Universidad Autónoma de Madrid

Escuela politécnica superior

Trabajo de fin de máster

DISECCIÓN DE TRÁFICO WEB A ALTA VELOCIDAD

Máster en Ingeniería Informática

Carlos Gonzalo Vega Moreno
Tutor: Javier Aracil Rico

Septiembre 2014

DISECCIÓN DE TRÁFICO WEB A ALTA VELOCIDAD

AUTOR: Carlos Gonzalo Vega Moreno

TUTOR: Javier Aracil Rico

HPCN Research Group
Dpto. de Tecnología Electrónica y Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Septiembre 2014

Resumen

Resumen

Este trabajo de fin de máster sobre disección de tráfico web a alta velocidad abarca el proceso de diseño y desarrollo de una herramienta de procesado de tráfico web (Disector de ahora en adelante) cuyo objetivo y motivación es proveer de información sobre el tráfico web en varios proyectos del laboratorio para analizar, entre otras métricas, los retardos en las respuestas a peticiones HTTP en una red de alto rendimiento, ya sea de forma online o mediante el procesado de ficheros de tráfico web, y de esta forma estudiar el comportamiento y rendimiento de la red analizada.

Este trabajo forma parte de un proyecto real del grupo de investigación High Performance Computing and Networking de la EPS UAM y ha sido usado en varios proyectos de este laboratorio satisfaciendo las necesidades de un problema real en estos proyectos del laboratorio en localizaciones como Sudamérica, España o Baréin entre otros.

La herramienta se diferencia de otras ya existentes por el rendimiento ofrecido en velocidad del análisis como en la gestión de los recursos y funcionalidades. Cabe destacar su portabilidad que permite ser ejecutado en distintos entornos sin necesidad de instalar ningún componente especial o externo. Esta herramienta está pensada para ser usada en redes con grandes flujos de datos y un gran número de conexiones simultáneas por segundo. Proporciona opciones de filtrado que permiten focalizar mejor el análisis a realizar permitiendo ahorrar tiempo de procesado.

El trabajo comienza sobre una primera versión de esta aplicación, creada también por el autor de este proyecto, sobre la que se realizarán grandes cambios de diseño y funcionalidad para afrontar los nuevos requisitos surgidos.

Palabras Clave

Disector de tráfico, HTTP, Análisis, Redes, Alto Rendimiento, Web

Summary

Summary

This master's dissertation about high speed web traffic dissection covers the design and development process of a web traffic processing tool ("Dissector"from now on) aimed to provide information about web traffic in various projects of the laboratory to analyse, among other metrics, the response times between HTTP requests and responses in a high speed networking either online or from files, and thus studying the behaviour and performance of the analysed network.

This work is part of a real project of the High Performance Computing and Networking research team in the EPS UAM and it's been used in some projects of this laboratory satisfying the needs of a real problem in South America, Spain or Baréin, among others.

The tool distinguish itself from others by the performance offered in the analysis speed, the resources management and the functionalities. One of the main improvements to highlight is the portability of the tool, which allows to execute it in different systems without installing special components. The tool is intended to be used on networks with huge amounts of traffic and a large number of simultaneous connexions per second. It offers filter options that allows to focus the analysis saving processing time.

This project starts from a first tool version, developed also by the author of this project, on which major design and functionality changes will be made to meet the new arisen requirements.

Keywords

Traffic Dissector, HTTP, Analysis, Networking, High Performance, Web

Agradecimientos

En primer lugar a mi tutor Javier Aracil y al equipo del laboratorio HPCN por darme la oportunidad de trabajar con ellos todos estos años. También quiero agradecer a todos los que han colaborado en la realización de las pruebas y resolución de los problemas surgidos.

También quiero hacer especial mención a los compañeros de máster, muchos de ellos también de carrera, que me han acompañado todos estos años. Estoy seguro de que allá donde vayan tendrán un buen futuro. A ti Marina y a mi familia por el apoyo, paciencia y ánimos dados durante todo este tiempo.

Índice general

Índice de figuras

Glosario

- **Libpcap**: Librería que permite tratar archivos PCAP y capturar paquetes de la interfaz de red.
- **PCAP**: Packet Capture. Paquete con información del paquete capturado o leído con libpcap
- **Glib**: Librería de propósito general que permite el uso de otros tipos de datos no estándar del lenguaje C así como tablas Hash, Árboles y un mejor tratamiento de hilos y semáforos, entre otros.
- **BPF**: Berkeley Packet Filter. Permite filtrar paquetes para hacer más eficiente el análisis de estos.
- **Callback**: Función que es llamada al producirse un evento. En este proyecto es llamada cuando un nuevo paquete es analizado. Se encarga de procesar ese nuevo paquete.

1 Introducción

1.1. Motivación del trabajo

La motivación principal del proyecto es mejorar una aplicación de altas prestaciones para el análisis del tráfico web en redes. Esta herramienta debe seguir satisfaciendo los requisitos de velocidad y uso de recursos pasados además de satisfacer los nuevos que se proponen para su mejora. La herramienta provee información con la que realizar estadísticas del tráfico web.

Como parte de un proyecto vivo del laboratorio High Performance Computing and Networking de la Universidad Autónoma de Madrid nace la necesidad real de evaluar las prestaciones ofrecidas en una red de Sudamérica a base de analizar el tráfico web de esta. Las nuevas mejoras van orientadas a mejorar la velocidad y portabilidad de la aplicación haciéndola más estable y capaz de analizar mayores cantidades de tráfico en las redes con velocidades de 10 y 40 Gbps.

El hecho de que la herramienta sea usada en un proyecto vivo durante el desarrollo de mejoras, implica que surjan nuevas necesidades y requisitos durante el desarrollo, motivando aún más la necesidad de realizar cambios y perfeccionar la herramienta.

1.2. Objetivos y enfoque

Los objetivos principales del proyecto son analizar los distintos cambios en el diseño y los métodos para satisfacer las nuevas necesidades surgidas. Estos cambios deben suponer una diferencia respecto a otras herramientas disponibles actualmente y las propias versiones pasadas de la aplicación. Por supuesto, los cambios realizados no deben invalidar los objetivos y requisitos pasados y en consecuencia muchos de ellos se mantienen.

Por tanto los objetivos son los siguientes:

- Diseñar una herramienta que permita un análisis eficiente, rápido y eficaz del tráfico en una red de altas prestaciones y alto flujo de datos.
- Presentar una solución al problema que mejore las herramientas actuales y ofrezca nuevas prestaciones y mejor rendimiento respecto las versiones anteriores.
- Realizar un desarrollo con un ciclo de vida iterativo que implique la realización de varios prototipos y cambios en el diseño durante todo el proceso de desarrollo de la aplicación y resulten en una herramienta final testada, estable y perfeccionada. Los detalles sobre el ciclo de vida del proyecto se explican en el capítulo de desarrollo.
- Satisfacer las necesidades que surjan durante el desarrollo de la herramienta y su uso en un contexto real.
- Establecer unas lineas generales sobre cómo continuar y mejorar el proyecto en el futuro.

1.3. Metodología y plan de trabajo

El trabajo ha consistido en una fase primera de análisis de requisitos (explicados en el capítulo siguiente) en la cual se han tenido en cuenta las distintas necesidades y aspectos a cumplir para poder realizar un diseño acorde con ello. Se ha realizado un análisis de las distintas opciones para satisfacer los requisitos así como un análisis de prestaciones entre ellas.

El ciclo de vida elegido ha permitido perfeccionar la aplicación y corregir sus defectos hasta cumplir con los requisitos establecidos. Se han ido añadiendo requisitos nuevos a medida que surgían nuevas necesidades y problemas en el proyecto real.

1.4. Contenido del documento

El documento comienza con una introducción y motivación del proyecto realizado, expone los problemas que se deben resolver, establece los objetivos a cumplir así como los requisitos que se deben satisfacer para solucionar el problema propuesto.

En los apartados intermedios se presenta el diseño y desarrollo de la solución al problema con los distintas mejoras, cambios y correcciones realizadas durante el desarrollo del proyecto.

Por último, al final del documento se muestran las pruebas realizadas, tanto los escenarios de pruebas como los resultados, así como las conclusiones obtenidas de las pruebas y un último capítulo que expone mejoras futuras del proyecto.

2

Análisis y solución del problema

2.1. Análisis de requisitos

A continuación se explican tanto los requisitos funcionales como los no funcionales referentes al disector de tráfico. Muchos de ellos se mantienen de versiones anteriores aunque no todos se mencionan en este trabajo.

2.1.1. Requisitos funcionales del disector de tráfico

- El requisito más importante se mantiene, ya que es el objetivo principal de la herramienta, el cual consiste en mostrar por pantalla los tiempos de retardo entre la petición y la respuesta de una transacción HTTP ordenados en función de la hora de llegada de la respuesta junto con los datos de la conexión como las IPs y puertos de origen y destino, la marca temporal, la URL y el código y mensaje de respuesta.
- Los datos se mostrarán a partir de un fichero de entrada o una lista de ellos en formato PCAP, RAW o directamente en tiempo real haciendo uso de la interfaz de red indicada en los parámetros de ejecución de la herramienta.
- Otros requisitos funcionales son la posibilidad de introducir como parámetro un filtro BPF (BSD Packet Filter), tanto para sustituir el filtro por defecto como para concatenarlo a este mediante operadores OR o AND.
- También la posibilidad de introducir un filtro de URL o por HOST. Estos filtros sirven para tener la posibilidad de filtrar el tráfico analizado por cualquiera de sus parámetros (IP de origen, destino, puerto, etc), así como por su URL o HOST respectivamente.
- Permitir introducir un fichero con los paquetes que se deben descartar en el procesamiento.

- Una opción para garantizar que la salida está correctamente ordenada.

- Ofrecer una opción para eliminar de la salida, es decir a posteriori, las retransmisiones de paquetes.

