Mejoras en la identificación de tráfico de aplicación basado en firmas

Néstor Santolaya, Eduardo Magaña, Mikel Izal y Daniel Morató
Universidad Pública de Navarra[1]
Departamento de Automática y Computación
Campus Arrosadia, 31006 Pamplona
Email: eduardo.magana@unavarra.es

***Abstract*— Traffic identification has been based traditionally on transport protocol ports, associating always the same ports with the same applications. Nowadays that assumption is not true and new methods like signature identification or statistical techniques are applied. This work presents a method based on signature identification with some improvements. The use of regular expressions for typical applications has been studied deeply and its use has been improved in the aspects of percentage identification and resources consumption. On the other hand, a flows-record structure has been applied in order to classify those packets that do not verify any regular expression. Results are compared with the open-source related project L7-filter, and the improvements are presented. Finally, detailed regular expressions for analyzed applications are included in the paper, especially P2P applications.**

I. INTRODUCCIÓN

Un tema de elevado interés en la monitorización de redes de datos es la identificación del tráfico que circula por ellas con aplicaciones en muy diversos campos. La identificación consiste en averiguar a qué aplicación pertenece determinado tráfico de la red, pudiendo llevar esta indentificación a granularidad de flujos o incluso de paquetes. Tradicionalmente esta labor se realizaba comprobando el número de puerto del nivel de transporte correspondiente por el que se había enviado o recibido cada paquete, ya que cada aplicación se encontraba habitualmente asociada y de manera unívoca a un identificador de puerto (los denominados como "puertos bien conocidos") [1] . Esta identificación es de utilidad en auditorías de red (conocer los servicios demandados por los usuarios), control de tráfico (poder bloquear servicios no deseados o incluso aspectos avanzados de seguridad), y calidad de servicio (poder priorizar unas aplicaciones frente a otras).

Sin embargo, en la actualidad, la especialización del software de comunicaciones y las restricciones impuestas por proxys y firewalls han hecho que la identificación basada en puertos no sea fiable. Se están imponiendo nuevas formas de proceder en las que cualquier programa se puede camuflar a nivel de red para evitar firewalls, suplantar identidades o enviar código malicioso. Ese camuflaje lo consiguen a base de utilizar puertos asociados a otras aplicaciones bien conocidas y que a efectos prácticos les permitan pasar inadvertidas ya que los controles de acceso se han basado tradicionalmente en el filtrado por puertos [2] . También se basan en montar túneles HTTP que permite atravesar proxys y firewalls sin mayores problemas [1] . Para complicarlo más aún, en el caso de aplicaciones P2P el problema es que cada instancia del programa puede utilizar un puerto diferente para de nuevo evitar estas medidas de control, con lo que se hace más complicada todavía la identificación correcta del tráfico basándonos únicamente en los identificadores de puertos [3] .

Existen otras posibilidades para la identificación de tráfico. Por un lado tenemos soluciones basadas en firmas, que consisten en buscar determinados patrones en el contenido de los paquetes analizando los propios datos generados por el nivel de aplicación [4] . Otra posibilidad es la de decodificar los datos del paquete para comprobar si sigue el flujo de determinado protocolo asociado a cierta aplicación [5] . Incluso se pueden encontrar combinaciones de ambas metodologías [6] . La ventaja de estos métodos es su elevada precisión consiguiendo elevados porcentajes de identificación correcta. Sin embargo, estos métodos introducen mucha sobrecarga debido al análisis detallado que tienen que hacer de los datos encapsulados en cada paquete. Además estos métodos no pueden hacer nada contra aquellas aplicaciones que encripten sus datos aunque en la actualidad estas aplicaciones son minoría [3] .

Por otro lado, recientemente han aparecido propuestas de identificación basadas en estadísticos [2] que si bien no proveen tanta precisión en los resultados su coste computacional es mucho menor y por tanto de aplicación en redes de alta velocidad. Además no se ven afectadas por la encriptación de los flujos por lo que parece un campo por el que se puede apostar en el futuro.

