

# Pontificia Universidad Javeriana



---

## Taller de Comunicación con Sockets en Java

---

### **Autor:**

Juan Manuel Lopez Vargas Juan Pablo Espinoza Robledo Johan Sebastian Mendez Juan  
Santiago Saavedra Holguín

11 de septiembre de 2025

Bogotá D.C.

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Objetivos</b>	<b>3</b>
<b>3. Implementación</b>	<b>3</b>
3.1. Servidor TCP (sockettcpser.java) . . . . .	3
3.2. Cliente TCP (sockettcpcli.java) . . . . .	3
3.3. Servidor UDP (socketudpser.java) . . . . .	3
3.4. Cliente UDP (socketudpcli.java) . . . . .	3
<b>4. Ejecución y Resultados</b>	<b>4</b>
4.1. Ejecución en Local . . . . .	4
4.1.1. TCP . . . . .	4
4.1.2. UDP . . . . .	4
4.2. Ejecución Remota . . . . .	5
4.2.1. Servidor TCP . . . . .	5
4.2.2. Cliente TCP . . . . .	5
4.2.3. Servidor UDP . . . . .	6
4.2.4. Cliente UDP . . . . .	6
<b>5. Conclusiones</b>	<b>7</b>

## 1. Introducción

Este informe presenta el desarrollo del taller de **Sockets en Java**, donde se implementaron ejemplos básicos de comunicación cliente-servidor utilizando los protocolos TCP y UDP.

El taller tuvo como propósito no solo comprender la teoría detrás de la comunicación en red, sino también ponerla en práctica a través de la ejecución de los programas en diferentes escenarios: tanto en un entorno local (cliente y servidor en la misma máquina) como en un entorno remoto (cliente y servidor en máquinas distintas).

De esta manera, se logró contrastar cómo los protocolos funcionan bajo distintas condiciones de red y se adquirió experiencia en la configuración de puertos y direcciones IP para establecer correctamente la comunicación.

## 2. Objetivos

- Comprender el funcionamiento de los sockets en la comunicación en red.
- Implementar un servidor y un cliente TCP.
- Implementar un servidor y un cliente UDP.
- Identificar diferencias clave entre TCP (orientado a conexión) y UDP (no orientado a conexión).
- Realizar pruebas de comunicación en dos escenarios: local (misma máquina) y remoto (máquinas distintas).
- Analizar el comportamiento de los protocolos bajo estas condiciones y reconocer sus ventajas y limitaciones.

## 3. Implementación

### 3.1. Servidor TCP (sockettcpser.java)

Escucha en el puerto 6001 y atiende los mensajes enviados por el cliente. Mantiene la conexión hasta que se recibe la palabra clave `fin`.

### 3.2. Cliente TCP (sockettcpcli.java)

Se conecta al servidor usando la dirección IP o `localhost`, y envía mensajes hasta que se envíe `fin`.

### 3.3. Servidor UDP (socketudpser.java)

Escucha datagramas en el puerto 6000. No mantiene conexión persistente, cada mensaje se procesa de manera independiente.

### 3.4. Cliente UDP (socketudpccli.java)

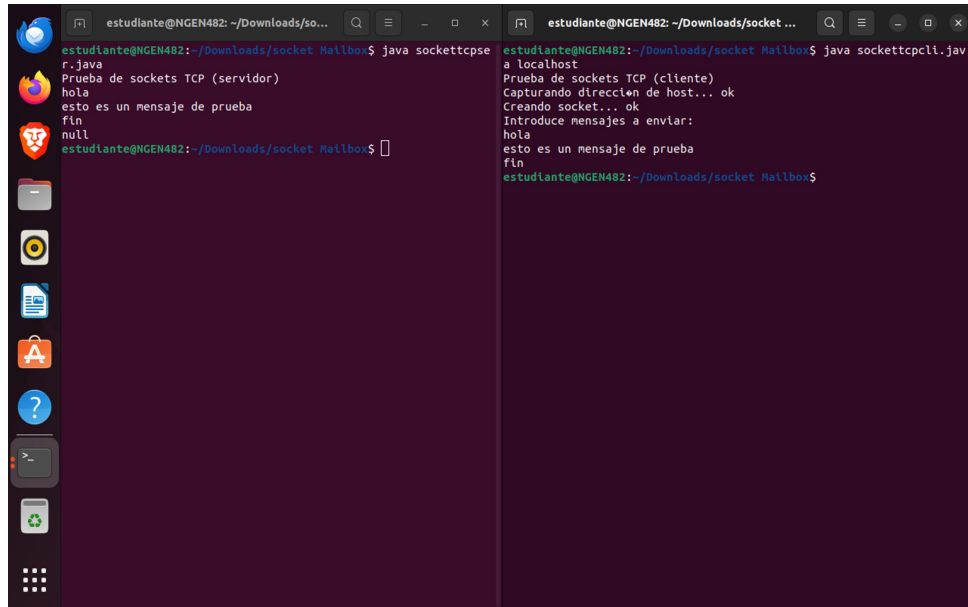
Envía datagramas al servidor UDP en `localhost` o en una IP específica. Termina al enviar `fin`.