- Permitir imprimir en un fichero la salida del recolector de basura, es decir las peticiones eliminadas por llevar excesivo tiempo esperando.

2.1.2. Requisitos no funcionales del disector de tráfico

- El rendimiento ofrecido en velocidad y recursos de memoria es elemental para poder analizar grandes cantidades de tráfico haciendo un uso eficiente e inteligente de los recursos que permitan aprovechar la velocidad de los discos, cuando se lee de fichero, o la de la interfaz si se realiza un análisis online. El mayor requisito es la velocidad aunque sea a costa del gasto de recursos, pero estos no deben gastarse sin justificación. Esto permitirá analizar el tráfico tan rápido como se genera.

- Otro factor importante es la portabilidad ya que su ejecución debe ser posible en distintos sistemas Linux/Unix con el menor número de dependencias y manteniendo la estabilidad sea cual sea el sistema en el que se ejecute. En consecuencia uno de los requisitos consiste en mantener la funcionalidad anteriormente explicada eliminando el uso de la librería glib, la cual ha traído problemas de portabilidad por las numerosas versiones existentes y las dificultades para su instalación en determinados entornos.

- Debido a este ultimo punto es necesario modificar la tabla hash para no hacer uso de librerías externas en este aspecto así como el tratamiento de los hilos.

- Es necesario cambiar la forma en la que usan los recursos de memoria para reducir el gran número de reservas y liberaciones de memoria realizadas durante la ejecución del programa, las cuales repercuten en una perdida de tiempo importante.

- Todos estos cambios deberán hacerse manteniendo la estabilidad de la aplicación.

2.2. Análisis y solución del problema

En esta sección se analizan tanto los problemas como las soluciones a implementar de las distintas partes del proyecto. Algunos detalles que se explican en esta sección serán determinantes en los capítulos de diseño y desarrollo. Para explicar algunos problemas y requisitos surgidos es necesario entender mínimamente el funcionamiento de la aplicación que se comenta superficialmente en este apartado y que se explicará a fondo en posteriores apartados. En mayor medida el problema a resolver sigue siendo el mismo que en las versiones anteriores de la herramienta pero la forma de resolverlo cambia ya que es necesario cumplir los nuevos requisitos de velocidad y portabilidad.

2.2.1. Problema principal

Como se ha explicado en la sección de requisitos, el principal objetivo del disector de tráfico es mostrar la diferencia entre el tiempo de una petición y su respuesta. A continuación exponemos la operación básica que realiza el programa con los paquetes.

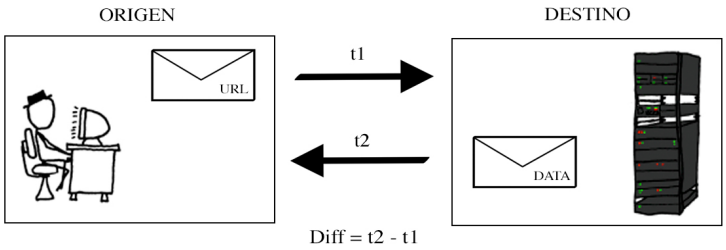


Figura 2.1: Ilustración de transacción

Cada petición tiene un origen, un destino, una marca temporal y la URL que estamos pidiendo. Como es obvio la respuesta también tiene un origen, un destino y una marca temporal, además del dato correspondiente a la URL. La tarea principal de la aplicación es la clasificación de conexiones dentro de la tabla hash de acuerdo con sus direcciones IP y puertos de origen y destino (de ahora en adelante 4-tupla).

Los paquetes se leen mediante la librería libpcap la cual soporta un filtrado de paquetes mediante un Berkeley Packet Filter. El filtro por defecto de la herramienta descarta todos aquellos paquetes que no sean una petición HTTP o una respuesta. Tras pasar este filtrado, la herramienta procesa el paquete.

El paquete se procesa en el módulo HTTP que reconoce el tipo de petición o respuesta y comprueba las cabeceras. La función hash retorna una clave basada en la 4-tupla a la cual se le aplica el operador módulo con el tamaño de la tabla hash. Hay una lista de colisiones para cada celda de la tabla así como una lista de peticiones para cada conexión en la lista de colisiones.

Figura 2.2: Un grupo de usuarios mandando peticiones con distintas conexiones

En un caso real cada usuario realizará un sinfín de peticiones, en distintas conexiones, es decir, con distintos destinos. En la figura 2.2, cada sobre ilustra una petición y cada

color una conexión con un destino diferente. Además de una gran cantidad de peticiones realizadas por un usuario tenemos un gran número de usuarios, es decir, de orígenes. En consecuencia el número de peticiones y respuestas se vuelve enorme.

En el apartado de diseño se explica con más detalle la solución implementada con las estructuras y algoritmos utilizados tanto para clasificar y almacenar las conexiones como para emparejar las respuestas con sus correspondientes peticiones.

2.2.2. Paquetes desordenados

Las respuestas pueden llegar desordenadas. Normalmente los paquetes son ordenados en la capa OSI y por tanto las aplicaciones superiores reciben los paquetes ordenados. Al capturar el tráfico este está en bruto lo que implica que leyendo este tráfico podemos encontrar la respuesta a una segunda petición y más tarde la respuesta a la primera petición. En consecuencia, en la herramienta estas transacciones eran emparejadas erróneamente. A continuación se expone cómo evitarlo:

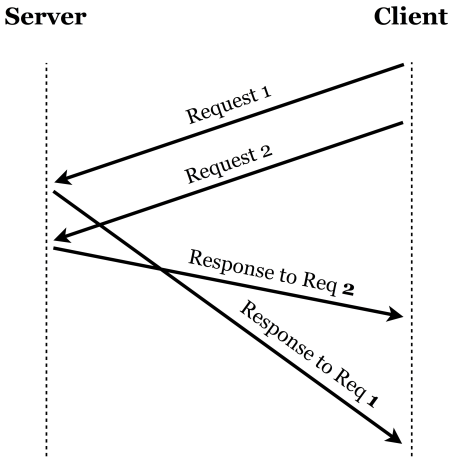


Figura 2.3: Las respuestas de una misma conexión TCP pueden llegar desordenadas

Cuando una respuesta llega, la 4-tupla es calculada igual que con las peticiones. El proceso es similar. Se obtiene la conexión correspondiente, si existe, mediante la tabla gracias a la posición dada por el id de la 4-tupla. En la lista de peticiones de dicha conexión se busca una que posea el mismo número ACK que el número de secuencia de la respuesta. Si ambos coinciden la información de la transacción es mostrada por pantalla o escrita en el fichero.

2.2.3. Salida ordenada

A pesar de solucionar el problema mencionado ahora una transacción anterior puede ser impresa después de otra más reciente, como la figura 2.3 indica. Por tanto aparece la necesidad de ordenar la salida, que se explicará en capítulos posteriores.

2.2.4. Eliminación de duplicados y retransmisiones

Debido a que la herramienta no reensambla los flujos TCP no puede ser consciente de las retransmisiones ni de los duplicados. Por tanto, una forma de solucionar el problema consiste en filtrar las transacciones antes de que estas sean imprimidas. Este procedimiento se explica en detalle en capítulos posteriores.

2.2.5. Portabilidad

Debido a las necesidades de portabilidad es necesario prescindir de ciertas librerías que a pesar de facilitar la implementación de la herramienta dificultaban su ejecución en entornos con restricciones en cuanto a la instalación de librerías y software. Esto implica realizar cambios en algunos de los pilares de la aplicación como es la tabla hash. Será necesario desarrollar una propia y estudiar su rendimiento.

2.2.6. Velocidad

Gracias a que ciertos proyectos del laboratorio hacen uso de esta herramienta se ha podido apreciar que el requisito de velocidad puede verse mejorado debido al gran número de llamadas realizadas para reservar y liberar recursos de memoria. Por tanto será necesario crear un sistema de gestión de la memoria en forma de pool de estructuras y estudiar su rendimiento. Este punto junto con el anterior se explican en profundidad en los apartados de diseño y desarrollo.

3

Tecnologías a utilizar

3.1. Lenguaje de programación

El lenguaje de programación se mantiene ya que no hay motivos para cambiar a otro y hacerlo supondría un mayor trabajo y coste en tiempo. La aplicación está programada en C debido a que ofrece un gran rendimiento porque permite utilizar características de bajo nivel y realizar una implementación con un rendimiento óptimo. Además es un lenguaje ampliamente utilizado en el campo que nos incumbe y existen numerosas librerías útiles, optimizadas y testadas que ayudan a la hora de implementar una aplicación eficiente así como mucha documentación y discusión sobre las tablas hash y gestión del uso de memoria. Al ser un lenguaje de menor nivel es necesario dedicar más tiempo a la codificación pero las ventajas que ofrece frente a los inconvenientes hacen que merezca la pena utilizarlo.

Este lenguaje fue escogido respecto a otros como Python porque además de permitir operaciones a bajo nivel permite controlar mucho mejor el uso de memoria realizado por el programa y satisface los requisitos de altas prestaciones propios del contexto de la aplicación.

3.2. Librerías externas

Aquí se exponen las librerías utilizadas que requieren de una instalación explícita. Esto quiere decir que otras librerías estándar o comunes entre sistemas no se mencionan aquí, como pueden ser la librería de expresiones regulares regex o relacionadas con estructuras y funciones de red (arpa/inet).

3.2.1. Libpcap

La principal librería usada para el desarrollo del disector es libpcap la cual permite tanto el tratamiento de los archivos PCAP (packet capture) como la captura directa desde la interfaz de red. Es una librería openSource testada, optimizada y bien documentada que permite el procesado de paquetes de red. Ha sido fundamental en el desarrollo de esta aplicación desde sus inicios. Fue desarrollada por los creadores de tcpdump del Lawrence Berkeley National Laboratory, otra gran herramienta ampliamente utilizada en el mundo de las redes.

