

Relatório do trabalho prático de sockets Redes de computadores I

Professor Filipe Nunes Ribeiro

Arthur Enrique, Hellrison Soares, Yuri Ribeiro
Universidade Federal de Ouro Preto - Campus ICEA

Sumário

Introdução.....	3
Implementação.....	3
Protocolo TCP.....	4
Código do cliente.....	4
Código do servidor.....	6
Tempos de execução.....	7
Protocolo UDP.....	10
Código do cliente.....	10
Código do servidor.....	11
Tempos de execução.....	12
Análise dos resultados e conclusão.....	14
Referências Bibliográficas.....	15

Introdução

Este trabalho tem como objetivo desenvolver uma aplicação cliente-servidor para realizar o teste de envio de arquivos, utilizando os protocolos TCP (Transmission Control Protocol) e UDP (User Datagram Protocol). Na aplicação, feita na linguagem Python, o cliente solicita o envio de um arquivo .txt para o servidor, escolhendo o tamanho do arquivo (pequeno, médio ou grande). O servidor, ao receber a solicitação do cliente, realiza a transmissão do arquivo linha a linha, enquanto o cliente recebe as linhas e reconstrói o arquivo, salvando um novo arquivo de texto. Após o envio do arquivo, o cliente informa ao usuário o tempo total gasto na transmissão.

Nessa implementação, fazemos o uso de sockets para a criação do servidor e do cliente. Sockets são canais de comunicação fim a fim e de duas vias entre dois programas executando em dispositivos da rede. Os protocolos TCP e UDP, que utilizam sockets, são amplamente utilizados na comunicação em redes de computadores. O TCP é um protocolo orientado a conexão, que oferece um serviço confiável e garante a entrega ordenada dos dados transmitidos. Ele estabelece uma conexão entre o remetente e o destinatário, garantindo a integridade e a precisão na transmissão dos dados. Esse protocolo é ideal para aplicações que requerem entrega precisa e sem perdas de dados, como transferência de arquivos. (Tanenbaum, 2003)

Por outro lado, o UDP é um protocolo não orientado a conexão. Ele é mais leve e rápido que o TCP, porém não oferece as mesmas garantias de entrega confiável e ordenada. O UDP é adequado para aplicações em que a velocidade e a baixa latência são prioritárias, mesmo que alguns pacotes possam ser perdidos ou cheguem fora de ordem. Ele é comumente utilizado em aplicações de streaming de mídia e jogos online, em que a velocidade de transmissão é essencial. (Tanenbaum, 2003)

Em resumo, este trabalho se propõe a implementar a aplicação cliente-servidor usando sockets com TCP e UDP para o envio de arquivos, comparando os dois protocolos e suas respectivas velocidades de transmissão.

Implementação

Como dito anteriormente, foi utilizada a linguagem de programação Python para a implementação do código do trabalho. Por ser um código relativamente simples, não foi necessária nenhuma lib externa, apenas as libs socket e time, do próprio Python. As duas implementações seguiram o mesmo padrão, a única alteração entre os códigos é na forma de realizar a conexão, que é diferente para cada protocolo.

Protocolo TCP

Código do cliente



```
1 def main(argv):
2     try:
3         with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as TCPClientSocket:
4             TCPClientSocket.connect((HOST, PORT))
5             print("Conectado ao servidor!")
6
7             while(True):
8                 print("\nSelecione o que deseja fazer:")
9                 print("1. Receber um arquivo do servidor")
10                print("2. Encerrar o programa\n")
11
12                option = input("Digite o número da opção desejada: ")
13
14                if (option == '1'):
15                    receivefile(TCPClientSocket)
16                    continue
17
18                elif (option == '2'):
19                    print('\nFechando a conexão com o servidor!')
20                    TCPClientSocket.close()
21                    print('Encerrando o programa!')
22                    return
23                else:
24                    print("\nOpção inválida!\n")
25                    continue
26
27 except Exception as error:
28     print("Exceção - Programa será encerrado!")
29     print(error)
30     return
```

