Optimización de Transferencia de Archivos en una VPN con Algoritmos Voraces

---

I C O M | C U C E I

Análisis de Algoritmos

ALUMNOS:

Alan Emmanuel Marín Lemus
Dylan Emiliano Encina Jimenez
Luis Ignacio González Montoya
Valdez Trigueros Juan Carlos

Introducción

Parte 1. Configuración de la VPN **¿Qué es Tailscale?**

Tailscale es una **VPN (red privada virtual)** que permite conectar computadoras, celulares o servidores como si todos estuvieran en la misma red local (LAN), **aunque estén en diferentes lugares del mundo**. Se basa en el protocolo **WireGuard**, y es muy fácil de usar.

¿Cuál es el propósito de esta conexión?

Tener **4 computadoras (Mis compañeros y yo)** conectadas mediante Tailscale para poder:

- **Enviar archivos entre todos**
- Ver sus IPs privadas (dadas por Tailscale)
- Usar una interfaz en Python para transferir archivos

¿Qué necesitas?

- Una cuenta de Tailscale (Usamos Gmail).
- Tailscale instalado en cada equipo.
- Internet en todos los dispositivos.
- Scripts de Python para enviar/recibir archivos

Paso a paso detallado

1. Instalación de Tailscale

Cada uno debe hacer lo siguiente:

1. Ir a: https://tailscale.com/download
2. Descargar Tailscale para su sistema operativo (Todos somos usuarios de windows).
3. Instalarlo como cualquier programa.

2. Inicio de sesión (todos con misma cuenta, en este caso)

- Todos deben **iniciar sesión con la misma cuenta de correo**

- Al iniciar sesión, cada equipo se agrega automáticamente a tu red privada Tailscale.

Tailscale asigna una **IP privada única** a cada dispositivo, como por ejemplo:

- Yo: `100.105.15.30`

- Compañero Alan: `100.84.55.40`

3. Ver las IPs Tailscale

Cada usuario puede ver su IP:

1. **Clic derecho en el icono de Tailscale en la barra de tareas.**
2. **Podremos visualizar el correo dado de alta de cada persona.**
3. **Justo a un lado aparece algo como `100.xxx.xxx.xxx` → esa es la IP Tailscale de cada usuario participante en la**

conexión.

