UNIVERSIDAD PERUANA CAYETANO HEREDIA

Actividad 11 : Conceptos de introducción a las redes

Objetivos

Fundamentos de redes:

1. Comprender los conceptos básicos de ARPANET y su importancia histórica en el desarrollo de Internet.
2. Identificar el papel de un backbone en una red de computadoras y su función en el enrutamiento de datos.
3. Describir las características principales de la tecnología Bluetooth y su aplicación en redes inalámbricas de corto alcance.
4. Explicar el concepto de broadcast en redes de computadoras y su uso para la difusión de datos a múltiples destinatarios.
5. Entender el funcionamiento y la importancia de la memoria caché en la optimización del rendimiento de las redes.
6. Explicar el propósito y el proceso de la comprobación de integridad mediante checksum en la transferencia de datos.
7. Comprender la diferencia entre un cliente y un servidor en un entorno de red cliente-servidor y cómo interactúan entre sí.
8. Identificar los principios básicos de encriptación y descifrado, así como la importancia de la gestión de claves de cifrado.
9. Describir cómo se implementa la seguridad de red mediante firewalls y el papel que desempeñan en la protección de la red contra amenazas externas.
10. Analizar el protocolo TCP/IP y su papel en la comunicación de datos en Internet, incluida la diferencia entre TCP y UDP.
11. Identificar los componentes básicos de una red Ethernet, como los hubs, switches y routers, y entender su función en la conectividad de red.
12. Describir la topología de una red de área local (LAN) y compararla con una red de área amplia (WAN) en términos de alcance y tecnologías utilizadas.
13. Explicar el concepto de direccionamiento MAC y su importancia en la comunicación de datos a nivel de enlace de datos.
14. Comprender el papel de los dispositivos de red como los repetidores y los routers en la interconexión de redes y el enrutamiento de datos.
15. Analizar el modelo OSI y su estructura de capas para entender cómo se lleva a cabo la comunicación de datos en una red.
16. Explorar el concepto de conmutación de paquetes y su importancia en la transferencia eficiente de datos en redes de computadoras.
17. Identificar los diferentes tipos de redes, como las redes peer-to-peer y las redes cliente-servidor, y comprender sus características y aplicaciones.
18. Entender los principios básicos de la comunicación unicast, multicast y broadcast en una red y cómo se utilizan en diferentes escenarios.
19. Describir la función de los protocolos de enrutamiento como el protocolo de enrutamiento por vectores de distancia (RIPv2) y el protocolo de enrutamiento por estado de enlace (OSPF) en la determinación de rutas de red.

20. Analizar la importancia de los estándares de comunicación como los RFC en la interoperabilidad y el desarrollo de tecnologías de red.

Aspectos básicos de la actividad

Sean los siguientes conceptos dados en clase:

ARPANET Backbone Bluetooth Broadcast Cache Memory Checksum Client Client–server network Computer network CRC DDN Encryption and decryption Encryption key Ethernet Firewall Frame Relay Gateway Hub Internet LAN MAC address MODEM Multicast Network cache Network device NIC Nonce value OSI model Packet switching Peer-to-peer network. Private key Protocol Protocol stack public key Repeater RFC Router Server Switch TCP/IP Topology Unicast WAN, [1]

Problema 1: Diseño de red segura

Escenario: Una empresa necesita diseñar una red segura que conecte tres sucursales ubicadas en diferentes ciudades utilizando tecnología WAN y LAN. La empresa maneja datos confidenciales y requiere que la comunicación entre sucursales sea cifrada.

Preguntas:

1. ¿Qué tipo de tecnología de WAN utilizarías para conectar las sucursales y por qué? (Considera opciones como Frame Relay, MPLS, etc.)