## 4. Ejecución y Resultados

### 4.1. Ejecución en Local

En esta prueba se ejecutaron tanto el servidor como el cliente en la misma máquina (usando localhost). Se abrieron dos terminales: una para el servidor y otra para el cliente.

#### 4.1.1. TCP

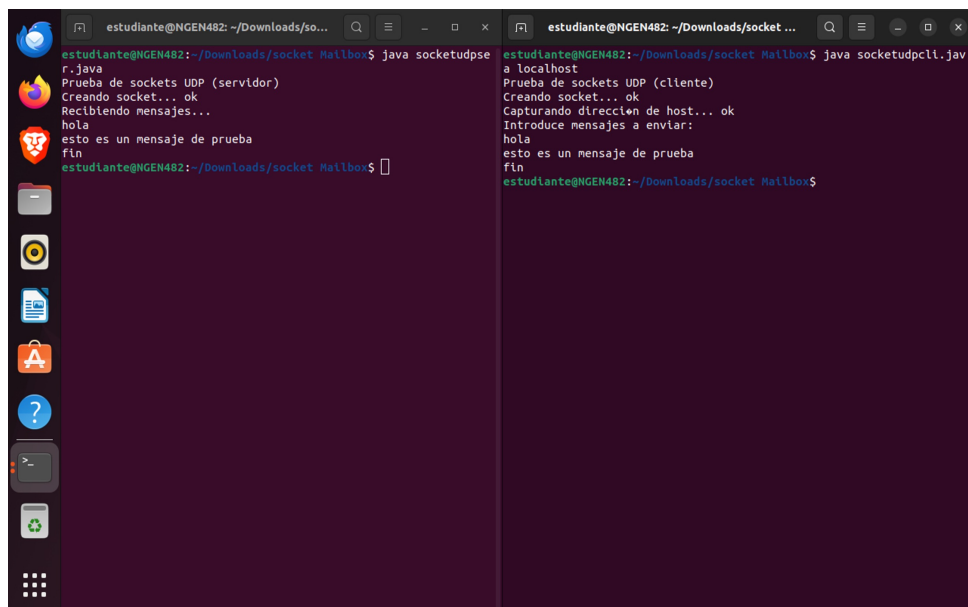


```
estudiante@NGEN482: ~/Downloads/socket Mailbox$ java sockettcpse
r.java
Prueba de sockets TCP (servidor)
hola
esto es un mensaje de prueba
fin
null
estudiante@NGEN482:~/Downloads/socket Mailbox$

estudiante@NGEN482:~/Downloads/socket Mailbox$ java sockettcpcli.java
a localhost
Prueba de sockets TCP (cliente)
Capturando dirección de host... ok
Creando socket... ok
Introduce mensajes a enviar:
hola
esto es un mensaje de prueba
fin
estudiante@NGEN482:~/Downloads/socket Mailbox$
```

Figura 1: Ejecución local del servidor y cliente TCP.

