Tarea 3: Consultar Fabricantes a una API

Alejandro Dinamarca Cáceres, alejandro.dinamarca@alumnos.uv.cl

Thean Orlandi Guzman, thean.orlandi@alumnos.uv.cl

1. Introducción

Se propone desarrollar una herramienta de línea de comandos en Python denominada “OUILookup”, con el objetivo de consultar el fabricante de una tarjeta de red utilizando su direccion MAC o dirección IP. Se utilizará la tabla ARP del dispositivo que ejecute el programa, y además se proporcionará una salida detallada sobre el fabricante dada una dirección MAC, incorporando mediciones de latencia entre la aplicacion y la API REST asociada.

1.1. Materiales y métodos

1.2. Materiales

Se utilizó un sistema basado en MAC OS, que a su vez está basado en Unix.

1.3. Herramientas

1.3.1. Python

Para la implementación de la herramienta OUI Lookup Tool se utilizó el paradigma de la programación funcional utilizando el esqueleto proporcionado en GitHub (https://github.com/Stry12/Tarea02-Redes-De-Computadoras).

1.3.2. iTerm2

El programa iTerm2 es un emulador del terminal, específico para MacOS, con el cual se ejecutará la aplicación desarrollada.

1.3.3. API de MacLookup

A modo de obtener la información del fabricante dada la dirección MAC de un dispositivo, se utilizó la API de MacLookup.

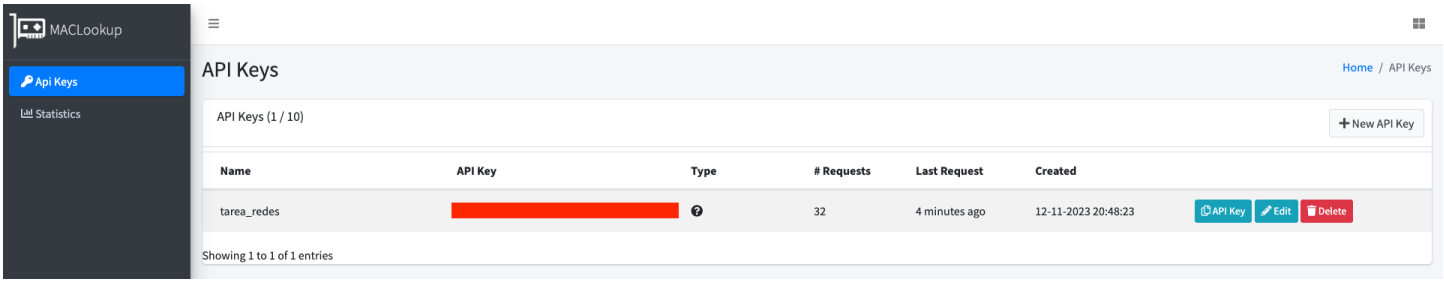

Figura 1. Vista de interfaz de MacLookup para obtener credenciales para realizar peticiones a su API.

2. Resultados

2.1. OUILookup

De acuerdo al intérprete de comandos tipo BASH:

2.1.1. Ejemplo de uso con parámetro --ip

Para comprobar una IP específica, es posible utilizar el siguiente comando, el cuál dependiendo de si pertenece o no al área local (que a diferencia de la versión anterior del programa, se valida internamente por una función verificando el NETID) proporcionará como salida los datos correspondientes o un mensaje de que la IP está fuera del área local:

```
alejandrodinamarcacaceres@arthur tarea3 % python3.10 OUILookup.py --ip 192.168.1.1
MAC address: 00:15:51:86:eb:5d
Fabricante: Radiopulse Inc.

alejandrodinamarcacaceres@arthur tarea3 %
```

Figura 2. Terminal con parámetro –ip y argumento de IP en red local ya en tabla ARP.

```
alejandrodinamarcacaceres@arthur tarea3 % python3.10 OUILookup.py --ip 192.168.1.9
MAC address: Not found
Fabricante: Not found

alejandrodinamarcacaceres@arthur tarea3 %
```

Figura 3. Terminal con parámetro --ip y argumento de IP en red local, pero no aún en tabla ARP.

```
alejandrodinamarcacaceres@arthur tarea3 % python3.10 OUILookup.py --ip 192.162.1.9
Error: ip is outside the host network

alejandrodinamarcacaceres@arthur tarea3 %
```

Figura 4. Terminal con parámetro --ip y argumento de IP fuera de la red local (dar cuenta que el filtro es dinámico, por lo que se detecta automáticamente el NETID).

```
alejandrodinamarcacaceres@arthur tarea3 % python3.10 OUILookup.py -i 192.168.1.2
MAC address: 00:15:5d:86:eb:5f
Fabricante: Microsoft Corporation

alejandrodinamarcacaceres@arthur tarea3 %
```

Figura 5. Terminal con atajo -i.