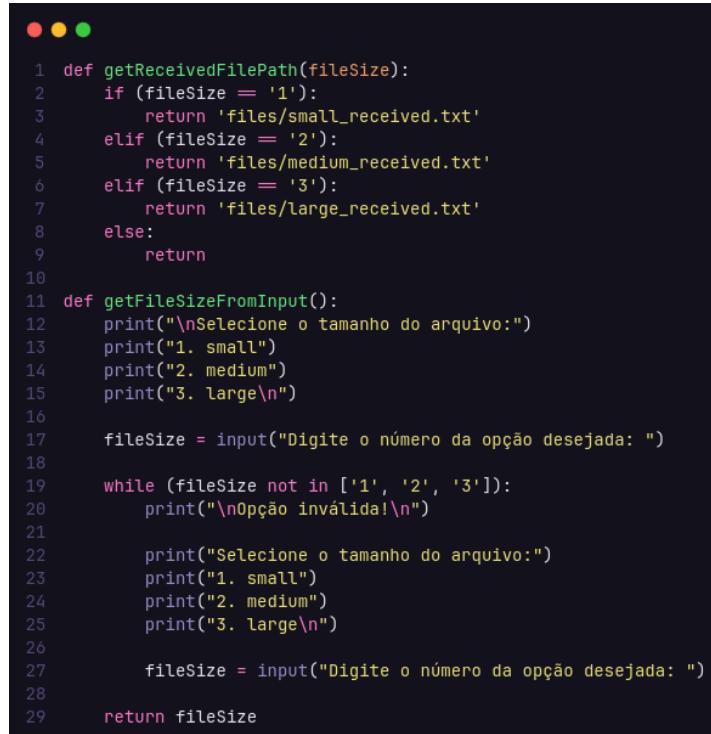
Figura 1 - Código main do cliente TCP

Todo o código main do cliente é envolto por um try except, para tratamento de erros caso ocorra algum problema na execução. O código começa criando um socket TCP e conectando ao servidor utilizando a porta e host (IP) no qual o servidor está rodando. Em seguida, é escrito na tela o menu para que o cliente escolha o que quer fazer (encerrar o programa ou receber algum arquivo do servidor).



```
1 def receiveFile(TCPClientSocket):
2     fileSize = getFileSizeFromInput()
3
4     TCPClientSocket.send(fileSize.encode())
5
6     filePath = getReceivedFilePath(fileSize)
7
8     file = open(filePath, "w")
9
10    startTime = time.time()
11
12    # loop para receber cada linha do arquivo
13    while True:
14        # recebe a linha
15        data = TCPClientSocket.recv(BUFFER_SIZE)
16
17        receivedText = data.decode('utf-8')
18
19        print("texto recebido: " + receivedText)
20
21        receivedEndFlag = False
22
23        # se recebeu a flag, remove ela da string
24        if (END_TRANSMISSION_FLAG in receivedText):
25            receivedText = receivedText.replace(END_TRANSMISSION_FLAG, '')
26            receivedEndFlag = True
27
28        file.write(receivedText)
29
30        # se recebeu a flag, finaliza o loop
31        if (receivedEndFlag):
32            break
33
34    endTime = time.time()
35
36    elapsedTimeInSeconds = endTime - startTime
37
38    file.close()
39
40    print("\nArquivo recebido!")
41    print("Tempo gasto na transferência: {:.2f} segundos".format(elapsedTimeInSeconds))
```

Figura 2 - Função que recebe o arquivo no cliente TCP



```

1 def getReceivedFilePath(fileSize):
2     if (fileSize == '1'):
3         return 'files/small_received.txt'
4     elif (fileSize == '2'):
5         return 'files/medium_received.txt'
6     elif (fileSize == '3'):
7         return 'files/large_received.txt'
8     else:
9         return
10
11 def getFileSizeFromInput():
12     print("\nSelecione o tamanho do arquivo:")
13     print("1. small")
14     print("2. medium")
15     print("3. large\n")
16
17     fileSize = input("Digite o número da opção desejada: ")
18
19     while (fileSize not in ['1', '2', '3']):
20         print("\nOpção inválida!\n")
21
22         print("Selecione o tamanho do arquivo:")
23         print("1. small")
24         print("2. medium")
25         print("3. large\n")
26
27     fileSize = input("Digite o número da opção desejada: ")
28
29     return fileSize

```

Figura 3 - Funções auxiliares do cliente TCP

A função receiveFile (figura 2) é a principal função do código. Como o próprio nome dela diz, ela é responsável por executar todo o processo de recepção de arquivos do servidor. Para começar, ela permite que o cliente escolha o tamanho do arquivo que deseja receber utilizando a função getFileSizeFromInput (figura 3). Em seguida, envia a solicitação para o servidor com o tamanho do arquivo a ser recebido, e cria um novo arquivo no qual irá escrever as strings recebidas (o path do arquivo é retornado pela função getReceivedFilePath, mostrada também na figura 3).

Então a função entra em um loop para receber o arquivo do servidor, escrevendo a string recebida no arquivo aberto, até que o mesmo envie uma flag que indica o fim da transmissão do arquivo. Ao receber essa flag, o cliente remove a flag de dentro do texto recebido, escreve o restante no arquivo e finaliza o recebimento, escrevendo na tela o tempo gasto na transmissão.

