Desempenho dos Protocolos TCP, UDP e UDP com garantia na Transferência de Arquivos

Alonso Lucca Fritz[1], Igor André Engler[1]

[1]Ciência da Computação – Universidade Estadual do Oeste do Paraná (UNIOESTE)
Caixa Postal 801 – 85.814-110 – Cascavel – PR – Brazil

alonso.fritz@unioeste.br, igor.engler@unioeste.br

Abstract. *This work aims to study and compare the performance of TCP and UDP protocols in case they are used in a File Transfer scenario. Considering that, for this case, the guarantee of delivery is fundamental, the TCP protocol would be the ideal choice for this case, for the use of UDP we will have two cases to be inplemented and compared, firstly, the conventional UDP that has speed as its main objective but does not have delivery guarantee and the UDP adapted with guarantee, thus guaranteeing the delivery of packages in the File Transfer scenario.*

Resumo. *Este trabalho tem como objetivo estudar e comparar o desempenho dos protocolos TCP e UDP para o caso de serem utilizados em um cenário de Transferência de Arquivos. Considerando que para este caso a garantia de entrega é fundamental o protocolo TCP seria a escolha ideal para este caso, para a utilização do UDP teremos dois casos a serem implementados e comparados, primeiramente o UDP convencional que tem como objetivo principal a velocidade porém não possui garantia de entrega e o UDP adaptado com garantia, dessa forma garantindo a entrega dos pacotes no cenário de Transferência de Arquivos.*

1. Introdução

O protocolo TCP é atualmente o mais utilizado na camada de transporte para aplicações WEB. O TCP provê confiabilidade, entrega na sequência correta e verificação de erros dos pacotes de dados, entre os diferentes nós de redes, para a camada de aplicação. Para manter a confiabilidade dos dados, o TCP utiliza o que chamamos de "aperto de mão" de três vias ou, three-way-handshake. Desse modo, o TCP é ideal para casos em que a confiabilidade dos dados é essencial, como quando se trata de transferência de arquivos, por exemplo.

O UDP, tem como característica essencial o desempenho/velocidade e falta de confiabilidade. O protocolo UDP não exige que a origem e o destino estabeleçam um handshake triplo antes que a transmissão ocorra. O protocolo UDP apenas envia os pacotes, o que significa que ele tem muito menos sobrecarga de largura de banda e latência. Com ele, os pacotes podem seguir caminhos diferentes entre o emissor e o receptor e, como resultado, alguns pacotes podem ser perdidos ou recebidos fora de ordem. O UDP é comumente usado em comunicações sensíveis ao tempo, em que, ocasionalmente, descartar pacotes é melhor do que esperar.

Caso garantias sejam necessárias, é preciso implementar uma série de estruturas de controle, tais como timeouts, retransmissões, acknowledgements, controle de fluxo, entre outros. Este é o caso atual, onde utilizaremos os protocolos TCP e UDP para realizar uma transferência de arquivos pela rede estabelecida.

O objetivo é portanto comparar o desempenho da utilização desses protocolos na transferência de arquivos, para isso teremos três implementações, uma do TCP convencional o qual já é o ideal para a tarefa proposta, uma do UDP convencional em que não possuímos nenhuma garantia de entrega de pacotes, é mais rápido porém ocorre perda de dados e por último a implementação do UDP com algum tipo de garantia para que os dados sejam realmente entregues completamente. O método em questão de garantia do UDP será do tipo para-e-espere, onde um pacote de dados é enviado, e então aguarda uma resposta de entrega com sucesso, caso isso não ocorra então o pacote é reenviado até receber a confirmação e só assim partir para o envio do próximo.

2. Protocolos TCP e UDP

O Protocolo de Controle de Transmissão, ou TCP, é um dos protocolos de comunicação, presente na cama de transporte da rede de computadores do modelo OSI, os quais dão suporte a rede global Internet, verificando se os dados são enviados corretamente e sem erros via rede.

Nesta camada de transporte é onde encontramos a maioria das aplicações WEB, como SSH, FTP, HTTP dessa forma podemos dizer que o TCP é o protocolo mais utilizado em aplicações WEB, devido a sua versatilidade e robustez já que ele garante confiabilidade, entrega de dados na sequência correta e verificação de erros dos pacotes de dados.