Una de las características más importantes de la librería es la posibilidad de utilizar un filtro BPF (Berkeley Packet Filter **[4]**) que permita descartar los paquetes no útiles desde la capa de kernel, evitando que estos se copien al nivel de usuario y reduciendo la carga de la CPU ya que se realiza en las capas inferiores evitando que estos paquetes lleguen a capas superiores lo cual conlleva recursos y tiempo.

3.2.2. Glib

Una de las librerías más importantes en la anterior versión era Glib, la cual permitía usar tablas hash, listas enlazadas, y demás tipos de datos no disponibles en C de forma estándar. Se ha prescindido de esta librería por el coste en portabilidad ya que depende de muchas otras librerías que no siempre están disponibles para su instalación. En algunos proyectos no era posible actualizar las librerías del sistema de las máquinas y el coste en tiempo para poder usar la aplicación en determinados sistemas hizo que se plantease la portabilidad como un objetivo más serio. En consecuencia muchas estructuras han sido implementadas desde cero en C.

3.2.3. NDleeTrazas

Esta librería, proporcionada por el grupo de investigación de la Escuela Politécnica Superior, High Performance Computing and Performance de la UAM es un envoltorio (wrapper) de las funciones de **libpcap** para poder utilizar tanto archivos en formato RAW como archivos en formato PCAP. El programa depende mayormente de ella para procesar los paquetes.

4
Diseño

4.1. Estructura

A continuación se explica la estructura de diseño del disector dando primero una visión general y después una más detallada. Los aspectos referentes al rendimiento se explican en el capítulo siguiente (**Desarrollo**) mientras que en esta sección se consideran únicamente los aspectos teóricos a priori del diseño.

Los detalles de implementación se tratan en este capítulo en la sección de **Implementación**. No obstante muchos aspectos de procedimiento son necesarios para entender la estructura de la herramienta y es por esto que se hace referencia a ellos.

4.1.1. Estructura general del disector

La estructura general es simple, en primer lugar mediante NDLeeTrazas los paquetes, ya sean en formato RAW o PCAP, son leídos uno a uno en un bucle que se ejecuta hasta que el fichero ha sido leído en su totalidad. En el caso de leer de la interfaz de red el bucle continua hasta que se interrumpa la ejecución. En ningún caso el tráfico TCP será reensamblado ya que supondría un coste en tiempo.

La función Callback es llamada por cada paquete y esta se encarga de procesar el paquete haciendo uso de los distintos módulos del programa. En primer lugar se analiza el paquete para ver si este es correcto y después el módulo HTTP analiza el payload TCP para ver comprobar que el paquete es HTTP y está bien formado.

En caso de que el paquete no sirva, este es descartado y el control devuelto al bucle de lectura de paquetes. Por otro lado, si el paquete es HTTP este o bien es insertado en la tabla o se procede a buscar su correspondiente pareja en la tabla. Este procedimiento se explica en detalle en las siguientes secciones.

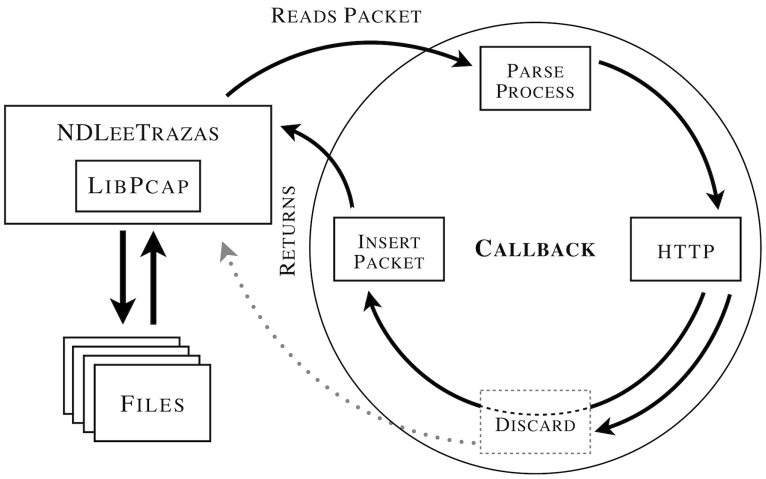


Figura 4.1: Estructura general

4.1.2. Estructura interna del disector

Filtrado de paquetes

Como se explicó en los capítulos anteriores NDLeeTrazas permite el uso de filtros BPF, estos descartan los paquetes que no nos son útiles según el filtro que hemos establecido. En el caso de realizar una lectura desde la interfaz de red estos serán descartados en las capas inferiores del modelo OSI reduciendo el tiempo y uso de recursos de la aplicación.

En nuestro caso el filtrado por defecto descarta todos aquellos paquetes que no sean una petición o respuesta, esto es, únicamente el primer paquete de la petición y el primer paquete de la respuesta, nada más. Igualmente, dado que el usuario puede establecer filtros propios la aplicación comprueba si los paquetes son adecuados descartando los que no procedan.

Clasificación de las peticiones HTTP

Después de procesar el paquete y confirmar que este es una petición, se procede a introducirla en la tabla hash. Para ello se busca la celda en base a la clave hash (calculada mediante la 4-tupla). En cada celda hay una lista de colisiones debido a que varias 4-tuplas pueden llevar a la misma celda ya que el tamaño de la tabla nunca será tan grande como para haber una celda por cada 4-tupla. En esta lista se busca la conexión y de no existir se introduce, añadiendo la petición a la lista de peticiones de la conexión.

Emparejando peticiones y respuestas

Cuando una respuesta llega, la clave hash es calculada de la misma forma que con las peticiones. El proceso es similar, la conexión se obtiene de la lista de colisiones en la celda de la tabla hash y de existir se compara el número de secuencia de la respuesta con el número ACK de cada petición en la lista de peticiones de la conexión. Si coinciden se imprimen los datos correspondientes a la transacción.

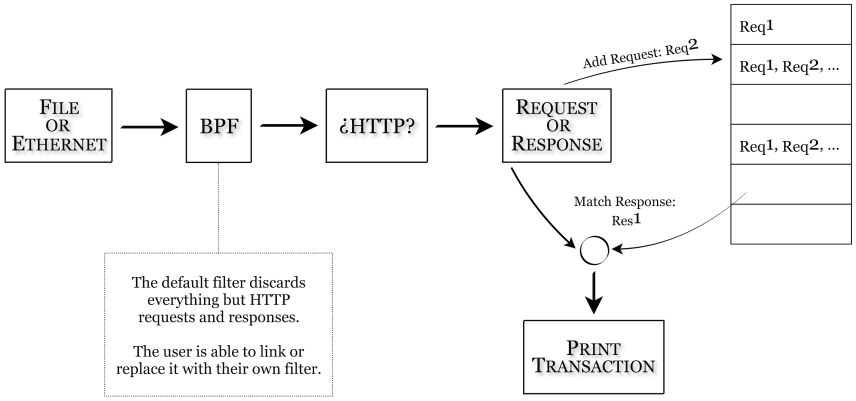


Figura 4.2: Recorrido de un paquete

Ordenamiento de la salida

En caso de que la opción esté activada, las líneas a imprimir se almacenan en un buffer para ser ordenadas antes de imprimirlas por pantalla. Esto es necesario debido a que algunas transacciones, como se explicó en apartados anteriores, pueden aparecer desordenadas. Más detalles en el apartado de Implementación y en el capítulo de Desarrollo.

Eliminación de duplicados y retransmisiones

De nuevo este apartado se explica más detalladamente en el capítulo de Desarrollo. El proceso es similar a la ordenación añadiendo un proceso de eliminación de peticiones en base a los números de secuencia y ACK.

4.2. Implementación

En esta sección se explican detalles como los algoritmos de clasificación de conexiones, las estructuras con las que se almacenan los datos en memoria, la forma en que se guardan los datos a representar en la interfaz web, etc.

La herramienta hace uso de distintas estructuras que se describen a continuación. Estas hacen referencia a las de la versión final de la aplicación. En el capítulo de desarrollo se exponen algunos hechos relevantes o cambios surgidos que han llevado a la versión actual de las estructuras y del procedimiento del programa.

4.2.1. Tabla hash

Para clasificar y almacenar las conexiones y las peticiones se hace uso de una tabla hash donde la clave está compuesta de: dirección IP de origen (IP_o), puerto de origen (P_o), dirección IP de destino (IP_d) y puerto de destino (P_d). La clave por tanto será la concatenación de estos cuatro datos ($IP_oP_oIP_dP_d$) llamada 4-tupla.

El dato asociado a la clave es una estructura que contiene la lista de colisiones para esta celda en la tabla. La tabla hash tiene un tamaño de 2^{30} elementos de tipo collision_list.

La lista de colisiones posee un puntero a una lista enlazada en la que se almacenan las conexiones.

El gran tamaño de la tabla es necesario para tratar de reducir el número de colisiones. La mayor parte de la tabla estará almacenada en memoria virtual hasta que sea necesario usarla salvo que el sistema operativo posea memoria física suficiente, lo cual es probable en los entornos en los que la herramienta va a ser usada.

```
typedef struct __attribute__((packed)) {
    node_l *list;
    unsigned short n;
} collision_list;

typedef struct __attribute__((packed)) {
    in_addr_t ip_client_int;
    in_addr_t ip_server_int;
    tcp_seq last_client_seq;
    tcp_seq last_client_ack;
    tcp_seq last_server_seq;
    tcp_seq last_server_ack;
    struct timespec last_ts;
    node_l *active_node;
    node_l *list;
    unsigned short port_client;
    unsigned short port_server;
    unsigned short n_request;
    unsigned short n_response;
    unsigned short deleted_nodes;
} connection;
```

Figura 4.3: Lista de colisiones y estructura de conexión

4.2.2. Conexión

La estructura connection contiene los datos de la conexión a la que pertenecen las peticiones que esta almacena. Entre otros datos importantes guarda la 4-tupla, el timestamp de la ultima interacción con esta conexión y la lista de peticiones.

