Simulador de Protocolo de Transferência orientado a conexão

Nomes: Gabriel Fanto Stundner, Luiz Guerra, Ramon Fernandes

Data: 23/11/2020

Pegando o Arquivo texto do Computador

O Arquivo `fileManager.py` lida com o arquivo texto requisitado pelo usuário, onde com ele podemos selecionar o Arquivo do computador, onde tem a seguinte descrição do código:

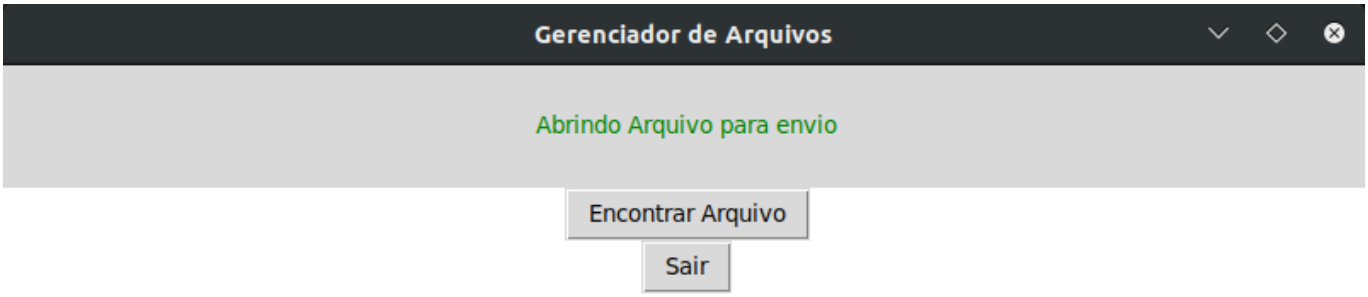
Foi utilizado o **tkinter** para criar uma visualização gráfica para o usuário selecionar o seu arquivo.

A construção da tela de diálogo com o usuário foi construido da seguinte forma:

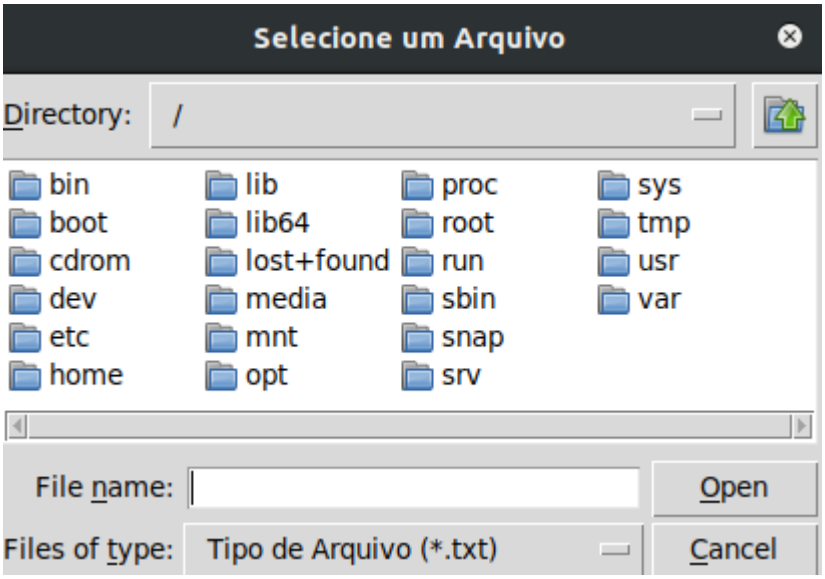
```
from tkinter import *
from tkinter import filedialog

# Criando uma Tela Inicial
window = Tk()

# Definindo o título da tela
window.title("Gerenciador de Arquivos")

# Definindo tamanho da tela
window.geometry("800x500")

# Definindo a cor de fundo da tela
window.config(background = "white")

# Label interno da tela
label_file = Label(window,text = "Abrindo Arquivo para envio",width = 100,
height = 4,fg = "green")

# Botão de encontrar o Arquivo
button_explore = Button(window,text = "Encontrar Arquivo",command =
browseFile)

# Botão de fechar
button_exit = Button(window,text = "Sair",command = exit)

# Colocar as informações na tela
label_file.grid(column = 1, row = 1)
button_explore.grid(column = 1, row = 2)
button_exit.grid(column = 1, row = 3)
```

```
# Esperando uma resposta
window.mainloop()
```