La identificación del tráfico de red a nivel de aplicación se ha aplicado principalmente a los firewalls y más concretamente a la detección del protocolo HTTP. Este tipo de identificación constituye sin duda un paso más en la evolución de los sistemas firewalls, ya que en este caso analizan todo el paquete a nivel de aplicación o, lo que es lo mismo, controlan no sólo los puertos o las sesiones, sino el protocolo que se utiliza para la comunicación, evitando que puedan falsearse servicios. Por ejemplo, sería posible prohibir el acceso HTTP independientemente de que el

[1] Este trabajo ha sido financiado por el Proyecto Integrado Evergrow (FP6-IP-001935) y STREP Moment (FP7-STREP-0215225) de Programas de la Unión Europea.

servicio HTTP estuviera levantado en el puerto 80 o en el puerto 145, ya que el firewall analizaría el protocolo de los paquetes y al ver HTTP bloquearía la conexión.

Este trabajo se centra en el estudio de un sistema de identificación de tráfico basado en firmas. Si bien es de los métodos más empleados para la identificación de tráfico, no existen propuestas abiertas que permitan replicar su implementación para aplicarlo a necesidades concretas ni tampoco resultados exhaustivos de identificación [2] [4] .

El trabajo se organiza como sigue. En la siguiente sección se introducen trabajos anteriores en identificación de tráfico basado en firmas. A continuación se introduce el funcionamiento de las expresiones regulares como núcleo de la identificación basada en firmas. En la sección cuarta se presenta la arquitectura del sistema desarrollado. En la sección quinta y sexta se exponen respectivamente las trazas de tráfico utilizadas y la mejora en expresiones regulares. En la sección séptima se presenta el análisis del sistema propuesto. Para finalizar se presentan las conclusiones del trabajo.

II. Estado del arte en indentificación de tráfico basado en firmas

En la literatura se utilizan sistemas de identificación de tráfico basado en firmas. Se trata de buscar cadenas específicas de texto o datos binarios en el payload de los paquetes. Estas cadenas están asociadas unívocamente al protocolo de la aplicación y pueden ser palabras clave, comandos, opciones o cualquier otro contenido identificable.

En [7] parten de la documentación de los protocolos y trazas de tráfico real para identificar las firmas a buscar en los datos de nivel de aplicación, centrándose en aplicaciones P2P. Evalúa características de precisión (falsos positivos y falsos negativos), escalabilidad (complejidad) y fortaleza (a pérdidas, reordenamiento, etc.) de las firmas encontradas. La búsqueda de firmas se basa en el uso de expresiones regulares. Sin embargo, las firmas resultantes no se presentan en el trabajo.

En otro trabajo [8] se realiza un estudio de identificación basada en firmas sobre un periodo de 2 años y utilizando tráfico real. Sin embargo, las firmas utilizadas son muy simples, usando por ejemplo la búsqueda de la cadena "GNUTELLA" para identificar los paquetes que corresponden a la aplicación P2P gNutella. Además se compara su efectividad con heurísticos basados en direcciones IP y puertos. No analizan la posibilidad de falsos positivos por lo que el análisis no es completo.

Para aplicar disciplinas de calidad de servicio a las diferentes aplicaciones, en [4] utilizan identificación basada en firmas. No entran en detalles de los procedimientos utilizados en cuento a firmas y se centran en añadir características estadísticas a la identificación.

En cuanto a propuestas de software libre existentes, el firewall de nivel de aplicación más conocido es el Application Layer Packet Classifier for Linux (L7-filter) **[7]**. El L7-filter es un clasificador de protocolos basado en Netfilter/IPtables de Linux, el cual identifica paquetes a nivel de aplicación mediante expresiones regulares. Este proyecto soporta una gran variedad de protocolos desde HTTP a malware pasando por diversos tipos de P2P e incluso identificación de ficheros. Para cada uno de ellos aporta su correspondiente expresión regular y los clasifica según la precisión en la identificación y la velocidad de la misma. L7-filter trabaja aplicando expresiones regulares paquete a paquete teniendo en cuenta flujos mantenidos por Netfilter. Además corre a nivel de kernel junto a Netfilter/IPtables por lo que es complejo aplicar su código para otras tareas diferentes a la original.

En este trabajo se realiza una optimización de las expresiones regulares para identificación de aplicaciones, se presentan mejoras en el motor de búsquedas y se detallarán resultados exhaustivos comparando los mismos con los que obtiene la aplicación L7-filter.

III. Expresiones regulares

Las expresiones regulares son una serie de caracteres que forman un patrón, normalmente representativo de otro grupo de caracteres mayor, de tal forma que podamos comparar el patrón con otro conjunto de caracteres para ver las coincidencias [10] . La identificación de paquetes basada en firmas utiliza las expresiones regulares como base fundamental.

