

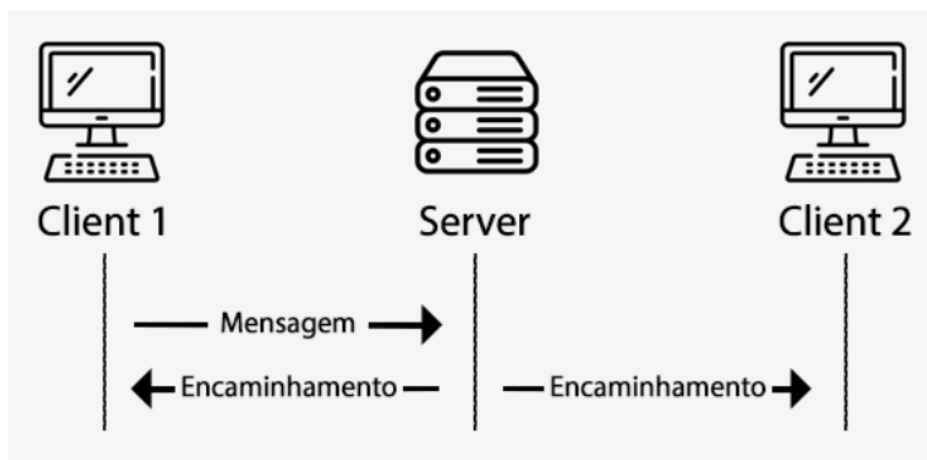
# Trabalho 2 - Lucas Munhoz

---

## Introdução

Este projeto consiste na implementação de um sistema cliente-servidor utilizando os protocolos **TCP** e **UDP**, desenvolvido em Python. Ele permite que clientes troquem mensagens via servidor e ainda inclui um sistema de monitoramento de latência, que mede o tempo de ida e volta entre cliente e servidor UDP para avaliar a qualidade da conexão e informar ao administrador, via e-mail, se a conexão está lenta, normal ou indisponível.

O sistema segue uma arquitetura cliente-servidor, onde o servidor atua como intermediário, recebendo e distribuindo mensagens entre os clientes conectados. Além disso, o monitor de latência utiliza UDP para enviar pings e medir a qualidade da conexão.



## Comparação entre TCP e UDP

Aspecto	TCP	UDP
Confiabilidade	Garante entrega de dados.	Não garante entrega.
Velocidade	Mais lento devido à confiabilidade.	Mais rápido.
Repasso de mensagens	Ordenado e com confirmação.	Apenas transmite pacotes.
Uso no projeto	Chat confiável entre clientes.	Chat rápido entre clientes.

## Servidor TCP: Repasse de Mensagens

```
Servidor TCP aguardando conexões...
Cliente conectado: ('127.0.0.1', 61534)
Cliente conectado: ('127.0.0.1', 61536)
('127.0.0.1', 61536) disse: Olá, tudo bem?
('127.0.0.1', 61534) disse: Olá! Tudo, e com você?
```

Python

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(("localhost", 12345))
server_socket.listen()
print("Servidor TCP aguardando conexões...")

while True:
    conn, addr = server_socket.accept() # Aceita uma conexão
    clients.append(conn)
    thread = threading.Thread(target=handle_client, args=(conn, addr, clients))
    thread.start() # Inicia a thread para o client
```

O servidor é configurado para escutar conexões na porta 12345, entrando em um loop infinito para aceitar novos clientes. Quando um client se conecta ao servidor, o socket é adicionado à lista clients e uma nova thread é criada para gerenciar a comunicação com esse cliente, permitindo que mais clientes se conectem ao servidor de forma simultânea.

Python

```
message = conn.recv(1024).decode() # Recebe mensagens do client
if not message or message == "/sair":
    print(f"Cliente desconectado: {addr}")
    clients.remove(conn)
    conn.close()
    break

print(f"{addr} disse: {message}")

# Repassa a mensagem para os demais clients
for client in clients:
    if client != conn:
        client.send(f"{addr} disse: {message}".encode())
```

O client pode enviar mensagens ao servidor, que repassa as mensagens para todos os outros clientes conectados. Quando um cliente envia a mensagem “/sair” ou desconecta, o servidor o remove da lista de clients e encerra a thread associada a este cliente.

## Client TCP: Envio e recebimento de mensagens

```
Conectado ao servidor.  
( '127.0.0.1', 61536) disse: Olá, tudo bem?  
Olá! Tudo, e com você?  
█
```

```
Conectado ao servidor.  
Olá, tudo bem?  
( '127.0.0.1', 61534) disse: Olá! Tudo, e com você?  
█
```

Python

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
client_socket.connect(("localhost", 12345))  
print("Conectado ao servidor.")
```

```
while True:  
    message = input()  
    if message == "/sair":  
        client_socket.close()  
        exit()  
    client_socket.send(message.encode())
```

O socket do client é criado usando protocolo TCP e se conecta ao servidor no endereço localhost e porta 12345. Através de um loop infinito o usuário pode inserir mensagens, que são codificadas e enviadas ao servidor.