2.1.2. Ejemplo de uso con parámetro --mac

Para comprobar una MAC, se debe utilizar el parámetro –mac con un argumento de MAC válida, la que será validada por la API de MacLookup para obtener la información del fabricante.

El tiempo de ejecución variará por dispositivo y red empleada. A continuación, se adjuntan dos ejecuciones, una en un ordenador Windows:

```
C:\Users\IIfie\Desktop\tarea3>python3 OUILookup.py --mac 00:15:51:86:eb:5d
MAC address: 00:15:51:86:eb:5d
Fabricante: RadioPulse Inc.
Tiempo de respuesta: 1054.502ms
```

Figura 6. Terminal con parámetro --mac y argumento de MAC conocida en dispositivo Windows.

Y otra en MAC OS:

```
alejandrodinamarcacaceres@arthur tarea3 % python3.10 OUILookup.py --mac 00:15:51:86:eb:5d
MAC address: 00:15:51:86:eb:5d
Fabricante: RadioPulse Inc.
Tiempo de respuesta: 759.0ms

alejandrodinamarcacaceres@arthur tarea3 %
```

Figura 7. Terminal con parámetro --mac y argumento de MAC conocida en dispositivo MAC OS.

Por lo que, a modo de resumen y tras 10 ejecuciones en cada ordenador, los tiempos de ejecución por integrante fueron:

Integrante	Tiempo de respuesta max. (ms)	Tiempo de respuesta avg. (ms)
Alejandro Dinamarca	852.2	791.6
Thean Orlandi	1153.0	978.3

Tabla 1. Tiempos de respuesta por integrante.

Además, se puede obtener el fabricante dada una dirección MAC:

```
alejandrodinamarcacaceres@arthur tarea3 % python3.10 OUILookup.py --mac 00:15:51:as:eb:5d
MAC address: 00:15:51:as:eb:5d
Fabricante: Not found
Tiempo de respuesta: 627.0ms

alejandrodinamarcacaceres@arthur tarea3 %
```

Figura 8. Terminal con parámetro --mac y argumento de MAC desconocida.

2.1.3. Ejemplo de uso con parámetro --arp

Para obtener la tabla ARP:

```
alejandrodinamarcacaceres@arthur tarea3 % python3.10 OUILookup.py --arp
IP              /       MAC                     /       VENDOR
192.168.1.0     /       00:15:5d:86:eb:5f       /       Microsoft Corporation
192.168.1.30    /       00:00:00:00:00:01       /       Xerox Corporation
192.168.1.1     /       00:15:51:86:eb:5d       /       Radiopulse Inc.
192.168.1.2     /       00:15:5d:86:eb:5f       /       Microsoft Corporation
192.162.1.3     /       00:15:5d:86:eb:5d       /       Microsoft Corporation
192.168.1.4     /       00:10:5d:82:ec:5f       /       Draeger Medical
192.168.1.5     /       00:15:51:86:eb:5e       /       Radiopulse Inc.
192.168.1.6     /       00:12:51:86:eb:5e       /       Silink
```

Figura 9. Terminal con parámetro --arp.

2.1.4. Ejemplo de uso con parámetro --help

Para obtener ayuda:

```
alejandrodinamarcacaceres@arthur tarea3 % python3.10 OUILookup.py --help
Use: ./OUILookup --ip <IP> | --mac <IP> | --arp | [--help]
--ip : IP del host a consultar.
--mac: MAC a consultar. P.e. aa:bb:cc:00:00:00.
--arp: muestra los fabricantes de los host
disponibles en la tabla arp.
--help: muestra este mensaje y termina.
```

Figura 10. Terminal con parámetro --help.

3. Implementación

A continuación, se expondrá el código de fuente de la implementación, el cuál será dividido en secciones para ser explicado. Cabe destacar que todas las secciones, en conjunto, conforman el programa.
Para ver el programa en su totalidad, ir al enlace al repositorio.