En nuestro caso, según esta definición, las expresiones regulares son cadenas de caracteres que agrupan posibilidades en torno a la búsqueda de un determinado patrón de un protocolo dentro de un contenido mayor, que en nuestro caso corresponde al payload del paquete o de los paquetes a analizar. Esto significa que mediante una expresión regular bien definida podemos realizar la búsqueda de cualquier cadena que queramos, como un número de teléfono del que sabemos parte en una guía telefónica, buscar en una página web algún dato o palabra del que sabemos parte de su contenido, o incluso, como es nuestro caso, encontrar patrones previamente definidos en los paquetes que atraviesan la red.

En realidad, las expresiones regulares son la base de programación de cualquier tipo de búsqueda, ya sea simple o avanzada, lo que desemboca en sus numerosas aplicaciones. En nuestro caso, las expresiones regulares son la herramienta que se necesita para agrupar todas las opciones de búsqueda del patrón. Por ejemplo, en el caso real del protocolo gNutella, se puede decidir que cualquier paquete que contenga al comienzo del mismo una de las dos siguientes líneas tiene una alta probabilidad de pertenecer al protocolo gNutella:

gnutella
get /uri-res

Las expresiones regulares consiguen agrupar estas dos cadenas de búsquedas totalmente diferenciadas en una sola solución de búsqueda. A la vez permiten aplicar opciones de búsqueda, como que la cadena que se desea encontrar sólo sirve si se encuentra al comienzo del paquete, o incluso definir un carácter específico o una cantidad variable de caracteres que se pueden encontrar tras la palabra gNutella, como el carácter hexadecimal \x20 o el \x2F. Con todas estas opciones, la expresión regular del protocolo gNutella quedaría de la siguiente forma:

^(gnutella[\x20\x2f]|get /uri-res)

En esta expresión regular se pueden encontrar dos partes diferenciadas. El carácter "|" significa que para que la expresión se cumpla, basta con que sea cierta una sola de las dos opciones que se sitúan a su izquierda y derecha. En este caso, las dos opciones son las que ya hemos comentado:

gnutella[\x20\x2f]
get /uri-res

El carácter "^" que aparece al comienzo indica que todo lo que le sigue debe estar al comienzo del paquete, por lo que no nos servirá un paquete que tenga una de estas dos cadenas en una posición diferente.

IV. ARQUITECTURA DEL SISTEMA

A. Introducción

La aplicación desarrollada para la identificación de tráfico de red a nivel de aplicación mediante el uso de firmas se ha denominado XePI.

El programa tiene dos tipos de funcionamiento, en vivo y con trazas. El modo de funcionamiento en vivo toma el control de la tarjeta de red, captura los paquetes que llegan en tiempo real y los va analizando según se capturan. El otro modo de funcionamiento es el análisis de trazas previamente capturadas, funcionalidad útil para su evaluación con un tráfico controlado. Mediante este modo se puede realizar diferentes búsquedas sobre las trazas según lo que interese estudiar en cada momento.

B. Identificación de paquetes

Una vez que se ha identificado un paquete mediante una de las firmas de la biblioteca de expresiones regulares, lo que realiza el sistema es guardar un registro de paquetes detectados. De esta forma, cuando se encuentren paquetes que no han sido identificados por ninguna expresión regular, el sistema es capaz de clasificar estos paquetes por similitud con el historial de los paquetes que se han detectado por expresión regular previamente. En el esquema de la Figura 1 se muestra el proceso que sigue cada paquete que se analiza.

El procesado se realiza a nivel de usuario con la flexibilidad que eso supone para el desarrollo de aplicaciones a medida. A diferencia de L7-filter, todos los paquetes se comprueban contra las firmas de los diferentes protocolos a identificar y en caso de no verificar ninguna se acude a utilizar el historial de flujos que se explicará posteriormente. Las expresiones regulares están optimizadas de manera que la capacidad de proceso necesario se reduce con respecto a la de L7-filter sin perder porcentajes de identificación.

Una vez que se detecta un paquete por expresión regular, los puertos y las IPs de dicho paquete (datos característicos de cada flujo) quedan almacenados en una estructura de datos que se comentará posteriormente en detalle. Dichos flujos se almacenan a su vez junto al timestamp del paquete y el identificador del protocolo al que pertenece dicho paquete. De esta forma se puede identificar el resto de paquetes del mismo flujo sin que verifiquen la expresión regular, sólo asumiendo que todos los paquetes de un mismo flujo pertenecen a la misma aplicación.