Connection también almacena el número de peticiones y el de respuestas que hay en la lista enlazada así como cuantas peticiones han sido liberadas.

```
typedef struct node_l node_l;
struct node_l {
    node_l *prev;
    node_l *next;
    void *data;
};
```

Figura 4.4: Estructura node_l

4.2.3. Packet_info

Esta estructura contiene todos los datos que componen una petición o respuesta. A destacar: La marca temporal, la IP origen, el puerto origen, la IP destino, el puerto

destino, su URL (en el caso de la petición), el código y el mensaje de respuesta (en el caso de ser una respuesta), una variable que indica si el paquete es una petición o una respuesta. Esta estructura es usada de forma auxiliar mientras se analiza el paquete antes de decidir si este es válido o no.

En la figura siguiente se muestra el código que define la estructura packet_info.

```
typedef struct {
    struct sniff_ethernet *ethernet;    // The ethernet header
    struct sniff_ip *ip;                // The IP header
    struct sniff_tcp *tcp;              // The TCP header
    unsigned int ip_src;
    unsigned int ip_dst;
    struct timespec ts;
    char url[URL_SIZE];
    char host[HOST_SIZE];
    u_int size_ip;
    u_int size_tcp;
    u_int size_payload;
    u_char *payload;                    // Packet payload
    char response_msg[RESP_MSG_SIZE];
    short responseCode;
    unsigned short port_src;
    unsigned short port_dst;
    short request;
    http_op op;
} packet_info;
```

Figura 4.5: Estructura packet_info

4.2.4. Algoritmo del callback

Como se esbozó en el apartado de estructura general, cada vez que un paquete es leído mediante NDLeeTrazas este es procesado por la función Callback. Primero es necesario comprobar que las cabeceras IP y TCP son correctas, esta tarea la realiza la función parse_packet. En el caso de capturar paquetes desde la interfaz se hace uso de una función alternativa de callback que prepara los datos para poder llamar a la función callback por defecto.

4.2.5. Parse_packet

Mediante un casting se obtienen rápidamente los datos de IP, puerto, tamaños de cabecera y demás datos necesarios para comprobar que el paquete está correctamente formado. Dentro de esta misma función se hace uso del módulo HTTP que recibe el payload TCP para procesarlo y determinar si es una petición, una respuesta o un paquete inválido.

4.2.6. Módulo HTTP

Este módulo convierte el payload TCP en una estructura con los campos del paquete HTTP. Permite trabajar con el paquete HTTP de forma más cómoda y mediante diversas

```
struct _internal_http_packet{
    http_header *headers;
    char *data;
    http_op op;
    char method[32];
    char version[32];
    char uri[2048];
    char host[256];
    int response_code;
    char response_msg[256];
    short has_host;
};
```

Figura 4.6: Estructura http

funciones permite obtener los campos de la estructura. Esta estructura es privada y solo se puede acceder a los campos mediante las funciones.

Entre otras cosas el módulo busca la cabecera host entre las muchas que vienen con la petición HTTP. Esta cabecera no siempre es añadida y por tanto en caso de que no esté disponible se sustituye por la dirección IP del host.

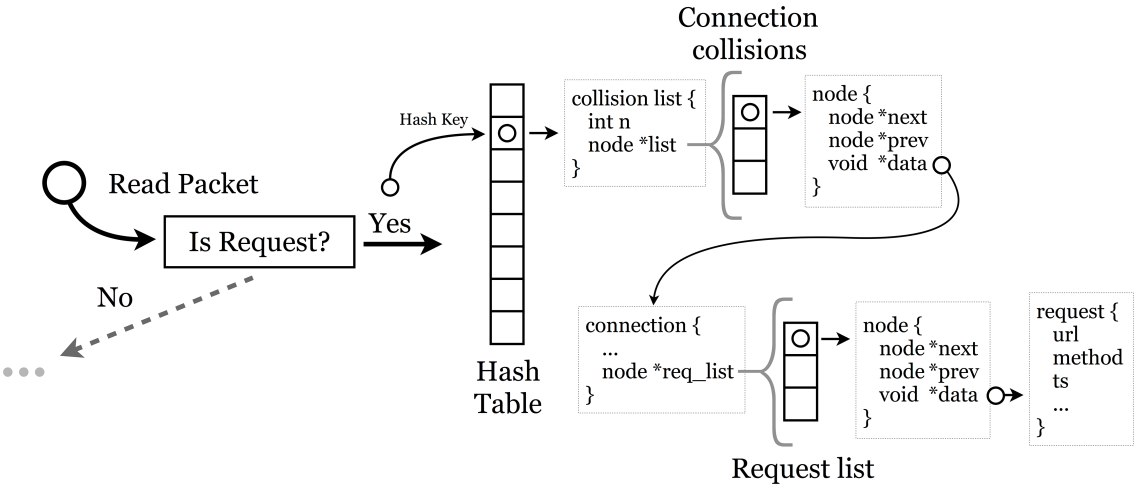


Figura 4.7: Recorrido de un paquete por las distintas estructuras

5 Desarrollo

5.1. Introducción

En este capítulo se explicarán principalmente los problemas surgidos durante el desarrollo del proyecto y de cómo se han afrontado. También se hablará a posteriori de aspectos relacionados con el rendimiento de las distintas partes del programa y algoritmos que no se trataron en el capítulo de Diseño.

También se explicará cómo se han ido introduciendo nuevos requisitos y funcionalidades a lo largo del proyecto y de que forma han afectado a la versión final y al trabajo futuro.

La próxima sección explica el ciclo de vida elegido para el desarrollo de este proyecto y los motivos por los que se ha escogido.

5.2. Ciclo de vida

Debido a que la herramienta se ha utilizado en casos reales de proyectos durante todo el proceso de desarrollo de esta, se ha elegido un ciclo de vida iterativo debido a que es el que mejor se adapta a las necesidades del proyecto. Este ciclo de vida permite el desarrollo de distintos prototipos o versiones funcionales y usables, las cuales, repetimos, se han ido utilizando en ejemplos reales.

A pesar de que la solución del problema es clara y su diseño general es simple y conocido, el desarrollo de esta solución requiere realizar numerosas y distintas pruebas para probar cada una de las partes que lo componen y sus funcionalidades.

Este ciclo de vida tiene la desventaja de requerir un cliente muy involucrado con el desarrollo de la aplicación para que decida si las necesidades del sistema se están viendo satisfechas y los requisitos cumplidos. Dado que el rol de cliente en este caso es el propio

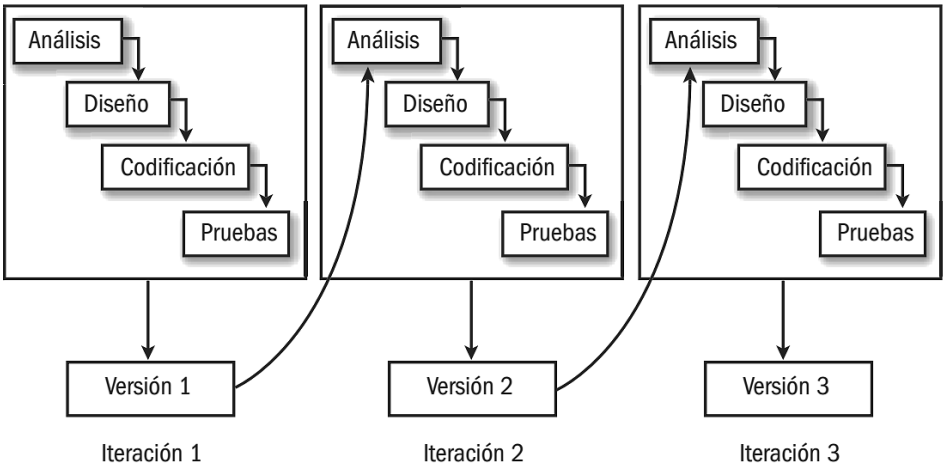


Figura 5.1: Ilustración del desarrollo iterativo

tutor que realiza las pruebas en sistemas reales de pruebas en México, Bahrein u otras localizaciones.

Para este proyecto los requisitos iniciales no van a verse alterados durante el desarrollo, sin embargo sí pueden aparecer otros nuevos que conlleven la realización de modificaciones en el proyecto. Estos nuevos requisitos no sustituyen los anteriores o los eliminan sino que son adicionales y se complementan. Se irán introduciendo en forma de adiciones al anterior prototipo y se verán satisfechas en el ulterior.

El desarrollo empezó con una implementación simple de los requerimientos e iterativamente se mejoraron las versiones producidas hasta completar los requerimientos del sistema. En cada iteración se corrigen problemas, se mejoran los módulos, se añaden nuevas funcionalidades o se rediseñan u optimizan distintas partes del sistema.

En este proyecto se ha seguido con ese desarrollo añadiendo más funcionalidades, mejorando los requisitos iniciales establecidos y cambiando varios pilares de la herramienta para satisfacerlos.

5.3. Cambios en el diseño y procedimientos

Una de los principales requisitos de este proyecto era eliminar las dependencias de librerías externas para aumentar la portabilidad del programa y su estabilidad evitando que incompatibilidades entre versiones de la librería externa y sistemas operativos. Por otro lado era necesario aumentar el rendimiento de la aplicación reduciendo el tiempo de procesado por cada paquete para aumentar la velocidad total del programa.

En esta sección se explican algunos de los cambios de diseños más importantes que se han tenido que afrontar en el desarrollo de aplicación así como procedimientos para satisfacer los requisitos.