É importante destacar aqui que, apesar de o servidor enviar uma linha por vez, como isso acontece muito rápido, o cliente não consegue receber uma linha de cada vez, e acaba por receber várias linhas juntas. Por isso, a última mensagem recebida pelo cliente vem com a flag dentro da string, que é uma junção das últimas linhas do arquivo mais a flag no final.

Código do servidor



```
1 def main(argv):
2     try:
3         # AF_INET: indica o protocolo IPv4. SOCK_STREAM: tipo de socket para TCP,
4         with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as TCPServerSocket:
5             TCPServerSocket.bind((HOST, PORT))
6             print('Servidor iniciado. Aguardando conexões... ')
7             while True:
8                 TCPServerSocket.listen()
9                 TCPClientSocket, addr = TCPServerSocket.accept()
10                print('Conectado ao cliente no endereço:', addr)
11                t = Thread(target=handleNewClientSocket, args=(TCPClientSocket,addr))
12                t.start()
13            except Exception as error:
14                print("Erro na execução do servidor!!")
15                print(error)
16            return
```

Figura 4 - Código main do servidor TCP

De forma similar ao cliente, o código main do servidor também possui um try except para tratamento de erros. O código começa iniciando um novo socket de servidor TCP, usando o comando bind para executar o servidor no host e porta determinados e em seguida entra em um loop para escutar novas conexões de clientes. Ao receber uma nova conexão, o servidor a aceita, criando uma nova thread em seu processo para executar a função handleNewClientSocket, exibida a seguir na figura 5.



```
1 def handleNewClientSocket(TCPClientSocket,addr):
2     while True:
3         try:
4             data = TCPClientSocket.recv(BUFFER_SIZE)
5
6             if not data:
7                 break
8
9             receivedText = data.decode('utf-8') # converte os bytes em string
10
11             filePath = getFilePath(receivedText)
12
13             print("Iniciando transmissão de arquivo...")
14
15             # abre o arquivo para leitura
16             with open(filePath, 'r') as file:
17
18                 # loop para enviar cada linha do arquivo
19                 for line in file:
20                     print("\nenviando a linha: " + line)
21                     TCPClientSocket.send(line.encode())
22                     time.sleep(0.000000000001)
23
24             file.close()
25
26             print("\n" + END_TRANSMISSION_FLAG)
27
28             TCPClientSocket.send(END_TRANSMISSION_FLAG.encode())
29         except Exception as error:
30             print("Erro na conexão com o cliente!!")
31             print(error)
32             return
```

Figura 5 - Função que trata a conexão com um novo cliente no servidor TCP



```
1 def getPath(receivedText):
2     if (receivedText == '1'):
3         return 'files/small.txt'
4     elif (receivedText == '2'):
5         return 'files/medium.txt'
6     elif (receivedText == '3'):
7         return 'files/large.txt'
8     else:
9         return
```

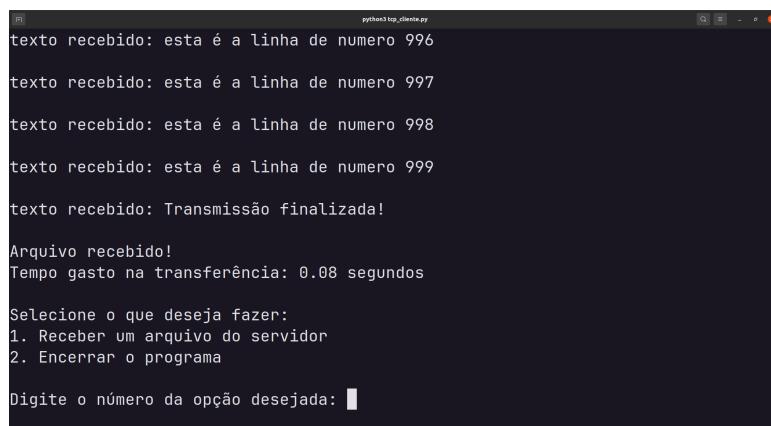
Figura 6 - Função auxiliar do servidor TCP

Essa função é a responsável por tratar a conexão com um novo cliente. O código entra em um loop para aguardar o recebimento do tamanho do arquivo que o cliente deseja receber. Ao receber esse dado, a função `getPath` (figura 6) retorna o path do arquivo a ser enviado, baseado no que foi recebido.

O servidor então inicia a transferência do arquivo para o cliente, abrindo o arquivo e entrando em um loop que lê linha por linha, enviando cada uma delas para o cliente. Vale destacar o `sleep` na linha 22, que evita que o loop execute tão rápido que o cliente não consegue receber todos os dados. Sem ele, o cliente perderia a maior parte das linhas enviadas, iria receber a flag e salvar apenas a última string no arquivo final.

