**Socket UDP, Mejorado**

# **Paso 1:** Le pedí a deepseek que me hiciera unas mejoras técnicas, proporcionándoles las especificaciones correctas para las mejoras. Me brindó códigos ciertas mejoras para la comodidad.

**Código del servidor:**

import socket

import signal

import sys

import os

import argparse

from datetime import datetime

# Manejar argumentos de línea de comandos

parser = argparse.ArgumentParser(description='Servidor UDP mejorado con soporte dual-stack IPv4/IPv6')

parser.add\_argument('--ip', default=os.getenv('SERVER\_IP', '::'),

help='Dirección IP para escuchar (por defecto: todas las interfaces)')

parser.add\_argument('--port', type=int, default=os.getenv('SERVER\_PORT', 12345),

help='Puerto para escuchar (por defecto: 12345)')

parser.add\_argument('--buffer-size', type=int, default=os.getenv('BUFFER\_SIZE', 1024),

help='Tamaño del buffer de recepción (por defecto: 1024)')

args = parser.parse\_args()

# Validación de parámetros

if not (1 <= args.port <= 65535):

print(f"❌ Error: Puerto inválido {args.port}. Debe estar entre 1-65535")

sys.exit(1)

# Estadísticas de comunicación

stats = {

'clientes\_unicos': set(),

'total\_bytes': 0,

'inicio': datetime.now(),

'mensajes\_por\_cliente': {}

}

def cerrar\_servidor(sig, frame):

print("\n\n--- Estadísticas de Comunicación ---")

print(f"Tiempo activo: {datetime.now() - stats['inicio']}")

print(f"Clientes únicos: {len(stats['clientes\_unicos']}")

print(f"Total de bytes recibidos: {stats['total\_bytes']}")

print("Mensajes por cliente:")

for addr, count in stats['mensajes\_por\_cliente'].items():

print(f" {addr[0]}:{addr[1]} - {count} mensajes")

print("\nCerrando servidor UDP...")

if 'server\_socket' in globals():

server\_socket.close()

sys.exit(0)

# Registrar manejador de señales

signal.signal(signal.SIGINT, cerrar\_servidor)

try:

# Crear socket dual-stack (IPv4 + IPv6)

server\_socket = socket.socket(socket.AF\_INET6, socket.SOCK\_DGRAM)

server\_socket.setsockopt(socket.IPPROTO\_IPV6, socket.IPV6\_V6ONLY, 0) # Habilitar IPv4

server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1) # Reutilizar dirección

print(f"🔄 Iniciando servidor en [{args.ip}]:{args.port} (dual-stack IPv4/IPv6)")

server\_socket.bind((args.ip, args.port))

print(f"✅ Servidor UDP escuchando en [{args.ip}]:{args.port}")

print("Presiona Ctrl+C para detener el servidor\n")

except OSError as e:

print(f"❌ Error crítico al iniciar servidor: {e}")

sys.exit(1)

except Exception as e:

print(f"❌ Error inesperado: {e}")

sys.exit(1)

while True:

try:

data, client\_addr = server\_socket.recvfrom(args.buffer\_size)

stats['total\_bytes'] += len(data)

# Normalizar dirección IPv4-mapeada (::ffff:192.0.2.1 → 192.0.2.1)

if client\_addr[0].startswith('::ffff:'):

normalized\_addr = (client\_addr[0][7:], client\_addr[1])

else:

normalized\_addr = (client\_addr[0], client\_addr[1])

stats['clientes\_unicos'].add(normalized\_addr)

stats['mensajes\_por\_cliente'][normalized\_addr] = stats['mensajes\_por\_cliente'].get(normalized\_addr, 0) + 1

# Decodificación robusta con manejo de errores

try:

mensaje = data.decode('utf-8', errors='replace')

except UnicodeDecodeError:

mensaje = f"<Datos binarios: {len(data)} bytes>"

timestamp = datetime.now().strftime("%H:%M:%S")

print(f"[{timestamp}] {normalized\_addr[0]}:{normalized\_addr[1]} >> {mensaje}")

# Enviar ACK con reintentos

respuesta = f"ACK: Recibido ({len(data)} bytes)"

for \_ in range(3): # 3 intentos de reenvío

try:

server\_socket.sendto(respuesta.encode('utf-8'), client\_addr)

break

except socket.error as e:

print(f"⚠️ Error al enviar ACK (intento {\_+1}/3): {e}")

time.sleep(0.5) # Esperar antes de reintentar

except socket.timeout:

continue # Timeout de lectura, continuar normalmente

except socket.error as e:

print(f"🚨 Error de socket: {e}")

time.sleep(1) # Esperar antes de reintentar

except Exception as e:

print(f"⚠️ Error inesperado: {e}")

**Código del cliente:**

import socket

import time

import argparse

import ipaddress

def validar\_ip(ip):

try:

ipaddress.ip\_address(ip)

return True

except ValueError:

return False

def main():

parser = argparse.ArgumentParser(description='Cliente UDP mejorado con validación y reintentos')

parser.add\_argument('--ip', default='127.0.0.1', help='IP del servidor (por defecto: localhost)')

parser.add\_argument('--port', type=int, default=12345, help='Puerto del servidor (por defecto: 12345)')

parser.add\_argument('--timeout', type=float, default=5.0, help='Timeout en segundos (por defecto: 5)')

parser.add\_argument('--family', choices=['ipv4', 'ipv6', 'auto'], default='auto',

help='Familia de direcciones (ipv4/ipv6/auto)')

args = parser.parse\_args()