Essa página do Tkinter irá ficar da seguinte forma:



Quando o usuário clicar no botão **Encontrar Arquivo** ele vai poder selecionar o Arquivo desejado dentro dos diretórios internos do programa



Assim que for iniciado o processo de procurar o arquivo internamente dentro do computador, ele vai iniciar um Método onde vai pegar o caminho para o arquivo selecionado e o copiar para um Arquivo interno do Projeto chamado **file.txt** como mostrado o código abaixo:

```
from tkinter import filedialog
from shutil import copyfile

# Tela do File Manager
def browseFile():
    filename = filedialog.askopenfilename(initialdir = "/",title =
"Selecione um Arquivo",filetypes=[("Tipo de Arquivo","*.txt")])
    # Alterar o conteudo
    label_file.configure(text = "Arquivo Aberto: " + filename)
    copyfile(filename,"./file.txt")
```

Só vão ser pegos arquivos .txt do sistema, definido assim para evitar de criar situações complexas com arquivos de video ou audio.

Para iniciarmos o programa do fileManager, devemos primeiro possuir o python3 instalado no computador e baixar o tkinter para python3:

```
> sudo apt-get install python3-tk
```

Depois de instalado, rodamos o programa do fileManager:

```
> python3 fileManager.py
```

Iniciando o Servidor UDP

Nosso Servidor UDP está configurado no Arquivo `udpserver.py`, onde ele deve ser iniciado em um Terminal Específico para ele.

Foi utilizado Sockets para construir o Servidor, onde no python precisamos somente chamar a biblioteca *socket* do python.

```
import socket
```

Foram construídas as Seguintes variáveis para serem usada em nosso Servidor:

```
serverIP = "127.0.0.1" #IP do servidor, sendo Localhost
localPort = 8184 #Porta definida para o Servidor
bufferSize = 300 #Tamanho de bytes sendo enviados
msgFromServer = "Mensagem Encaminhada" #String que o Cliente deve receber
bytesToSend = str.encode(msgFromServer) #Tamanho de bytes da mensagem
```

Como estamos trabalhando com Sockets UDP, devemos construir um Datagram Socket para enviar controlar o envio

```
# Criando um Datagram Socket
UDPServerSocket = socket.socket(family=socket.AF_INET,
type=socket.SOCK_DGRAM)
```

Devemos então fazer um *bind* (definir o IP e Porta do Servidor)

```
UDPServerSocket.bind((serverIP,localPort))
```

Vai ter que possuir uma mensagem para o nosso Terminal para vermos se o Servidor foi inicializado

```
print("UDP Server UP and LISTENING!")
```

Agora temos uma estrutura onde vai ficar esperando uma resposta do cliente, onde vamos tratar com erros que podem ocorrer no Servidor, de forma geral:

```
# Esperando por qualquer Datagram para o Servidor
try:
   # ...
except Exception as e:
    print("\n========================")
    print("Server exception ocurred")
    print(e)
    print("========================")
finally:
    UDPServerSocket.close()
    print("\n========================")
    print("     Server closed")
    print("========================")
```

Dentro do **try** iremos construir uma estrutura **While** que vai permanentemente esperar uma resposta de usuário, onde ele vai ir tratando essa resposta

```
 while(True):
        # Recebendo mensagem do Cliente
        bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
        message = bytesAddressPair[0].decode('utf-16')
        address = bytesAddressPair[1]

        # Apresentação da Mensagem
        print("Mensagem do Cliente: {}".format(message))
```

```
        print("Porta do Cliente: {}".format(address))
        print("\n\n")
        sleep(1)

        # Guardando os dados em um novo arquivo
        f = open("fileCopied.txt","a")
        f.write(message)
        f.close()

        # Enviando resposta ao Cliente
        UDPServerSocket.sendto(bytesToSend,address)
```

- Então seguindo os passos que ele vai fazer:
    - Primeiro ele vai receber a mensagem e vai guardar em uma lista, onde a **mensagem é a primeira posição** e o endereço do usuário é a segunda, onde esse **endereço é a Porta do Cliente**.
    - Apresenta no terminal esses dados, depois de apresentar o sistema vai dar 1 segundo de *sleep* antes de receber a proxima parte.
    - Depois de mostrar qual é a mensagem, ele vai guardar essa String em um novo Arquivo texto, chamado **fileCopied.txt**.

Para iniciarmos o Servidor, devemos em um Terminal possuir o **python3**, onde com ele iremos rodar o nosso Servidor:

```
> python3 udpserver.py
```

Deixe esse Terminal com o Servidor rodando.

Iniciando o Client

O nosso Arquivo para rodar o Cliente se chama `udpclient.py` onde é com ele que iremos ler o arquivo **file.txt** e enviar para o Servidor.

Estamos usando a biblioteca do Socket também para o cliente, onde temos as seguintes variáveis:

```
# Pegar o Arquivo texto no tipo utf-8
file = open("file.txt", "r", encoding='utf-8', errors="strict")

# Ler o arquivo e salvar em utf-16 em uma variável
encodedStr = file.read().encode("utf-16", errors="replace")

# Tupla com os dados para enviar ao Servidor
serverAddressPort = ("127.0.0.1", 8184)

# Número de bytes para enviar para o Servidor
bufferSize = 300
```

```
# Divisão do texto em pacotes de 300 bytes
packages = [encodedStr[i:i+bufferSize] for i in range (0, len(encodedStr),
bufferSize)]
```

Agora criamos um Socket Datagram como é no Servidor

```
# Criando Socket UDP do cliente
UDPClientSocket = socket.socket(family=socket.AF_INET,
type=socket.SOCK_DGRAM)
```

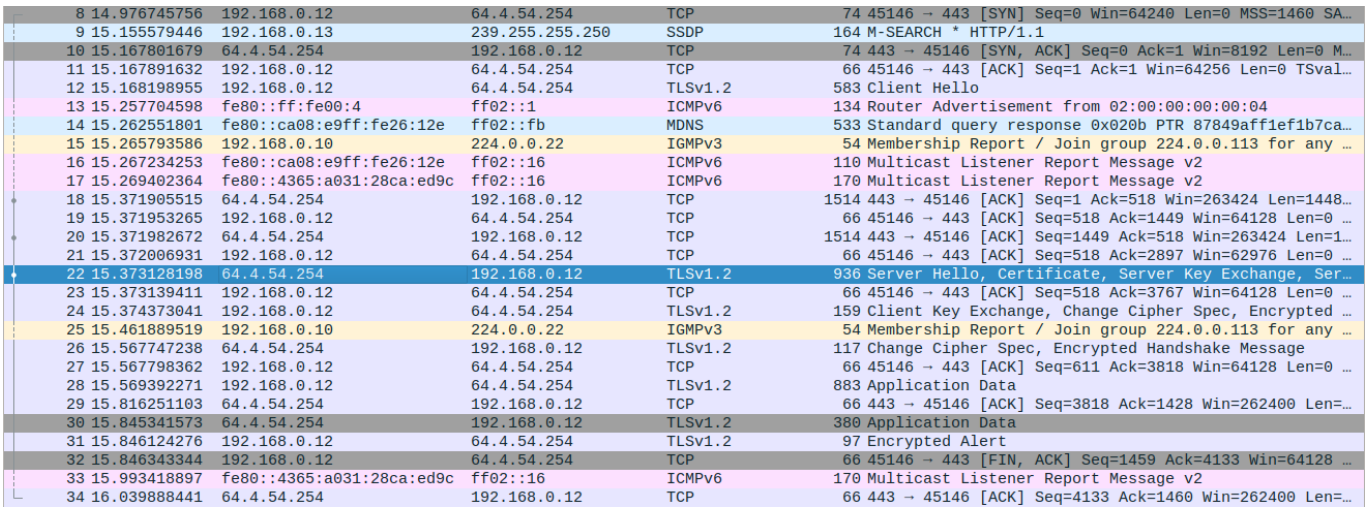
E agora iremos enviar os pacotes para o Servidor, utilizando um For para podermos gerenciar quais pacotes foram enviados para o Servidor e cada vez que é enviado um pacote, o Servidor vai responder para o cliente se foi recebido.

```
i = 0
for package in packages:
    UDPClientSocket.sendto(package,serverAddressPort)
    # Pegando mensagem do Servidor
    msgFromServer = UDPClientSocket.recvfrom(bufferSize)
    print("Enviando pacote {}".format(i))
    print("Mensagem do Servidor: {}".format(msgFromServer[0]))
    i = i + 1
```

Dessa forma temos como ver se o pacote foi enviado e se o Servidor recebeu esse pacote.