5.3.1. Pools de estructuras

La velocidad es el factor más importante a la hora de analizar tráfico y más cuando nos movemos en las cantidades propias de los entornos de alto rendimiento. Por ello mejorar la velocidad era un requisito primordial. Como se ha comentado anteriormente las reservas y liberaciones de memoria durante el callback, es decir por cada paquete, eran demasiadas y suponían un tiempo del que no se disponía. Muchas de estas reservas se hacían para ser liberadas poco después al llegar la respuesta correspondiente. Y de nuevo, esa liberación de recursos se hacía para de nuevo volver a reservar recursos para el siguiente paquete.

Para solventar este problema se decidió crear un sistema de gestión de memoria propio reservando los recursos al inicio de la ejecución del programa. Mientras que en el anterior esquema de recursos únicamente se reservaba la cantidad que se estaba utilizando, en este debemos reservar recursos teniendo en cuenta que puede haber picos donde haya muchas conexiones con muchas peticiones en cada conexión.

Esto significa que debemos estar preparados para procesar ficheros muy densos, con mucha carga de peticiones y conexiones aunque ello suponga utilizar muchos recursos igualmente al procesar pequeños ficheros. En el apartado de Pruebas y Resultados se discute el uso de memoria de esta solución.

Según sea necesario se interactúa con los pools pidiendo variables o elementos y liberándolos, metiéndolos de nuevo en el pool. De esta forma no se realizan reservas de memoria durante la ejecución, resultando en una mayor velocidad. Hay tres tipos de pools que guardan distintas estructuras, a continuación se enuncian:

Pool de nodos genéricos

La estrctura node_l explicada en el apartado de Implementación es utilizada para las listas enlazadas de colisiones y peticiones. El tamaño del pool de estas estructuras es de dos millones de elementos ya que cuando este pool es utilizado alguno de los otros dos también, es decir, un elemento de este pool acompaña o bien a un elemento de tipo connection o bien a una petición.

Pool de conexiones

La estructura connection almacena los datos de la conexión y la lista de peticiones de esta. Las conexiones se almacenan en la lista de colisiones de la celda que le corresponda. Hay un millón de elementos de este tipo queriendo decir que el programa soporta un millón de conexiones simultáneas en un mismo momento. Se considera, en base a las pruebas realizadas, que este tamaño es suficiente.

Pool de peticiones

El tercer y último pool es el que corresponde a las peticiones. De nuevo el tamaño es de un millón, permitiendo que un millón de peticiones en un mismo momento se encuentren esperando su respuesta en la tabla. De nuevo, según las pruebas realizadas este tamaño

es suficiente. En la figura 4.7 del capítulo de Diseño se aprecia el uso de las distintas estructuras.

5.3.2. Tabla hash

Debido a que la tabla hash usada por el disector la ofrecía Glib esta tuvo que crearse desde cero para satisfacer los requisitos de portabilidad. Entre los requisitos de la tabla hash están que se produzcan pocas colisiones, que la función hash sea rápida para que no afecte a la velocidad del programa y que no se llene de datos innecesarios.

Al principio del desarrollo se optó por una tabla hash reservada estáticamente en memoria con 2^{26} elementos, cada uno con una estructura de tipo collision_list, explicada en el apartado de Implementación del capitulo anterior. Esta tabla es suficientemente grande como para mantener un número reducido de colisiones.

Además, el uso inteligente de la tabla hash reduce la cantidad de datos que hay en esta y por tanto las colisiones debido a que en cuanto todas las peticiones de una conexión han sido satisfechas (han sido emparejadas con su correspondiente respuesta) esta conexión es eliminada de la tabla hash. Todo ello sin importar que la siguiente petición pueda pertenecer a esa misma conexión y esta deba ser añadida de nuevo ya que con el uso de los pools de memoria anteriormente explicados no es necesario volver a reservar los recursos. Lo mismo ocurre con la petición individualmente si esta es satisfecha pero quedan más en la conexión. La variable que contenía la petición es devuelta al pool para que pueda ser usada de nuevo más tarde.

Junto a este uso inteligente y como se explica en el siguiente apartado, el recolector de basura se encarga de que no haya datos innecesarios en la tabla.

Como función hash se escogió realizar el XOR, una operación generalmente muy rápida en los procesadores, de los campos que forman la 4-tupla, esto es, la IP y puerto de origen así como la IP y el puerto de destino.

El tamaño de la tabla hash ha variado y se ha estudiado el comportamiento de los datos dentro de esta. En el apartado de Pruebas y Resultados se exponen los distintos números de colisiones según el tamaño de la tabla hash. Además debido al tipo de problema que supone el análisis de tráfico y la clasificación de paquetes es imposible predecir los accesos a la tabla hash, siendo estos totalmente aleatorios y repercutiendo en un coste de tiempo por acceso aleatorio a memoria.

Por otro lado se ha tenido en cuenta que el compilador añade bytes de padding a las estructuras para alinear la memoria haciendo que cada elemento ocupe más. Esto es un problema a la hora de reservar 2^{30} (como es en la versión final) ya que con padding estaríamos hablando de 17GB de recursos, mientras que, al forcar que las estructuras sean packed, es decir, sin padding, el tamaño de la tabla hash se queda en 10GB, una reducción del tamaño bastante significativa.

5.3.3. Recolector de basura

Desde las primeras versiones fue necesario un recolector que buscase datos que ya no eran útiles. En un primer momento el recolector era bastante primitivo pero ahora, con el uso de los pools se vuelve esencial.

En el caso actual el recolector consiste en un hilo dormido que despierta cada 25 segundos para comprobar, no toda la tabla, que llevaría muchísimas iteraciones, sino solo los elementos activos de esta tabla. Para ello hay una lista enlazada de nodos activos. Esta lista, doblemente enlazada, para mayor optimización está ordenada de más reciente a más antiguo, por tanto se recorre en orden inverso, desde la conexión más antigua a la más reciente.

Ni siquiera es necesario recorrer todas las conexiones de esta lista porque en cuanto una de las conexiones no cumpla la condición de llevar más de 60 segundos inactiva el resto tampoco lo cumplen. De esta forma se comprueban únicamente las conexiones justas y necesarias. Los 60 segundos de inactividad se miden comparando el timestamp de la ultima interacción de la conexión contra el timestamp del último paquete procesado globalmente.

Una opción permite al usuario pedir que el programa escriba en un fichero las peticiones que no fueron satisfechas.

5.3.4. Ordenación de la salida

Como se comentó en el capítulo de análisis de requisitos algunas transacciones pueden llegar desordenadas implicando que una transacción anterior, de la misma conexión, podría ser impresa después de una transacción más reciente, tal como la figura 2.3 ilustra.

Esto es debido a que la captura de paquetes se realiza directamente desde la interfaz, sin pasar por toda la capa OSI significando que los paquetes que se suelen ordenar en la capa de transporte pueden estar desordenados.

Para solventar este problema se añadió la opción para forzar que la salida esté ordenada. Esta operación se realiza mediante un buffer de un millón de líneas que al llenarse se ordena en base al timestamp de la petición. Después se imprime y se vacía. De esta forma podemos garantizar que, al menos, la salida estará ordenada en bloques de un millón de líneas. Las posibilidades de que entre bloque y bloque hayan paquetes desordenados es muy pequeña.

5.3.5. Eliminación de las retransmisiones y duplicados

Debido a que la herramienta no reensambla los flujos TCP, no es consciente de ninguna de las retransmisiones ni de los duplicados. Una forma de solucionar esta cuestión es filtrar las transacciones a posteriori, antes de imprimirlas por pantalla.

Lo primero que se realiza es una ordenación de las transacciones en base a la 4-tupla, el número de secuencia y el timestamp de la petición como último criterio. Después se comprueban las transacciones comparando cada una únicamente con la siguiente. Si ambas coinciden en la 4-tupla, y el número de secuencia, la segunda transacción es marcada para ser omitida antes de imprimirla por pantalla. Se decide marcar la transacción en vez de eliminarla del array porque el coste de eliminar la transacción del array supondría crear otro auxiliar.

Después de recorrer todas las transacciones estas se ordenan de la misma forma que en el apartado anterior de ordenación de la salida, es decir, por tiempo de respuesta.

5.4. Inserción de nuevos requisitos

Durante el desarrollo del proyecto se han ido añadiendo nuevos requisitos y funcionalidades entre cada una de las versiones. Algunos de estos son:

- **Más opciones para el filtro BPF**: La opción de no solo concatenar, con un operador AND, un filtro al filtro por defecto, sino reemplazarlo o concatenar otro con el operador OR se añadió durante el desarrollo de la aplicación.
- **Paquetes descartados**: Surgió la necesidad de poder introducir un fichero con los números de los paquetes que interesa descartar.
- **Imprimir la salida del recolector de basura**: También fue fruto de una necesidad real para poder estudiar qué peticiones eran descartadas.

6

Pruebas y resultados

6.1. Método de pruebas

En este capítulo se van a discutir los resultados obtenidos para comprobar que los requisitos han sido satisfechos. Estas pruebas han sido realizadas en distintos entornos dependiendo de la prueba. Para las pruebas mencionadas realizadas con los ficheros de la tabla 1 y 2 se ha utilizado una máquina con doce núcleos Xeon a 2,67Ghz y 70GB de RAM. Los ficheros consisten en tráfico HTTP y HTTPS capturado y guardado en ficheros PCAP. La herramienta ignora el tráfico HTTPS y durante los tests de estos ficheros se ha filtrado el tráfico por el puerto 80.

Los ficheros de la primera tabla son pequeñas muestras usadas únicamente para los test de precisión y asegurar que la información imprimida coincide con la dada por otros programas bien probados y fiables como Bro **[8] [9]**, Tshark u otros propios del laboratorio HPCN.

Por otro lado, los ficheros de la segunda tabla han sido usados para tests intensivos orientados a medir el uso de la tabla hash, la memoria y los pools de estructuras así como el tiempo de procesamiento.

Una serie de scripts bash han sido utilizados para procesar la información de los logs de la herramienta y crear los gráficos. Estos no modifican la información sino que seleccionan los campos útiles y reformatean la información.

