Documentación – Análisis de Trafico de Red

Introducción

Este proyecto consiste en una aplicación desarrollada en Python que permite capturar paquetes de red en una interfaz específica, almacenar los datos en una base de datos SQLite y mostrar estadísticas básicas del tráfico en tiempo real.

El objetivo es ofrecer una herramienta ligera y portable, ejecutable dentro de un contenedor Docker, que cumpla con los requisitos de análisis de tráfico planteados en el desafío.

Ejecutar (rápida)

Linux (recomendado para capturar interfaces reales)

Para que el contenedor vea y capture tráfico de las interfaces físicas del host (Wi-Fi, Ethernet), ejecuta con red de host y capacidades de red:

```
# Genera iface.txt / crea data/ y ejecuta (si usas run.sh, ya lo hace)
./run.sh

# o ejecutar manualmente:
docker run -it --rm --net=host --cap-add=NET_ADMIN \
  -v "$(pwd)/iface.txt:/app/iface.txt" \
  -v "$(pwd)/data:/app/data" \
  --name trafico-test sniffer-lite
```

macOS / Windows (limitación de Docker Desktop)

Docker Desktop no expone las interfaces físicas del host al contenedor de la misma forma que Linux. En macOS el sniffer dentro del contenedor solo ve la interfaz del contenedor (ej. `eth0`) y Windows no es capaz de detectar interfaces WiFi. Para pruebas en macOS/Windows:

1. Genera `iface.txt` (el `run.*` lo hace).
2. Ejecuta el contenedor normalmente (sin `--net=host` en macOS/Windows; `run.ps1` / `run.sh` ya usan la configuración adecuada):

```
sudo ./run.sh       # macOS / Linux

powershell -ExecutionPolicy Bypass -File .\run.ps1  # Windows PowerShell

(ejecutar PowerShell como administrador)
```

3. Para generar tráfico de prueba desde el host que el contenedor capture (macOS/Windows), usa este comando dentro del contenedor o con `docker exec`:

```
# Genera tráfico TCP hacia 8.8.8.8 (funciona sin instalar paquetes extra)
docker exec -it trafico-test python -c "import socket; s=socket.socket();
s.connect(('8.8.8.8',53)); s.close()"
```

Uso (flujo)

1.Ejecuta `run.sh` o `run.ps1`.

2.Si existe `iface.txt`, `app.py` lista las interfaces y pide seleccionar un índice (si solo hay 1 línea, se auto-selecciona).

3.El sniffer captura y:

- Guarda paquetes en buffer (tamaño `BUFFER_SIZE = 100`).
- Inserta en SQLite cuando el buffer se llena o al abortar (Ctrl+C).
- Muestra estadísticas en consola cada 25 paquetes.

4.Parar: `Ctrl+C` → hace `flush` del buffer, imprime estadísticas finales y sale.

Consultas rápidas a la base de datos (host)

La base persiste en `./data/packets.db`. Comandos útiles:

```
# Últimas 10 entradas
sqlite3 data/packets.db "SELECT id,date,src_ip,dst_ip,protocol,length,interface
FROM packets ORDER BY id DESC LIMIT 10;"

# Conteo total
sqlite3 data/packets.db "SELECT COUNT(*) FROM packets;"

# Top 5 IPs origen
sqlite3 data/packets.db "SELECT src_ip, COUNT(*) AS total FROM packets GROUP BY
src_ip ORDER BY total DESC LIMIT 5;"

# Top 5 IPs destino
sqlite3 data/packets.db "SELECT dst_ip, COUNT(*) AS total FROM packets GROUP BY
dst_ip ORDER BY total DESC LIMIT 5;"
```

Esquema de la tabla `packets`

```
CREATE TABLE IF NOT EXISTS packets (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    date TEXT,
    src_ip TEXT,
    dst_ip TEXT,
    protocol TEXT,
    length INTEGER,
    interface TEXT
);
```

Explicación de campos:

- `date`: timestamp `YYYY-MM-DD HH:MM:SS`.
- `src_ip` / `dst_ip`: direcciones IPv4 capturadas.

- `protocol`: `TCP` / `UDP` / `ICMP` / `OTRO`.
- `length`: tamaño del paquete en bytes.
- `interface`: interfaz donde se capturó (valor de `iface.txt` o `eth0`).

Diseño y decisiones técnicas (resumen)

- Scapy: elegida por su potencia para captura y parsing de paquetes.
- SQLite: persistencia simple y portable; no requiere servidor.
- Buffer de escritura: evita commits por cada paquete, mejora rendimiento.
- Hilo de captura: `threading.Thread` para no bloquear la main loop; así se maneja `KeyboardInterrupt`limpiamente.
- Interfaz por archivo (`iface.txt`): elegido para reproducibilidad en Docker Desktop (genera interfaz desde el host y se monta), más predecible que auto-detección en entornos virtualizados.

Pruebas realizadas

- macOS (Docker): pruebas funcionales con tráfico creado desde contenedor y por `docker exec`(comando socket mostrado arriba). Limitación: no captura tráfico Wi-Fi del host.
- Ubuntu (Docker): pruebas completas con `--net=host` y `--cap-add=NET_ADMIN` → captura desde interfaces reales (Wi-Fi/Ethernet).
- Windows (Docker): pruebas exitosas creando `iface.txt` desde PowerShell y ejecutando el contenedor; la captura es funcional según configuración de red local.

Troubleshooting (errores comunes y soluciones rápidas)

`docker: invalid reference format`

- Causa: rutas con espacios o falta el `.` en `docker build`.
- Solución: usar comillas en volúmenes:

```
-v "$(pwd)/iface.txt:/app/iface.txt" -v "$(pwd)/data:/app/data"
```

`sqlite3.OperationalError: unable to open database file`

- Causa: la carpeta `data` no existía o no tenía permisos.
- Solución:

```
mkdir -p data
```

```
chmod 777 data    # Linux
```

`ValueError: Interface 'en0' not found !` o interfaz no encontrada

- Causa: contenedor no ve la interfaz del host (Docker Desktop).
- Solución:
  - En Linux: ejecutar con `--net=host --cap-add=NET_ADMIN`.
  - En macOS/Windows: usa `eth0` (interfaz del contenedor) o prueba en Linux nativo para captura real.

Seguridad y permisos

- Capturar paquetes requiere privilegios de red (promiscuous mode en algunos casos). En Docker Linux puede usarse `--cap-add=NET_ADMIN`.

Archivos útiles (resumen)

- `run.sh` — genera `iface.txt`, crea `data/` y ejecuta Docker (Linux/macOS).
- `run.ps1` — script PowerShell (Windows) equivalente.
- `Dockerfile` — instala `scapy` y utilidades necesarias (si quieres `ping`/`curl` agregar `apt install`).

Referencias consultadas

- Documentación oficial de Scapy

Tutoriales networking en Python y Docker en YouTube (HolaMundo, Pentester 77).

ChatgGPT (Corrección de errores en script run.ps1, pruebas y análisis de errores)

Conclusión

`sniffer-lite` cumple los requisitos del reto: captura paquetes, extrae campos clave, calcula estadísticas (total, por protocolo, top 5 origen/destino) y persiste en SQLite. Es portable en Docker, con la nota técnica sobre limitaciones de macOS/ Docker Desktop.