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 empleará **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. Ademas 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 la mejor para este proyecto:

En este caso dado el tiempo y el tamaño del equipo elegimos que la metodología que vamos a usar va a ser la scrum. Planeamos hacer 2 sprints de 6 días empezando hoy 30 de abril y siendo el 10 de mayo el último día de sprint para el 11 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.

La asignación en cuanto scrum será que:

- 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.

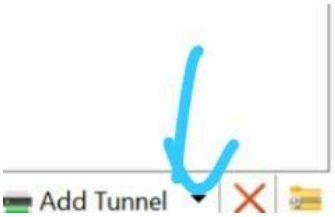
Dado el proyecto y que puede llegar a ser complejo la idea es poder hacer 2 daily meeting o más por día, esto para ir viendo nuestro progreso y en caso de tener problemas en alguna parte de codificaciones poder ayudarnos entre todos para así dar los entregables que corresponde al día.

Reportaje de la vpn

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

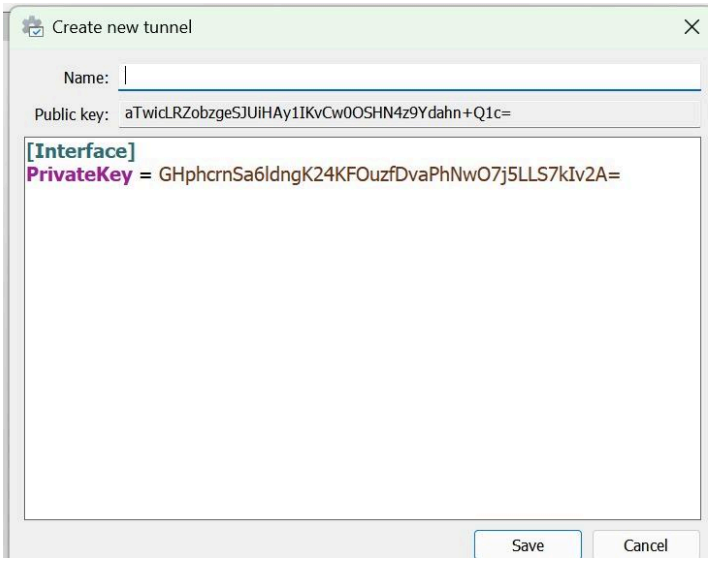
- Creación del servidor
- Creación de los usuarios

Una vez dentro de WireGuard necesitamos seleccionar esta opción:

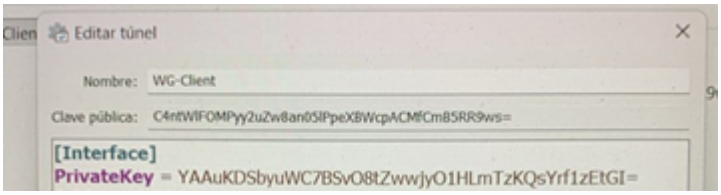


Una vez que seleccionamos eso nos aparecerá el siguiente recuadro:

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:

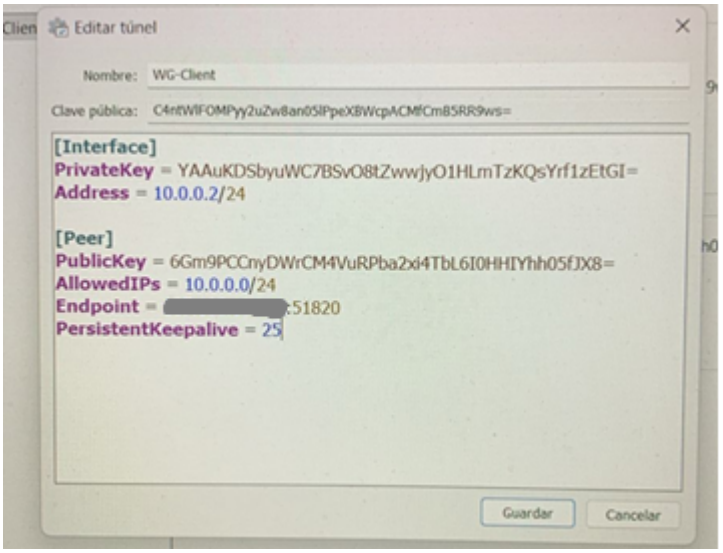


Una vez que le damos clic al tunnel que hemos creado nos aparece la información de la imagen, de la cual tendremos la clave pública y la clave privada.
La clave pública o “public key” se la tendremos que proporcionar a nuestro compañero encargado de crear el servidor.