Mencionado a robustez do protocolo podemos listar algumas das características fundamentais do TCP. Ele é Orientado à conexão, ou seja, é necessário estabelecer-se uma conexão por meio de uma sequência de passos definida no protocolo, onde a aplicação envia um pedido de conexão para o destino e usa dessa "conexão" para transferir os dados; o Handshake é um mecanismo de estabelecimento (três etapas) e finalização (quatro etapas) de conexão, permitindo a autenticação e o encerramento de uma sessão completa, garantindo assim que ao final da conexão todos os pacotes foram recebidos apropriadamente; é Ponto-a-ponto, ou seja, uma conexão é estabelecida entre dois pontos; a Confiabilidade, utiliza-se de várias técnicas para proporcionar a entrega confiável de pacotes de dados, que em aplicações onde são sensíveis a entrega de dados ele possui grande vantagem em relação ao UDP, como por exemplo na transferência de arquivos, permite também a recuperação de dados corrompidos e pode recuperar a ligação em caso de problemas no sistema e na rede; é Full Duplex, onde é possível a transferência simultânea em ambas as direções (cliente-servidor / servidor-cliente) durante toda a sessão; Entrega ordenada, dessa forma a aplicação faz a entrega ao TCP de blocos de dados com um tamanho arbitrário num fluxo de dados, partindo esses dados em segmentos/pacotes de tamanho especifico, apesar da circulação dos pacotes ao longo da rede possa fazer com que os pacotes não cheguem ordenados o TCP garante a reconstrução do stream no destinatário mediante os números de sequência; possui Controle de fluxo em que o receptor, a medida que recebe os dados, envia mensagens ACK (Acknowledgement), confirmando a recepção de um segmento; e o Controle de Congestionamento, onde baseado no número de mensagens de reconhecimentos ACK (Acknowledgement), recebidos pelo remetente por unidade de tempo calculada com os dados do tempo de ida e de volta, ou RTT (Round Trip Travel), pode-se predizer o

quanto a rede esta congestionada diminuindo a sua taxa de transmissão de modo que o núcleo da rede nao se sobrecarregar.

O protocolo TCP durante a conexão especifica três fases: o estabelecimento da ligação, a transferência e o término da ligação. Como foi mencionado, o estabelecimento da ligação é feito em três etapas, enquanto o término é feito em quatro.

Para estabelecer uma conexão, e também a confiabilidade dos dados, o TCP utiliza o que chamamos de "aperto de mãos de três vias", o three way handshake, também conhecido como SYN, SYN+ACK, ACK. Para melhor esclarecimento dessas etapas, seguem suas descrições.

No SYN, a abertura ativa é realizada por meio do envio de um segmento com uma flag SYN pelo cliente ao servidor. O cliente define o número de sequência de segmento como um valor aleatório A.

Para o SYN+ACK, em resposta, o servidor responde com um segmento SYN+ACK. O número de reconhecimento (ACKnowledgement) é definido como sendo um a mais do que o número de sequência recebido (A+1), e o número de sequência que o servidor escolhe para o pacote é outro número aleatório B.

Finalmente para o ACK, o cliente envia um ACK de volta ao servidor. O número de sequência é definido ao valor de reconhecimento recebido, (A+1), e o número de reconhecimento definido como um a mais que o número de sequência recebido (B+1).

Neste ponto, o cliente e o servidor receberam um reconhecimento de conexão. As etapas 1 e 2 estabelecem o parâmetro (número de sequência) de conexão para uma direção e ele é reconhecido. As etapas 2 e 3 estabelecem o parâmetro de conexão (número de sequência) para a outra direção e ele é reconhecido. Com isto, uma comunicação full-duplex é estabelecida.

Durante a fase de transferência o TCP está equipado com vários mecanismos que asseguram a confiabilidade e robustez: números de sequência que garantem a entrega ordenada, código detector de erros (checksum) para detecção de falhas em segmentos específicos, confirmação de recepção e temporizadores que permitem o ajuste e contorno de eventuais atrasos e perdas de segmentos.