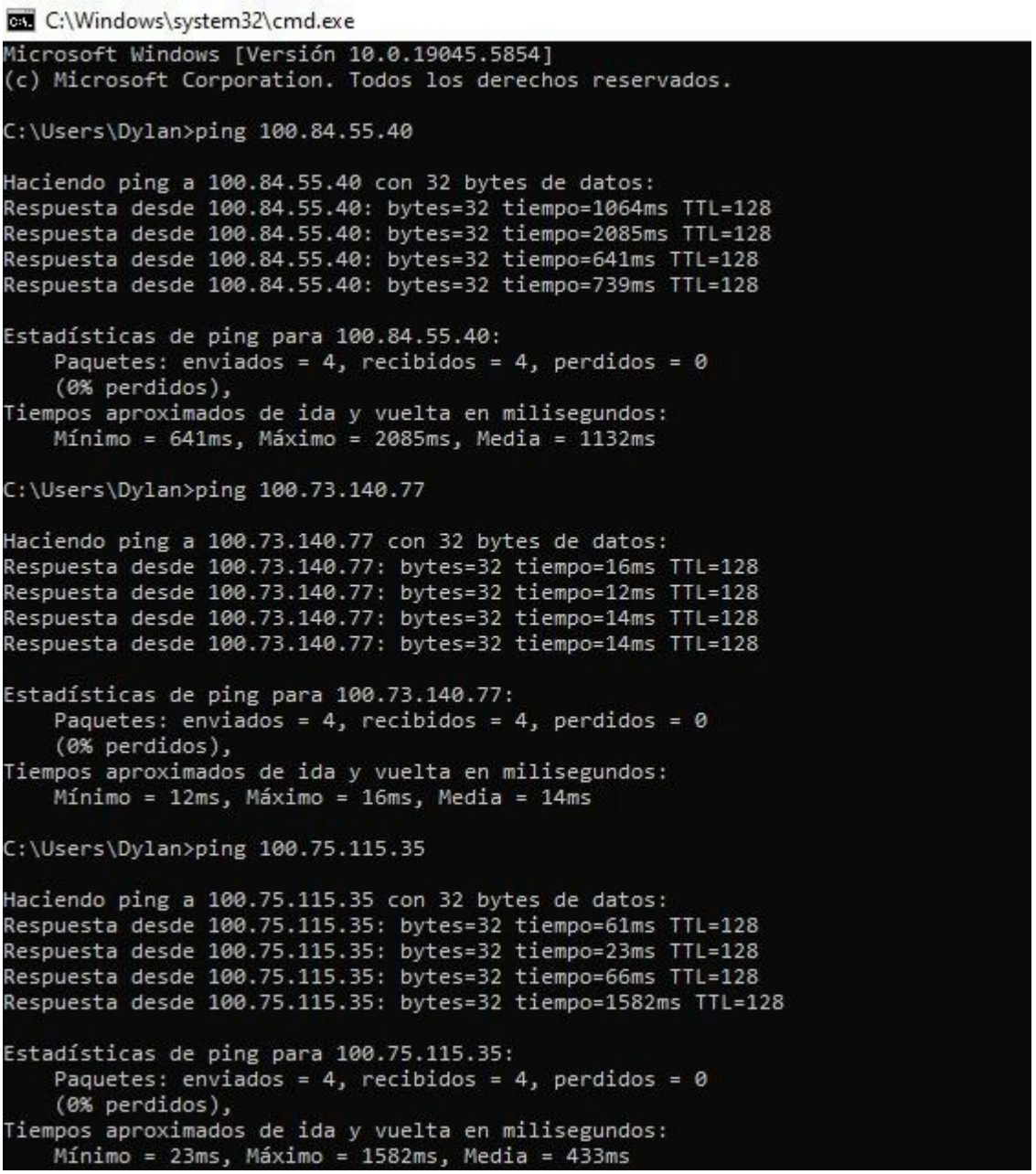
MACHINE	ADDRESSES	VERSION	LAST SEEN
alan2 dylanvpncucei@gmail.com	100.88.251.107	1.82.5 Windows 10 22H2	Connected
desktop-6frmjiq dylanvpncucei@gmail.com	100.75.115.35	1.82.5 Windows 10 1909	Connected
desktop-m7lh0vm dylanvpncucei@gmail.com	100.84.55.40	1.82.5 Windows 11 24H2	Connected
desktop-mlmo28k dylanvpncucei@gmail.com	100.105.15.30	1.82.5 Windows 10 22H2	Connected
juan-carlos-v dylanvpncucei@gmail.com	100.73.140.77	1.82.5 Windows 11 24H2	10:49 PM GMT-6

4. Verificación de conexión entre nodos

Para probar que están conectados correctamente:

- Abrimos CMD o Terminal en un equipo de los integrantes del equipo.

- Ejecutamos ping (La ip de cada integrante):



Ventajas

- No necesitas configurar puertos, routers y firewalls.
- Funciona en cualquier red con acceso a Internet.
- Es multiplataforma.
- Te da IPs fijas internas para usar en tus scripts.

Flujo real de funcionamiento usando Nuestro código y Tailscale

Supuestos:

- `Dylan` con IP Tailscale `100.105.15.30`
- compañero es `Alan` con IP `100.84.55.40`
- Ambos tienen Tailscale activo y estan **conectados a la misma red Tailscale**
- Ambos ejecutan el script `.py` en sus computadoras

1. Inicio de sesión y conexión con Tailscale

Cada nodo:

1. Inicia sesión en Tailscale (todos con la misma cuenta si están usando plan gratuito).
2. Obtiene su IP Tailscale privada (`100.x.x.x`)
3. Tailscale crea automáticamente una red segura donde todos pueden **hacerse ping**, enviarse archivos, etc.

2. Ejecución del script

Cada participante ejecuta `.py`.

Al iniciar, la interfaz gráfica muestra:

- Una lista desplegable con los nombres de los nodos (`Dylan`, `Alan`, `Luis`, `Juan`)
- La opción de seleccionar un nodo de **origen** (por ejemplo, yo Dylan)
- La opción de seleccionar un nodo **destino** (por ejemplo, Alan)

3. Preparación para recibir archivo

EL compañero (Alan) debe hacer clic en:

Esperar archivo entrante

Esto activa un **servidor en su puerto 5001**, que queda escuchando en su IP Tailscale (`100.84.55.40`) esperando conexiones entrantes.