Por fim, quando todas as linhas do arquivo foram enviadas, o servidor termina o loop, fecha o arquivo e envia para o cliente a flag de fim de transmissão, finalizando o processo. Assim, o servidor retorna ao estado de aguardo do recebimento do tamanho do próximo arquivo que o cliente deseja receber.

Tempos de execução



```
python3 tcp_cliente.py
texto recebido: esta é a linha de numero 996
texto recebido: esta é a linha de numero 997
texto recebido: esta é a linha de numero 998
texto recebido: esta é a linha de numero 999
texto recebido: Transmissão finalizada!
Arquivo recebido!
Tempo gasto na transferência: 0.08 segundos
Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa

Digite o número da opção desejada: 1
```

Figura 7 - Tempo de execução da transmissão de um arquivo pequeno usando protocolo TCP, com o cliente e servidor rodando no mesmo dispositivo

```

python3 tcp_cliente.py
texto recebido: esta é a linha de numero 9996
texto recebido: esta é a linha de numero 9997
texto recebido: esta é a linha de numero 9998
texto recebido: esta é a linha de numero 9999
texto recebido: Transmissão finalizada!
Arquivo recebido!
Tempo gasto na transferência: 0.71 segundos
Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa

Digite o número da opção desejada: 
```

Figura 8 - Tempo de execução da transmissão de um arquivo médio usando protocolo TCP, com o cliente e servidor rodando no mesmo dispositivo

```

python3 tcp_cliente.py
texto recebido: esta é a linha de numero 99996
texto recebido: esta é a linha de numero 99997
texto recebido: esta é a linha de numero 99998
texto recebido: esta é a linha de numero 99999
texto recebido: Transmissão finalizada!
Arquivo recebido!
Tempo gasto na transferência: 7.02 segundos
Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa

Digite o número da opção desejada: 
```

Figura 9 - Tempo de execução da transmissão de um arquivo grande usando protocolo TCP, com o cliente e servidor rodando no mesmo dispositivo

Como podemos ver nas figuras 7, 8 e 9, o código demorou 0.08 segundos para transferir um arquivo pequeno, 0.71 segundos para transferir um arquivo médio, e 7.02 segundos para transferir um arquivo grande, com o cliente e o servidor executando no mesmo dispositivo.

```

tcp-udp-socket-python - python3 tcp_cliente.py - python3 -- Python - 95x24
esta é a linha de numero 987
esta é
texto recebido: a linha de numero 988
esta é a linha de numero 989
esta é a linha de numero 990
esta é a linha de numero 991
esta é a linha de numero 992
esta é a linha de numero 993
esta é a linha de numero 994
esta é a linha de numero 995
esta é a linha de numero 996
esta é a linha de numero 997
esta é a linha de numero 998
esta é a linha de numero 999
Transmissão finalizada!

Arquivo recebido!
Tempo gasto na transferência: 0.12 segundos

Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa

Digite o número da opção desejada: 
```

Figura 10 - Tempo de execução da transmissão de um arquivo pequeno usando protocolo TCP, com o cliente e servidor rodando em dispositivos separados

```
esta é a linha de numero 9987
esta é a linha de numero 9988
esta é a linha de numero 9989
esta é a lin
texto recebido: ha de numero 9990
esta é a linha de numero 9991
esta é a linha de numero 9992
esta é a linha de numero 9993
esta é a linha de numero 9994
esta é a linha de numero 9995
esta é a linha de numero 9996
esta é a linha de numero 9997
esta é a linha de numero 9998
esta é a linha de numero 9999
Transmissão finalizada!

Arquivo recebido!
Tempo gasto na transferência: 0.85 segundos

Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa

Digite o número da opção desejada:
```

Figura 11 - Tempo de execução da transmissão de um arquivo médio usando protocolo TCP, com o cliente e servidor rodando em dispositivos separados

```
esta é a linha de numero 99986
esta é a linha de numero 99987
esta é a linha de numero 99988
esta é a linha de numero 99989
esta é a linha de numero 99990
esta é a linha de numero 99991
esta é a linha de numero 99992
esta é a linha de numero 99993
esta é a linha de numero 99994
esta é a linha de numero 99995
esta é a linha de numero 99996
esta é a linha de numero 99997
esta é a linha de numero 99998
esta é a linha de numero 99999
Transmissão finalizada!

Arquivo recebido!
Tempo gasto na transferência: 8.50 segundos

Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa

Digite o número da opção desejada:
```

Figura 12 - Tempo de execução da transmissão de um arquivo grande usando protocolo TCP, com o cliente e servidor rodando em dispositivos separados

Já para a transmissão em dispositivos diferentes, o código demorou 0.12 segundos para transferir um arquivo pequeno, 0.85 segundos para transferir um arquivo médio, e 8.50 segundos para transferir um arquivo grande, como visto nas figuras 10, 11 e 12.

