >= 20):
        header = bytes('{:0>8}'.format(0), 'utf-8')     #Faz um header com HEADER_SIZE 0's, em hexadecimal
        package = b''.join([header, b'END!'])

        while True: #Caso o não receba a confirmação em um determinado tempo, envia o pacote de novo, até um máximo de 5 vezes
            try:
                sock.sendto(package, client_addr)
                data, client_addr = sock.recvfrom(PACKAGE_SIZE)

                if data == b'OK!':
                    break
            except socket.timeout:
                if count_timeout == 5:
                    break
                else:
                    count_timeout = count_timeout + 1
                    continue
        break
```

Teste de upload

No teste de upload, o timeout é declarado para 0, e então é feita a contagem dos pacotes. Depois que eles são confirmados, é feito o envio dos pacotes e o teste é finalizado. Caso não haver nenhuma confirmação, os pacotes são enviados novamente em até no máximo 5 vezes, e se não der certo a conexão com o outro host é perdida.

```
sock.settimeout(None)

package_counter = 0

while True:
    package, server_socket = sock.recvfrom(PACKAGE_SIZE)
    package_counter = package_counter + 1

    package_id = int(str(package)[2:HEADER_SIZE+2], 16)

    if package_id != 0:
        continue

    sock.sendto(b'OK!', client_addr)

    break
sock.settimeout(3)
```

```
while True: #Caso o não receba a confirmação em um determinado tem
    try:
        sock.sendto(str(package_counter).encode(), client_addr)
        data, client_addr = sock.recvfrom(PACKAGE_SIZE)

        if data == b'OK!':
            break
    except socket.timeout:
        continue
    except ConnectionResetError:
        break

sock.close()
```

Receber pacotes

O programa começa primeiramente perguntando as informações necessárias para o funcionamento da ferramenta como ip, porta e tamanho do pacote. O endereço é definido pela combinação de host e porta. Depois é feita a inicialização das variáveis de header e tamanho do payload.

```
HOST = input('IP: ')
PORT = int(input('PORTA: '))
PACKAGE_SIZE = int(input('TAMANHO DO PACOTE: '))

ADDR = (HOST, PORT)

HEADER_SIZE = 8
PAYLOAD_SIZE = PACKAGE_SIZE - HEADER_SIZE
print(f'\nTamanho do HEADER: {HEADER_SIZE}')
print(f'Tamanho do PAYLOAD: {PAYLOAD_SIZE}\n')
```

Em seguida é criado o socket de protocolo UDP e então, é declarado um timeout de 3 segundos e então é feita uma verificação para saber se o outro host conseguiu se conectar ao ip digitado ou não.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.settimeout(3)

while True: #Caso o não receba a confirmação em um determinado tempo, envia o pacote de novo
    try:
        sock.sendto(b'CONNECT!', ADDR)
        data, server_addr = sock.recvfrom(PACKAGE_SIZE)

        if data:
            break
    except socket.timeout:
        continue

sock.settimeout(None)
```

Algumas variáveis são inicializadas com um valor inicial, e o tempo de transmissão é iniciado.

```
starttime = datetime.datetime.now()
package_counter = 0
print_count = 1
```

Teste de download

O programa começa a receber os pacotes vindos do outro host e faz sua contagem. Depois que os pacotes são enviados, é calculado então os valores de bytes transferidos, pacotes perdidos, pacotes recebidos, a velocidade média de transferência e a duração da transferência.

```
print('### Testando Download ###')
while True:
    endtime = datetime.datetime.now()
    delta = endtime - starttime

    package, server_socket = sock.recvfrom(PACKAGE_SIZE)

    package_counter = package_counter + 1

    package_id = int(str(package)[2:HEADER_SIZE+2], 16)

    if package_id != 0:
        if(delta.seconds >= print_count):
            print_count = print_count + 1
            print('.', end='', flush=True)

        continue
```

```
sock.sendto(b'OK!', ADDR)
data, server_addr = sock.recvfrom(PACKAGE_SIZE)
sock.sendto(b'OK!', ADDR)
packages_sended = int(data.decode())

packages_lost = packages_sended - package_counter
bytes_transferred = package_counter * PACKAGE_SIZE / 1024 / 1024
delta = endtime - starttime
speed = round(bytes_transferred * 8 / delta.seconds, 2)

print('\n\n## RESULTADO ##')
print(f'Pacotes Recebidos: {package_counter}')
print(f'Pacotes Perdidos: {packages_lost}')
print(f'Bytes Transferidos: {round(bytes_transferred, 2)} MB')
print(f'Velocidade Média: {speed} Mbps')
print(f'Tempo: {delta.seconds} segundos\n')
break

UPLOAD
```

Teste de upload

É feito um timeout de 3 segundos para garantir que os dados possam chegar até o outro host, para assim continuar o programa. Algumas variáveis são setadas com um valor inicial, e o tempo de transmissão é iniciado.

```
## UPLOAD

sock.settimeout(3)

starttime = datetime.datetime.now()
package_id = 1
count_timeout = 0
print_count = 1
payload = b'x' * PAYLOAD_SIZE
```

O header e payload são criados, a quantidade de pacotes começa a ser contada e o programa então começa a enviar os dados. O tempo máximo de envio de 20 é definido, e a cada vez que o tempo chega nos 20 segundos, é feita uma verificação de se caso não tiver tido a confirmação em um determinado tempo, o pacote é reenviado em até no máximo 5 vezes.

```
print('\n### Testando Upload ###')
while True:
    header = bytes('{:0>8}'.format(format(package_id, 'X')), 'utf-8')
    package_id += 1
    package = b''.join([header, payload])

    sock.sendto(package, ADDR)

    endtime = datetime.datetime.now()
    delta = endtime - starttime

    if(delta.seconds >= print_count):
        print_count = print_count + 1
        print('.', end='', flush=True)

    if(delta.seconds >= 20):
        header = bytes('{:0>8}'.format(0), 'utf-8')     #Faz um header c
        package = b''.join([header, b'END!'])
```

```
while True: #Caso o não receba a confirmação em um determi
    try:
        sock.sendto(package, ADDR)
        data, server_socket = sock.recvfrom(PACKAGE_SIZE)

        if data == b'OK!':
            break
    except socket.timeout:
        if count_timeout == 5:
            break
        else:
            count_timeout = count_timeout + 1
            continue
break
timeout(None)
```

Depois que o teste é finalizado o socket fecha, e é feito o cálculo do número de pacotes que foram perdidos, o número de pacotes enviados, o número de bytes transferidos, a velocidade média da transmissão é o tempo de transmissão.

```
sock.settimeout(None)
data, server_addr = sock.recvfrom(PACKAGE_SIZE)
sock.sendto(b'OK!', ADDR)
packages_received = int(data.decode())

sock.close()

packages_lost = package_id - packages_received
bytes_transferred = packages_received * PACKAGE_SIZE / 1024 / 1024
delta = endtime - starttime
speed = round(bytes_transferred * 8 / delta.seconds, 2)

print('\n\n## RESULTADO ##')
print(f'Pacotes Enviados: {package_id}')
print(f'Pacotes Perdidos: {packages_lost}')
print(f'Bytes Transferidos: {round(bytes_transferred, 2)} MB')
print(f'Velocidade Média: {speed} Mbps')
print(f'Tempo: {delta.seconds} segundos\n')
```