A fase de encerramento da sessão TCP é um processo de quatro fases, em que cada interlocutor responsabiliza-se pelo encerramento do seu lado da ligação. Quando um deles pretende finalizar a sessão, envia um pacote com a flag FIN ativa, ao qual deverá receber uma resposta ACK. Por sua vez, o outro interlocutor irá proceder da mesma forma, enviando um FIN ao qual deverá ser respondido um ACK.

O User Datagram Protocol, ou UDP, é um protocolo simples da camada de transporte que permite a aplicação enviar um datagrama encapsulado num pacote IPv4 ou IPv6 a um destino, porém diferente do TCP sem qualquer tipo de garantia de entrega ou a prova de erros.

O protocolo UDP é conhecido como veloz porém não confiável, como já mencionado, caso garantias sejam necessárias é preciso implementar adicionalmente uma série de estruturas de controle. Cada datagrama UDP tem um tamanho e pode ser considerado como um registro indivisível, diferentemente do TCP, que é um protocolo orientado a

fluxos de bytes sem início e sem fim. Podemos dizer também que o UDP é um serviço sem conexão, desde que não há necessidade de manter um relacionamento longo entre cliente-servidor, ou seja, um cliente UDP pode criar um socket, enviar um datagrama para um servidor e imediatamente enviar outro datagrama com o mesmo socket para um servidor diferente e reciprocamente, um servidor poderia ler datagramas vindos de diversos clientes, usando um único socket.

Como o UDP é um protocolo que não é voltado a conexão, basicamente não existe a necessidade do handshake que ocorre no TCP, ou seja, não é necessário estabelecer uma comunicação previamente.

O UDP faz a entrega de mensagens independentes, designadas por datagramas, entre aplicações ou processos, em sistemas host. A entrega pode ser feita fora de ordem e os datagramas podem ser perdidos. A integridade dos dados pode ser conferida por um "checksum" (um campo no cabeçalho de checagem por soma) baseado em complemento de um, de 16 bits.

A diferença básica entre o UDP e o TCP é o fato de que o TCP é um protocolo orientado à conexão e, portanto, inclui vários mecanismos para iniciar, manter e encerrar a comunicação, negociar tamanhos de pacotes, detectar e corrigir erros, evitar congestionamento do fluxo e permitir a retransmissão de pacotes corrompidos, independente da qualidade do meio físicos.

O UDP, por sua vez, é feito para transmitir dados pouco sensíveis, como fluxos de áudio e vídeo, ou para comunicação sem conexão, como é o caso da negociação DHCP ou tradução de endereços por DNS. Os dados são transmitidos apenas uma vez, incluindo apenas um frágil, e opcional, sistema de CRC de 16 bits. Os pacotes que chegam corrompidos são simplesmente descartados, sem que o emissor sequer saiba do problema. Por outro lado, a ausência de estruturas de controle complexas garante ao UDP alta eficiência, já que cada pacote é composto praticamente somente por dados.

Dadas essas informações pode-se questionar a utilização viável do protocolo UDP, porém o mesmo tem sim suas aplicações e vantagens com relação ao protocolo TCP, podemos citar:

O UDP é uma escolha adequada para fluxos de dados em tempo real, especialmente aqueles que admitem perda ou corrompimento de parte de seu conteúdo, tais como vídeos ou voz (VoIP). Aplicações sensíveis a atrasos na rede, mas poucos sensíveis a perdas de pacotes, como jogos de computadores, também podem se utilizar do UDP. As garantias de TCP envolvem retransmissão e espera de dados, como consequência, intensificam os efeitos de uma alta latência de rede; o UDP também suporta broadcasting e multicasting. Caso esses recursos sejam necessários, o UDP deverá necessariamente ser utilizado. Algum tratamento de erro pode ser adicionado, mas geralmente aplicações multicast também admitem perda de parte dos pacotes ou fazem retransmissões constantes; o UDP não perde tempo com criação ou destruição de conexões. Durante uma conexão, o UDP troca apenas 2 pacotes, enquanto no TCP esse número é superior a 10. Por isso, aplicações que encaixam-se num modelo de pergunta-resposta também são fortes candidatas a usar UDP. Entretanto, pode ser necessário implementar algoritmos de timeouts, acks e, no mínimo, retransmissão.