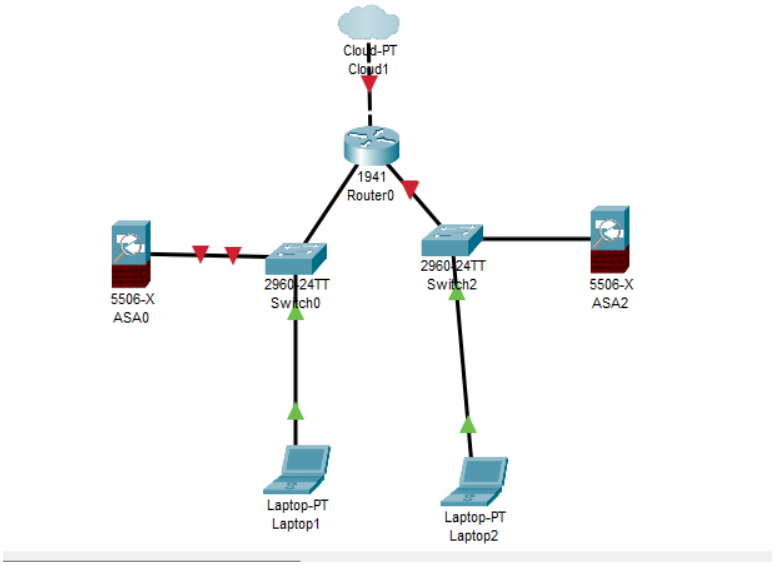
   Para poder conectar las sucursales de una empresa podemos utilizar la tecnología MPLS ya que es muy eficiente en el enrutamiento, calidad de servicio, flexibilidad y es compatible con VPNs.

2. Describe cómo implementarías el cifrado en la red. ¿Qué tipos de claves y protocolos utilizarías?

   Al momento de manejar datos confidenciales es necesario utilizar un cifrado fuerte, las claves que podemos utilizar serían las siguientes: claves simétricas y asimétricas, para el uso de los protocolos podemos utilizar IPsec, así mismo el protocolo SSL/TLS para el cifrado y la capa de transporte.

3. Dibuja una topología de red que incluya dispositivos como routers, switches, y firewalls

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



4. Explica la función de cada dispositivo en tu diseño. Puedes utilizar Packet Tracer

   Lo que se va a realizar en es simulación es que lo Routers van a enrutar el tráfico entre las sucursales y los Switches para conectar los dispositivos dentro de cada sucursal.

5. ¿Cómo garantizarías la integridad y autenticidad de los datos transmitidos entre las sucursales? Detalla el uso de checksums o CRC.

Para tu presentación y código a presentar puedes utilizar:

Requisitos:

1. Conexión WAN Segura: Utilizar VPNs para asegurar todas las comunicaciones entre sucursales.
2. Cifrado: Implementar cifrado de extremo a extremo.
3. Topología de Red: Incluir dispositivos de red como routers, switches y firewalls.
4. Automatización: Usar Python para configurar aspectos de la red automáticamente.

Parte 1: Diseño de topología de red

Preguntas:

1. Dibuja una topología de red para este escenario que incluya los dispositivos de red necesarios en cada sucursal.
2. Explica cómo cada dispositivo contribuye a la seguridad y eficiencia de la red.

Parte 2: Configuración de VPN con Python

Utilizando la biblioteca paramiko de Python, escribe un script que configure una VPN en los routers de cada sucursal. Supondremos que los routers son dispositivos Cisco y que el script debe configurar automáticamente la VPN utilizando IPsec.

Código Python: import paramiko def

connect_to_router(hostname, username, password):

client = paramiko.SSHClient()

client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

client.connect(hostname, username=username, password=password)

return client def configure_ipsec_vpn(client, peer_ip, local_network,

remote_network):

commands = [

'crypto isakmp policy 10',

'encr aes 256',

'authentication pre-share',

'group 5',
'crypto isakmp key mysharedsecret address ' + peer_ip,

'crypto ipsec transform-set myset esp-aes 256 esp-sha-hmac',

'crypto map mymap 10 ipsec-isakmp',

'set peer ' + peer_ip,

'set transform-set myset',

'match address 100',

'access-list 100 permit ip ' + local_network + ' ' + remote_network,

'interface g0/0',

'crypto map mymap',

'end'

]

for command in commands:

```
    stdin, stdout, stderr =

  client.exec_command(command)

  print(stdout.read().decode()) client.close()

# Ejemplos de uso hostname = '192.168.1.1' username =

'admin' password = 'password' client =

connect_to_router(hostname, username, password)

configure_ipsec_vpn(client, '192.168.2.1', '192.168.1.0 255.255.255.0', '192.168.3.0
255.255.255.0')
```

Preguntas adicionales:

1. ¿Cómo implementarías el cifrado de extremo a extremo además de la VPN? Considera el uso de claves públicas y privadas.