# Validación de parámetros

if not validar\_ip(args.ip):

print(f"❌ Error: Dirección IP inválida: {args.ip}")

return

if not (1 <= args.port <= 65535):

print(f"❌ Error: Puerto inválido {args.port}. Debe estar entre 1-65535")

return

print("="\*50)

print("📱 CLIENTE UDP MEJORADO")

print("="\*50)

print(f"Conectando a {args.ip}:{args.port} | Timeout: {args.timeout}s")

try:

# Selección automática de familia de direcciones

if args.family == 'auto':

try:

# Intentar conexión IPv6 primero

socket.getaddrinfo(args.ip, args.port, socket.AF\_INET6)

family = socket.AF\_INET6

print("🔗 Usando stack IPv6")

except socket.gaierror:

family = socket.AF\_INET

print("🔗 Usando stack IPv4")

else:

family = socket.AF\_INET6 if args.family == 'ipv6' else socket.AF\_INET

client\_socket = socket.socket(family, socket.SOCK\_DGRAM)

client\_socket.settimeout(args.timeout)

except OSError as e:

print(f"❌ Error crítico de socket: {e}")

return

except Exception as e:

print(f"❌ Error inesperado: {e}")

return

print("\n" + "="\*50)

print(f"✅ Conectado a {args.ip}:{args.port}")

print("Escriba 'exit' para salir")

print("="\*50 + "\n")

try:

while True:

mensaje = input("Ingrese mensaje >> ").strip()

if not mensaje:

continue

if mensaje.lower() == "exit":

break

# Enviar con reintentos

for intento in range(1, 4): # 3 intentos

try:

client\_socket.sendto(mensaje.encode('utf-8'), (args.ip, args.port))

start = time.time()

try:

respuesta, \_ = client\_socket.recvfrom(1024)

elapsed = round((time.time() - start) \* 1000, 2)

print(f"🟢 Respuesta ({elapsed} ms): {respuesta.decode()}\n")

break # Salir del bucle de reintentos

except socket.timeout:

print(f"🔴 Timeout (intento {intento}/3): Sin respuesta del servidor")

if intento == 3:

print("✖️ Servidor no responde después de 3 intentos\n")

except UnicodeDecodeError:

print("⚠️ Respuesta recibida no pudo ser decodificada\n")

break

except socket.error as e:

print(f"🚨 Error de red (intento {intento}/3): {e}")

if intento == 3:

print("✖️ Error persistente de red\n")

time.sleep(1) # Esperar entre reintentos

except KeyboardInterrupt:

print("\n🛑 Operación cancelada por el usuario")

finally:

client\_socket.close()

print("🔌 Conexión cerrada")

if \_\_name\_\_ == "\_\_main\_\_":

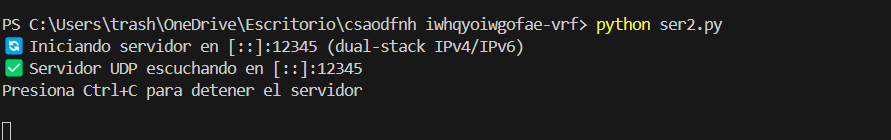
main()

**Mejoras notables:**

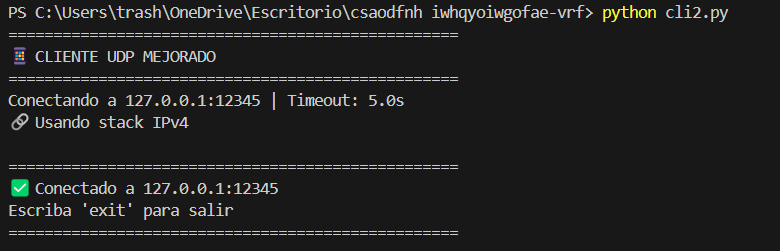
1. **Validación Avanzada de Parámetros:** Se implementó el uso de “ipaddress” que funcionan como una forma de automatizar la dirección ip predeterminada para una opción más cómoda y fácil. Como también validación de argumentos mediante argparse.
2. **Manejo Robustode Errores:** Tres intentos de reenvío de mensaje, como el apoyo de errores graves y soluciones accesibles como la decodificación UTF-8 con manejo de errores.
3. **Resiliencia de red:** Se agregó “SO\_REUSEADDR” para reinicios rápidos del servidor, como normalización de direcciones ip’s.
4. **Compatibilidad:** Se hizo más compatible con otros sistemas operativos.

## **Paso 2:** Nuevos inicios simples.

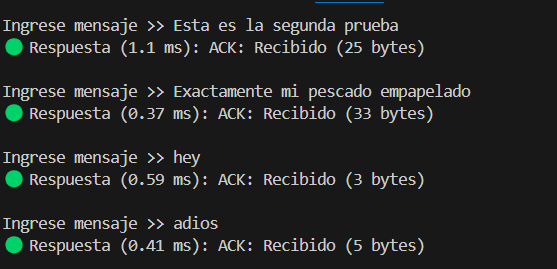
A pesar de que no hace mucha la diferencia, se aprecia un cambio más cómodo como la automatización de la dirección ips.



Aquí mismo podemos apreciar el cambio tras automatización.



Tras el ingreso de mensajes, se aprecia una diferencia ligera.



El servidor recibe los paquetes de datos entre el cliente y el servidor.