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.



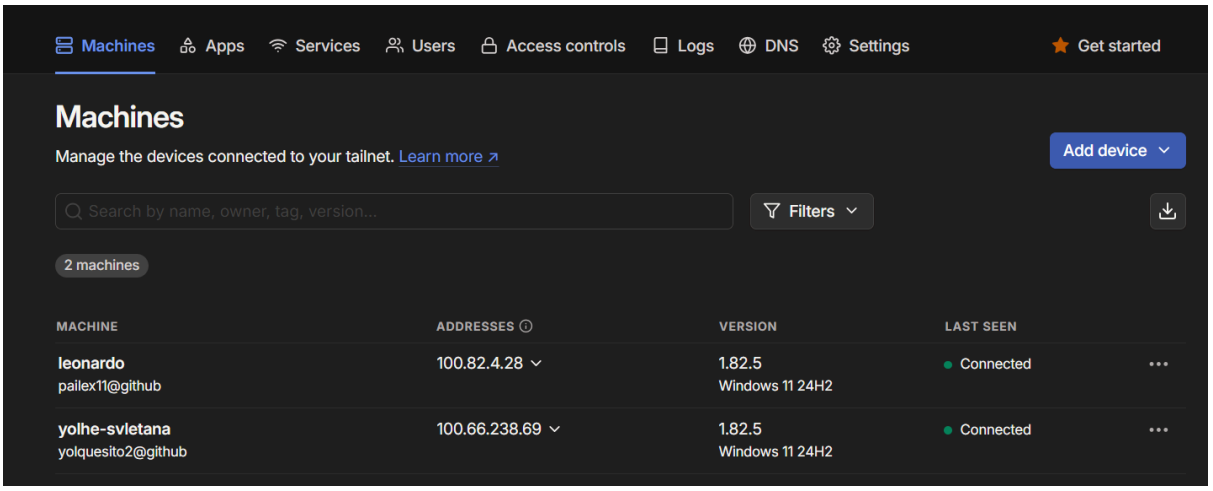
Solo para tener un mejor control y una “copia de seguridad” lo que haremos será guardar esta configuración en un archivo .zip, lo haremos dándole clic a la carpeta que está a la derecha de “Add Túnel” en la parte inferior izquierda y ya solo seleccionamos en donde queremos que se guarde y le ponemos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

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.



Ping

```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\L13Yoga>ping 100.82.4.28

Pinging 100.82.4.28 with 32 bytes of data:
Request timed out.
Request timed out.
Reply from 64.76.240.89: Destination net unreachable.
Reply from 64.76.240.89: Destination net unreachable.

Ping statistics for 100.82.4.28:
    Packets: Sent = 4, Received = 2, Lost = 2 (50% loss),

C:\Users\L13Yoga>
```

Mediciones de comunicación:

[Metodo utilizado para la medicion]

Utilizamos el metodo de ping porque es mas rapido y efectivo que un script o por lo menos en nuestro caso fue mas sencillo, obteniendo asi los siguientes resultados:

