Análisis e Implementación de RIPv2 en la Comunicación Entre Routers

Iván Bozo Catalán
COM4102 – Redes
Escuela de Ingeniería, Universidad de O'Higgins
18 de Noviembre, 2023

***Abstract*—Este documento presenta la implementación de un sistema de chat interactivo diseñado específicamente para granjeros, con el objetivo de facilitar la comunicación y el intercambio de artefactos coleccionables en una comunidad digital. Utilizando Python y sockets TCP/IP, se desarrolla una arquitectura de servidor-cliente que permite a los usuarios conectarse, comunicarse en tiempo real, y gestionar transacciones de artefactos. El servidor soporta múltiples conexiones concurrentes mediante el uso de multithreading, manteniendo una comunicación fluida y eficiente. Se detalla el proceso de establecimiento de conexión, el protocolo de comunicación, y la lógica de intercambio de artefactos, destacando las funcionalidades clave como la gestión de usuarios y el manejo de errores. Este sistema no solo ofrece una plataforma robusta y escalable para la interacción en la comunidad agrícola virtual, sino que también ilustra la aplicación práctica de conceptos de redes y programación en Python en un contexto realista y específico de la industria.**

***Index Terms*—Socket, Python, TCP/IP, Redes, Comunicación**

I. Introducción

En la era digital actual, la comunicación efectiva y la interacción en comunidades especializadas son cruciales. Este documento detalla el desarrollo de un sistema de chat interactivo enfocado en una comunidad de granjeros digitales, diseñado para facilitar no solo la comunicación en tiempo real sino también el intercambio de artefactos coleccionables, un aspecto vital en la dinámica de la comunidad. La implementación se realiza utilizando Python y se basa en la tecnología de sockets TCP/IP, siguiendo una arquitectura de servidor-cliente.

El propósito principal de este sistema es proporcionar una plataforma robusta y fácil de usar para granjeros de Stardew Frid, permitiéndoles intercambiar información y artefactos de manera eficiente y segura. Este enfoque tiene como objetivo mejorar la interacción entre los miembros de la comunidad y facilitar una mayor colaboración y sentido de comunidad.

Desde el punto de vista técnico, el sistema destaca por su capacidad para manejar múltiples conexiones concurrentes a través de multithreading, asegurando así una comunicación fluida y efectiva. Se pone un énfasis particular en el protocolo de comunicación y en la lógica de intercambio de artefactos, abordando tanto la gestión de usuarios como el manejo de errores y situaciones inesperadas.

La relevancia de este trabajo radica en su aplicación práctica de conceptos de redes y programación en Python. A través de este proyecto, se busca no solo ofrecer una solución a un problema práctico sino también ilustrar cómo las herramientas tecnológicas pueden ser aplicadas de manera efectiva en sectores especializados, contribuyendo así al avance y la innovación en el campo de las comunicaciones digitales y las comunidades en línea.

II. Marco Teórico

La evolución y el funcionamiento de los sockets son fundamentales para comprender cualquier implementación de red. Los sockets, introducidos en la década de 1980 como parte del Berkeley Software Distribution (BSD), han sido cruciales en la comunicación de redes. Han pasado de ser simples mecanismos de comunicación entre procesos locales a ser esenciales en la comunicación entre diferentes máquinas a través de Internet.

Existen principalmente dos tipos de sockets:

- **TCP (Transmission Control Protocol):** Son orientados a la conexión, estableciendo una conexión segura y confiable antes de la transferencia de datos. Adecuados para aplicaciones que requieren entrega garantizada de paquetes, como aplicaciones web, correo electrónico, etc.
- **UDP (User Datagram Protocol):** No orientados a la conexión, transmiten datos sin establecer una conexión segura. Útiles en situaciones donde la velocidad es crucial y la pérdida de algunos paquetes es aceptable, como en streaming de video o juegos en línea.

