

Tarea 2: Consultar Fabricantes de tarjetas de red a través de una API

Fernanda Fuentes Pizarro, fernanda.fuentes@alumnos.uv.cl

Lorena Uribe Miranda, lorena.uribe@alumnos.uv.cl

1. Introducción

En esta tarea se implementó una herramienta llamada OUILookup, desarrollada en Python, que permite consultar el fabricante de una tarjeta de red a través de su dirección MAC. Las direcciones MAC son identificadores únicos asignados a interfaces de red, y conocer el fabricante de una tarjeta puede ser útil en diversas aplicaciones, como la gestión de redes o la identificación de dispositivos. Para la consulta de fabricantes, se utilizó una API REST pública que proporciona esta información a partir de una dirección MAC dada. La herramienta funciona en línea de comandos, permitiendo al usuario realizar consultas tanto de MACs específicas como de las direcciones MAC locales obtenidas de la tabla ARP.

2. Descripción del problema y diseño de la solución

El problema para resolver consistía en crear una herramienta que permita consultar el fabricante de una tarjeta de red dada su dirección MAC. El programa debía ser capaz de recibir una dirección MAC como entrada y devolver el fabricante asociado a dicha MAC. Para esto, se utilizó la API pública de `maclookup.app`. Además, el programa debía ser capaz de obtener las direcciones MAC de la tabla ARP local y consultar los fabricantes correspondientes.

Diseño del Programa

- `lookup_mac()`: Consulta el fabricante de una MAC utilizando la API.
- `lookup_arp()`: Obtiene las direcciones MAC de la tabla ARP y consulta sus fabricantes.
- `is_special_mac()`: Filtra las direcciones MAC que no son útiles para consultar, como las de broadcast o multicast.
- `test_macs()`: Función de prueba para verificar el correcto funcionamiento del programa con direcciones MAC específicas.
- El flujo del programa se maneja mediante la función `main()`, que procesa los argumentos de entrada usando `getopt` para determinar si se debe consultar una MAC específica o extraer la tabla ARP.