Si se reciben paquetes del mismo flujo se actualiza el timestamp. Este timestamp se utilizará para aplicar un temporizador que permita liberar las estructuras de flujos que no han generado tráfico en los últimos minutos.

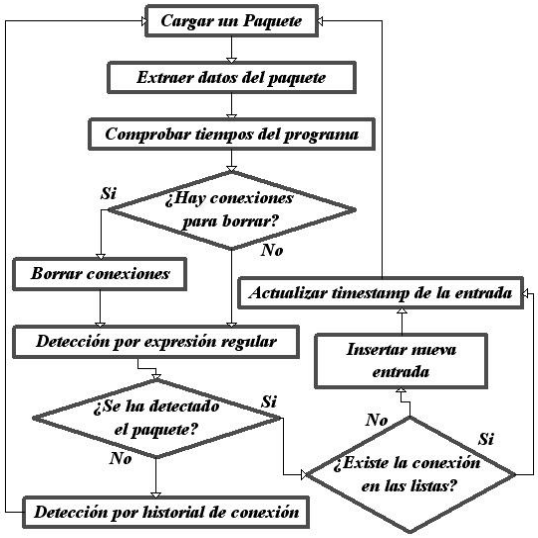

Figura 1 – Esquema de funcionamiento del bucle principal

C. Historial de flujos

Por una parte es importante optimizar las expresiones regulares porque permitirá realizar la identificación más rápidamente. Por otro lado, se necesita llevar cuenta de todos los flujos establecidos en cada momento para poder identificar todos los paquetes del flujo aunque sólo parte de ellos hayan verificado la expresión regular correspondiente. Es lo que denominaremos historial de flujos.

Al tener que revisar todos los flujos almacenados cada vez que queremos identificar un paquete no reconocido por expresión regular, la estructura que almacena los flujos debe estar pensada para hacer búsquedas eficientes.

El tiempo de permanencia de los flujos (modificable por el usuario) permite eliminar todos aquellos flujos que no han cursado tráfico en cierto tiempo, dando con ello el flujo por cerrado y evitando que la estructura de datos crezca indefinidamente. El modo de funcionamiento es muy simple, basta tan sólo con comparar el tiempo que lleva el flujo en la estructura con el tiempo de permanencia definido por el usuario. Si se ven nuevos paquetes de un flujo se actualizará el timestamp en la estructura y con ello se extenderá el tiempo de vida del flujo.

La estructura creada es una variante de tabla de hash. La estructura dispone de un número variable (definido por el usuario) de posiciones de hash sobre un vector indexado, y en cada posición hay una lista sobre la cual se va distribuyendo la información de los flujos que colisionan en ese mismo hash. Para saber sobre qué lista se debe almacenar cierto flujo aplicamos un hash sobre los puertos y la parte más variable de las direcciones IPs como mostramos a continuación:

*Hash=(Puerto origen + Puerto destino + Byte 3 IP origen * 256 + Byte 4 IP origen + Byte 3 IP destino * 256 + Byte 4 IP destino) Mod (Numero de listas)*

Cuanto mejor distribuidos estén los flujos a lo largo de todas las listas, mejor será la eficiencia del programa porque las listas serán más cortas y la búsqueda en cada lista es secuencial, con lo que el tiempo de búsqueda crece con el tamaño de las listas. En la Figura 2 podemos ver la distribución de 2.000 conexiones distribuidas a lo largo de 50 listas con un hash que sólo tiene en cuenta los puertos, mientras que en la Figura 3 podemos ver la distribución de las mismas conexiones empleando el hash que hemos propuesto. Estos resultados se han obtenido en alrededor de 15 minutos de una captura en vivo de varios protocolos cuando se estaban ejecutando aplicaciones de los protocolos eMule y BitTorrent. En la primera hay dos listas considerablemente más largas que las demás lo que supone mayor tiempo de búsqueda medio, mientras que en la segunda la distribución de flujos por lista es uniforme y de esta forma reducimos al mínimo el tiempo empleado por el proceso de búsqueda.

