



Instituto Politécnico Nacional  
Escuela Superior de Cómputo



## Ingeniería en Sistemas Computacionales Aplicaciones para comunicaciones en red

Academia “Sistemas Distribuidos”  
Plan 2020

Práctica 1:  
**“Sockets Orientados a Conexiones  
Bloqueantes”**

2015090269

González González Armando Omar

Profesor: Ojeda Santillan Rodrigo

## Contenido

Objetivo .....	3
Introducción .....	3
Desarrollo .....	4
Cliente .....	4
Servidor .....	4
Conclusión .....	12
González González Armando Omar: .....	12
Pregunta .....	12
¿Cómo podrían generar un negocio a través de lo visto en la práctica? .....	12
Bibliografía.....	12

## Objetivo

El propósito de esta práctica es entender e implementar sockets de conexión bloqueante en un entorno cliente-servidor utilizando Python. El objetivo es establecer una conexión confiable y secuencial a través del protocolo TCP (Transmission Control Protocol), donde las acciones de red, como conectar, enviar y recibir datos, bloquean la ejecución del programa hasta que se completan. Al finalizar la práctica, se espera que los estudiantes sean capaces de establecer conexiones bloqueantes entre un cliente y un servidor, transmitir datos de manera eficiente y analizar el tráfico de red generado con Wireshark.

## Introducción

Los sockets de conexión bloqueante son una forma de comunicación en red que sigue el modelo cliente-servidor, utilizando el protocolo TCP, que asegura una comunicación confiable y ordenada. En este tipo de sockets, las operaciones de red, como conectar y enviar o recibir datos, son bloqueantes, lo que significa que la ejecución del programa se detiene hasta que la operación se complete con éxito. Esto los hace ideales para situaciones donde es fundamental mantener el orden y la integridad de los datos transmitidos, como en aplicaciones de mensajería, transferencia de archivos o servicios web.

El uso de sockets bloqueantes asegura que los datos lleguen al destino en el orden adecuado, sin pérdida, lo cual es crucial en entornos que requieren conexiones fiables. Durante la práctica, se desarrollaron dos scripts en Python, uno para el cliente y otro para el servidor, que se comunican mediante sockets TCP en modo bloqueante. Además, se utilizó Wireshark para analizar el tráfico de red y observar detalladamente el intercambio de paquetes.

# Desarrollo

## Cliente

El cliente realiza una conexión al servidor en la IP 127.0.0.1 (localhost) en el puerto 5555 (puerto que escogimos nosotros) a través de un socket bloqueante.

Posteriormente procede a mandar un mensaje ‘hola mundo!’ hacia el servidor y finaliza la ejecución cerrando el socket.

```
import socket

def client(message: str, host: str = "127.0.0.1", port: int = 5555):
    # creating socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # blocking socket
    client_socket.setblocking(True)
    # connecting to server
    client_socket.connect((host, port))
    print(f"Connecting to server @ {host}:{port}")
    # sending message to server
    client_socket.sendall(message.encode())
    print(f"Message '{message}' sent to server")
    # closing socket
    client_socket.close()

client("hola mundo!")
```

Bloque de Código 1. Código del cliente.

## Servidor

El servidor crea un socket y lo asocia a la dirección IP 127.0.0.1 en el puerto 5555 y procede a escuchar las conexiones entrantes que lleguen a este.

El servidor entonces acepta la conexión entrante del cliente con un socket de conexión que se crea en algún puerto intermedio, nosotros podemos ver la procedencia de la conexión del cliente con las variables ***o\_host*** y ***o\_port***.

El socket de conexión también se pone en modo bloqueante, y se procede a recibir el mensaje del cliente. Finalmente se cierra el socket de conexión y se cierra también el socket del servidor.

```
import socket

def server(host: str = "127.0.0.1", port: int = 5555):
    # creating socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # blocking socket
    server_socket.setblocking(True)
    # binding socket to host:port
    server_socket.bind((host, port))
    # listening
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}")
    # waiting to accept a connection
    conn, (o_host, o_port) = server_socket.accept()
    print(f"Accepted connection from {o_host}:{o_port}")
    # blocking connection sokcet
    conn.setblocking(True)
    # receiving message
    message = conn.recv(1024).decode()
    # show message
    print(f"Received message: '{message}'")
    # closing connection and socket
    conn.close()
    server_socket.close()

server()
```

Bloque de Código 2. Código del servidor.