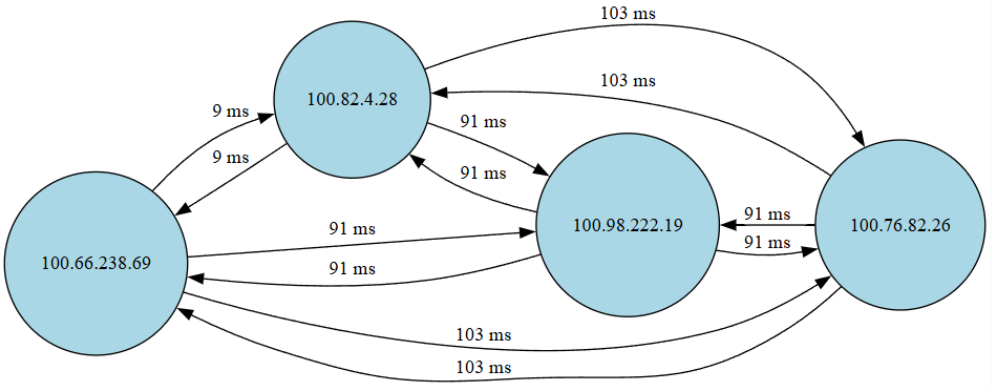
```
C:\Users\Lexpo>ping 100.66.238.69

Haciendo ping a 100.66.238.69 con 32 bytes de datos:
Respuesta desde 100.66.238.69: bytes=32 tiempo=43ms TTL=128
Respuesta desde 100.66.238.69: bytes=32 tiempo=12ms TTL=128
Respuesta desde 100.66.238.69: bytes=32 tiempo=10ms TTL=128
Respuesta desde 100.66.238.69: bytes=32 tiempo=7ms TTL=128

Estadísticas de ping para 100.66.238.69:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 7ms, Máximo = 43ms, Media = 18ms

C:\Users\Lexpo>ping 100.82.4.28

Haciendo ping a 100.82.4.28 con 32 bytes de datos:
Respuesta desde 100.82.4.28: bytes=32 tiempo<1m TTL=128
Respuesta desde 100.82.4.28: bytes=32 tiempo<1m TTL=128
Respuesta desde 100.82.4.28: bytes=32 tiempo<1m TTL=128
Respuesta desde 100.82.4.28: bytes=32 tiempo<1m TTL=128

Estadísticas de ping para 100.82.4.28:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 0ms, Máximo = 0ms, Media = 0ms
```

```
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
```

[Medicion de ancho de banda (SppedTest)]

```
   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)
```