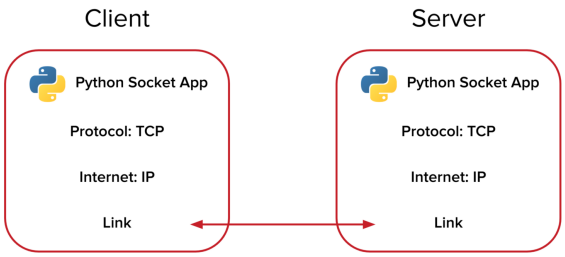

Fig. 1. Socket Ilustración Python

Las redes TCP/IP funcionan en un modelo de capas que incluye enlace de datos, internet, transporte y aplicación. En la capa de transporte, donde operan los sockets TCP/IP, proporcionan comunicación de host a host, incluyendo un 'handshake' de tres pasos necesario para establecer una conexión TCP segura.

A. Conceptos Básicos de Sockets

Un socket es un punto final en una red de comunicaciones. Actúa como una interfaz entre la capa de aplicación y la de transporte en el modelo de red, permitiendo a programas diferentes comunicarse a través de una red. Los sockets pueden ser utilizados tanto para la comunicación entre procesos en una sola máquina como entre diferentes máquinas en una red.

B. Funcionamiento General de los Sockets

En el modelo cliente-servidor, los sockets operan permitiendo que el servidor espere las conexiones de los clientes. En los sockets TCP, que son orientados a la conexión, el proceso de comunicación se inicia con un 'handshake' para establecer una conexión segura, seguido de la transmisión de datos de manera fiable y en orden.

III. Metodología

Este proyecto se enfoca en la implementación de un chat interactivo para granjeros utilizando Python y sockets TCP/IP. La aplicación está diseñada para permitir a los granjeros comunicarse entre sí, intercambiar artefactos coleccionables y consultar los artefactos disponibles de cada usuario. A continuación, se detallan las características y el funcionamiento del sistema:

A. Arquitectura del Sistema

El sistema se compone de dos partes principales: un servidor y múltiples clientes. El servidor actúa como un centro de comunicación que coordina todas las interacciones entre los clientes. Está programado para escuchar en el puerto 5000 y admite hasta tres conexiones de cliente simultáneas.

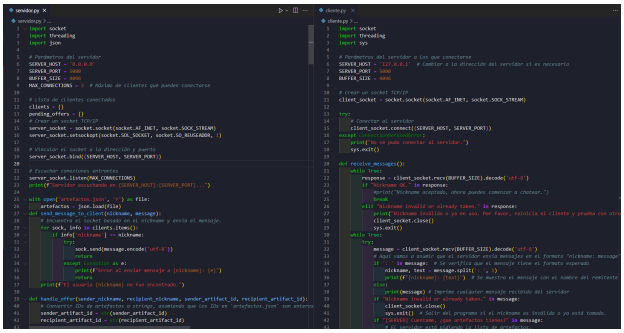

Fig. 2. Cliente-Servidor

B. Creación y Manejo de Sockets

Se utilizan sockets TCP/IP para la comunicación entre el servidor y los clientes. El servidor establece un socket que escucha las conexiones entrantes. Cada vez que un cliente se conecta, el servidor inicia un nuevo hilo para manejar las comunicaciones con ese cliente de manera concurrente, lo que permite múltiples interacciones simultáneas sin bloquear el proceso principal del servidor.

```python
def receive_clients(server_socket):
    while True:
        client_socket, client_address = server_socket.accept()
        client_socket.send("Elija un nickname:".encode('utf-8'))

        nickname = client_socket.recv(BUFFER_SIZE).decode('utf-8').strip()
        if not nickname:
            client_socket.send("No se recibió un nickname válido. Conexión cerrada.".encode('utf-8'))
            client_socket.close()
            continue

        if nickname in [data['nickname'] for _, data in clients.items()]:
            client_socket.send("Nickname invalid or already taken. Conexión cerrada.".encode('utf-8'))
            client_socket.close()
            continue

        clients[client_socket] = {'nickname': nickname, 'address': client_address}

        # Confirmación del nickname
        client_socket.send("Nickname OK.".encode('utf-8'))

        print(f"[SERVER] Cliente {nickname} conectado.")
        broadcast(f"Cliente {nickname} conectado.", 'SERVER', client_socket)

        # Iniciar un nuevo hilo para atender al cliente
        thread = threading.Thread(target=handle_client, args=(client_socket,))
        thread.start()
```