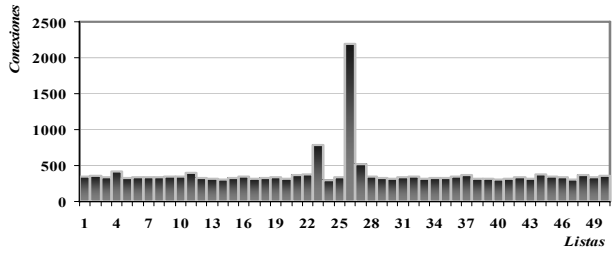

Figura 2 – Distribución sobre un hash mal definido

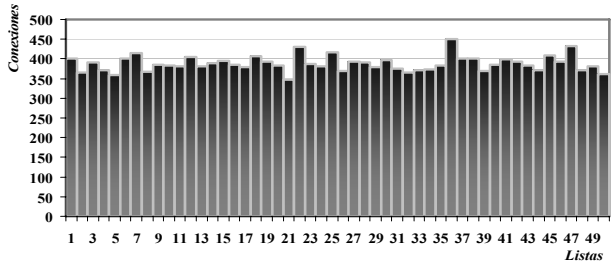

Figura 3 – Distribución sobre un hash bien definido

La definición de un buen hash ayuda a distribuir correctamente los flujos detectados entre todas las listas disponibles. Si se consigue mantener las listas disponibles con un número reducido de entradas conseguiremos que el tiempo que dedica el programa a procesar las listas (buscar flujos, borrar flujos caducados o actualizar timestamps) se reduzca. De esta forma disminuiremos los tiempos de procesado de cada paquete. Al elevar el número de listas obtendríamos mejores resultados hasta el caso óptimo en el que las listas fueran de un elemento. Sin embargo, esto supone un consumo de recursos de memoria muy importante que las listas dinámicas solventan.

Para realizar el estudio del tiempo de ejecución del programa por paquetes que mostramos en las siguientes figuras, se han colocado unos marcadores de tiempo con precisión de µsg en el interior del programa, en cada una de las 8 fases del esquema en que se ha dividido el mismo. Los resultados se obtienen de la ejecución del programa sobre las trazas almacenadas de tráfico eDonkey, que se detallarán en la sección siguiente.

La Figura 4 muestra los resultados obtenidos de dicha ejecución para el caso de una sola lista (sin tabla de hash) sobre la captura de tráfico de 15 minutos anterior. Cada barra vertical es un paquete diferente y tan sólo se ha graficado uno de cada mil para una correcta visualización.

Como se puede ver en la gráfica, los primeros paquetes (los de la izquierda) consumen muy poco tiempo de procesado total. En negro aparece el tiempo empleado en aplicar las expresiones regulares y en gris el tiempo de procesado relacionado con la búsqueda en el historial para aquellos paquetes que lo necesitaran. Sin embargo, conforme se avanza en la ejecución y la lista se va llenando de entradas correspondientes a nuevos flujos, las fases que dependen del tamaño de las listas se van haciendo más notorias. Hacia el final de la ejecución, el tiempo dedicado a procesar las listas llega a ser de más de dos milisegundos por paquete.