Python

```
thread = threading.Thread(target=receive_messages, args=(client_socket,))  
thread.start()
```

```
def receive_messages(client_socket):  
    while True:  
        try:  
            message = client_socket.recv(1024).decode() # Recebe mensagens do  
servidor  
            print(message)  
        except:  
            print("Desconectado do servidor.")  
            break
```

Para receber mensagens, uma thread separada é iniciada, evitando que o client fique bloqueado enquanto aguarda mensagens do servidor. As mensagens recebidas são codificadas para texto e exibidas no terminal. Caso a conexão seja interrompida, é exibida uma mensagem.

## Servidor UDP: Repasse de mensagens

```
Servidor UDP aguardando mensagens...
Mensagem recebida de ('127.0.0.1', 8161)
Novo participante no chat.
Mensagem recebida de ('127.0.0.1', 8796)
Novo participante no chat.
Enviando mensagem para ('127.0.0.1', 8161)
Mensagem recebida de ('127.0.0.1', 8796)
Olá, tudo bem?
Enviando mensagem para ('127.0.0.1', 8161)
Mensagem recebida de ('127.0.0.1', 8161)
Tudo bem, e com você?
Enviando mensagem para ('127.0.0.1', 8796)
Mensagem recebida de ('127.0.0.1', 8796)
Também
Enviando mensagem para ('127.0.0.1', 8161)
[]
```

Python

```
def receive():
    while True:
        try:
            message, addr = server_socket.recvfrom(1024)
            message_queue.put((message, addr))
            print(f"Mensagem recebida de {addr}")
        except:
            print("Erro ao receber mensagem.")

message_queue = queue.Queue() # Fila de encaminhamento de mensagens
clients = set()

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(("localhost", 12345))
print("Servidor UDP aguardando mensagens...")

t1 = threading.Thread(target=receive)
t1.start()
```

O servidor é configurado para escutar no endereço localhost e na porta 12345. As mensagens recebidas pelo client são armazenadas em uma fila, juntamente com o endereço do remetente. Quando um novo client envia uma mensagem, seu endereço é armazenado em uma lista,

permitindo ao servidor saber para quais endereços deve encaminhar as mensagens recebidas. A função de recebimento de mensagens roda em uma thread separada para que o servidor esteja sempre pronto para receber os dados.

```
Python
def broadcast():
    while True:
        while not message_queue.empty():
            message, addr = message_queue.get()
            print(message.decode())

            if addr not in clients:
                clients.add(addr)

            # Se a mensagem for "ping", responde para o remetente
            if message.decode() == "ping":
                server_socket.sendto(b"ping", addr)
                break

            for client in clients:
                if client != addr:
                    try:
                        server_socket.sendto(f"{addr} disse: {message.decode()}".encode(), client)
                        print(f"Enviando mensagem para {client}")
                    except:
                        clients.remove(client)

t2 = threading.Thread(target=broadcast)
t2.start()
```

Uma outra thread analisa, encaminha e apaga as mensagens da fila. Primeiro, é verificado se trata-se de uma mensagem de ping. Se positivo, é encaminhada uma resposta apenas ao endereço remetente da mensagem. Essa ação é necessária para permitir que o monitor de latência funcione corretamente. Caso não seja uma mensagem de ping, a mensagem é encaminhada para todos os endereços da lista de clients, menos para o do próprio remetente.

