UNIVERSIDAD PERUANA
CAYETANO HEREDIA

Actividad 12 : Construyendo una área de red local

Objetivos

- Comprender los principios básicos de las redes de computadoras, incluyendo la topología de red, la conectividad y la transmisión de datos.
- Familiarizarse con los diferentes tipos de redes, como las redes inalámbricas, las redes celulares y las redes cableadas.
3. Entender los conceptos clave de las redes inalámbricas, incluyendo la ad hoc wireless network, BSSID, ESSID, roaming, y WLAN.
4. Describir las tecnologías de modulación utilizadas en las comunicaciones inalámbricas, como MIMO-OFDM y FHSS.
5. Familiarizarse con los protocolos de acceso al medio, como CSMA/CA y CSMA/CD, utilizados en redes Ethernet y Wi-Fi.
6. Conocer los estándares IEEE relacionados con redes de computadoras, como 802.11 para Wi-Fi y 802.3 para Ethernet.
7. Comprender las amenazas de seguridad comunes en las redes, como el crosstalk, el jam signal y el flooding.
8. Conocer los mecanismos de seguridad utilizados en redes inalámbricas, incluyendo WEP, WPA y WPA2.
9. Entender los conceptos relacionados con la infraestructura de red, como los repetidores, hubs, switches y routers.
10. Describir las diferentes tecnologías de cableado, como UTP, ThickNet y Thinnet, y sus aplicaciones en las redes.
11. Explorar las tecnologías emergentes en redes de computadoras, como Power over Ethernet (PoE) y Metro Ethernet.
12. Conocer las tendencias actuales en redes inalámbricas, como las redes mesh y las redes 5G.

Aspectos básicos de la actividad

Sean los siguientes conceptos dados en clase:

5-4-3 rule Ad hoc wireless network Association Autonegotiation Band Basic service set BSSID Cellular network Chips Co-channel interference Contention-free period Coverage area Crossover cable Cross-talk CSMA/CA CSMA/CD Distribution system DSSS Dumb terminal ESSID Ethernet repeater Extended service set EUI-48 EUI-64 Fast Ethernet FHSS Flooding Full-duplex mode Gigabit Ethernet Half-duplex mode Hotspot Independent basic service set IEEE standards Initialization vector Jam signal LAN edge MAC table Metro Ethernet MICHAEL MIMO-OFDM Modulation OFDM Power over Ethernet Probe Pseudonoise RJ45 jack Roaming SIM card Simplex mode SSID T-connector Terminator ThickNet Thinnet TKIP Trunk UTP VLAN VPN WAP WEP Wireless ad hoc network WLAN WLAN frame WNIC WPA WPA2 XOR [1].

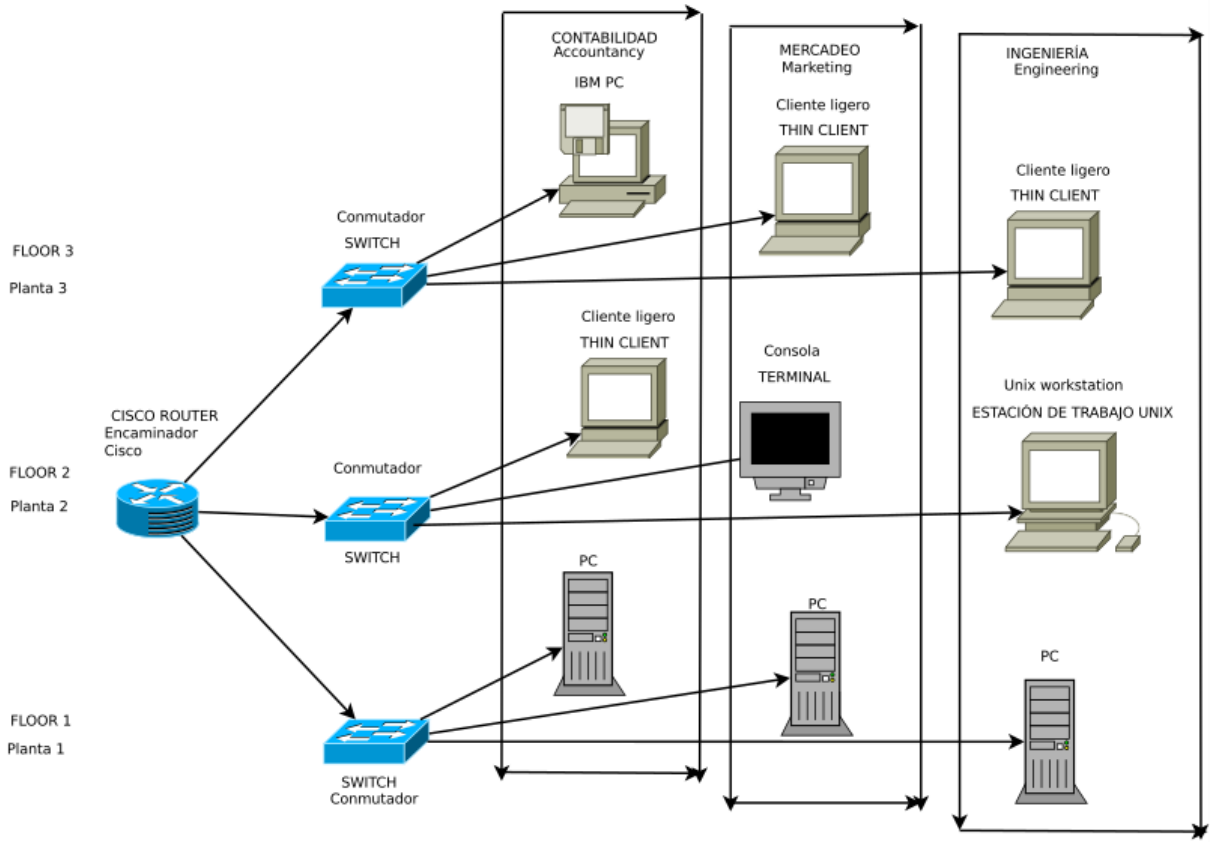
[1] Puedes utilizar el internet o tus notas para verificar los conceptos dados

