Universidade Estadual de Londrina

JOÃO VITOR FERREIRA
WALLACY SEBASTIAN APARECIDO JERONIMO DE ALMEIDA

TESTE DE VELOCIDADE UTILIZANDO SOCKET

LONDRINA - PR
2022

Documentação do projeto

Neste projeto utilizamos a linguagem de programação python para desenvolver um script que fizesse o envio de pacotes do ponto A ao ponto B da rede, a fim de medir a velocidade de download e upload entre os dois pontos.

Foi utilizado também a biblioteca de python socket para a implementação da conexão entre os dois ponto. Os protocolos de rede utilizado para realizar estas conexões foram: TCP e UDP.

A seguir está o script em python para conexão entre os pontos utilizando o protocolo TCP:

PONTO A (UPLOAD)

```
# -*- coding: utf-8 -*-
import socket
import os
import time
from math import floor
import sys

HOST = "127.0.0.1"
PORT = 8000

ponto_env = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

ponto_env.connect((HOST, PORT))

print("CONECTADO")

tam = 512 #tamanho do pacote
print("Enviando tamanho do pacote: ", tam)

file = open('enviar/teste.01.10MB.rar', 'rb')
tam_arq = os.path.getsize('enviar/teste.01.10MB.rar')
n_pac = floor(tam_arq/tam)
# print("Enviando numero de pacotes")
# ponto_env.send(n_pac.to_bytes(10, sys.byteorder))

inicio = time.time()
numero_pacotes = 1

while True:
progresso = numero_pacotes*tam
packet = file.read(tam)
if not packet:
file.seek(0, 0)
packet = file.read(tam)
sent = ponto_env.send(bytes(packet))
if sent == 0:
raise RuntimeError("socket connection broken")
```

```
fim = time.time()
if(fim - inicio) >= 20:
break
print(f"{progresso}/{int(tam*n_pac)}b")
numero_pacotes += 1
file.close()
ponto_env.close()
print(f"Número de pacotes: {numero_pacotes}")
print(f"Upload\nPacotes/s: {numero_pacotes/(fim-inicio)}\nBits/s:
{(numero_pacotes*512*8)/(fim-inicio)}")
print(f"Total de bytes: {numero_pacotes*512}\nTempo: {fim-inicio}")
```

PONTO B (DOWNLOAD)

```
from os import times
import socket
import time
import sys

HOST = "127.0.0.1"
PORT = 8000
arq = open('receber/download.rar','wb')
ponto_rec = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ponto_rec.bind((HOST,PORT))
ponto_rec.listen(1)
# ponto_rec.settimeout(10.0)
print("Esperando conexão ...")

conn,addr= ponto_rec.accept()
# print("Conectado\nEsperando tamanho do pacote: ")
# tam = int.from_bytes(conn.recv(512), sys.byteorder)
# print(tam)

# print("Esperando número de pacotes: ")
# n_pac = int.from_bytes(conn.recv(512), sys.byteorder)
# print(n_pac)

start = time.time()
numero_pacotes = 1


while True:
dados=conn.recv(512)
end = time.time()
if(end-start) >= 23:
break
if not dados:
print("Acabou os dados")
```

```
    break
    arq.write(dados)
    print(numero_pacotes)
    numero_pacotes += 1
    arq.close()
    ponto_rec.close()
    print(f"Download\nPacotes/s: {numero_pacotes/(end-start)}\nBits/s:
    {(numero_pacotes*512*8)/(end-start)}")
    print(f"Total de bytes: {numero_pacotes*512}\nTempo: {end-start}")
```

As saídas apresentadas para o script do ponto A:

Upload
Pacotes/s: 170796.3035503427
Bits/s: 699581659.3422037
Total de bytes: 1831439360
Tempo: 20.943251848220825

As saídas apresentadas para o script do ponto B:

Download
Pacotes/s: 170796.3862967368
Bits/s: 699581998.271434
Total de bytes: 1831439872
Tempo: 20.9432475566864

Abaixo está o script em python para conexão entre os pontos utilizando o protocolo UDP:

Ponto A (Download)

```
import socket
import time
import threading
from bufferPacotes import BufferPacotes
from pacote import Pacote

parar = False
tamanhoPacote = 512
quantidadeMaximaPacotes = 256
tamanhoBuffer = 30720
tempoEsperaMaximo = 5
buffer = BufferPacotes(tamanhoPacote, quantidadeMaximaPacotes, tamanhoBuffer,
tempoEsperaMaximo)
confirmados = []
mutexConfirmados = threading.Lock()
delay = 0.004
HOST = 'localhost'
PORT = 5000
ADDRESS = (HOST, PORT)
dados = bytes()

for i in range(0, 500):
dados = bytes().join([dados, b'a'])

def __modificarConfirmados(inserir = False):
numero = False

mutexConfirmados.acquire()
try:
if not inserir:
numero = confirmados.pop(0)
else:
confirmados.append(inserir)
except:
numero = False
finally:
mutexConfirmados.release()

return numero

def inserirDados():
global parar
while not parar:
buffer.inserirDados(dados)
```

```
print("Terminou t1")

def criarPacotes():
global parar
while not parar:
buffer.criarPacotes()

print("Terminou t2")

def enviarPacotes(sock):
global parar
while not parar:
pacote = buffer.obterPacote()
if pacote:
sock.sendall(pacote)
time.sleep(delay)

print("Terminou t3")

def confirmarTransmissao():
global parar
inicio = time.time()
agora = time.time()
while (agora - inicio) < 20:
numero = __modificarConfirmados()
if numero:
buffer.confirmarTransmissao(numero)

time.sleep(delay)

agora = time.time()

parar = True
print('')
buffer.encerrar()
print("Terminou t4")

# Inicializando client
print("Configurando cliente UDP")
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.connect(ADDRESS)
sock.settimeout(10.0)
sock.sendall(b'1')

# Enviando pacotes
print("Enviando pacotes ao servidor")

buffer.iniciar()
```

```
t1 = threading.Thread(target=inserirDados, args=())
t1.start()

t2 = threading.Thread(target=criarPacotes, args=())
t2.start()

t3 = threading.Thread(target=enviarPacotes, args=(sock,))
t3.start()

t4 = threading.Thread(target=confirmarTransmissao, args=())
t4.start()

sock.settimeout(2.0)

while not parar:
try:
numero = int.from_bytes(sock.recv(8), byteorder='little')
__modificarConfirmados(numero)
except:
continue


sock.sendall(Pacote(999999, tamanhoPacote).montar())

print('\nEncerrando teste e obtendo estatísticas...\n')
t1.join()
t2.join()
t3.join()
t4.join()
# Limpando buffers e sockets
sock.close()

print("Upload")
print("Taxa de pacotes/s: %.2f" % buffer.obterTaxaPacotesSegundo())
print("Taxa de bits/s: %.2f" % buffer.obterTaxaBitsSegundo())
print("Tempo total executado: %.2f segundos" % buffer.obterTempoTotal())
print("Total de bits transmitidos: %.2f bits" % buffer.obterTotalBits())
```

Ponto B (Upload)

```
import socket
import time
from pacote import Pacote

# Server Setup
print("Configurando servidor UDP")
HOST = 'localhost'
PORT = 5000
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((HOST, PORT))
sock.settimeout(10.0)
tamanhoPacote = 512

print("Aguardando conexão...")
data, address = sock.recvfrom(4)
print(address)

# Recebendo pacotes
sock.settimeout(5.0)

print(f"Recebendo pacotes...")
qtdPacotes = 0
inicio = time.time()
while 1:
pacote = sock.recv(tamanhoPacote)
numero = Pacote(pacotePronto=pacote).obterNumero()
if numero != 999999:
sock.sendto(numero.to_bytes(6, "little"), address)
qtdPacotes += 1
else:
break

agora = time.time()

print("\nCalculando estatísticas...\n")

print("Download")
print("Taxa de pacotes/s: %.2f" % (qtdPacotes / (agora - inicio)))
print("Taxa de bits/s: %.2f" % ((qtdPacotes * tamanhoPacote * 8) / (agora - inicio)))
print("Tempo total executado: %.2f segundos" % (agora - inicio))
print("Total de bits transmitidos: %.2f bits" % (qtdPacotes * tamanhoPacote * 8))

# Limpando buffers e sockets
sock.close()
```

Classe Pacote

```
import sys

class Pacote:
header = {
"numero": bytes("000000", encoding='utf8'),
"tamanho": bytes("0000", encoding='utf8'),
"transmitido": bytes("0", encoding='utf8')
}

dados = bytes()
```

```
def __init__(self, numero = 0, tamanho = 0, pacotePronto = False) -> None:
if not pacotePronto:
self.header["numero"] = bytes(str(numero).zfill(6), encoding='utf8')
self.header["tamanho"] = bytes(str(tamanho).zfill(4), encoding='utf8')
else:
self.header["numero"] = pacotePronto[0:6]
self.header["tamanho"] = pacotePronto[6:10]
self.header["transmitido"] = pacotePronto[10:11]
self.dados = pacotePronto[11:]

def obterNumero(self):
return int(self.header["numero"])

def obterTamanho(self):
return int(self.header["tamanho"])

def obterTransmitido(self):
return bool(int(self.header["transmitido"]))

def obterTamanhoDadosPermitido(self):
return (self.obterTamanho() - 44)

def obterTamanhoDados(self):
return (sys.getsizeof(self.dados) - sys.getsizeof(bytes()))

def inserirDados(self, dados):
tamanhoDados = sys.getsizeof(dados) - sys.getsizeof(bytes())

if tamanhoDados > self.obterTamanhoDadosPermitido():
print("O tamanho dos dados é maior que o permitido no pacote")
return False
self.dados = dados
return True

def montar(self):
header = bytes().join([self.header["numero"], self.header["tamanho"],
self.header["transmitido"]])
payload = self.dados

pacote = bytes().join([header, payload])

return pacote
```