A screenshot of a terminal window on a Mac OS X system. The window title bar reads "practica1 — python3 server.py — python3 — Python server.py — 52x30". The main pane of the terminal shows the following command and output:

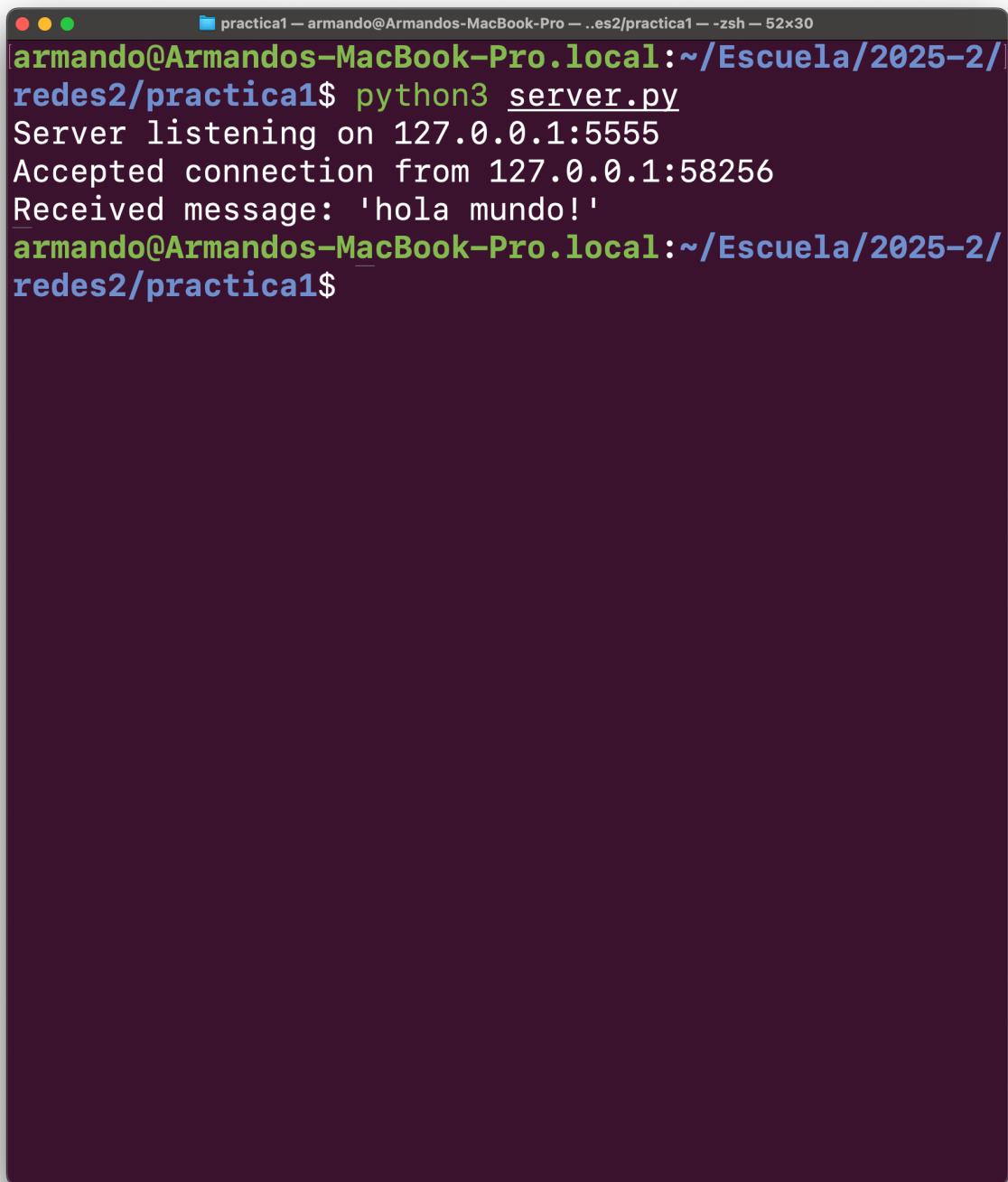
```
armando@Armandos-MacBook-Pro.local:~/Escuela/2025-2/|redes2/practica1$ python3 server.py
Server listening on 127.0.0.1:5555
```

The terminal has its characteristic dark blue background and light blue text.

