ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE COMPUTADORES

Seguridad por Diseño en Redes de Infraestructuras de IT
Security by Design in IT Infraestructures Networks

Realizado por
Aarón Mesa Basaga

Tutorizado por
Guillermo Pérez Trabado

Departamento
Arquitectura de Computadores

UNIVERSIDAD DE MÁLAGA
MÁLAGA, junio de 2024

1 Indice

2 Introducción

2.1 Porque es necesario la seguridad en las redes

No disponer de la seguridad adecuada en una red puede acarrear multitud de problemas. En primer lugar, aumenta el riesgo de filtración de datos, ya que personas no autorizadas pueden acceder a información confidencial, como datos de clientes y registros financieros. Esto no sólo daña la confianza, sino que también conlleva posibles pérdidas financieras y responsabilidades legales. Además, sin las medidas de seguridad adecuadas, la confidencialidad de los datos se ve comprometida, lo que permite a personas no autorizadas interceptar y ver información sensible, lo que puede suponer una violación de la normativa sobre privacidad y la filtración de datos confidenciales.

Además, las redes inseguras son susceptibles de manipulación de datos, ya que los atacantes pueden alterar o manipular los datos mientras viajan por la red. Además, sin las defensas adecuadas, las redes son vulnerables a los ataques de denegación de servicio (DoS), en los que los atacantes inundan la red con un tráfico excesivo o peticiones maliciosas, provocando la interrupción o el cierre completo de los servicios, con el consiguiente tiempo de inactividad.

Además, las redes inseguras proporcionan un punto de entrada para programas maliciosos como virus, gusanos y ransomware. Una vez dentro de la red, el malware puede propagarse, robar datos, interrumpir las operaciones y mantener los sistemas como rehenes hasta que se pague un rescate.

Si no se aplican las medidas de seguridad adecuadas, se puede incumplir la normativa del sector y las leyes de protección de datos, lo que expone a las organizaciones a multas y sanciones legales. Los clientes pueden perder la confianza en la capacidad de la organización para proteger su información

confidencial, lo que conduce a la pérdida de negocio e ingresos.

2.2 Problema de no tener seguridad

Desde los primeros días de la computación, la ciberseguridad ha sido un desafío constante. Problemas como virus informáticos, gusanos y malware han estado presentes desde los años 70 y 80. Junto con ellos, los ataques de denegación de servicio (DDoS) han interrumpido servicios en línea. También ha habido casos de robo de información y espionaje cibernético, que se remontan a la Guerra Fría. El fraude y el robo de identidad se han vuelto comunes a medida que aumentan las transacciones en línea.

Además, los sistemas críticos, como los relacionados con la energía y la infraestructura, han demostrado ser vulnerables a intrusiones cibernéticas. Amenazas emergentes como el ransomware han agregado nuevas capas de complejidad a estos problemas. En resumen, la ciberseguridad ha sido y sigue siendo un área de preocupación constante en la era digital.

Abordar la ciberseguridad en una red presenta una serie de desafíos significativos:

- **Complejidad de la infraestructura**: Las redes modernas suelen ser complejas, con una variedad de dispositivos, sistemas y aplicaciones interconectados. Gestionar y asegurar todos estos componentes puede ser difícil, especialmente cuando se trata de mantener un panorama de amenazas en constante evolución.
- **Vulnerabilidades de software y hardware**: Los sistemas operativos, aplicaciones y dispositivos de red a menudo contienen vulnerabilidades que pueden ser explotadas por los atacantes. Mantener estos elementos actualizados y parcheados puede ser un desafío, especialmente en grandes redes empresariales.
- **Ataques avanzados y persistentes**: Los ciberataques están en constante evolución, con adversarios que utilizan técnicas cada vez más sofisticadas para infiltrarse en las redes y eludir la detección. Los ataques persistentes avanzados (APT) pueden ser difíciles de detectar y contener debido a su naturaleza sigilosa y de largo plazo.
- **Falta de conciencia y capacitación del usuario**: Los usuarios finales a menudo son el eslabón más débil en la cadena de seguridad, ya que pueden caer en trampas de phishing, descargar software malicioso o compartir información sensible inadvertidamente. La falta de conciencia y capacitación en seguridad entre los usuarios puede aumentar el riesgo de brechas de seguridad.
- **Gestión de accesos y privilegios**: Gestionar quién tiene acceso a qué recursos en una red puede ser complicado, especialmente en entornos empresariales donde hay numerosos empleados, contratistas y socios externos. La gestión inadecuada de los accesos y privilegios puede conducir a violaciones de datos y compromisos de seguridad.
- **Privacidad y cumplimiento normativo**: Las organizaciones deben cumplir con una variedad de regulaciones y estándares de privacidad de datos, lo que puede requerir controles de seguridad específicos y la implementación de medidas adicionales de protección de la privacidad. Cumplir con estos requisitos mientras se mantiene la funcionalidad y la eficiencia de la red puede ser un desafío.

2.3 Clouds Comerciales que proveen Seguridad por Diseño

2.3.1 Amazon Web Services (AWS)

AWS ofrece una amplia gama de servicios y características de seguridad, como Identity and Access Management (IAM), Virtual Private Cloud (VPC) para el aislamiento de redes, servicios de cifrado como AWS Key Management Service (KMS) y AWS Certificate Manager, y AWS Firewall Manager para la administración centralizada de grupos de seguridad.

2.3.2 Microsoft Azure

Azure hace hincapié en la seguridad con funciones como Azure Active Directory (AAD) para la gestión de identidades, Azure Security Center para la protección frente a amenazas, Azure Key Vault para la gestión segura de claves y Azure Firewall para la seguridad de la red.

2.3.3 Google Cloud Platform (GCP)

GCP ofrece funciones de seguridad como la gestión de identidades y accesos (IAM), VPC para la seguridad de la red, Cloud Key Management Service (KMS) para el cifrado y Cloud Security Command Center para la gestión centralizada de la seguridad.

2.3.4 IBM Cloud

IBM Cloud ofrece servicios de seguridad como IBM Cloud Identity and Access Management, IBM Cloud Security Advisor para la gestión de amenazas, IBM Key Protect para la gestión de claves de cifrado e IBM Cloud Hyper Protect Services para entornos de alta seguridad.

2.3.5 Oracle Cloud Infrastructure (OCI)

OCI hace hincapié en la seguridad a través de funciones como la gestión de identidades y accesos (IAM), las redes virtuales en la nube (VCN) para el aislamiento de redes, la gestión de claves para la gestión de claves de cifrado y las zonas de seguridad para entornos de alta seguridad.

2.4 Estado del Arte

2.4.1 ¿Qué es la seguridad de la red?

El "estado del arte" se refiere al nivel actual de desarrollo, avance o sofisticación en un campo o área particular de la tecnología. En el contexto de la seguridad de las redes, el estado de la técnica evoluciona continuamente a medida que surgen nuevas amenazas y se desarrollan soluciones de seguridad para contrarrestarlas. Estos son algunos de los aspectos a tener en cuenta a la hora de diseñar una red segura:

- **Arquitectura de confianza cero**: La confianza cero es un enfoque de la seguridad de la red que no asume ninguna confianza por defecto, tanto dentro como fuera del perímetro de la red. Hace hincapié en controles de acceso estrictos, supervisión continua y mecanismos de autenticación para verificar cada usuario y dispositivo que intenta conectarse a la red.
- **IA y técnicas para el aprendizaje automático**: Las tecnologías de IA y funciones de aprendizaje automático se aplican cada vez más a la seguridad de la red para la detección de amenazas, detección de anomalías y análisis de comportamiento. Estas tecnologías permiten a los sistemas de seguridad detectar y responder a las amenazas emergentes en tiempo real con mayor eficacia.
- **Seguridad de redes definidas por software (SDN)**: SDN permite la gestión centralizada de la red y la programabilidad, lo que puede mejorar la seguridad al permitir un control más granular sobre el tráfico de red, la segmentación y la aplicación de políticas.
- **Seguridad de los contenedores**: Con el auge de la contenedorización y las arquitecturas de microservicios, la seguridad de los contenedores se ha vuelto crítica. Las prácticas más avan-

zadas incluyen la seguridad de las plataformas de orquestación de contenedores, la aplicación de protección en tiempo de ejecución y la gestión segura de imágenes.

- **Seguridad en el cloud**: En estos tiempos cuantas más organizaciones trasladen sus cargas de trabajo a la nube, la seguridad de los entornos en la nube se ven a tener una relevancia más primordial. Las prácticas de seguridad en la nube más avanzadas incluyen la implantación de controles de seguridad nativos de la nube, cifrado, gestión de identidades y accesos (IAM) y supervisión continua.
- **Criptografía cuántica**: Ante la amenaza potencial que suponen los ordenadores cuánticos para los algoritmos criptográficos tradicionales, avanza la investigación en criptografía cuántica. Se están desarrollando algoritmos y protocolos criptográficos resistentes a la cuántica para resistir los ataques de los ordenadores cuánticos.
- **Inteligencia sobre amenazas e intercambio de información**: El intercambio colaborativo de inteligencia sobre amenazas entre organizaciones y el uso de plataformas de inteligencia sobre amenazas son cada vez más importantes para adelantarse a las amenazas en evolución y comprender las tácticas, técnicas y procedimientos (TTP) utilizados por los actores de las amenazas.
- **DevSecOps**: La integración de la seguridad en el proceso DevOps desde el principio (DevSecOps) está ganando adeptos. Esto implica pruebas de seguridad automatizadas, supervisión continua de la seguridad e integración de controles de seguridad en el proceso de desarrollo para identificar y corregir los problemas de seguridad en una fase temprana.
- **Seguridad de los puntos finales(endpoints)**: Con la proliferación del trabajo remoto y los dispositivos móviles, la seguridad de los puntos finales se ha vuelto más crítica. Las soluciones avanzadas de protección de endpoints incorporan aprendizaje automático, análisis de comportamiento e inteligencia de amenazas para detectar y responder a las amenazas de endpoints de manera efectiva.
- **Seguridad de la cadena de bloques**: Continúa la investigación en seguridad de blockchain para abordar las vulnerabilidades en redes de blockchain, contratos inteligentes y aplicaciones descentralizadas (DApps). Se están explorando técnicas como la verificación formal, las mejoras de los mecanismos de consenso y las prácticas de codificación segura.

El estado actual de la seguridad de las redes es dinámico, impulsado por la investigación en curso, las amenazas emergentes, los cambios normativos y los avances tecnológicos. Mantenerse al día de los últimos avances y adoptar las mejores prácticas es esencial para que las organizaciones protejan eficazmente sus redes y datos frente a las ciberamenazas.

La seguridad desde el diseño es un planteamiento del desarrollo de software y hardware que intenta que los sistemas estén lo más libres posible de vulnerabilidades y sean inmunes a los ataques mediante medidas como pruebas continuas, salvaguardas de autenticación y el cumplimiento de las mejores prácticas de programación.

La seguridad de la red se refiere a las diversas contramedidas aplicadas para proteger la red y los datos almacenados en ella o que pasan a través de ella. La seguridad de la red trabaja para mantenerla a salvo de ciberataques, intentos de piratería informática y negligencia de los empleados. Hay tres componentes de la seguridad de la red: hardware, software y servicios en la nube.

2.4.2 Que diferencias podemos encontrar entre seguridad de red y ciberseguridad

La seguridad de red se considera un tipo de seguridad relacionado con la ciberseguridad que se basa particularmente en la seguridad del área digital de una empresa, se podría decir que es la seguridad

dentro de nuestra organización. Dentro de la organización tenemos la infraestructura TI de ella, lo que podemos enumerar como, los componentes digitales, los componentes físicos, la arquitectura de alamcenamiento de datos y elementos de red de cada uno de los usuarios y dispositivos..

2.5 Aspectos básicos de la seguridad de redes

2.5.1 Control de acceso

El sistema comienza verificando la identidad del usuario utilizando un identificador específico, como su ID, nombre de usuario o número de cuenta. Seguido a esto verifica la persona verificando las credenciales del usuario y la contraseña para darle permiso y acceso. Por último, pero no menos importante, la responsabilidad, que implica monitorear la actividad del usuario para que sea responsable de lo que haga en un sistema.

2.5.2 Fragmentación de la red

Se ocupa de dividir una red en componentes lógicos más pequeños para que se puedan agregar controles entre ellos, mejorando el rendimiento y la seguridad.

Las redes de área local virtual (VLAN, por sus siglas en inglés) son una forma común de segmentar la red.

2.5.3 Seguridad del perímetro

Las redes convencionales funcionan en un datacenter físico desde donde se ven conectadas al mundo exterior. Los firewalls (FW), los sistemas de detección de intrusiones (IDS) y los sistemas de prevención de intrusiones (IPS) son algunos ejemplo de elementos que nos ayudan a dar seguridad a nuestra organización por dentro a elementos exterioress. .

2.5.4 Seguridad Cloud

Todos los recursos en línea, incluidas aplicaciones, datos confidencialesvirtual , Ies y servicios, están protegidos por la seguridad enos cloudsbe. Parpoder hacerer estrealidad, sese requieren políticas y técnicas de seguridad robustas, incyendoos firewalls, controles de acceso, encrtacióńnvirtual private networkses (VPN) y programas de recuperaci de datos a posibles eliminaciones o ediciones erróneas de datosóón.

2.5.5 Virtual Private Networks (VPN)

Una red privada virtual, también conocida como VPN, permite enmascarar nuestra dirección IP y ubicación al conectarse a un servidor que nos garantizará la seguridad antes de permitir que accedamos a internet directamente.

2.5.6 Salvaguarda de los endpoints

La seguridad de los endpoints, también conocida como “punto final”, implica el hecho de aplicar un enfoque de multiples capas para todos los dispositivos conectados a una red que sea capaz de reciber una señal de red dentro de esta.

2.6 ¿Qué es la seguridad por diseño?

La seguridad por diseño es el proceso de garantizar la seguridad general de una red incorporando medidas de seguridad en el diseño de la red en lugar de añadir seguridad a posteriori. Estos pilares trabajan conjuntamente para garantizar la seguridad de la red y la protección de los datos sensibles.

El objetivo principal de la seguridad por diseño es garantizar que la seguridad se incorpora al proyecto desde el principio, en lugar de ser una idea tardía. Desde la codificación segura hasta la arquitectura de red, incluye la evaluación de vulnerabilidades potenciales, la identificación de áreas de mejora y la implantación de controles de seguridad sólidos que se adapten a las necesidades específicas de la organización. Al adoptar un enfoque de seguridad por diseño, las organizaciones pueden reducir la probabilidad de que se produzca un ataque y limitar los daños potenciales en caso de que se produzca.

Una de las principales ventajas de la seguridad por diseño es que permite a las organizaciones adelantarse a las amenazas emergentes. Con el rápido ritmo del cambio tecnológico y la constante evolución de las ciberamenazas, es esencial que las organizaciones sean capaces de adaptarse y responder rápidamente a los nuevos retos de seguridad. Al incorporar consideraciones de seguridad en el diseño y desarrollo de una red, las organizaciones pueden garantizar que su arquitectura de red sea capaz de resistir las amenazas más recientes.

Otra ventaja de la seguridad por diseño es que puede ayudar a las organizaciones a cumplir las normativas y estándares del sector. Con un número cada vez mayor de reglamentos y normas, las organizaciones necesitan poder demostrar que están tomando las medidas de seguridad adecuadas para proteger la información sensible, como el Reglamento General de Protección de Datos (GDPR) y el Estándar de Seguridad de Datos de la Industria de Tarjetas de Pago (PCI DSS). Al incorporar consideraciones de seguridad en el diseño y desarrollo de una red, las organizaciones pueden garantizar que cumplen estos requisitos normativos y evitar posibles consecuencias legales y financieras.

2.6.1 Los seis pilares de la arquitectura de seguridad de las redes informáticas

Para protegerse frente a posibles amenazas y garantizar el cumplimiento de la normativa del sector, la seguridad mediante el diseño de la arquitectura de red es esencial para las organizaciones de hoy en día. Mediante la implantación de estos pilares, las organizaciones pueden limitar la propagación de amenazas, autenticar usuarios y ordenadores, supervisar la actividad de la red, aplicar políticas, evitar la filtración de datos y garantizar el cumplimiento de la normativa

- **Segmentation**: Uno de los pilares clave de la seguridad por diseño para la arquitectura de red es la segmentación. La segmentación es la división de una red en segmentos más pequeños y aislados para limitar la propagación de posibles amenazas a la seguridad.La macrosegmentación se utiliza para separar redes que no tienen acceso directo entre sí. Por ejemplo, una red de invitados separada de la red de usuarios corporativos, o una red de gestión separada de los usuarios corporativos. La microsegmentación se utiliza para filtrar dentro de un macrosegmento, por ejemplo, para restringir la comunicación entre usuarios o entre dispositivos. Una vez que un atacante ha obtenido acceso a un único dispositivo o sistema, la segmentación ayuda a limitar el movimiento lateral impidiendo que el atacante se mueva de un segmento a otro
- **Control de acceso a la red**: Esto incluye implementar mecanismos que autentiquen a los usuarios y ordenadores antes de permitirles el acceso a la red. Esto puede incluir el uso de

802.1X para la autenticación de LAN, cortafuegos para aplicar reglas al flujo de tráfico entre segmentos y proxies para aplicar políticas de uso de Internet. Sólo los usuarios y dispositivos autorizados pueden acceder a la red y a los datos confidenciales.

- **Visibilidad**: Esto incluye la comprensión de los flujos de tráfico, la identificación de amenazas potenciales y la supervisión de actividades inusuales. Esto puede lograrse mediante el uso de productos de terceros, como la inspección profunda de paquetes, NetFlow, las escuchas de paquetes, la duplicación de paquetes y otras herramientas de seguridad. Esta visibilidad permite a las empresas detectar y responder a posibles amenazas en tiempo real
- **CID**: Confidencialidad, integridad y disponibilidad- es otro pilar importante de la seguridad por diseño para la arquitectura de redes. La confidencialidad consiste en mantener los datos de la organización privados y secretos. La integridad es la garantía de que los datos no han sido manipulados ni alterados. La disponibilidad consiste en garantizar que los datos son accesibles y utilizables cuando se necesitan. La protección de estos tres elementos garantiza que los datos estén protegidos contra el acceso no autorizado, la modificación o la destrucción.
- **Cumplimiento de la normativa**: El último pilar de la seguridad por diseño para la arquitectura de redes es el cumplimiento de la normativa y los estándares. Esto se refiere a garantizar que la organización cumple los estándares establecidos por las normativas del sector, como HIPAA o PCI DSS.

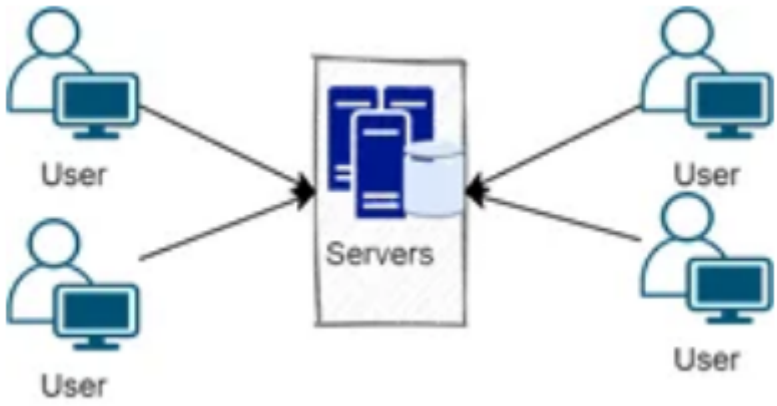
3 Modelo Teórico

3.1 Modelos de arquitectura de Red

3.1.1 Modelo de 2 capas

El modelo cliente/servidor es uno de los modelos de aplicaciones distribuidas de mayor éxito en las últimas décadas. Consiste en construir un sistema basado en capas en el que se distribuyen roles entre un ordenador cliente y uno o más servidores. Organizan los roles de cada equipo dependiendo de su funcionalidad y se caracteriza por la estanqueidad de cada nivel.

Este tipo de arquitectura utiliza un cliente que ataca los servicios de un servidor que hace todo lo necesario para servir la aplicación. Es el propio servidor el que utiliza recursos internos para servir como datos, lógica e información al cliente.



Un modelo de dos capas se refiere típicamente a la simplificación de las funciones de la red en

dos niveles principales en las arquitecturas de red. Esta simplificación se puede aplicar tanto a las redes locales como a las redes más amplias. Aquí se enumeran los componentes y características principales de un modelo de red de dos capas:

3.1.2 Capa de Acceso

En una arquitectura de red de dos capas, la capa de acceso es la primera. Sus principales características y funciones son:

- Conexión de dispositivos finales: esta capa se encarga de la conexión directa de dispositivos finales, como impresoras, teléfonos y computadoras de usuario.
- Los enrutadores se utilizan en esta capa para conectar dispositivos a la red local (LAN).
- Control de acceso: Aplicar reglas de seguridad fundamentales, como control de acceso mediante autenticación y reglas de VLAN para segmentación de la red.
- QoS (Calidad de Servicio): Aplicar políticas de calidad de servicio fundamentales para priorizar ciertos tipos de tráfico.

3.1.3 Capa de Distribución

La capa de distribución sirve como un puente entre la capa de acceso y el núcleo de la red. Sus características y funciones incluyen:

- Agregar tráfico: recopilar y consolidar el tráfico de varias capas de acceso antes de enviarlo al núcleo de la red.
- Enrutamiento: llevar a cabo tareas de enrutamiento para guiar el tráfico entre varias subredes y redes.
- Seguridad: Implementar políticas de seguridad más avanzadas, como filtrado de tráfico y control de acceso a nivel de red.
- Redundancia y disponibilidad: implementar configuraciones de alta disponibilidad y protocolos de redundancia de enrutadores para garantizar la redundancia y la disponibilidad de la red.
- QoS avanzada: gestión y priorización del tráfico más avanzadas para garantizar el rendimiento

3.2 Beneficios del Modelo de Dos Capas

1. Simpledad: En comparación con modelos más complejos de múltiples capas, las arquitecturas de dos capas son más fáciles de diseñar, implementar y administrar.
2. La escalabilidad permite la expansión de una red agregando más interruptores a la capa de acceso sin necesidad de rediseñar la infraestructura de la red.
3. Costo: Debido a la menor cantidad de dispositivos y la simplicidad de la gestión, puede ser más rentable.
4. Centralización de control: el servidor controla el acceso, los recursos y la integridad de los datos, por lo que un programa cliente defectuoso o no autorizado no puede dañar el sistema.
5. La escalabilidad significa que se puede aumentar la capacidad tanto de los servidores como de los clientes por separado.
6. Tecnologías: existen algunas suficientemente desarrolladas, diseñadas para el paradigma de C/S, que brindan seguridad en las transacciones, interfaz amigable y facilidad de uso.

3.2.1 Modelo de 3 Capas

A diferencia del modelo de dos capas, el servidor que los clientes atacan sirve como intermediario entre otro equipo que almacena los datos. Esta máquina se llama Midelware y normalmente se encarga de proporcionar la lógica de aplicación.

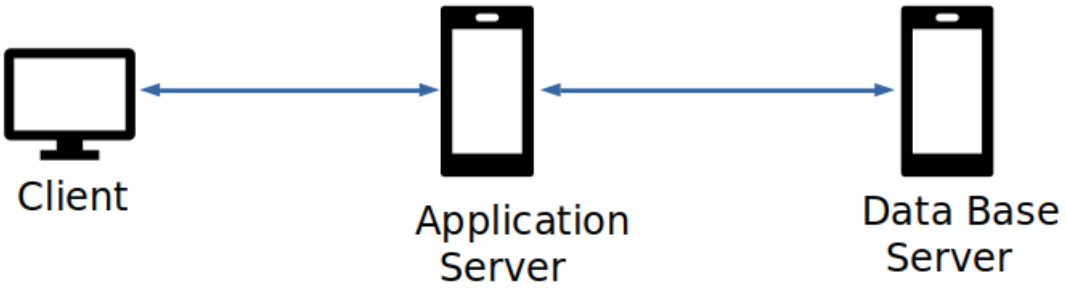
El cliente solo ejecuta la capa de presentación y el servidor de datos transmite y registra los datos procesados de la capa de aplicación.

Esta separación de roles es bastante flexible, pero su mayor activo es que se puede modificar cualquiera de las capas sin tener incidencia en el resto de estas. Esta peculiaridad aporta al modelo versatilidad y adaptabilidad en entornos distribuidos que evolucionan a gran velocidad.

En las arquitecturas de red, el modelo de tres capas organiza los dispositivos de red en tres niveles diferentes: la capa de acceso, la capa de distribución y la capa núcleo. Las redes comerciales suelen usar esta arquitectura para mejorar la gestión, la escalabilidad y el rendimiento.

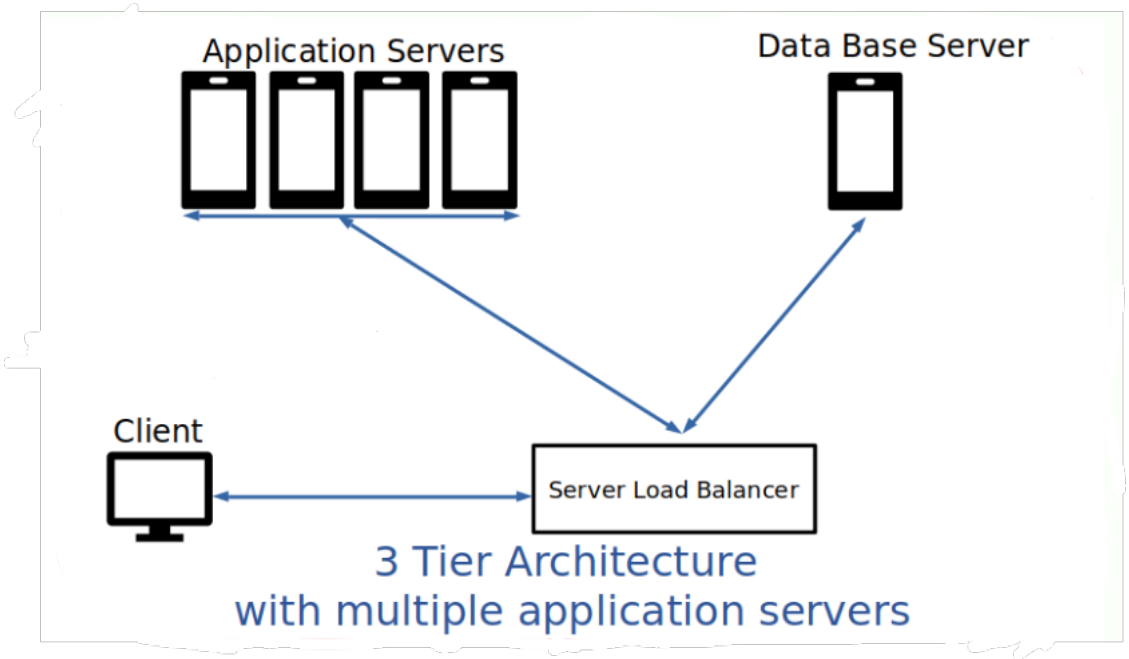
1. La capa de acceso está más cerca de los usuarios finales. Los dispositivos en esta capa, que suelen ser interruptores de capa 2, permiten la conexión directa a dispositivos finales como computadoras y teléfonos. La capa de acceso gestiona tareas importantes como la segmentación de VLAN, la seguridad de puertos y el PoE (Power over Ethernet) para dispositivos como teléfonos VoIP y puntos de acceso inalámbricos.
2. La capa de distribución conecta la capa de acceso con la capa núcleo. Antes de transferir el tráfico a la capa núcleo, su función principal es agregar el tráfico de múltiples interruptores de acceso. Además, implementa políticas de seguridad, balanceo de carga y enrutamiento entre VLANs. Los interruptores de capa 3 se utilizan con frecuencia en esta capa, lo que permite un mayor control y gestión del tráfico.
3. La capa central, también conocida como la columna vertebral de la red, administra el tráfico de alto rendimiento entre los diferentes segmentos de la red. La capa núcleo debe ser altamente redundante y eficiente y diseñada para maximizar la velocidad de conmutación de paquetes y reducir la latencia. Aquí se utilizan interruptores de alta capacidad y velocidad, que pueden manejar grandes cantidades de datos y crear conexiones rápidas entre los diversos módulos de distribución.

La estructura modular proporcionada por el diseño de tres capas facilita la expansión y el mantenimiento de la red. La arquitectura de núcleo colapsado, que combina las funciones de las capas de distribución y el núcleo para reducir costos y simplificar el diseño, mientras se mantiene un nivel adecuado de rendimiento y redundancia, es una alternativa a este modelo en entornos más pequeños.



3 Tier Architecture

En la arquitectura de 3 niveles, el usuario debe acceder al servidor de aplicaciones. El servidor de aplicaciones se encarga de la transferencia de datos entre el cliente y el servidor de base de datos. Puede haber varios servidores de aplicaciones trabajando para una sola aplicación, que puede ser gestionada por un equilibrador de carga del servidor (SLB). La función de SLB se puede entender por la imagen de abajo.



El usuario solicita el acceso a la aplicación al servidor de aplicaciones. El servidor de aplicaciones conecta la base de datos del cliente y crea la interfaz de usuario del cliente.

En este artículo, describo las bases de la arquitectura de redes de 2 y 3 niveles. Espero que aprendas algo de este pequeño documento. Para cualquier consulta o sugerencia sobre este tema o este sitio, puede dejar un comentario a continuación o comunicarse con nosotros.

3.2.2 Integración de Servicios en el Modelo

La integración de servicios dentro del modelo de tres capas en arquitecturas de red sigue una estructura lógica donde cada capa desempeña funciones específicas para garantizar la eficiencia, seguridad y escalabilidad de la red.

El marco conceptual que se utiliza para comprender y aplicar protocolos de red en siete capas es el modelo OSI. Cada capa ofrece servicios específicos que ayudan a las personas a comunicarse a través de una red.

4 Implementación

Necesitaremos instalar de antemano algunos modulos por linea de comando para poder usar NetworkX, pyplot, matplotlib y pydot

[1]:
```
#!pip install pydot
#!pip install graphviz
#!pip install networkx
#!pip install matplotlib
```

Inicialmente necesitaremos importar las librerias necesarias para poder crear nodos y aristas en un grafo. A su vez necesitaremos librerias para poder dibujar los grafos, las colocaremos como variables de entorno

[ ]:
```
```

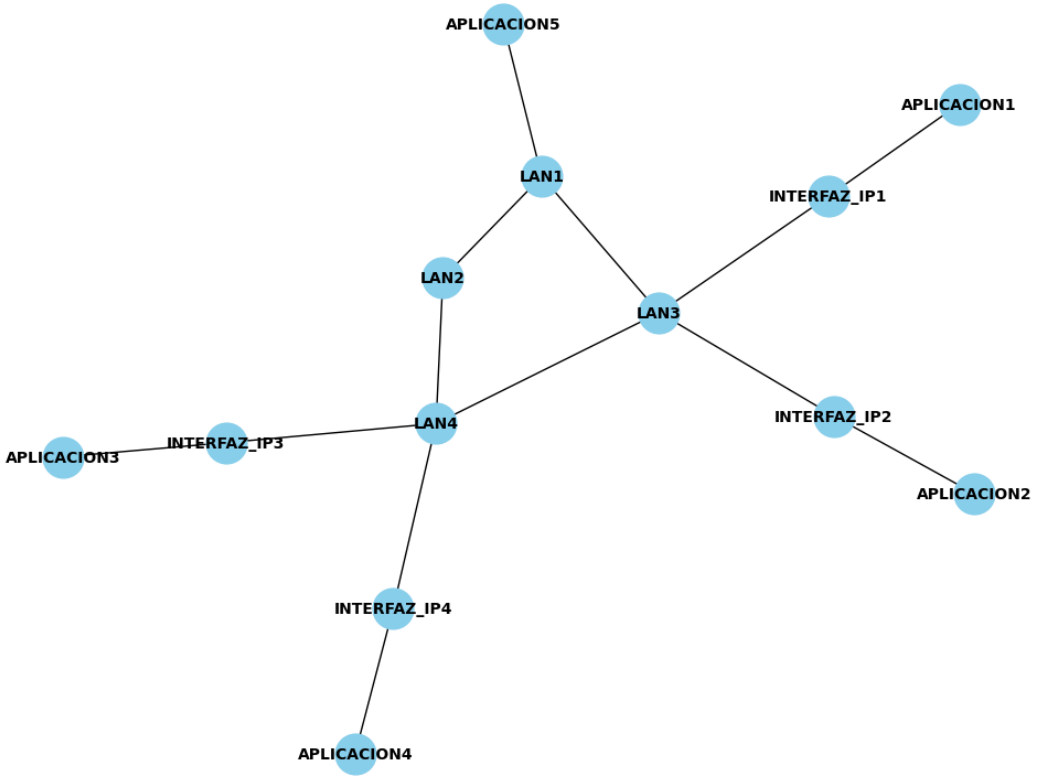
[1]:
```
import networkx as nx
import matplotlib.pyplot as plt
import copy
import json
#import pydot
#import graphviz_layout
```

[3]:
```
with open('./assets/nodos_aristas.json', 'r') as f:
    data = json.load(f)

# Crear un grafo
G = nx.Graph()

# Añadir nodos con atributos
for node in data['nodes']:
    G.add_node(node['id'], **{k: v for k, v in node.items() if k != 'id'})

# Añadir aristas
for edge in data['edges']:
    G.add_edge(edge['source'], edge['target'])


print("De los siguientes nodos cuales son routers('fin' para terminar): ")
```

```
for node in G.nodes():
    if node.startswith("APLICACION"):
        print(node)

while True:
    dato = input()

    if dato.lower() == 'fin':
        break

    if dato in G.nodes():
        G.nodes[dato]['ip_forward'] = True

for node,atributos in G.nodes(data=True):
    if atributos.get('ip_forward') is True:
        G.nodes[node]['nivel2'] = False
        G.nodes[node]['nivel3'] = True
        G.nodes[node]['nivel4'] = True

    if node.startswith("APLICACION") and 'ip_forward' not in atributos:
        G.nodes[node]['ip_forward'] = False
        G.nodes[node]['nivel2'] = False
        G.nodes[node]['nivel3'] = False
        G.nodes[node]['nivel4'] = True

    if node.startswith("LAN"):
        G.nodes[node]['nivel2'] = True
        G.nodes[node]['nivel3'] = True
        G.nodes[node]['nivel4'] = True

    if node.startswith("INTERFAZ_IP"):
        G.nodes[node]['nivel2'] = False
        G.nodes[node]['nivel3'] = True
        G.nodes[node]['nivel4'] = True

# Dibujar el grafo para verificar que se ha creado correctamente
plt.figure(figsize=(10, 8))
pos = nx.spring_layout(G)  # Posición de los nodos para la visualización
nx.draw(G, pos, with_labels=True, node_size=700, node_color="skyblue",␣
 ↪font_size=10, font_color="black", font_weight="bold")
plt.show()
```

```
De los siguientes nodos cuales son routers('fin' para terminar):
APLICACION1
APLICACION2
APLICACION3
APLICACION4
```

```
APLICACION5
 APLICACION2
 fin
```



[2]:
```
G = nx.Graph()
#pos = nx.tree_layout(G)

# Vamos a hacer que la aplicacion3 y la aplicacion4 si puedan hacer de gateway␣
 ↪de nivel3 y el interfaz_ip2 no lo sea

G.add_node("APLICACION1", ip_forward=True, nivel2=False, nivel3=True,␣
 ↪nivel4=True)
G.add_node("LAN1", rango="10.68.1.0", nivel2=True, nivel3=True, nivel4=True)
G.add_node("APLICACION2", ip_forward=True, nivel2=False, nivel3=True,␣
 ↪nivel4=True)
G.add_node("LAN2", rango="10.68.2.0", nivel2=True, nivel3=True, nivel4=True)
G.add_node("INTERFAZ_IP1", nivel2=False, nivel3=True, nivel4=True)
G.add_node("INTERFAZ_IP2", nivel2=False, nivel3=False, nivel4=True)
```

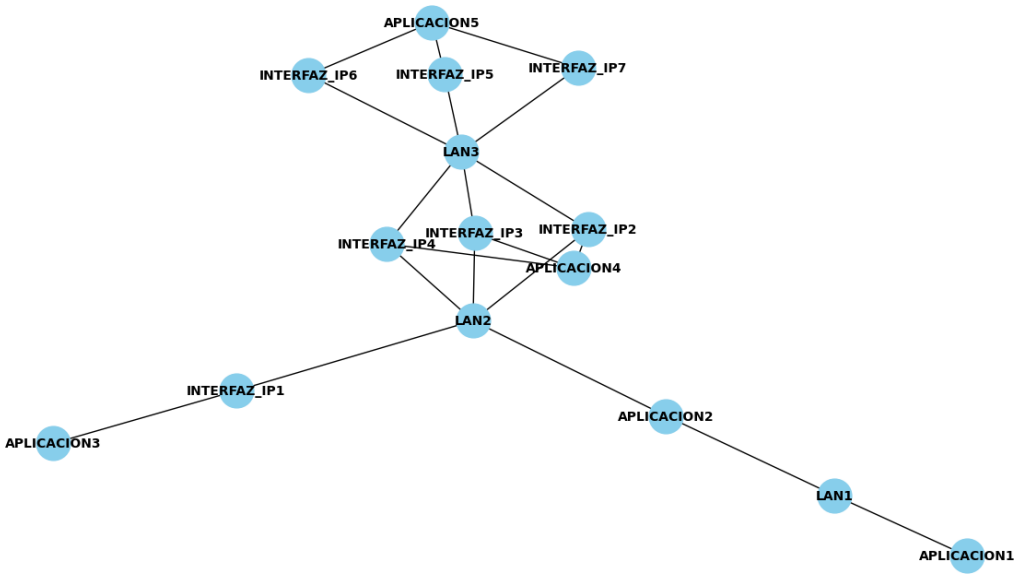
```
G.add_node("INTERFAZ_IP3", nivel2=False, nivel3=True, nivel4=True)
G.add_node("INTERFAZ_IP4", nivel2=False, nivel3=True, nivel4=True)
G.add_node("APLICACION3", ip_forward=False, nivel2=False, nivel3=True,␣
 ↪nivel4=True)
G.add_node("APLICACION5", ip_forward=False, nivel2=False, nivel3=False,␣
 ↪nivel4=True)
G.add_node("LAN3", rango="10.68.3.0", nivel2=True, nivel3=True, nivel4=True)
G.add_node("INTERFAZ_IP5", ip_forward=False, nivel2=False, nivel3=True,␣
 ↪nivel4=True)
G.add_node("INTERFAZ_IP6", ip_forward=False, nivel2=False, nivel3=True,␣
 ↪nivel4=True)
G.add_node("INTERFAZ_IP7", ip_forward=False, nivel2=False, nivel3=True,␣
 ↪nivel4=True)
G.add_node("APLICACION4", nivel2=False, nivel3=True, nivel4=True)


G.add_edge("APLICACION1","LAN1", direccion_ip = "10.68.1.1")
G.add_edge("LAN1","APLICACION2", direccion_ip = "10.68.2.1")
G.add_edge("APLICACION2","LAN2", direccion_ip = "10.68.2.2")
G.add_edge("LAN2","INTERFAZ_IP1", direccion_ip = "10.68.2.3")
G.add_edge("LAN2","INTERFAZ_IP2", direccion_ip = "10.68.2.4")
G.add_edge("LAN2","INTERFAZ_IP3", direccion_ip = "10.68.2.5")
G.add_edge("LAN2","INTERFAZ_IP4", direccion_ip = "10.68.2.6")
G.add_edge("APLICACION3","INTERFAZ_IP1")
G.add_edge("APLICACION4","INTERFAZ_IP2")
G.add_edge("APLICACION4","INTERFAZ_IP3")
G.add_edge("APLICACION4","INTERFAZ_IP4")
G.add_edge("INTERFAZ_IP2","LAN3", direccion_ip = "10.68.3.1")
G.add_edge("INTERFAZ_IP3","LAN3", direccion_ip = "10.68.3.2")
G.add_edge("INTERFAZ_IP4","LAN3", direccion_ip = "10.68.3.3")
G.add_edge("LAN3","INTERFAZ_IP5", direccion_ip = "10.68.3.4")
G.add_edge("LAN3","INTERFAZ_IP6", direccion_ip = "10.68.3.5")
G.add_edge("LAN3","INTERFAZ_IP7", direccion_ip = "10.68.3.6")
G.add_edge("APLICACION5","INTERFAZ_IP5")
G.add_edge("APLICACION5","INTERFAZ_IP6")
G.add_edge("APLICACION5","INTERFAZ_IP7")
```

[6]:
```
# print("Lista de nodos:")
# i = 0
# for node in G.nodes():
#     i+=1
#     print(node + " ", end='')
#     if i == 6:
#         print("")
#         i = 0

plt.figure(figsize=(12, 7))
```

```
pos = nx.spring_layout(G, seed=0)  # Posición de los nodos para la visualización
nx.draw(G, pos, with_labels=True, node_size=700, node_color="skyblue",␣
 ↪font_size=10, font_color="black", font_weight="bold")
plt.show()

print("Elige un Nodo Inicial para empezar la búsqueda")
while True:
    nodo_inicial = input()
    if nodo_inicial in G.nodes():
        break
print("Elige un Nodo Destino para finalizar la búsqueda")
while True:
    nodo_destino = input()
    if nodo_destino in G.nodes():
        break


print("Que nivel de caminos quieres buscar")
nivel = int(input())

# Función recursiva para poder encontrar todos los caminos desde un nodo␣
 ↪inicial a un destino con restricción sobre un atributo

def encontrar_caminos(grafo, nodo_actual, nodo_destino, nivel,camino_actual=[]):
    camino_actual = camino_actual + [nodo_actual]  # Agregar el nodo actual al␣
 ↪camino actual
    nivel_aceptado = False
    if nodo_actual == nodo_destino:   # Si hemos alcanzado el nodo de destino,␣
 ↪añadir el camino actual a la lista de caminos
        return [camino_actual]

    caminos = []  # Lista para almacenar todos los caminos encontrados

    for vecino in grafo.neighbors(nodo_actual):

        if nivel == 2:
            nivel_aceptado = grafo.nodes[vecino].get('nivel2', False)
        if nivel == 3:
            nivel_aceptado = grafo.nodes[vecino].get('nivel3', False)
        if nivel == 4:
            nivel_aceptado = grafo.nodes[vecino].get('nivel4', False)

        if nivel_aceptado:
            if vecino not in camino_actual:  # Evitar ciclos
                nuevos_caminos = encontrar_caminos(grafo, vecino, nodo_destino,␣
 ↪nivel, camino_actual)
                for nuevo_camino in nuevos_caminos:
```

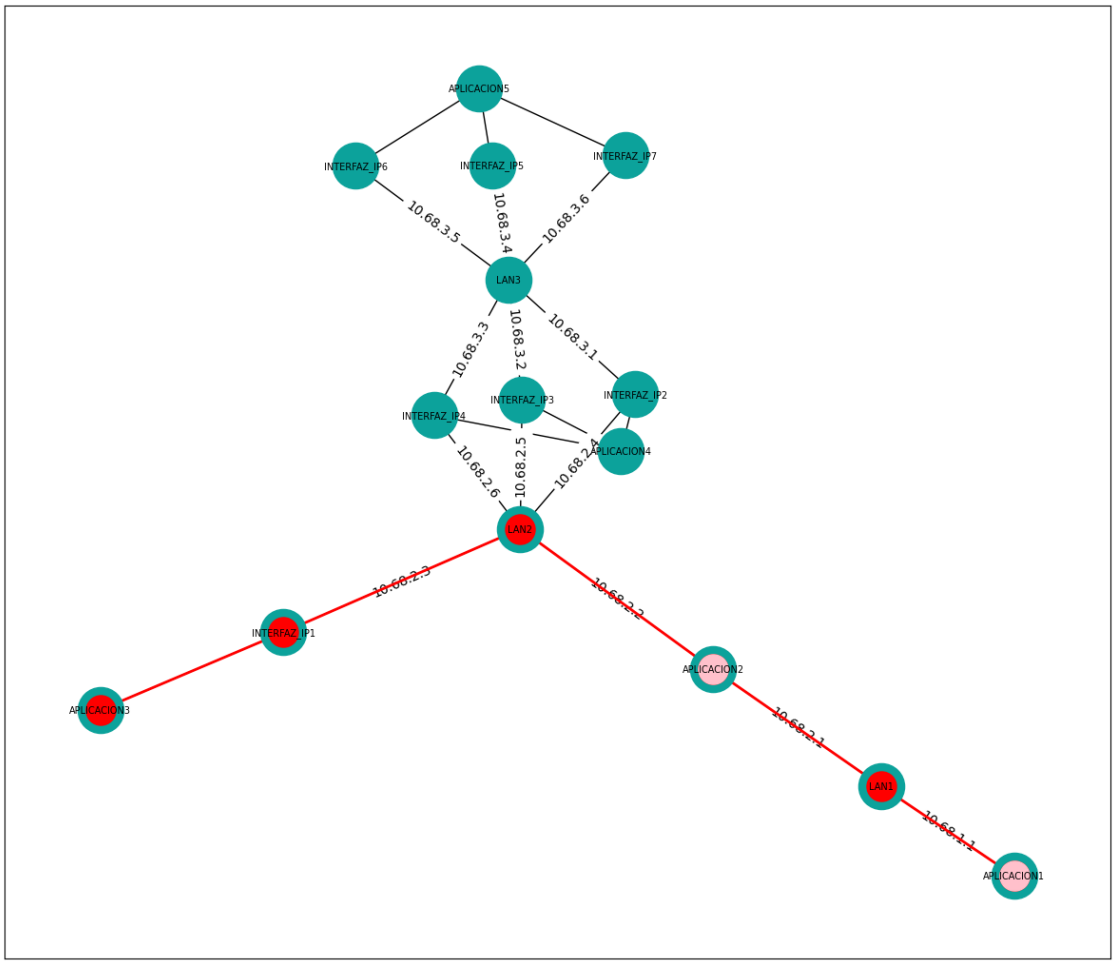
```
                    caminos.append(nuevo_camino)

    return caminos

#Lista de caminos encontrados
caminos = encontrar_caminos(G,nodo_inicial, nodo_destino, nivel)

# for camino in caminos:
#     print(camino)

# Encontramos todos los caminos desde el nodo inicial hasta el nodo de destino

# todos_caminos = nx.all_simple_paths(G, source=nodo_inicial,␣
 ↪target=nodo_destino)

# Creamos un diccionario para mapear los nodos y bordes de los caminos
camino_nodes = set()
camino_edges = set()
for camino in caminos:
    for i in range(len(camino) - 1):
        camino_nodes.add(camino[i])
        camino_edges.add((camino[i], camino[i+1]))
    #print(camino)
    camino_nodes.add(camino[-1])  # Agregar el último nodo del camino


# Colocar en el plano cada nodo para que sea un arbol Top-down
posicion = nx.spring_layout(G, seed=0)
posicion_manual = {"APLICACION1": (0, 0), "LAN1": (0, -32), "APLICACION2": (0,␣
 ↪-64), "LAN2": (0, -96), "INTERFAZ_IP1": (-4, -96), "APLICACION3": (-8, -96),␣
 ↪"INTERFAZ_IP2": (-4, -128),  "INTERFAZ_IP3": (0, -128),  "INTERFAZ_IP4": (4,␣
 ↪-128),  "APLICACION4": (12, -144),  "LAN3": (0, -160),  "INTERFAZ_IP5": (-4,␣
 ↪-192),  "INTERFAZ_IP6": (0, -192),  "INTERFAZ_IP7": (4, -192), ␣
 ↪"APLICACION5": (0, -224)}

# Obtener el atributo direccion_ip de la arista
etiquetas_aristas = nx.get_edge_attributes(G, 'direccion_ip')

fig = plt.figure(figsize=(15, 13))

# draw graph
nx.draw_networkx(G, posicion, node_size=1200, node_color = "#0CA29B", font_size␣
 ↪= 7)
# draw edge attributes
nx.draw_networkx_edge_labels(G, posicion, edge_labels=etiquetas_aristas);
```

```
nx.draw_networkx_nodes(G, posicion, nodelist=camino_nodes, node_color='red',␣
 ↪node_size=500)  # Superponemos los nodos de los caminos
nx.draw_networkx_edges(G, posicion, edgelist=camino_edges, edge_color='red',␣
 ↪width=2)  # Superponemos los bordes de los caminos

camino_nodes = [node for node, data in G.nodes(data=True) if 'ip_forward' in␣
 ↪data and data['ip_forward']]
nx.draw_networkx_nodes(G, posicion, nodelist=camino_nodes, node_color='pink',␣
 ↪node_size=500)  # Superponemos los nodos de los caminos
```




```
Elige un Nodo Inicial para empezar la búsqueda
 APLICACION3
Elige un Nodo Destino para finalizar la búsqueda
 APLICACION1
Que nivel de caminos quieres buscar
 3
```

[6]: <matplotlib.collections.PathCollection at 0x24c0bd73cb0>



4.1 Complejidad Temporal de la función de búsqueda de caminos

La función encontrar_caminos busca todos los caminos entre dos nodos en un grafo, considerando una restricción de nivel para los nodos vecinos. La complejidad temporal de esta función puede ser analizada considerando los siguientes factores:

4.1.1 Estructura del Grafo

El grafo podemos dividirlo en tres elementos esenciales para el calculo de la complejidad, N como el número de nodos del grafo, E como el número de aristas del grafo y D como el grado máximo de cualquier nodo del grafo.

4.1.2 Recursividad

La función explora todos los caminos posibles desde el nodo actual hasta el nodo destino. En el peor de los casos, la profundidad de la recursión puede ser el número total de nodos N. A la hora de realizar la búsqueda de vecinos la función explora todos los vecinos del nodo actual. Si

consideramos que, en el peor de los casos, cada nodo puede tener hasta d vecinos, la función realiza d llamadas recursivas por cada nodo.

El número de caminos potenciales entre dos nodos puede ser exponencial en el tamaño de un grafo no dirigido y sin restricciones. En el peor de los casos, la función podría explorar todos los caminos disponibles, lo cual tiene una complejidad exponencial.

4.1.3 Complejidad Temporal

Para determinar la complejidad temporal exacta de la función, consideramos los factores antes mencionados. En el peor de los casos:

1. Número de Caminos: La función tiene la capacidad de explorar un número exponencial de caminos. En el caso más extremo, la función podría tener que explorar hasta $O(d^N)$ rutas, con d vecinos para cada nodo y N niveles de profundidad.
2. Verificación de Niveles: Para cada vecino, la función verifica una condición de nivel, que tiene una complejidad constante $O(1)$.

Por lo tanto, la complejidad temporal en el peor de los casos es: $O(d^N)$

4.1.4 Verificación de Niveles en la Función

Cada vez que se verifica si un vecino cumple con la condición de nivel, se realizan las siguientes operaciones:

- Acceso al diccionario de nodos para obtener los atributos de un nodo específico.
- Búsqueda de un atributo dentro del diccionario del nodo.
- Comparación del nivel actual con el nivel requerido.

El proceso completo de verificación de nivel para un nodo vecino es una operación de tiempo constante porque todas estas operaciones (acceso al diccionario, búsqueda de clave en el diccionario y comparación) tienen una complejidad $O(1)$.

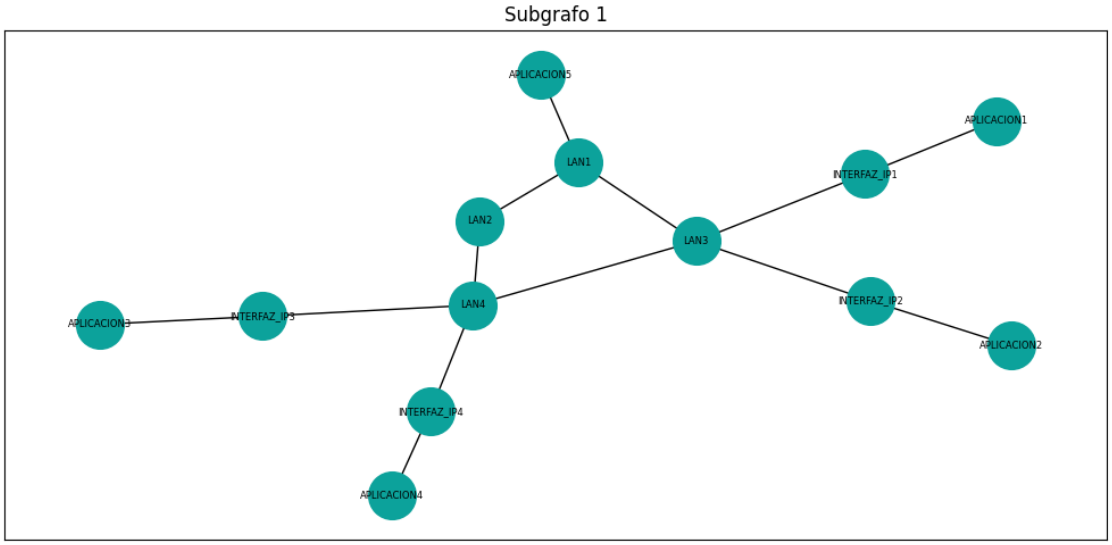
4.2 Encontrar áreas de conectividad dentro de un grafo

[5]:
```
G2 = G.copy()

print("Que nivel de caminos quieres buscar")
nivel = int(input())

fig = plt.figure(figsize=(15, 13))

Conectados_router = [] # Lista de nodos vecinos a los routers

for node, data in G.nodes(data=True): # data=True: Este argumento opcional␣
 ↪indica que quieres incluir los datos de los nodos en la vista devuelta.Al␣
 ↪establecerlo en True, la vista contendrá tuplas (nodo, datos_del_nodo).
    #if 'ip_forward' in data and data['ip_forward']:
    if nivel == 2 and not data['nivel2']:
        vecinos = list(G2.neighbors(node))
        vecinos.insert(0,node) # Añadimos el nodo que vamos a quitar el inicio
```

```
        Conectados_router.append(vecinos) # Añado a la lista los nodos␣
 ↪conectados al nodo que vamos a quitar
        #G2.remove_node(node)
    elif nivel == 3 and not data['nivel3']:
        vecinos = list(G2.neighbors(node))
        vecinos.insert(0,node) # Añadimos el nodo que vamos a quitar el inicio
        Conectados_router.append(vecinos) # Añado a la lista los nodos␣
 ↪conectados al nodo que vamos a quitar
        #G2.remove_node(node)
    elif nivel == 4 and not data['nivel4']:
        vecinos = list(G2.neighbors(node))
        vecinos.insert(0,node) # Añadimos el nodo que vamos a quitar el inicio
        Conectados_router.append(vecinos) # Añado a la lista los nodos␣
 ↪conectados al nodo que vamos a quitar
        #G2.remove_node(node)


# Imprimir el nodo quitado y sus vecinos
for lista in Conectados_router:
    G2.remove_node(lista[0])
    print(f"Vecinos: {lista}")
#           Resultado
# Vecinos: ['APLICACION1', 'LAN1']
# Vecinos: ['APLICACION2', 'LAN1', 'LAN2']

# Encuentra los componentes conectados después de eliminar los nodos
componentes_conectados = list(nx.connected_components(G2))

# Crea subgrafos para cada componente conectado
subgrafos = [G2.subgraph(componente) for componente in componentes_conectados]

# draw graph
# nx.draw_networkx(G2, posicion_manual, node_size=1200, node_color = "#0CA29B",␣
 ↪font_size = 7)

# Dibuja los subgrafos obtenidos
plt.figure(figsize=(10, 5))
for i, subgrafo in enumerate(subgrafos): # enumerate devuelve tanto el índice␣
 ↪como el elemento correspondiente en cada iteración.
    subgrafo_editable = nx.Graph(subgrafo) # Crear un nuevo grafo editable
    for lista in Conectados_router:
        for elemento_lista in lista:
            for node in subgrafo.nodes:
                #print(f"Subgrafo: {node}")
                if node == elemento_lista:
                    subgrafo_editable.add_node(lista[0])
                    subgrafo_editable.add_edge(lista[0],node)
```

```
    plt.subplot(1, len(subgrafos), i+1)
    # if i == 1:
    #     G3 = subgrafo_editable
    # if i == 2:
    #     G4 = subgrafo_editable
    etiquetas_aristas_subgrafo = nx.get_edge_attributes(subgrafo_editable,␣
 ↪'direccion_ip')
    nx.draw_networkx(subgrafo_editable, posicion, node_size=900,␣
 ↪node_color="#0CA29B", font_size=6, with_labels=True)
    plt.title(f"Subgrafo {i+1}")
plt.tight_layout()
plt.show()

#G5 = nx.Graph()

# Obtener la Unión de Subgrafo 2 y 3

# plt.figure(figsize=(5, 7))
# G5.add_nodes_from(G3.nodes(data=True))
# G5.add_edges_from(G3.edges(data=True))
# G5.add_nodes_from(G4.nodes(data=True))
# G5.add_edges_from(G4.edges(data=True))
# nx.draw_networkx(G5, posicion=posicion_manual, node_size=900,␣
 ↪node_color="#0CA29B", font_size=6, with_labels=True)
# plt.title(f"Subgrafo (2 + 3)")
# plt.tight_layout()
# plt.show()
```

```
Que nivel de caminos quieres buscar
 3
Vecinos: ['APLICACION1', 'INTERFAZ_IP1']
Vecinos: ['APLICACION3', 'INTERFAZ_IP3']
Vecinos: ['APLICACION4', 'INTERFAZ_IP4']
Vecinos: ['APLICACION5', 'LAN1']
<Figure size 1500x1300 with 0 Axes>
```



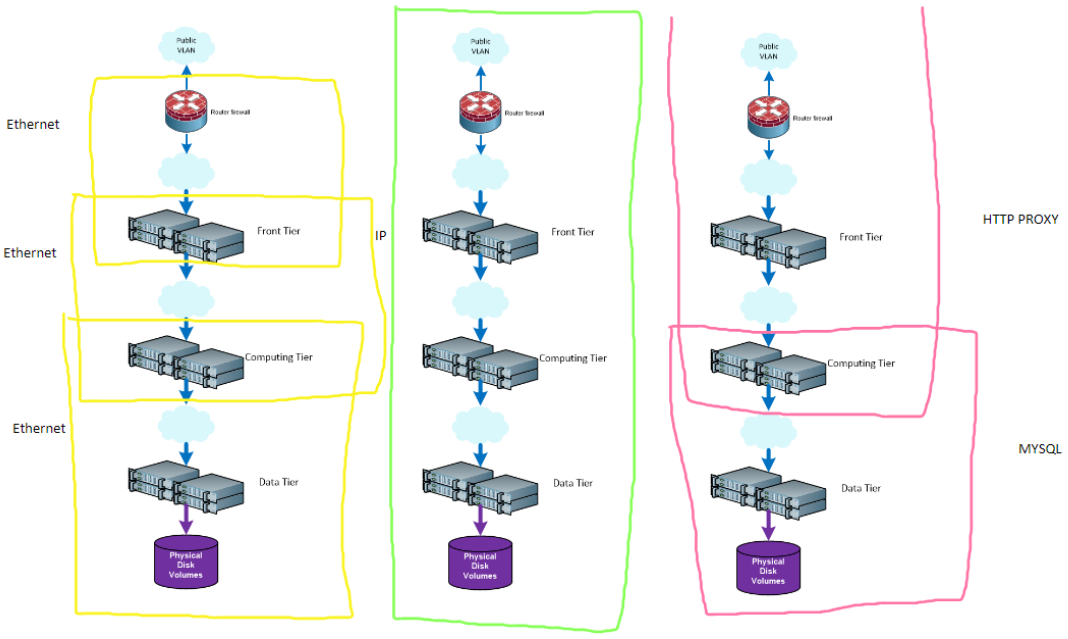
4.3 all_simple_paths

Este algoritmo utiliza una búsqueda por profundidad modificada para generar los caminos. Se puede encontrar un camino simple en tiempo $O(V + E)$, pero el número de caminos simples en un grafo puede ser muy grande, por ejemplo $O(n!)$ en el grafo completo de orden n.

4.4 Explorar si se pueden contruir zonas de conectividad de un protocolo de nivel superior combinando zonas de conectividad de un nivel inferior

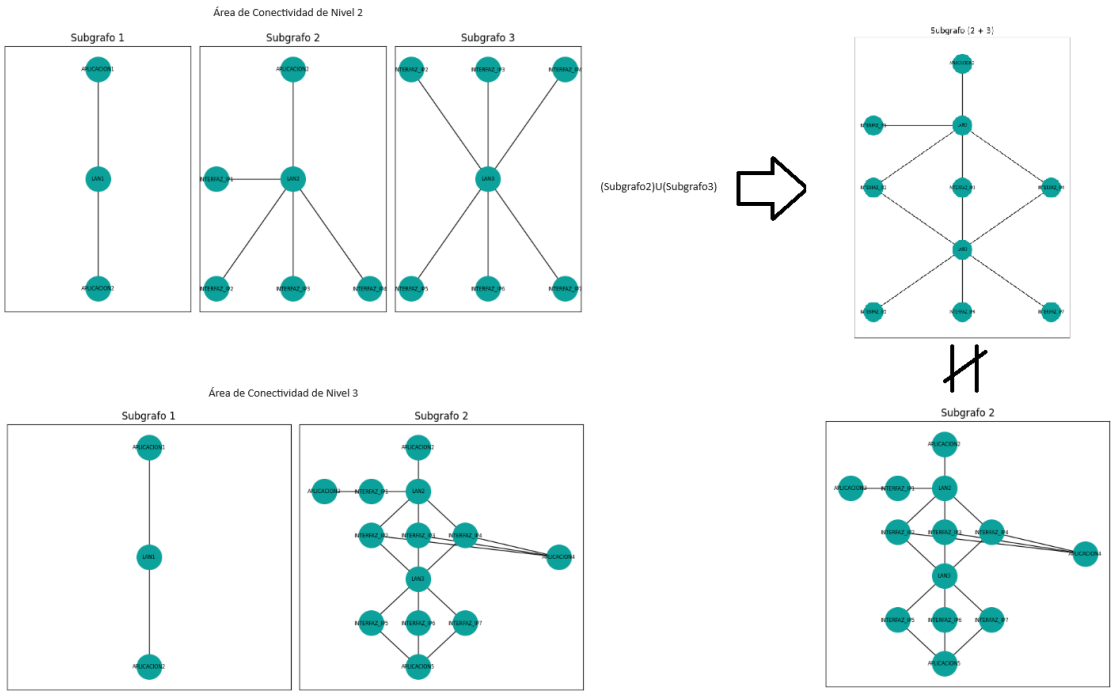
Ya que hemos conseguido determinar las zonas de conectividad de ciertos niveles en el algoritmo anterior, ahora vamos a estudiar si podemos obtener una zona de conectividad de un nivel superior.

Si conseguimos las distintas zonas de nivel X de una red representada como un grafo podremos llegar al siguiente razonamiento. ¿Es posible obtener obtener las zonas de conectividad del grafo haciendo la intersección de las zonas de conectividad de las del nivel X?



Tal como observamos en la imagen en el nivel 2 se nos detalla como color amarilo y abarca la misma zona que el nivel 3 de IP pero si realizasemos la union de los nodos intersección entre los subgrafos resultantes de Front-Tier junto con el Computing Tier y el Data Tier. Para llegar a ello necesitamos realizar una serie de operaciones sobre los grafos. Si calculamos la intersección de los subgrafos resultantes y volvemos a crear subgrafos con ellos podriamos llegar a obtener areas de conectividad similares a los de niveles superiores.

Este supuesto no se cumple ya que para obtener las areas de conectividad necesitamos obtener los nodos que no hacen gateway de ese nivel, los quitamos del grafo para luego añadirlos a los subgrafos que han quedado conectados. Pero se produce un caso especial en el que si obtenemos que dos nodos que estan conectados no son gateway de dicho nivel, para el nivel superior puede estar conectado a uno de los vecinos del area de conectividad.



Como podemos observar en la imagen la union de las areas de interesección del nivel 2 no dan como resultante el nivel 3.

4.5 Obstáculos a la hora de realizar el código

Inicialmente la idea que se nos presenta para realizar el código sería como representamos los nodos dentro de la estructura y como vamos a movernos dentro de los nodos y aristas. Los nodos tendrian como atributos las IP junto con sus MACs y las aristas no tendrían ningún atributo. Primero se realizaría la ejecución de all_simple_paths para hallar todos los caminos, al haber encontrado todos los caminos ya podemos ir quitando los caminos que no queremos con nuestras restricciones. Este método tiene un problema fundamental y es que no estamos acotando los caminos encontrados, simplemente encontramos todos lo caminos y eliminamos los que no nos interesen, esto no nos resulta de ninguna manera útil para lo que queremos ya que si el grafo tiene un número considerable de nodos y conexiones entre ellos la búsqueda de caminos no podría llegar a terminar ya que como podemos observar en el algoritmo de all_simple_paths, este algoritmo utiliza una búsqueda por profundidad modificada para generar los caminos. Se puede encontrar un camino simple en tiempo $O(V+E)$, pero el número de caminos simples en un grafo puede ser muy grande, por ejemplo $O(n!)$ en el grafo completo de orden n.

Debido a todos los problemas que acarreaba usar all_simple_paths, necesitaba encontrar un algoritmo de NetworkX que busque caminos llamando a un callback en cada nodo o arista para filtrar si el camino puede pasar por el nodo o arista. Ante la ausencia de encontrar tal algoritmo, decidimos crear un algoritmo propio para nuestro problema, para ello planteamos dos alternativas:

- Hacer una función iterativa que evaluase todos los nodos y sus vecinos y fuera viendo los distintos caminos factibles sin crear ciclos.

- Hacer una función recursiva que fuese evaluando los nodos actuales y vecinos y vaya descartando los nodos que no llevarán al destino.

4.6 Recursividad vs Iteración

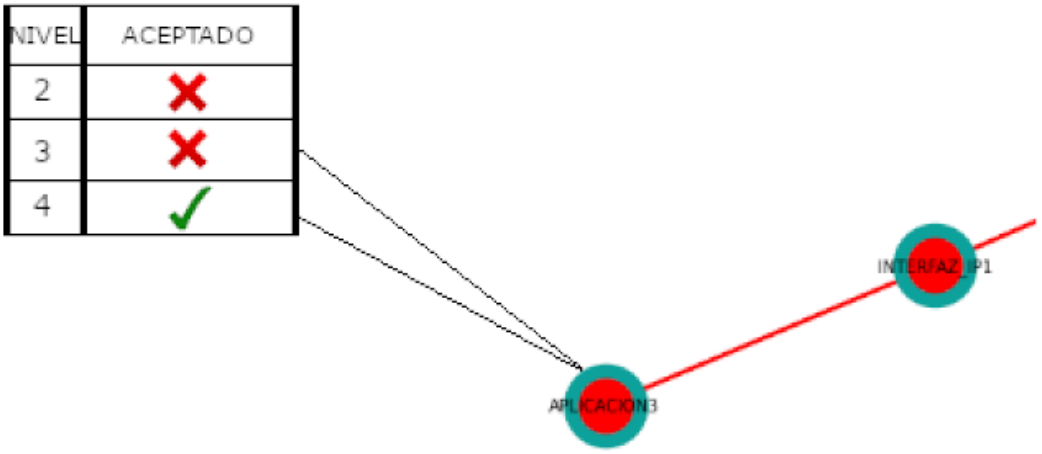
4.6.1 Recursividad

Lo primero que se nos viene a la mente a la hora de pensar de hacer algún código con recursividad se trata de la pila de llamadas del sistema, esta nos permite guardar el estado actual de la exploración y cada llamada guarda su estado en la pila mientras explora más profundamente. A la hora de crear la función recursiva iba a ser un código más simple y fácil de explicar ya que nuestro problema se puede dividir en subproblemas de forma fácil. El único problema que se nos viene a la mente de forma sencilla es que este método esta limitado por la profundidad de la pila, esto nos puede dar problemas para grafos muy grandes o con caminos muy largos.

4.6.2 Iteración

Un enfoque iterativo puede ser más difícil ya que es necesario el uso manual de la pila de llamadas para simular la recursión, pero esto nos puede dar cierta ventaja si queremos optimizar el uso de la memoria para asi evitar desbordamiento de la pila o estructura que estemos usando. Tenemos menos riesgo ya que como no tenemos un límite en lo grande que es la pila de llamadas, podemos operar sobre cualquier tipo de grafo dando igual como de grande sea.

Tras optar por la opción de un enfoque recursivo se le dio al algoritmo un enfoque nuevo donde habría que añadirle a cada nodo un atributo de nivel para por ver si dicho tipo de nodo dentro de la red puede dejar pasar paquetes de nivel X. Trabajaremos sobre el nivel 2, 3 y 4 de la capa OSI para poder determinar los distintos caminos del nivel X.



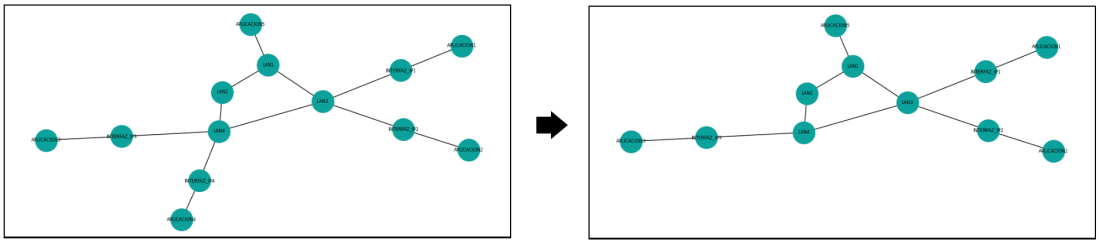
Tras definir una función capaz de encontrar caminos en un grafo, era necesario preguntarle al usuario que nivel en concreto quiere buscar ya que se nos planteo la idea de poder obtener caminos de nivel X si obteniamos los caminos de nivel direacmente inferiores. Esta idea se vio fácilmente descartada ya que no era factible el calcular los caminos inferiores para obtener los de niveles

superiores, tendriamos al final más iteraciones del algoritmo si simplemente calculamos los caminos del nivel en concreto que queremos.

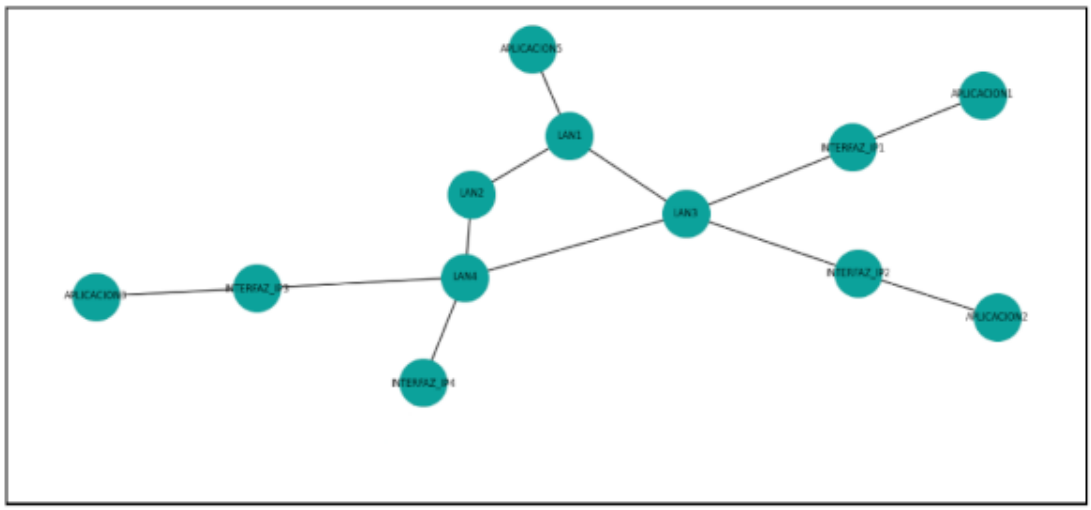
Esto dio paso a otro tipo de idea, el concepto de área de conectividad de una red. Se define con la zona dentro de una red donde todos los nodos se pueden comunicar entre sí. Ya que como estamos trabajando sobre los distintos niveles de la torre OSI de la red, pareció una gran idea el hecho de obtener las áreas de conectividad de la red que estamos operando.

Para poder hallarlas se debe operar sobre el grafo completo y hallar los nodos que no son gateway del nivel que estamos buscando, eliminamos del grafo dichos nodos que no son gateway para dicho nivel, observamos que subgrafos conexos quedan y a cada subgrafo resultante que nos quedó se trata de cada zona de conectividad del grafo pero esto no sería la zona de conectividad al completo, necesitamos añadir a los subgrafos resultantes los nodos que hacian gateway y hemos quitado ya que sino no sería la zona de conectividad completa.

Se podria pensar que al final la union de todos los subgrafos resultantes debe dar el grafo inicial pero no siempre se puede dar ese caso. Si tenemos dos nodos que son vecinos y no son gateway del nivel X, si uno de ellos tiene grado 1 entonces ese nodo no se verá reflejado en la unión de grafos.



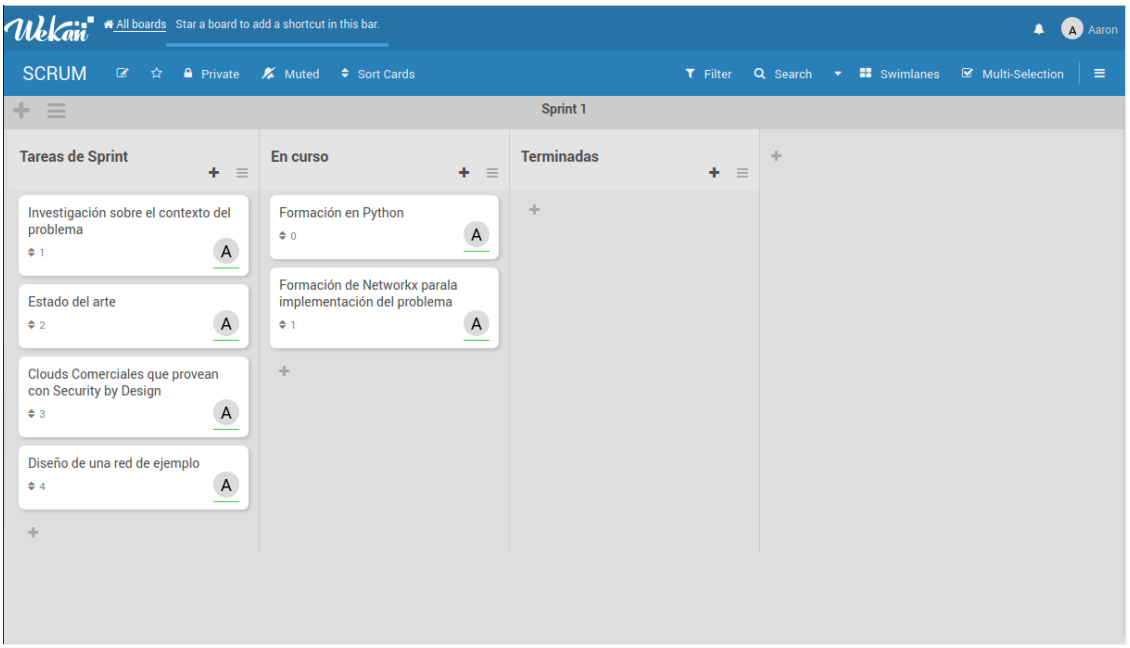
Como vemos en la imagen, si por ejemplo obtuvieramos la zona de conectividad de nivel 2 de dicho grafo y los nodos INTERFAZ_IP4 y APLICACION4 no permitieran en su tabla de niveles el paso del tráfico de paquetes a nivel 2, a la hora de calcular la área de conectividad, añadiriamos el nodo INTERFAZ_IP4 que era el nodo no gateway conectado al subgrafo resultante pero el nodo APLICACION4 no se vería reflejado en la área de conectividad.



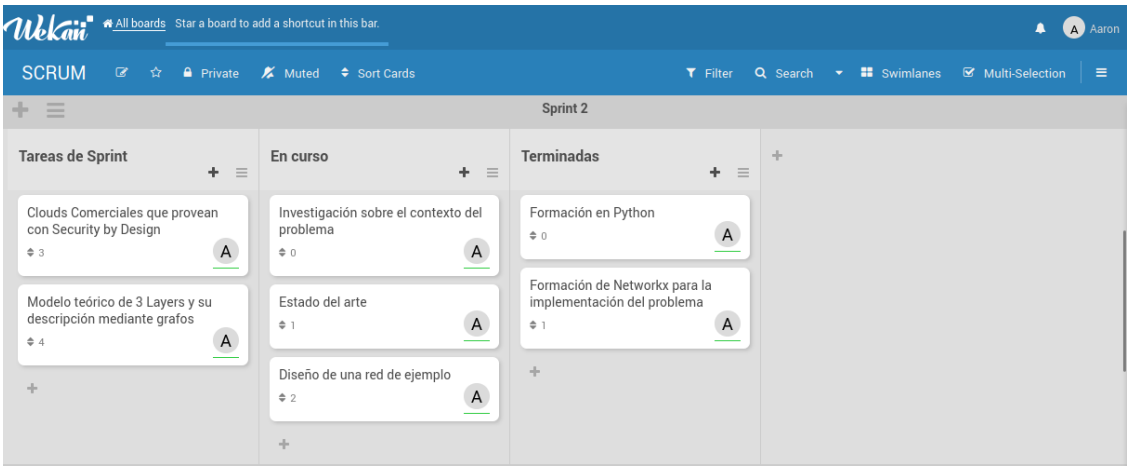
Teniendo entendido estos temas ahora vamos a pasar a los atributos extras que les hemos añadido a los nodos y que nos pueden resultar útiles, la dirección ip por ejemplo. Este nos permite darle un "nombre" a nivel3 a las conexiones entre nodos aplicacion y su interfaz_ip. Podemos añadir a los nodos que sean router(que tenga al atributo ip_forward=True) unas lista ACL para poder restringir ciertas conexiones no deseadas dentro de nuestra red. Las ACL es un elemento fundamental dentro de una red...

5 Scrum

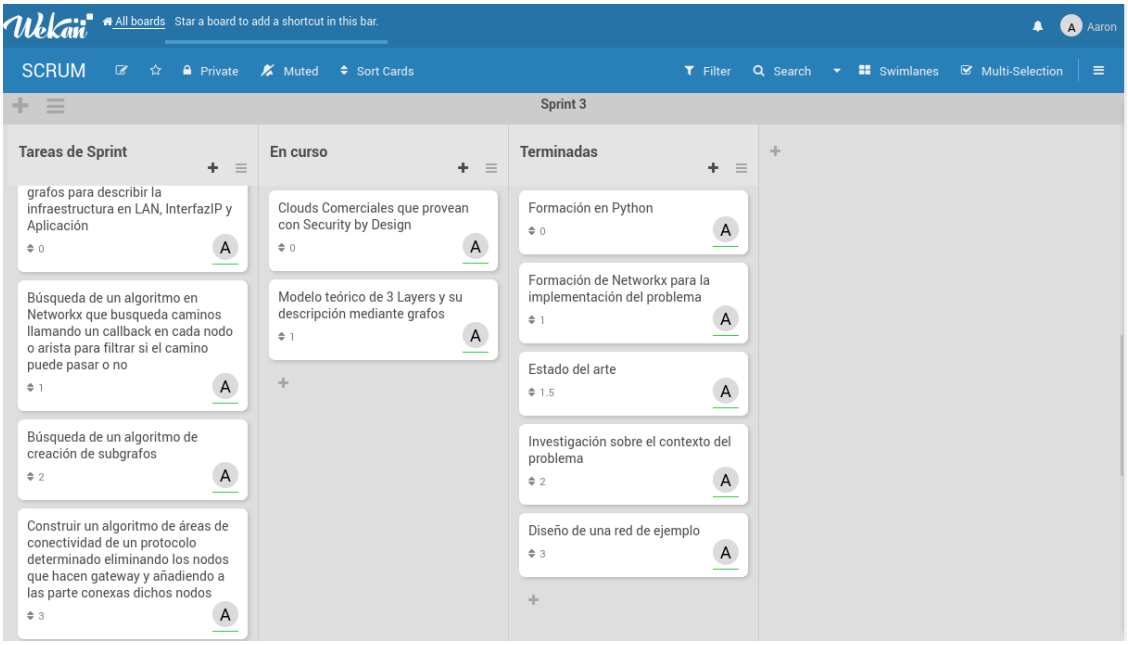
5.1 Sprint 1



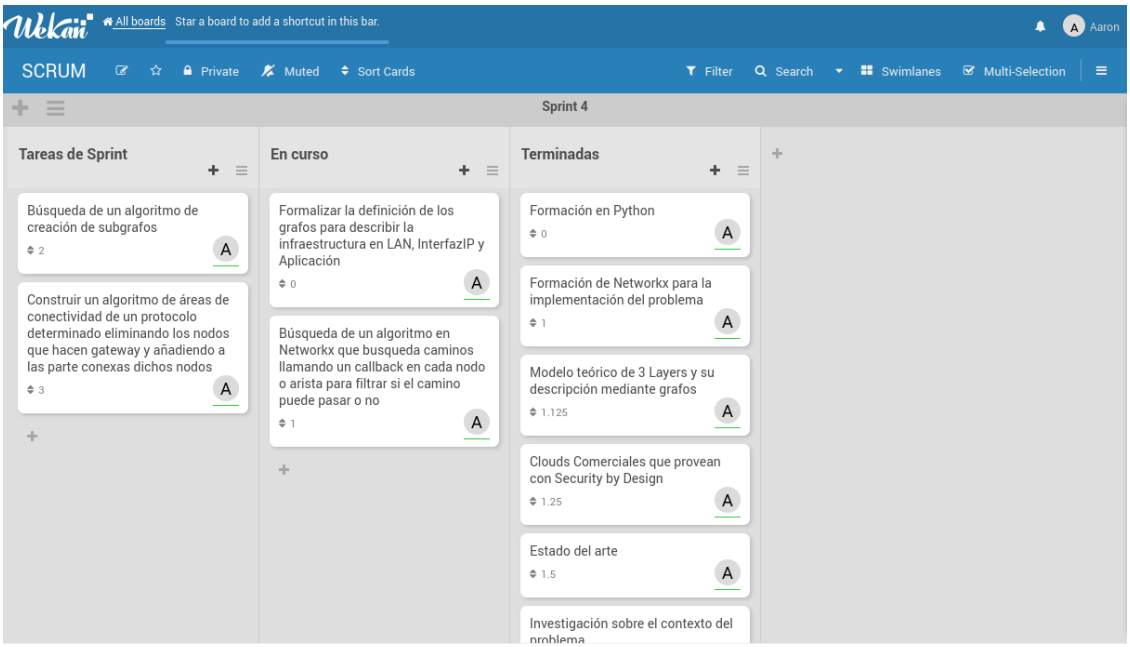
5.2 Sprint 2



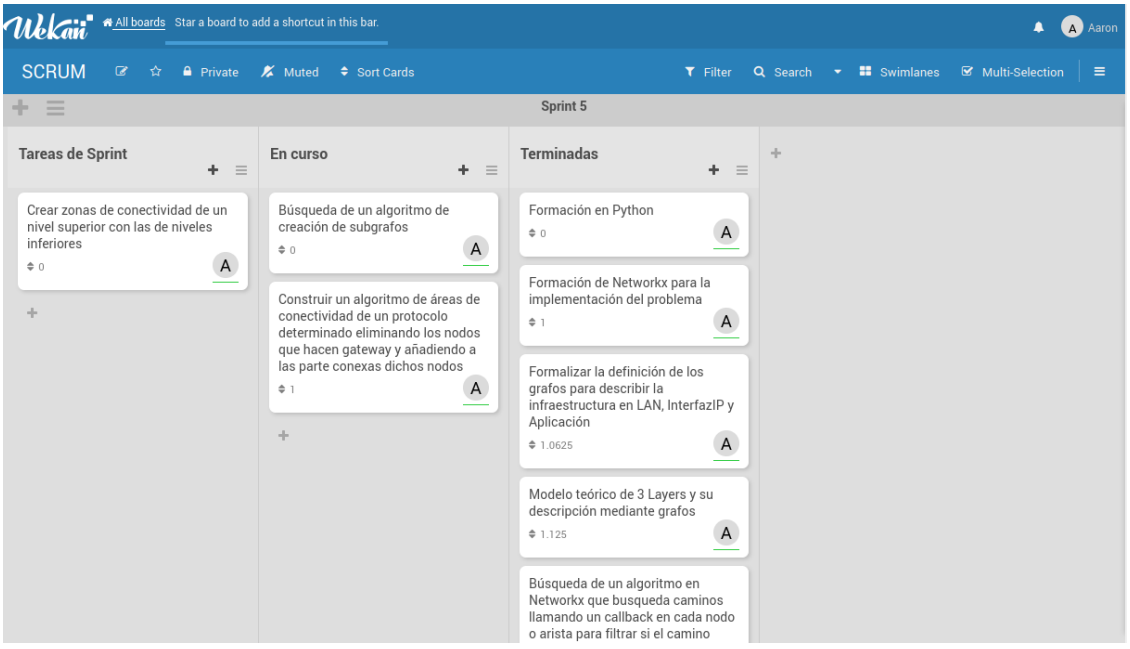
5.3 Sprint 3



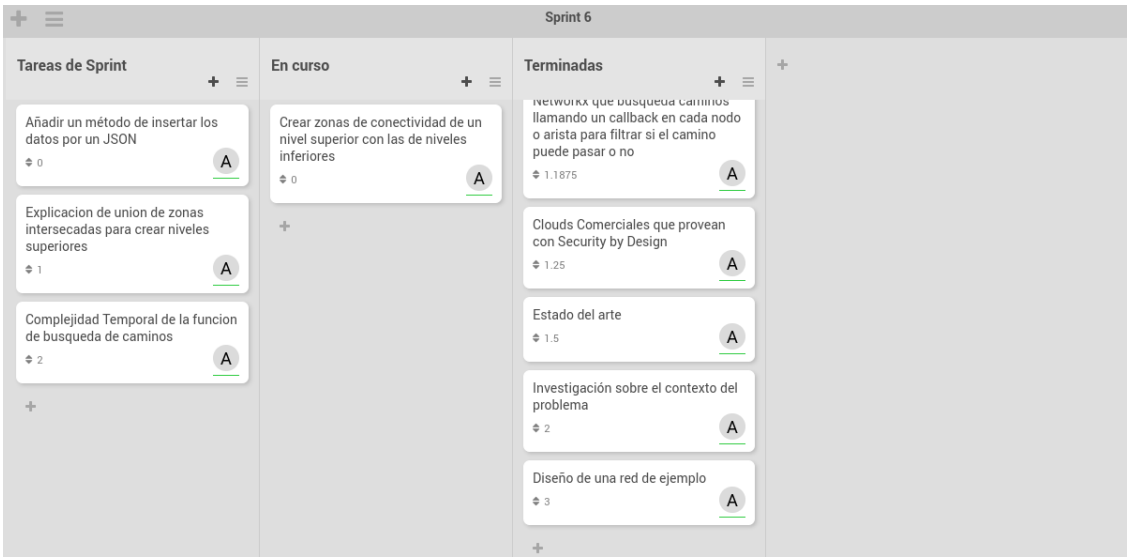
5.4 Sprint 4



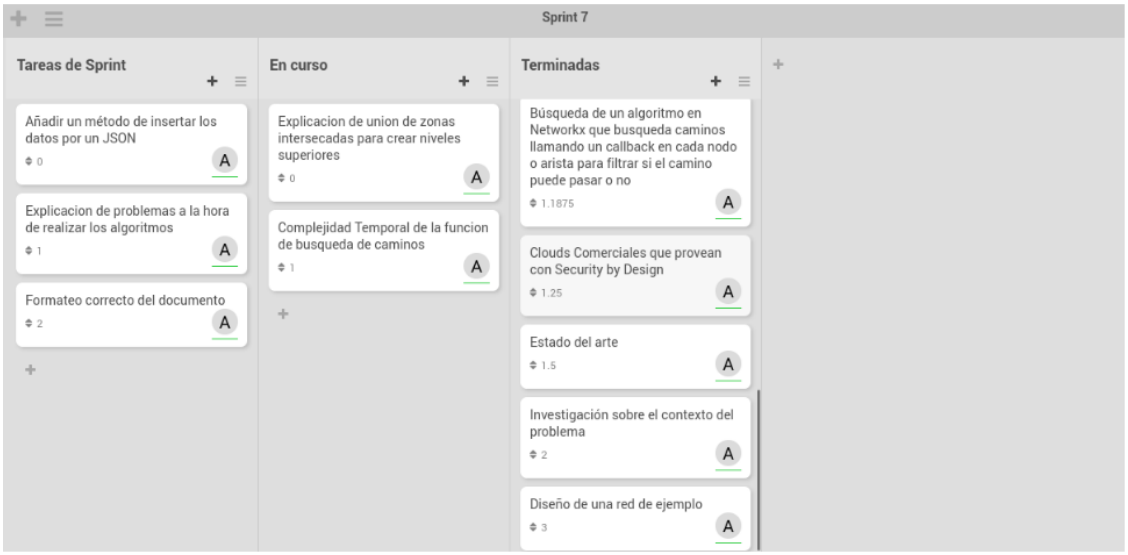
5.5 Sprint 5



5.6 Sprint 6



5.7 Sprint 7



6 Conlusiones y Future Work

6.1 Seguridad

Como hemos podido observar en los últimos tiempos la seguridad se ha vuelto uno de los pilares principales a la hora de mantener nuestro sistema privado y sin filtraciones de datos. En unos tiempos donde la tecnología avanza de forma tan continua y veloz también lo hacen los métodos de penetración en sistemas, los sistemas de suplantación de identidad, ataques de inundación, etc. Para poder asegurar la información, se debe contar con tecnologías avanzadas y prácticas que hagan uso del autenticación multifactor y cifrado de datos. Es esencial seguir aprendiendo en el ámbito

de la ciberseguridad para poder mantenernos fuera de riesgos en un entorno donde cada vez todo se vuelve más complicado.

6.2 Seguridad por Diseño

Hoy en día este es un concepto que se tiene poco en cuenta cuando se va a realizar el diseño de una red para una infraestructura pero este elemento es sumamente importante a la hora de dar seguridad a nuestra red y quitarnos trabajos posteriores que al final van a resumirse en tiempo y dinero. La seguridad por diseño se acopla desde su fase inicial del desarollo del diseño de sistemas y aplicaciones, obteniendo una garantia de qe la seguridad dentro de esta sea un componente fundamental y no un añadido posterior.

6.3 Lineas Futuras

6.3.1 Terraformer

Como inicio para el planteamiento del problema que vemos en este TFG de extrapolar una red con sus nodos y conexiones a un grafo, necesitabamos un estructura de datos o clase para poder representarlas. Usamos durante este TFG la libreria de networkX de python donde podemos usar todos los elementos de los nodos y aristas que necesitemos, pero se planteó como una buena idea la de insertar código de terraformer para poder realizar los nodos y aristas a medida para nuestro problema. Un tipo nodo que tenga las características ya de base de un nodo de red y unas aristas que tengas las características de las conexiones físicas de una red. También sería necesario la creación de un traductor capaz de interpretar nuestro código en terraformer y poder sacarlo en un fichero que podamos interpretarlo tal que como un idl.

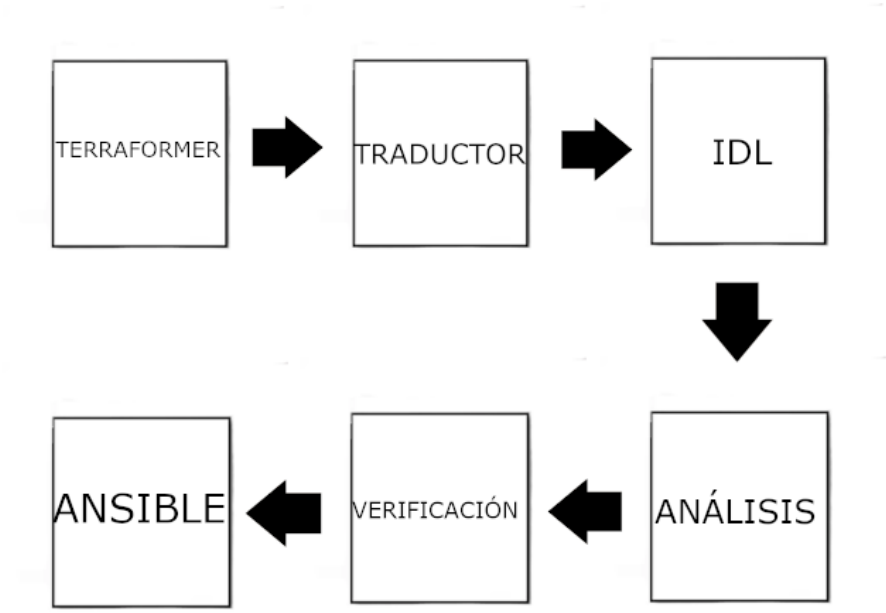
6.3.2 ACLs

Dentro de los nodos cada uno de ellos digamos que tiene una “tabla de niveles” donde puedes ver que nivel de la torre OSI es capaz de pasar dicho nodo, pero se planteó darle una utilidad más al algoritmo y que sea capaz de buscar caminos posibles entre dos nodos realizando restricciones de

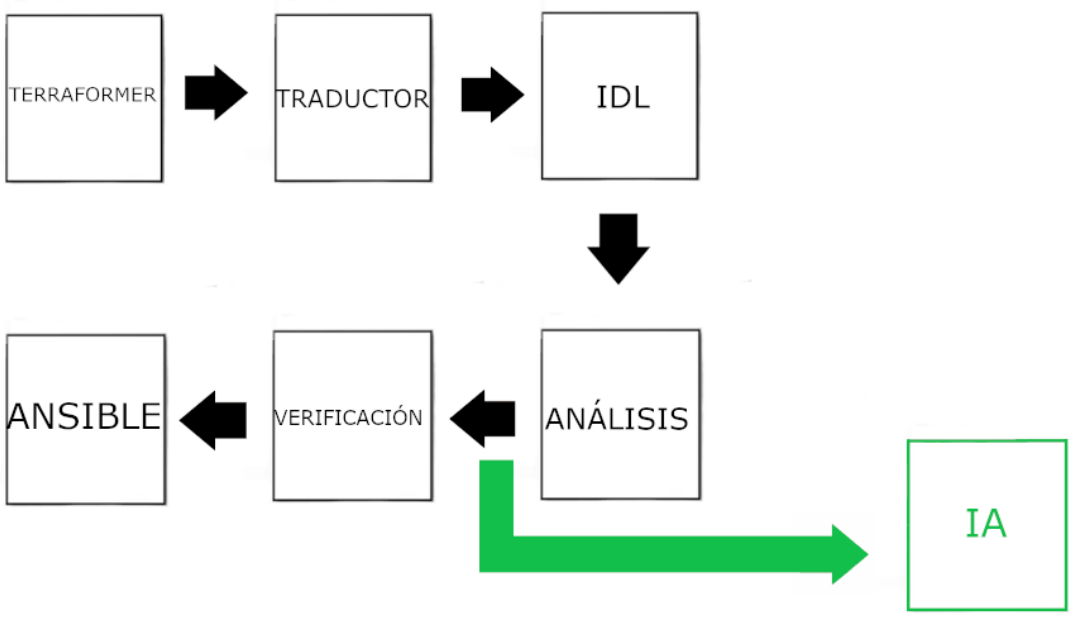
nivel 3 dentro de los nodos router, creando asi reglas de ACL para poder permitir el paso o no hacia cierto destino. Este elemento haría que nuestro problema se viera más realista dentro de una red "enterprise" donde los nodos router tienen ACL que permiten el paso hacia ciertas direcciones y eliminan todo lo demás. Esto daría cavidad a que atributos esenciales como la dirección ip y la MAC pudierán ser parte de los nodos que añadimos al grafo.

6.3.3 Archivo de Accesos complementado con Ansible

A la hora de realizar el algoritmo nos da un mapa de caminos del nivel X que hayamos escogido dentro de nuestra red, esto podría ser exportado a un archivo de accesos de los distintos nodos implicados en los caminos, si este archivo de accesos de lo diésemos como input de entrada a Ansible sería capaz de verificar las configuraciones que estan erróneas dentro de nuestro sistema y nos cedería de forma automática una configuración óptima para que nuestra red sea capaz de eliminar los caminos que no queríamos o no teniamos conocimiento que eran desacertados.



6.3.4 IA



7 Bibliografía

https://iberasync.es/arquitectura-cliente-servidor-modelo-de-3-capas/

https://prezi.com/xs3gghmtkvyd/arquitectura-de-las-aplicaciones-web-en-el-modelo-de-2-capas/

https://www.ciscopress.com/articles/article.asp?p=2202410&seqNum=4

https://www.tutorialspoint.com/2-tier-and-3-tier-architecture-in-networking

https://ccnatutorials.in/network-fundamentals-ccna-200-301/2-tier-and-3-tier-architecture-in-networking/