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

Contexto: Una empresa necesita diseñar una red Metro Ethernet para conectar varias sucursales en una área metropolitana. Se requiere calidad de servicio (QoS) para priorizar el tráfico de voz sobre IP (VoIP) sobre el tráfico regular de datos.

Requisitos:

- Explique cómo utilizaría VLAN y Trunk para segmentar y gestionar el tráfico en la red.
- Diseñe una política de QoS que utilice Ethernet y VLAN tags para garantizar la prioridad del tráfico de VoIP.
- Simule el entorno utilizando software de simulación de redes y describa cómo los conceptos de Fast Ethernet, Gigabit Ethernet, y Full-duplex mode se aplican en este diseño.



Para tu presentación y código a presentar puedes utilizar:

Objetivos:

1. Diseño de Red: Utilizar VLANs y troncales para segmentar el tráfico de VoIP y datos.
2. Configuración de QoS: Establecer políticas de QoS para priorizar el tráfico VoIP.
3. Simulación con Python: Automatizar la configuración y simulación de la red utilizando Python.

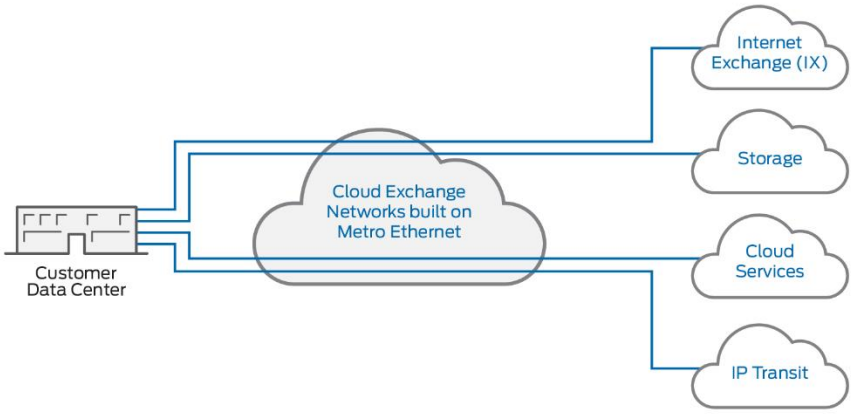
Parte 1: Diseño de red utilizando VLANs y troncales

Requisitos:

- Definir diferentes VLANs para VoIP y datos en cada sucursal.
- Configurar troncales para permitir el tráfico de ambas VLANs entre los switches y routers de la red Metro Ethernet.

Ejemplo de configuración de VLAN y troncales en Python:

Supongamos que utilizaremos una biblioteca ficticia llamada netmiko para simular la conexión y configuración de dispositivos de red. Aquí, configuraremos un switch con dos VLANs, una para datos y otra para VoIP.



```
 from netmiko import ConnectHandler


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
with ConnectHandler(**switch) as conn:
    output = conn.send_config_set(commands)
    print(output)
    conn.disconnect()

# Ejemplo de uso
configure_switch('192.168.1.100', 'admin', 'password')
```

Parte 2: Configuración de QoS para priorizar VoIP

Requisitos:

- Utilizar políticas de QoS para asegurar que el tráfico de VoIP tenga la máxima prioridad.

