Reconstrucción de caminos y detección de dispositivos

David Moreno Maldonado

Tutor: Guillermo Julián Moreno
Ponente: Javier Aracil Rico

Universidad Autónoma de Madrid
Ingeniería Informática y Matemáticas

15 junio, 2020

Descripción general

Situación:

- Existencia de redes de gran tamaño y complejidad.
- Varios puntos de captura de tráfico en estas redes.

Problemas:

- Pérdida de paquetes.
- Ralentización de red.
- Paquetes duplicados.

$\Longrightarrow$ Necesidad de **detectar los puntos exactos** de la red donde se producen estos problemas.

Solución:

Reconstrucción de caminos IP

Detección y clasificación de dispositivos

Objetivos:

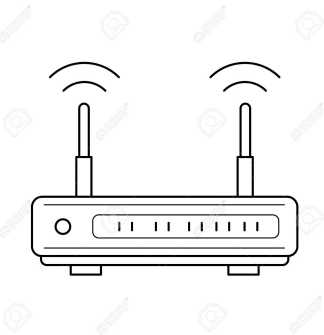
- Eficiencia en tiempo de ejecución y uso de memoria.
- Abstracción para su uso en diferentes entornos.

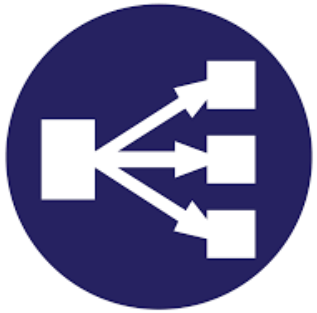
Reconstrucción de flujos IP

- Trazas *pcap* capturadas.
- Útiles como herramientas auxiliares.
- Análisis no específico en reconstrucción.
- Poca eficiencia con trazas de gran tamaño.

Detección de dispositivos

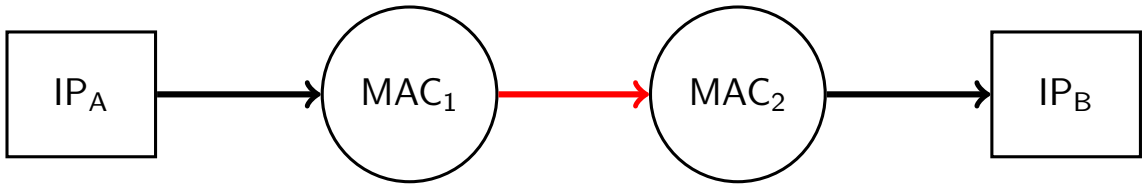
- No existen métodos generalizados.
- Estudio de las características para inferir el dispositivo.
- Existencia de dispositivos compuestos.





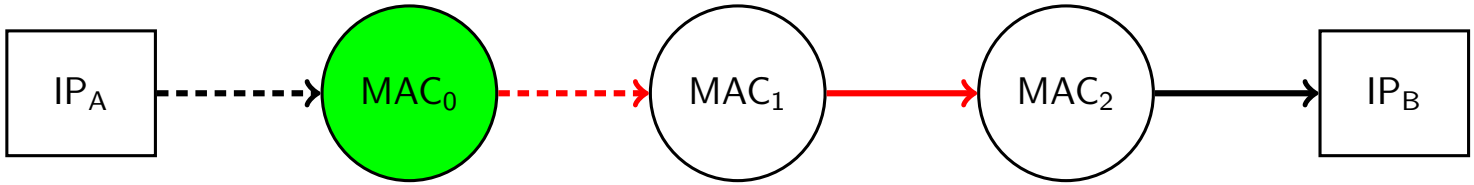
Planteamiento del problema y caso inicial

- La información recibida es a nivel de paquete con IP origen y destino y MAC origen y destino.
- Almacenar los flujos IP en un grafo unidireccional.
- Aristas abiertas y cerradas dependiendo de si la conexión entre nodos se ha visto físicamente.

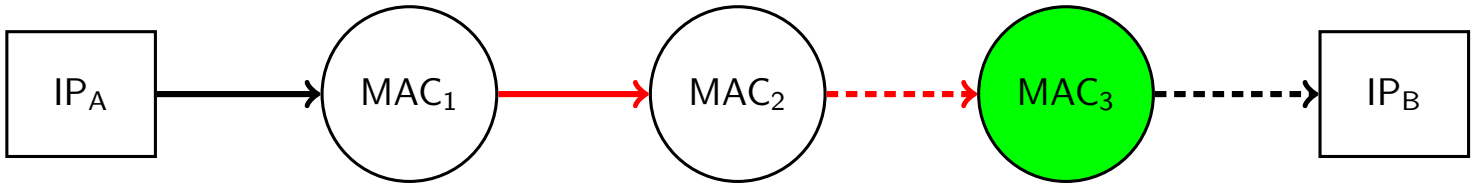


Inserción estándar

Cuando vemos un paquete con MAC destino igual al primer nodo MAC del grafo:



Cuando vemos un paquete con MAC origen igual al último nodo MAC del grafo:

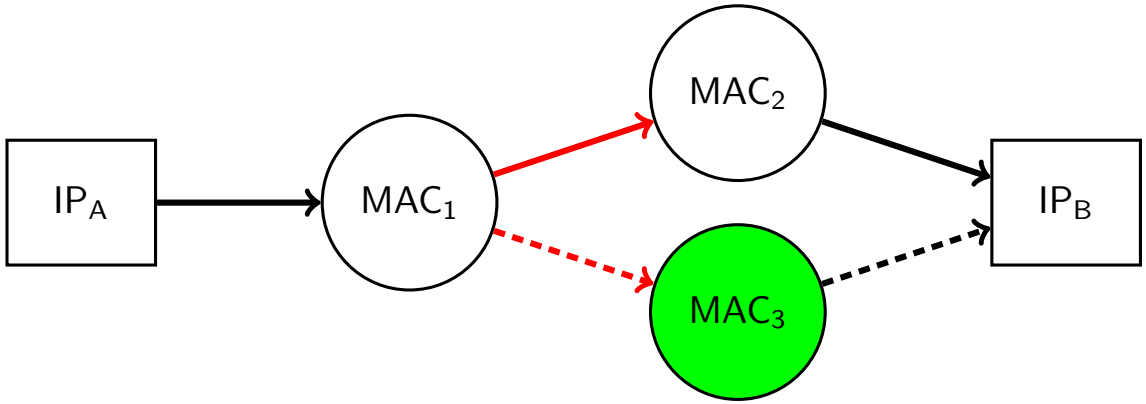


Bifurcación

¿Por qué ocurren?

- Caída temporal de una parte de la red.
- Congestión de la red.
- Funcionamiento usual de la red.

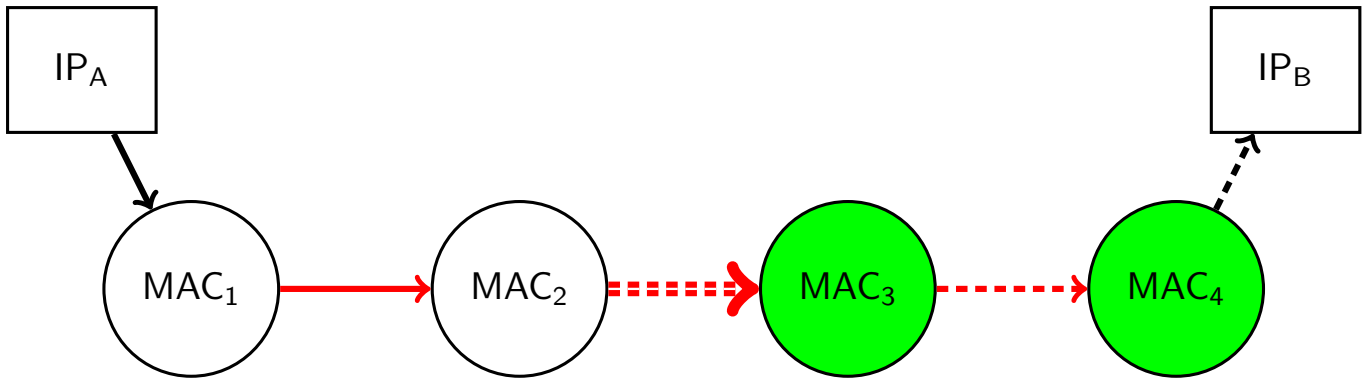
Las detectamos cuando observamos un paquete con un nodo MAC en el grafo, pero no estamos en el caso de inserción estándar:



Camino huérfano

Analizamos un paquete cuyas MAC no están en el grafo. $\Longrightarrow$ Necesitamos utilizar información extra. TTL del paquete.

Se genera una arista virtual conectando los nodos.



Identificación de dispositivos

1. Identificamos como dispositivos las aristas virtuales.
2. Agregamos la información de todos los flujos (MAC de entrada y de salida, bytes, *frames*, ...).
3. Resolvemos las dependencias existentes entre dispositivos.