#### 4.1.2. UDP



```
estudiante@NGEN482: ~/Downloads/socket Mailbox$ java socketudpse
r.java
Prueba de sockets UDP (servidor)
Creando socket... ok
Recibiendo mensajes...
hola
esto es un mensaje de prueba
fin
estudiante@NGEN482:~/Downloads/socket Mailbox$

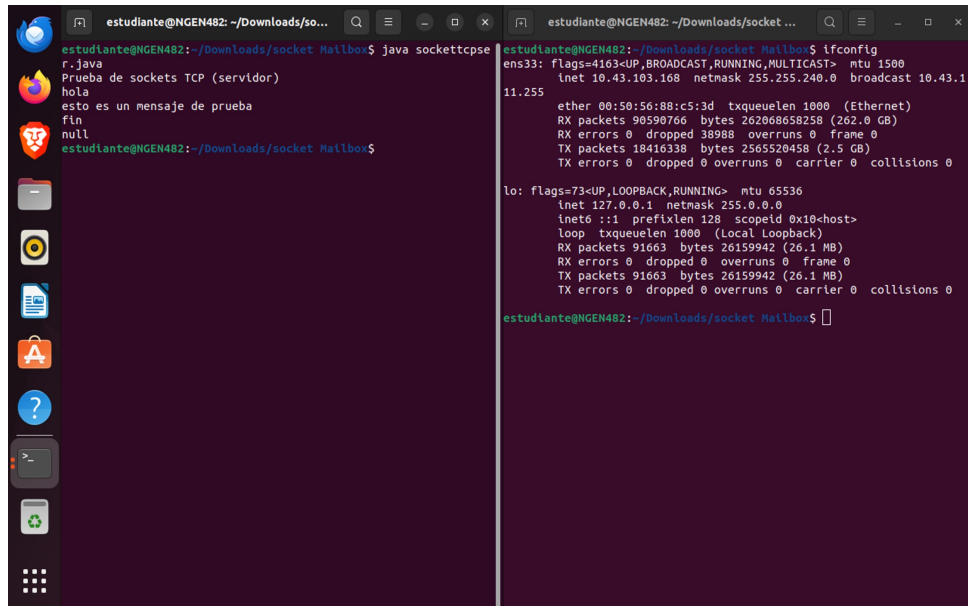
estudiante@NGEN482:~/Downloads/socket Mailbox$ java socketudpcli.java
a localhost
Prueba de sockets UDP (cliente)
Creando socket... ok
Capturando dirección de host... ok
Introduce mensajes a enviar:
hola
esto es un mensaje de prueba
fin
estudiante@NGEN482:~/Downloads/socket Mailbox$
```

Figura 2: Ejecución local del servidor y cliente UDP.

## 4.2. Ejecución Remota

En esta prueba el servidor se ejecutó en una máquina y el cliente en otra, utilizando la dirección IP del servidor en lugar de localhost.

### 4.2.1. Servidor TCP



```
estudiante@NGEN482: ~/Downloads/socket Mailbox$ java sockettcpse r.java
Prueba de sockets TCP (servidor)
hola
esto es un mensaje de prueba
fin
null
estudiante@NGEN482:~/Downloads/socket Mailbox$

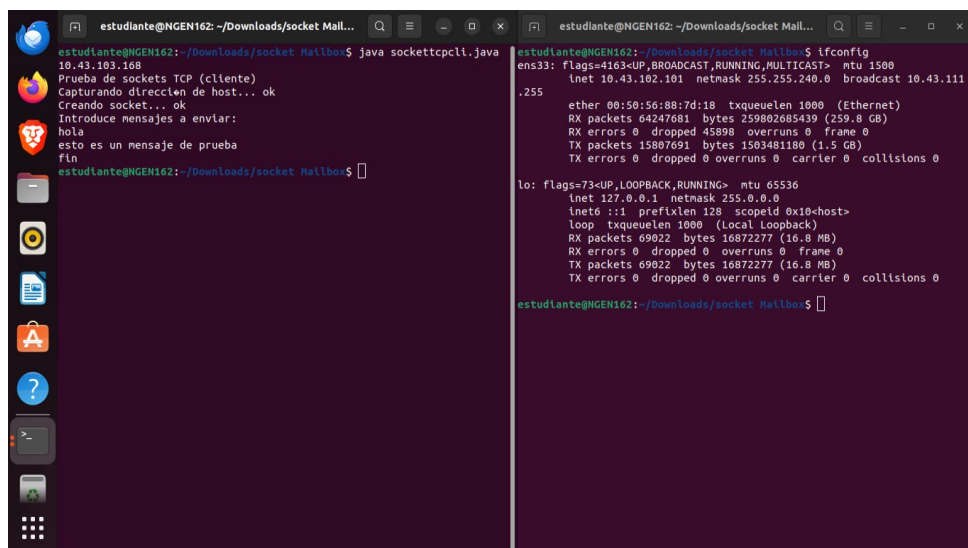
estudiante@NGEN482:~/Downloads/socket Mailbox$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.43.103.168 netmask 255.255.240.0 broadcast 10.43.1
11.255
    ether 00:50:56:88:c5:3d txqueuelen 1000 (Ethernet)
    RX packets 90590766 bytes 262068658258 (262.0 GB)
    RX errors 0 dropped 38988 overruns 0 frame 0
    TX packets 18416338 bytes 2565520458 (2.5 GB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 91663 bytes 26159942 (26.1 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 91663 bytes 26159942 (26.1 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

estudiante@NGEN482:~/Downloads/socket Mailbox$
```