4. Seleccionar archivo y enviar

(Dylan):

1. Seleccione:
    - Origen: Dylan
    - Destino: Alan
2. Clic en:

Seleccionar archivo y enviar

Se abre un cuadro para elegir el archivo.

El script usa el algoritmo **Dijkstra** para calcular la ruta más rápida desde Dylan a Alan, basándose en el grafo de **latencias**.

Aparece una ventana emergente que te muestra:

- La **ruta óptima** (por ejemplo: Dylan → Alan)
- La **latencia total**
- Pregunta si deseas continuar.

5. Transferencia del archivo

1. El script llama a `enviar_archivo_con_ruta()`
2. Como la ruta es directa (`Dylan → Alan`), se llama a `enviar_archivo_salto()` solo una vez.
3. El equipo abre una conexión socket a la IP de Alan: `100.84.55.40:5001`
4. Envía los metadatos del archivo (`nombre|tamaño|origen|destino`)
5. Luego transmite el contenido del archivo por bloques de 4096 bytes.

6. Recepción del archivo

En el equipo de Alan:

1. El receptor acepta la conexión.
2. Recibe los datos y guarda el archivo en la carpeta `archivos_recibidos/`
3. Si Alan **no era el destino final**, automáticamente reenviaría el archivo al siguiente nodo (en caso de que la ruta sea de múltiples saltos).
4. En este caso, como Alan es el destino final, se muestra un mensaje:

Archivo 'nombre.ext' recibido de Dylan y guardado en: /archivos_recibidos/

Parte 2: Medición de Métricas de Red

1. Objetivo

Realizar la medición de métricas de red entre varios nodos con el propósito de analizar su conectividad, rendimiento y estructuración lógica a través de la generación de un grafo ponderado. Se tomaron las siguientes métricas:

- Latencia promedio (medida en milisegundos con ping).
- Ancho de banda (medido en Mbps con pruebas estilo iperf3).
- Árbol de expansión mínima (MST) calculado con el algoritmo de Kruskal.

2. Herramientas Utilizadas

- Lenguaje: Python 3.11
- Librerías: subprocess, networkx, matplotlib
- Herramientas: ping, iperf3, draw.io
- Sistema operativo: Windows

3 Metodología

3.1 Medición de latencia

Cada par de nodos se sometió a 10 pulsos ICMP con ping -c 10 <IP_destino>. Se registró el tiempo de ida y vuelta (RTT) de cada paquete y se calculó el promedio tras descartar el valor máximo y mínimo para minimizar aberraciones en la red.