Classe BufferPacote

```
import threading
import time
from pacote import Pacote


class BufferPacotes():
## Limitações
quantidadePacotesPermitido = 0
tamanhoPacote = 0
tamanhoBufferPermitido = 0

## Informações relevantes para a classe
ultimoPacoteCriado = -1
pacotes = {}
pacotesEspera = {}
dados = bytes()

## Estatísticas
erros = []
tempoEspera = 0
pacotesEnviados = 0
tempoMedioEnvio = 0
taxaBitsS = 0
taxaPacotesS = 0
tempoTotal = 0
totalBits = 0
inicio = 0

## Variáveis auxiliares de threading
mutexDados = threading.Lock()
mutexPacotes = threading.Lock()
mutexPacotesEspera = threading.Lock()

def __init__(self, tamanhoPacote, quantidadeMaximaPacotes, tamanhoBuffer,
tempoEsperaMaximo) -> None:
self.tamanhoPacote = tamanhoPacote
self.quantidadePacotesPermitido = quantidadeMaximaPacotes
self.tamanhoBufferPermitido = tamanhoBuffer
self.tempoEspera = tempoEsperaMaximo

def __modificarDados(self, inserir = False):
dados = bytes()

self.mutexDados.acquire()
try:
if not inserir:
dados = self.dados
```

```
self.dados = bytes()
else:
self.dados = bytes().join([self.dados, inserir])
except:
print("Não foi possível modificar os dados.")
finally:
self.mutexDados.release()

return dados

def __modificarPacotes(self, chave, inserir = False):
self.mutexPacotes.acquire()
try:
if not inserir:
pacote = self.pacotes.pop(chave)
else:
pacote = {
"pacote": inserir,
"momentoEnviado": 0
}

self.pacotes[chave] = pacote
except:
print("Não foi possível modificar os pacotes.")
pacote = False
finally:
self.mutexPacotes.release()

return pacote

def iniciar(self):
self.inicio = time.time()

def criarPacotes(self):
dados = self.__modificarDados()
tamanhoPacotePermitido = Pacote(-1,
self.tamanhoPacote).obterTamanhoDadosPermitido()

for i in range(0, len(dados), tamanhoPacotePermitido):
inicio = time.time()
agora = time.time()
while (agora - inicio) < 5:
if len(self.pacotes) < self.quantidadePacotesPermitido:
break
agora = time.time()

if (agora - inicio) >= 5:
break
pacote = Pacote(self.ultimoPacoteCriado+1, self.tamanhoPacote)
```

```
pacote.inserirDados(self.dados[i:(i + tamanhoPacotePermitido)])
self.ultimoPacoteCriado += 1
self.__modificarPacotes(self.ultimoPacoteCriado, pacote.montar())

def __modificarPacotesEspera(self, chave = 0, inserir = False):
pacote = bytes()

self.mutexPacotesEspera.acquire()
try:
if not inserir:
pacote = self.pacotesEspera.pop(chave)
else:
self.pacotesEspera[chave] = inserir
except:
print("O pacote já foi enviado.")
pacote = False
finally:
self.mutexPacotesEspera.release()

return pacote

def obterPacote(self):
erroEnvio = False
try:
if len(self.pacotesEspera) > 0:
numero = list(self.pacotesEspera)[0]
if (time.time() - self.pacotesEspera[numero]["momentoEnviado"]) >
self.tempoEspera:
erroEnvio = True

if erroEnvio:
pacote = self.__modificarPacotesEspera(numero)
if pacote:
agora = time.time()
pacote["momentoEnviado"] = agora
self.__modificarPacotesEspera(numero, pacote)
self.erros.append(numero)
else:
return False
else:
numero = list(self.pacotes)[0]
pacote = self.__modificarPacotes(numero)
agora = time.time()
pacote["momentoEnviado"] = agora

self.__modificarPacotesEspera(numero, pacote)

return pacote["pacote"]
except:
```

```
print("Nenhum pacote foi obtido")
return False

def inserirDados(self, dados):
inicio = time.time()
agora = time.time()
while (agora - inicio) < 5:
if (len(self.dados) + len(dados)) < self.tamanhoBufferPermitido:
self.__modificarDados(dados)
break
agora = time.time()

def confirmarTransmissao(self, chave):
self.__modificarPacotesEspera(chave)
self.pacotesEnviados += 1

def obterTaxaPacotesSegundo(self):
return self.taxaPacotesS

def obterTaxaBitsSegundo(self):
return self.taxaBitsS

def obterTotalBits(self):
self.totalBits = self.taxaBitsS * self.tempoTotal

return self.totalBits

def obterTempoTotal(self):
return self.tempoTotal
def encerrar(self):
agora = time.time()
self.tempoTotal = agora - self.inicio
self.taxaPacotesS = self.pacotesEnviados / self.tempoTotal
self.taxaBitsS = self.tamanhoPacote * self.taxaPacotesS * 8
```

As saídas apresentadas para o script do ponto A:

Upload
Taxa de pacotes/s: 21.07
Taxa de bits/s: 86313.94
Tempo total executado: 20.36 segundos
Total de bits transmitidos: 1757184.00 bits

As saídas apresentadas para o script do ponto B:

Download
Taxa de pacotes/s: 21.42
Taxa de bits/s: 87743.80
Tempo total executado: 20.35 segundos
Total de bits transmitidos: 1785856.00 bits