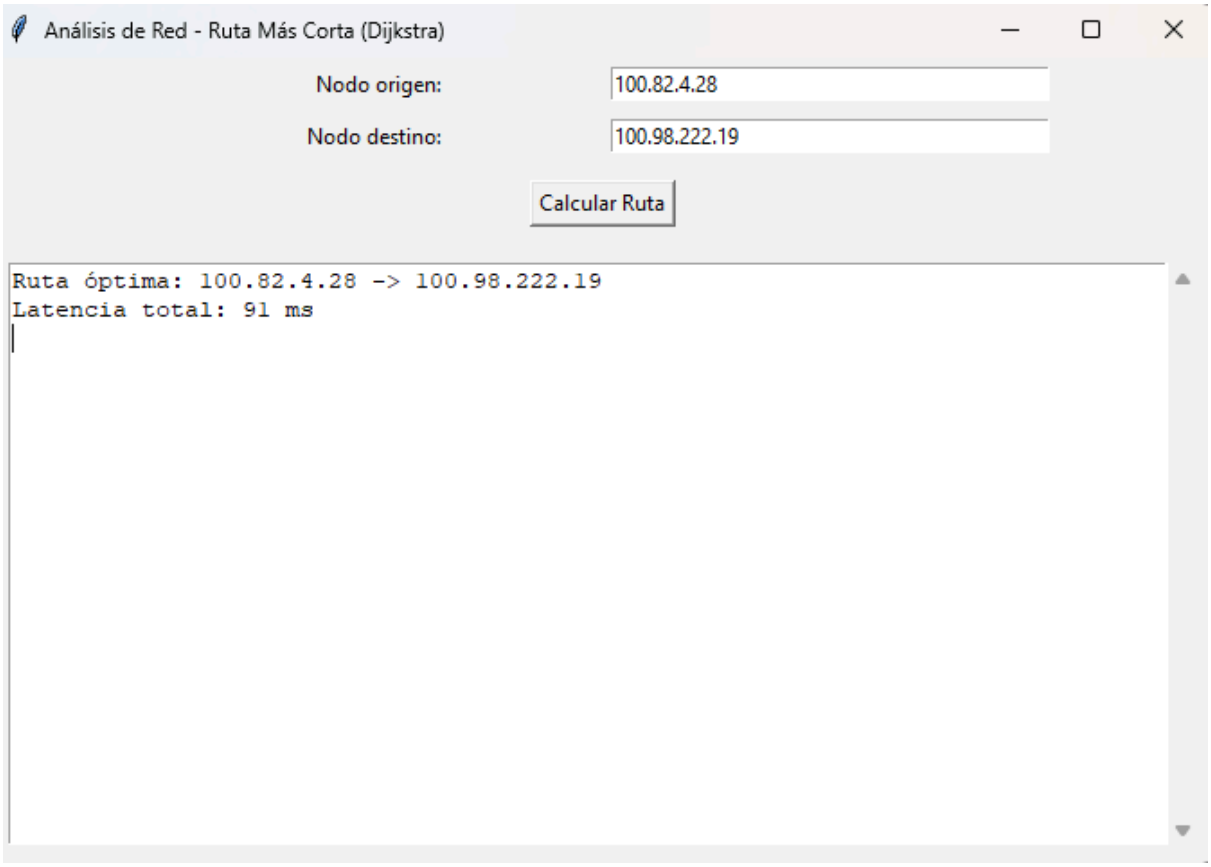
Grafo Ponderado



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.

[Gafro de latencias y dijkstra]



Código Dijkstra:

Para empezar con la explicación de este código es fundamental la explicación de las librerías que vamos a utilizar.

```python
import heapq
import json
import subprocess
import re
import tkinter as tk
from tkinter import messagebox, scrolledtext
```

“heapq”: Es una librería estándar de Python que proporciona una implementación de colas de prioridad (heaps). La utilizamos en el algoritmo de Dijkstra para seleccionar el siguiente nodo con la menor latencia de forma eficiente.
“json”: Es una librería estándar para trabajar con datos en formato JSON. La utilizaremos para parsear la salida JSON del comando “tailscale status --json”, que contiene la información de los nodos de la red.
“subprocess”: Es un módulo estándar que permite ejecutar comandos del sistema operativo desde Python. La utilizaremos para ejecutar “tailscale status --json” y

obtener la red. Además de ejecutar “tailscale ping <nodo>” y medir la latencia real entre nodos.
“re”: Es una librería estándar para trabajar con expresiones regulares en Python. Para extraer la latencia en milisegundos desde la salida del comando “tailscale ping”, que normalmente viene en texto.
“tkinter”: Esta librería la utilizaremos principalmente para la elaboración completa de la GUI.
“messagebox”: Es un módulo de utilidades gráficas que forma parte de tkinter. Permite mostrar cuadros de diálogo emergentes (pop-ups) para dar información al usuario o pedir confirmación. Se utiliza para mostrar mensajes de alerta, advertencia o confirmación en ventanas emergentes.
“scrolledtext”: Es un widget de tkinter que permite mostrar y editar texto largo, con barra de desplazamiento automática integrada. Se usa para mostrar salida extensa, como logs, rutas calculadas, resultados de pings, errores, etc.

Damos inicio a la explicación del código:

```
def get_tailscale_network(log_callback):
```

Función para obtener grafo de la red Tailscale: aquí se define una función que devuelve un grafo con latencias entre nodos usando pings.

```
try:
    result = subprocess.run(["tailscale", "status", "--json"], capture_output=True, text=True)
    data = json.loads(result.stdout)
```

Se ejecuta tailscale status --json para obtener información de la red en formato JSON. La salida (result.stdout) se convierte en un diccionario con json.loads.

```
except Exception as e:
    messagebox.showerror("Error", f"No se pudo obtener el estado de Tailscale: {e}")
    return {}
```

En caso de que algo falle (no se puede ejecutar el comando o no hay JSON válido), muestra un error y devuelve un grafo vacío.

```
nodes = [info['DNSName'] for _, info in data['Peer'].items()]
```