   Se podría utilizar el protocolo TLS para hacer un intercambio seguro de calves públicas y privadas.

2. Proporciona un esquema para implementar un sistema robusto de logs y monitoreo de la red utilizando herramientas modernas de gestión de red. ¿Cómo podría Python automatizar la recopilación y análisis de logs?

Para tener un sistema Robusto podemos utilizar los syslog para la centralización, Elk para el almacenamiento y por último Prometheus y Grafana para el monitoreo del rendimiento de la red. Python se puede automatizar para recopilar, analizar y generar alertas.

Problema 2: Optimización de protocolos y caché

Escenario: Una empresa de streaming de video experimenta interrupciones frecuentes en la entrega de contenido a los clientes. La infraestructura actual utiliza multicast para distribuir contenido, pero aún enfrenta problemas de latencia y pérdida de paquetes.

Preguntas:

1. Explica cómo mejorarías el rendimiento utilizando técnicas de caché de red. ¿Dónde colocarías estos cachés y por qué?

Para mejorar el rendimiento en la entrega del video y reducir la latencia, podemos utilizar técnicas de cache de red, así como, Cache de contenido cache DNS, cache de aplicaciones lo cual disminuye la carga a los servidores centrales.

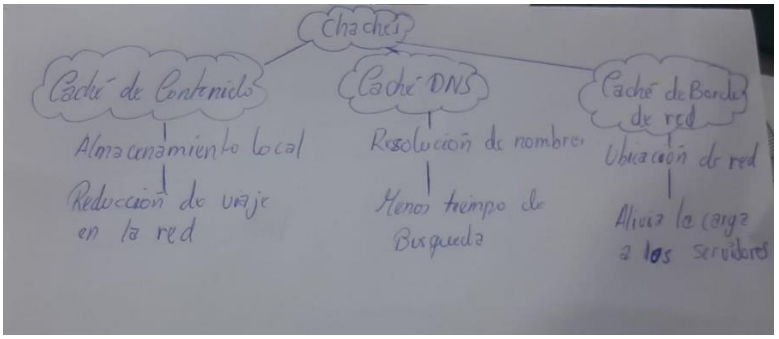
2. ¿Cómo afecta el protocolo de transporte al rendimiento del streaming de video? Considera TCP vs UDP y justifica tu elección.

   Puede tener un impacto considerable en el rendimiento, para este caso podemos utilizar UDP porque tiene baja latencia y tiene velocidad para la transmisión en vivo.

3. Propone una solución usando Anycast para optimizar la entrega de contenido. ¿Cómo funcionaría en este contexto?

El Anycast puede acelerar la entrega de contenido web, así mejorando la velocidad y la fiabilidad.

4. Desarrolla un modelo simplificado para calcular el efecto de la caché en la reducción de latencia.



Para tu presentación y código a presentar puedes utilizar:

Requisitos:

1. Mejora del rendimiento utilizando técnicas de caché de red.
2. Elección y configuración del protocolo de transporte adecuado para reducir la latencia y mejorar la confiabilidad.
3. Implementación de Anycast para optimizar la entrega de contenido a los usuarios.
4. Simulación y automatización utilizando Python.

Parte 1: Implementación de caché de red con Python

Usaremos Python para simular un sencillo sistema de caché que pueda almacenar y recuperar videos solicitados frecuentemente para reducir la latencia y la carga en el servidor principal.

Código Python:

```
class VideoCache:
    def __init__(self):
        self.cache = {}

    def get_video(self, video_id):
        if video_id in self.cache: print(f"Video {video_id}
            retrieved from cache") return
            self.cache[video_id]
        else:
            print(f"Video {video_id} not in cache, downloading...")
            video_data = self.download_video(video_id)
            self.cache[video_id] = video_data return video_data

    def download_video(self, video_id):
        # Simula la descarga del video desde un servidor remoto
        return f"Video data for {video_id}"

# Ejemplo de uso cache =
VideoCache() video =
cache.get_video("video1234")
print(video) # Primera vez descarga, segunda vez desde caché
video = cache.get_video("video1234")
```

Parte 2: Selección del protocolo de transporte

Exploraremos cómo usar UDP en lugar de TCP para mejorar la eficiencia de la transmisión de video, debido a la menor sobrecarga de UDP.