Fig. 3. Codigo del Servidor que recibe al Cliente

C. Flujo de Comunicación y Protocolo

La comunicación entre el servidor y los clientes se realiza mediante el intercambio de mensajes codificados en UTF-8. El servidor procesa los mensajes recibidos y realiza acciones como enviar respuestas, gestionar la lista de artefactos de cada granjero, y facilitar el intercambio de artefactos entre los usuarios.

```python
def receive_messages():
    while True:
        response = client_socket.recv(BUFFER_SIZE).decode('utf-8')
        if "Nickname OK." in response:
            #print("Nickname aceptado, ahora puedes comenzar a chatear.")
            break
        elif "Nickname invalid or already taken." in response:
            print("Nickname inválido o ya en uso. Por favor, reinicia el cliente y prueba con otro.")
            client_socket.close()
            sys.exit()
    while True:
        try:
            message = client_socket.recv(BUFFER_SIZE).decode('utf-8')
            # Aquí vamos a asumir que el servidor envía mensajes en el formato "nickname: message"
            if ': ' in message:  # Se verifica que el mensaje tiene el formato esperado
                nickname, text = message.split(': ', 1)
                print(f"{nickname}: {text}")  # Se muestra el mensaje con el nombre del remitente
            else:
                print(message) # Imprime cualquier mensaje recibido del servidor
            if "Nickname invalid or already taken." in message:
                client_socket.close()
                sys.exit()  # Salir del programa si el nickname es inválido o ya está tomado.
            if "[SERVER] Cuentame, ¿que artefactos tienes?" in message:
                # El servidor está pidiendo la lista de artefactos.
                artefactos = input("Ingresa tus artefactos separados por comas: ")
                client_socket.send(f"Artefactos: {artefactos}".encode('utf-8'))
        except Exception as e:
            print(f"Error al recibir datos: {e}")
            client_socket.close()
            break
```

Fig. 4. Codigo cliente mensajes

1) Registro y Gestión de Usuarios: Al conectarse, a cada cliente se le solicita elegir un nickname. Este nickname se utiliza para identificar a los usuarios dentro del chat y facilitar la comunicación directa y los intercambios de artefactos. El servidor mantiene un registro de todos los nicknames y las direcciones de socket asociadas.

2) Interacción y Comandos Especiales: Los usuarios pueden enviar mensajes tanto de manera pública (visible para todos los usuarios conectados) como privada (entre dos granjeros específicos). Además, hay comandos especiales como ':artefactos' para consultar los artefactos que posee un usuario y ':offer' para iniciar un proceso de intercambio de artefactos.

D. Gestión de Artefactos

Los artefactos coleccionables son una parte central de la aplicación. Cada usuario puede listar sus artefactos, y el