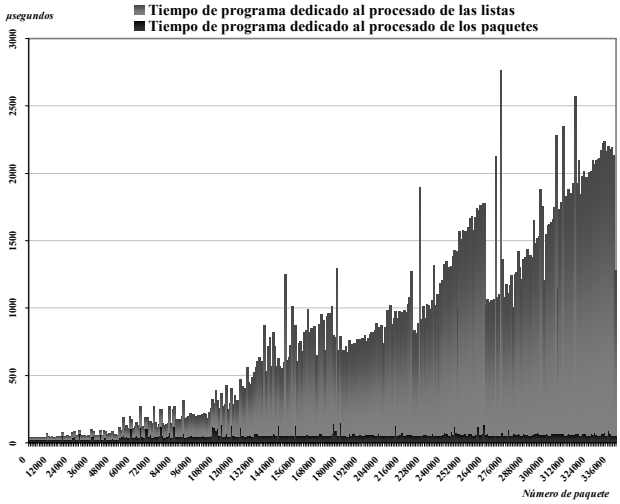

Figura 4 – Tiempo de programa por paquete para una lista

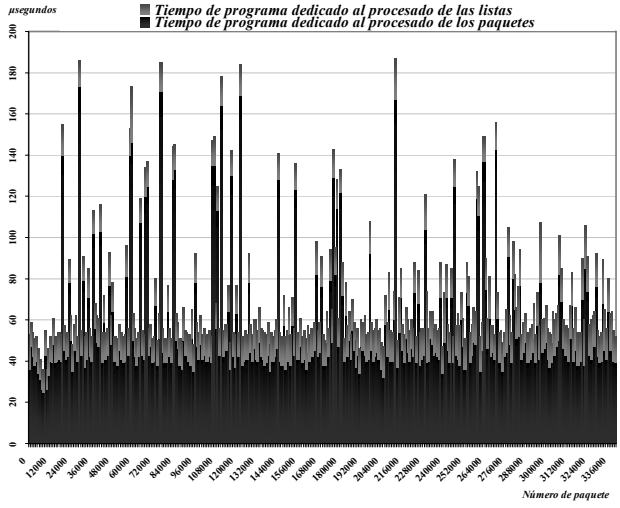

Figura 5 – Tiempo de programa por paquete para 10.000 listas

La Figura 5 muestra los resultados obtenidos para el caso de tener el hash con 10.000 listas sobre la captura de tráfico de 15 minutos anterior. Cada barra vertical al igual que en el caso anterior es un paquete diferente, y corresponde al primero de cada mil paquetes.

En este caso los resultados son mucho mejores. Como podemos ver en la Figura 5, los resultados de cada paquete se mantienen independientes unos de otros y los primeros paquetes tardan prácticamente lo mismo en ser procesados que los últimos. También se observa que todos los paquetes están por debajo de los 200 µsg, y que los que sobrepasan los 60 µsg se comprueba que es porque son paquetes muy grandes (1.500 bytes), lo que supone mayor tiempo de búsqueda en la expresión regular. Además, ahora la fase que más tiempo le lleva es procesar los paquetes con expresión regular frente al procesado relacionado con las listas.

D. Comparativa de versiones Windows-Linux

El programa XePI ha sido desarrollado para ambas plataformas Windows y Linux. Ambas versiones han sido desarrolladas en lenguaje C++ y comparten gran parte de código común. La diferencia fundamental entre ambas plataformas es la utilización de diferentes librerías para la captura de paquetes (Winpcap [12] /Libpcap [13]) y para la evaluación de expresiones regulares (Greta [14] /Boost [15]) como aparece en la Tabla 1.

Si bien en cuanto a las librerías de captura apenas hay diferencias, en las librerías de expresiones regulares existen diferencias y resultan de importancia debido al alto coste computacional que supone la aplicación de expresiones regulares. En [11] podemos encontrar una comparativa entre ambas librerías Greta y Boost que se resume en los resultados de la Figura 6 aplicando diferentes expresiones regulares sobre textos extensos. En la mayor parte de los casos Boost reduce los tiempos de manera significativa. En efecto, la versión Linux del programa consigue tiempos de procesado sensiblemente inferiores a los de la versión Windows, únicamente debido a las peculiaridades de cada librería de evaluación de expresiones regulares.

Windows 2000 Profesional 5.00.2195 SP4	Linux Fedora Core 4 2.6.11
WinPcap 4.0.1	LibPcap 0.9.8
GRETA 2.6.4	BOOST Regex 1.34.1

Tabla 1 – Librerías empleadas en cada versión del programa

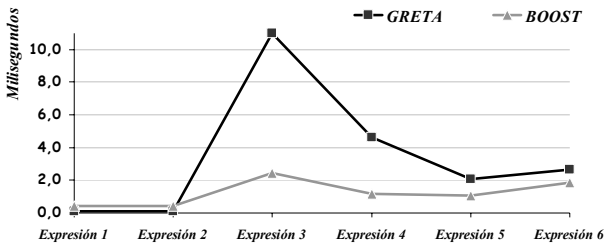

Figura 6 – Comparativa Boost-Greta para un texto extenso

E. Definición de firmas extensible

La carga de firmas asociadas a aplicaciones se realiza de manera dinámica en el programa. Para ello se definen lo que hemos venido a llamar archivos de protocolo por cada aplicación que se quiera soportar. Se trata de no tener que modificar el código fuente del programa para soportar nuevas aplicaciones sino que sea tan sencillo como crear un archivo de texto con las especificaciones de la aplicación.

Los archivos de protocolo contienen los campos mostrados en la Tabla 2, y será necesario crear un archivo con ese formato por cada aplicación que se desee soportar. En la Figura 7 se muestra un ejemplo del archivo de protocolo para BitTorrent.