Filename	Size (MB)	Total Packets	Ratio of Processed Packets	Time (secs)	Packets / sec
REAL2min	182	221,802	100 %	2	110,901
REAL2min_sinduplicados	91	110,982	100 %	1	110,982

Tabla I

Cuadro 6.1: Información de los ficheros

Filename	Size (GB)	Total Packets	Ratio of Processed Packets	Time (min)	Packets / sec
traza.20140508	200	775,398,672	7 %	8.5	1,505,628
traza.20140509	221	847,033,287	7 %	9	1,540,060
traza.20140601	63	268,811,478	7 %	2.6	1,609,626
traza.20140607	84	357,130,531	6.7 %	4	1,378,882
traza.Oficinas	83	306,456,202	0.06 %	2.1	1,939,596

Tabla II

Cuadro 6.2: Información de los ficheros

6.2. Precisión de los resultados de la herramienta

A pesar de que nuestro primer requisito es la velocidad nuestra mayor preocupación es mantener la precisión de los datos entregados para permitir realizar análisis posteriores con exactitud. Aquí se exponen algunas pruebas para comprobar que la información dada por la herramienta es precisa.

6.2.1. Tiempo de respuesta

El tiempo de respuesta es una de las métricas más interesantes para medir la calidad de servicio. Este campo, dado en la salida del disector, es muy interesante para realizar gráficos CCDF (Complementary Cumulative Distribution Function). Para realizar la prueba se analizó el fichero Real2min_sinduplicados (Tabla I) mediante el disector y la herramienta ProcesaConexiones. Este herramienta permite analizar el tráfico TCP y reconstruir los flujos para dar información en profundidad de los flujos.

El CCDF nos dice la probabilidad de obtener un tiempo de respuesta mayor que el dado. Como la figura 6.1 muestra no hay una pérdida significativa de precisión en esta métrica.

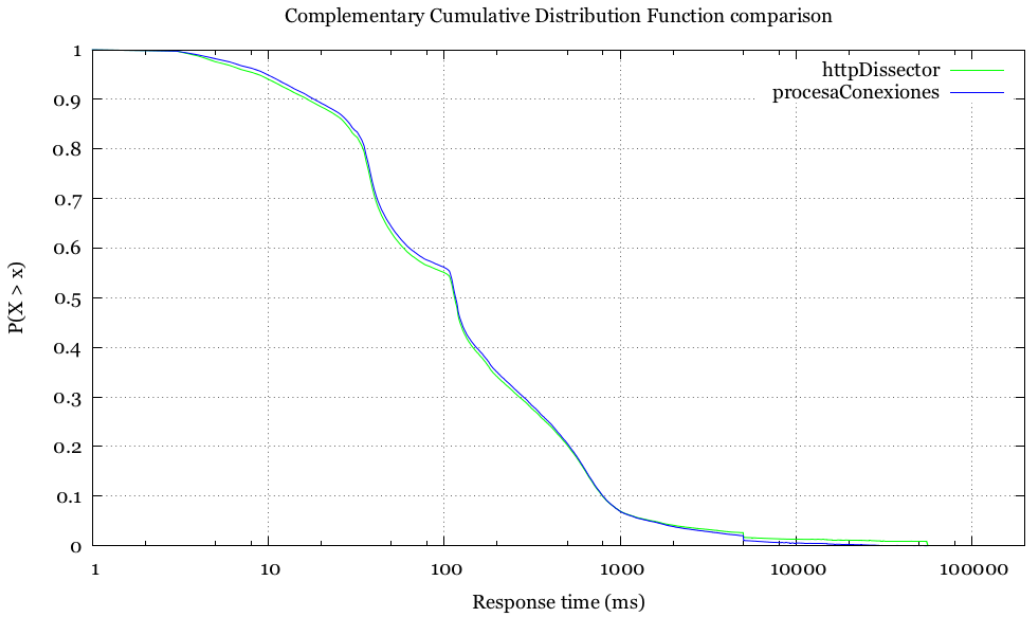


Figura 6.1: Gráfico mostrando el CCDF del tiempo de respuesta calculado por procesaConexiones y httpDissector

6.2.2. Perdida potencial de información en la URL de la petición

Debido al hecho de que solo se hace uso del primer paquete de la petición HTTP (porque no se reensambla el flujo TCP), la URL puede estar truncada si el paquete es mayor que la MTU (1518 bytes). Este apartado explica si esto es posible y con qué frecuencia.

En el RFC 2616 (Hypertext Transfer Protocol HTTP/1.1) sección 3.2.1 **[5]** dice que ."El protocolo HTTP no establece a priori un límite en la longitud de la URI. Los servidores deben ser capaces de manejar la URI de cada recurso que sirvan". La verdad es que la mayoría de los navegadores de Internet soportan 80.000 caracteres de media y que el límite de del Servidor Apache es de 8192 caracteres. Algunos navegadores como Internet Explorer tienen un límite de 2048 caracteres. **[6]**

Las URLs largas no son buenas si se pretende que estas sean indexadas por los motores de búsqueda ya que el protocolo de los mapas webs (sitemaps **[7]**) tienen un límite de 2048 caracteres para la URL.

Se realizó una prueba con el fichero Real2min_sinduplicados (Tabla I) el cual contiene 50.000 peticiones. La prueba se realizó comparando las URLs mostradas por la salida del disector con las mostradas por la herramienta Bro. El 99,9757 % de las URLs coincidían. Se realizó otra prueba eliminando los datos pertenecientes a la query, esto es, los parámetros adicionales de la URL, y el porcentaje se incrementó.

Los experimentos del disector de tráfico deberán probar que se cumplen los requisitos de velocidad y uso eficiente de recursos que se exigían.

6.2.3. Códigos de respuesta

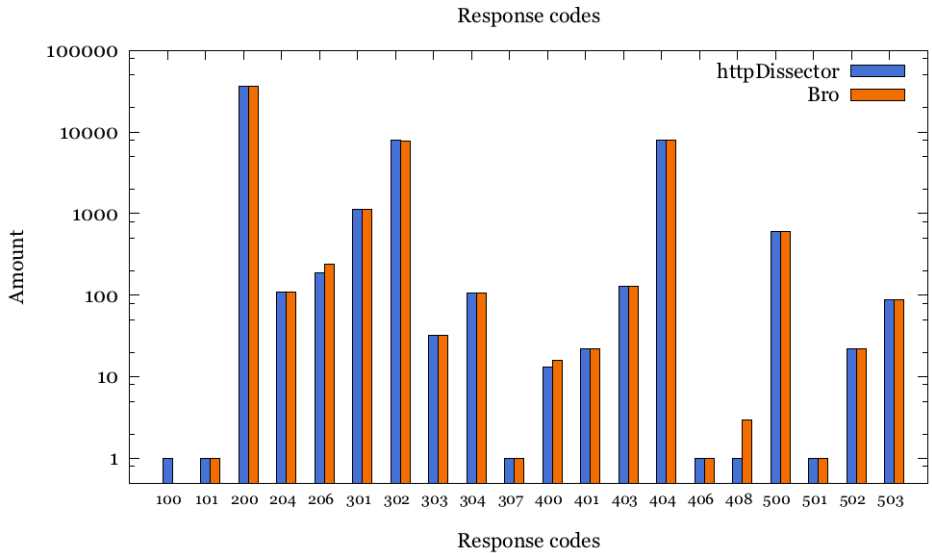


Figura 6.2: Comparación de la contabilización de los códigos de respuesta entre el http-Dissector y Bro

Evaluar los códigos de respuesta es una forma de encontrar errores de cliente (códigos 4xx) extraños como, por ejemplo, un gran número de códigos 404 (Not Found) que podrían

significar enlaces muertos en una página web. Los códigos 5xx, errores del servidor, son también interesantes.

Se realizó una prueba, con el mismo fichero que en las anteriores pruebas y realizando la comparación con la aplicación Bro. Como la figura 6.2 muestra la cantidad de cada código de respuesta es prácticamente idéntica y la diferencia es debida a alguna pérdida de peticiones cuando estas vienen seguidas en una petición HTTP. Una comparación similar se puede realizar con el resto de atributos HTTP que imprime el disector en su salida como pueden ser el método de la petición HTTP o el mensaje de respuesta.

6.3. Velocidad de procesamiento

La velocidad de procesamiento es crucial para obtener resultados de grandes cantidades de datos. Cuando se trata de analizar ficheros de capturas es fundamental hacer uso de un sistema RAID que a base de distribuir o replicar los datos en múltiples discos duros y usar ese grupo de discos como una única entidad permita obtener buenas velocidades de lectura, de tal forma que los discos duros no supongan un cuello de botella para el programa.

Otro factor importante es la densidad de los ficheros, en el sentido de que aunque un fichero únicamente tenga tráfico HTTP, puede haber mayor o menor cantidad de peticiones y respuestas HTTP ya que las transacciones HTTP pueden ser mayores. No es lo mismo una transacción para transmitir una imagen pesada, que conlleva muchos paquetes en el flujo TCP que otra con tan solo el código HTML de una web. El porcentaje de paquetes procesados de la tabla indica qué cantidad de paquetes eran peticiones y respuestas. No obstante, habría que ver cual es la densidad normal a esperar en casos reales debido a que el primer paquete de la petición y el primero de la respuesta constituyen muy poco tráfico del total y por tanto es normal que el porcentaje de paquetes procesados sea bajo.

En la tabla II se puede apreciar que el tamaño de los ficheros es mayor (que el de la tabla I, donde están en MB) y por tanto son ficheros más aptos para pruebas de velocidad. No obstante mediante otro sistema RAID se podrían haber conseguido mejores resultados ya que la tasa en MB/s no fue mayor de 400MB/s en ninguna de las pruebas mientras que en otras máquinas hemos conseguido velocidades de 700MB/s.