Nesse caso, vale lembrar que se a troca de mensagens for muito intensa, o protocolo TCP pode voltar a ser vantajoso, já que seu custo de conexão será amortizado.

Embora o processamento dos pacotes UDP seja realmente mais rápido, quando as garantias de confiabilidade e ordenação são necessárias, é pouco provável que uma implementação em UDP obterá resultados melhores do que o uso direto do TCP. Segundo essa afirmação, os testes de desempenho que foram realizados entre TCP, UDP e UDP com garantia, demonstraram de forma prática essas vantagens e desvantagens de implementação dos protocolos para a Transferência de Arquivos.

3. Implementação e Linguagem

A linguagem de programação Python v3.10.4 foi utilizada para a implementação dos protocolos.

3.1. UDP Sem Garantia

Por se tratar do protocolo UDP, o servidor não precisa de uma conexão. Portanto, é necessário apenas criar um socket e vinculá-lo à uma porta usando a função bind(), após isso o servidor aguarda pelos pacotes. A Imagem 1 mostra o código do server UDP.

```python
import socket, tqdm, select, os, sys
from datetime import datetime

SERVER_HOST = "0.0.0.0"
SERVER_PORT = 5001
TIMEOUT = 3
BUFFER =  int(sys.argv[1]) #500

def server_udp():
    blocks = 0

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind((SERVER_HOST, SERVER_PORT))

    while True:
        data, address = server_socket.recvfrom(BUFFER)

        print(f"{address} is connected.")
        if data:

            file_name, file_size = data.split(b"</>")
            file_name = os.path.basename(file_name)

            f = open(file_name, 'wb')

            file_size = int(file_size)

            progress_bar = tqdm.tqdm(range(file_size), f"Receiving {file_name}", unit="B", unit_scale=True, unit_divisor=1024)
            while True:
                ready = select.select([server_socket], [], [], TIMEOUT)
                if ready[0]:
                    data, address = server_socket.recvfrom(BUFFER)
                    f.write(data)
                    blocks = blocks + 1
                else:
                    f.close()
                    server_socket.close()
                    return blocks
                progress_bar.update(len(data))


if __name__ == "__main__":
    print(f"BLOCKS RECEIVED: {server_udp()}")

```

Imagem 1 - código do server UDP sem garantia de entrega

Para receber dados de um socket, é utilizado a função recvfrom(), que retorna uma tupla (string, address), onde string representa os dados e address é o endereço do socket que transmite os dados do cliente. O parâmetro BUFFER na função recvfrom() é utilizado para indicar a quantidade de bytes recebidos por pacotes.

O cliente UDP, exibido na Imagem 2, deve enviar mensagens para o servidor, sem se preocupar com confirmação de recebimento. Para isso, para realizar o envio de dados é utilizada a função sendto(), cujos parâmetros são a mensagem a ser enviada e endereço de destino.

```python
import socket, time, os, sys, tqdm
from datetime import datetime

HOST = "localhost"
PORT = 5001
FILENAME = sys.argv[1]
FILESIZE = os.path.getsize(FILENAME)
BUFFER = int(sys.argv[2])

def client_udp():
    try:
        blocks = 0
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.sendto(f"{FILENAME}</>{FILESIZE}".encode(), (HOST, PORT))

        f = open(FILENAME, "rb")
        data = f.read(BUFFER)

        progress_bar = tqdm.tqdm(range(FILESIZE), f"Sending {FILENAME}", unit="B", unit_scale=True, unit_divisor=1024)
        while(data):
            if(sock.sendto(data, (HOST, PORT))):
                data = f.read(BUFFER)
                progress_bar.update(len(data))
                blocks = blocks + 1
    finally:
        sock.close()
        f.close()
        return blocks

if __name__ == "__main__":
    tstart = datetime.now()
    print(f"BLOCKS GENERATED: {client_udp()}")
    tend = datetime.now()
    print (f"TIME ELAPSED: {(tend - tstart).total_seconds()*1000}")
```