Protocolo UDP

Código do cliente

```
 1 def main(argv):
 2     try:
 3         # Cria o socket UDP
 4         with socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM) as UDPClientSocket:
 5             while(True):
 6                 print("\nSelecione o que deseja fazer:")
 7                 print("1. Receber um arquivo do servidor")
 8                 print("2. Encerrar o programa\n")
 9
10                 option = input("Digite o número da opção desejada: ")
11
12                 if (option == '1'):
13                     receiveFile(UDPClientSocket)
14                     continue
15
16                 elif (option == '2'):
17                     print('Encerrando o programa!')
18                     return
19                 else:
20                     print("\nOpção inválida!\n")
21                     continue
22
23         except Exception as error:
24             print("Exceção - Programa será encerrado!")
25             print(error)
26             return
```

Figura 13 - Função main do código do cliente UDP

```
 1 def receiveFile(UDPClientSocket):
 2     fileSize = getFileSizeFromInput()
 3
 4     UDPClientSocket.sendto(fileSize.encode(), (HOST, PORT))
 5
 6     filePath = getReceivedFilePath(fileSize)
 7
 8     file = open(filePath, "w")
 9
10     startTime = time.time()
11
12     # loop para receber cada linha do arquivo
13     while True:
14         # recebe a linha
15         data = UDPClientSocket.recvfrom(BUFFER_SIZE)
16
17         receivedText = data[0].decode('utf-8')
18
19         print("texto recebido: " + receivedText)
20
21         receivedEndFlag = False
22
23         # se recebeu a flag, remove ela da string
24         if (END_TRANSMISSION_FLAG in receivedText):
25             receivedText = receivedText.replace(END_TRANSMISSION_FLAG, '')
26             receivedEndFlag = True
27
28         file.write(receivedText)
29
30         # se recebeu a flag, finaliza o loop
31         if (receivedEndFlag):
32             break
33
34     endTime = time.time()
35
36     elapsedTimeInSeconds = endTime - startTime
37
38     file.close()
39
40     print("\nArquivo recebido!")
41     print("Tempo gasto na transferência: {:.2f} segundos".format(elapsedTimeInSeconds))
```

Figura 14 - Função que recebe o arquivo no cliente UDP

Como podemos ver nas figuras 13 e 14, o código do cliente UDP é basicamente o mesmo do TCP. As funções auxiliares getFileSizeFromInput e getReceivedFilePath não foram exibidas aqui pois são exatamente iguais às funções de mesmo nome no código do cliente TCP, portanto podem ser conferidas na figura 3.

A diferença entre o cliente TCP e UDP está na forma que a conexão com o servidor é feita. A criação do socket utiliza parâmetros diferentes para criar um cliente UDP, e o host e porta do servidor são definidos na hora de enviar o dado.

Código do servidor

```
● ● ●
1 def main(argv):
2     try:
3         # Create a datagram socket
4         UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
5         # Bind to address and ip
6         UDPServerSocket.bind((HOST, PORT))
7         print('Servidor iniciado!\n')
8         # Listen for incoming datagrams
9         listenToClient(UDPServerSocket)
10    except Exception as error:
11        print("Erro na execução do servidor!!")
12        print(error)
13        return
14
```

Figura 15 - Código main do servidor UDP

```
● ● ●
1 def listenToClient(UDPServerSocket):
2     while(True):
3         data = UDPServerSocket.recvfrom(BUFFER_SIZE)
4
5         receivedText = data[0].decode('utf-8')
6         clientAddress = data[1]
7
8         filePath = getFilePath(receivedText)
9
10        print("Iniciando transmissão de arquivo...")
11
12        # abre o arquivo para leitura
13        with open(filePath, 'r') as file:
14
15            # loop para enviar cada linha do arquivo
16            for line in file:
17                print("\nenviando a linha: " + line)
18                UDPServerSocket.sendto(line.encode(), clientAddress)
19                time.sleep(0.0001)
20
21        file.close()
22
23        print("\n" + END_TRANSMISSION_FLAG)
24
25        UDPServerSocket.sendto(END_TRANSMISSION_FLAG.encode(), clientAddress)
```

Figura 16 - Função que recebe dados dos clientes no servidor UDP

No código main do servidor UDP, também realizamos o procedimento padrão de declarar um socket UDP e usar o comando bind para linkar a porta e o host ao servidor. Em seguida, invocamos a função listenToClient, que executa um loop aguardando o recebimento de dados dos clientes.