Aquí se extrae la lista de nombres DNS de todos los pares Tailscale conectados.

```
graph = {node: {} for node in nodes}
```

Se crea un diccionario graph donde cada nodo es una clave y su valor será otro diccionario con vecinos y latencias.

```
for src in nodes:
    for dst in nodes:
        if src == dst:
            continue
```

El for recorre todos los pares nodos src → dst) excepto cuando son iguales y no se hace ping a sí mismo.

```
try:
    ping_result = subprocess.run(
        ["tailscale", "ping", dst, "--timeout=1s"],
        capture_output=True,
        text=True
    )
```

Se ejecuta tailscale ping dst --timeout=1s, para así intentar medir la latencia desde src hasta dst.

```
match = re.search(r"in (\d+\.?\d*)ms", ping_result.stdout)
```

Aquí busca en la salida una expresión como "in 10.3ms" para extraer la latencia.

```
if match:
    latency = float(match.group(1))
    graph[src][dst] = latency
    log_callback(f"{src} → {dst}: {latency} ms")
```

En caso de si encontrar latencia, se guarda en el grafo como graph[src][dst] = latencia. Y también se registra en la interfaz.

```
else:
    log_callback(f"{src} → {dst}: sin respuesta")
```

En caso de no detectar latencia se indica que no hubo respuesta.

```
except Exception as e:
    log_callback(f"Error al hacer ping de {src} a {dst}: {e}")
```

En caso de haber cualquier error durante el ping se registra.

```
return graph
```

Devuelve el grafo de latencias entre todos los nodos.

```
def dijkstra(graph, start):
```

Se implementa la función Dijkstra para encontrar la ruta más corta (mínima latencia) desde start.

```
distances = {node: float('inf') for node in graph}
distances[start] = 0
```

En esta parte se inicializa la distancia a todos los nodos como infinito, excepto el nodo inicial (start) que es 0.

```
previous_nodes = {node: None for node in graph}
```

Aquí se guarda el nodo anterior en la ruta óptima hacia cada nodo.

```
priority_queue = [(0, start)]
```

Es la cola de prioridad donde se insertan nodos según su distancia.

```
while priority_queue:
    current_distance, current_node = heapq.heappop(priority_queue)
```

En esta parte se extrae el nodo con menor latencia acumulada.

```
if current_distance > distances[current_node]:
    continue
```

Aquí ignora si ya se ha encontrado una mejor ruta antes.

```
for neighbor, weight in graph[current_node].items():
    distance = current_distance + weight
```

Aquí para cada vecino, calcula la distancia total a través del nodo actual.

```
if distance < distances[neighbor]:
    distances[neighbor] = distance
    previous_nodes[neighbor] = current_node
    heapq.heappush(priority_queue, (distance, neighbor))
```

Si se encuentra una mejor ruta, se actualizan los registros y se agrega el vecino a la cola.

```
return distances, previous_nodes
```

Devuelve los costos mínimos y los nodos previos para reconstruir rutas.

```
def reconstruct_path(prev_nodes, end):
    path = []
    while end is not None:
        path.insert(0, end)
        end = prev_nodes[end]
    return path
```

Usa previous_nodes para reconstruir el camino desde end hacia el inicio. Inserta los nodos al principio de la lista para que el camino quede en orden correcto.

```
class TailscaleGUI:
```

Esta clase se encarga de construir la interfaz y manejar la interacción del usuario.

```
def __init__(self, root):
```

El constructor recibe la ventana principal (root) de Tkinter.

```
self.root = root
self.root.title("Análisis de Red Tailscale")
```

Guarda la ventana principal en self.root y le pone un título.

```
tk.Label(root, text="Nodo origen:").grid(row=0, column=0, sticky="e")
tk.Label(root, text="Nodo destino:").grid(row=1, column=0, sticky="e")
```