Ejemplo de configuración de QoS en Python:

Continuamos utilizando la biblioteca netmiko para aplicar políticas de QoS que prioricen VoIP sobre el tráfico de datos normales.

```
from netmiko import ConnectHandler

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',
    ]

    with ConnectHandler(**switch) as conn:
        output = conn.send_config_set(qos_commands)
        print(output)
        conn.disconnect()

# Ejemplo de uso
configure_qos('192.168.1.100', 'admin', 'password')
```

Parte 3: Simulación y análisis

Utilizar herramientas de simulación o scripts adicionales en Python para simular el tráfico y medir la efectividad de las políticas de QoS. Aquí, se podría usar scapy para generar tráfico de VoIP y de datos, y observar la priorización basada en los DSCP tags asignados a los paquetes.

```
from scapy.all import IP, UDP, Ether, sendp

import time

def send_voip_traffic(destination_ip, source_mac):

    packet = Ether(src=source_mac) / IP(dst=destination_ip, tos=0x2e) / UDP(dport=5004)

    sendp(packet, verbose=False)

    print("Paquete de VoIP enviado")

def send_data_traffic(destination_ip, source_mac):

    packet = Ether(src=source_mac) / IP(dst=destination_ip, tos=0x00) / UDP(dport=5004)

    sendp(packet, verbose=False)

    print("Paquete de datos enviado")

voip_destination_ip = "192.168.1.10"

data_destination_ip = "192.168.1.20"


source_mac = "00:00:00:00:00:01"


for i in range(10):

    send_voip_traffic(voip_destination_ip, source_mac)

    time.sleep(1)

    send_data_traffic(data_destination_ip, source_mac)

    time.sleep(1)
```

Problema 2: Implementación de seguridad en WLAN con tecnología MIMO-OFDM

Contexto: Una universidad desea actualizar su red WLAN para mejorar la cobertura, el rendimiento y la seguridad. Se decide implementar tecnología MIMO-OFDM y actualizar los protocolos de seguridad a WPA 3.

Requisitos:

- Describa el proceso de configuración de una WLAN utilizando MIMO-OFDM para maximizar la cobertura y el rendimiento.
- Explique cómo WPA 3 mejora la seguridad en comparación con WPA y WPA2, particularmente en relación con la gestión de claves y la protección contra ataques comunes.
- Proponga un esquema para el uso efectivo de ESSID y BSSID en el contexto de un servicio distribuido y un servicio extendido en la red.

Para tu presentación y código a presentar puedes utilizar:

Objetivos:

1. Configuración de MIMO-OFDM: Establecer configuraciones que optimicen el uso de MIMO-OFDM para mejorar la capacidad de la red y la cobertura.
2. Seguridad con WPA3: Implementar y configurar WPA 3 para mejorar la seguridad en la red WLAN.
3. Simulación con Python: Usar Python para simular la configuración de la red y testear la seguridad.

Parte 1: Configuración de MIMO-OFDM

MIMO-OFDM (Multiple Input Multiple Output - Orthogonal Frequency Division Multiplexing) es una tecnología clave en redes WLAN modernas que permite transmisiones de alta velocidad y es especialmente útil en entornos con alto nivel de reflexiones y obstáculos.

Simulación de MIMO-OFDM:

Para simular MIMO-OFDM en Python, podríamos usar bibliotecas como NumPy para manipular señales y simular el procesamiento de múltiples antenas. Aqui tienes un ejemplo de un esquema básico:

```
import numpy as np

def simulate_ofdm_signal(num_subcarriers):

    # Crear señales aleatorias para simular datos enviados en cada subportadora

    data = np.random.randint(0, 2, num_subcarriers)

    # OFDM implica la transformada de Fourier inversa para pasar de frecuencia a tiempo
```


```
    ofdm_signal = np.fft.ifft(data)

    return ofdm_signal def


mimo_transmission(signal, num_antennas): #


Simular la transmisión desde múltiples antenas


    transmitted_signal = np.tile(signal, (num_antennas, 1))

    # Agregar diversidad de antenas, ejemplo simple

    for i in range(num_antennas):

        transmitted_signal[i, :] *= np.exp(1j * np.random.rand()) # Fase aleatoria por antena

    return transmitted_signal

# Ejemplo de uso

num_subcarriers = 64 # Número de subportadoras en OFDM

num_antennas = 4 # Número de antenas en MIMO

ofdm_signal = simulate_ofdm_signal(num_subcarriers)

mimo_signal = mimo_transmission(ofdm_signal, num_antennas)
```

Parte 2: Configuración y Simulación de WPA3

WPA3 es el último estándar de seguridad para redes WLAN que ofrece mejoras significativas sobre sus predecesores, incluyendo cifrado más fuerte y protección contra ataques comunes como KRACK.

