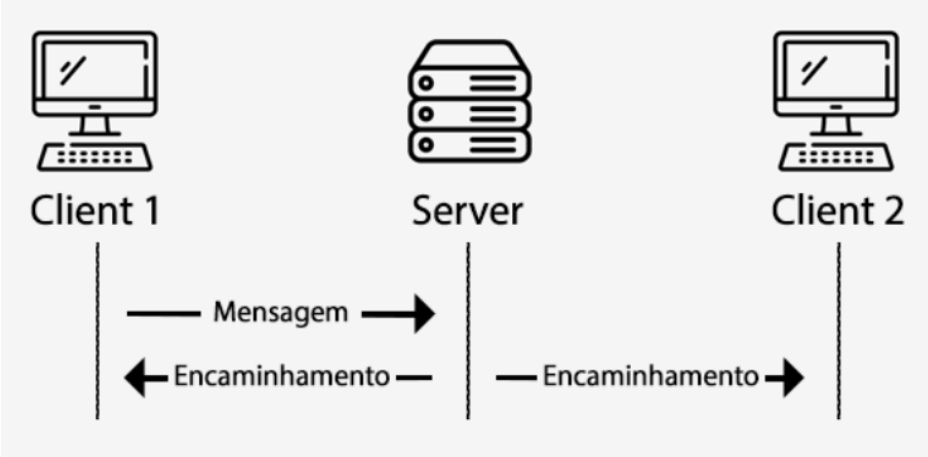
Trabalho 2 - Lucas Munhoz

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.

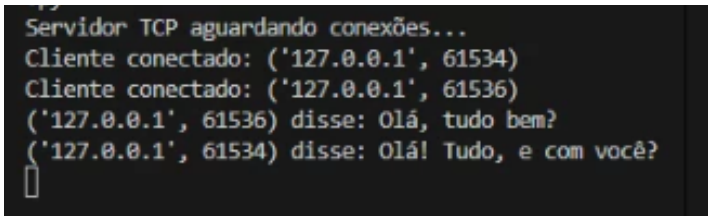
O sistema segue uma arquitetura cliente-servidor, onde o servidor atua como intermediário, recebendo e distribuindo mensagens entre os clientes conectados. Além disso, o monitor de latência utiliza UDP para enviar pings e medir a qualidade da conexão.



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



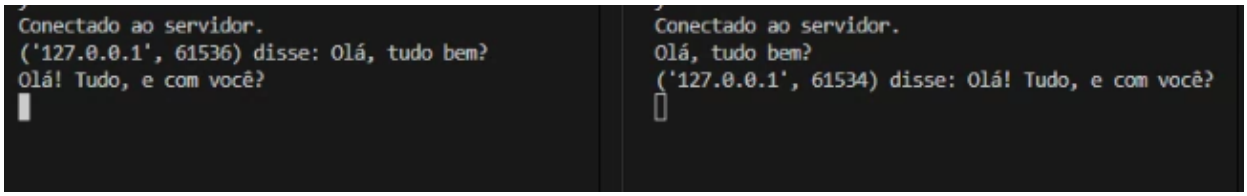

```python
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(("localhost", 12345))
server_socket.listen()
print("Servidor TCP aguardando conexões...")

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



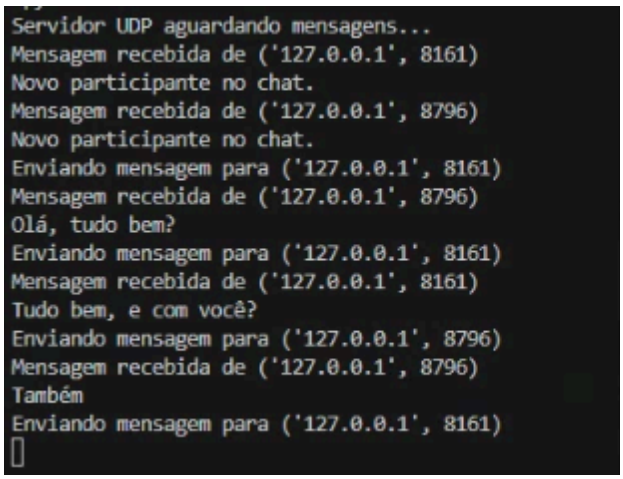

```python
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(("localhost", 12345))
print("Conectado ao servidor.")