Así mismo, para aislar totalmente el efecto de los discos, se han realizado pruebas leyendo los paquetes tras haberlos cargado en memoria. La máquina en cuestión utilizada para realizar esta prueba es muy similar con 12 núcleos Xeon pero con la diferencia de que posee un sistema RAID de discos más rápido y 128GB de RAM.

En la tabla III se muestran los detalles del fichero leído desde fichero. Este está formado por el tráfico de los laboratorios de la EPS de una serie de días de Marzo. Se filtró el fichero para conservar únicamente el tráfico HTTP.

Se cargaron 44 millones de paquetes en memoria y se enviaron a una cola fifo 10 veces mediante un programa encargado de esta tarea. El disector, leyendo de esta cola consiguió una tasa de 4.6 Gbps. Como referencia, si estos paquetes se leyeran sin ser procesados desde la misma cola fifo, se conseguiría una tasa de 14 Gbps, unos 6.2 mpps.

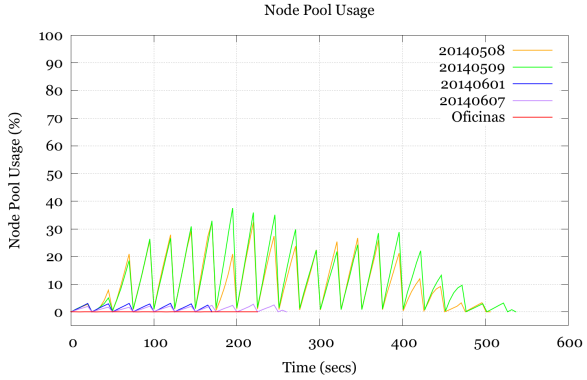
Filename	Size (GB)	Total Packets	Ratio of Processed Packets	Time (secs)	Packets / sec
httpLabs (De fichero)	13	44,011,250	2 %	18	2,588,897

Tabla III

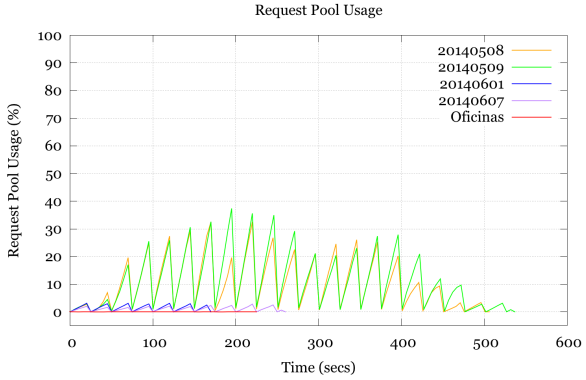
Cuadro 6.3: Información de los ficheros

6.4. Uso de memoria

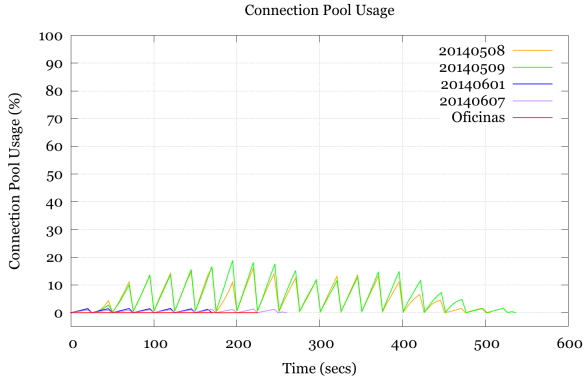
Esta sección discute el rendimiento en términos de recursos de memoria por parte de la herramienta y el comportamiento interno de los pools de estructuras. En la figura 6.3 se puede apreciar la memoria reservada por defecto, al inicializar el programa, alrededor de 15GB. Sin embargo, ninguna de las pruebas alcanza a utilizar ni siquiera la mitad de esta memoria.



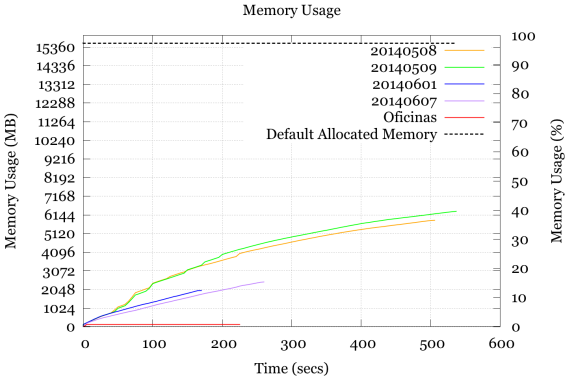
(a) Node Pool



(b) Request Pool



(c) Connection Pool



(d) Uso de Memoria

Figura 6.3: Uso de los pools de estructuras y uso de memoria

Es necesario precisar que aunque el uso de memoria parezca incrementarse en el tiempo no significa que el programa, en un segundo determinado esté usando esa cantidad de recursos que se indica aunque sí los tenga reservados.

Se puede apreciar que los ficheros que requieren un mayor uso de los pools también lo hacen de memoria. El gráfico de memoria, sin embargo, no refleja las bajadas producidas por el recolector de basura. Cuando se reserva la memoria al principio del programa el sistema operativo mantiene en memoria virtual todos los recursos hasta que se utilizan.

A medida que las estructuras se extraen del pool el uso de memoria crece, pero cuando estas se devuelven al pool la memoria residente (Resident Set Size) no decrece. Esto es debido a que el sistema operativo no es consciente del uso que le damos a los recursos reservados y piensa que todavía están siendo utilizados.

El peor caso para este esquema sería la presencia de un pico de consumo de los pools al principio de la ejecución que supondría que los 15GB reservados estarían en memoria residente. El sistema operativo es responsable, con sus estrategias, de evaluar si los recursos deben volver a la memoria virtual o no.

Forzar ese comportamiento es un proceso complejo y totalmente inútil ya que en un entorno de alto rendimiento es esperable disponer de esta cantidad de RAM y de que en la máquina no haya procesos compitiendo por los recursos, ya sean de disco, procesador o memoria para conseguir un resultado óptimo. Como puede apreciarse, el recolector de basura cumple con su función y el uso de los pools no supera en ningún caso el 50 %.

6.5. Colisiones en la tabla hash

Basándonos en el problema de hashing del cumpleaños, la probabilidad de colisión se incrementa a medida que el número de elementos introducidos en la tabla crece. El gráfico A de la figura 6.4 muestra unos casos de probabilidad de colisión dependiendo del tamaño de la tabla hash. Las curvas han sido calculadas haciendo uso de la ecuación 6.1 con k siendo el número de elementos insertados y N el tamaño de la tabla hash.

$$Probabilidad\ de\ colisiones = 1\ -\ e^{\frac{-k\ (k-1)}{2N}} \tag{6.1}$$

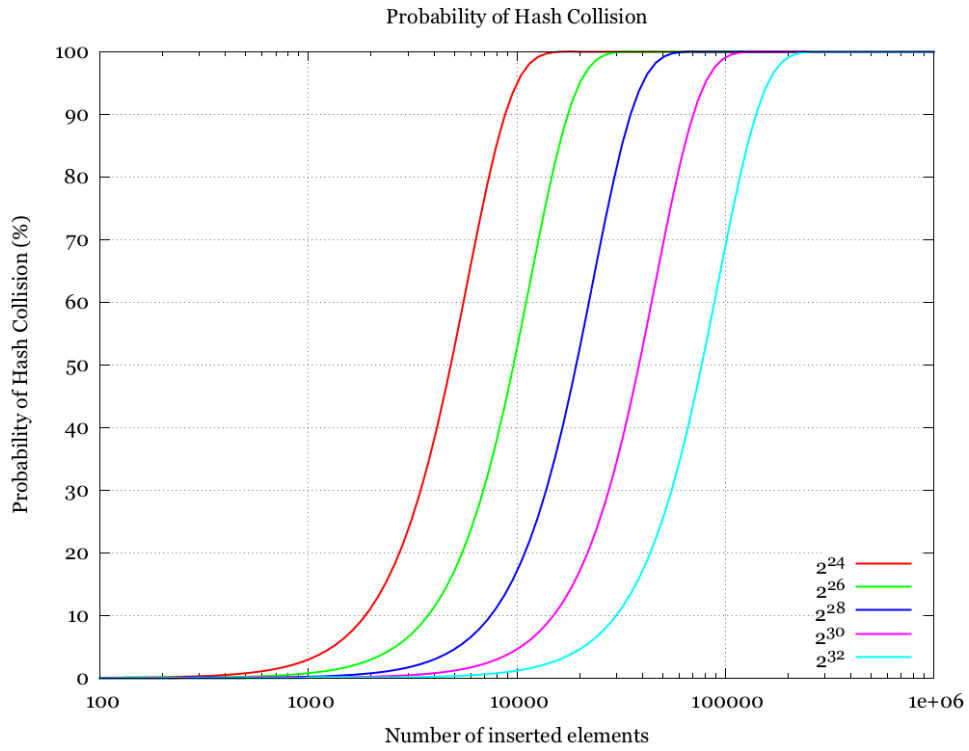
Es una apreciación teórica considerando que en la tabla del disector los elementos son eliminados de la tabla cuando se cumplen las condiciones adecuadas para ello. Esto no se tiene en cuenta en estos gráficos pero es una referencia útil a tener en cuenta.

La imagen B de la figura 6.4 muestra el número de colisiones para cada fichero. Las curvas indican el número teórico de colisiones en función del tamaño de la tabla hash. Esta curva ha sido calculada con la ecuación 6.2 en la que m es el tamaño de la tabla hash y n el número de elementos a insertar.