3. Diagrama de flujo

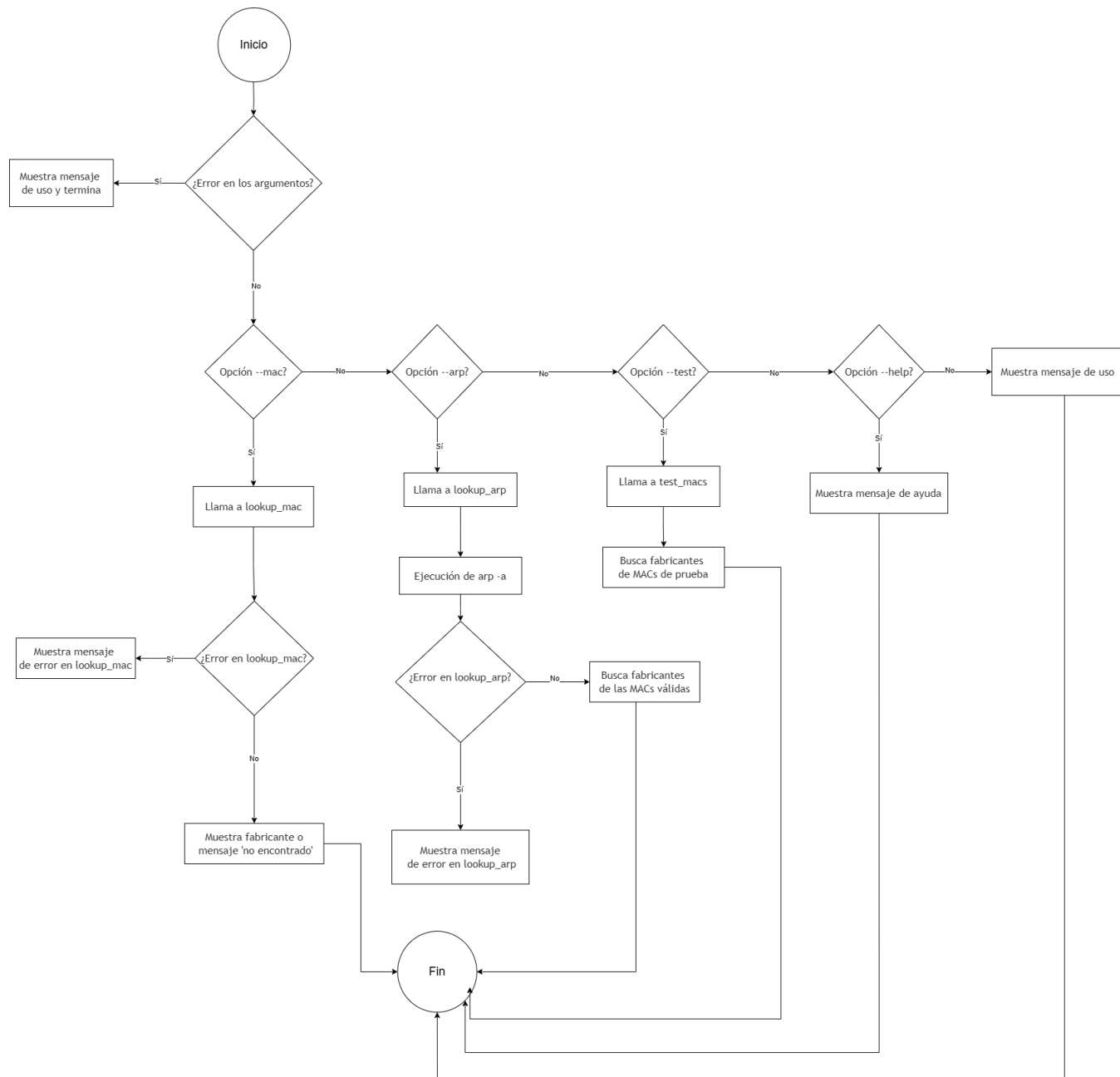


Ilustración 1

El diagrama de flujo representa de la **Ilustración 1** el proceso lógico del programa "OUILookup". Inicia verificando los argumentos de línea de comandos, que determinan qué función ejecutar: consultar un fabricante con una MAC específica, mostrar los fabricantes de las MACs en la tabla ARP, o realizar pruebas con MACs predefinidas. Si ocurre un error, se muestra un mensaje de ayuda o error, y si la consulta es exitosa, se muestra el fabricante o un mensaje de que no se encontró. Finaliza después de completar la acción correspondiente.

4. Implementación

Procesamiento de Argumentos de Línea de Comandos

El manejo de los parámetros ingresados por el usuario en la línea de comandos se realiza mediante la función `getopt` como se visualiza en la Imagen 1. Esto permite identificar qué acción ejecutar dependiendo del argumento recibido: consultar una MAC específica, consultar la tabla ARP o mostrar el mensaje de ayuda. La siguiente parte del código muestra cómo se procesan estos argumentos:

```
def main():
    try:
        # Procesamos los argumentos de entrada (como --mac, --arp, --help, --test)
        opts, args = getopt.getopt(sys.argv[1:], "", ["mac=", "arp", "help", "test"])
    except getopt.GetoptError as err:
        # Si hay un error en los argumentos, mostramos el uso correcto y salimos
        print(err)
        usage()
        sys.exit(2)

    mac = None
    arp = False
    test = False

    # Revisamos cada opción ingresada en la línea de comandos
    for o, a in opts:
        if o == "--mac":
            mac = a # Guardamos la MAC a consultar
        elif o == "--arp":
            arp = True # Indicamos que se debe consultar la tabla ARP
        elif o == "--test":
            test = True # Activamos el modo de prueba
        elif o == "--help":
            usage()
            sys.exit()

    # Si se ingresó una MAC, la consultamos
    if mac:
        lookup_mac(mac)
    # Si se pidió la tabla ARP, la obtenemos y consultamos las MACs
    elif arp:
        lookup_arp()
    # Si se activó el modo de prueba, ejecutamos las pruebas
    elif test:
        test_macs()
    else:
        # Si no se ingresó ninguna opción válida, mostramos cómo usar el programa
        usage()
```