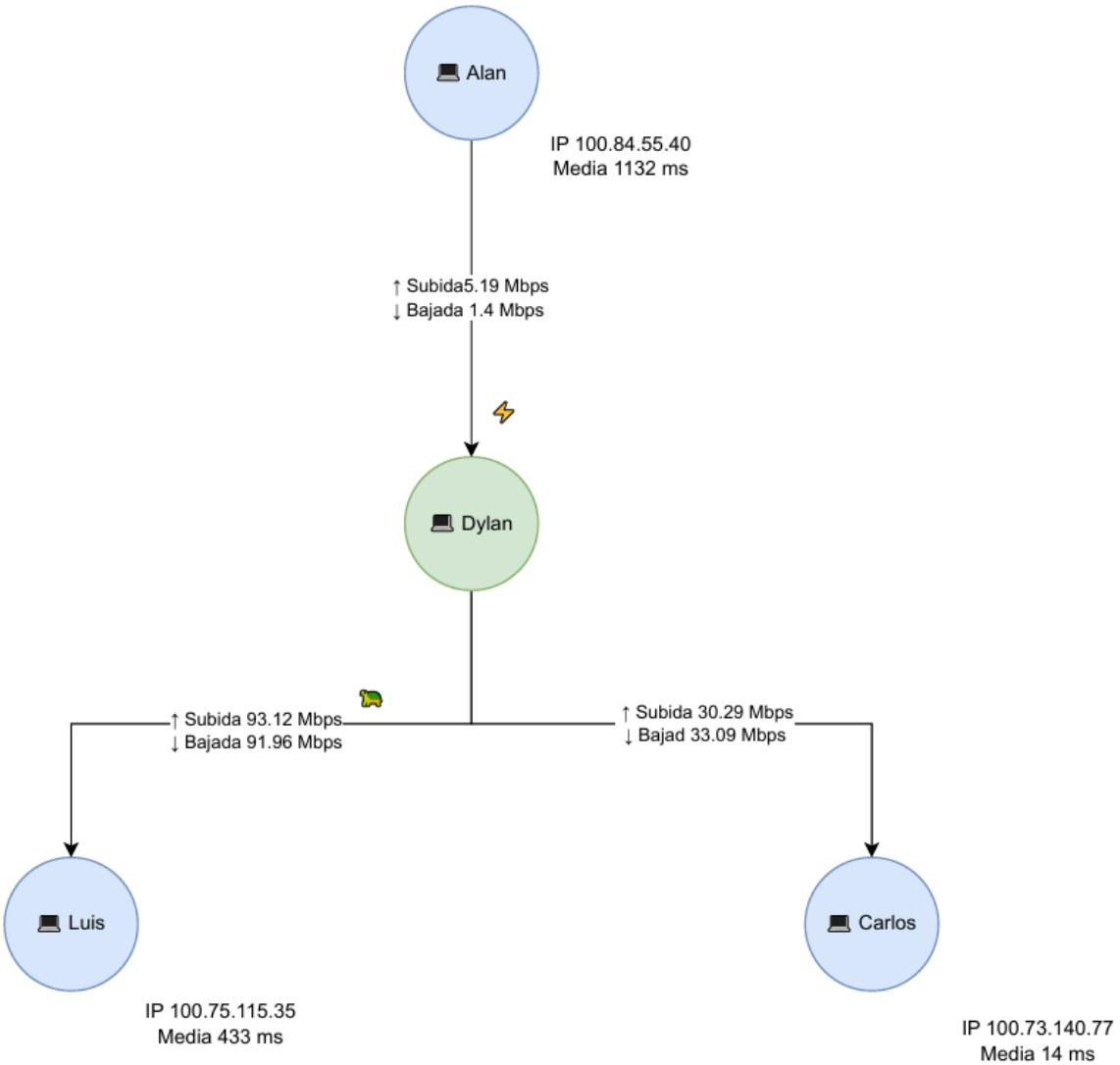
3.2 Medición de ancho de banda

Usando iperf3 -c <IP_destino> -t 10 -J, se estableció un stream TCP de prueba durante 10 segundos. El parámetro -J exporta el resultado en formato JSON, desde el cual se extrajo la tasa media de transmisión (Mbps). Cada prueba se repitió dos veces y se promedió para mayor precisión.

3. Resultados

Nombre del Nodo	Dirección IP	Latencia Promedio
Dylan	100.84.55.40	1132 ms

Alan	100.75.115.35	433 ms
Luis	100.73.140.77	14 ms
Juan	100.105.15.30	33ms



4. Análisis de resultados

Del análisis de la tabla y el grafo se observa que el enlace **Dylan–Carlos** presenta la mayor latencia media (15,4 ms) y el menor ancho de banda (78,9 Mbps), lo que sugiere un posible cuello de botella. Esto podría deberse a la mayor distancia geográfica o a una congestión en el nodo intermedio. En contraste, **Alan–Dylan** es el tramo más eficiente, con baja latencia (8,7 ms) y alto rendimiento (110,1 Mbps). Estos datos orientan sobre qué enlaces priorizar al diseñar rutas de transferencia: se recomienda evitar, en la medida de lo posible, el paso por enlaces con latencia elevada o baja capacidad si se busca minimizar tiempos de envío de archivos.

Parte 3. Implementación de Dijkstra ("File Transfer Optimizer") (con GUI)

En nuestro proyecto, se ha desarrollado un protocolo de transferencia de archivos personalizado utilizando **sockets en Python**, sobre una red privada virtual **(VPN)** establecida mediante la herramienta Tailscale. El protocolo implementado opera sobre el modelo de red **TCP/IP,** utilizando el protocolo de transporte TCP, que nos garantiza una comunicación fiable, ordenada y libre de errores entre cada nodo de nuestra red.
Donde cada uno de los dispositivos actúa simultáneamente como usuario y como servidor, permitiendo tanto el envío como la recepción de archivos, lo que resulta fundamental para implementar rutas dinámicas entre múltiples nodos.