Se crean dos etiquetas para los campos de entrada "Nodo origen" y "Nodo destino". Aquí, sticky="e" alinea el texto a la derecha dentro de su celda.

```
self.entry_start = tk.Entry(root, width=40)
self.entry_end = tk.Entry(root, width=40)
```

Dos cajas de entrada para que el usuario escriba el nodo de inicio y el de destino.

```
self.entry_start.grid(row=0, column=1, padx=5, pady=5)
self.entry_end.grid(row=1, column=1, padx=5, pady=5)
```

Coloca las cajas de entrada en la interfaz con espacio (padding) alrededor.

```
self.button_run = tk.Button(root, text="Analizar Red", command=self.run_analysis)
self.button_run.grid(row=2, column=0, columnspan=2, pady=10)
```

Crea un botón "Analizar Red" que al hacer clic ejecuta self.run_analysis.

```
tk.Label(root, text="Archivo local:").grid(row=3, column=0, sticky="e")
self.entry_file = tk.Entry(root, width=60)
self.entry_file.grid(row=3, column=1, padx=5, pady=5)
```

Etiqueta y campo de entrada para el archivo que se desea transferir.

```
tk.Label(root, text="Destino (usuario@host:/ruta):").grid(row=4, column=0, sticky="e")
self.entry_target = tk.Entry(root, width=60)
self.entry_target.grid(row=4, column=1, padx=5, pady=5)
```

Etiqueta y entrada para el destino del archivo (ej. usuario@host:/ruta).

```
self.button_transfer = tk.Button(root, text="Transferir archivo", command=self.transfer_file)
self.button_transfer.grid(row=5, column=0, columnspan=2, pady=10)
```

Botón "Transferir archivo" que llama al método self.transfer_file al hacer clic.

```
self.output = scrolledtext.ScrolledText(root, width=80, height=20)
self.output.grid(row=6, column=0, columnspan=2, padx=10, pady=10)
```

Crea un cuadro de texto con barra de desplazamiento para mostrar mensajes y resultados.

```
def log(self, message):
    self.output.insert(tk.END, message + "\n")
    self.output.see(tk.END)
```

Imprime un mensaje al final del cuadro de texto y se desplaza automáticamente hacia el final para que se vea.

```
def run_analysis(self):
    self.output.delete(1.0, tk.END)
```

Limpia el área de salida antes de iniciar el análisis.

```
start = self.entry_start.get().strip()
end = self.entry_end.get().strip()
```

Lee el nodo origen y destino desde las cajas de texto.

```
if not start or not end:
    messagebox.showwarning("Entrada incompleta", "Por favor, ingresa ambos nodos.")
    return
```

Muestra advertencia si no se completaron ambos campos.

```
self.log(f" Escaneando red Tailscale...")
graph = get_tailscale_network(self.log)
```

Llama a get_tailscale_network, pasando el método self.log para mostrar los resultados.

```
if start not in graph or end not in graph:
    messagebox.showerror("Nodos inválidos", "Uno o ambos nodos no se encontraron en la red.")
    return
```

Verifica que ambos nodos existan en el grafo.

```
distances, prev = dijkstra(graph, start)
path = reconstruct_path(prev, end)
```

Ejecuta el algoritmo de Dijkstra y reconstruye el camino desde “start” hasta “end”.

```
if distances[end] == float('inf'):
    self.log(f"\n No hay ruta disponible de {start} a {end}.")
else:
    self.log(f"\n Ruta óptima: {' -> '.join(path)}")
    self.log(f" Latencia total: {distances[end]} ms")
```

Muestra la ruta más corta o informa que no hay conexión entre los nodos.

```
def transfer_file(self):
    archivo = self.entry_file.get().strip()
    destino = self.entry_target.get().strip()
```

Obtiene la ruta del archivo y el destino del formulario.

```
if not archivo or not destino:
    messagebox.showwarning("Campos vacíos", "Debes especificar el archivo y el destino.")
    return
```

Verifica que ambos campos estén completos.

```
self.log(f"\n Transfiriendo archivo: {archivo} → {destino}")
```

Registra en el área de texto que comenzó la transferencia.

```
try:
    result = subprocess.run(
        ["scp", archivo, destino],
        capture_output=True,
        text=True
    )
```

Ejecuta el comando scp para copiar el archivo a través de la red.

```
if result.returncode == 0:
    self.log(" Transferencia completada exitosamente.")
else:
    self.log(f" Error en la transferencia:\n{result.stderr}")
```

Si el código de salida es 0, la transferencia fue exitosa. Si no, muestra el error.