3.1. Variables globales

A modo de facilitar la modificación del “boilerplate code”, se colocaron al principio del código. Además, se implementó el cálculo automático del NET-ID mediante la separación de las cadenas en octetos válidos, y el respectivo cálculo mediante la operación AND.

```
import getopt
import sys

# Redes
RED_HOST = "192.168.1.30"
MASK = "255.255.255.0"
# Operación entre RED_HOST y MASK (AND por cada octeto)
NET_ID = ".".join([str(i & j) for i, j in zip(map(int, RED_HOST.split(".")), map(int,
MASK.split(".")))])
# Strings de respuesta
MISMA_RED = "MAC address: {mac}\nFabricante: {manufacturer}\n"
OTRA_RED = "Error: ip is outside the host network\n"
MAC_EN_BASE_DE_DATOS = "MAC address: {mac}\nFabricante: {manufacturer}\n"
AYUDA = """Use: ./OUILookup --ip <IP> | --mac <IP> | --arp | [--help]
--ip : IP del host a consultar.
--mac: MAC a consultar. P.e. aa:bb:cc:00:00:00.
--arp: muestra los fabricantes de los host
disponibles en la tabla arp.
--help: muestra este mensaje y termina."""
```

Figura 11. Declaración de variables globales.

3.2. Función para validar IP en red local

A diferencia de la versión anterior del programa, ahora la validación es efectiva y dinámica mediante la operación AND por cada octeto de la IP y la máscara de red.

En este caso, se separa las cadenas de caracteres (la IP y la máscara) por puntos, obteniéndose dos listas que serán mapeadas convirtiendo cada elemento a entero, para posteriormente ejecutar la operación AND entre cada octeto de la IP y la máscara. De este modo, se obtiene el NETID de la IP pasada como argumento para luego compararla con el NETID original, devolviendo un booleano que indicará si la IP pertenece o no al área local.

```python
def pertenece_a_red(ip: str) -> bool:
    """
    Verifica si una IP pertenece a la red del host.

    Parámetros:
        ip (str): IP a verificar.

    Retorna:
        True si la IP pertenece a la red del host, False en caso contrario.
    """
    # Operación entre IP y MASK (AND por cada octeto)
    ip_net_id = ".".join([str(i & j) for i, j in zip(map(int, ip.split(".")), map(int,
MASK.split(".")))])
    return ip_net_id == NET_ID
```

Figura 12. Función para validar IP en red local.

3.3. Funciones para obtener IP, MAC, y datos en general de tabla ARP

A continuación, se expondrán las funciones que obtienen una IP, MAC o fabricante de una tarjeta de red (a través de la tabla ARP del dispositivo), por lo que, las funciones hacen uso de la obtención de datos de la tabla ARP actualizada en el tiempo de ejecución:

```python
# Función para obtener los datos de fabricación de una tarjeta de red por IP
def obtener_datos_por_ip(ip: str) -> str:
    """
    Obtiene los datos de fabricación de una tarjeta de red por IP.

    Parámetros:
        ip : IP del host a consultar.

    Retorna:
        Datos de fabricación de la tarjeta de red o "Not found" si no se encuentra.
    """
    mac = obtener_mac_por_ip(ip)
    if mac == "Not found":
        return "Not found"
    return obtener_fabricante_mac(mac)["data"]
```

Figura 13. Función para obtener datos por IP mediante uso de la tabla ARP (que se utiliza en la función obtener_mac_por_ip)

3.4. Función para obtener tabla ARP

La función para obtener la tabla ARP hace uso de las funciones previamente definidas, como también, del formatear correctamente la cadena para que tenga estructura de tabla:

```
# Función para obtener la tabla ARP
def obtener_tabla_arp():
    """
    Obtiene la tabla ARP.

    Retorna:
        Texto con la tabla ARP.
    """
    tabla_arp = f"IP\t\t/\tMAC\t\t\t/\tVENDOR\n"
    # Solo nos quedamos con las filas que tienen 4 elementos (IP, tipo, dirección MAC y tipo de
interfaz)
    tabla_arp_so = [fila.split(" ")[0:4] for fila in subprocess.check_output(["arp", "-
a"]).decode("latin-1").split("\n")]
    tabla_arp_so = [fila for fila in tabla_arp_so if len(fila) == 4]
    for fila in tabla_arp_so:
        # Si la MAC no está vacía, se agrega a la tabla (ya que si está vacía, no se conoce la MAC y
por ende aún no se ha hecho ARP)
        if valida_mac(fila[3]):
            ip = fila[1].removeprefix('(').removesuffix(')')
            mac = fila[3]
            fabricante = obtener_fabricante_mac(mac)["data"]
            tabla_arp += f"{ip}\t/\t{mac}\t/\t{fabricante}\n"
    return tabla_arp
```

Figura 14. Función para obtener tabla ARP. Destacar cómo se utilizan funciones previamente definidas.