Figura 3: Servidor TCP recibiendo conexión remota.

### 4.2.2. Cliente TCP



```
estudiante@NGEN162:~/Downloads/socket Mailbox$ java sockettcpcli.java
10.43.103.168
Prueba de sockets TCP (cliente)
Capturando dirección de host... ok
Creando socket... ok
Introduce mensajes a enviar:
hola
esto es un mensaje de prueba
fin
estudiante@NGEN162:~/Downloads/socket Mailbox$

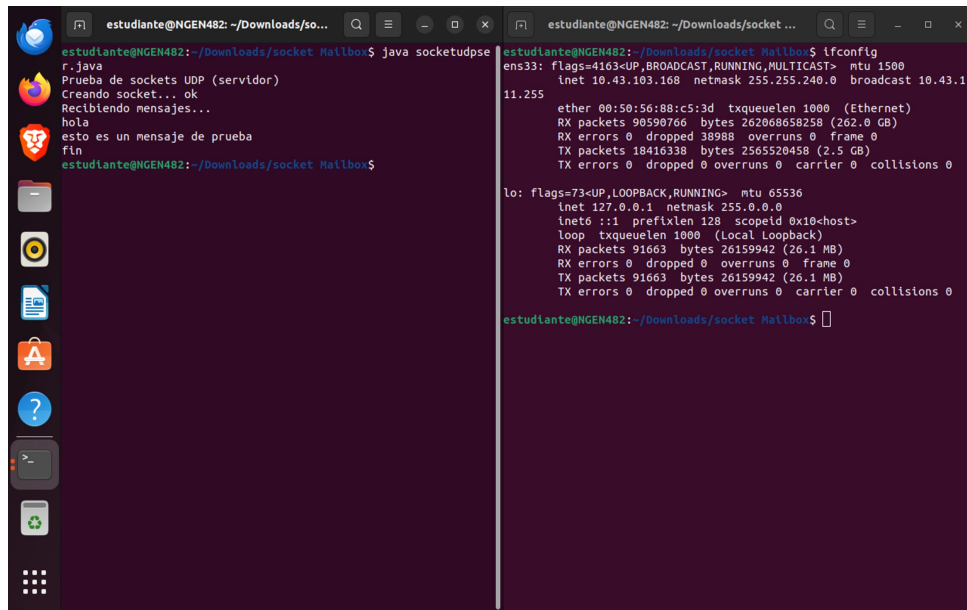
estudiante@NGEN162:~/Downloads/socket Mailbox$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.43.102.101 netmask 255.255.240.0 broadcast 10.43.111
.255
    ether 00:50:56:88:7d:18 txqueuelen 1000 (Ethernet)
    RX packets 64247681 bytes 259802685439 (259.8 GB)
    RX errors 0 dropped 45898 overruns 0 frame 0
    TX packets 15807691 bytes 1503481180 (1.5 GB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 69022 bytes 16872277 (16.8 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 69022 bytes 16872277 (16.8 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

estudiante@NGEN162:~/Downloads/socket Mailbox$
```

Figura 4: Cliente TCP conectado a servidor remoto.