Clasificación de los dispositivos

Diferenciamos entre *firewalls*, *routers* y balanceadores de carga a nivel MAC. Se siguen estas reglas:

1. Los dispositivos con $\Delta\text{TTL} = 0$ se clasifican como *firewalls*.
2. Aquellos que conecten más de una MAC física y *unicast* se clasifican como balanceadores de carga a nivel MAC.
3. El resto se clasifica como *router*.

Planteamiento inicial del desarrollo

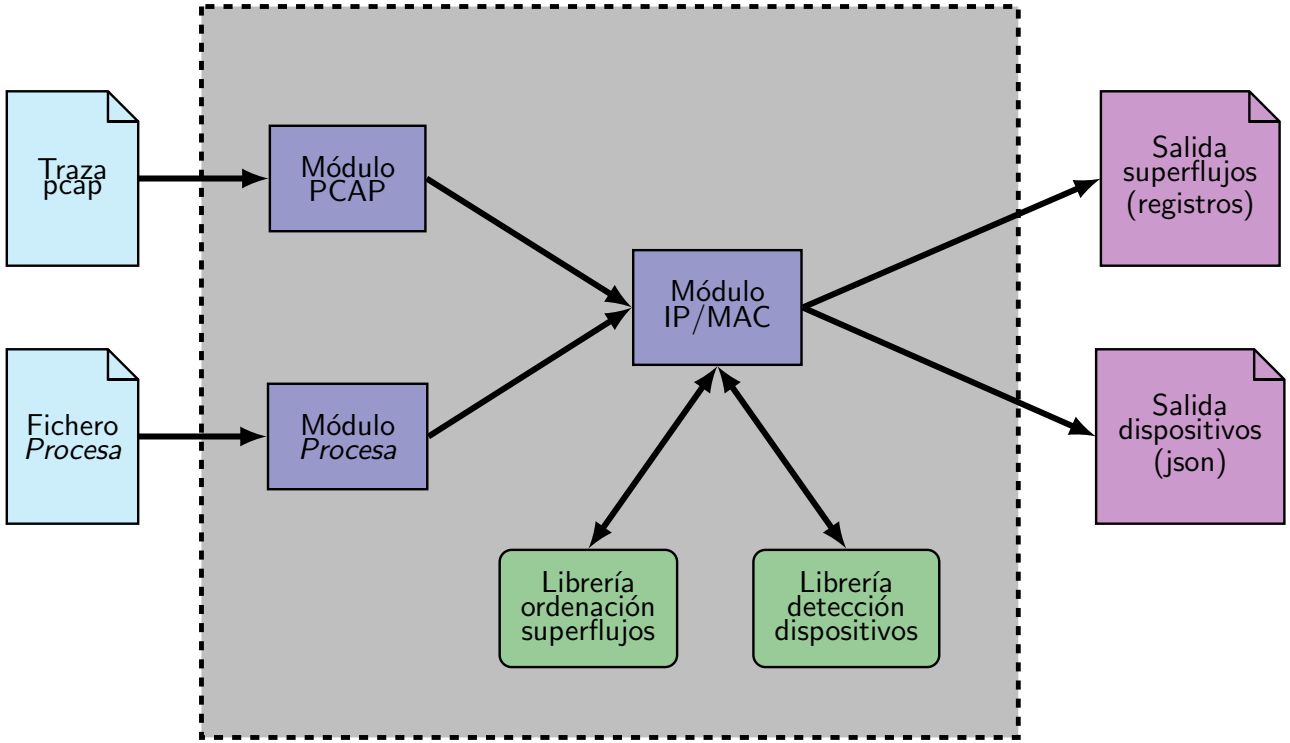
El programa se desarrollo para la empresa Naudit, dedicada al análisis y monitorización de redes.

Intento de herramienta similar en Python $\Longrightarrow$ Poca eficiencia en tiempo $\Longrightarrow$ Utilizamos C para el desarrollo

Integración con *fisher*

- Aplicación dedicada a la lectura de trazas pcap.
- Se utilizaron y modificaron librerías internas (estructuras IP y MAC, diccionarios, listas, log, test).
- Uso de memoria estática. Se evita el uso continuo de `malloc` y `free`.

Estructura general del programa



Librería de ordenación de superflujos

Librería abstracta, capaz de manejar flujos multicapa de diferentes características. Estructuras de datos definidas:

- Grafo unidireccional.
- Información de cada capa.
- Nodos.
- Aristas.
- Camino: Enlace entre estructuras externas.

Es necesario indicar la memoria disponible para estos elementos.

Módulo IP/MAC

- Especifica la librería de ordenación de flujos para las capas IP y MAC
- Encargada de definir el número de estructuras de cada tipo requerido.