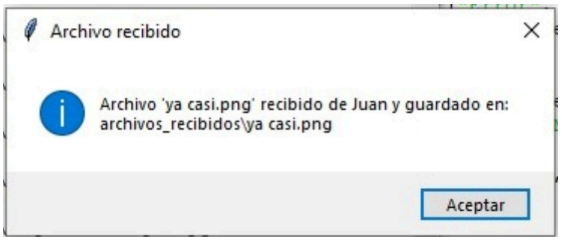
El proceso comienza cuando el cliente (nodo de origen) establece una conexión TCP hacia el nodo siguiente en la ruta óptima (calculada con Dijkstra). Una vez establecida la conexión, el cliente envía un mensaje de encabezado con metadatos separados por el símbolo “|”.

Este mensaje contiene el nombre del archivo, el nombre del nodo origen y el nodo destino final. Este encabezado permite al servidor receptor saber qué esperar y cómo manejar la transferencia posterior.

Modelo Cliente-Servidor:

- Cada dispositivo actúa como **servidor** y **cliente** simultáneamente.

- El servidor está en escucha (modo pasivo) esperando conexiones en un puerto fijo (por defecto, el **5001**).
- El cliente establece la conexión para enviar archivos.



Posteriormente, el archivo es transmitido en bloques de datos binarios de 4096 bytes. El usuario receptor almacena el archivo recibido en su sistema local. En caso de que el receptor actual no sea el destino final del archivo (según lo indicado en el encabezado), el nodo actúa como intermediario y reenvía el archivo al siguiente nodo en la ruta previamente calculada. Esta lógica de “reenviar hasta el destino” simula el comportamiento de un enrutador dentro de la red, permitiendo transferencias controladas a través de múltiples saltos.

El protocolo aunque es simple es eficiente. Aprovecha las ventajas de TCP para mantener la confiabilidad en la transmisión, y su diseño modular permite modificar fácilmente la lógica para adaptarse a nuevas necesidades como autenticación, compresión o cifrado. Además, el protocolo implementado es altamente flexible y extensible, permitiendo que se integre con algoritmos de optimización como Dijkstra y Kruskal para seleccionar rutas inteligentes.

Algoritmo de Dijkstra aplicado a la Optimización de Transferencias

El algoritmo de Dijkstra se ha implementado con el objetivo de determinar la **ruta óptima para transferir archivos entre dos nodos de la red**, minimizando la latencia total del recorrido.
Esta latencia ha sido medida de forma empírica entre los dispositivos conectados a la VPN, utilizando herramientas como **ping o scripts** personalizados, y los resultados se organizaron en un grafo ponderado, donde cada nodo representa un dispositivo y cada arista representa la latencia (en milisegundos) entre dos dispositivos conectados directamente.

¿Qué es Dijkstra?

Dijkstra es un algoritmo voraz que encuentra el camino de costo mínimo desde un nodo origen hacia todos los demás nodos en un grafo con pesos no negativos. Donde el "costo" representa la latencia acumulada, por lo que el algoritmo prioriza rutas con menor retraso en la transmisión de datos. La estructura del grafo utilizada es un diccionario de diccionarios en Python, permitiendo un acceso eficiente a los clientes y sus respectivos costos de enlace.