Imagem 2 - código do cliente UDP sem garantia de entrega

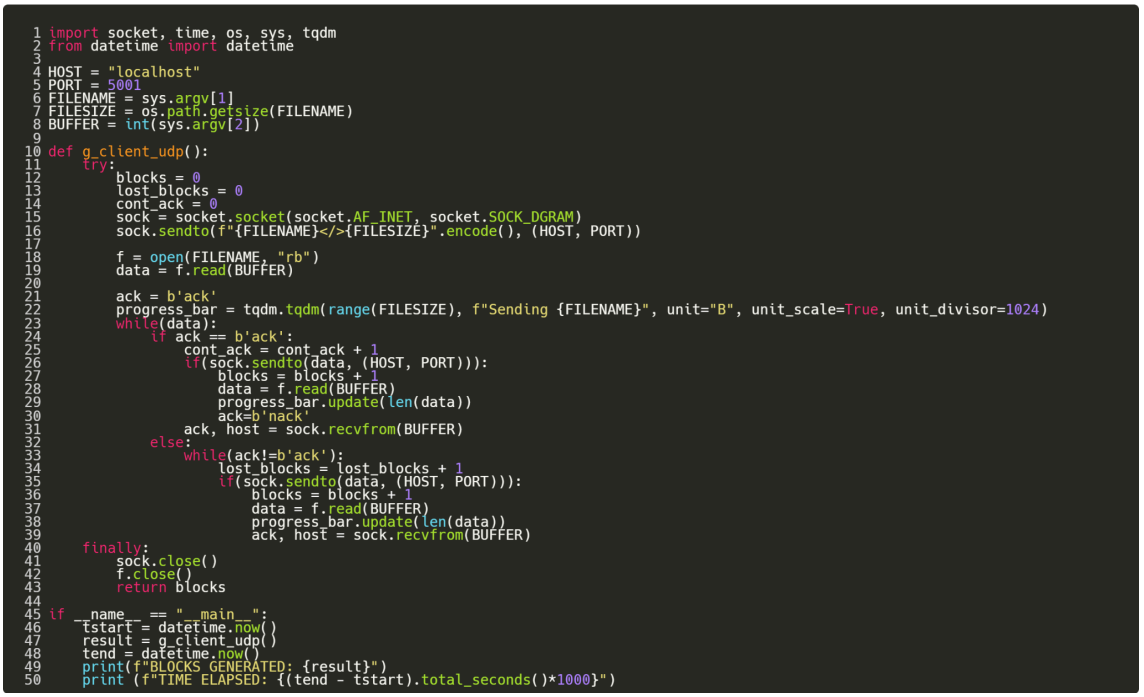
3.2. UDP Com Garantia

O código do UDP com garantia é o mesmo do UDP sem garantia, porém, nesse caso, ao receber um pacote do cliente, o servidor envia uma mensagem confirmando esse recebimento, essa mensagem é chamada de “ACK”. Este processo é mostrado na Imagem 3.

```python
import socket, tqdm, select, os, sys
from datetime import datetime

SERVER_HOST = "0.0.0.0"
SERVER_PORT = 5001
TIMEOUT = 3
BUFFER =  int(sys.argv[1])

def g_server_udp():
    blocks = 0

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind((SERVER_HOST, SERVER_PORT))

    while True:
        data, address = server_socket.recvfrom(BUFFER)

        print(f"Waiting as {SERVER_HOST}:{SERVER_PORT}")

        print(f"{address} is connected.")
        if data:

            file_name, file_size = data.split(b"</>")
            file_name = os.path.basename(file_name)

            f = open(file_name, 'wb')

            file_size = int(file_size)

            progress_bar = tqdm.tqdm(range(file_size), f"Receiving {file_name}", unit="B", unit_scale=True, unit_divisor=1024)
            while True:
                ready = select.select([server_socket], [], [], TIMEOUT)
                if ready[0]:
                    data, address = server_socket.recvfrom(BUFFER)
                    f.write(data)
                    blocks = blocks + 1
                    server_socket.sendto(b"ack", address)
                else:
                    server_socket.sendto(b"nack", address)
                    f.close()
                    server_socket.close()
                    return blocks
                progress_bar.update(len(data))


if __name__ == "__main__":
    print(f"BLOCKS RECEIVED: {g_server_udp()}")
```

