

✓ Problema 1: Diseño y Simulación de una red metro Ethernet con QoS

```
!pip install netmiko
```

```
Collecting netmiko
  Downloading netmiko-4.3.0-py3-none-any.whl (219 kB)
    219.2/219.2 kB 4.6 MB/s eta 0:00:00
Collecting ntc-templates>=2.0.0 (from netmiko)
  Downloading ntc_templates-5.0.0-py3-none-any.whl (450 kB)
    450.9/450.9 kB 14.8 MB/s eta 0:00:00
Collecting paramiko>=2.9.5 (from netmiko)
  Downloading paramiko-3.4.0-py3-none-any.whl (225 kB)
    225.9/225.9 kB 12.0 MB/s eta 0:00:00
Collecting pyserial>=3.3 (from netmiko)
  Downloading pyserial-3.5-py2.py3-none-any.whl (90 kB)
    90.6/90.6 kB 11.6 MB/s eta 0:00:00
Requirement already satisfied: pyyaml>=5.3 in /usr/local/lib/python3.10/dist-packages (from netmiko) (6.0.1)
Collecting scp>=0.13.6 (from netmiko)
  Downloading scp-0.14.5-py2.py3-none-any.whl (8.7 kB)
Collecting textfsm>=1.1.3 (from netmiko)
  Downloading textfsm-1.1.3-py2.py3-none-any.whl (44 kB)
    44.7/44.7 kB 4.8 MB/s eta 0:00:00
Collecting bcrypt>=3.2 (from paramiko>=2.9.5->netmiko)
  Downloading bcrypt-4.1.2-cp39-abi3-manylinux_2_28_x86_64.whl (698 kB)
    698.9/698.9 kB 46.6 MB/s eta 0:00:00
Requirement already satisfied: cryptography>=3.3 in /usr/local/lib/python3.10/dist-packages (from paramiko>=2.9.5->netmiko) (42)
Collecting pynacl>=1.5 (from paramiko>=2.9.5->netmiko)
  Downloading PyNaCl-1.5.0-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_24_x86_64.whl (856 kB)
    856.7/856.7 kB 52.7 MB/s eta 0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from textfsm>=1.1.3->netmiko) (1.16.0)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from textfsm>=1.1.3->netmiko) (0.18.3)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography>=3.3->paramiko>=2.9.5->netmiko) (1.16.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography>=3.3->paramiko>=2.9.5->netmiko) (2.21)
Installing collected packages: pyserial, textfsm, bcrypt, pynacl, ntc-templates, paramiko, scp, netmiko
Successfully installed bcrypt-4.1.2 netmiko-4.3.0 ntc-templates-5.0.0 paramiko-3.4.0 pynacl-1.5.0 pyserial-3.5 scp-0.14.5 textfsm-1.1.3
```

```
from netmiko import ConnectHandler
```

✓ Parte 1: Diseño de red utilizando VLANs y troncales

```
def configure_switch(ip_address, username, password):
# Crear una conexión al switch
switch = {
    'device_type': 'cisco_ios',
    'ip': ip_address,
    'username': username,
    'password': password,
}
# Comandos para configurar las VLANs y troncales
commands = [
    'vlan 10',
    'name DATA',
    'exit',
    'vlan 20',
    'name VOIP',
    'exit',
    'interface range gig0/1-2',
    'switchport trunk encapsulation dot1q',
    'switchport mode trunk',
    'switchport trunk allowed vlan 10,20',
]
# Iniciar la conexión y enviar comandos
try:
    with ConnectHandler(**switch) as conn:
        output = conn.send_config_set(commands)
        print(output)
        conn.disconnect()
except Exception as e:
    print(f'Ocurrió un error: {e}')
# Ejemplo de uso
configure_switch('192.168.1.100', 'admin', 'password')
```

Ocurrió un error: TCP connection to device failed.

Common causes of this problem are:
1. Incorrect hostname or IP address.