Como podemos ver, a conexão entre cliente e servidor pelo protocolo UDP é bem mais simples do que no TCP. O servidor pode receber dados de qualquer cliente e não precisa criar uma thread separada para cada cliente que deseja se conectar, como é feito no protocolo TCP. Ao receber uma mensagem de um cliente, o servidor também recebe seu endereço, assim consegue saber para onde enviar uma possível mensagem de resposta.

As demais partes do código são bem parecidas com o servidor do protocolo TCP, mantendo a lógica de abrir o arquivo ao receber do cliente a informação do tamanho a ser enviado, enviar

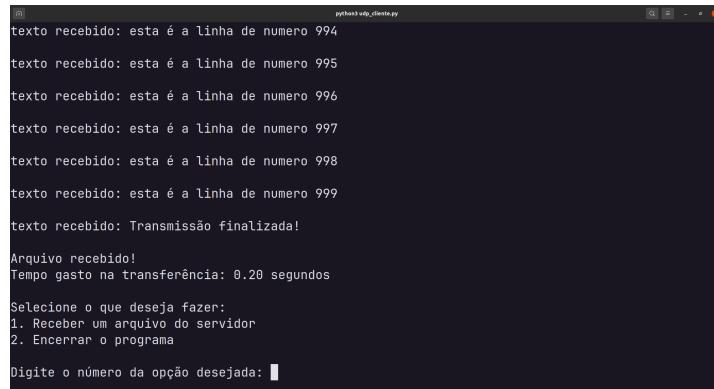
linha por linha do arquivo, e ao final enviar a flag de fim de transmissão. A função getPath também foi excluída dos prints aqui, pois também é igual à mesma função do servidor TCP, portanto pode ser conferida na figura 6.

Porém, existe uma pequena diferença entre os códigos que impacta totalmente no tempo de transmissão do arquivo. Utilizando o mesmo tempo de sleep do TCP no UDP, o código frequentemente perdia pacotes devido à velocidade de envio. E muitas vezes o pacote perdido era o último pacote, o qual contém a flag de fim de transmissão. Assim, como a flag não chegava no cliente, o código entrava em um loop infinito esperando receber a sinalização de que a transmissão do arquivo acabou, que nunca ia chegar.

Para resolver isso, precisamos aumentar consideravelmente o tempo de sleep do código do servidor. Entretanto, dependendo da ocasião, ainda é possível que ocorram perdas consideráveis, e que com isso o código não finalize a transmissão, entrando nesse loop infinito.

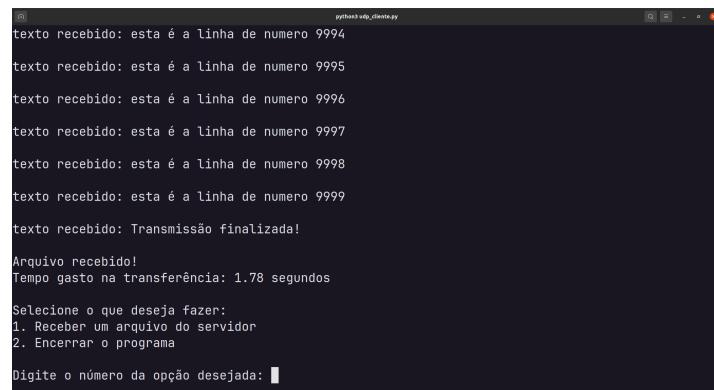
A alteração do tempo de sleep afetou consideravelmente os tempos de transmissão do protocolo UDP, como vemos a seguir.

Tempos de execução



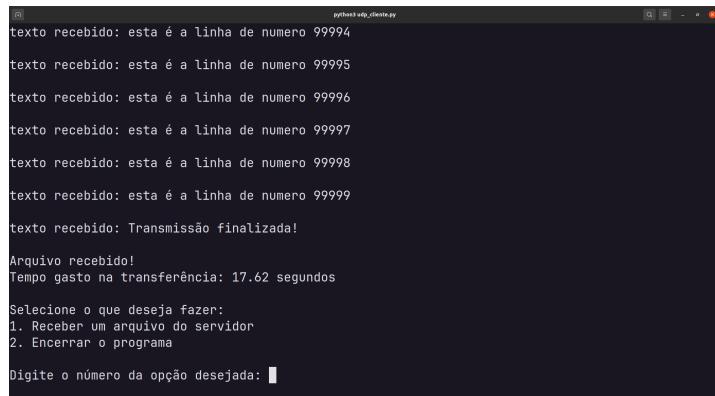
```
python3 udp_cliente.py
texto recebido: esta é a linha de numero 994
texto recebido: esta é a linha de numero 995
texto recebido: esta é a linha de numero 996
texto recebido: esta é a linha de numero 997
texto recebido: esta é a linha de numero 998
texto recebido: esta é a linha de numero 999
texto recebido: Transmissão finalizada!
Arquivo recebido!
Tempo gasto na transferência: 0.20 segundos
Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa
Digite o número da opção desejada: 
```

