

Optimización de Transferencia de Archivos en una VPN con Algoritmos Voraces

Equipo: Polinesios

José Eloy Hernández Olivera
Sebastián Salcido Solorio
Arlette Michelle Castañeda Solís
Leonardo Axel Nochebuena

ELOY

IMPLEMENTACIÓN DE UNA RED PRIVADA VIRTUAL PARA COLABORACIÓN REMOTA

El Desafío Inicial

- Necesidad de conectar de forma segura y eficiente a un equipo de trabajo distribuido geográficamente.
- Requerimiento de compartir archivos de manera directa y establecer una comunicación fluida entre nodos.
- Intento inicial con WireGuard resultó complejo debido a la configuración manual de claves, interfaces y direccionamiento IP.

ELOY

IMPLEMENTACIÓN DE UNA RED PRIVADA VIRTUAL PARA COLABORACIÓN REMOTA

La Solución: Tailscale

- Selección de Tailscale como una alternativa más accesible y con configuración simplificada.
- Facilidad de Implementación: Instalación de la aplicación cliente en los dispositivos de cada miembro del equipo.
- Autenticación Centralizada: Inicio de sesión mediante cuentas de GitHub (o un proveedor de identidad común), lo que permitió la unión automática a una misma red privada virtual ("tailnet").

IMPLEMENTACIÓN DE UNA RED PRIVADA VIRTUAL PARA COLABORACIÓN REMOTA

Conexión Automática y Gestión de Nodos

- Una vez autenticados, los dispositivos se conectaron automáticamente a la tailnet, sin necesidad de configuración manual de red.
- Tailscale asignó direcciones IP privadas (100.x.y.z) a cada nodo, facilitando la identificación y comunicación.
- Visualización clara de los dispositivos conectados y sus IPs en la aplicación cliente y el panel de administración web.

MACHINE	ADDRESSES ⓘ
leonardo pailex11@github Shared out +1	100.82.4.28 ▾
samsung-sm-s918b pailex11@github	100.76.82.26 ▾
xiaomi-2109119dg yolquesito2@github	100.98.222.19 ▾
yolhe-svletana yolquesito2@github	100.66.238.69 ▾