2. Wrong TCP port.
3. Intermediate firewall blocking access.

Device settings: cisco_ios 192.168.1.100:22

✓ Parte 2: Configuración de QoS para priorizar VoIP

```
def configure_qos(ip_address, username, password):
    switch = {
        'device_type': 'cisco_ios',
        'ip': ip_address,
        'username': username,
        'password': password,
    }
    qos_commands = [
        'access-list 101 permit ip any any',
        'class-map match-any VOIP',
        'match access-group 101',
        'exit',
        'policy-map VOIP-Policy',
        'class VOIP',
        'set ip dscp ef',
        'exit',
        'interface gig0/1',
        'service-policy output VOIP-Policy',
    ]
    try:
        with ConnectHandler(**switch) as conn:
            output = conn.send_config_set(qos_commands)
            print(output)
            conn.disconnect()
    except Exception as e:
        print(f"Ocurrió un error: {e}")
# Ejemplo de uso
configure_qos('192.168.1.100', 'admin', 'password')
```

Ocurrió un error: TCP connection to device failed.

Common causes of this problem are:

1. Incorrect hostname or IP address.
2. Wrong TCP port.
3. Intermediate firewall blocking access.

Device settings: cisco_ios 192.168.1.100:22

✓ Parte 3: Simulación y análisis

```
!pip install scapy
```

```
Collecting scapy
  Downloading scapy-2.5.0.tar.gz (1.3 MB)
    1.3/1.3 MB 15.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: scapy
  Building wheel for scapy (setup.py) ... done
  Created wheel for scapy: filename=scapy-2.5.0-py2.py3-none-any.whl size=1444327 sha256=3869d71703fa63f3539cc5a500f1822cb55a48:
  Stored in directory: /root/.cache/pip/wheels/82/b7/03/8344d8cf6695624746311bc0d389e9d05535ca83c35f90241d
Successfully built scapy
Installing collected packages: scapy
Successfully installed scapy-2.5.0
```

```

from scapy.all import *
import random

# Función para generar tráfico de voip
def trafico_voip(ip_switch, Cantidad_paquetes):
    TOS_voip = 0x2E # DSCP EF, para tráfico de alta prioridad
    paquete = IP(dst=ip_switch, tos=TOS_voip)/UDP(dport=5060)/Raw(load="VoIP")
    send(paquete, count=Cantidad_paquetes, inter=random.uniform(0.01,0.1)) # Simulacion del envio de los paquetes void en un interv

# Función para generar tráfico de datos
def trafico_data(ip_switch, TOS_data, Cantidad_paquetes):
    TOS_data = 0x00 # DSCP CS1, no asigna prioridad
    paquete = IP(dst=ip_switch, tos=TOS_data)/TCP(dport=80)/Raw(load="Data")
    send(paquete, count=Cantidad_paquetes, inter=random.uniform(0.01, 0.1)) # Simulacion del envio de los paquetes data en un interv

# Dirección IP del switch
ip_switch = '192.168.1.100'

# Número de paquetes a enviar
Cantidad_paquetes = 100

# Llamamos a las funciones para generar trafico
trafico_voip(ip_switch, Cantidad_paquetes)
trafico_data(ip_switch, Cantidad_paquetes)

```

Sent 100 packets.

Sent 100 packets.

✓ **Problema 3:** Estrategias de mitigación para interferencia en redes Ad Hoc

✓ **Parte 1:** Implementación de CSMA/CA en Python

```

import random
import time
def simulate_csma_ca(node_id, attempt_limit=5):
    attempt = 0
    while attempt < attempt_limit:
        # Sensar el medio (simulado por una función que retorna True si el medio está ocupado)
        if is_channel_busy():
            print(f"Node {node_id}: Canal ocupado, aplicando backoff")
            time_to_wait = exponential_backoff(attempt)
            time.sleep(time_to_wait)
            attempt += 1
        else:
            print(f"Node {node_id}: Canal libre, transmitiendo datos")
            send_data(node_id)
            break
def is_channel_busy():
    # Aquí iría la lógica para determinar si el canal está realmente ocupado
    return random.choice([True, False])
def exponential_backoff(attempt):
    return random.randint(0, 2**attempt - 1)
def send_data(node_id):
    print(f"Node {node_id}: Datos enviados exitosamente")
# Ejemplo de uso
simulate_csma_ca(node_id=1)