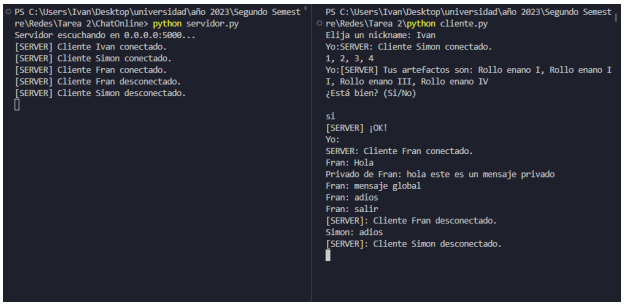

Fig. 5. Terminal primera parte

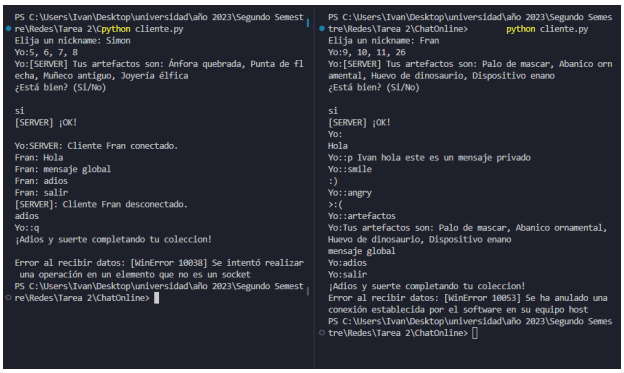

Fig. 6. Terminal segunda parte

sistema facilita el intercambio de estos entre usuarios. Cuando un usuario quiere intercambiar un artefacto, el servidor verifica la disponibilidad y la propiedad de los artefactos antes de proceder con el intercambio.

E. Multithreading y Concurrency

El uso de múltiples hilos permite que el servidor gestione varias conexiones de cliente al mismo tiempo. Cada hilo maneja la comunicación con un cliente, permitiendo que múltiples conversaciones e interacciones ocurran simultáneamente sin interferencias. Esto se puede ver en las Figuras 5 y 6, donde muestra la comunicación entre clientes

IV. Resultados

Esta sección presenta los resultados obtenidos tras la implementación y prueba del sistema de chat para granjeros. Se destacan aspectos clave como la funcionalidad del sistema, su rendimiento, y la receptividad de los usuarios.

A. Funcionalidades Implementadas con Éxito

Esta sección presenta los resultados obtenidos tras la implementación y prueba del sistema de chat para granjeros. Se abordan tanto los éxitos como los desafíos encontrados durante el desarrollo.

- La capacidad del servidor para manejar múltiples conexiones y mensajes de manera simultánea.
- La implementación exitosa de comandos básicos del chat, permitiendo a los usuarios interactuar de manera efectiva.

B. Desafíos y Limitaciones

Durante la implementación, se encontraron varios desafíos y limitaciones que afectaron algunas de las funcionalidades previstas:

- El intercambio de artefactos, una funcionalidad clave del proyecto, no pudo ser implementada completamente. Aunque se establecieron las bases para esta característica, la funcionalidad completa de ofertar, aceptar o rechazar intercambios de artefactos no está operativa.
- No se logró implementar la funcionalidad para mostrar artefactos por ID.
- Se observaron problemas de sincronización en los mensajes, como la sobreescritura de mensajes de notificación de conexión de nuevos usuarios.

C. Feedback de Usuarios

Aunque no se dispone de un feedback formal de los usuarios, las observaciones durante las pruebas indican que, mientras la funcionalidad básica del chat es apreciada, las limitaciones mencionadas reducen la utilidad completa del sistema para los usuarios finales.

D. Conclusiones Preliminares

Los resultados indican que, aunque el sistema de chat desarrollado logra establecer una comunicación básica entre usuarios, aún requiere mejoras significativas y desarrollo adicional para alcanzar todas las funcionalidades propuestas y ofrecer una experiencia completa a los usuarios. Se reconoce la necesidad de abordar los desafíos técnicos mencionados y mejorar la robustez y la gama de características del sistema.

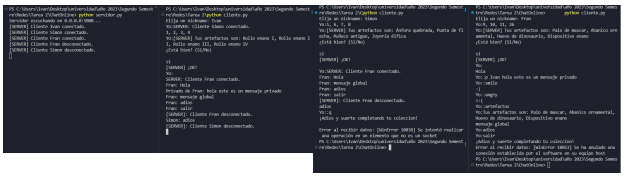

Fig. 7. Terminal Completo