Imagem 3 - Código do server UDP com garantia de entrega

Como mostrado na Imagem 4, o cliente deve esperar a mensagem para realizar um envio de um novo pacote e, caso não receba, tentará enviar o mesmo pacote novamente.



```python
import socket, time, os, sys, tqdm
from datetime import datetime

HOST = "localhost"
PORT = 5001
FILENAME = sys.argv[1]
FILESIZE = os.path.getsize(FILENAME)
BUFFER = int(sys.argv[2])

def g_client_udp():
    try:
        blocks = 0
        lost_blocks = 0
        cont_ack = 0
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.sendto(f"{FILENAME}</>{FILESIZE}".encode(), (HOST, PORT))

        f = open(FILENAME, "rb")
        data = f.read(BUFFER)

        ack = b'ack'
        progress_bar = tqdm.tqdm(range(FILESIZE), f"Sending {FILENAME}", unit="B", unit_scale=True, unit_divisor=1024)
        while(data):
            if ack == b'ack':
                cont_ack = cont_ack + 1
                if(sock.sendto(data, (HOST, PORT))):
                    blocks = blocks + 1
                    data = f.read(BUFFER)
                    progress_bar.update(len(data))
                    ack=b'nack'
                ack, host = sock.recvfrom(BUFFER)
            else:
                while(ack!=b'ack'):
                    lost_blocks = lost_blocks + 1
                    if(sock.sendto(data, (HOST, PORT))):
                        blocks = blocks + 1
                        data = f.read(BUFFER)
                        progress_bar.update(len(data))
                        ack, host = sock.recvfrom(BUFFER)
    finally:
        sock.close()
        f.close()
        return blocks

if __name__ == "__main__":
    tstart = datetime.now()
    result = g_client_udp()
    tend = datetime.now()
    print(f"BLOCKS GENERATED: {result}")
    print (f"TIME ELAPSED: {(tend - tstart).total_seconds()*1000}")
```

Imagem 4 - Código do cliente UDP com garantia de entrega

3.3. TCP

O protocolo TCP estabelece uma conexão entre os sockets cliente e servidor, antes do envio de qualquer mensagem de dados. É necessário que o servidor aceite a conexão vinda do cliente antes de começar a transmissão do arquivo. A função listen() deixa o socket em modo de espera, e a função accept() aceita a conexão, caso haja alguma. Esse processo é mostrado na Imagem 5

```python
from fileinput import filename
import socket, tqdm, os, sys
from datetime import datetime

SERVER_HOST = "0.0.0.0"
SERVER_PORT = 5001
BUFFER = int(sys.argv[1])

def server_tcp():
    try:
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.bind((SERVER_HOST, SERVER_PORT))

        server_socket.listen(5)

        client_socket, address = server_socket.accept()

        print(f"{address} is connected.")

        received = client_socket.recv(BUFFER).decode()
        filename, filesize = received.split("</>")
        filename = os.path.basename(filename)
        filesize = int(filesize)

        progress_bar = tqdm.tqdm(range(filesize), f"Receiving {filename}", unit="B", unit_scale=True, unit_divisor=1024)
        with open(filename, "wb") as f:
            while True:
                data = client_socket.recv(BUFFER)
                if not data:
                    break
                f.write(data)
                progress_bar.update(len(data))

    finally:
        client_socket.close()
        server_socket.close()
        return

if __name__ == "__main__":
    server_tcp()
```

Imagem 5 - código do server TCP

Como mostrado na Imagem 6, é necessário que o cliente TCP se conecte ao servidor, para isso ele utiliza a função connect(), cujo parâmetro é o endereço do servidor. A função sendall() é utilizada para enviar os dados, e os mesmos são recebidos com a função recv().

```python
import socket, tqdm, os, sys
from datetime import datetime

HOST = "localhost"
PORT = 5001
FILENAME = sys.argv[1]
FILESIZE = os.path.getsize(FILENAME)
BUFFER = int(sys.argv[2])


def client_tcp():
    try:
        blocks = 0
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((HOST, PORT))
        client_socket.send(f"{FILENAME}</>{FILESIZE}".encode())

        progress_bar = tqdm.tqdm(range(FILESIZE), f"Sending {FILENAME}", unit="B", unit_scale=True, unit_divisor=1024)
        with open(FILENAME, "rb") as f:
            while True:
                data = f.read(BUFFER)
                if not data:
                    break

                client_socket.sendall(data)

                progress_bar.update(len(data))
                blocks = blocks + 1
    finally:
        client_socket.close()
        return blocks

if __name__ == "__main__":
    tstart = datetime.now()
    print(f"BLOCKS GENERATED: {client_tcp()}")
    tend = datetime.now()
    print (f"TIME ELAPSED: {(tend - tstart).total_seconds()*1000}")
```

Imagem 6 - código do cliente TCP

4. Resultados

Os testes foram realizados via conexão cabeada entre dois computadores com as seguintes conexões:

PC 01 - Cliente:

- Adaptador Ethernet: Realtek PCIe GbE Family Controller
- Processador: Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz
- RAM: 8GB DDR3

PC 02 - Server:

- Adaptador Ethernet: Realtek PCIe GbE Family Controller
- Processador: Intel(R) Core(TM) i3-7100 CPU @ 3.90GHz
- RAM: 8GB DDR4

Foram realizados dez testes para três tamanhos de buffer (100 bytes, 500 bytes e 1000 bytes) em cada um dos algoritmos, totalizando noventa testes. O arquivo enviado do cliente para o servidor é um zip de 107,303 MB.

A Tabela 1 mostra os testes realizados no algoritmo TCP, com o tempo de cada teste, e quantidade de blocos gerados com o tamanho de buffer utilizado.

TCP		
Buffer 1000	Buffer 500	Buffer 100

tempo (milissegundos)	blocos gerados	tempo (milissegundos)	blocos gerados	tempo (milissegundos)	blocos gerados
1391,782	109888	1572,563	219776	4129,143	1098878
1374,202		1436,279		3858,343	
1400,739		1493,629		3868,133	
1399,86		1452,43		3908,909	
1394,451		1417,213		3801,073	
1388,342		1428,679		3749,966	
1386,07		1550,372		3774,699	
1384,932		1492,111		3865,7	
1381,072		1439,862		3821,498	
1392,169		1457,3		3802,864	

Tabela 1 - testes realizados com o algoritmo TCP

Na Tabela 2, além do tempo de execução e blocos gerados, também foi calculada a quantidade de blocos recebidos para o UDP sem garantia de entrega. Em todos os casos o arquivo chegou corrompido, já que nem todos os blocos enviados foram recebidos

UDP sem garantia								
Buffer 1000			Buffer 500			Buffer 100		
tempo (milissegundos)	blocos enviados	blocos recebidos	tempo (milissegundos)	blocos enviados	blocos recebidos	tempo (milissegundos)	blocos enviados	blocos recebidos
1569,94	109888	104793	2794,227	219776	216234	11612,336	1098878	1093882
1557,33		106937	2869,051		219328	11167,848		1098823
1490,33		105803	3120,769		215836	11016,45		1095376
1578,98		105311	3036,656		216028	10805,528		1094898
1546,15		105357	2829,362		216837	10922,52		1095773
1578,92		108299	2795		215442	11035,61		1095798
1631,28		106055	2765,54		216327	10828,638		1096265
1663,3		105925	2591,25		217029	10432,237		1095780
1576,69		105236	2896,836		216249	10798,089		1096467
1654,39		107334	2768,364		216771	11044,712		1096337

Tabela 2 - testes realizados com o algoritmo UDP sem garantia de entrega

Semelhante aos outros casos, a Tabela 3 mostra os testes realizados para o UDP com garantia de entrega. A diferença mais notável nesse caso é o grande aumento no tempo de execução do código.

UDP com garantia					
Buffer 1000		Buffer 500		Buffer 100	
tempo	blocos enviados	tempo	blocos enviados	tempo	blocos enviados
48743,797	109888	93733,705	219776	455971,485	1098878
48544,551		94919,376		454427,019	
48338,887		94188,86		458442,522	
47998,1		94869,696		454699,678	
49546,27		93612,901		460234,036	
49188,737		94572,549		456489,831	
49412,132		94222,181		456609,215	
48523,6		93904,2		453214,822	
48778,558		94533,352		454201,754	
49154,212		94827,804		490467,085	

Tabela 3 - testes realizados com o algoritmo UDP com garantia de entrega

As médias do tempo de execução dos algoritmos são exibidas na Tabela 4, enquanto a Tabela 5 apresenta o desvio padrão em relação à média para o conjunto de testes.