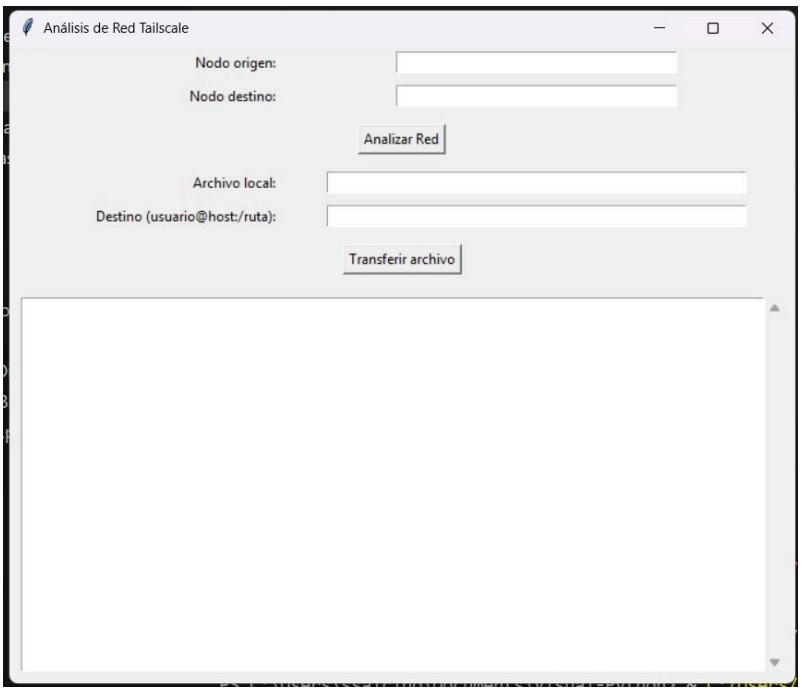
```
except Exception as e:
    self.log(f" Excepción durante la transferencia: {e}")
```

Captura y muestra cualquier excepción ocurrida durante la ejecución del comando.

```
root = tk.Tk()
app = TailscaleGUI(root)
root.mainloop()
```

Crea la ventana principal (root).
Inicializa la interfaz con la clase TailscaleGUI.
Inicia el bucle principal de Tkinter (el código queda abierto esperando eventos).

Evidencia de la GUI y transferencia exitosa.



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:

Iniciamos con la explicación arrancando por las librerías.

```
import networkx as nx
import matplotlib.pyplot as plt
```

“networkx”: Es una librería para trabajar con grafos en Python.
“matplotlib.pyplot”: Es para generar gráficos (visualización de topologías).

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

Este es un ejemplo, a la hora de la ejecución lo tendremos que cambiar. Pero básicamente es una lista de aristas del grafo original. Cada tupla representa: (nodo1, nodo2, ancho_de_banda).

```
class DisjointSet:
    def __init__(self, vertices):
        self.parent = {v: v for v in vertices}
```

Inicializa la estructura. Cada nodo es su propio “padre” inicialmente.

```
def find(self, v):
    if self.parent[v] != v:
        self.parent[v] = self.find(self.parent[v])
    return self.parent[v]
```

Encuentra el representante de un nodo y hace compresión de caminos para optimizar el tiempo.

```
def union(self, u, v):
    root_u = self.find(u)
    root_v = self.find(v)
    if root_u != root_v:
        self.parent[root_u] = root_v
        return True
    return False
```

Une dos conjuntos si no están conectados ya. Retorna True si la unión fue posible (no forma ciclo), False si no.

```
def kruskal(vertices, edges):
    ds = DisjointSet(vertices)
    mst = []
    total_bandwidth = 0
```