Imagen 1

Consulta del Fabricante de una MAC

La función `lookup_mac()` que se muestra en la Imagen 2 consulta el fabricante de una dirección MAC utilizando la API de `maclookup.app`. Si la API responde con información válida, muestra el fabricante; si no, informa que la MAC no fue encontrada:

```
def lookup_mac(mac):  
    try:  
        # Se construye la URL para consultar la MAC  
        url = f"https://api.maclookup.app/v2/mac/{mac}"  
        # Se hace la solicitud GET a la API  
        response = requests.get(url)  
        # La respuesta se convierte en formato JSON  
        data = response.json()  
  
        # Si se encuentra el campo 'company' con valor, se muestra el fabricante  
        if 'company' in data and data['company']:  
            print(f"MAC address : {mac}")  
            print(f"Fabricante : {data['company']}")  
        else:  
            # Si no se encuentra el fabricante  
            print(f"MAC address : {mac}")  
            print("Fabricante : No se encontró en la base de datos")  
    except Exception as e:  
        # Si ocurre un error durante la solicitud, se muestra un mensaje de error  
        print(f"Error al consultar la MAC: {e}")
```

Imagen 2

Esta función consulta el fabricante de una dirección MAC utilizando la API `maclookup.app`.

- Se recibe una dirección MAC como argumento y se forma una solicitud GET para la API.
- Si la API responde con información del fabricante (campo 'company'), se muestra el fabricante.
- Si no, muestra que no se encontró la MAC en la base de datos.
- La función también maneja errores, como problemas de red o respuestas incorrectas de la API.

Y devuelve al fabricante si está disponible. En caso de error, captura y muestra el mensaje correspondiente.

Filtrado de MACs Especiales

La función `is_special_mac()` de la Imagen 3 filtra las MACs especiales de tipo broadcast y multicast, que no tienen un fabricante asociado:

Estas MACs son conocidas por no tener un fabricante específico, por lo que no es útil consultarlas en la API.

```
def is_special_mac(mac):  
    # Las MACs que empiezan con 'ff-ff-ff' (broadcast) o '01-00-5e' (multicast) son filtradas  
    return mac.startswith('ff-ff-ff') or mac.startswith('01-00-5e')
```

Imagen 3

Consulta de la Tabla ARP

La función `lookup_arp()` que se muestra en Imagen 4 obtiene la tabla ARP del sistema utilizando el comando `arp -a`, luego filtra las MACs duplicadas y especiales, y consulta el fabricante de cada MAC válida:

- Se ejecuta el comando del sistema 'arp -a' para obtener las MACs conectadas.
- Se filtran las MACs especiales y las duplicadas.
- Se consulta el fabricante para cada MAC válida en la API.

```
def lookup_arp():
    try:
        # Ejecutamos el comando 'arp -a' para obtener la tabla ARP
        result = subprocess.run(['arp', '-a'], stdout=subprocess.PIPE, text=True)
        # Guardamos la salida del comando (la tabla ARP)
        arp_output = result.stdout
        print("Tabla ARP:")
        print(arp_output)

        # Usamos una expresión regular para encontrar todas las MACs en la tabla ARP
        macs = re.findall(r'([0-9a-fA-F]{2}[:-][0-9a-fA-F]{2}[:-][0-9a-fA-F]{2}[:-][0-9a-fA-F]{2}[:-][0-9a-fA-F]{2}[:-][0-9a-fA-F]{2}[:-][0-9a-fA-F]{2})', arp_output)
        # Removemos duplicados para no consultar la misma MAC más de una vez
        unique_macs = set(macs)

        # Para cada MAC en la tabla ARP
        for mac in unique_macs:
            # Si la MAC no es especial (como broadcast o multicast), la consultamos en la API
            if not is_special_mac(mac):
                print(f"\nConsultando fabricante para la MAC: {mac}")
                lookup_mac(mac)

    except Exception as e:
        # Si ocurre algún error al obtener la tabla ARP, La mostramos
        print(f"Error al obtener la tabla ARP: {e}")
```