Figura 17 - Tempo de execução da transmissão de um arquivo pequeno usando protocolo UDP, com o cliente e servidor rodando no mesmo dispositivo



```
python3 udp_cliente.py
texto recebido: esta é a linha de numero 9994
texto recebido: esta é a linha de numero 9995
texto recebido: esta é a linha de numero 9996
texto recebido: esta é a linha de numero 9997
texto recebido: esta é a linha de numero 9998
texto recebido: esta é a linha de numero 9999
texto recebido: Transmissão finalizada!
Arquivo recebido!
Tempo gasto na transferência: 1.78 segundos
Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa
Digite o número da opção desejada: 
```

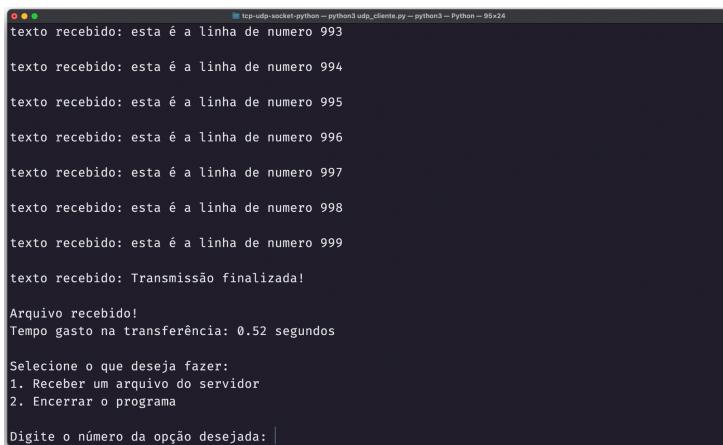
Figura 18 - Tempo de execução da transmissão de um arquivo média usando protocolo UDP, com o cliente e servidor rodando no mesmo dispositivo



```
python3 udp_cliente.py
texto recebido: esta é a linha de numero 99994
texto recebido: esta é a linha de numero 99995
texto recebido: esta é a linha de numero 99996
texto recebido: esta é a linha de numero 99997
texto recebido: esta é a linha de numero 99998
texto recebido: esta é a linha de numero 99999
texto recebido: esta é a linha de numero 99999
texto recebido: Transmissão finalizada!
Arquivo recebido!
Tempo gasto na transferência: 17.62 segundos
Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa
Digite o número da opção desejada: |
```

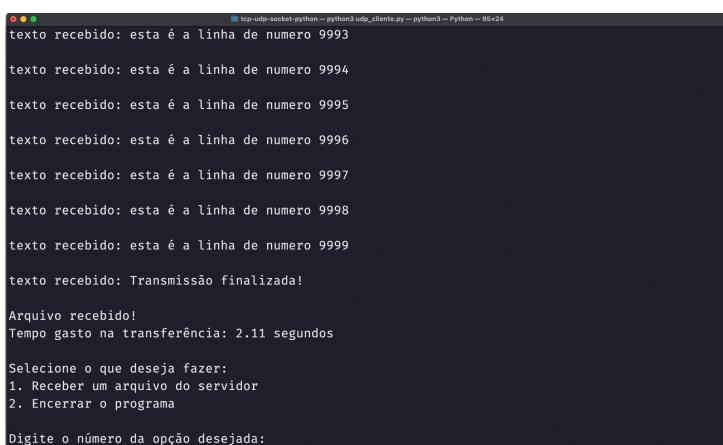
Figura 19 - Tempo de execução da transmissão de um arquivo grande usando protocolo UDP, com o cliente e servidor rodando no mesmo dispositivo

Como podemos ver nas figuras 17, 18 e 19, o código demorou 0.20 segundos para transferir um arquivo pequeno, 1.78 segundos para transferir um arquivo médio, e 17.62 segundos para transferir um arquivo grande.



```
tcp-udp-socket-python -- python3 udp_cliente.py -- python3 -- Python -- 95x24
texto recebido: esta é a linha de numero 993
texto recebido: esta é a linha de numero 994
texto recebido: esta é a linha de numero 995
texto recebido: esta é a linha de numero 996
texto recebido: esta é a linha de numero 997
texto recebido: esta é a linha de numero 998
texto recebido: esta é a linha de numero 999
texto recebido: Transmissão finalizada!
Arquivo recebido!
Tempo gasto na transferência: 0.52 segundos
Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa
Digite o número da opção desejada: |
```