Imagen 1. Ejecución del servidor.

```
practica1 — armando@Armandos-MacBook-Pro.local:~/Escuela/2025-2/rede...  
redes2/practica1$ python3 client.py  
Connecting to server @ 127.0.0.1:5555  
Message 'hola mundo!' sent to server  
armando@Armandos-MacBook-Pro.local:~/Escuela/2025-2/rede...  
redes2/practica1$
```

Imagen 2. Ejecución del cliente, conexión con el servidor y envío del mensaje.



A screenshot of a macOS terminal window titled "practica1". The window shows a command-line session where a Python server is running. The server is listening on port 127.0.0.1:5555 and has accepted a connection from 127.0.0.1:58256. It then receives a message from the client: "hola mundo!". The terminal window has a dark background and light-colored text.

```
practica1 — armando@Armandos-MacBook-Pro.local:~/Escuela/2025-2/redes2/practica1$ python3 server.py
Server listening on 127.0.0.1:5555
Accepted connection from 127.0.0.1:58256
Received message: 'hola mundo!'
practica1$
```

Imagen 3. Recepción del mensaje del cliente en el servidor.

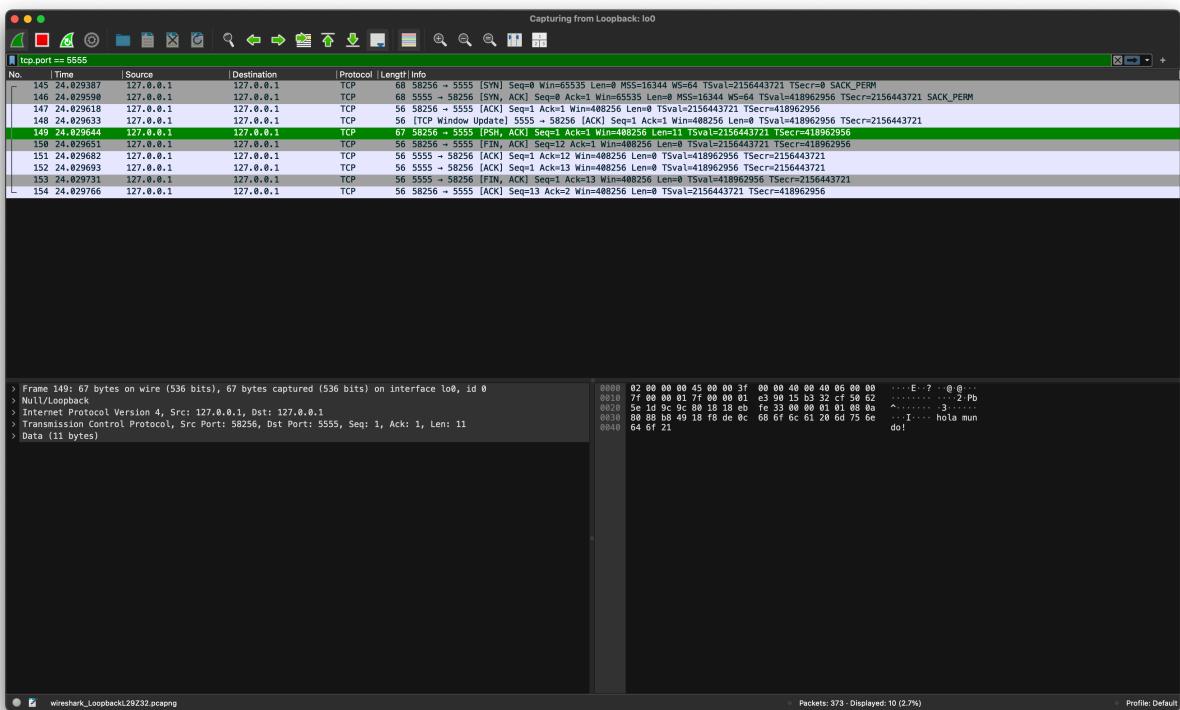


Imagen 4. Captura de la ejecución de servidor y cliente en Wireshark.

Imagen 5. Intento de ejecución del cliente sin el servidor.