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 infitino 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



Python

```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()
```

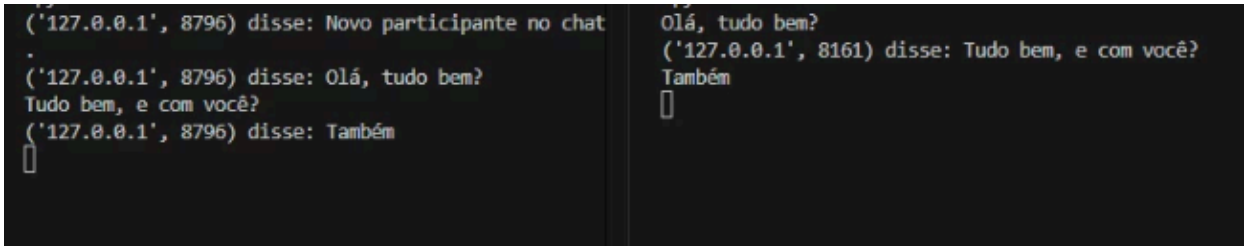
O servidor é configurado para escutar no endereço localhost e na porta 12345. As mensagens recebidas pelo client são armazenadas em uma fila, juntamente com o endereço do remetente. Quando um novo client envia uma mensagem, seu endereço é armazenado em uma lista,

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.

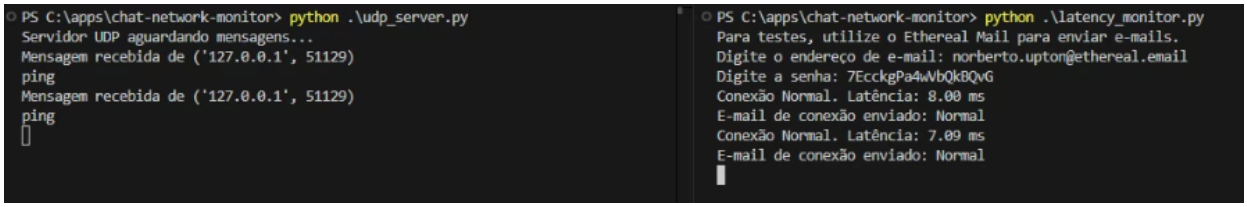
Client UDP: Envio e recebimento de mensagens




```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 multiplos 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)



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 Ethereal e a senha temporária disponibilizada. O código já está configurado com o SMTP e portas do Ethereal.

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	7EcckgPa4wVbQkBQvG

```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)
```

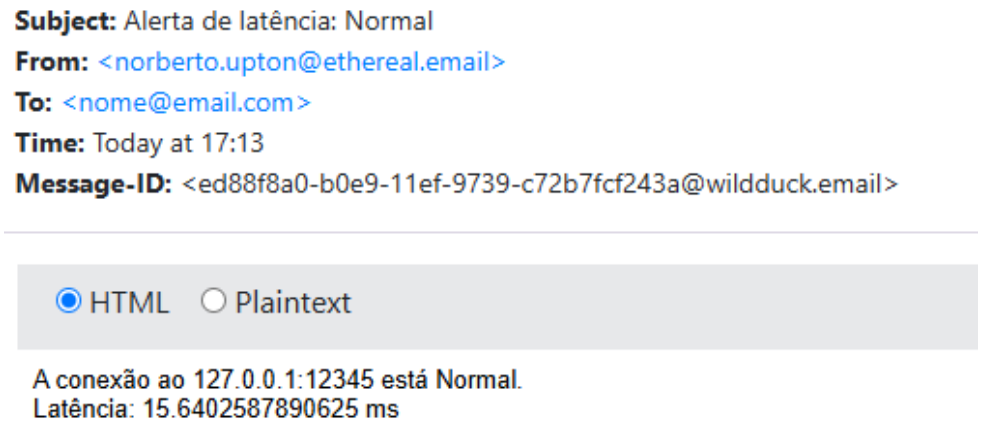
Após a configuração do e-mail, o monitor se conecta ao servidor UDP, envia uma mensagem de ping e aguarda a resposta, medindo a latência em milissegundos e classificando a conexão como normal, lenta, muito lenta ou inativa.

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}")
```