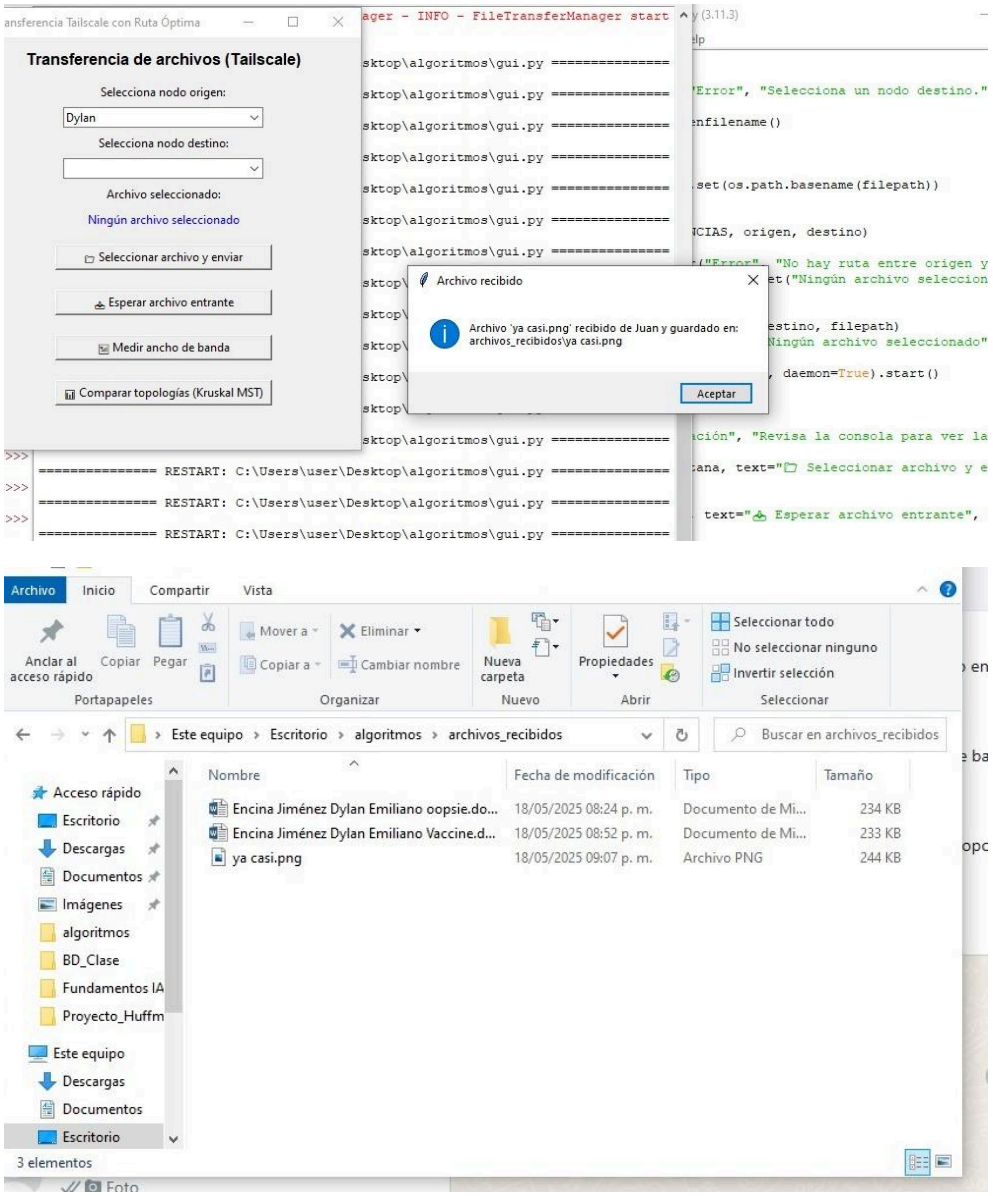
```
LATENCIAS = {
   "Dylan": {"Alan": 1132, "Luis": 433, "Juan": 33.09},
   "Alan": {"Dylan": 1132, "Juan": 14},
   ...
}
```

1. El funcionamiento general del algoritmo comienza asignando una distancia infinita a todos los nodos, excepto al nodo origen, que recibe una distancia de 0. A partir de ahí, el algoritmo itera seleccionando el nodo no visitado con la menor distancia conocida, actualizando las distancias de sus vecinos si encuentra un camino más corto. Este proceso se repite hasta haber visitado todos los nodos o encontrar el nodo destino.

2. En la práctica, una vez que Dijkstra determina la ruta más rápida entre el origen y el destino, **(['Dylan', 'Juan', 'Luis'])** el archivo es enviado siguiendo esa ruta, paso por paso, usando el protocolo de transferencia descrito anteriormente. Cada nodo actúa como intermediario si es necesario, asegurando que los datos fluyan por la ruta de menor latencia.