3.5. Función para obtener datos del fabricante a partir de una dirección MAC

Además de emular datos en la red local, es posible obtener datos del fabricante dada una dirección MAC. La siguiente función envía una petición simple a la API de MacLookup, que verificará si la MAC es válida, y posteriormente buscará en su base de datos centralizada la dirección devolviendo el fabricante, el que será procesada por la función. En este caso, se utilizó la librería por defecto de Python, http, a modo que el usuario solo deba instalar una distribución de Python3 en su dispositivo (así se evita instalar un paquete como requests).

```python
def obtener_fabricante_mac(mac: str) -> str:
    """
    Obtiene el fabricante de una tarjeta de red por MAC.

    Parámetros:
        mac (str): MAC del host a consultar.

    Retorna:
        Fabricante de la tarjeta de red o "Not found" si no se encuentra.
    """
    from json import loads
    from http import client
    start = time.time()
    conn = client.HTTPSConnection("api.maclookup.app")
    conn.request("GET", "/v2/macs/" + mac + f"?apiKey={obtiene_api_key()['apiKey']}")
    response = conn.getresponse()
    end = time.time()
    time_elapsed = round((end - start)*1000, 3)
    if response.status == 200:
        data: dict = loads(response.read())
        if data["found"] == False:
            return {"data": "Not found", "time(ms)": time_elapsed}
        return {"data": data["company"], "time(ms)": time_elapsed}
    else:
        return {"data": "Not found", "time(ms)": time_elapsed}
```

Figura 15. Función para obtener fabricante a partir de la API de MacLookup.

3.6. Función principal main

Finalmente, en la función main se ejecutan de acuerdo con los parámetros pasados por terminal las funciones correspondientes, por lo que es un condicional que, además, se encarga del manejo de excepciones para brindar ayuda al usuario.

```
def main(argv):

    try:
        opts, args = getopt.getopt(argv, "i:", ["ip=", "mac=", "arp", "help"])

    except getopt.GetoptError:
        #Modificar para coincidir con tarea
        print("Use: python OUILookup.py --ip <IP> | --Arg2 <Arg2> | --Arg3 | [--help] \n --ip : IP del
host a consultar. \n --Arg2:  \n --Atg3: \n --help:")
        sys.exit(2)

    for opt, arg in opts:
        if opt in ("-i", "--ip"):
            if pertenece_a_red(arg):
                print(MISMA_RED.format(mac=obtener_mac_por_ip(arg),
manufacturer=obtener_datos_por_ip(arg) if obtener_datos_por_ip(arg) != "" else "Not found"))
            else:
                print(OTRA_RED)
        elif opt in ("--mac"):
            vendor, time = obtener_fabricante_mac(arg).values()
            print(MAC_EN_BASE_DE_DATOS.format(mac=arg, manufacturer=vendor + f"\nTiempo de respuesta:
{time}ms"))
        elif opt in ("--arp"):
            print(obtener_tabla_arp())
        elif opt in ("--help"):
            print(AYUDA)
        else:
            print("Debe proporcionar una opción válida (-i, -m o -a).")
            sys.exit(2)
        sys.exit(0)
```

Figura 16. Función principal.

3.7. Esquema general de resolución (diagrama de flujo)

El siguiente diagrama de flujo proporciona una visión macro del flujo del programa:

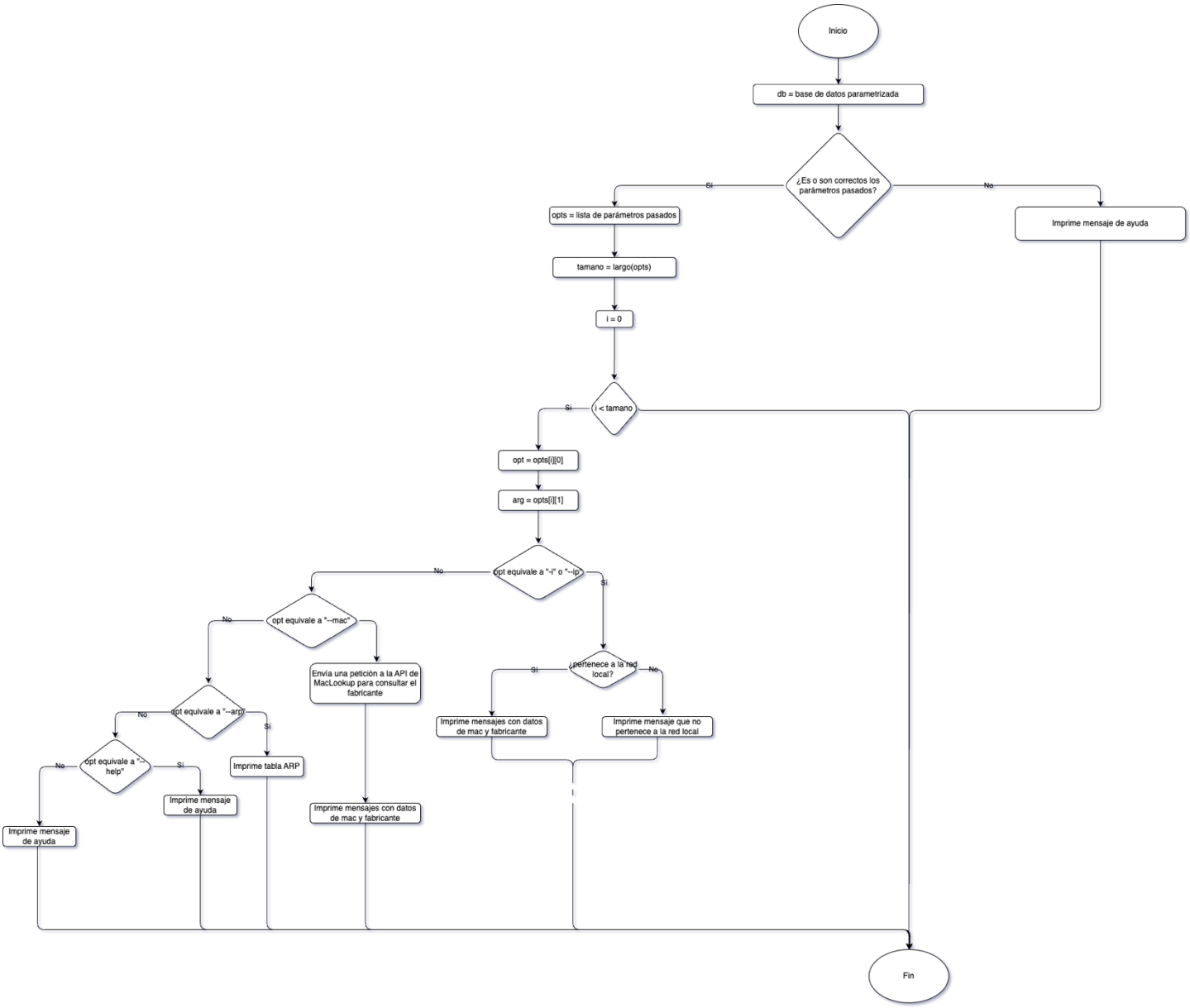

Figura 17. Diagrama de flujo.

3.8. Enlace al repositorio en Github

El repositorio está en el siguiente enlace: https://github.com/adinamarca/Tarea03-Dinamarca-Caceres-Alejandro.

4. Preguntas

4.1. ¿Qué es REST? ¿Qué es una API?

REST o Transferencia de Estado Representacional es un enfoque para crear servicios web que utiliza estándares web y opera el protocolo HTTP. Se basa en la idea de tratar a los recursos (datos o servicios) como objetos accesibles a través de URLs, y se utilizan operaciones estándar como GET, POST, PUT y DELETE para realizar acciones en estos recursos.

Por otro lado, una API o Interfaz de Programación de Aplicaciones es un conjunto de reglas y herramientas que permite que diferentes aplicaciones se comuniquen entre sí. Proporciona un conjunto de funciones y métodos para que los desarrolladores accedan a las características o datos de una aplicación, sistema operativo o servicio. En el contexto de servicios web, una API define como otros programas pueden interactuar con un software especifico.

4.2. ¿Cómo se relaciona el protocolo HTTP con las API REST y cuál es su función en la comunicación entre clientes y servidores?

HTTP es el protocolo que permite que los clientes (como el navegador o una aplicación) se comuniquen con los servidores que almacenan la información. Las API REST usan HTTP para realizar acciones específicas en los datos, como obtener información (GET), agregar nueva información (POST), actualizar (PUT), o eliminar (DELETE). La información se identifica mediante URLs, y el protocolo HTTP asegura que la comunicación entre el cliente y el servidor sea efectiva y estandarizada.