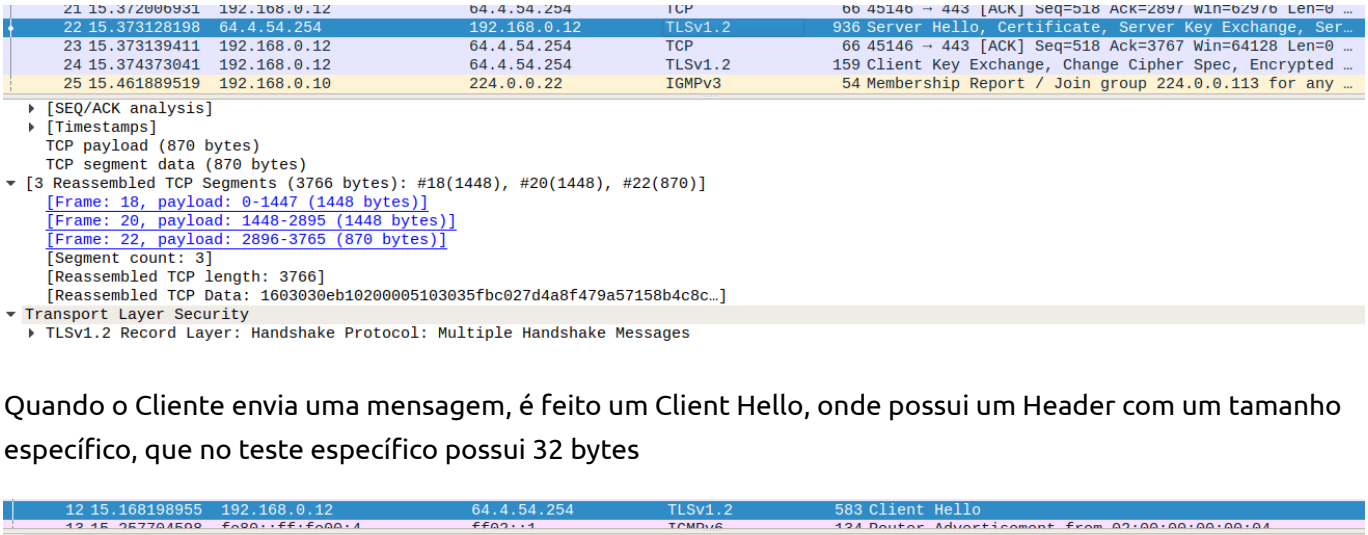
Verificando no Wireshark

Podemos ver os envios TCP pelo Wireshark, onde são divididos em fragmentos cada vez que ele envia, onde as partes do envio aparecem assim no Wireshark:

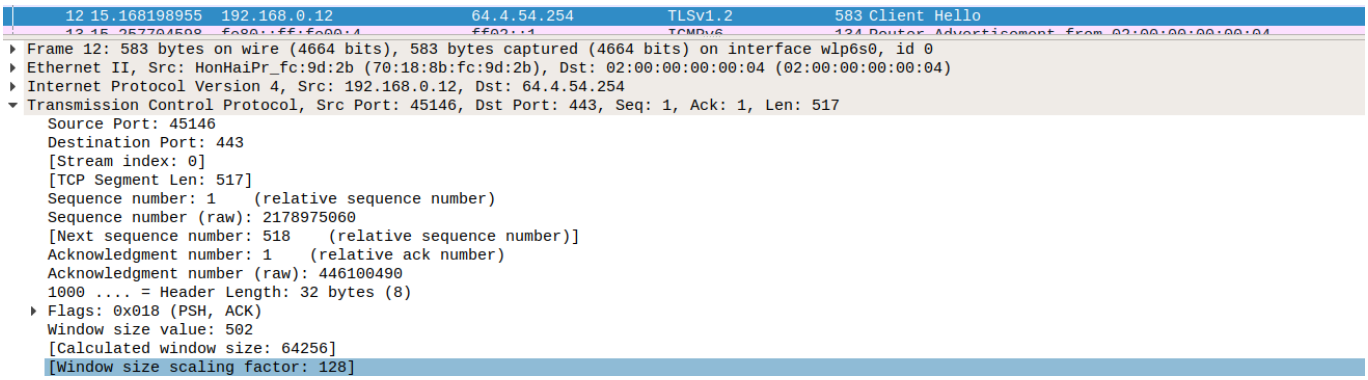


8	14.976745756	192.168.0.12	64.4.54.254	TCP	74	45146 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SA…
9	15.155579446	192.168.0.13	239.255.255.250	SSDP	164	M-SEARCH * HTTP/1.1
10	15.167801679	64.4.54.254	192.168.0.12	TCP	74	443 → 45146 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 M…
11	15.167891632	192.168.0.12	64.4.54.254	TCP	66	45146 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval…
12	15.168198955	192.168.0.12	64.4.54.254	TLSv1.2	583	Client Hello
13	15.257704598	fe80::ff:fe00:4	ff02::1	ICMPv6	134	Router Advertisement from 02:00:00:00:00:04
14	15.262551801	fe80::ca08:e9ff:fe26:12e	ff02::fb	MDNS	533	Standard query response 0x020b PTR 87849aff1ef1b7ca…
15	15.265793586	192.168.0.10	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.113 for any …
16	15.267234253	fe80::ca08:e9ff:fe26:12e	ff02::16	ICMPv6	110	Multicast Listener Report Message v2
17	15.269402364	fe80::4365:a031:28ca:ed9c	ff02::16	ICMPv6	170	Multicast Listener Report Message v2
18	15.371905515	64.4.54.254	192.168.0.12	TCP	1514	443 → 45146 [ACK] Seq=1 Ack=518 Win=263424 Len=1448…
19	15.371953265	192.168.0.12	64.4.54.254	TCP	66	45146 → 443 [ACK] Seq=518 Ack=1449 Win=64128 Len=0 …
20	15.371982672	64.4.54.254	192.168.0.12	TCP	1514	443 → 45146 [ACK] Seq=1449 Ack=518 Win=263424 Len=1…
21	15.372006931	192.168.0.12	64.4.54.254	TCP	66	45146 → 443 [ACK] Seq=518 Ack=2897 Win=62976 Len=0 …
22	15.373128198	64.4.54.254	192.168.0.12	TLSv1.2	936	Server Hello, Certificate, Server Key Exchange, Ser…
23	15.373139411	192.168.0.12	64.4.54.254	TCP	66	45146 → 443 [ACK] Seq=518 Ack=3767 Win=64128 Len=0 …
24	15.374373041	192.168.0.12	64.4.54.254	TLSv1.2	159	Client Key Exchange, Change Cipher Spec, Encrypted …
25	15.461889519	192.168.0.10	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.113 for any …
26	15.567747238	64.4.54.254	192.168.0.12	TLSv1.2	117	Change Cipher Spec, Encrypted Handshake Message
27	15.567798362	192.168.0.12	64.4.54.254	TCP	66	45146 → 443 [ACK] Seq=611 Ack=3818 Win=64128 Len=0 …
28	15.569392271	192.168.0.12	64.4.54.254	TLSv1.2	883	Application Data
29	15.816251103	64.4.54.254	192.168.0.12	TCP	66	443 → 45146 [ACK] Seq=3818 Ack=1428 Win=262400 Len=…
30	15.845341573	64.4.54.254	192.168.0.12	TLSv1.2	380	Application Data
31	15.846124276	192.168.0.12	64.4.54.254	TLSv1.2	97	Encrypted Alert
32	15.846343344	192.168.0.12	64.4.54.254	TCP	66	45146 → 443 [FIN, ACK] Seq=1459 Ack=4133 Win=64128 …
33	15.993418897	fe80::4365:a031:28ca:ed9c	ff02::16	ICMPv6	170	Multicast Listener Report Message v2
34	16.039888441	64.4.54.254	192.168.0.12	TCP	66	443 → 45146 [ACK] Seq=4133 Ack=1460 Win=262400 Len=…