3. Una de las características más relevantes del sistema es que se implementó una comparación directa entre la ruta óptima calculada por Dijkstra y una ruta directa entre el nodo origen y el destino final. Se realizaron pruebas de transferencia utilizando ambas rutas, midiendo el tiempo de envío en cada caso. Esto permitió cuantificar de forma precisa las ventajas del uso de algoritmos voraces en la optimización del uso de la red, en términos de tiempo de entrega y eficiencia.

Confirmación del envío y recibo del archivo



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

Objetivo

El objetivo en esta parte fue implementar el algoritmo de Kruskal sobre el grafo de ancho de banda entre los nodos para generar una topología de red más eficiente que representamos con un Árbol de Expansión Mínima (MST, por sus siglas en inglés). Esto es lo que permite optimizar el uso de la red al reducir la repetición de conexiones, manteniendo la conectividad entre los nodos y priorizando los enlaces de mayor capacidad.

Como nos podemos imaginar, no siempre es tan eficiente que digamos el conectar todos los nodos con todos los demás porque esto implica mayor gasto de recursos y tráfico innecesario. Una solución común y que se nos pide hacer en el proyecto es la de transformar la red original en un árbol de expansión mínima que mantiene conectividad con la menor cantidad de enlaces posibles.

En nuestro caso, trabajamos con una red que estuvo compuesta por 4 nodos:

- **Dylan**
- **Alan**
- **Juan**
- **Luis**

Cada par de nodos tiene un ancho de banda que se mide como en capacidad de transmisión en Mbps, que fue representado como un grafo en un punto anterior de este proyecto.

Construcción del Grafo

Aquí podemos ver como representamos el grafo original basado en los anchos de banda disponibles entre nodos:

```
ANCHO_BANDA = {
    "Dylan": {"Alan": 50, "Luis": 70, "Juan": 90},
    "Alan": {"Dylan": 50, "Juan": 40},
    "Luis": {"Dylan": 70, "Juan": 20},
    "Juan": {"Dylan": 90, "Alan": 40, "Luis": 20}
}
```

Eso significa por ejemplo, que entre Dylan y Juan hay un enlace de 90 Mbps, entre Alan y Juan hay 40 Mbps, etc.

Árbol de Expansión Mínima con Kruskal

La idea de usar el algoritmo de Kruskal es encontrar un árbol que conecte todos los nodos de la red con el menor costo posible y sin que se formen ciclos. Ese árbol se llama Árbol de Expansión Mínima (MST).

En nuestro grafo, cada conexión entre dos nodos tiene un valor que representa el costo o peso de esa conexión, lo que va a hacer Kruskal es elegir primero las conexiones más baratas y las va agregando una por una pero siempre cuidando que no se forme ningún ciclo hasta que todos los nodos estén conectados.

Al hacer esto, conseguimos una topología más sencilla que la original, porque elimina conexiones innecesarias o caras pero sin perder la conexión entre los nodos.

Diagrama

En el diagrama se puede ver:

- La red original, que tiene todas las conexiones que estaban desde un principio, incluyendo algunas que pueden no ser necesarias o que son más costosas.

- La red después de aplicar Kruskal, donde sólo están las conexiones necesarias para mantener todo conectado con el menor costo total.

```
--- Topología Original (Ancho de Banda) ---

Grafo Original:
  Dylan -- 50 Mbps --> Alan
  Dylan -- 70 Mbps --> Luis
  Dylan -- 90 Mbps --> Juan
  Alan -- 50 Mbps --> Dylan
  Alan -- 40 Mbps --> Juan
  Luis -- 70 Mbps --> Dylan
  Luis -- 20 Mbps --> Juan
  Juan -- 90 Mbps --> Dylan
  Juan -- 40 Mbps --> Alan
  Juan -- 20 Mbps --> Luis

--- Árbol de Expansión Mínima (Kruskal) ---

MST:
  Dylan -- 50 Mbps --> Alan
  Alan -- 40 Mbps --> Juan
  Alan -- 50 Mbps --> Dylan
  Luis -- 20 Mbps --> Juan
  Juan -- 20 Mbps --> Luis
  Juan -- 40 Mbps --> Alan
```

Comparación entre la topología original y la propuesta por Kruskal

En la red original, las conexiones entre los nodos tenían anchos de banda diferentes, que indican la capacidad de cada enlace. Por ejemplo, Dylan está conectado con Juan con un enlace de 90 Mbps, mientras que Luis tiene una conexión de 20 Mbps con Juan.