Média de tempo em milissegundos			
Tamanho de buffer	1000	500	100
TCP	1389,36	1474,04	3858,03
UDP	1584,73	2846,71	10966,4
UDP/G	48822,88	94338,46	459475,74

Tabela 4 - Média de tempo em milissegundos dos testes realizados

Desvio Padrão em relação a média de tempo			
Tamanho de buffer	1000	500	100
TCP	8,23	52,57	106,72
UDP	52,38	148,29	303,99
UDP/G	494,85	479,42	11089,7

Tabela 5 - Desvio padrão em relação à média do conjunto de testes

Após obter os dados de teste, é possível calcular o intervalo de confiança de 95% considerando que $c1 \leq u \leq c2 = 0,95$, onde o intervalo (c1,c2) é chamado de intervalo de confiança da média da população. A tabela 6 mostra o cálculo de cada intervalo.

Intervalo de confiança						
Tamanho de buffer	1000		500		100	
Intervalo	C1	C2	C1	C2	C1	C2
TCP	1383,476207	1395,247593	1436,440575	1511,647005	3781,697687	3934,368
UDP	1547,261396	1622,198784	2740,630971	2952,780029	10748,95116	11183,84
UDP/G	48468,91303	49176,85577	93995,53243	94681,39237	451543,2031	467408,3

Tabela 6 - Intervalo de confiança de 95% da média

5. Conclusão

Com base nos dados obtidos, foi concluído que, apesar de tempo de execução semelhantes, o TCP leva vantagens para o envio de arquivos grandes em comparação ao UDP, já que o último possivelmente sempre terá perda de pacotes, levando à entrega de arquivos corrompidos. Quanto ao UDP com garantia de entrega, apesar de enviar e receber o arquivo em sua totalidade, o tempo de execução é significantemente maior que dos outros algoritmos estudados.

Referencias

Vinton G. Cerf, Robert E. Kahn (1974). A Protocol for Packet Network Intercommunication, IEEE Transactions on Communications, Vol. 22, No. 5, pp. 637-648

Orestes, Yan (2019) “O que é TCP, UDP e quais as diferenças?”, https://www.alura.com.br/artigos/quais-as-diferencas-entre-o-tcp-e-o-udp.

Seabra, Giulianna (2021) “O que é UDP e quais as diferenças com o TCP?”, https://blog.betrybe.com/desenvolvimento-web/udp-diferencas-tcp/#:~:text=O%20protocolo%20de%20datagramas%20do,conex%C3%A3o%20de%20ponta%20a%20ponta.

Postel, J. (1980) “RFC: User Datagram Protocol”, https://datatracker.ietf.org/doc/html/rfc768

Defense Advanced Research Projects Agency Information Processing Techniques Office 1400 Wilson Boulevard, Arlington, Virginia 22209 (1981), RFC: 793, “TRANSMISSION CONTROL PROTOCOL”, https://datatracker.ietf.org/doc/html/rfc793

KUROSE, J. F. e ROSS, K. W. Redes de Computadores e a Internet: uma abordagem top-down. 6 ed., São Paulo: Pearson, 2013.

E. Ferreira, Rubem. «23». In: Novatec. Linux:guia do administrador do sistema. Janeiro de 2013 2ª ed. São Paulo: [s.n.] ISBN 9788575221778

Marques, Eduardo. "PROJETO DE UMA REDE LOCAL DE COMPUTADORES DE ALTA VELOCIDADE"

Baset, S. A.; Schulzrinne, H. G. (abril de 2006). "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol"

"Portas TCP e UDP usadas por produtos de software da Apple". Apple Support.

"Discord Developer Portal — API Docs for Bots and Developers". Discord Developer Portal.

Ibrahim, Issam (2011) "Conjunto de Protocolos TCP/IP e suas falhas", https://repositorio.utfpr.edu.br/jspui/bitstream/1/19988/2/CT_TELEINFO_XIX_2011_12.pdf