Crea el conjunto disjunto y prepara la lista del MST. “total_bandwidth” sumará los anchos usados por el MST.

```
edges.sort(key=lambda x: x[2])
```

Ordena las aristas por peso (menor ancho de banda primero).

```
for u, v, weight in edges:
    if ds.union(u, v):
        mst.append((u, v, weight))
        total_bandwidth += weight
```

Para cada arista, si los nodos no forman ciclo (union == True), se agrega al MST.

```
return mst, total_bandwidth
```

Devuelve el MST (lista de enlaces) y su peso total.

```
nodes = set()
for u, v, _ in edges:
    nodes.add(u)
    nodes.add(v)
```

Recolecta todos los nodos únicos del grafo original.

```
mst, mst_bandwidth = kruskal(nodes, edges)
```

Llama a la función de Kruskal para obtener el MST y su peso total.

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

Calcula el total del ancho de banda de toda la red original.

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

Imprime cada conexión de la topología original.

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

Muestra las conexiones seleccionadas por Kruskal (topología optimizada).

```
print(f"\nAhorro: {original_bandwidth - mst_bandwidth} unidades\n")
```

Imprime el ahorro total de ancho de banda.

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

Crea un grafo de networkx y le agrega los nodos/conexiones.

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

Define posiciones automáticas de los nodos para la visualización. Extrae los pesos de las aristas para mostrar como etiquetas.

```python
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 colores, etiquetas y pesos de las conexiones.

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

Genera los diagramas visuales: uno del grafo completo, y otro del MST.

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:

```python
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)

Diagrama de topología (Antes)

Diagrama de topología (Después)

Cronograma:

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 Review y entrega (11 de mayo)

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

Cronograma de Gantt

	30-Apr	01-may	02-may	03-may	04-may	05-may	06-may	07-may	08-may	09-may	10-may	11-may	12-may
Planeación y asignación de tareas													
Configuración de VPN													
Implementación base de Dijkstra													
Implementación base de Kruskal													
Mediciónes de red													
Integrar Dijkstra con la transferencia real de archivos.													
optimizar Kruskal													
Realizar mediciones para evaluar mejoras en velocidad y eficiencia													
Elaborar documentación													
Presentar al profesor la topología, métricas y resultados obtenidos													
Demo funcional de la transferencia optimizada													
Entrega de la documentación final.													

Reporte de Meetings

Meeting 5 mayo (duración 15 minutos)

Temas tratados:

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.

Meeting 10 mayo (duración 20 minutos)

Temas tratados:

· Verificación de funcionamiento de la VPN (primeros dos nodos conectados).

· Se reportaron avance en la estructura del algoritmo de Dijkstra.

· Se mostró resultados iniciales de latencias

Conclusión

A lo largo del proyecto se trabajó en la optimización de la transferencia de archivos en una red VPN utilizando algoritmos voraces, específicamente Dijkstra y Kruskal, integrando conocimientos de redes, algoritmos y estructuras de datos. Se observó que Dijkstra es eficaz para encontrar rutas rápidas basadas en latencias, mientras que Kruskal permite construir una topología mínima eficiente en el uso del ancho de banda. Sin embargo, también se identificaron desventajas: Dijkstra no considera el estado dinámico de la red y Kruskal no toma en cuenta las latencias. A pesar del avance en la implementación, no fue posible realizar la prueba de transferencia real de archivos, ya que no se logró establecer conexión entre los dispositivos a través de Tailscale, lo que limitó la validación práctica de los algoritmos. Esta actividad permitió reforzar habilidades técnicas y de trabajo en equipo, y evidenció la importancia de adaptar herramientas y soluciones según las limitaciones del entorno real.

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%A

9%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. Setting up a server on your Tailscale network · Tailscale Docs
11.

Recursos:

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

Github:
https://github.com/Yolhe/vpn_voraces_polinesios

https://colab.research.google.com/drive/13ELNr3sH97OfvhR2Mnj0IYKhuNXz7anZ?usp=sharing