4.3. ¿Qué papel juega la dirección IP en el acceso a recursos a través de una API REST?

La dirección IP es como la dirección del dispositivo en la red. Cuando se usa una API REST, el dispositivo se comunica con el servidor usando su dirección IP. Esto asegura que las solicitudes y respuestas lleguen a los lugares correctos, permitiendo acceder a los recursos que se necesitan. En resumen, la dirección IP ayuda a la comunicación entre un dispositivo y el servidor a través de la API REST.

4.4. ¿Por qué es importante considerar la latencia de red y el ancho de banda? ¿Cómo afectan estos factores al rendimiento de la API?

La latencia de red es como la velocidad de entrega de un mensaje, y el ancho de banda es cuanta información se puede enviar a la vez. Si la latencia es alta, las respuestas de la API pueden tardar. Si el ancho de banda es bajo, puede haber demoras al enviar o recibir datos grandes. En resumen, una latencia baja y un buen ancho de banda hacen que las respuestas de la API sean rápidas y eficientes.

4.5. ¿Por qué el programa desarrollado utilizando API REST es más lenta su ejecución?

La lentitud de un programa con API REST puede deberse a cosas como la distancia entre el usuario y el servidor, la cantidad de datos que se están moviendo o como está escrito el código. No siempre es culpa de la API, a veces hay otras razones. Una buena implementación y una conexión rápida ayudan a que todo funcione más rápido.

4.6. ¿Cuál es la diferencia entre la dirección MAC (Media Access Control) y la dirección IP, y en que capa de la red se utilizan cada una de ellas?

La dirección MAC es como el numero único de serie de la tarjeta de red, y se usa para hablar con dispositivos cercanos en la red local, como en casa, por ejemplo. La dirección IP es como la dirección postal del dispositivo y se usa para hablar con dispositivos en redes más grandes, como en internet. La dirección MAC se usa para comunicarse localmente, y la dirección IP para comunicarse en redes más amplias.

4.7. ¿Cómo pueden las redes LAN (Local Area Networks) y WAN (Wide Area Networks) afectar la accesibilidad y la velocidad de respuesta de una API REST?

En una LAN, la accesibilidad y la velocidad suelen ser altas debido a la proximidad física y la infraestructura de red rápida. En una WAN, especialmente a través de internet, la accesibilidad y la velocidad pueden verse afectadas por la distancia, la latencia y la capacidad de la conexión. En resumen, la arquitectura de red ya sea LAN o WAN, pueden impactar la accesibilidad y la velocidad de respuesta de una API REST, siendo las LAN generalmente más rápidas y eficientes localmente, mientras que las WAN pueden presentar desafíos adicionales en términos de latencia y distancia.

4.8. ¿Qué es un enrutador y como se utiliza para dirigir el tráfico de datos? ¿Qué relación tiene esto con el enrutamiento de solicitudes en una API REST?

Un enrutador es como el guía de trafico de la red. Decide como dirigir los datos de un lugar a otro. En una API REST, el enrutador hace algo similar, asegurándose de que las solicitudes vayan a la parte correcta de la aplicación según la dirección solicitada. En resumen, un enrutador ayuda a que la información o solicitudes lleguen a donde deben.

4.9. ¿Cómo se asocian los puertos de red con servicios y aplicaciones específicas?

Los puertos de red están asociados con servicios y aplicaciones específicas mediante el concepto de “números de puerto”. Cada dispositivo en una red tiene 65535 puertos disponibles, y algunos de estos están reservados para servicios comunes. Los números de puerto indican a que servicio o aplicación deben ir los datos. Puertos bien conocidos (0-1023) están asignados a servicios comunes. Puertos registrados y dinámicos se utilizan para aplicaciones específicas. En resumen, los puertos de red permiten que los datos se dirijan a servicios y aplicaciones específicas. Cada número de puerto está asociado a un servicio partículas, facilitando la comunicación entre dispositivos en una red.

5. Discusión y conclusiones

Se implementó la herramienta “OUILookup” en Python, consolidando un programa versátil capaz de capturar parámetros desde el terminal de manera eficiente. A lo largo de este proceso, no sólo se ha creado una aplicación funcional, sino que también se ha profundizado en la comprensión de conceptos clave como el enrutamiento en redes, la importancia de los números de puerto y la influencia de la latencia en la eficiencia de una API REST. En conclusión, se logró un aprendizaje significativo en el desarrollo de aplicaciones de línea de comandos, el manejo de argumentos y la interacción con API REST para consultas específicas.