### 4.2.3. Servidor UDP



```
estudiante@NGEN482: ~/Downloads/socket Mailbox$ java socketudpse
r.java
Prueba de sockets UDP (servidor)
Creando socket... ok
Recibiendo mensajes...
hola
esto es un mensaje de prueba
fin
estudiante@NGEN482: ~/Downloads/socket Mailbox$

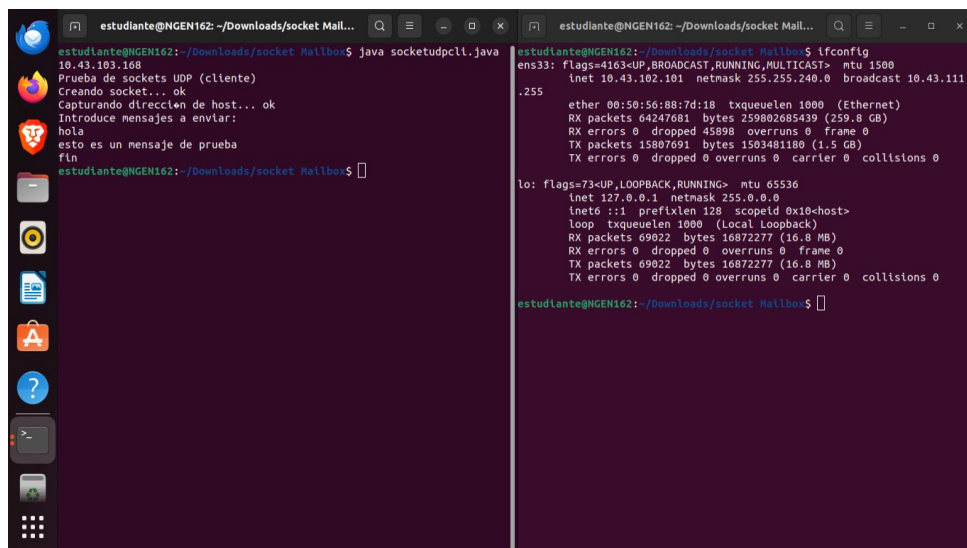
estudiante@NGEN482: ~/Downloads/socket Mailbox$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.43.103.168 netmask 255.255.240.0 broadcast 10.43.1
11.255
    ether 00:50:56:98:c5:3d txqueuelen 1000 (Ethernet)
    RX packets 90590766 bytes 262086650258 (262.0 GB)
    RX errors 0 dropped 38988 overruns 0 frame 0
    TX packets 18416338 bytes 2565520458 (2.5 GB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 91063 bytes 26159942 (26.1 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 91063 bytes 26159942 (26.1 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

estudiante@NGEN482: ~/Downloads/socket Mailbox$
```

Figura 5: Servidor UDP recibiendo datagramas desde un cliente remoto.

### 4.2.4. Cliente UDP



```
estudiante@NGEN162: ~/Downloads/socket Mailbox$ java socketudpci.java
10.43.103.168
Prueba de sockets UDP (cliente)
Creando socket... ok
Capturando dirección de host... ok
Introduce mensajes a enviar:
hola
esto es un mensaje de prueba
fin
estudiante@NGEN162: ~/Downloads/socket Mailbox$

estudiante@NGEN162: ~/Downloads/socket Mailbox$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.43.102.101 netmask 255.255.240.0 broadcast 10.43.111
.255
    ether 00:50:56:88:7d:18 txqueuelen 1000 (Ethernet)
    RX packets 64247681 bytes 259802685439 (259.8 GB)
    RX errors 0 dropped 45898 overruns 0 frame 0
    TX packets 15807691 bytes 1503481180 (1.5 GB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 69022 bytes 16872277 (16.8 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 69022 bytes 16872277 (16.8 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

estudiante@NGEN162: ~/Downloads/socket Mailbox$
```

Figura 6: Cliente UDP enviando mensajes a servidor remoto.

## 5. Conclusiones

- Se logró implementar correctamente la comunicación cliente-servidor utilizando los protocolos TCP y UDP en Java.
- Las pruebas en entorno local (servidor y cliente en la misma máquina) permitieron validar la lógica básica de envío y recepción de mensajes.
- Las pruebas en entorno remoto (servidor y cliente en máquinas distintas) demostraron el uso adecuado de direcciones IP y la correcta apertura de puertos para la comunicación.
- Se evidenció que **TCP** es un protocolo confiable y orientado a conexión, adecuado para aplicaciones donde la entrega de datos debe estar garantizada (chats, transferencias de archivos, correos, etc.).
- Se comprobó que **UDP** es más ligero y rápido, aunque menos confiable, lo que lo hace ideal para aplicaciones en tiempo real como streaming, juegos en línea o videollamadas.
- El taller permitió afianzar conceptos prácticos de programación en red, comparando directamente las diferencias entre ambos protocolos en escenarios controlados.