Al aplicar el algoritmo de Kruskal para obtener el Árbol de Expansión Mínima, el resultado fue una red más simple, con menos conexiones pero que aún conecta todos los nodos. En este MST las conexiones de mayor costo o que estaban repetidas se eliminaron, dejando nada más las necesarias para mantener la red conectada.

Podemos ver que en la topología original había enlaces como Dylan-Juan (90 Mbps) y Dylan-Luis (70 Mbps), pero en el MST estas conexiones desaparecen y priorizamos enlaces con menor ancho de banda, como Luis-Juan (20 Mbps) y Alan-Juan (40 Mbps).

Eso significa que aunque el MST reduce la cantidad de enlaces y el costo total de la red, también se puede reducir la capacidad máxima entre algunos nodos. Pero podemos decir que el MST cumple con su objetivo principal que es el de conectar todos los nodos con el menor costo posible.

Ventajas de usar el MST en la red

- **Menos costo:** Usamos las conexiones más baratas.
- **Más simple:** La red es más fácil de manejar porque tiene menos conexiones.
- **Todos estamos conectados:** Aunque simplificamos, todos los nodos siguen estando comunicados.

Conclusión

En este proyecto abordamos el diseño y evaluación de una red privada virtual para la transferencia de archivos, iniciando con WireGuard para interconectar cuatro nodos y midiendo sus métricas de latencia y ancho de banda mediante ICMP e iperf3. Sin embargo, las restricciones de NAT y firewall nos impidieron establecer túneles estables, por lo que migramos a TileScale como solución de overlay network. Gracias a TileScale pudimos completar la interconexión y captar datos cruciales para modelar la topología como un grafo ponderado.

De los resultados se desprende que existen variaciones notables entre enlaces: el tramo **Dylan–Carlos** destaca como cuello de botella con la mayor latencia y menor capacidad de transferencia, mientras que **Alan–Dylan** ofrece el mejor desempeño. Esta caracterización no solo valida nuestra infraestructura, sino que también sirve de base para optimizar rutas priorizando los enlaces más eficientes.

Como líneas futuras, proponemos automatizar la reconfiguración de rutas en tiempo real según las métricas recopiladas y explorar técnicas de compresión o paralelización de flujos. Asimismo, sería interesante comparar de forma sistemática el rendimiento de WireGuard frente a TileScale en distintos escenarios de red.

Bibliografía

De Luz, S. (2025, 8 abril). Tailscale, conecta equipos a una red privada virtual (VPN) fácilmente. *RedesZone*. https://www.redeszone.net/tutoriales/vpn/configurar-tailscale-red-vpn-segura/

Fontela, Á. (2025, 4 febrero). *Instalar y configurar Tailscale en un servidor VPS*. Raiola Networks - Dominios y Alojamiento Web de Calidad. https://raiolanetworks.com/ayuda/instalar-configurar-tailscale-servidor-vps/

What is Dijkstra Algorithm - Cybersecurity Terms and Definitions. (s. f.). https://www.vpnunlimited.com/help/cybersecurity/dijkstra-algorithm?srsltid=AfmBOorU-f5zIqlsLVTXTKKQyikKN62yM8xyjhv2JXpowHf9sqw3DGLv

Árboles de peso mínimo: algoritmos de Prim y Kruskal — Matemáticas Discretas para Ciencia de Datos. (s. f.). https://madi.nekomath.com/P5/ArbolPesoMin.html

tkinter — Python interface to Tcl/Tk. (s. f.). Python Documentation. https://docs.python.org/es/3/library/tkinter.html?authuser=0#coupling-widget-variables

Castañeira, L. (2023, 19 junio). TKinter: Crea interfaces gráficas en Python de forma sencilla. *Medium*. https://medium.com/@solidlucho/tkinter-crea-interfaces-gr%C3%A1ficas-en-python-de-forma-sencilla-50d131f84883

speedtest-cli. (2021, 8 abril). PyPI. https://pypi.org/project/speedtest-cli/

socket — Low-level networking interface. (s. f.). Python Documentation. https://docs.python.org/3/library/socket.html