

Universidad de Guadalajara.
Centro Universitario de Ciencias Exactas e Ingenierías.

Optimización de Transferencia de Archivos en una VPN con Algoritmos Voraces



Mtro: Jorge Ernesto Lopez-Arce Delgado
Análisis de Algoritmos NCR:204835
Calendario: 25-A Clave: IL355

Datos del Equipo

Chicas Superpoderosas

Giovanni Castrejon Fuentes 219884678

Rol: Programador

Andrés Salazar Torres 219130169

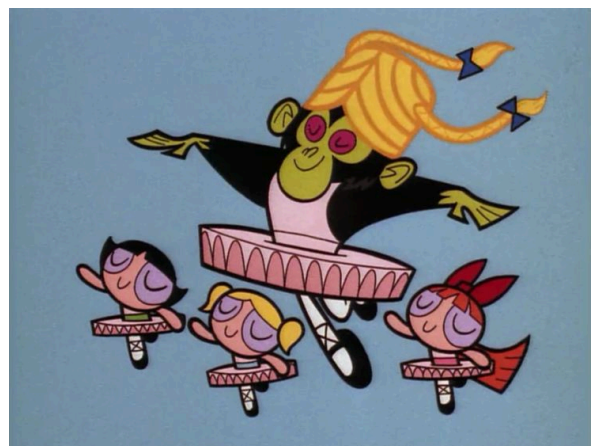
Rol: Project Manager

Marlene Ximena Sandoval García 219511456

Rol: Programador / Secretaria

Alan Jared Sandoval Suarez 219518415

Rol: Programador



Índice

<u>Introducción</u>	<u>3</u>
<u>Descripción</u>	<u>3</u>
<u>Desarrollo</u>	<u>4</u>
Paso 1: Crear la VPN	4
Paso 2: Medición de las métricas.	5
Paso 3: Implementación de Dijkstra	10
Paso 4: Implementación de Kruskal ("Topología Eficiente")	12
Topología:	13

Introducción

Una VPN (Red Privada Virtual) crea un canal cifrado entre el cliente y el servidor. Aunque proporciona seguridad, puede introducir **latencia** y **cuellos de botella** debido a:

- Cifrado y descifrado en tiempo real.
- Ruta indirecta de los paquetes (por ejemplo, pasar por nodos VPN).
- Limitaciones de ancho de banda del servidor VPN.

Cuando se transfieren archivos grandes o múltiples archivos, estos factores afectan el rendimiento general.

Descripción

En el presente proyecto implementamos la transferencia de archivos por medio del algoritmo de Dijkstra. Para esto fue requerido la aplicación de la página web de Tailscale para obtener y conocer las IPs de cada uno de los usuarios que participaran en el proyecto. Posteriormente de haber conseguido cada una de las IPs las mismas nos sirvieron para realizar métricas, como la latencia y el ancho de banda. Seguido de esto, implementamos Dijkstra para la transferencia de archivos y Kruskal para conocer el árbol de expansión mínima (MST) con la finalidad de optimizar el uso de la red.

Desarrollo

Paso 1: Crear la VPN

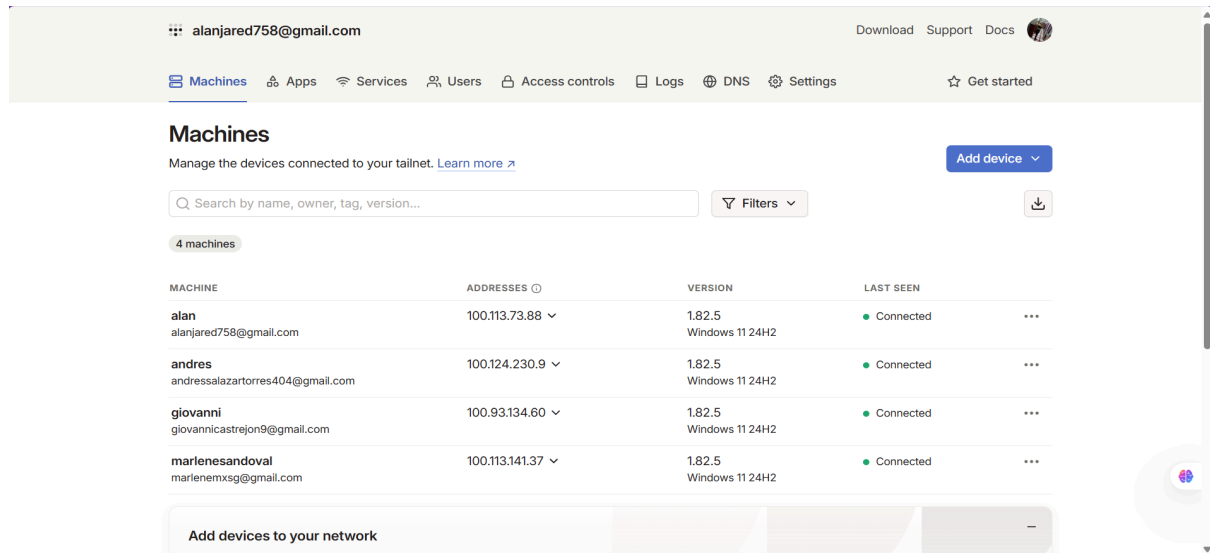
Para la creación de la VPN decidimos hacer uso de la herramienta Tailscale, que nos ayudó a crear rápidamente una VPN para después comenzar a agregar al resto de colaboradores y sus dispositivos.

Esta herramienta permitió interconectar cada uno de nuestros equipos facilitando muchas de las funcionalidades de nuestros programas

Para conectarse a la red tailscale del creador de la VPN, necesitamos pagar la suscripción de tailscale ya que únicamente permite a 3 usuarios lo cual en nuestro equipo no era apto. Seguido de ello, el Cowner de la VPN, compartirá la red ya sea por medio de correo electrónico o link para que los demás colaboradores se unan

con un correo electrónico de preferencia y después seleccionen unirse a la red compartida. Cabe mencionar, que el owner es el encargado de asignar que roles tendrán los invitados antes de unirse a la red.

Evidencia del paso 1:



Img 1. IP's de cada integrante desde "Tailscale".

Paso 2: Medición de las métricas.

Para la mediciones de las métricas fue necesario dividirlo en dos partes. La primera parte se basó únicamente en medir la latencia que se tenía al conectar cada equipo, para esto se hizo uso de un script de python, donde se utilizó la librería pythonping para poder hacer ping con cada uno de las computadoras y poder obtener valores como el mínimo, máximo y latencia promedio que se tenía entre cada conexión

```
C:\WINDOWS\system32\cmd. x + v
Estadísticas de ping para 100.113.73.88:
  Paquetes: enviados = 4, recibidos = 4, perdidos = 0
  (0% perdidos),
  Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 106ms, Máximo = 963ms, Media = 336ms

C:\Users\marle>ping 100.124.230.9

Haciendo ping a 100.124.230.9 con 32 bytes de datos:
Respuesta desde 100.124.230.9: bytes=32 tiempo=389ms TTL=128
Respuesta desde 100.124.230.9: bytes=32 tiempo=322ms TTL=128
Respuesta desde 100.124.230.9: bytes=32 tiempo=161ms TTL=128
Respuesta desde 100.124.230.9: bytes=32 tiempo=260ms TTL=128

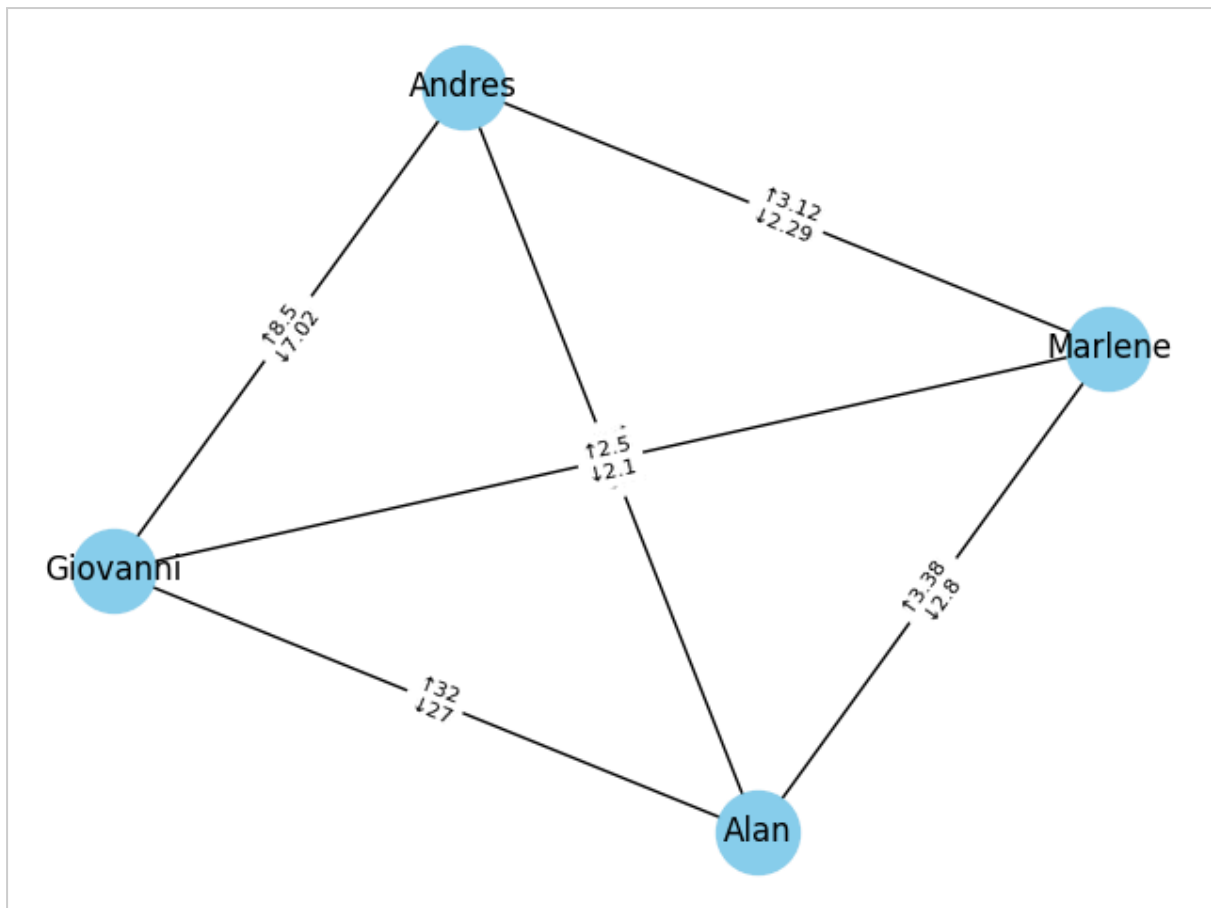
Estadísticas de ping para 100.124.230.9:
  Paquetes: enviados = 4, recibidos = 4, perdidos = 0
  (0% perdidos),
  Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 161ms, Máximo = 389ms, Media = 283ms

C:\Users\marle>ping 100.93.134.60

Haciendo ping a 100.93.134.60 con 32 bytes de datos:
Respuesta desde 100.93.134.60: bytes=32 tiempo=549ms TTL=128
Respuesta desde 100.93.134.60: bytes=32 tiempo=142ms TTL=128
Respuesta desde 100.93.134.60: bytes=32 tiempo=96ms TTL=128
Respuesta desde 100.93.134.60: bytes=32 tiempo=106ms TTL=128

Estadísticas de ping para 100.93.134.60:
  Paquetes: enviados = 4, recibidos = 4, perdidos = 0
  (0% perdidos),
  Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 96ms, Máximo = 549ms, Media = 223ms
```

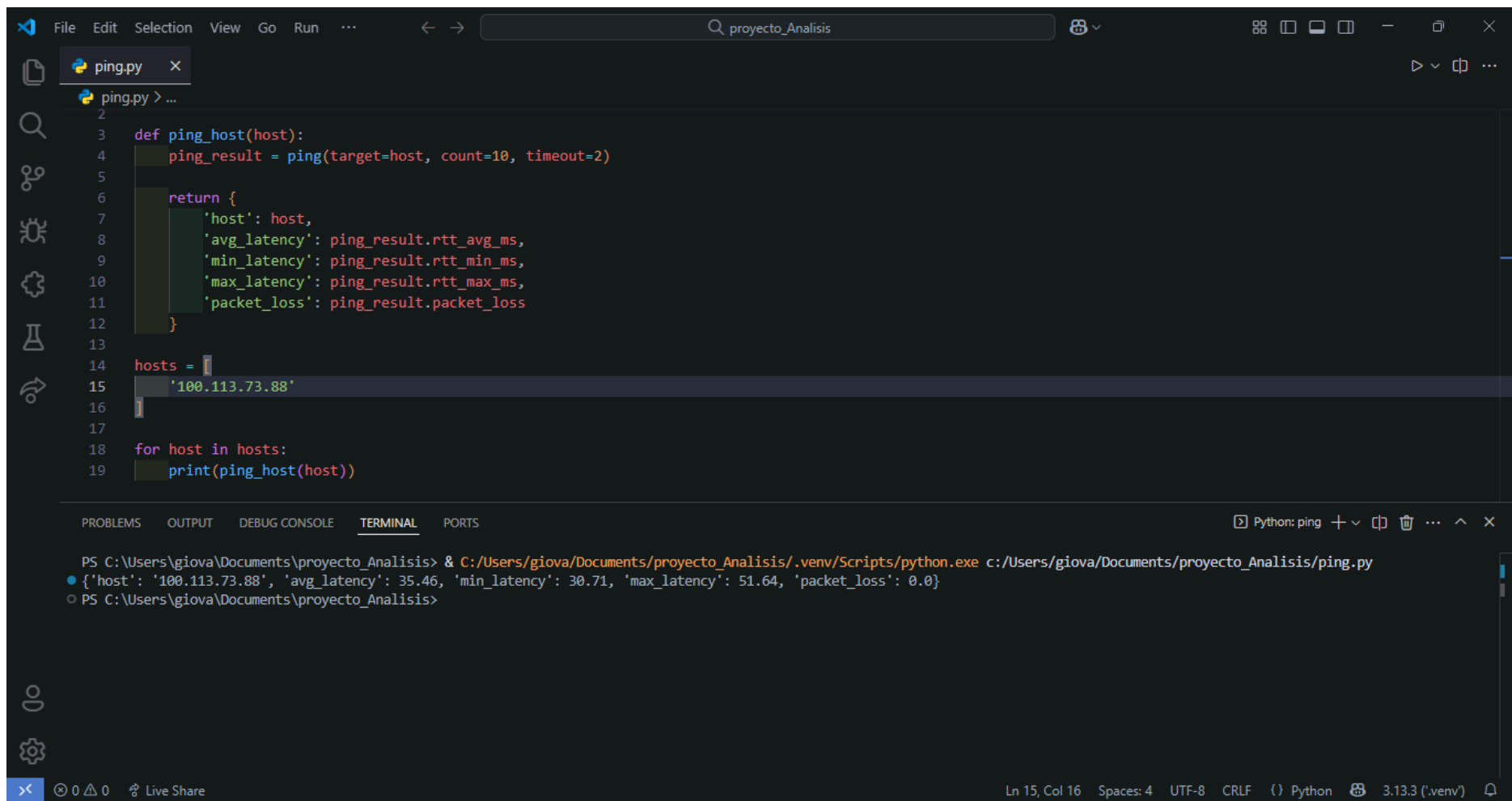
Img 2. Resultados de la consulta ping.



Img 3. Grafo ponderado con la latencia / ancho de banda.

	ANDRÉS	ALAN	MARLENE	GIOVANNI
ANDRÉS		Ping:32ms Ancho de Banda: 29.1 y 24.3 Mbits/sec	Ping: 125ms Ancho de Banda: 3.12 y 2.29 Mbits/sec	Ping:89ms Ancho de Banda: 7.88 y 6.52
ALAN	Ping: 69ms Ancho de Banda:8.25 y 6.86 Mbits/sec		Ping: 137ms Ancho de Banda: 3.38 y 2.80 Mbits/sec	Ping: 50ms Ancho de Banda: 22.8 y 19.0 Mbits/sec
MARLENE	Ping: 283ms Ancho de Banda: 3.25 y 2.67 Mbits/sec	Ping: 336ms Ancho de Banda: 7.62 y 6.27 Mbits/sec		Ping: 223ms Ancho de Banda: 4.88 y 3.98 Mbits/sec
GIOVANNI	Ping: 162ms Ancho de Banda: 8.50 y 7.02 Mbits/sec	Ping:89ms Ancho de Banda: 32.8 y 27.1 Mbits/sec	Ping: 129ms Ancho de Banda: 2.50 y 2.10 Mbits/sec	

Tab 1. Métricas



The image shows a Visual Studio Code editor window with a file named `ping.py` open. The code defines a function `ping_host` that takes a `host` parameter and returns a dictionary with ping statistics. The `hosts` list contains the IP address `'100.113.73.88'`. The script is executed from the terminal, and the output shows the ping statistics for the specified host.

```
1 ping.py > ...
2
3 def ping_host(host):
4     ping_result = ping(target=host, count=10, timeout=2)
5
6     return {
7         'host': host,
8         'avg_latency': ping_result.rtt_avg_ms,
9         'min_latency': ping_result.rtt_min_ms,
10        'max_latency': ping_result.rtt_max_ms,
11        'packet_loss': ping_result.packet_loss
12    }
13
14 hosts = [
15     '100.113.73.88'
16 ]
17
18 for host in hosts:
19     print(ping_host(host))
```

Terminal Output:

```
PS C:\Users\giova\Documents\proyecto_Analisis> & C:/Users/giova/Documents/proyecto_Analisis/.venv/Scripts/python.exe c:/Users/giova/Documents/proyecto_Analisis/ping.py
• {'host': '100.113.73.88', 'avg_latency': 35.46, 'min_latency': 30.71, 'max_latency': 51.64, 'packet_loss': 0.0}
○ PS C:\Users\giova\Documents\proyecto_Analisis>
```

Img 4. ping de latencia

The image shows a Visual Studio Code editor window with a Python script named `cliente_banda.py` and its terminal output. The script defines a function `medir_ancho_de_banda` that uses `subprocess.run` to execute `iperf3` on a remote server. The terminal output shows the results of four test intervals, followed by a summary table and the message "iperf Done."

```
def medir_ancho_de_banda(ip_servidor, duracion=10):
    try:
        print(f"Iniciando prueba de ancho de banda hacia {ip_servidor} por {duracion} segundos...\n")

        # Ejecutar iperf3 con la ruta específica
        resultado = subprocess.run(
            ["C:\\Users\\giova\\Desktop\\iperf3.18_64\\iperf3.18_64\\iperf3.exe", "-c", ip_servidor, "-t", str(duracion)],
            capture_output=True, text=True
        )

        # Mostrar la salida
        print(resultado.stdout)
        if resultado.stderr:
            print("Errores:")
            print(resultado.stderr)

    except FileNotFoundError:
        print("No se encuentra iperf3 en la ruta proporcionada o hay un error al ejecutar el comando.")
```

Terminal Output:

```
[ 5] 6.01-7.01 sec 2.25 MBytes 19.0 Mbits/sec
[ 5] 7.01-8.01 sec 2.50 MBytes 20.8 Mbits/sec
[ 5] 8.01-9.01 sec 2.50 MBytes 21.0 Mbits/sec
[ 5] 9.01-10.00 sec 2.25 MBytes 19.1 Mbits/sec
-----
[ ID] Interval          Transfer      Bitrate
[ 5] 0.00-10.00 sec 25.5 MBytes 21.4 Mbits/sec
[ 5] 0.00-10.05 sec 25.1 MBytes 21.0 Mbits/sec

iperf Done.
```

ID	Interval	Transfer	Bitrate	sender	receiver
5	6.01-7.01 sec	2.25 MBytes	19.0 Mbits/sec		
5	7.01-8.01 sec	2.50 MBytes	20.8 Mbits/sec		
5	8.01-9.01 sec	2.50 MBytes	21.0 Mbits/sec		
5	9.01-10.00 sec	2.25 MBytes	19.1 Mbits/sec		
5	0.00-10.00 sec	25.5 MBytes	21.4 Mbits/sec		
5	0.00-10.05 sec	25.1 MBytes	21.0 Mbits/sec		

Img 5. Métricas de ancho de banda.

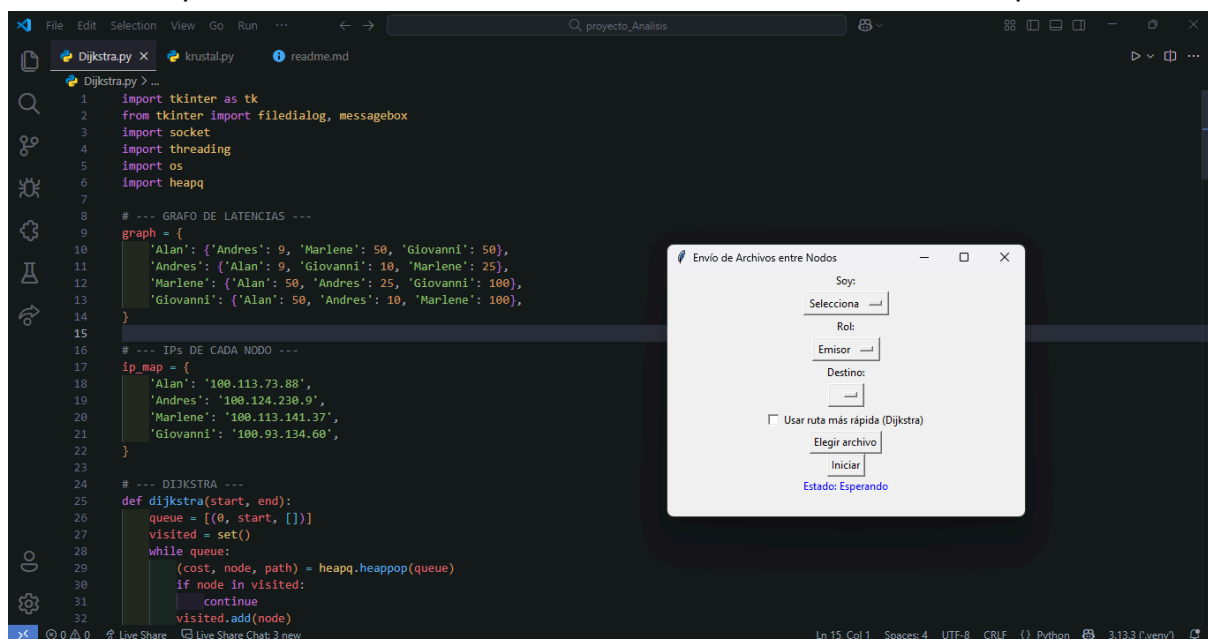
Paso 3: Implementación de Dijkstra

Para comenzar con la implementación tuvimos que hacer uso del grafo de latencias creado anteriormente, que fue el encargado de darnos la pauta para la elaboración.

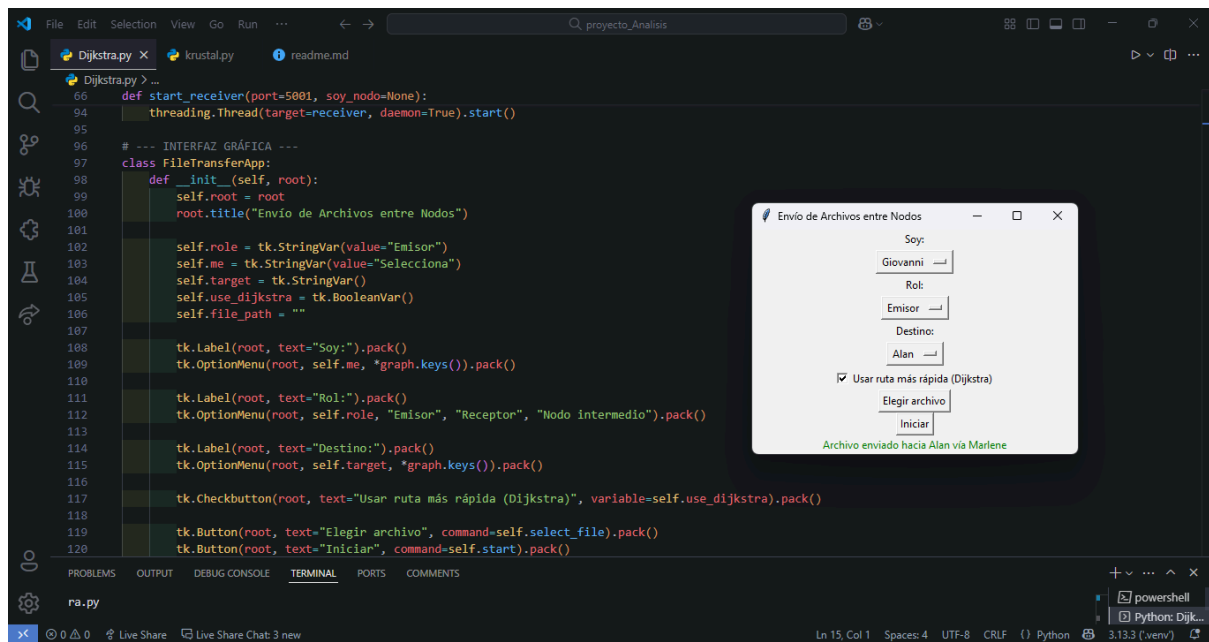
Por medio del uso de Dijkstra, el programa se encarga de encontrar la ruta más óptima entre dos nodos para enviar un archivo, siendo capaz de utilizar nodos intermedios (dispositivos) formando así, una ruta más eficiente al momento de mandar un archivo. Asi mismo, el programa es capaz de enviar archivos directamente sin necesidad de pasar por nodos intermedios, dándole la opción al usuario por medio de una interfaz gráfica de seleccionar quien es el origen, el rol que tiene, el destino al que quiere llegar, además, de poder seleccionar el archivo de su preferencia.

Probando el algoritmo, nos dimos cuenta que el programa no está del todo diseñado para archivos demasiado grandes, por lo que se recomienda, hacer envío de archivos pequeños, esto con el fin de garantizar un correcto funcionamiento del programa.

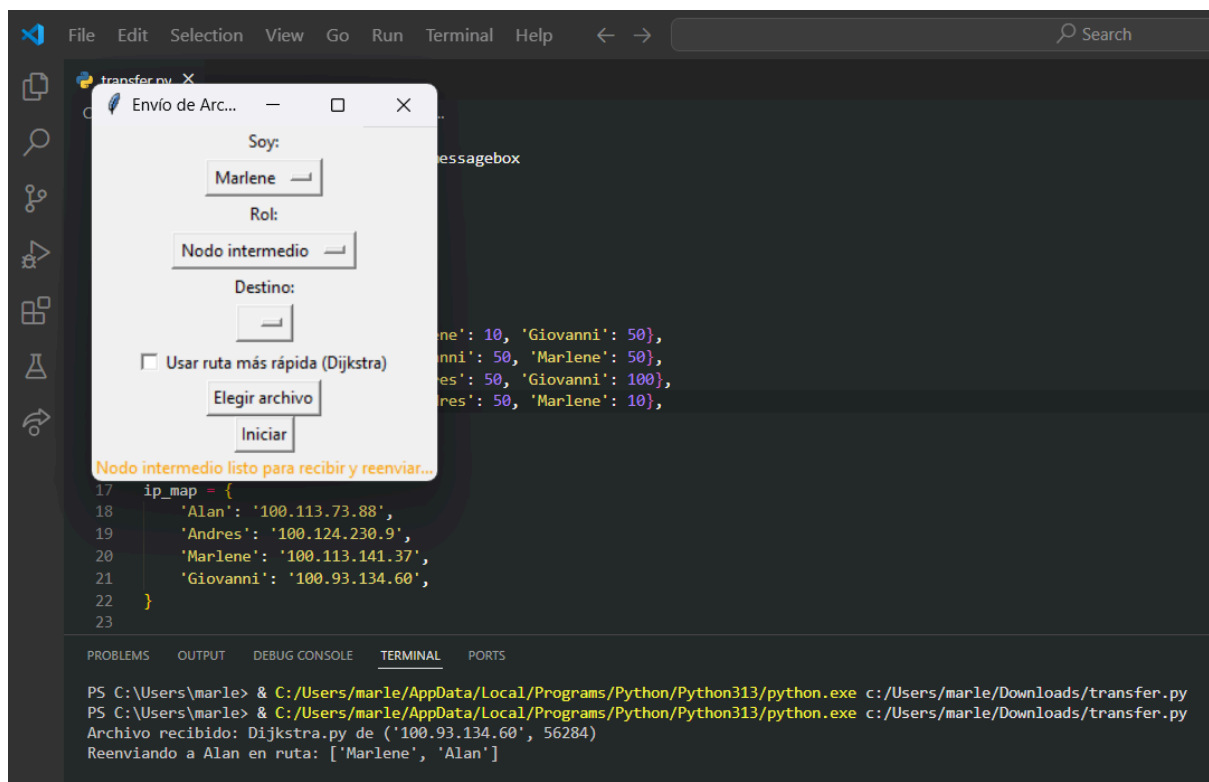
Esta es una prueba de cómo funciona el envío buscando la ruta más rápida:



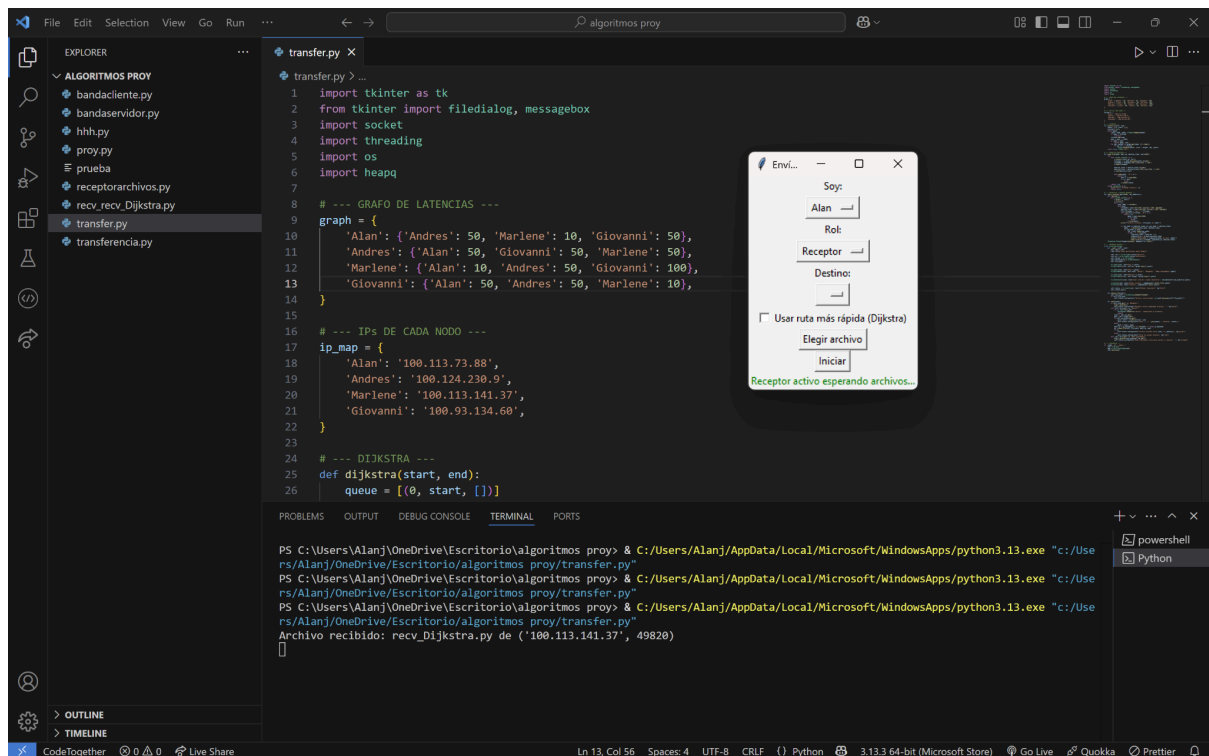
Vista del Emisor:



Vista del nodo intermedio:



Vista del Receptor:



Paso 4: Implementación de Kruskal ("Topología Eficiente")

Kruskal es un algoritmo clásico de algoritmos de grafos que se usa para encontrar el árbol de expansión mínima de un grafo conectado, no dirigido y ponderado.

Este proyecto implementa el algoritmo de Kruskal para encontrar un Árbol de Expansión Mínima (MST) en una red representada como grafo ponderado. El objetivo es optimizar el uso de la red basándose en los valores de ancho de banda entre los nodos.

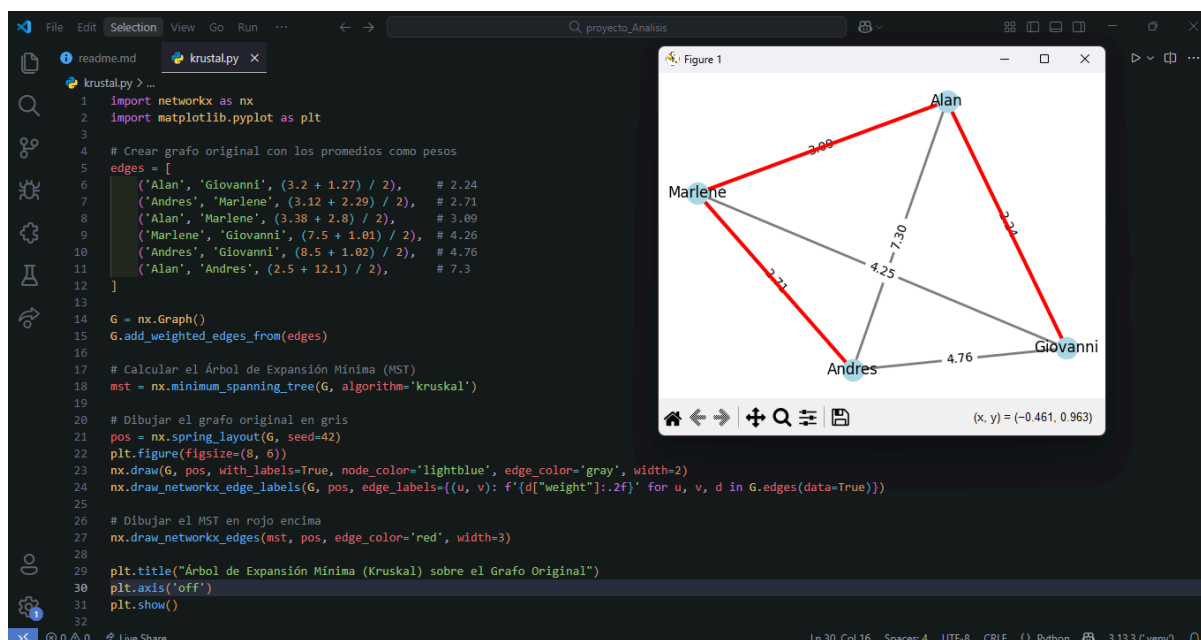
el grafo de ancho de banda con las conexiones entre 4 nodos (personas) las cuales son: **Alan**, **Andres**, **Marlene** y **Giovanni**

y cada arista tiene dos valores los cuales son:

Valor superior: Ancho de banda de subida (Mbps)

Valor inferior: Ancho de banda de bajada (Mbps)

Una vez que identificamos los valores del grafo, para los cálculos se tomó el promedio entre subida y bajada



Topología:

Enlace	Subida	Bajada	Promedio
Alan - Andrés	2.5	2.1	2.30
Alan - Marlene	3.38	2.8	3.09
Alan - Giovanni	3.2	1.27	2.24
Andrés - Marlene	3.12	2.29	2.71
Andrés -Giovanni	1.8	1.02	1.41
Marlene -Giovanni	3.5	1.02	2.26
Costo total :			13.99

Tab 2.Comparación entre la topología original con la propuesta por Kruskal

Enlace	Promedio
Andrés - Giovanni	1.41
Alan - Giovanni	2.24
Andrés - Marlene	2.71
Costo total del MST:	6.36

Tab 3. Árbol de Expansión Mínima (MST) con Kruskal

Topología	Costo Total
Original	13.99
Árbol (Kruskal)	6.36
Ahorro total	7.23 Mbps en promedio de ancho de banda