Dai quando o Servidor responde para o cliente ele junta os fragmentos na resposta, como mostra a imagem abaixo:



Quando o Cliente envia uma mensagem, é feito um Client Hello, onde possui um Header com um tamanho específico, que no teste específico possui 32 bytes



Enviando um Arquivo e lidando com ele

Após utilizarmos o arquivo **fileManager.py** para pegarmos o arquivo externo e salvarmos ele no arquivo **file.txt**, iremos utilizar esse arquivo para enviarmos do cliente para o Servidor para podermos construir um novo arquivo chamado **fileCopied.txt**.

Inicializamos o **udpserver.py** em um Terminal, onde ele estará rodando na porta 8184, onde todos os clientes devem ser conectados com ele.

Depois inicializamos o **udpclient.py** em outro Terminal, onde ele vai pegar o arquivo **file.txt**, ler ele e enviar para o servidor, onde cada vez que um cliente é iniciado é uma porta diferente, não sendo a mesma porta do Servidor.

Para esse envio foi desenvolvido 3 métodos que lidam com o envio do Arquivo, onde primeiro é iniciado o **Slow Start**, onde o código dele é o abaixo:

```
# Algoritmo de Slow Start
def slowStart():
    global cwnd, index, receivedAcks
    while cwnd < threshold and index < len(packages):
        print("\nStarting new Slow Start loop. Pacotes restantes:
{}".format(len(packages)-index))
        newAcks = []
        for i in range(cwnd):
            if index >= len(packages):
                break
            message = formatUDP(True, index, packages[index])
```

```
            ack = sendPackage(message)
            print("Sending package {}/{}, ack: {}".format(i+1, cwnd, ack))
            newAcks.append(ack)
            index += 1
        receivedAcks = receivedAcks + newAcks
        failedAcks = verify(receivedAcks)
        while len(failedAcks) > 0:
            failedAcks = fastRetransmit(failedAcks)
        if len(newAcks) == cwnd:
            cwnd += cwnd
        else:
            cwnd = 1
        newAcks.clear
    print("Reached threshold!" if (cwnd>=threshold) else "Ended
streaming!")
```

Quando chegar o limite(threshold) e o arquivo continuar, ele vai entrar no congestion avoidance, onde vai continuar a enviar o programa, onde o código do congestion avoidance é o seguinte:

```
# Algoritmo Congestion Avoidance
def congestionAvoidance():
    global cwnd, index, receivedAcks
    while cwnd >= threshold and index < len(packages):
        print("\nStarting new Congestion Avoidance loop. Pacotes restantes:
{}".format(len(packages)-index))
        newAcks = []
        for i in range(cwnd):
            print("Sending package {}/{}".format(i+1, cwnd))
            newAcks.append(sendPackage(formatUDP(True, index,
packages[index])))
            index += 1
        failedAcks = verify(receivedAcks)
        while len(failedAcks) > 0:
            failedAcks = fastRetransmit(failedAcks)
        if len(newAcks) == cwnd:
            cwnd += 1
        else:
            cwnd /= 2
```

Em questão á perda de pacotes, criamos um Método chamado **verify()** onde verifica se os pacotes estão sendo enviados com sucesso, tem uma parte comentada que serve para testarmos a perda de pacotes, como mostra o código abaixo:

```
def verify(receivedAcks):
    notReceivedPackages = []
    # PARA SIMULAR A PERDA DE PACOTES
    # if len(receivedAcks) > 5:
    #     del receivedAcks[2]
    for i in range(1, len(receivedAcks)):
```

```
            if i not in receivedAcks:
                notReceivedPackages = notReceivedPackages + [i]
        return notReceivedPackages
```

Se ocorrer a perda de pacotes ele vai reenviar, utilizando o método **fastRetransmit**, como mostra o código abaixo:

```
def fastRetransmit(failedPackages):
    restoredPackages = []
    for i in range(len(failedPackages)):
        sendPackage(formatUDP(True, failedPackages[i],
packages[failedPackages[i]]))
        restoredPackages.append(failedPackages[i])
    for i in range(len(restoredPackages)):
        if restoredPackages[i] in failedPackages:
            failedPackages.remove(restoredPackages[i])
    return failedPackages
```