Implementación de WPA3:

Configurar WPA3 requiere soporte de hardware y software, pero podemos simular el proceso de configuración utilizando scripts que podrían ser ejecutados en puntos de acceso compatibles con WPA3.

```
def configure_wpa3(device_config):
    # Simular la configuración de WPA3 en un
    dispositivo print("Configurando WPA3 en el
    dispositivo") device_config['security'] = 'WPA3'
    device_config['ssid'] = 'Universidad-RedSegura'
    device_config['password'] =
    'contraseña_segura_123' return device_config

# Ejemplo de uso device_config = {}
updated_config = configure_wpa3(device_config)
print(updated_config)
```

Parte 3: Evaluación de la seguridad

Finalmente, para evaluar la seguridad de la implementación de WPA3, podríamos utilizar herramientas como Scapy para simular intentos de ataques y verificar la robustez del cifrado y autenticación en la red.

```
from scapy.all import *



# Función para simular un intento de ataque de fuerza bruta contra la red WPA3

def simulate_brute_force_attack():

    # lista de posibles contraseñas para probar

    passwords = ['password123', 'securepass', 'wpa3rocks', '12345678']

    for password in passwords:

 #paquete de autenticación

        pkt = RadioTap() / Dot11(addr1='ff:ff:ff:ff:ff:ff', addr2='11:22:33:44:55:66',
addr3='11:22:33:44:55:66') / Dot11Auth(algo=0, seqnum=1, status=0)

        sendp(pkt, iface='wlan0')  # Suponiendo que wlan0 es la interfaz de red adecuada
```

Ejemplo de uso

simulate_brute_force_attack()

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

Contexto: En un entorno de red ad hoc inalámbrico, los dispositivos sufren de interferencia co-canal y de problemas de acceso al medio debido a la naturaleza descentralizada de la red.

Requisitos:

- Explique cómo los protocolos CSMA/CA y CSMA/CD difieren y por qué uno es más adecuado que el otro para redes ad hoc inalámbricas.
- Discuta el impacto de la interferencia co-canal y proponga métodos de modulación como FHSS o DSSS para mitigar este problema.

- Desarrolle un algoritmo o protocolo para mejorar el período libre de contención en este tipo de redes, incluyendo el uso de vectores de inicialización para la seguridad de las transmisiones.

Para tu presentación y código a presentar puedes utilizar:

Objetivos:

1. Implementar CSMA/CA para manejo de acceso al medio.
2. Mitigar la interferencia co-canal utilizando técnicas de modulación como FHSS o DSSS.
3. Mejorar el período libre de contención utilizando un algoritmo personalizado.
4. Simulación con Python de las estrategias propuestas.

Parte 1: Implementación de CSMA/CA en Python

CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) es un protocolo utilizado para mejorar el uso del medio en redes inalámbricas, ayudando a reducir las colisiones mediante la "escucha" antes de transmitir y empleando un mecanismo de backoff cuando el canal está ocupado. Simulación de CSMA/CA:

```
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)
```

Parte 2: Mitigación de Interferencia Co-canal

La interferencia co-canal ocurre cuando múltiples nodos transmiten en la misma frecuencia, causando degradación de la señal. Las técnicas como FHSS (Frequency-Hopping Spread Spectrum) o DSSS (Direct Sequence Spread Spectrum) pueden ser simuladas para demostrar cómo podrían mitigar este problema.

Simulación básica de DSSS:

```
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)
```

Parte 3: Mejora del Período Libre de Contención

Para mejorar el período libre de contención, podríamos desarrollar un algoritmo que ajuste dinámicamente los tiempos de espera basándose en el tráfico reciente o en la detección de patrones de uso del canal. Algoritmo de ajuste dinámico:

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

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

Contexto: Una red empresarial compuesta por diferentes segmentos de red, incluyendo Ethernet, WLAN y VPN, experimenta problemas de conectividad y rendimiento intermitentes.