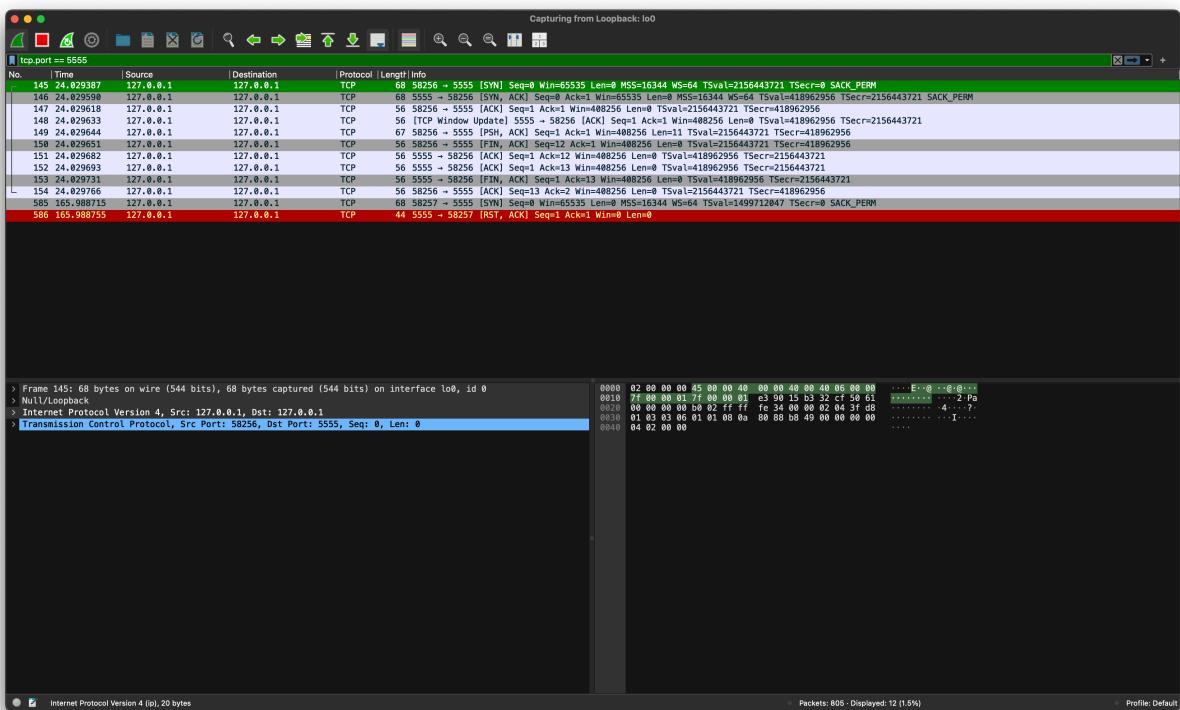


Imagen 6. Captura del socket bloqueado en Wireshark.

# Conclusión

González González Armando Omar:

El uso de sockets bloqueantes nos permite realizar aplicaciones de red de una forma muy sencilla, sin tener que preocuparnos de términos y lógica más compleja que conllevaría el hacer uso de sockets no bloqueantes, ya que en el modo bloqueante la comunicación es asíncrona o concurrente, i.e., múltiples conexiones que pueden realizar y terminar en distintos tiempos, por lo que sería necesario implementar una lógica que maneje ese tipo de conexiones.

Al usar sockets bloqueantes la comunicación es síncrona, básicamente requiere que termine un proceso antes de que empiece otro, en el caso de la práctica, se asocia primero el socket del servidor y se pone en modo escucha, esperando a recibir una conexión, después realizada la conexión se procede a recibir el mensaje y finalmente se cierra el socket de conexión y el socket del servidor.

## Pregunta

**¿Cómo podrían generar un negocio a través de lo visto en la práctica?**

Una aplicación que podríamos hacer implementando el uso de sockets bloqueantes podría ser un chat uno a uno con otra persona, o alguna otra aplicación de comunicación que requiera esperar y escuchar conexiones entrantes, esperar a recibir un mensaje, posteriormente mandar un mensaje y esperar una respuesta, así sucesivamente, con un flujo parecido a la comunicación por radio.

## Bibliografía

- **Postel, J. (1981).** Transmission Control Protocol - DARPA Internet Program Protocol Specification. RFC 793. <https://doi.org/10.17487/RFC0793>
- **Stevens, W. R. (1998).** *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley.
- Comer, Douglas E. Internetworking con TCP/IP Volumen Uno. Prentice Hall, 2006.
- Kurose, James F., y Keith W. Ross. Redes de Computadoras: Un Enfoque Descendente. Pearson, 2017.
- Beazley, David, y Brian K. Jones. Python Cookbook: Recetas para Dominar Python 3. O'Reilly Media, 2013.