Um exemplo de perda de pacotes e reenvio:

```
Recebido pacote do cliente.
IP: 127.0.0.1, porta: 58749
Fragment with index: 5
Fragment size: 145
Conteúdo do pacote foi adicionado com sucesso no arquivo fileCopied
.txt

Recebido pacote do cliente.
IP: 127.0.0.1, porta: 58749
Fragment with index: 31
Fragment size: 71
Conteúdo do pacote foi adicionado com sucesso no arquivo fileCopied
.txt

Recebido pacote do cliente.
IP: 127.0.0.1, porta: 58749
Fragment with index: 3
Fragment size: 145
Conteúdo do pacote foi adicionado com sucesso no arquivo fileCopied
.txt

Recebido pacote do cliente.
IP: 127.0.0.1, porta: 58749
Fragment with index: 4
Fragment size: 145
Conteúdo do pacote foi adicionado com sucesso no arquivo fileCopied
.txt

Recebido pacote do cliente.
IP: 127.0.0.1, porta: 58749
Fragment with index: 5
Fragment size: 145
Conteúdo do pacote foi adicionado com sucesso no arquivo fileCopied
.txt

Recebido pacote do cliente.
IP: 127.0.0.1, porta: 58749
Fragment with index: 6
Fragment size: 145
Conteúdo do pacote foi adicionado com sucesso no arquivo fileCopied
.txt
```

```
Starting new Slow Start loop. Pacotes restantes: 31
Sending package 1/2, ack: 2
Sending package 2/2, ack: 3

Starting new Slow Start loop. Pacotes restantes: 29
Sending package 1/4, ack: 4
Sending package 2/4, ack: 5
Sending package 3/4, ack: 6
Sending package 4/4, ack: 7

Starting new Slow Start loop. Pacotes restantes: 25
Sending package 1/8, ack: 8
Sending package 2/8, ack: 9
Sending package 3/8, ack: 10
Sending package 4/8, ack: 11
Sending package 5/8, ack: 12
Sending package 6/8, ack: 13
Sending package 7/8, ack: 14
Sending package 8/8, ack: 15

Starting new Slow Start loop. Pacotes restantes: 17
Sending package 1/16, ack: 16
Sending package 2/16, ack: 17
Sending package 3/16, ack: 18
Sending package 4/16, ack: 19
Sending package 5/16, ack: 20
Sending package 6/16, ack: 21
Sending package 7/16, ack: 22
Sending package 8/16, ack: 23
Sending package 9/16, ack: 24
Sending package 10/16, ack: 25
Sending package 11/16, ack: 26
Sending package 12/16, ack: 27
Sending package 13/16, ack: 28
Sending package 14/16, ack: 29
Sending package 15/16, ack: 30
Sending package 16/16, ack: 31

Starting new Slow Start loop. Pacotes restantes: 1
Sending package 1/32, ack: 32
Ended streaming!
M4TRIX:Project (test_merge *) >
```

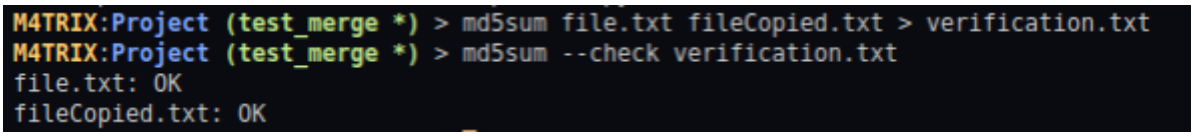
Os *index* mostram que os pacotes de index 5 foi perdido e depois foi reenviado de novo.

Após lido o Arquivo e criado o novo Arquivo texto fileCopied.txt, podemos verificar se eles são iguais da seguinte forma:

```
# Verificar as hashes dos arquivos
> md5sum file.txt fileCopied.txt > verification.txt
```

```
# Verificar a integridade dos arquivos
> md5sum --check verification.txt
```

E é esperado a seguinte resposta do terminal:



Com isso, significa que foi transmitido de forma correta o arquivo e está identico ao original

---

Repositório Original

O Link para o Repositório Original onde esse projeto foi trabalhado é
https://github.com/F4NT0/Socket_Network_Transfer