```

Node 1: Canal ocupado, aplicando backoff

Node 1: Canal libre, transmitiendo datos

Node 1: Datos enviados exitosamente

✓ **Parte 2:** Mitigación de Interferencia Co-canal

```
def dsss_encode(data, chip_code):
    encoded = []
    for bit in data:
        encoded_bit = [chip * int(bit) for chip in chip_code]
        encoded.extend(encoded_bit)
    return encoded

def dsss_decode(encoded_data, chip_code):
    decoded = []
    index = 0
    while index < len(encoded_data):
        segment = encoded_data[index:index+len(chip_code)]
        decoded_bit = 1 if sum(segment) > len(chip_code)/2 else 0
        decoded.append(decoded_bit)
        index += len(chip_code)
    return decoded

# Ejemplo de uso
data = [1, 0, 1]
chip_code = [1, -1, 1, -1, 1, -1] # Ejemplo de un código chip
encoded_data = dsss_encode(data, chip_code)
decoded_data = dsss_decode(encoded_data, chip_code)
print("Encoded Data:", encoded_data)
print("Decoded Data:", decoded_data)

Encoded Data: [1, -1, 1, -1, 1, -1, 0, 0, 0, 0, 0, 0, 1, -1, 1, -1, 1, -1]
Decoded Data: [0, 0, 0]
```

✓ **Parte 3:** Mejora del Período Libre de Contención

```
def adjust_contention_window(node_id, success_rate):
    if success_rate < 0.5:
        increase_backoff(node_id)
    else:
        decrease_backoff(node_id)

def increase_backoff(node_id):
    print(f"Node {node_id}: Incrementando el tiempo de backoff debido a baja tasa de éxito")

def decrease_backoff(node_id):
    print(f"Node {node_id}: Disminuyendo el tiempo de backoff debido a alta tasa de éxito")

# Simulación de la función para calcular y ajustar la ventana de contención
def contention_window_adjustment(node_id):
    # Simulando una tasa de éxito basada en transmisiones anteriores
    # Esta podría calcularse a partir de datos reales de transmisiones exitosas vs intentos
    success_rate = calculate_success_rate(node_id)
    adjust_contention_window(node_id, success_rate)

def calculate_success_rate(node_id):
    # Supongamos que esta función calcula la tasa de éxito de las transmisiones
    # basada en alguna lógica de seguimiento de éxito/fallo
    # Aquí devolvemos un valor aleatorio para la demostración
    return random.random()

# Ejemplo de uso del ajuste de la ventana de contención
node_id = 1
contention_window_adjustment(node_id)
```

Node 1: Disminuyendo el tiempo de backoff debido a alta tasa de éxito

✓ **Problema 4:** Análisis y Resolución de Problemas de conectividad en una red compleja

✓ **Parte 1:** Diagnóstico de problemas de Ethernet

```
import random

def simulate_ethernet_traffic():
    # Simular el tráfico de Ethernet y detectar problemas potenciales
    traffic_patterns = ['normal', 'crosstalk', 'jam']
    for _ in range(10): # Simular 10 ciclos de tráfico
        traffic_type = random.choice(traffic_patterns)
        if traffic_type == 'crosstalk':
            print("Crosstalk detected! Adjusting cable configurations.")
        elif traffic_type == 'jam':
            print("Jam signal detected! Resetting Ethernet interfaces.")
        else:
            print("Normal traffic.")
    simulate_ethernet_traffic()
```

Normal traffic.
Crosstalk detected! Adjusting cable configurations.

Normal traffic.
Crosstalk detected! Adjusting cable configurations.
Jam signal detected! Resetting Ethernet interfaces.
Jam signal detected! Resetting Ethernet interfaces.
Normal traffic.
Normal traffic.
Crosstalk detected! Adjusting cable configurations.
Normal traffic.

✓ **Parte 2:** Optimización de WLAN

```
def simulate_roaming(user, access_points):
    current_ap = None
    print(f"{user} starts connection attempt...")
    for ap in access_points:
        if random.random() > 0.5: # Simula la probabilidad de conectarse a un punto de acceso
            current_ap = ap
            print(f"{user} connected to {ap}")
            break
    if not current_ap:
        print(f"{user} could not connect to any access point.")
simulate_roaming('User1', ['AP1', 'AP2', 'AP3'])
```

User1 starts connection attempt...
User1 could not connect to any access point.
User1 connected to AP2

✓ **Parte 3:** Configuración y optimización de VPN

```
def configure_vpn_settings(vpn_connection):
    print("Configuring VPN...")
    vpn_connection['latency_reduction'] = True
    vpn_connection['bandwidth_optimization'] = True
    return vpn_connection
# Ejemplo de uso
vpn_settings = configure_vpn_settings({})
print("VPN Settings Adjusted:", vpn_settings)
```

Configuring VPN...
VPN Settings Adjusted: {'latency_reduction': True, 'bandwidth_optimization': True}

a