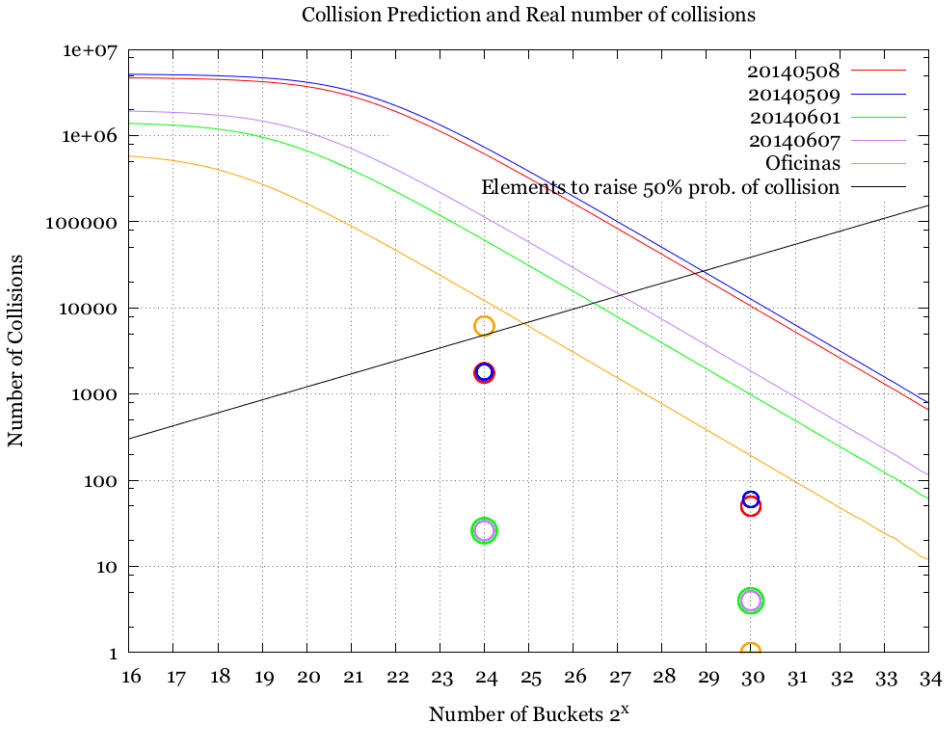
$$Colisiones = n - m(1 - (\frac{m-1}{m})^n) \tag{6.2}$$

Por tanto, las curvas representan el caso peor mientras que los círculos indican el número real de colisiones para cada fichero con una tabla hash de tamaño 2^{24} y otro de 2^{30}. La diagonal negra indica el número teórico de elementos que es necesario introducir para alcanzar un 50 % de probabilidad de colisiones (Comparar con la imagen A).

Como puede observarse, el número de colisiones disminuye enormemente cuando se hace uso de una tabla mayor, lo cual no representa un problema de recursos. Independientemente del tamaño de la tabla hash, el número de elementos introducidos es siempre el mismo, así que a pasar de tener una tabla hash muy grande, la mayor parte estará en memoria virtual y los accesos, independientemente del tamaño de la tabla hash, seguirán siendo aleatorios.



(a) Probabilidad de colision



(b) Colisiones en la tabla hash

Figura 6.4: Colisiones de la tabla hash

6.6. Portabilidad

Se han realizado pruebas de portabilidad en las siguientes plataformas:

- **Mac OS X 10.9 Mavericks**: Se ha instalado libpcap mediante el gestor de paquetes homebrew.
- **Ubuntu 12.04 LTS**: Se han instalado los paquetes libpcap-dev.
- **OpenSuse 11.1 SP1**: Se han instalado los paquetes de desarrollo correspondientes de libpcap.
- **Fedora 18**: Se han instalado los paquetes de desarrollo correspondientes de libpcap.

El funcionamiento ha sido el correcto en los distintos sistemas operativos.

6.7. Conclusiones

Las pruebas demuestran que la herramienta es capaz de operar con ficheros propios de entornos de alto rendimiento con una buena velocidad y con datos precisos. Aprovecha los sistemas de alto rendimiento donde es ejecutado. Demuestra también que se ha investigado para tratar de optimizar cada punto del programa y justifica las medidas tomadas durante el desarrollo.

7
Conclusiones

El punto fuerte de este proyecto radica en su uso en proyectos y entornos reales donde satisface necesidades reales y resuelve problemas concretos. El disector de tráfico HTTP es muy estable y satisface los requisitos establecidos al principio del proyecto. Se adapta correctamente a los entornos de alto rendimiento.

La aplicación es rápida, usa inteligentemente los recursos de memoria y aprovecha al máximo la velocidad de los discos a la hora de procesar ficheros. Además se ha realizado un esfuerzo notable para aumentar la portabilidad y reducir las dependencias para su uso en distintas plataformas.

Hay aspectos que siempre se podrán mejorar del disector y que quedan para el trabajo futuro, estos pueden ser la velocidad de procesamiento o mejorar aún más si cabe el uso de los recursos de memoria. Dado que es utilizado en proyectos vivos no cabe duda de que surgirán nuevas necesidades que requieran realizar cambios y mejoras en el programa. No obstante los objetivos planteados por el proyecto se han cumplido en su totalidad.

En resumen, este proyecto resuelve un problema actual y real y ofrece un sistema de análisis rápido y eficaz del tráfico web para entornos de alto rendimiento.

8

Trabajo futuro

Se insiste en que este proyecto está en continuo desarrollo por su uso en varios proyectos del laboratorio HPCN de la UAM y por tanto cabe espacio para añadir nuevas funcionalidades y mejoras. En este capítulo se nombran varias de ellas, de las cuales algunas ya se están estudiando y trabajando en ellas en el momento de escribir este documento.

8.1. Aumento de la velocidad de procesamiento

8.1.1. Multiples instancias

Actualmente para aumentar la velocidad de procesamiento se está trabajando en un sistema que reparte los paquetes entre varias instancias del disector para aumentar la tasa de paquetes procesados. El sistema consiste en encolar los paquetes en varias colas fifo repartiendo lo más equitativamente posible las conexiones mediante una función hash. Mientras, las instancias de los disectores procesan los paquetes de las colas correspondientes. Se han realizado pruebas, con el fichero en memoria de la misma forma que se comentó en el apartado 6.3 de Velocidad de procesamiento en el capítulo de Pruebas y Resultados.

Tomando como referencia el máximo de 14 Gbps que supone leer los paquetes sin aplicar ningún procesamiento tenemos que, con una instancia se obtiene una velocidad de 4.66 Gbps, con dos instancias 6.7 Gbps y finalmente con tres instancias 9.3 Gbps y 4 millones de paquetes por segundo. El reparto de los paquetes y conexiones es bastante equitativo. A día de hoy las pruebas realizadas son básicas y orientativas y se continua trabajando en esta línea.

8.1.2. Mejorar NDLeeTrazas

Trabajando en grupo con compañeros del laboratorio estamos tratando de desarrollar una versión optimizada de esta librería que supone el principal cuello de botella a la aplicación debido a la forma en que procesa los paquetes y al gran número de funcionalidades que ofrece que puede ser reducido en favor de ganar velocidad.

Con la primera versión de NDLeeTrazas mejorada, aún en desarrollo, se han conseguido, con una instancia de httpDissector, procesar a 8.7 Gbps, es decir, casi tanto como con tres instancias de la versión comentada en este trabajo. Así mismo con dos instancias se han conseguido 12.84 Gbps y 5 millones de paquetes por segundo, y finalmente con tres instancias se llega al máximo de 14.38 Gbps y 6 millones de paquetes por segundo. De nuevo se advierte de que estas pruebas son primerizas y sencillas pero se continua trabajando en esta línea tratando de crear una versión de NDLeeTrazas que ofrezca las funcionalidades que necesita el disector.

8.1.3. Muestreo de paquetes

Se pretende aplicar un algoritmo de muestreo que en función de la media simple móvil determine cuando es adecuado empezar a procesar los paquetes y cuando descartarlos. El momento más adecuado para procesarlos es cuando hay una gran cantidad de peticiones por segundo. De esta forma, aumentamos la tasa gracias a que descartamos más paquetes, todo esto tratando de mantener la precisión.

Es necesario realizar un estudio de los distintos algoritmos y realizar pruebas con distintos valores de los parámetros de este.

8.1.4. Uso de índices

Otra opción que se plantea para aumentar la tasa es, habiendo preprocesado los ficheros en una primera pasada determinando qué paquetes son útiles para el procesado y que nos permiten mantener la precisión. Estos paquetes se anotarían en un índice. De esta forma se puede realizar una segunda pasada saltando en el fichero a las posiciones donde se encuentran los paquetes útiles y aumentar la velocidad de procesamiento del sistema.

8.2. Más funcionalidades

8.2.1. HTTPS

Una funcionalidad a estudiar y desarrollar consistiría en descifrar el tráfico HTTPS mediante las claves privadas para poder realizar un estudio sobre este. Para ello es necesario utilizar librerías externas y comprobar si es posible obtener una buena tasa de procesamiento a pesar del tiempo extra que conlleva descifrar los paquetes.

Bibliografía y Referencias

[1] **James F. Kurose y Keith W. Ross** Redes de computadoras : Un enfoque descendente basado en Internet.

[2] **Libpcap** : `http://www.tcpdump.org/pcap.html`

[3] **BPF** : `http://www.tcpdump.org/papers/bpf-usenix93.pdf`

[4] **GLib** : `https://developer.gnome.org/glib`

[5] **W3C/MIT, "Hypertext transfer protocol** : `http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.2.1`

[6] **Boutell.com, "What is the maximum length of a url?"** : `http://www.boutell.com/newfaq/misc/urllength.html`

[7] **Sitemaps.org, "Sitemaps xml format,"** : `http://www.sitemaps.org/protocol.html`

[8] **bro.org, "The bro network security monitor"** : `http://www.bro.org`

[9] **V. Paxson, "Bro: A system for detecting network intruders in real- time"** : `https://www.usenix.org/legacy/publications/library/proceedings/sec98/fullpapers/paxson/paxson.pdf`