## Client UDP: Envio e recebimento de mensagens

```

('127.0.0.1', 8796) disse: Novo participante no chat
.
('127.0.0.1', 8796) disse: Olá, tudo bem?
Tudo bem, e com você?
('127.0.0.1', 8796) disse: Também
[]

Olá, tudo bem?
('127.0.0.1', 8161) disse: Tudo bem, e com você?
Também
[]

```

Python

```

client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client.bind(("localhost", random.randint(8000, 9000)))

def receive_messages():
    while True:
        try:
            message, _ = client.recvfrom(1024)
            message = message.decode()
            print(message)
        except:
            print("Erro ao receber mensagem.")
            break

t1 = threading.Thread(target=receive_messages)
t1.start()

client.sendto("Novo participante no chat.".encode(), ("localhost", 12345))

while True:
    message = input()
    if message == "/sair":
        exit()
    client.sendto(message.encode(), ("localhost", 12345))

```

O socket do client é configurado para comunicação via UDP e associado a uma porta aleatória entre 8000 e 9000 no localhost, para que múltiplos clients possam ser executados simultaneamente no mesmo dispositivo. Em seguida, é enviada uma mensagem inicial ao servidor, para que o endereço do client seja adicionado à lista de endereços do servidor e passe a receber as mensagens encaminhadas. Uma thread é iniciada para receber as mensagens do servidor, permitindo que o client continue enviando mensagens enquanto escuta o servidor. As mensagens digitadas pelo usuário são codificadas e enviadas ao servidor.

## Monitoramento de Latência (UDP)

```
PS C:\apps\chat-network-monitor> python .\udp_server.py
Servidor UDP aguardando mensagens...
Mensagem recebida de ('127.0.0.1', 51129)
ping
Mensagem recebida de ('127.0.0.1', 51129)
ping
[]

PS C:\apps\chat-network-monitor> python .\latency_monitor.py
Para testes, utilize o Ethereum Mail para enviar e-mails.
Digite o endereço de e-mail: norberto.upton@ethereal.email
Digite a senha: 7EcckgPa4wVbQk8QvG
Conexão Normal. Latência: 8.00 ms
E-mail de conexão enviado: Normal
Conexão Normal. Latência: 7.09 ms
E-mail de conexão enviado: Normal
```

O monitor de latência verifica a qualidade da conexão com o servidor UDP, medindo o tempo de ida e volta de uma mensagem. O monitor envia um alerta por e-mail informando a latência ou se a conexão está inativa.

Os parâmetros iniciais do monitoramento são os seguintes:

```
Python
TARGET_IP = "127.0.0.1" # Determina que o destino que será monitorado é o
localhost;
TARGET_PORT = 12345      # Determina que a porta monitorada é a 12345;
TIMEOUT = 2              # Define que o monitor poderá esperar, no máximo, 2
segundos pela mensagem de resposta;
INTERVAL = 10            # Define que uma nova medição será realizada a cada 10
segundos;
LATENCY_NORMAL = 100     # Define como normal uma conexão com latência menor do
que 100ms;
LATENCY_SLOW = 300       # Define como lenta uma conexão com latência entre 100
e 300ms;
```

Para que seja possível testar o envio dos alertas por e-mail, é necessário utilizar uma conta temporária do [Ethereal Email](#). Ao executar o programa, deve-se inserir o endereço de e-mail criado no Ethereum e a senha temporária disponibilizada. O código já está configurado com o SMTP e portas do Ethereum.

## Account credentials

Account details generated using [Faker.js](#)

**NB!** these credentials are shown only once. If you do not write these down then you have to create a new account.

Name	Norberto Upton
Username	norberto.upton@ethereal.email – this account can not be used for inbound emails ⓘ
Password	7EcckgPa4wVbQk8QvG

```
Python
def monitor_latency():
```

```

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as udp_socket:
    udp_socket.settimeout(TIMEOUT)
    while True:
        try:
            start_time = time.time()
            udp_socket.sendto(b"ping", (TARGET_IP, TARGET_PORT))
            udp_socket.recvfrom(1024)
            latency = (time.time() - start_time) * 1000 # Em milissegundos

            if latency < LATENCY_NORMAL:
                print(f"Conexão Normal. Latência: {latency:.2f} ms")
                send_email("Normal", latency)
            elif latency < LATENCY_SLOW:
                print(f"Conexão Lenta. Latência: {latency:.2f} ms")
                send_email("Lenta", latency)
            else:
                print(f"Conexão Muito Lenta. Latência: {latency:.2f} ms")
                send_email("Muito Lenta", latency)
        except socket.timeout:
            print("Conexão Inativa.")
            send_email("Inativa", "Sem Resposta")

    time.sleep(INTERVAL)

```

Após a configuração do e-mail, o monitor se conecta ao servidor UDP, envia uma mensagem de ping e aguarda a resposta, medindo a latência em milissegundos e classificando a conexão como normal, lenta, muito lenta ou inativa.

Python

```

def send_email(status, latency):
    subject = f"Alerta de latência: {status}"
    body = f"A conexão ao {TARGET_IP}:{TARGET_PORT} está {status}.\nLatência: {latency} ms"

    message = MIMEMultipart()
    message["From"] = EMAIL_FROM
    message["To"] = EMAIL_TO
    message["Subject"] = subject
    message.attach(MIMEText(body, "plain"))

    try:
        with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as server:

```



```

server.starttls()
server.login(EMAIL_FROM, EMAIL_PASSWORD)
server.sendmail(EMAIL_FROM, EMAIL_TO, message.as_string())
print(f"E-mail de conexão enviado: {status}")
except Exception as e:
    print(f"Não foi possível enviar o e-mail: {e}")

```

O status é exibido no terminal e enviado via e-mail ao destinatário configurado no código. Após cada medição, o monitor espera pelo tempo configurado e repete o processo.

**Subject:** Alerta de latência: Normal

**From:** <norberto.upton@ethereal.email>

**To:** <nome@email.com>

**Time:** Today at 17:13

**Message-ID:** <ed88f8a0-b0e9-11ef-9739-c72b7fcf243a@wildduck.email>

☒ HTML ☐ Plaintext

A conexão ao 127.0.0.1:12345 está Normal.  
Latência: 15.6402587890625 ms

## Captura de pacotes

### Troca de pacotes no Chat TCP

Dois client conectam ao servidor e um dele envia uma mensagem:

```

PS C:\apps\chat-network-monitor> python .\tcp_server.py
Servidor TCP aguardando conexões...
Cliente conectado: ('127.0.0.1', 62513)
Cliente conectado: ('127.0.0.1', 62518)
('127.0.0.1', 62518) disse: Olá!

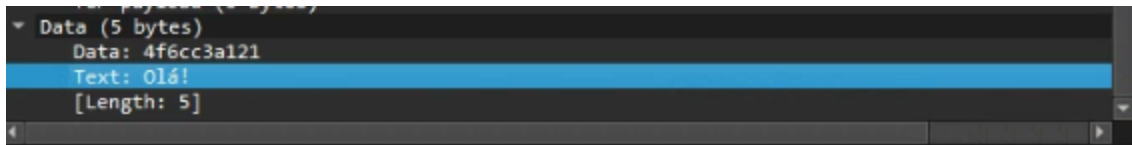
PS C:\apps\chat-network-monitor> python .\tcp_client.py
Conectado ao servidor.
('127.0.0.1', 62518) disse: Olá!

PS C:\apps\chat-network-monitor> python .\tcp_client.py
Conectado ao servidor.
Olá!

```

No.	Time	Source	Destination	Protocol	Length	Info
20	56.162594	127.0.0.1	127.0.0.1	TCP	56	62513 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
21	56.162801	127.0.0.1	127.0.0.1	TCP	56	12345 → 62513 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
22	56.162869	127.0.0.1	127.0.0.1	TCP	44	62513 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
133	221.042624	127.0.0.1	127.0.0.1	TCP	56	62518 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
134	221.042771	127.0.0.1	127.0.0.1	TCP	56	12345 → 62518 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
135	221.042852	127.0.0.1	127.0.0.1	TCP	44	62518 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
136	239.640217	127.0.0.1	127.0.0.1	TCP	49	62518 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=5
137	239.640296	127.0.0.1	127.0.0.1	TCP	44	12345 → 62518 [ACK] Seq=1 Ack=6 Win=2619648 Len=0
138	239.641349	127.0.0.1	127.0.0.1	TCP	77	12345 → 62513 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=33
139	239.641411	127.0.0.1	127.0.0.1	TCP	44	62513 → 12345 [ACK] Seq=1 Ack=34 Win=2619648 Len=0

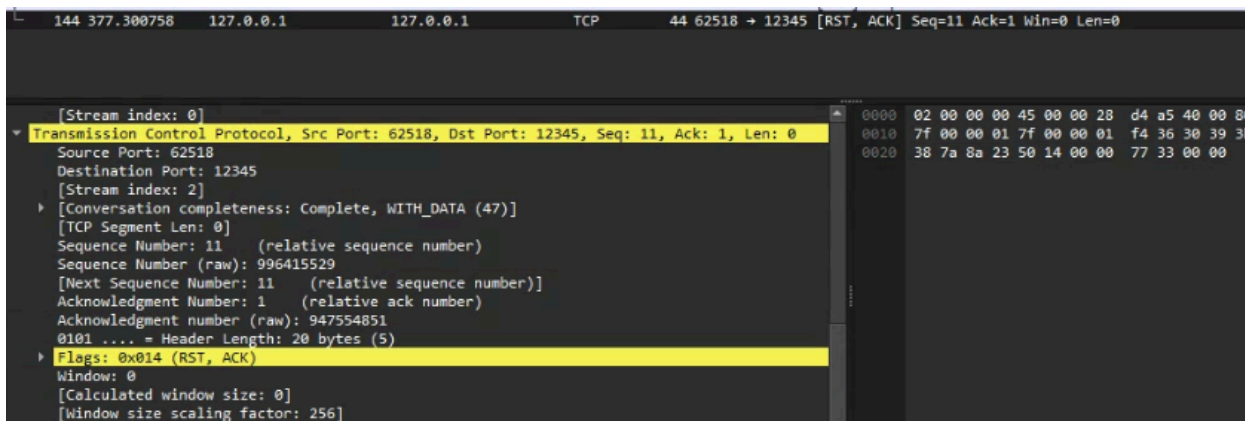
Dados da mensagem enviada:



Envio da mensagem /sair para desconectar do servidor:

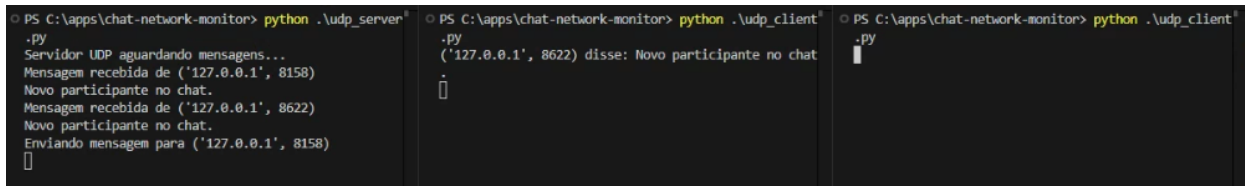
```
/sair
Desconectado do servidor.
```

Flag de fechamento da conexão:



## Troca de pacotes no Chat UDP

Dois client entram conectam-se ao servidor e enviam a mensagem de conexão:



Mensagem do segundo client é redirecionada ao primeiro pelo servidor:

1	0.000000	127.0.0.1	127.0.0.1	UDP	58 8158 → 12345 Len=26
2	2.391632	127.0.0.1	127.0.0.1	UDP	58 8622 → 12345 Len=26
3	2.393917	127.0.0.1	127.0.0.1	UDP	85 12345 → 8158 Len=53

```
▼ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 81
    Identification: 0xd4bc (54460)
  ▶ 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
    [Stream index: 0]
  ▶ User Datagram Protocol, Src Port: 12345, Dst Port: 8158
  ▼ Data (53 bytes)
    Data: 28273132372e302e31272c2038363232292064697373653a204e6f76662070617274696369706169
    Text: ('127.0.0.1', 8622) disse: Novo participante no chat.
    [Length: 53]
```

Um dos client é encerrado e uma mensagem é enviada pelo outro client:

```
PS C:\apps\chat-network-monitor> python .\udp_server.py
Servidor UDP aguardando mensagens...
Mensagem recebida de ('127.0.0.1', 8389)
Novo participante no chat.
Mensagem recebida de ('127.0.0.1', 8426)
Novo participante no chat.
Enviando mensagem para ('127.0.0.1', 8389)
Mensagem recebida de ('127.0.0.1', 8389)
Olá!
Enviando mensagem para ('127.0.0.1', 8426)
Erro ao receber mensagem.
█

PS C:\apps\chat-network-monitor> python .\udp_client.py
('127.0.0.1', 8426) disse: Novo participante no chat.
Olá!
█
```

Client encerrado não pode ser alcançado:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	58	8158 → 12345 Len=26
2	2.391632	127.0.0.1	127.0.0.1	UDP	58	8622 → 12345 Len=26
3	2.393917	127.0.0.1	127.0.0.1	UDP	85	12345 → 8158 Len=53
4	150.035406	127.0.0.1	127.0.0.1	UDP	58	Novo participante no chat.
5	150.043605	127.0.0.1	127.0.0.1	UDP	85	('127.0.0.1', 8400) disse: Novo participante no chat.
6	150.043663	127.0.0.1	127.0.0.1	ICMP	113	Destination unreachable (Port unreachable)
7	150.044082	127.0.0.1	127.0.0.1	UDP	85	('127.0.0.1', 8400) disse: Novo participante no chat.
8	150.044117	127.0.0.1	127.0.0.1	ICMP	113	Destination unreachable (Port unreachable)