Discusión:

- Explica las ventajas de usar UDP sobre TCP para streaming de video, considerando las características de ambos protocolos.
- Analiza los posibles problemas de confiabilidad y orden de llegada de los paquetes y cómo mitigarlos.

Ventajas de la UDP sobre TPC

- Menor sobrecarga
- Menor latencia
- Transmisión continua

Problemas de confiabilidad y orden de llegada

Perdida de paquete: Dado que UDP suele perder algunos paquetes lo que resulta una calidad disminuida

Sin embargo, al UDP es considera para el streaming debido a su menor latencia. Mientras que la TCP es mas de presentar fiabilidad y el orden de los datos.

Parte 3: Implementación de anycast con Python

Simularemos el uso de anycast para dirigir las solicitudes de los usuarios al servidor de caché más cercano utilizando Python. Este ejemplo es conceptual, ya que la implementación real de anycast se manejaría a un nivel más bajo en la red.

```
import random

class AnycastService:
    def __init__(self): self.servers = ['192.168.1.1',
        '192.168.2.1', '192.168.3.1']

    def get_nearest_server(self, user_ip):
        # Simula la selección del servidor más cercano (simplificado)
        return random.choice(self.servers)

# Ejemplo de uso anycast
= AnycastService()
nearest_server = anycast.get_nearest_server("192.168.1.100")
print(f"Nearest server for user is {nearest_server}")
```

Parte 4: Monitorización y análisis

Proponer un sistema de monitoreo y análisis para evaluar el rendimiento de la red y la eficacia de las estrategias implementadas, usando herramientas como Wireshark para analizar el tráfico de red.

- Usa wireshark para capturar paquetes de red y analiza el tráfico específico de video para identificar patrones de uso y posibles cuellos de botella.

Podemos utilizar la siguiente configuración

- Captura de paquetes de red
- Filtrado de trafico de video
- Análisis de patrones de uso
- Identificar cuellos de botella
- Integración de sistema de monitoreo
- Automatización del análisis

Problema 3: Simulación de ataque y respuesta

Escenario: Eres el administrador de seguridad de una red corporativa y has notado un aumento en el tráfico ARP anómalo. Sospechas que esto podría ser parte de un ataque de ARP spoofing, donde un atacante podría estar intentando interceptar la comunicación entre dos partes para robar o modificar datos transmitidos.

Preguntas:

1. Describe cómo identificarías si estas transmisiones ARP son realmente maliciosas.
2. ¿Qué medidas tomarías para mitigar el ataque una vez confirmado?
3. Explica cómo un switch y un firewall pueden configurarse para prevenir futuros ataques de este tipo.
4. Formula un plan para educar a los empleados sobre medidas de seguridad que pueden tomar para reducir el riesgo de ataques futuros.

Para tu presentación y código a presentar puedes utilizar:

Requisitos:

1. Detección de ARP Spoofing: Implementar un sistema de detección para identificar posibles ataques.

   Podeos utilizar herramientas de monitoreo de red como Wireshark para analizar el trafico ARP. También se puede buscar en la tabla de ARP las múltiples direcciones IP.

2. Mitigación del Ataque: Desarrollar un método para mitigar o detener el ataque una vez detectado.

   Podemos tomar alagunas medidas como poner la localización y aislar el dispositivo Atacante.

3. Automatización y Monitorización: Usar Python para automatizar la detección y respuesta, y monitorear continuamente la red para futuros ataques.

   Los swiches se puede configurar para la inspección del ARP, así mismo los firewalls se puede configurar para bloquear el tráfico ARP sospechoso.

4. Educación de Empleados: Proporcionar un plan para educar a los empleados sobre cómo pueden ayudar a reducir el riesgo de futuros ataques.

   Tomar medidas de seguridad y tener los sistemas actualizados y la necesidad de utilizar contraseñas fuertes.

Parte 1: Detección de ARP Spoofing con Python

Utilizaremos Python y la biblioteca **Scapy,** que es muy potente para la manipulación y análisis de paquetes de red, para detectar anomalías en los mensajes ARP.

Código Python para la Detección de ARP Spoofing:

```
from scapy.all import ARP, sniff

def detect_arp_spoofing(packet):
```

```
    if packet.haslayer(ARP):

        if packet[ARP].op == 2: # is it an ARP response (ARP reply)?

            try:

                real_mac = getmacbyip(packet[ARP].psrc)
                response_mac = packet[ARP].hwsrc

                if real_mac != response_mac:

                    print(f"[ALERT] ARP Spoofing Detected: {packet[ARP].psrc} has been
spoofed!") except

            IndexError:

                # Unable to find the MAC address for the IP address in the ARP response

                pass


def sniff_network(interface): sniff(iface=interface, store=False,

    prn=detect_arp_spoofing, filter="arp")


# Ejemplo de uso

sniff_network('eth0')
```

Este script escucha en la interfaz de red especificada para paquetes ARP y compara la dirección MAC real con la dirección MAC que envía respuestas ARP. Si no coinciden, se detecta un ataque de spoofing.

Parte 2: Mitigación del ataque

Una vez detectado el ataque, se puede implementar una estrategia para mitigar el daño y prevenir futuras ocurrencias, como la reconfiguración de switches para habilitar ciertas características de seguridad.

Discusión de Estrategias de Mitigación:

- Seguridad en el Switch: Habilitar funciones de seguridad como Dynamic ARP Inspection (DAI) en los switches.

- Uso de Seguridad de Puerto: Configurar la seguridad de puerto en los switches para limitar el número de MACs por puerto y prevenir posibles ataques.

Parte 3: Automatización y monitorización

Podemos usar Python para crear scripts que configuran automáticamente los switches para habilitar estas funciones de seguridad, usando bibliotecas como Netmiko para gestionar dispositivos de red.

```
from netmiko import ConnectHandler
def enable_dai(switch_details):
    commands = [
        'ip arp inspection vlan 1-100',
        'interface range fa0/1-24',
        'ip arp inspection limit rate 100'
    ]
    with ConnectHandler(**switch_details) as switch:
        output = switch.send_config_set(commands)
        print(output)

# Ejemplos de uso
switch_details = {
    'device_type': 'cisco_ios',
    'host': '192.168.1.1',
    'username': 'admin',
    'password': 'password',
}
enable_dai(switch_details)
```

Opcional:

Enfocarse en la importancia de la seguridad en la red y cómo las acciones de los empleados pueden afectarla.

- Indicar a tus compañeros a reconocer correos electrónicos sospechosos y otras posibles fuentes de ataques.

Evitar hacer clic en los enlaces o descargas de archivos adjuntos de fuentes desconocidas, mantener el software actualizado y tener una contraseña fuerte.

Problema 4: Análisis y diseño de red Peer-to-Peer (P2P)

Escenario: Un startup tecnológico desea implementar una red P2P robusta para permitir el intercambio eficiente de recursos computacionales entre usuarios distribuidos geográficamente. Esta red debe ser capaz de manejar intercambios dinámicos de archivos, distribución de carga, y debe incorporar medidas de seguridad para prevenir accesos no autorizados.

Preguntas:

1. Describe cómo se diferenciaría esta red P2P de una red cliente-servidor en términos de diseño de red y topología.

   Los clientes dependen de un servidor central par solicitar y recibir información y el P2P actúa tanto como cliente y servidor.

2. ¿Qué protocolos específicos usarías para gestionar las comunicaciones y el intercambio de archivos en esta red?

para el intercambio de archivos podemos usar el protocolo BitTorrent para la comunicación podemos utilizar los protocolos TCP/IP y transmisión de datos.

3. Analiza los posibles problemas de seguridad asociados con una red P2P y propone soluciones para mitigar estos riesgos.

   Pueden ser vulnerables a varios tipos de ataques, como la denegación del servicio o el programa maligno. Para mitigar estos riesgos podemos implementar medidas de seguridad como la implantación de privacidad de red (VPN).

4. Evalúa el impacto de incorporar nodos que actúan tanto como clientes como servidores. ¿Cómo gestionarías el balanceo de carga?

Al momento de utilizar el P2P que actúa como cliente y servidor puede llevar a uso desigual de los recursos. Para gestionar el desequilibrio podemos implementar algoritmos que distribuyan las solicitudes de manera equilibrada.

Para tu presentación y código a presentar puedes utilizar:

Requisitos:

1. Implementación de la Red P2P: Desarrollar un modelo de red P2P utilizando Python que permita a los nodos unirse, compartir recursos y desconectarse de manera dinámica.
2. Gestión de Recursos y Distribución de Carga: Asegurar que los recursos se compartan de manera eficiente y que la carga se distribuya equitativamente entre los nodos.
3. Seguridad: Implementar mecanismos de seguridad para autenticar a los usuarios y cifrar los intercambios de archivos.
4. Automatización y Simulación: Utilizar Python para simular el comportamiento de la red y automatizar tareas comunes de gestión de la red.

Parte 1: Implementación de la Red P2P con Python

Utilizaremos Python para simular una red P2P básica donde los nodos pueden unirse, compartir archivos y salir de la red. Para esto, usaremos sockets para la comunicación entre nodos.

Código Python para la simulación de Red P2P:

```
import socket

import threading



class Peer:

    def __init__(self, host, port):
        self.host = host self.port = port self.peers = [] # List to keep track of

        peers self.server = socket.socket(socket.AF_INET,

        socket.SOCK_STREAM)

        self.server.bind((self.host, self.port)) self.server.listen(5)

        print(f"Node started on {self.host}:{self.port}")

        threading.Thread(target=self.accept_connections).start(

        )
```


```
def accept_connections(self):
    while True:
        client, address = self.server.accept() print(f"Connected with
        {address[0]}:{address[1]}") self.peers.append(client)
        threading.Thread(target=self.handle_client,
        args=(client,)).start()

def handle_client(self, client):
    while True:
        try:
            data = client.recv(1024)
            if data:
                print(f"Received: {data.decode()}")
        self.broadcast_data(data, client) except
        Exception as e:
            print(f"Error: {e}")
            client.close()
            break

def broadcast_data(self, data, sender):
    for peer in self.peers:
        if peer is not
        sender:
            peer.send(data)
```


```
    def connect_to_peer(self, host, port):

        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        client.connect((host, port))

        self.peers.append(client)

        print(f"Connected to peer at {host}:{port}")


# ejemplos de uso node =

Peer('127.0.0.1', 5000)

node.connect_to_peer('127.0.0.1',

6000)
```

Parte 2: Gestión de recursos y distribución de carga

Discusión sobre cómo optimizar la distribución de recursos en la red, utilizando algoritmos de distribución de carga y equilibrio de carga basados en la disponibilidad de recursos de cada nodo.

Estrategias de equilibrio de carga:

- Implementar un algoritmo que asigne más carga a los nodos con más recursos disponibles.
- Usar un algoritmo round-robin para asegurar que todos los nodos compartan de manera equitativa la carga de trabajo.

Parte 3: Seguridad en la Red P2P

Implementar autenticación y cifrado en la comunicación entre nodos para asegurar que solo los usuarios autorizados puedan unirse y que la transferencia de archivos sea segura.

```
import hashlib import os def generate_key(password): salt = os.urandom(16)

key = hashlib.pbkdf2_hmac('sha256', password.encode('utf-8'), salt, 100000)

return salt + key def verify_key(stored_key, provided_password):

    salt = stored_key[:16]

     key = stored_key[16:]
```

```
    new_key =hashlib.pbkdf2_hmac('sha256', provided_password.encode('utf-8'), salt,
100000) return

new_key == key


password = "secure_password" key =

generate_key(password) print("Key generated

successfully.") print("Verification:",

verify_key(key, password))
```

Monitorización de una red P2P

```
import time def

monitor_peers(peers): while

True: for peer in peers:

        try:

            peer.send(b'ping')
            response = peer.recv(1024) if response != b'pong':

        raise Exception("Peer is not responding correctly.")

        except Exception as e:

            print(f"Peer {peer.getpeername()} failed: {e}")

peers.remove(peer) time.sleep(60) # Check every

minute # In the Peer class, handle 'ping' messages

def handle_client(self, client):

    while True:

        try:

            data = client.recv(1024)

            if data:
```

```
            if data == b'ping':
                client.send(b'pong')
            else:
                print(f"Received: {data.decode()}")
        self.broadcast_data(data, client) except
        Exception as e:
            print(f"Error: {e}")
            client.close()
            break
```

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.