Imagen 4

Esta función obtiene la tabla ARP del sistema mediante el comando `arp -a`, filtra las direcciones MAC duplicadas y especiales, y consulta los fabricantes de las MAC válidas.

Función de Prueba

La función `test_macs()` que se visualiza en la Imagen 5 verifica el funcionamiento del programa utilizando un conjunto de direcciones MAC predefinidas:

```
def test_macs():  
  
    test_mac_addresses = ['98:06:3c:92:ff:c5', '9c:a5:13', '48-E7-DA']  
  
    # Para cada MAC de prueba, llamamos a 'lookup_mac' para obtener el fabricante  
    for mac in test_mac_addresses:  
        print(f"\nProbando la MAC: {mac}")  
        lookup_mac(mac)
```

Imagen 5

Esta función es solo para pruebas. Verifica que el programa pueda detectar y consultar las siguientes MACs:

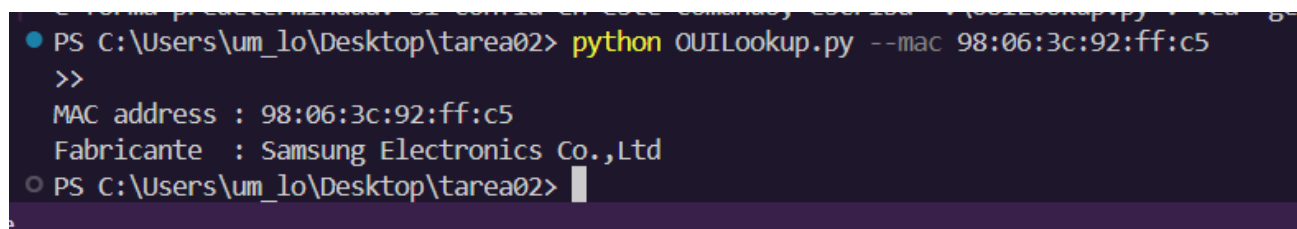
- 98:06:3c:92:ff:c5
- 9c:a5:13
- 48-E7-DA

Esta función es utilizada para verificar el correcto funcionamiento del programa, probando con un conjunto de MACs de prueba y mostrando el fabricante correspondiente.

5. Pruebas

Durante la fase de pruebas, se realizaron consultas con MACs que estaban y no estaban en la base de datos de la API. También se probó la funcionalidad de extracción de la tabla ARP para verificar que el programa podía obtener las direcciones MAC locales y consultar los fabricantes correspondientes.

Caso de prueba 1 (MAC encontrada): En la Imagen 6 se muestran el comando "python OUILookup.py --mac 98:06:3c:92:ff:c5" y su resultado esperado: Samsung Electronics Co.,Ltd



```
PS C:\Users\um_lo\Desktop\tarea02> python OUILookup.py --mac 98:06:3c:92:ff:c5  
>>  
MAC address : 98:06:3c:92:ff:c5  
Fabricante : Samsung Electronics Co.,Ltd  
PS C:\Users\um_lo\Desktop\tarea02>
```

Imagen 6

Caso de prueba 2 (MAC no encontrada): En la Imagen 7 se ejecuta el comando "python OUILookup.py --mac 98:06:3f:92:ff:c5" y su resultado esperado: No se encontró en la base de datos

```
PS C:\Users\um_lo\Desktop\tarea02> python OUILookup.py --mac 98:06:3f:92:ff:c5
MAC address : 98:06:3f:92:ff:c5
Fabricante : No se encontró en la base de datos
PS C:\Users\um_lo\Desktop\tarea02>
```

Imagen 7

Se verificó el funcionamiento correcto del código realizando consultas y asegurando que los resultados de la API coincidieran con las expectativas.