Requisitos:

- Utilice técnicas de diagnóstico para identificar y resolver problemas de crosstalk y jam signal en la red Ethernet.
- Explique cómo el roaming y la utilización de SIM card pueden afectar la experiencia de los usuarios en la WLAN y proponga soluciones para optimizar la transición entre puntos de acceso.
- Evalúe cómo diferentes configuraciones de VPN podrían influir en la latencia y el ancho de banda de la red, y proponga ajustes para mejorar la conectividad y seguridad.

Para tu presentación y código a presentar puedes utilizar:

Objetivos:

1. Diagnóstico de Problemas de Ethernet: Identificar y resolver problemas de crosstalk y señales de jam en la red Ethernet.
2. Optimización de WLAN: Examinar el impacto del roaming y la utilización de tarjetas SIM en la experiencia del usuario y la conectividad.
3. Configuración y Optimización de VPN: Evaluar y ajustar la configuración de VPN para mejorar la latencia y el ancho de banda.
4. Simulación con Python de las estrategias propuestas.

Parte 1: Diagnóstico de problemas de Ethernet

Para diagnosticar y resolver problemas de crosstalk y jam signals en la red Ethernet, utilizaremos Python para simular la red y para detectar y resolver conflictos.

Código Python para simular y diagnosticar 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()
```

Parte 2: Optimización de WLAN

El impacto del roaming y la utilización de tarjetas SIM puede ser crucial en entornos WLAN. Aquí podemos usar Python para simular cómo los cambios en la infraestructura de WLAN afectan la conectividad del usuario.

Código Python para Simular roaming en 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'])
```

Parte 3: Configuración y optimización de VPN

Configurar y optimizar una VPN para mejorar la latencia y el ancho de banda puede involucrar ajustes en la configuración del protocolo y la infraestructura de red.

Código Python para simular y optimizar 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)
```

Presentación:

Parte 1: Prepara una presentación para resolver los anteriores problemas:

Estructura de la exposición:

1. Introducción:

- Saludo y presentación del presentador/es.
- Breve descripción del objetivo de la exposición y los temas que se cubrirán.
- Importancia de entender los conceptos de redes de computadoras en el contexto actual de la tecnología de la información.

2. Desarrollo de los temas:

- Fundamentos de Redes: Explicación de los conceptos básicos como ARPANET, backbone, Bluetooth, broadcast, cache memory, checksum, etc.
- Seguridad y fiabilidad de redes: Discusión sobre temas como encriptación, firewalls, TCP/IP, etc.
- Infraestructura de redes: Descripción de los componentes de una red, topologías, direccionamiento MAC, dispositivos de red, etc.
- Protocolos y comunicación: Explicación de protocolos como Ethernet, TCP/IP, OSI, así como los diferentes tipos de redes y protocolos de enrutamiento.

3. Estudios de caso:

- Análisis de estudios de caso reales relacionados con problemas de redes de computadoras y cómo se resolvieron.
- Discusión sobre lecciones aprendidas y mejores prácticas identificadas en cada caso.

4. Preguntas y respuestas:

- Sesión interactiva donde los asistentes pueden hacer preguntas sobre los temas tratados.
- Responder preguntas y proporcionar aclaraciones adicionales sobre los conceptos presentados.

6. Conclusión:

- Resumen de los puntos clave cubiertos durante la exposición.
- Reflexión sobre la importancia de comprender los conceptos de redes de computadoras en la era digital.
- Agradecimiento a los asistentes y cierre de la exposición.

Consejos para una exposición exitosa:

1. Conocimiento profundo: Asegúrate de comprender completamente los temas que vas a presentar y estar preparado para responder preguntas.
2. Claridad y concisión: Explica los conceptos de manera clara y sencilla, evitando tecnicismos innecesarios.

3. Ejemplos prácticos: Utiliza ejemplos prácticos y casos de uso para ilustrar los conceptos teóricos.
4. Interacción con la audiencia: Fomenta la participación de la audiencia haciendo preguntas o invitándolos a compartir sus experiencias.
5. Material visual atractivo: Utiliza diapositivas, gráficos o demostraciones visuales para mantener el interés de la audiencia.

Parte 2: Entrega de la actividad completa

Elementos a entregar:

1. Código fuente de la implementación del protocolo de red en un repositorio público en GitHub.
2. Presentación detallada en formato PDF, que incluya los objetivos, la metodología, los resultados y las conclusiones de la actividad y la presentación corregida.
3. En la presentación y en el repositorio, indicar claramente que se ha trabajado en equipo en el desarrollo del proyecto.

Fecha de entrega: 28 de abril hasta las 23:59

La actividad completa debe ser entregada en los repositorios individuales de los estudiantes en la fecha indicada.