ELOY

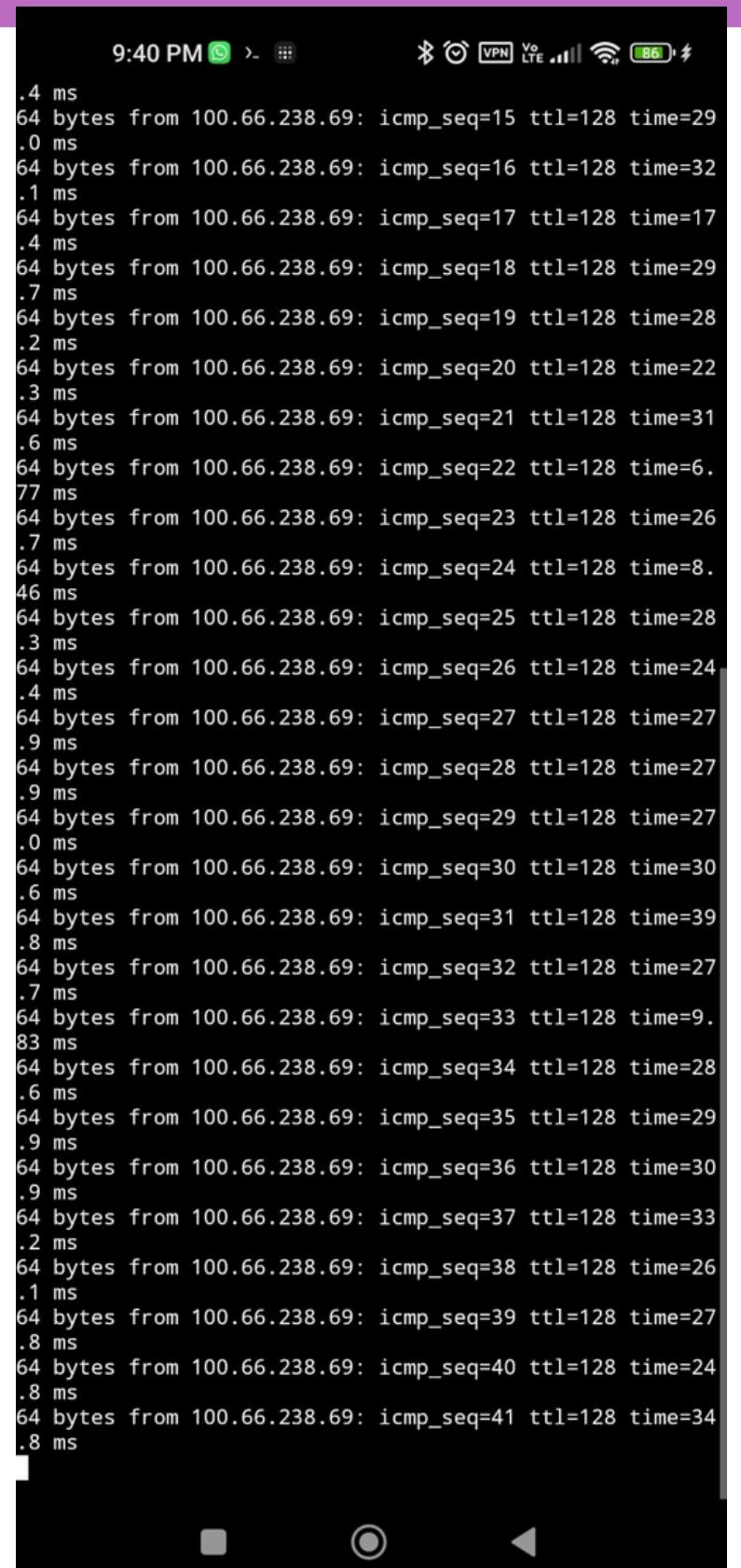
IMPLEMENTACIÓN DE UNA RED PRIVADA VIRTUAL PARA COLABORACIÓN REMOTA

esto desde mi
compu al
telefono

```
C:\Users\L13Yoga>ping 100.98.222.19

Pinging 100.98.222.19 with 32 bytes of data:
Reply from 100.98.222.19: bytes=32 time=68ms TTL=64
Reply from 100.98.222.19: bytes=32 time=25ms TTL=64
Reply from 100.98.222.19: bytes=32 time=42ms TTL=64
Reply from 100.98.222.19: bytes=32 time=37ms TTL=64

Ping statistics for 100.98.222.19:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 25ms, Maximum = 68ms, Average = 43ms
```



LEONARDO

```
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

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
```

Se recibieron los paquetes de los 4 nodos que se pusieron, incluyendo al host y esto se logro por medio de ping, no se necesito algo como un script ya que no fue necesario

IMPLEMENTACIÓN DEL ALGORITMO DIJKSTRA

Se crea la función get_tailscale_network que construye un grafo de latencias de red usando tailscale para obtener los nodos conectados

```
def get_tailscale_network(log_callback):
    """
    Construye un grafo de la red Tailscale usando 'tailscale ping'
    y devuelve un diccionario con nodos y latencias.
    """

    try:
        result = subprocess.run(["tailscale", "status", "--json"], capture_output=True, text=True)
        data = json.loads(result.stdout)
    except Exception as e:
        messagebox.showerror("Error", f"No se pudo obtener el estado de Tailscale: {e}")
        return {}

    nodes = [info['DNSName'] for _, info in data['Peer'].items()]
    graph = {node: {} for node in nodes}
```

IMPLEMENTACIÓN DEL ALGORITMO DIJKSTRA

implementamos el algoritmo de dijkstra para encontrar la ruta mas corta

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

    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_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(priority_queue, (distance, neighbor))

    return distances, previous_nodes

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

PROTOCOLO UTILIZADO PARA LA TRANSFERENCIA DE ARCHIVOS

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.

IMPLEMENTACIÓN DE KRUSKAL

Este código es para diseñar una topología de red eficiente. Esto lo hacemos mediante MST (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).

IMPLEMENTACIÓN DE KRUSKAL

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

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

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

    return mst, total_bandwidth
```

Esta función toma un grafo de conexiones y usa el algoritmo de Kruskal para:

- Ordenar las conexiones por costo.

- Agregar las más baratas que no formen ciclos.

- Construir el MST que conecta todos los nodos usando el menor ancho de banda total posible.

MUCHAS GRACIAS