Caso de prueba 3 (Consulta ARP): En la Imagen 8 se muestra el comando "python OUILookup.py --arp" y su resultado esperado: Mostrar la lista de MACs locales y sus fabricantes asociados.

```
PS C:\Users\um_lo\Desktop\tarea02> python OUILookup.py --arp
>>
Tabla ARP:

Interfaz: 192.168.0.17 --- 0xd
Dirección de Internet      Dirección física      Tipo
192.168.0.1                c0-89-ab-a3-ac-c0    din mico
192.168.0.252              00-00-ca-01-02-03    din mico
192.168.0.255              ff-ff-ff-ff-ff-ff    est tico
224.0.0.22                 01-00-5e-00-00-16    est tico
224.0.0.251                01-00-5e-00-00-fb    est tico
224.0.0.252                01-00-5e-00-00-fc    est tico
239.192.152.143            01-00-5e-40-98-8f    est tico
239.255.255.250            01-00-5e-7f-ff-fa    est tico
255.255.255.255            ff-ff-ff-ff-ff-ff    est tico

Consultando fabricante para la MAC: c0-89-ab-a3-ac-c0
MAC address : c0-89-ab-a3-ac-c0
Fabricante : ARRIS Group, Inc.

Consultando fabricante para la MAC: 00-00-ca-01-02-03
MAC address : 00-00-ca-01-02-03
Fabricante : ARRIS Group, Inc.
PS C:\Users\um_lo\Desktop\tarea02>
```

Imagen 8

Se verificó que los resultados obtenidos coinciden con los esperados según la base de datos de la API.

6. Funcionamiento de MACs aleatorias en dispositivos electrónicos

En los dispositivos electrónicos modernos, especialmente en teléfonos inteligentes y computadoras portátiles, es común el uso de direcciones MAC aleatorias cuando estos buscan conectarse a redes Wi-Fi. Este mecanismo tiene como objetivo aumentar la privacidad del usuario al impedir que los dispositivos sean rastreados mediante sus direcciones MAC. Las direcciones MAC convencionales, asignadas de manera única por los fabricantes, son estáticas, mientras que las MACs aleatorias cambian periódicamente y se utilizan en redes públicas o no confiables para evitar la recolección de datos del dispositivo.

7. Discusión y conclusiones

El programa cumple con los objetivos establecidos, permitiendo consultar el fabricante de tarjetas de red a través de su dirección MAC. Además, se implementó correctamente la funcionalidad de obtener la tabla ARP y consultar los fabricantes de las MACs encontradas en la red local. Como mejora futura, se podría optimizar el manejo de errores en la conexión con la API y añadir más casos de prueba para MACs no válidas.

Este tipo de direcciones es utilizado principalmente para mejorar la privacidad en redes inalámbricas, ya que las MACs aleatorias impiden que los dispositivos sean identificados y rastreados fácilmente a través de diferentes redes.

8. Referencias

[1] API REST para consulta de MACs: <https://maclookup.app>

[2] Internet Assigned Numbers Authority (IANA). "Ethernet Numbers". Disponible en: <https://www.iana.org/assignments/ethernet-numbers>

[3] Python Software Foundation. *Python Documentation: getopt — C-style parser for command line options*. Disponible en: <https://docs.python.org/3/library/getopt.html>

[4] Fundación Python. Documentación de Python: *re — Operaciones con expresiones regulares*. La documentación oficial en español para el manejo de expresiones regulares en Python. Disponible en: <https://docs.python.org/es/3/library/re.html>

[5] Fundación Python. Módulo subprocess — Gestión de procesos secundarios. Este módulo permite ejecutar comandos del sistema desde un script en Python. Disponible en: <https://docs.python.org/es/3/library/subprocess.html>