Figura 20 - Tempo de execução da transmissão de um arquivo pequeno usando protocolo UDP, com o cliente e servidor rodando no mesmo dispositivo



```
tcp-udp-socket-python -- python3 udp_cliente.py -- python3 -- Python -- 95x24
texto recebido: esta é a linha de numero 9993
texto recebido: esta é a linha de numero 9994
texto recebido: esta é a linha de numero 9995
texto recebido: esta é a linha de numero 9996
texto recebido: esta é a linha de numero 9997
texto recebido: esta é a linha de numero 9998
texto recebido: esta é a linha de numero 9999
texto recebido: Transmissão finalizada!
Arquivo recebido!
Tempo gasto na transferência: 2.11 segundos
Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa
Digite o número da opção desejada: |
```

Figura 21 - Tempo de execução da transmissão de um arquivo médio usando protocolo UDP, com o cliente e servidor rodando no mesmo dispositivo

```

tcp-udp-socket-python - python3 udp_cliente.py - python3 - Python - 96x24
texto recebido: esta é a linha de numero 99993
texto recebido: esta é a linha de numero 99994
texto recebido: esta é a linha de numero 99995
texto recebido: esta é a linha de numero 99996
texto recebido: esta é a linha de numero 99997
texto recebido: esta é a linha de numero 99998
texto recebido: esta é a linha de numero 99999
texto recebido: Transmissão finalizada!
Arquivo recebido!
Tempo gasto na transferência: 25.03 segundos
Selecione o que deseja fazer:
1. Receber um arquivo do servidor
2. Encerrar o programa
Digite o número da opção desejada:

```

Figura 22 - Tempo de execução da transmissão de um arquivo grande usando protocolo UDP, com o cliente e servidor rodando no mesmo dispositivo

Já para a transmissão em dispositivos diferentes, o código demorou 0.52 segundos para transferir um arquivo pequeno, 2.11 segundos para transferir um arquivo médio, e 25.03 segundos para transferir um arquivo grande, como visto nas figuras 20, 21 e 22.

Análise dos resultados e conclusão

Após realizarmos vários experimentos para medir os tempos de transmissão dos arquivos com tamanhos diferentes, constatamos que o TCP, apesar de ser mais lento em comparação com o UDP, oferece uma maior confiabilidade na entrega dos dados.

Os tempos de transmissão no TCP quando executamos o cliente e o servidor no mesmo dispositivo foram de 0.08 segundos para um arquivo pequeno, 0.71 segundos para um arquivo médio e 7.02 segundos para um arquivo grande. Já em dispositivos separados, os tempos foram de 0.12 segundos, 0.85 segundos e 8.50 segundos, respectivamente.

No UDP, o código foi extremamente similar ao do TCP. Utilizamos a mesma lógica e a mesma estratégia para realizar o envio dos arquivos. No entanto, enfrentamos alguns problemas devido à natureza não confiável do UDP. Com a sua velocidade superior de envio, é comum ocorrer perda de pacotes, algo preocupante quando se utiliza uma flag para indicar o fim da transmissão, uma vez que, se a flag for afetada por alguma perda, o sincronismo é completamente comprometido, fazendo com que o cliente não saiba que a transmissão já foi finalizada, entrando então em um loop infinito.

Para diminuir os erros, foi necessário aumentar o tempo de sleep após o envio de cada linha do arquivo, mas mesmo assim não conseguimos garantir uma transmissão totalmente livre de perdas e 100% confiável. Além disso, esse aumento impactou significativamente o tempo de execução do UDP, dificultando uma comparação precisa das velocidades de transmissão entre os dois protocolos.

Os tempos de transmissão no UDP, quando executado no mesmo dispositivo, foram os seguintes: 0.20 segundos para arquivos pequenos, 1.78 segundos para arquivos médios e

17.62 segundos para arquivos grandes. Já em dispositivos diferentes, os tempos foram de 0.52 segundos, 2.11 segundos e 25.03 segundos, respectivamente. Como dito, os tempos foram consideravelmente maiores do que no TCP devido ao aumento do sleep, mesmo o UDP tendo a vantagem de ser mais rápido.

Em resumo, a escolha entre UDP e TCP deve ser baseada nas necessidades específicas da aplicação. Se a velocidade é um fator crucial e a perda ocasional de pacotes é tolerável, o UDP pode ser uma opção viável. No entanto, se a confiabilidade e a integridade dos dados são prioridades, o TCP é a escolha mais indicada, apesar de seu desempenho ser um pouco mais lento.

Referências Bibliográficas

TANENBAUM, A. S. **Redes de Computadores**. 4^a Ed., Editora Campus (Elsevier), 2003.