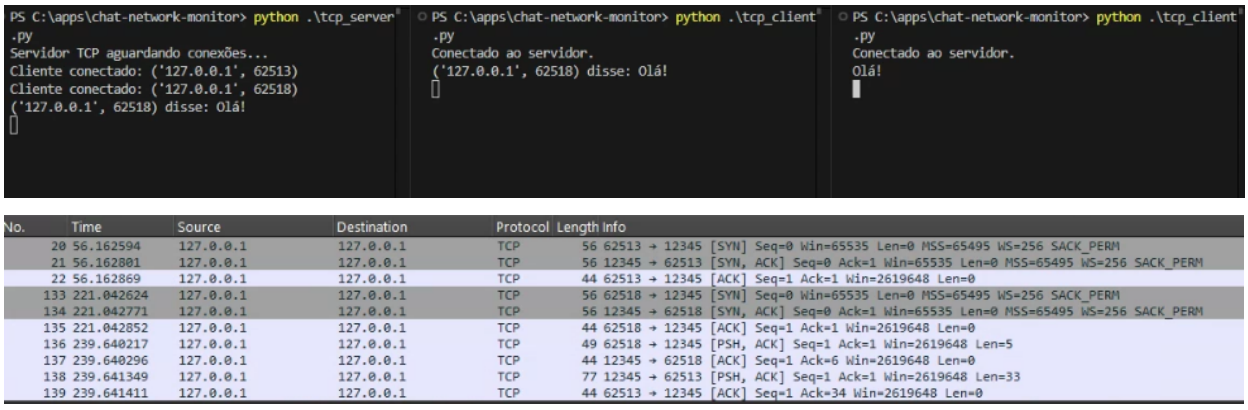
O status é exibido no terminal e enviado via e-mail ao destinatário configurado no código. Após cada medição, o monitor espera pelo tempo configurado e repete o processo.



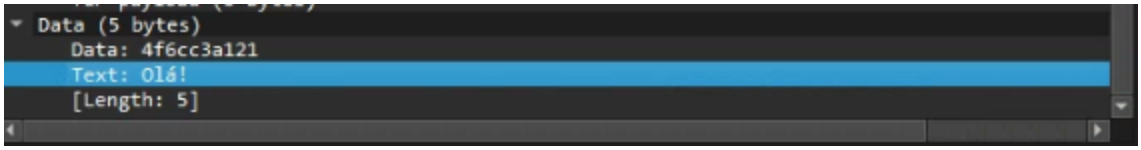
Captura de pacotes

Troca de pacotes no Chat TCP

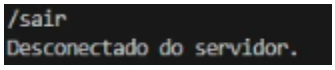
Dois client conectam ao servidor e um dele envia uma mensagem:



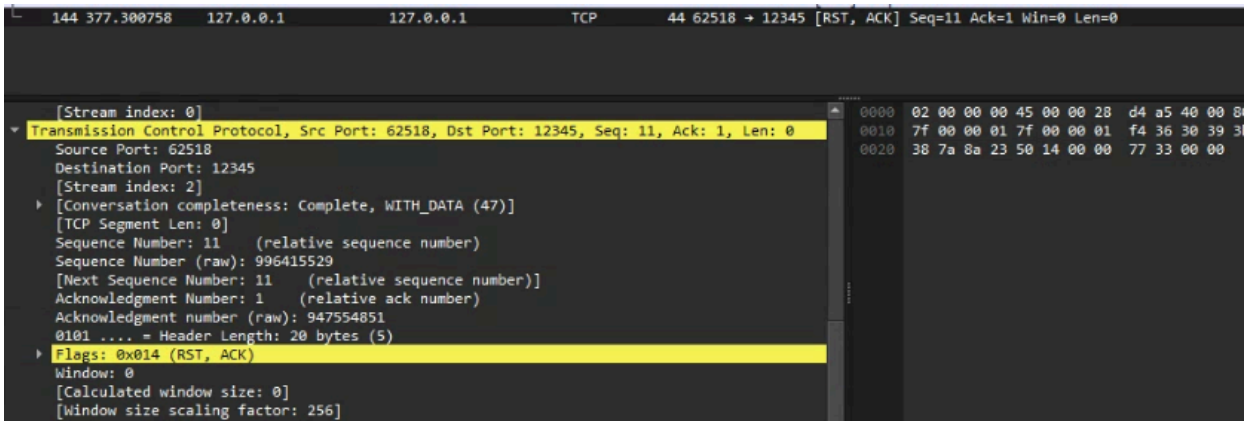
Dados da mensagem enviada:



Envio da mensagem /sair para desconectar do servidor:

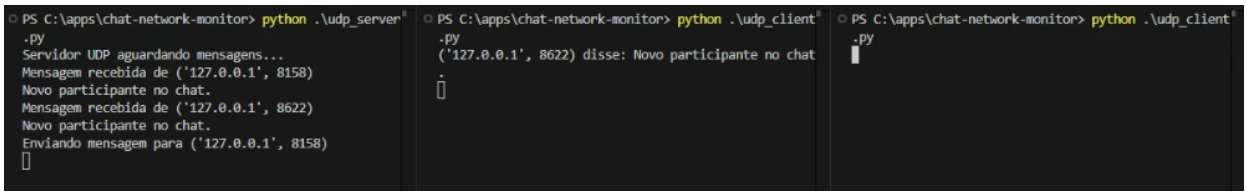


Flag de fechamento da conexão:

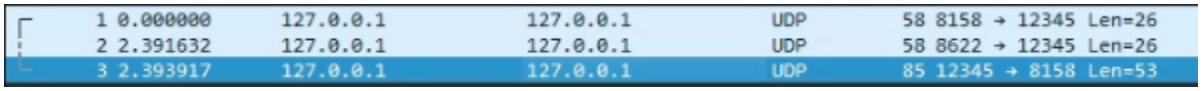


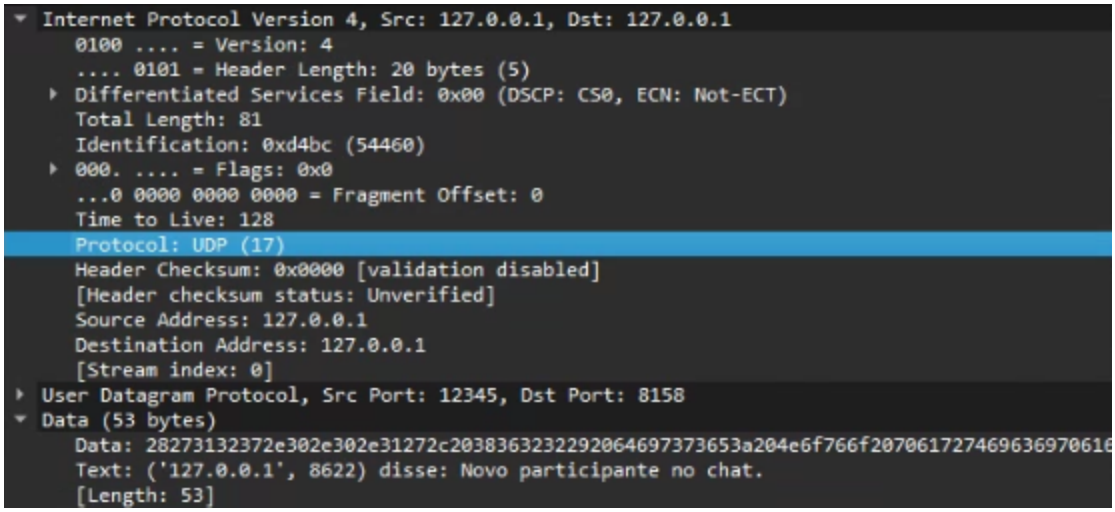
Troca de pacotes no Chat UDP

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

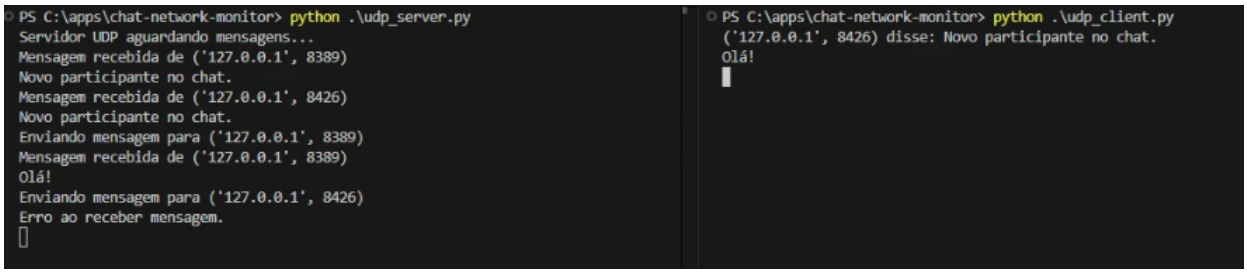


Mensagem do segundo client é redirecionada ao primeiro pelo servidor:





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



Client encerrado não pode ser alcançado: