
# Actividad 6: Optimización de Transferencia de Archivos en una VPN con Algoritmos Voraces

**Equipo: Polinesios**

José Eloy Hernández Olivera **Rol:** VPN.

Sebastián Salcido Solorio **Rol:** Dijkstra, Kruskal.

Arlette Michelle Castañeda Solís **Rol:** Administradora/ Dijkstra

Leonardo Axel Nochebuena **Rol:** Mediciones de red.

## Introducción

En la actualidad, la eficiencia en la transferencia de archivos a través de redes privadas virtuales (VPN) es un aspecto crucial para garantizar un rendimiento óptimo en entornos distribuidos. Este proyecto tiene como objetivo optimizar el proceso de transferencia de archivos en una VPN utilizando algoritmos voraces, enfocándose particularmente en las métricas de latencia y ancho de banda de la red. Para establecer la VPN se emplea **Tailscale**, una solución basada en WireGuard que permite una configuración rápida y segura de redes privadas entre dispositivos.

Como parte del análisis y optimización, se realizarán mediciones de red mediante el comando ping para obtener métricas de latencia entre los nodos de la VPN. Con estos datos se construirá un grafo de latencias, el cual servirá como entrada para el algoritmo de Dijkstra. Este algoritmo se implementará con el propósito de determinar la ruta más rápida entre dos nodos para la transferencia de archivos, mejorando significativamente los tiempos de envío. Además se implementará el algoritmo de Kruskal sobre un grafo basado en el ancho de banda disponible entre los nodos.

## Objetivo general

Optimizar la transferencia de archivos en una red VPN implementada con Tailscale, utilizando algoritmos voraces como Dijkstra y Kruskal para mejorar el rendimiento de la red en términos de latencia y ancho de banda, mediante la construcción de grafos de red, selección de rutas óptimas y comparación de topologías.

**Objetivo 1: Implementar algoritmos voraces en una red VPN funcional.**

**KR1.1**: Configurar una VPN operativa con al menos 4 nodos activos usando Tailscale.
**KR1.2**: Validar la conectividad mediante pruebas de ping entre todos los nodos (latencia aceptable).
**KR1.3**: Establecer la transferencia básica de archivos entre nodos sin errores de red.

**Objetivo 2: Optimizar la transferencia de archivos utilizando el algoritmo de Dijkstra.**

**KR2.1**: Construir un grafo de latencias a partir de mediciones reales entre nodos.
**KR2.2**: Implementar el algoritmo de Dijkstra para calcular rutas mínimas en tiempo de transferencia.
**KR2.3**: Integrar la lógica de Dijkstra con la transferencia real de archivos desde una GUI.

**KR2.4**: Transferir archivos de prueba (10 MB, 100 MB, 1 GB) utilizando rutas óptimas y registrar métricas.

**Objetivo 3: Evaluar y optimizar la topología de red con el algoritmo de Kruskal.**

**KR3.1:** Medir el ancho de banda disponible entre nodos y construir el grafo correspondiente.

**KR3.2:** Implementar el algoritmo de Kruskal para generar un Árbol de Expansión Mínima (MST).

**KR3.3:** Comparar la topología real con la optimizada (MST) en términos de eficiencia de red.

---

## ¿Qué es una vpn?

**Una VPN (Red Privada Virtual)** es una tecnología que permite establecer una conexión privada y segura entre dispositivos a través de Internet. Se utiliza para **transmitir datos de forma confidencial y anónima**, incluso cuando se usa una red pública.

El funcionamiento de una VPN se basa en dos principios clave:

1. **Cifrado de datos:** toda la información enviada entre los dispositivos viaja de forma codificada, evitando que pueda ser interceptada o leída por terceros no autorizados.

2. **Ocultamiento de IP:** la dirección IP real de los dispositivos se oculta, lo que protege la identidad del usuario y hace que el tráfico parezca provenir de otra ubicación o red.

---

## Tipos de metodologías:

- **Scrum:** Se basa en ciclos cortos de trabajo llamados "sprints" donde los equipos se organizan y trabajan de forma autogestionada para entregar valor de forma incremental.
- **Kanban:** Método muy visual muy utilizado en la gestión ágil de proyectos. Muestra una imagen del proceso de trabajo, que permite ver posibles cuellos de botella en el desarrollo, que permite entregar un producto con calidad y a tiempo. La estructura Kanban más sencilla cuenta con un panel con 3

columnas en las que irán moviéndose las tareas: Pendiente / Haciendo / Completado.

- **Extreme Programming (XP):** Está muy centrada en la satisfacción del cliente. Busca entregar al cliente lo que necesita ahora mismo de forma rápida, sin pensar en todo lo que podría necesitar en un futuro más lejano. La metodología XP se centra en lanzamientos frecuentes y ciclos de desarrollo cortos, a la vez que se apoya en una comunicación frecuente con el cliente.
- **Lean Software Development (LSD):** Se enfoca en la eficiencia y la eliminación de desperdicios. Su objetivo es maximizar el valor entregado al cliente mientras minimiza recursos innecesarios. El proceso se ajusta continuamente para optimizar el flujo y la eficiencia.
- **Feature Driven Development (FDD):** Construcción de características específicas del sistema de manera iterativa y detallada. El equipo se enfocará en construir y refinar esta característica antes de pasar a la siguiente.
- **Dynamic Systems Development Method (DSDM):** Es una metodología iterativa ágil que se enfoca en la entrega rápida y flexible de soluciones de software, proporcionando un conjunto de principios y prácticas sólidas para el desarrollo de sistemas. La filosofía DSDM es una versión modificada de un principio sociológico: el 80% del proyecto normalmente se entrega en el 20% del tiempo de entrega de la aplicación o el servicio entero. De esta forma, se realizan iteraciones basándose en este principio, donde el 20% de trabajo restante se completa más tarde, junto con otros cambios detectados.
- **Adaptive Software Development (ASD):** El DSA es un proceso de desarrollo de software que surgió del trabajo de Jim Highsmith y Sam Bayer sobre el desarrollo rápido de aplicaciones (RAD).Se centra en la colaboración humana y en la autoorganización prioriza la adaptabilidad para responder a requisitos y entornos cambiantes, centrándose en el aprendizaje continuo y se considera parte del desarrollo de software ágil.
- **Crystal:** Es una metodología ágil que se enfoca en las interacciones y el trabajo en equipo, adaptándose a las necesidades de cada proyecto. Es más flexible que otros métodos. Sus características son: - Entregas frecuentes, en base a un ciclo de vida iterativo e incremental. - Mejora reflexiva. - Comunicación osmótica. - Seguridad personal. - Enfoque. - Fácil acceso a usuarios expertos. - Entorno técnico con pruebas automatizadas, gestión de la configuración e integración continua. Suele ser bueno para proyectos grandes en los que un fallo pueda causar mayores problemas.

---

## Tipos de combinaciones

- **Scrumban:** Combina los ciclos de sprint de Scrum con la visualización y el flujo de trabajo continuo de Kanban.

- **Agile Hybrid:** Integra la planificación tradicional de Waterfall con la flexibilidad y la iteración de las metodologías ágiles, como Scrum o Kanban.
- **Scrum + Kanban:** Se usa Scrum para la planificación y organización del trabajo en sprints, y Kanban para el flujo de trabajo y la gestión de tareas en los sprints.
- **Scrum + Lean**: Se aplica Lean para eliminar desperdicios y optimizar procesos dentro de la estructura de Scrum.

---

## Cuál es metodología es la mejor para este proyecto:

En este caso dado el tiempo y el tamaño del equipo elegimos la metodología **scrum**. Planeamos hacer 3 sprints hoy 30 de abril y siendo el 13 de mayo el último día de sprint para el 14 de mayo poder entregar el proyecto.

Creemos que es muy buena esta metodología para el proyecto, ya que funciona mediante (Sprints), ciclos cortos y entregas incrementales. Además de que tiene autoasignación y roles claros, lo cual es fundamental para la realización de este proyecto. También tiene un seguimiento claro del avance y documentación. lo cual requerimos para este proyecto, así que nos damos cuenta de que esta metodología se adapta perfecto a nuestras necesidades y las del proyecto, ya que gracias a los Sprints podemos tener una revisión y mejora continua.

## Metodología Scrum

### 1. *Producto backlog*

En esta primera fase de la metodología Scrum se identificaron las necesidades clave del proyecto, se definieron las características esenciales, y se establecieron las primeras tareas necesarias para el desarrollo. El equipo asumió que las prioridades podrían ajustarse durante el proceso, permitiendo flexibilidad ante cambios técnicos o funcionales. A continuación, se presenta el backlog inicial:

| Tarea | Prioridad | Responsables(s) | Estado Inicial de la tarea |
|---|---|---|---|
| Definir alcance del proyecto y objetivos | Alta | Todo el equipo | Completada |
| Elegir Scrum Master y asignar roles secundarios | Alta | Todo el equipo | Completada |

| | | | |
|---|---|---|---|
| Configurar VPN con Tailscale | Alta | Eloy | En desarrollo |
| Medir latencias entre nodos (ping) | Alta | Leonardo | En desarrollo |
| Medir anchos de banda entre nodos | Media | Leonardo | Pendiente |
| Implementar grafo de latencias para Dijkstra | Alta | Michelle, Sebastian | En desarrollo |
| Implementar algoritmo de Dijkstra | Alta | Michelle, Sebastian | En desarrollo |
| Implementar grafo de ancho de banda para Kruskal | Media | Sebastian | Pendiente |
| Implementar algoritmo de Kruskal | Media | Sebastian | Pendiente |
| Probar la funcionalidad local de transferencia de archivos | Alta | Todo el equipo | Pendiente |
| Desarrollar GUI para selección de archivos y nodo destino | Media | (por asignar) | Pendiente |
| Integrar dijkstra con transferencia real de archivos | Alta | Michelle, Sebastian | Pendiente |
| Analizar topología optimizada con Kruskal y compararla con original | Media | Sebastian | Pendiente |
| Documentar avances, problemas técnicos y decisiones tomadas | Alta | Michelle | En desarrollo |
| Preparar presentación, demo final y video final de evidenci | Alta | Todo el equipo | Pendiente |

Al finalizar esta fase, se conformó el equipo con roles distribuidos según habilidades técnicas y disponibilidad. Además, consideramos que se estableció un marco de trabajo flexible que permite re-organizar tareas o prioridades según el avance y los obstáculos encontrados durante cada sprint.

**2. Sprints**

La asignación en cuanto scrum será :

- **Equipo de desarrollo:** En este caso por el tamaño del equipo seremos todos pero nos dividiremos las tareas.
- **Product owner**: Es el que revisa y define los objetivos pero para que sea de una manera justa nos iremos rotando este rol.

- **Scrum master**: Vamos a intentar que sean dos personas donde 1 será la que asignaremos como administrador (Michelle) y el otro nos iremos rotando ya que al final así podremos repartir la carga de trabajo, pero en específico será Michelle la principal que coordine las tareas, reuniones, etc.
- **DevOps / Infraestructura:** Eloy será responsable de la configuración de la VPN y pruebas de conectividad. Supervisó la transición de WireGuard a Tailscale y los aspectos técnicos de red.
- **Analista de Rendimiento de Red:** Leonardo realizará mediciones de latencia y ancho de banda mediante ping y otras herramientas, entregando datos clave para la construcción de los grafos.
- Algoritmos y Topología : Sebastián desarrollará el algoritmo de Kruskal, analizará la eficiencia del árbol de expansión mínima y propondrá mejoras tipológicas basadas en el ancho de banda.

### Sprint 0 (1 día – hoy, 30 de abril) – Planeación y asignación

- Definir tareas por persona según sugerencia del profe:
  Eloy(A): configuración de VPN.

  Michelle(B) y Sebastian(C): Dijkstra.

  Leonardo(D): Mediciones de red

  Sebastian(C): Kruskal.
  Elegir Scrum Master(quien lleve la bitácora y coordine entregas).

### Sprint 1 (1 al 6 de mayo) – Desarrollo funcional

- VPN operativa (Persona A)
- Implementación base de Dijkstra (B y C)
- Implementación base de Kruskal (D)
- Pruebas locales entre dispositivos
- Daily meetings (5 minutos por Discord, WhatsApp, etc.)

**Entregables Sprint 1:**

- VPN funcional con mínimo 2 dispositivos conectados.
- Dijkstra aplicado a rutas de prueba.
- Kruskal con topología mínima.
- Primera prueba de transferencia con log.

**Sprint 2 (7 al 10 de mayo) – Integración y pruebas finales**

- Integrar Dijkstra con la transferencia real de archivos.
- Ajustar la topología con Kruskal.
- Medir mejoras reales en velocidad o eficiencia.
- Documentación, gráficas, presentación.

**Sprint 3 Review y entrega (11 de mayo)**

- Presentación al profesor (topología, métricas, resultados).
- Demo de transferencia optimizada.
- Documentación final.

**Post-Sprint – Correcciones y entrega definitiva (12 y 13 de mayo)**

**12 de mayo – Reunión final con el profesor:**
Se revisó y corrigió el contenido de la presentación.

- Se recibieron observaciones sobre el documento.
- El profesor dio recomendaciones para resolver los problemas con la transferencia de archivos.

**12–13 de mayo – Implementación de mejoras:**

- **Corrección del código** para adecuarlo a los comentarios.
- **Actualización de la presentación** con todos los elementos requeridos.
- **Modificación del documento final** con base en la retroalimentación.
- **Grabación de video demostrativo** mostrando el funcionamiento del sistema y la transferencia de archivos.

Dado el proyecto y que puede llegar a ser complejo la idea es poder hacer 1 daily meeting por día, esto para ir viendo nuestro progreso y en caso de tener problemas en alguna parte de codificaciones poder ayudarnos entre todos para así dar los entregables que corresponde al día.

**3. Fase de control (Burn Down)**

Una vez finalizadas las tareas planificadas y presentadas las funcionalidades desarrolladas, se dio inicio a la fase de control del proyecto. Esta etapa tuvo como

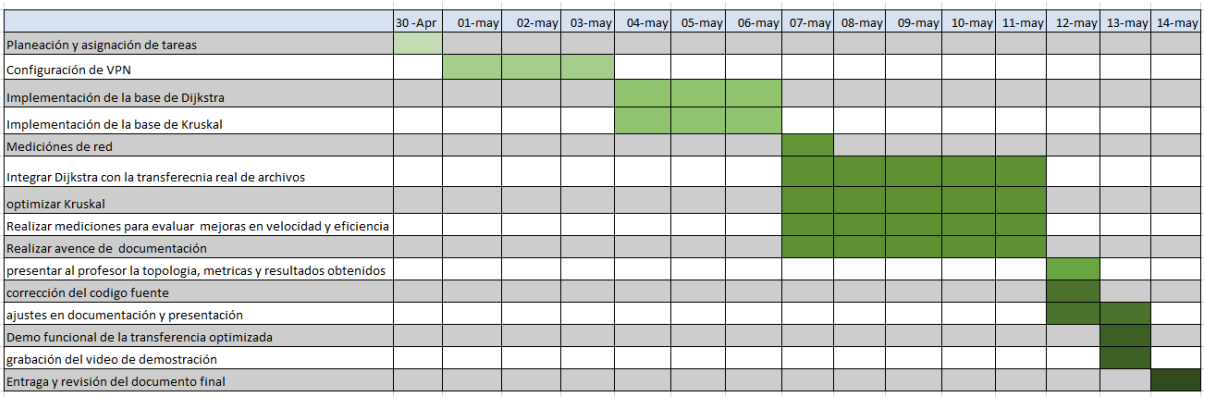
objetivos principales: verificar que todo estuviera en orden y reflexionar sobre el trabajo realizado. Durante el proceso, se llevaron a cabo reuniones periódicas con el equipo para revisar avances, resolver obstáculos y considerar ajustes necesarios.

En el último meeting, se identificaron varios problemas críticos, entre ellos el más significativo: los archivos no se enviaban correctamente entre los nodos. Ante esta situación, el equipo se dio a la tarea de revisar documentación técnica, investigar en fuentes oficiales y buscar ayuda externa para comprender adecuadamente el funcionamiento de la transferencia de archivos en la red Tailscale. Este proceso nos permitió entender las limitaciones del entorno y plantear posibles soluciones o alternativas viables.

Además, se detectó una falta de comunicación interna, que provocó ciertos desfases en la sincronización de tareas. Para remediarlo, se convocó a una reunión presencial final, en la que se discutieron los objetivos generales del proyecto, se aclararon dudas y se realizó una autocrítica constructiva sobre los puntos de mejora como equipo. Esta reunión fue clave, no solo para reforzar la coordinación y el compromiso grupal, sino también para terminar exitosamente el proyecto.

---

## Cronograma de Gantt



# Reporte de Meetings

**Kick-off  Meeting 5 mayo (duración 15 minutos)**

**Asistencia:** Eloy, Michelle, Leonardo y Sebastian

**Objetivo principal:** Definir los roles de cada integrante, los OKRs (Objetivos Clave y Resultados Esperados), así como establecer la estructura general del proyecto.

Temas tratados:

1.se termino de definir los OKRs (Objetivos Clave y Resultados) del equipo para el proyecto

2.Problemas técnicos con WireGuard:
Se intentó inicialmente la configuración de la VPN usando WireGuard, pero se encontraron dificultades considerables:

- Gestión de claves demasiado compleja.
- La configuración manual de interfaces de red y el direccionamiento IP representaron un desafío técnico significativo.
- Debido a estos obstáculos y la limitación de tiempo, la implementación con WireGuard fue considerada fallida dentro del plazo requerido.

Cambio de solución a Tailscale:
Por recomendación y facilidad de uso, se decidió utilizar Tailscale, que ofrece:

- Configuración automática de nodos.
- Gestión simplificada de identidad y encriptación.
- Ahorro considerable de tiempo en comparación con WireGuard.

**Revisión de avances Meeting 10 mayo (duración 20 minutos)**

**Asistencia:** Eloy, Michelle, Leonardo y Sebastian

**Objetivo principal:** Verificar el funcionamiento de la nueva VPN con Tailscale y revisar avances técnicos.

Temas tratados:

- Verificación de funcionamiento de la VPN (primeros dos nodos conectados).
- Se reportaron avances en la estructura del algoritmo de Dijkstra.
- Se mostró resultados iniciales de latencias
- No se presentó ningún obstáculo crítico durante esta reunión, aunque se notaba desincronización en avances individuales.

Decisiones/ soluciones tomadas:

- Seguir avanzando con Tailscale como base de la red.
- Centralizar resultados y mediciones para construir el grafo de red.

Lecciones aprendidas:

- Cuando la tecnología elegida es fácil de implementar, el equipo pudo enfocarse más rápido en los objetivos funcionales del proyecto.
- La medición temprana del rendimiento (latencia) es clave para guiar decisiones sobre algoritmos y rutas óptimas.

**Última reunión presencial: Retrospectiva Final 12 mayo ( duración de 90 minutos)**

**Asistencia:** Eloy, Michelle, Leonardo, Sebastian + profesor

**Objetivo principal:** Consolidar la fase final del proyecto, demostrar funcionalidad y resolver pendientes.

Temas tratados:

Problemas en la transferencia de archivos a través de Tailscale

- Se discutió que no se logró realizar con éxito la transferencia de archivos entre dispositivos conectados a la red Tailscale.
- Se identificó que el problema probablemente se relacionaba con el desconocimiento del flujo real de envío/recepción dentro del entorno VPN

Solución:

- Se acordó leer documentación oficial de Tailscale, investigar foros y buscar ejemplos funcionales de implementación de transferencia de archivos dentro de una red

Lecciones aprendidas:

- Aunque la VPN se configure correctamente, la transferencia de archivos requiere conocimientos adicionales sobre el flujo real de datos.
- La comunicación efectiva entre integrantes es esencial para evitar pérdidas de tiempo y malentendidos.

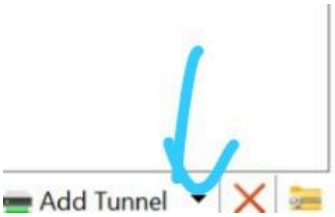
---

# Reportaje de la vpn

1- Crear una VPN **(WireGuard)**

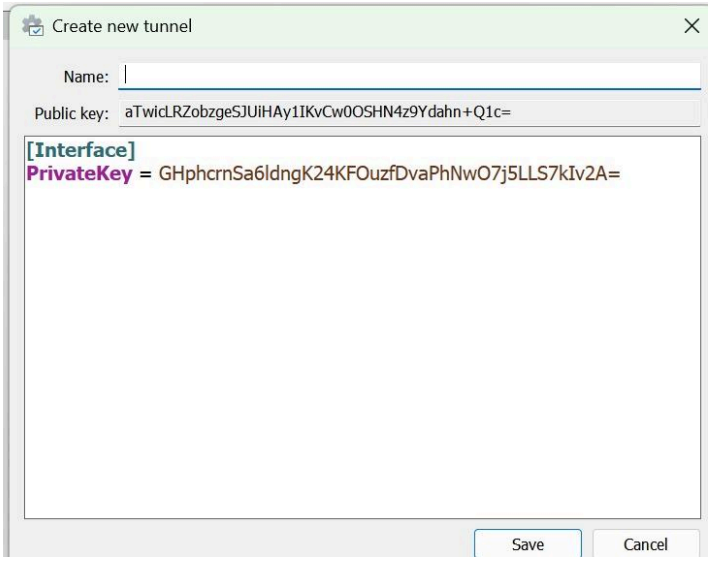
- Creación del servidor
- Creación de los usuarios

Una vez dentro de WireGuard necesitamos seleccionar esta opción:

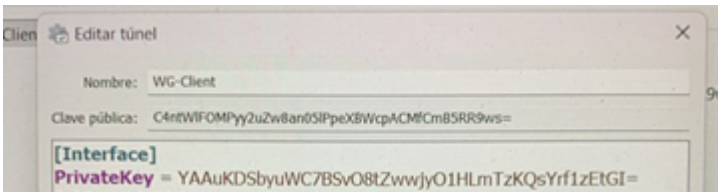



Una vez que seleccionamos eso nos aparecerá el siguiente recuadro:


Le daremos a “Add empty tunnel…” para que nos salga lo siguiente:




Así es como se vería el recuadro del cual, necesitamos sacar nuestra “public key” la cual la podremos obtener de la siguiente manera:

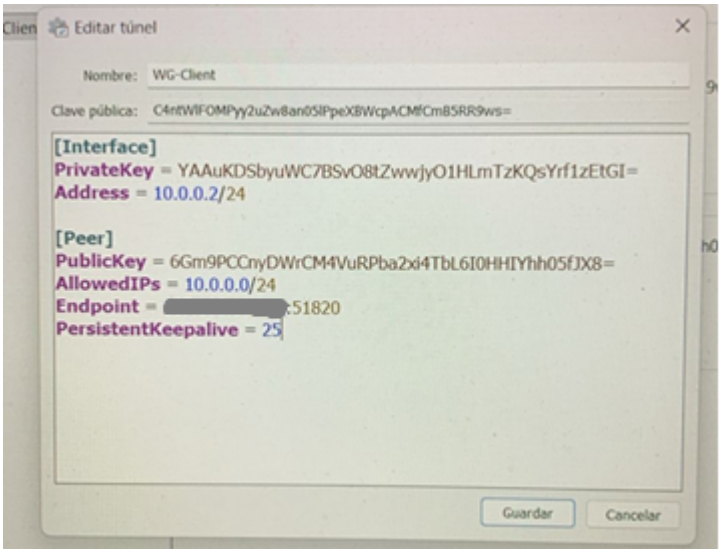



Una vez que le damos clic al tunnel que hemos creado nos aparece la información de la imagen, de la cual tendremos la clave pública y la clave privada.

La clave pública o “public key” se la tendremos que proporcionar a nuestro compañero encargado de crear el servidor.

Una vez que ya hayamos hecho, nuestro compañero nos proporcionará una la public key del servidor, una dirección ip y un listen port, con esta información es más que suficiente para terminar de configurar al cliente del servidor. Para esto necesitaremos además nuestra ip, la cual la podemos consultar desde la siguiente página web: https://www.whatismyip.com/

Una vez que ya hayamos agregado toda esta información, se vería algo así la configuración del usuario.




Solo para tener un mejor control y una “copia de seguridad” lo que haremos será guardar esta configuración en un archivo .zip, lo haremos dándole clic a la carpeta que está a la derecha de “Add Túnel” en la parte inferior izquierda y ya solo seleccionamos en donde queremos que se guarde y le ponemos nombre, con todo esto ya habremos terminado con el usuario.

**NOTA:**
Después de evaluar diversas opciones para establecer una red privada virtual entre los miembros de mi equipo, incluyendo WireGuard, la implementación con Tailscale demostró ser la solución más eficiente y accesible. Inicialmente, intenté configurar una VPN utilizando WireGuard, pero la complejidad inherente en la gestión de claves, la configuración de interfaces de red y el direccionamiento IP representaron un desafío significativo, resultando en una implementación fallida dentro del plazo requerido.

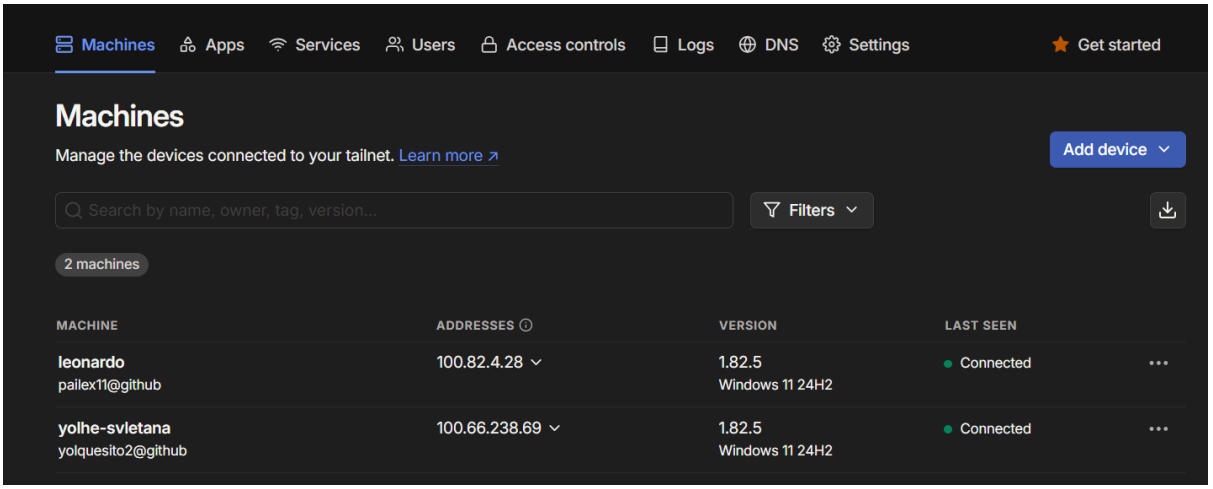
En contraste, Tailscale ofreció una experiencia notablemente más intuitiva. El proceso se centró en la instalación de la aplicación en los dispositivos de cada

integrante y la autenticación mediante cuentas de usuario (en nuestro caso, cuentas de GitHub para mantener la coherencia). Una vez que cada miembro inició sesión con su respectiva cuenta, la red privada virtual se estableció de manera automática, sin la necesidad de una configuración manual exhaustiva de parámetros de red o la manipulación de la configuración del router para eludir NAT o firewalls.

Desde un punto de vista técnico, Tailscale actúa como una capa de gestión sobre el protocolo WireGuard. Se encarga de establecer túneles cifrados de punto a punto entre dispositivos registrados bajo la misma cuenta o dominio. Además, maneja automáticamente el descubrimiento de nodos, la conexión a través de NATs, y el enrutamiento interno, proporcionando seguridad de extremo a extremo sin necesidad de configuración avanzada.

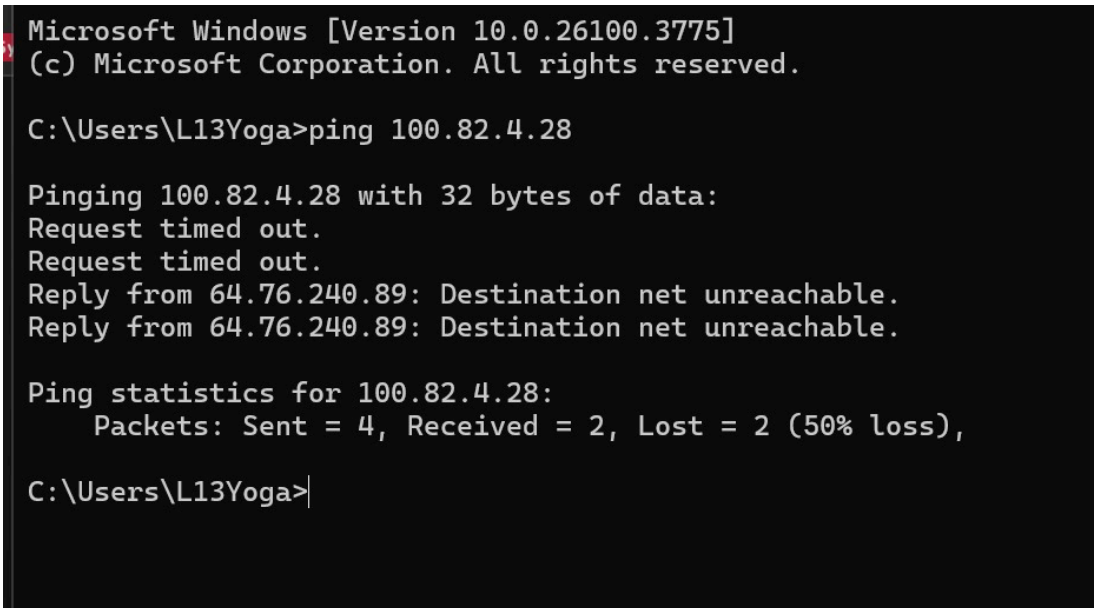
Además de la facilidad de configuración, Tailscale proporcionó una funcionalidad integrada crucial para nuestro objetivo de compartir archivos: Taildrop. Esta característica permitió el envío seguro y directo de archivos entre los nodos de la red con una operación simple de arrastrar y soltar, simplificando significativamente la implementación de un protocolo de transferencia de archivos en comparación con la necesidad de configurar servicios adicionales sobre una VPN tradicional.

En conclusión, la elección de Tailscale se fundamentó en su simplicidad de uso, su capacidad para manejar automáticamente los desafíos de conectividad en redes diversas y la inclusión de herramientas que facilitaron directamente nuestros requerimientos de colaboración y transferencia de archivos, superando las dificultades encontradas con la configuración manual de WireGuard.




En esta imagen podemos observar un poco de la interfaz de la vpn que se utilizó. Podemos observar que hay por el momento 2 dispositivos, el servidor (Leonardo) y el cliente (Eloy).

Ping

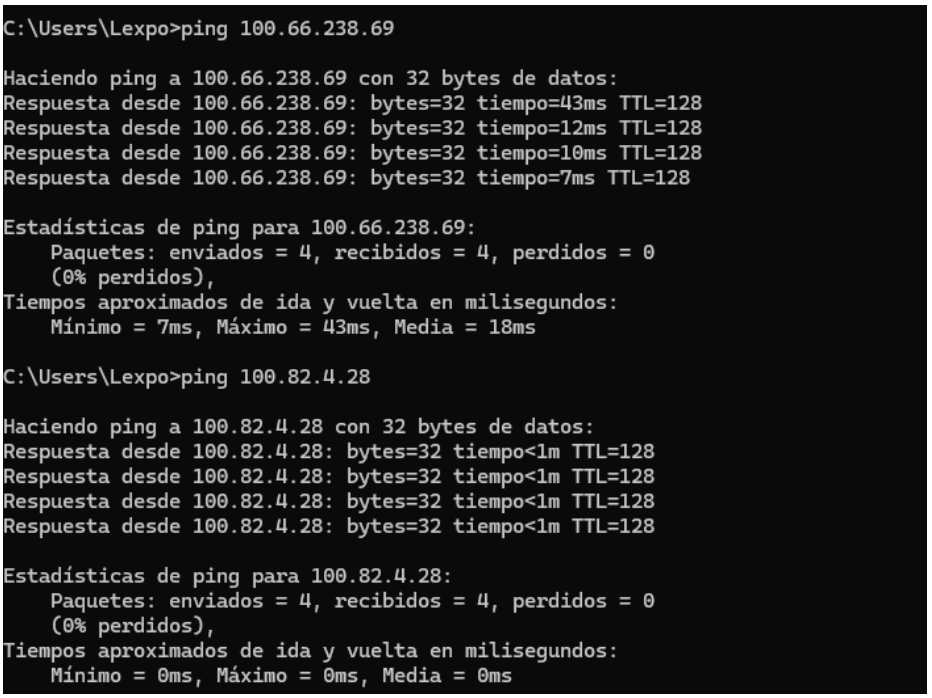



Aquí podemos ver cómo es que el primer ping de prueba trazó el camino para que el segundo ping llegará sin problemas.

## Mediciones de comunicación:

### Método utilizado para la medición

Utilizamos el método de ping porque es más rápido y efectivo que un script o por lo menos en nuestro caso fue más sencillo, obteniendo así los siguientes resultados:




Aquí podemos ver cómo es que los siguientes pings ya pudieron ser más rápidos, ya que el camino ya estaba trazado por el primer ping. Es por eso que se enviaron y recibieron los 4 paquetes con éxito.

```
C:\Users\Lexpo>ping 100.76.82.26

Haciendo ping a 100.76.82.26 con 32 bytes de datos:
Respuesta desde 100.76.82.26: bytes=32 tiempo=73ms TTL=64
Respuesta desde 100.76.82.26: bytes=32 tiempo=141ms TTL=64
Respuesta desde 100.76.82.26: bytes=32 tiempo=95ms TTL=64
Respuesta desde 100.76.82.26: bytes=32 tiempo=105ms TTL=64

Estadísticas de ping para 100.76.82.26:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 73ms, Máximo = 141ms, Media = 103ms

C:\Users\Lexpo>ping 100.98.222.19

Haciendo ping a 100.98.222.19 con 32 bytes de datos:
Respuesta desde 100.98.222.19: bytes=32 tiempo=188ms TTL=64
Respuesta desde 100.98.222.19: bytes=32 tiempo=12ms TTL=64
Respuesta desde 100.98.222.19: bytes=32 tiempo=124ms TTL=64
Respuesta desde 100.98.222.19: bytes=32 tiempo=43ms TTL=64

Estadísticas de ping para 100.98.222.19:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 12ms, Máximo = 188ms, Media = 91ms
```

Aquí podemos ver cómo es que se hizo un ping a otro dispositivo, y de igual manera fue exitoso con el 0% de paquetes perdidos, así en total teniendo 4 módulos en los cuales están conectados con el VPN.

## Medición de ancho de banda (SpeedTest)

```
  Speedtest by Ookla

     Server: Totalplay - Guadalajara (id: 36942)
        ISP: Totalplay
Idle Latency:     1.49 ms   (jitter: 0.05ms, low: 1.46ms, high: 1.56ms)
   Download:    93.19 Mbps (data used: 45.6 MB)
                43.16 ms   (jitter: 1.09ms, low: 2.42ms, high: 45.08ms)
     Upload:    93.39 Mbps (data used: 42.4 MB)
               155.64 ms   (jitter: 49.49ms, low: 1.44ms, high: 218.97ms)
```

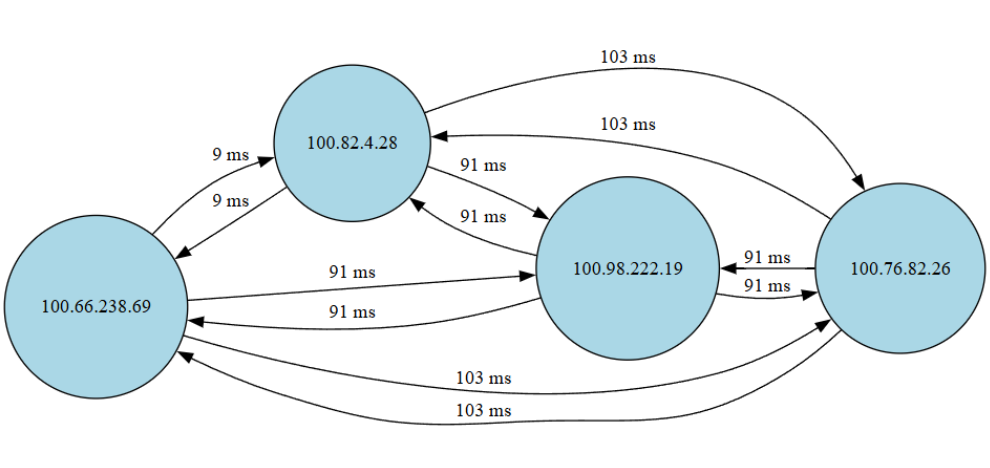
En esta imagen podemos ver la medición del ancho de banda del servidor, esto nos indica un aproximado de los tiempos de respuesta según el tamaño del archivo que enviemos, viendo así tanto la descarga como subida, latencia y ancho de banda.

| Metricas | | | | |
|---|---|---|---|---|
| **Direccion IP** | **TTL** | **Minimo (ms)** | **Maximo (ms)** | **Media (ms)** |
| 100.66.238.69 | 128 | 7 | 43 | 18 |
| 100.82.4.28 | 128 | 0 o 1 | 0 o 1 | 0 |
| 100.76.82.26 | 64 | 73 | 141 | 103 |
| 100.98.222.19 | 64 | 12 | 188 | 91 |

Aquí se recopilaron los datos de acorde a cada IP con sus respectivos pings, en los cuales nos da imagen de la latencia tanto máxima, como mínima y la media, la cual

nos sirve para poder tener un mejor análisis y compararlos con el resultados de los grafos.

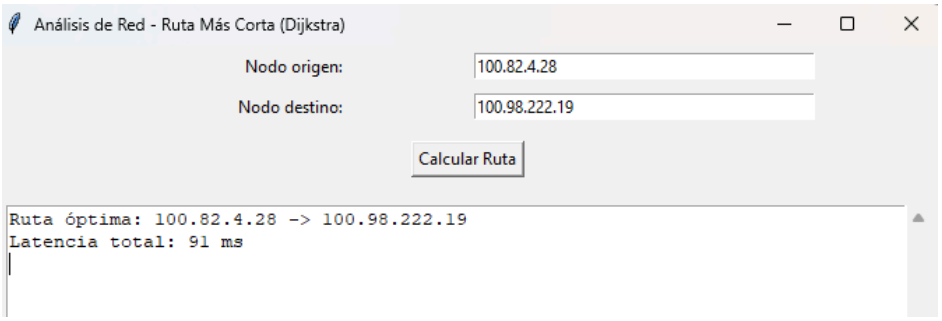
## Grafo Ponderado



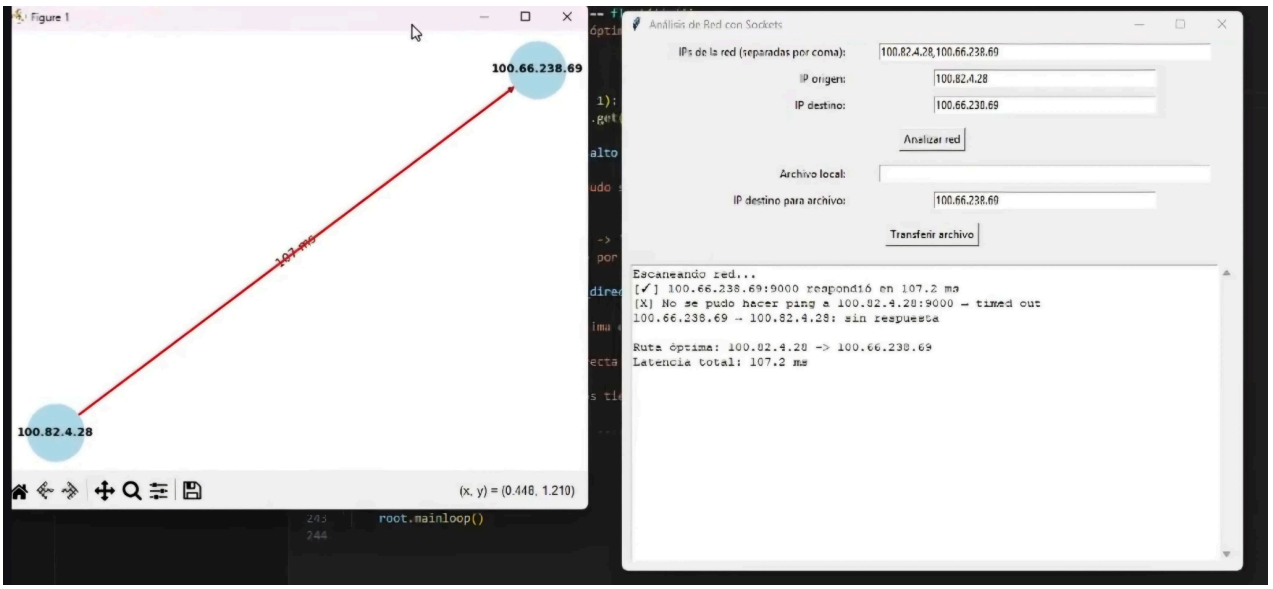

Entre estos nodos existen conexiones directas, representadas por flechas (o aristas dirigidas), que simbolizan el camino que siguen los datos al viajar de un dispositivo a otro. Cada flecha va en un solo sentido, lo cual indica que los datos viajan en una dirección específica, aunque muchos pares de nodos están conectados mutuamente en ambas direcciones.

Sobre cada flecha se indica un valor en milisegundos (ms), el cual representa la latencia de esa conexión. La latencia es el tiempo que tarda un paquete de datos en ir desde un nodo a otro. Por ejemplo, una conexión con 9 ms es muy rápida, mientras que una de 103 ms es relativamente más lenta. Este valor puede depender de varios factores como la distancia física, la calidad de los enlaces o el nivel de congestión de la red.

## Latencias (dijkstra)




En esta imagen podemos apreciar cómo es que se midió la latencia entre 2 nodos (nodo de origen y destino) del grafo generado a través de Dijkstra, en pocas palabras se realizó un ping dentro de los nodos.




(Imagen de grafo realizado al momento de la conexión con las IP de manera correcta, obteniendo así los datos)

También se realizó una mejora en el mismo código para poder mostrar el grafo de las conexiones o Ip que se realizaron en la prueba obteniendo así tanto la mejor conexión, latencia y transferencia de datos de manera correcta y efectiva.

## Código cliente

Para empezar con esta explicación, primero necesitamos importar todas las librerías necesarias para la ejecución de este código, que son las siguientes:

```python
import socket
import time
import tkinter as tk
from tkinter import messagebox, scrolledtext
import heapq
import os
import networkx as nx
import matplotlib.pyplot as plt
```

Luego empezamos con el código

```python
def ping_node(ip, port=9000):
    try:
        s = socket.socket()
        s.settimeout(1)
        start = time.time()
        s.connect((ip, port))
        s.send("ping".encode())
        resp = s.recv(1024).decode()
        s.close()
        if resp == "pong":
            end = time.time()
            return round((end - start) * 1000, 2)
    except:
        return None
```

En esta parte del código lo que sucede es que se crea un socket TCP a la IP indicada en este caso, luego se envía "ping" y espera "pong" de vuelta. Mientras mide el tiempo de ida y vuelta (RTT). Si todo va bien, devuelve la latencia en milisegundos. Si falla, retorna None.

```python
def send_file(ip, filename, port=9000, log_callback=None):
    try:
        s = socket.socket()
        s.connect((ip, port))
        s.send(f"file:{os.path.basename(filename)}".encode())
        time.sleep(0.1)
        with open(filename, "rb") as f:
            while chunk := f.read(4096):
                s.send(chunk)
        s.close()
        if log_callback:
            log_callback("Transferencia completada.")
    except Exception as e:
        if log_callback:
            log_callback(f"[!] Error al transferir: {e}")
```

En esta parte del código lo que sucede es que se establece conexión con la IP destino, para enviar primero el nombre del archivo, para luego, enviar su contenido por partes (chunk). Y llama a log_callback() si se quiere mostrar mensajes en la GUI.

```python
def construir_grafo(ip_list, log_callback):
    graph = {ip: {} for ip in ip_list}
    for src in ip_list:
        for dst in ip_list:
            if src == dst:
                continue
            latency = ping_node(dst)
            if latency is not None:
                graph[src][dst] = latency
                log_callback(f"{src} → {dst}: {latency} ms")
            else:
                log_callback(f"{src} → {dst}: sin respuesta")
    return graph
```

En esta parte del código para cada par de nodos IP, hace ping (usa ping_node), si hay respuesta, guarda la latencia como peso de la arista en un diccionario tipo grafo. También muestra los resultados usando log_callback.

## Algoritmo Dijkstra.

```python
def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    previous_nodes = {node: None for node in graph}
    queue = [(0, start)]

    while queue:
        current_distance, current_node = heapq.heappop(queue)
        if current_distance > distances[current_node]:
            continue
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous_nodes[neighbor] = current_node
                heapq.heappush(queue, (distance, neighbor))
    return distances, previous_nodes
```

Esta parte es la primordial del código, lo que hace este algoritmo es que encuentra el camino más corto (mínima latencia) desde start a todos los demás. Usa heapq como cola de prioridad y retorna: “distances”: distancia mínima desde start a cada nodo. Y “previous_nodes”: para reconstruir el camino.

```python
def reconstruir_ruta(prev, destino):
    ruta = []
    while destino:
        ruta.insert(0, destino)
        destino = prev[destino]
    return ruta
```

Aquí se toma el diccionario de nodos previos (prev) generado por Dijkstra y reconstruye la ruta desde origen a destino.

```python
def mostrar_grafo(graph, ruta=None):
    G = nx.DiGraph()
    for src, vecinos in graph.items():
        for dst, latencia in vecinos.items():
            G.add_edge(src, dst, weight=latencia)

    pos = nx.spring_layout(G)

    nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=2000, font_size=9, font_weight='bold')
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels={k: f"{v:.0f} ms" for k, v in labels.items()})

    if ruta and len(ruta) > 1:
        path_edges = list(zip(ruta, ruta[1:]))
        nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='red', width=2)

    plt.title("Grafo de Red (Latencias)")
    plt.tight_layout()
    plt.show()
```

Usa networkx para crear un grafo dirigido, luego dibuja nodos, aristas y pesos (latencias). Si se pasa una ruta, la resalta en rojo.

```python
class RedGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Análisis de Red con Sockets")

        # IPs
        tk.Label(root, text="IPs de la red (separadas por coma):").grid(row=0, column=0, sticky="e")
        self.entry_ips = tk.Entry(root, width=60)
        self.entry_ips.grid(row=0, column=1, padx=5, pady=5)

        tk.Label(root, text="IP origen:").grid(row=1, column=0, sticky="e")
        self.entry_origen = tk.Entry(root, width=40)
        self.entry_origen.grid(row=1, column=1, padx=5, pady=5)

        tk.Label(root, text="IP destino:").grid(row=2, column=0, sticky="e")
        self.entry_destino = tk.Entry(root, width=40)
        self.entry_destino.grid(row=2, column=1, padx=5, pady=5)

        self.btn_analizar = tk.Button(root, text="Analizar red", command=self.analizar_red)
        self.btn_analizar.grid(row=3, column=0, columnspan=2, pady=10)

        # Transferencia
        tk.Label(root, text="Archivo local:").grid(row=4, column=0, sticky="e")
        self.entry_archivo = tk.Entry(root, width=60)
        self.entry_archivo.grid(row=4, column=1, padx=5, pady=5)

        tk.Label(root, text="IP destino para archivo:").grid(row=5, column=0, sticky="e")
        self.entry_ip_transfer = tk.Entry(root, width=40)
        self.entry_ip_transfer.grid(row=5, column=1, padx=5, pady=5)

        self.btn_transferir = tk.Button(root, text="Transferir archivo", command=self.transferir_archivo)
        self.btn_transferir.grid(row=6, column=0, columnspan=2, pady=10)
```

En esta parte del código la verdad es que no hay mucho que explicar, solo es la parte de la GUI.

```
143         # Salida
144         self.output = scrolledtext.ScrolledText(root, width=80, height=20)
145         self.output.grid(row=7, column=0, columnspan=2, padx=10, pady=10)
146 
147     def log(self, mensaje):
148         self.output.insert(tk.END, mensaje + "\n")
149         self.output.see(tk.END)
150 
151     def analizar_red(self):
152         self.output.delete(1.0, tk.END)
153         ip_list = [ip.strip() for ip in self.entry_ips.get().split(",")]
154         origen = self.entry_origen.get().strip()
155         destino = self.entry_destino.get().strip()
156 
157         if not origen or not destino or not ip_list:
158             messagebox.showwarning("Campos incompletos", "Debes ingresar IPs, origen y destino.")
159             return
160 
161         self.log("Escaneando red...")
162         grafo = construir_grafo(ip_list, self.log)
163 
164         if origen not in grafo or destino not in grafo:
165             messagebox.showerror("Error", "IP de origen o destino no está en la lista.")
166             return
167 
168         dist, prev = dijkstra(grafo, origen)
169         ruta = reconstruir_ruta(prev, destino)
```

```
171           if dist[destino] == float('inf'):
172               self.log(" No hay ruta disponible.")
173           else:
174               self.log(f"\nRuta óptima: {' -> '.join(ruta)}")
175               self.log(f"Latencia total: {dist[destino]} ms")
176               mostrar_grafo(grafo, ruta)
```

Como podemos ver, en toda esta parte del código, se hacen varias cosas en la GUI, tenemos un par de entradas con la lista de las IPs de la red, IP de origen y destino para analizar ambas redes. Y un archivo e IP destino para transferencias.
También se crean un par de botones que son muy importantes, ya que uno es el botón de “analizar red”, el cual, ejecuta el ping entre nodos, construye el grafo, calcula y muestra la ruta más eficiente. Luego tenemos otro botón el cual es “Transferir archivo”, este transfiere directamente el archivo, luego simula transferencia por la ruta óptima, y compara los tiempos.

```python
    def transferir_archivo(self):
        archivo = self.entry_archivo.get().strip()
        ip_destino = self.entry_ip_transfer.get().strip()
        ip_origen = self.entry_origen.get().strip()
        ip_list = [ip.strip() for ip in self.entry_ips.get().split(",")]

        if not archivo or not ip_destino or not ip_origen:
            messagebox.showwarning("Campos vacíos", "Archivo, origen y destino requeridos.")
            return

        if not os.path.isfile(archivo):
            messagebox.showerror("Archivo no encontrado", "El archivo especificado no existe.")
            return

        # Transferencia directa
        self.log(f"\nEnviando archivo directamente a {ip_destino}...")
        start_direct = time.time()
        send_file(ip_destino, archivo, log_callback=self.log)
        end_direct = time.time()
        tiempo_directo = round((end_direct - start_direct) * 1000, 2)

        self.log(f"Tiempo de transferencia directa: {tiempo_directo} ms")
```

```python
        # Simular ruta óptima
        self.log("\nCalculando ruta óptima...")
        grafo = construir_grafo(ip_list, self.log)

        if ip_origen not in grafo or ip_destino not in grafo:
            self.log("IP origen o destino no está en el grafo.")
            return

        distancias, prev = dijkstra(grafo, ip_origen)
        ruta = reconstruir_ruta(prev, ip_destino)

        if distancias[ip_destino] == float('inf'):
            self.log("No hay ruta óptima disponible.")
            return
```

```python
    # Simular tiempo de ruta óptima sumando latencias
        tiempo_optimo = 0
        for i in range(len(ruta) - 1):
            salto = grafo[ruta[i]].get(ruta[i + 1])
        if salto:
            tiempo_optimo += salto
        else:
            self.log(f"No se pudo simular salto: {ruta[i]} -> {ruta[i + 1]}")
            return

        self.log(f"Ruta óptima: {' -> '.join(ruta)}")
        self.log(f"Tiempo estimado por ruta óptima (suma de latencias): {round(tiempo_optimo, 2)} ms")

        # Comparación
        diferencia = round(tiempo_directo - tiempo_optimo, 2)
        if diferencia > 0:
            self.log(f"La ruta óptima es más rápida por {diferencia} ms.")
        elif diferencia < 0:
            self.log(f"La ruta directa fue más rápida por {abs(diferencia)} ms.")
        else:
            self.log("Ambos métodos tienen tiempos similares.")
```

En esta parte del código ocurren varias cosas, lo primero es que se compara el tiempo real de transferencia directa (medido con time.time()). Y después pasa la suma de latencias de la ruta óptima calculada (simulación, no transferencia real hop por hop). Esto permite analizar cuál opción sería más eficiente.

```python
if __name__ == "__main__":
    root = tk.Tk()
    app = RedGUI(root)
    root.mainloop()

```

Esta parte final del código simplemente lanza la interfaz gráfica cuando se ejecuta el script directamente.

## Código servidor.

Este código igualmente inicia con las librerías que necesitaremos, las cuales son las siguientes.

```python
import socket
import threading
import os
```

Ahora si damos inicio con el código.

```python
def handle_client(conn, addr):
    try:
        msg = conn.recv(1024).decode()
        if msg == "ping":
            conn.send("pong".encode())
        elif msg.startswith("file:"):
            filename = msg.split(":")[1]
            with open(f"recibido_{filename}", "wb") as f:
                while True:
                    chunk = conn.recv(4096)
                    if not chunk:
                        break
                    f.write(chunk)
            print(f"[+] Archivo recibido de {addr}: {filename}")
    except Exception as e:
        print(f"[!] Error con {addr}: {e}")
    finally:
        conn.close()
```

En esta parte del código, lo que sucede es que se espera el primer mensaje del cliente. Puede ser "ping" o "file:nombre". Responde "pong" (esto se usa para pruebas de latencia). Luego de esto se extrae el nombre del archivo, para crear un archivo local llamado recibido_<nombre>.
Luego entra en un bucle para recibirlo por partes (chunks de 4096 bytes), cuando no llegan más datos, termina y cierra el archivo. Si ocurre cualquier error, se imprime en la consola. Y siempre se cierra la conexión al final.

```python
def start_server(port=9000):
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('', port))
    server.listen()
    print(f"[ 🖥 ] Servidor escuchando en puerto {port}")
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()

```

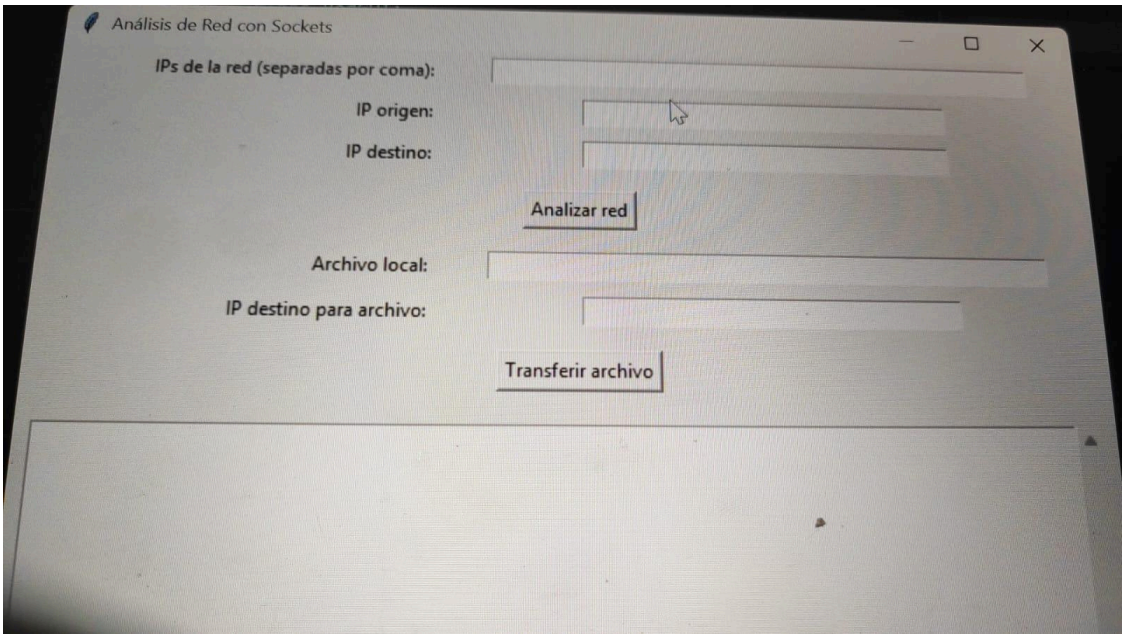
Lo primero que hace esa parte del código es crear un socket tipo TCP (SOCK_STREAM). Lo vincula a todas las interfaces (") en el puerto 9000, y lo pone a escuchar conexiones entrantes. Para poder recibir los archivos.

Cada vez que un cliente se conecta, se crea un nuevo hilo (thread) que atiende a ese cliente sin bloquear el resto del servidor.

```
35  if __name__ == "__main__":
36      start_server()
37
```

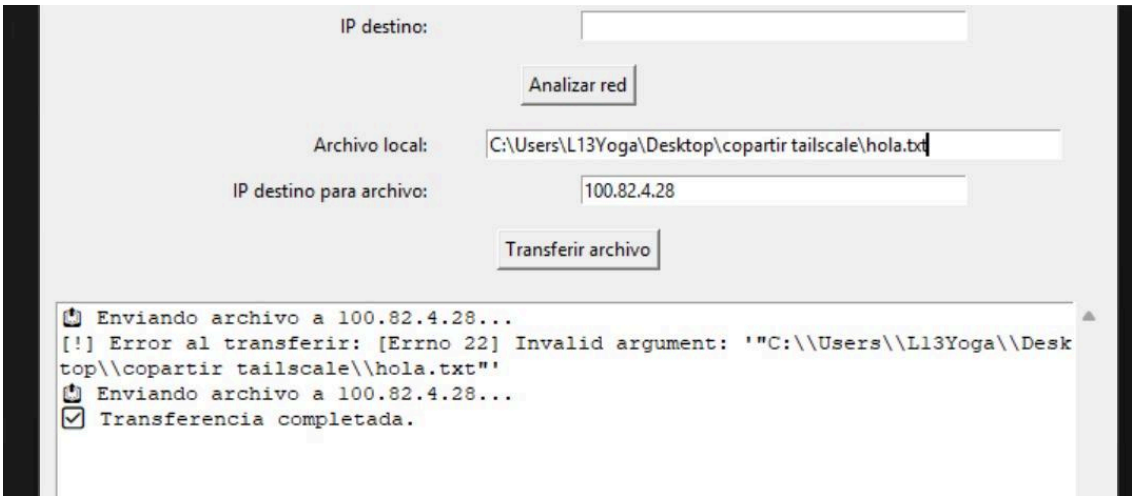
Esta parte del código simplemente ejecuta el script.

**Evidencia de la GUI.**




En esta imagen podemos ver evidencia de cómo es que se ve la GUI una vez que el programa es ejecutado. Tiene apartados como “Analizar red” y el apartado de “Transferir archivos”, en la parte inferior vemos un recuadro en el cual podemos visualizar las acciones que se hagan, se hayan podido hacer o si no nos saldrá el error.

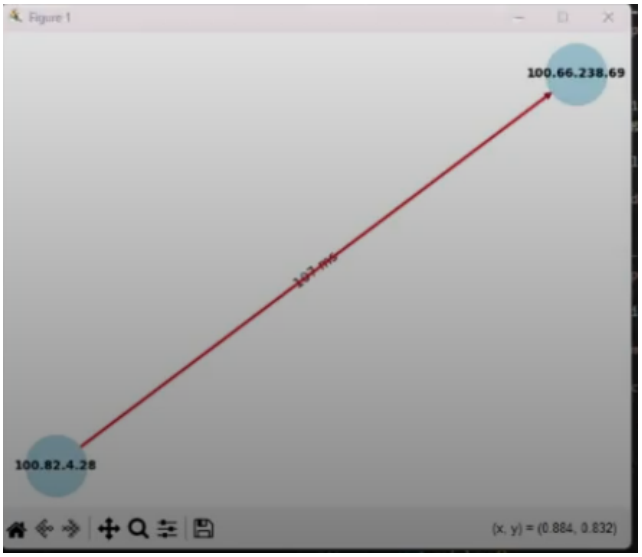
**Transferencia exitosa.**




Aquí vemos cómo es que la transferencia de paquetes fue exitosa, ya que se envió un archivo .txt desde la pc de cliente a la pc del servidor, se envió exitosamente.


Aquí vemos la perspectiva de la pc de servidor, podemos ver cómo es que si le llegó el archivo y lo recibió correctamente.

**Grafo de Dijkstra**




En esta imagen podemos ver la ruta que tomó el archivo a la hora de ser enviado a la pc destino (servidor), vemos que es un camino sencillo ya que el algoritmo también nos ayuda a calcular la mejor ruta para mandar este archivo.

**Explicación del protocolo utilizado:**

El protocolo utilizado fue **SCP (Secure Copy Protocol)**, es un protocolo de red basado en SSH (Secure Shell), que permite copiar archivos de forma segura entre dos computadoras remotas. La seguridad está garantizada porque:

-Toda la comunicación está cifrada a través de SSH.
-Se verifica la identidad de los nodos a través de claves públicas/privadas o contraseñas.

Esto es importante:

Tailscale crea una red privada cifrada (VPN mesh) entre dispositivos. SCP usa SSH sobre esta red privada, lo cual significa que el archivo:

- Viaja cifrado dos veces (por SSH y por Tailscale).
- Nunca sale a internet pública si ambos dispositivos están en Tailscale.

## Código Kruskal:

Damos inicio con la explicación del código, iniciamos poniendo las librerías que necesitaremos.

```
Kruskal.py > ...
1   import networkx as nx
2   import matplotlib.pyplot as plt
```

networkx: para manipular grafos (crear, analizar y visualizar).
matplotlib.pyplot: para generar gráficas.

```
8   edges = [
9       ('A', 'B', 10),
10      ('A', 'C', 15),
11      ('B', 'C', 5),
12      ('B', 'D', 20),
13      ('C', 'D', 30),
14      ('C', 'E', 25),
15      ('D', 'E', 10)
16  ]
```

Aquí se define una lista de aristas del grafo. Cada arista es una tupla (nodo1, nodo2, peso), donde el peso representa el ancho de banda utilizado entre esos nodos.

```
21  class DisjointSet:
22      def __init__(self, vertices):  # CORREGIDO: doble guion bajo
23          self.parent = {v: v for v in vertices}
```

En esta clase ayuda a mantener la estructura de conjuntos disjuntos (Union-Find) necesaria para detectar ciclos en el algoritmo de Kruskal.

```
25      def find(self, v):
26          if self.parent[v] != v:
27              self.parent[v] = self.find(self.parent[v])
28          return self.parent[v]
29
```

Busca el representante del conjunto de v.

```
30      def union(self, u, v):
31          root_u = self.find(u)
32          root_v = self.find(v)
33          if root_u != root_v:
34              self.parent[root_u] = root_v
35              return True
36          return False
```

Une dos conjuntos si no están conectados, y retorna True si se realizó la unión.

**Kruskal.**

```
38  def kruskal(vertices, edges):
39      ds = DisjointSet(vertices)
40      mst = []
41      total_bandwidth = 0
42
43      edges.sort(key=lambda x: x[2])
```

En este apartado se inicializa la estructura de conjuntos disjuntos. Se ordenan las aristas por peso (ancho de banda).

```
45      for u, v, weight in edges:
46          if ds.union(u, v):
47              mst.append((u, v, weight))
48              total_bandwidth += weight
49
50      return mst, total_bandwidth
```

Para cada arista: si no forma un ciclo (según union), se agrega al MST (árbol de expansión mínima).

```
55  nodes = set()
56  for u, v, _ in edges:
57      nodes.add(u)
58      nodes.add(v)
59
60  mst, mst_bandwidth = kruskal(nodes, edges)
```

Se extraen todos los nodos únicos a partir de las aristas, y se aplica el algoritmo de Kruskal.

```
65  original_bandwidth = sum(weight for _, _, weight in edges)
```

Se calcula el ancho de banda total del grafo original.

```
print("=== Topología Original ===")
for edge in edges:
    print(edge)
print(f"Total de ancho de banda utilizado: {original_bandwidth} unidades")

print("\n=== Árbol de Expansión Mínima (Kruskal) ===")
for edge in mst:
    print(edge)
print(f"Total de ancho de banda utilizado: {mst_bandwidth} unidades")
print(f"\nAhorro: {original_bandwidth - mst_bandwidth} unidades\n")
```

Se imprime la topología original y la optimizada.

```
def draw_graph(edges, title):
    G = nx.Graph()
    for u, v, weight in edges:
        G.add_edge(u, v, weight=weight)
```

Crea un grafo y agrega las aristas con pesos.

```
    pos = nx.spring_layout(G, seed=42)  # Distribución de nodos
    weights = nx.get_edge_attributes(G, 'weight')
```

Calcula la posición de los nodos y obtiene los pesos para mostrar en las aristas.

```
    plt.figure(figsize=(8, 6))
    nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=800, font_weight='bold', edge_color='gray')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=weights)
    plt.title(title)
    plt.show()
```

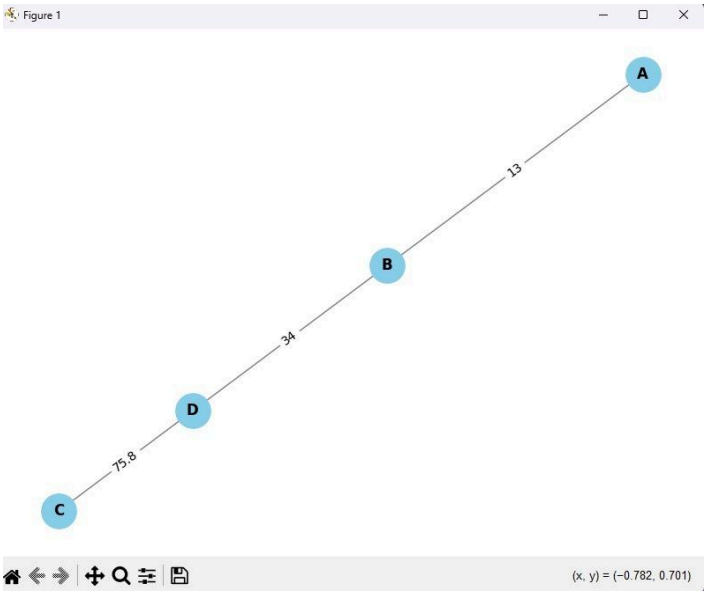
Dibuja el grafo con etiquetas, pesos, y muestra el título correspondiente.

```
draw_graph(edges, "Topología Original de la Red (con todos los enlaces)")
draw_graph(mst, "Topología Optimizada (Árbol de Expansión Mínima - Kruskal)")
```

Se muestran ambos grafos.

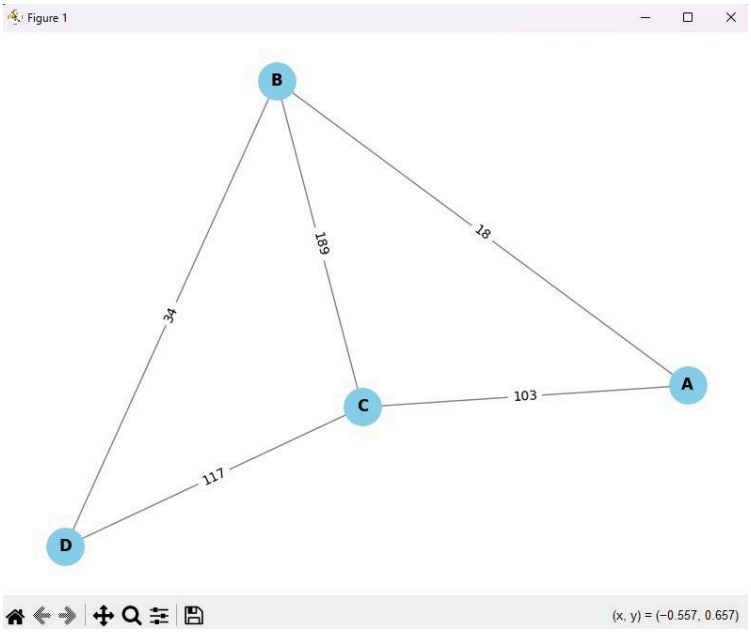
**Grafo generado con camino óptimo.**

A Leonardo compu
B compu Eloy
C celular Eloy
D celular Leonardo




Aquí vemos el grafo “ideal” para mandar un archivo desde un nodo a otro, el programa selecciona este camino mediante pesos que le fueron asignados, y el mejor camino es el de mayor peso.

**Grafo inicial.**



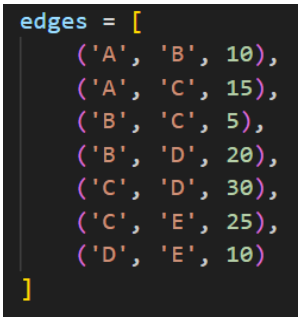

Aquí podemos ver el grafo antes de la asignación de pesos, ya que está en su forma base por así decirlo.

**Explicación del MST.**

MST significa **Minimum Spanning Tree** o **Árbol de Expansión Mínima**.
Es un subconjunto de las conexiones (aristas) de un grafo que cumple con:

- Conecta todos los nodos del grafo.
- No forma ciclos.
- Tiene el menor peso total posible (en tu caso, el peso es el ancho de banda utilizado).

En el código lo podemos ver de la siguiente manera:



```
edges = [
    ('A', 'B', 10),
    ('A', 'C', 15),
    ('B', 'C', 5),
    ('B', 'D', 20),
    ('C', 'D', 30),
    ('C', 'E', 25),
    ('D', 'E', 10)
]
```

Esto representa tu topología original: todos los enlaces entre nodos y sus anchos de banda.

El algoritmo Kruskal trabaja así:

- Ordena todas las conexiones de menor a mayor ancho de banda.
- Recorre cada conexión y la añade al MST si no forma un ciclo.
- Se detiene cuando ha conectado todos los nodos exactamente una vez (es decir, tiene n-1 enlaces si hay “n” nodos).

Esto fue muy útil en la tarea, ya que utilizamos una VPN con pocos posibles caminos entre nodos, pero aún así podemos ver que: El MST nos ayuda a identificar la **estructura más eficiente** para conectar todo.

Puedes usar esto para decidir:

- Qué túneles mantener.
- Cuáles eliminar o deshabilitar por baja eficiencia.
- Cómo balancear el tráfico. (de ser necesario)

---

## Conclusión / Retrospectiva

Durante el desarrollo de este proyecto, se implementó un sistema funcional que simula la comunicación entre nodos en una red local, utilizando sockets para medir la latencia entre equipos y realizar transferencia de archivos. A partir de esos datos, se construyó un grafo donde los nodos representan computadoras y las aristas indican los tiempos de respuesta entre ellas. Luego, se aplicaron algoritmos de grafos como Dijkstra y Kruskal para determinar las rutas óptimas.

Sin embargo, en el proceso surgieron varios inconvenientes. Uno de los principales fue la falla en la transferencia de archivos, ya que no se enviaban correctamente debido a errores en la lógica de conexión o en la sincronización entre el cliente y el servidor. Además, hubo falta de comunicación en ciertos aspectos del trabajo en equipo, lo que provocó que partes importantes del código no se integraran a tiempo o que ciertos errores se repitieran innecesariamente. Para resolver estos problemas, se revisó minuciosamente el flujo del programa, se consultó documentación y se estableció una mejor coordinación entre los integrantes, asignando tareas más claras y compartiendo el avance con mayor frecuencia, estos últimos aspectos a mejorar fueron hablados mediante una sesión presencial .

En cuanto a los algoritmos utilizados, Dijkstra resultó muy útil para hallar la ruta más corta entre dos nodos, optimizando el tiempo de envío de archivos. Su principal ventaja es que encuentra siempre la mejor ruta posible en términos de latencia, pero su desventaja es que no se adapta bien a cambios dinámicos en la red, como desconexiones inesperadas. Por otro lado, se utilizó el algoritmo de Kruskal para construir un árbol de expansión mínima, lo cual permitió conectar todos los nodos de la red con el menor costo total posible. Entre sus ventajas se destaca su eficiencia para encontrar la estructura de conexión óptima cuando se trata de abarcar toda la red. No obstante, su desventaja es que no garantiza la mejor ruta entre un par específico de nodos, ya que se enfoca en la conexión global, no en caminos individuales.

Como retrospectiva final, este proyecto dejó lecciones valiosas. Se comprendió que incluso un sistema bien planteado puede fallar si no se revisa detalladamente cada parte del código y si no existe una buena comunicación en el equipo. Se aprendió también que los algoritmos clásicos como Dijkstra y Kruskal tienen un papel importante en la solución de problemas reales, pero que deben ser elegidos y combinados según el objetivo específico de la aplicación. Por último, quedó clara la importancia de hacer pruebas frecuentes, documentar adecuadamente, y sobre todo, mantener una colaboración activa entre todos los miembros del equipo para asegurar el éxito del proyecto.

---

**Tareas**

Tipos ~~metodologias~~metodologías ~~agiles~~ágiles

- [x] ~~-y sus combinacinescombinaciones~~
- [x] ~~-y cual es la mejor para este proyecto~~
- [x] ~~Asignar roles~~
- [x] ~~creacioncreación de cronograma~~
- [x] ~~github https://github.com/michhsolis/ActividadVPN~~
- [x] ~~¿Qué es una vpn?~~
- [x] ~~INTRODUCCIÓN~~
- [x] ~~OBJETIVO~~
- [x] ~~DESARROLLO~~
- [x] ~~CONCLUSIÓN~~
- [x] ~~REFERENCIAS~~
- [x] ~~qué latencia existe de una máquina a otra~~
- [x] ~~Crear una VPN entre los dispositivos de los integrantes (usando herramientas como WireGuard, OpenVPN, o Tailscale).~~
- [x] ~~Asignar IPs estáticas a cada nodo.~~
- [x] ~~Medir latencias entre nodos (con ping o un script en Python).~~
- [x] ~~Medir ancho de banda (con iperf3 o speedtest-cli).~~
- [x] ~~Crear un grafo ponderado con estos datos (nodos = dispositivos, aristas = latencia/ancho de banda).~~
- [x] ~~Usar el grafo de latencias para implementar Dijkstra.~~
- [x] ~~Determinar la ruta más rápida para transferir archivos entre dos nodos.~~
- [x] ~~HACER UNA GUI donde se elija el o los archivos a transferir y al equipo que se va a enviar y Transferir un archivo de prueba (ej: 10 MB10MB, 100MB, 1GB, etc) usando la ruta óptima.~~
- [x] ~~Usar el grafo de ancho de banda para implementar Kruskal.~~
- [x] ~~Generar un árbol de expansión mínima (MST) que optimice el uso de la red.~~
- [x] ~~Comparar la topología original con la propuesta por Kruskal.~~
- [x] ~~Crear un script que actualice automáticamente el grafo cada 5 minutos y re-calculerecalcule las rutas.~~

---

**Bibliografía:**


1. (1 julio 2024). ¿Qué es el desarrollo de software adaptativo (ASD)?. geeksforgeeks.https://www.geeksforgeeks.org/adaptive-software-development-asd/
2. Vicente Sancho.Dynamic Systems Development Method (DSDM). Vicente Sancho.https://vicentesg.com/dynamic-systems-development-method-dsdm/
3. https://vicentesg.com/dynamic-systems-development-method-dsdm/
4. Julia Martins. (15 febrero 2025).Scrum: conceptos clave y cómo se aplica en la gestión de proyectos. asana. https://asana.com/es/resources/what-is-scrum
5. Scrumban: domina dos metodologías ágiles. atlassian. https://www.atlassian.com/es/agile/project-management/scrumban#:~:text=El%20scrumban%20combina%20dos%20metodolog%C3%ADas,en%20cuenta%20su%20enfoque%20%C3%BAnico?&text=En%20esta%20gu%C3%ADa%2C%20explicaremos%20qu%C3%A9,de%20gesti%C3%B3n%20%C3%A1gil%20de%20proyectos
6. Daiana Nieves Narducci.(18 diciembre 2024). Agile Hybrid: Enfoque flexible para proyectos complejos. OpenWebinars.https://openwebinars.net/blog/agile-hybrid-enfoque-flexible-para-proyectos-complejos/#:~:text=en%20t%C3%BA%20empresa.-,Qu%C3%A9%20es%20Agile%20Hybrid,conviene%20recurrir%20a%20esta%20metodolog%C3%ADa
7. (12/04/2021). Kanban y Scrum, dos metodologías ágiles diferentes. UNIR.https://www.unir.net/revista/ingenieria/kanban-scrum-metodologias-agiles/#:~:text=Kanban%20y%20Scrum%20pertenecen%20a,trabajo%20para%20alcanzar%20el%20segundo.
8. (19/23/2024). Metodologías ágiles: qué son, tipos y ejemplos.INESDI.https://www.inesdi.com/blog/que-son-las-metodologias-agiles-tipos-y-ejemplos/#:~:text=La%20metodolog%C3%ADa%20Lean%20se%20centra,de%20trabajo%20y%20apoyo%20constante.
9. Javier Garzas. (25 septiembre 2012). Las metodologías Crystal. Otras metodologías ágiles que, quizás, te puedan encajar más que Scrum. Javier Garzas. https://www.javiergarzas.com/2012/09/metodologias-crystal.html
10.


---

**Recursos:**

Página para crear VPN:
https://pyseek.com/2024/07/create-a-simple-vpn-server-using-python/

Github:
https://github.com/michhsolis/ActividadVPN