TP 4.2 - Chat con Sockets en Python (Explicación Didáctica)

Este documento explica paso a paso cómo realizar el Trabajo Práctico 4.2 de Redes de Datos. Se desarrolla un chat cliente-servidor en Python utilizando sockets, tanto en versión TCP como UDP. El objetivo es comprender la comunicación en red, el uso de puertos, el manejo de hilos y las diferencias entre protocolos.

1. Conceptos básicos

Un socket es un punto final de comunicación entre dos programas en red. Cada socket se identifica por una dirección IP y un número de puerto. Tipos de sockets: • TCP (orientado a conexión, fiable): asegura la entrega y el orden de los datos. • UDP (sin conexión, no fiable): más rápido, pero puede perder paquetes. En un chat, el servidor escucha conexiones en un puerto fijo (por ejemplo, 5000) y los clientes se conectan a ese puerto. Los clientes usan puertos efímeros asignados automáticamente (49152–65535).

2. Servidor de chat TCP

Código del servidor (con comentarios explicativos):

```
import socket, threading

HOST = '0.0.0.0'  # Escucha en todas las interfaces
PORT = 5000       # Puerto fijo del chat

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((HOST, PORT))
server.listen(5)
print(f"Servidor escuchando en {HOST}:{PORT}")

clients = []
names = {}

def broadcast(msg, sender=None):
    for c in clients:
        if c != sender:
            try:
                c.send(msg.encode('utf-8'))
            except:
                c.close()
                clients.remove(c)

def handle_client(conn, addr):
    print(f"Cliente conectado desde {addr}")
    name = conn.recv(1024).decode('utf-8')
    names[conn] = name
    broadcast(f"** {name} se ha unido al chat **", conn)
    while True:
        try:
            msg = conn.recv(1024).decode('utf-8')
            if not msg: break
            if msg == '/listar':
                usuarios = ', '.join(names.values())
                conn.send(f"Usuarios: {usuarios}".encode('utf-8'))
            elif msg == '/quitar':
```

```
                break
            else:
                broadcast(f"{name}: {msg}", conn)
        except:
            break
    conn.close()
    clients.remove(conn)
    broadcast(f"** {name} ha salido del chat **")
    print(f"{name} desconectado.")

while True:
    conn, addr = server.accept()
    clients.append(conn)
    threading.Thread(target=handle_client, args=(conn, addr), daemon=True).start()
```

3. Cliente de chat TCP

```
import socket, threading, os

HOST = '127.0.0.1'  # IP del servidor
PORT = 5000

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))
name = input("Tu nombre: ")
sock.send(name.encode('utf-8'))

def recibir():
    while True:
        try:
            data = sock.recv(1024).decode('utf-8')
            if not data: break
            print("\r" + data + "\n> ", end="")
        except:
            break
    sock.close()
    os._exit(0)

threading.Thread(target=recibir, daemon=True).start()
print("Comandos: /listar /quitar")
while True:
    msg = input("> ")
    sock.send(msg.encode('utf-8'))
    if msg == '/quitar':
        break
sock.close()
```

4. Versión UDP (simplificada)

```
# Servidor UDP
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('0.0.0.0', 5001))
clientes = {}
while True:
    data, addr = s.recvfrom(1024)
    msg = data.decode()
    if addr not in clientes:
        clientes[addr] = msg
```

```
        print(f"{msg} conectado.")
        continue
    if msg == '/quitar':
        print(f"{clientes[addr]} desconectado.")
        clientes.pop(addr)
        continue
    for c in clientes:
        if c != addr:
            s.sendto(f"{clientes[addr]}: {msg}".encode(), c)
```

5. Comparación TCP vs UDP

TCP garantiza la entrega y el orden de los mensajes, usando confirmaciones (ACK) y conexión persistente. UDP envía mensajes sin confirmar, más rápido pero sin fiabilidad. En Wireshark se observa handshake (SYN, ACK) en TCP, mientras que en UDP solo aparecen datagramas.

6. Recomendaciones de uso

Para ejecutar en Ubuntu: • Iniciar primero el servidor: python servidor_chat.py • Luego abrir varios clientes (mismo o distinto equipo). • En red local, usar la IP real del servidor (hostname -I). • Verificar puerto libre con: sudo ss -tulnp | grep 5000

Conclusión

Con este ejemplo se cumple el objetivo del TP 4.2: comprender e implementar comunicación en red mediante sockets. El código permite chatear en tiempo real, manejar múltiples usuarios y comparar TCP con UDP.