Estructura	**Número de estructuras por superflujo**
Superflujos	1
Elementos IP	2
Elementos MAC	α
Nodos	$2 + \alpha$
Aristas	$(\text{nodos} - 1) + \lambda$

Entrada al programa

Ficheros *pcap*:

- Aprovecha totalmente la estructura interna de *fisher*.
- Formato estándar de captura de tráfico.

Ficheros *Procesa*:

- Fichero por registros de conexiones. Formato interno Naudit.
- Ficheros más compactos que los *pcap*.
- Necesidad de calcular menos datos internamente (RTT, número de *frames*, ...).

Salida del programa

Salida de los superflujos:

- Un registro por arista en el grafo
- Se indica información recopilada (número de *frames*, bytes, ...)

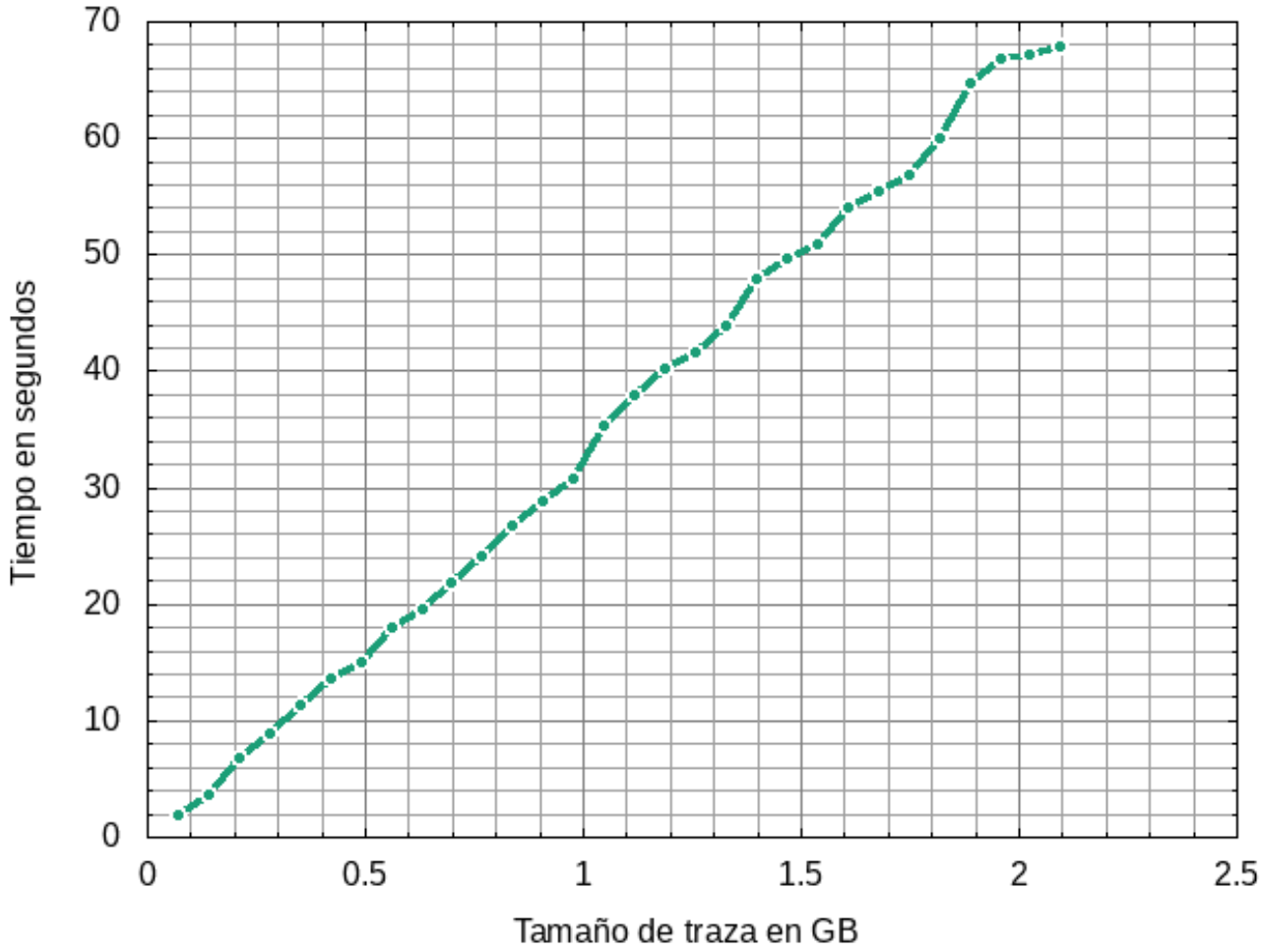
Salida de dispositivos:

- Formato JSON
- Información sobre interfaces MAC del dispositivo, flujo de paquetes y bytes o IPs que conecta.

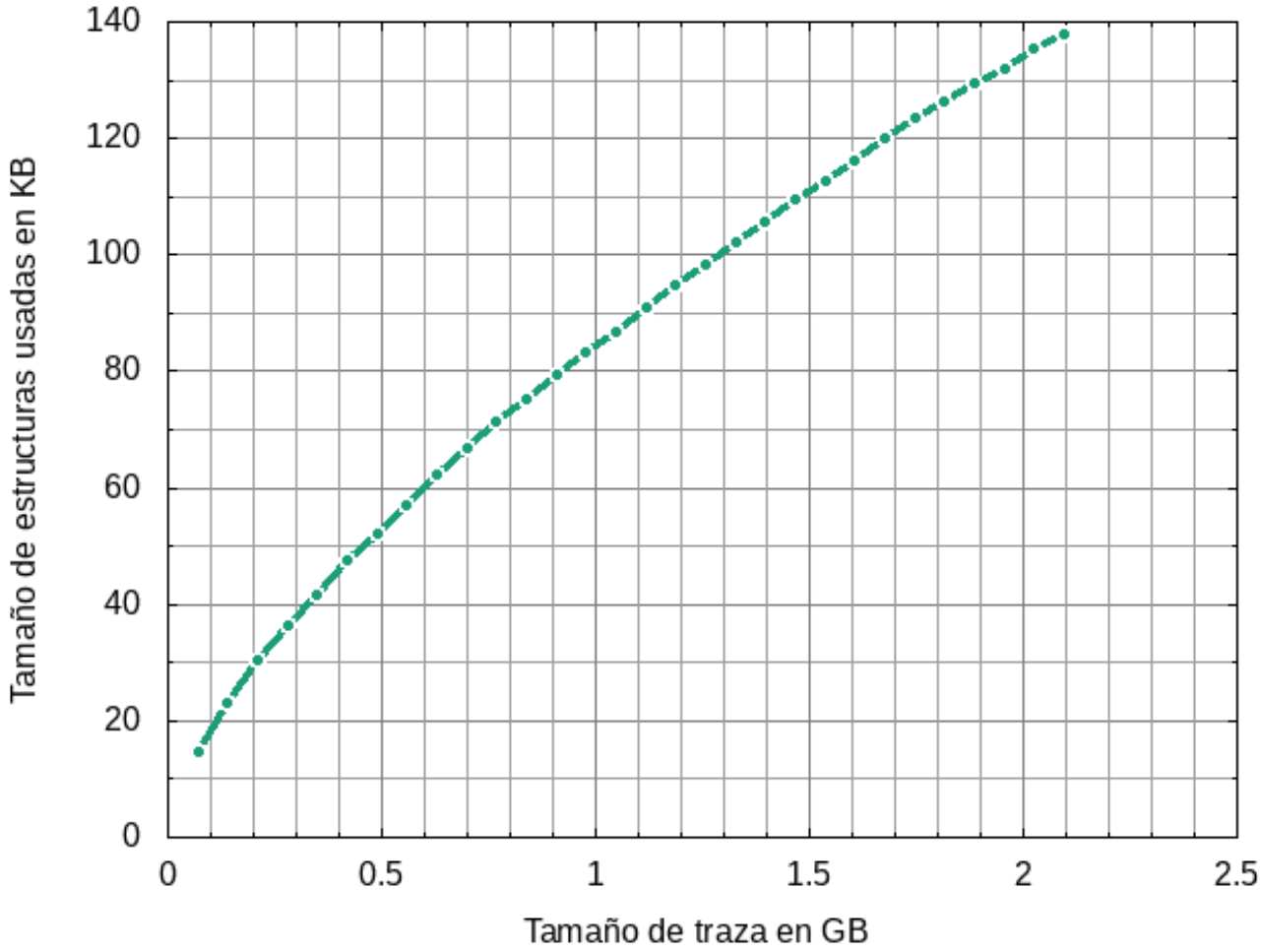
Pruebas realizadas al programa

- Uso del módulo de test integrado en *fisher*.
- Test específicos para las librerías de ordenación y clasificación.
- Uso del CI (*Continuous Integration*) disponible en GitLab.
- Depuración de memoria usada con Valgrind.

Tiempo de ejecución



Uso de memoria



Trabajo futuro

- Aumento de la información recopilada en cada dispositivo, por ejemplo, subredes.
- Clasificación más detallada (balanceadores de carga a nivel IP, routers NAT, ...).

Muchas gracias.