Campo	Explicación
Type:	Este campo define cómo aplicar la expresión regular: 0 sobre datos de transporte (la habitual), 2 sobre datos por encima de enlace
Name:	Nombre de la aplicación o el protocolo.
Number:	Identificador de protocolo. Un mismo protocolo puede tener varias expresiones regulares que se tomarán como pertenecientes al mismo protocolo si comparten este número.
TCP **UDP** **NOIP**	Sobre qué tipo de paquetes se aplica En el caso de que la expresión sea de tipo 2, este campo tomará el valor NOIP.
Activated **Disactivated**	Para tener o no en cuenta el fichero en el siguiente procesado.
Expression:	Expresión regular a buscar.

Tabla 2 – Estructura de los archivos de protocolo

```
Type:0
Name: bitTorrent TCP
Number:7
TCP
Activated
Expression:^\x13bittorrent protocol
```

Figura 7– Ejemplo del archivo de protocolo BitTorrent para TCP

V. TRAZAS DE TRÁFICO Y PROTOCOLOS SOPORTADOS

Para realizar un estudio en profundidad de la calidad en la detección de los protocolos, en lo que respecta a porcentaje de identificación y no identificación, y dentro de la identificación correcta o incorrecta, además de los tiempos de procesado necesarios, se hace necesario disponer de trazas de tráfico real. Además es imprescindible conocer a priori las aplicaciones en ese tráfico real por lo que no nos sirve una colección de trazas de tráfico cualquiera.

Para este trabajo hemos capturado trazas de tráfico real de varios clientes para cada protocolo ejecutando cada una de las aplicaciones soportadas, de manera que mediante los filtros de tráfico secundario que puedan generar las máquinas, podamos estar seguros de capturar en cada traza

únicamente tráfico de una determinada aplicación. Lo importante ha sido obtener trazas con tráfico específico por protocolo y no tanto la cantidad de usuarios o tráfico que no implicará más que un crecimiento lineal en el tiempo de procesado con el número de paquetes. En la Tabla 3 se presentan las trazas utilizadas en el trabajo. En ella se puede observar que para varios protocolos se han utilizado diferentes clientes con el objetivo de tener en cuenta las peculiaridades de implementación del protocolo por cada cliente.

Como la prioridad en la detección de protocolos era la detección de la mayoría de los clientes P2P más empleados, el estudio se ha centrado en dichos protocolos. Por ello, a pesar de detectar nada más que 6 protocolos P2P se es capaz de controlar hasta el 95% del tráfico de red generado por este tipo de aplicaciones P2P.

Protocolo	Cliente	Número capturas	Tamaño capturas
HTTP	Firefox	1	23.752 kB
	Explorer	1	22.654 kB
	Otros	19	8.744 kB
FTP		10	245.579 kB
DNS		3	1.854 kB
eDonkey	eDonkey	8	37.171 kB
	eMule	6	54.972 kB
gNutella	Bearflix	1	42.893 kB
	Limewire	4	44.195 kB
	Shareaza	2	12.571 kB
	Bearshare	1	2.947 kB
FastTrack	Kazaa	3	20.427 kB
bitTorrent	Azureus	2	23.473 kB
	bitTorrent	3	17.083 kB
	bitComet	1	9.219 kB
	bitTornado	1	7.186 kB
	µTorrent	1	4.934 kB
Ares	Ares	4	52.517 kB
Otros	WinMX	2	6.114 kB
	iMesh	1	2.342 kB

Tabla 3 – Trazas de paquetes empleadas

Como se ha comentado, se puede encontrar diversidad de clientes que soportan el mismo protocolo especialmente en el caso de aplicaciones P2P. En la Tabla 4 se muestran las aplicaciones que se pueden encontrar para los protocolos soportados por nuestro sistema con las expresiones regulares desarrolladas. Como se ha comentado, este conjunto de aplicaciones soportadas es fácilmente ampliable definiendo el archivo de protocolo para las aplicaciones que se quieran añadir.

VI. MEJORAS EN LAS EXPRESIONES REGULARES

Durante el trabajo se ha dedicado especial interés a la optimización de las expresiones regulares responsables de la efectividad de la identificación. En algunos casos se ha partido de las expresiones L7-filter [9] y en otros se han mejorado o directamente creado a partir de la documentación de los protocolos y de trazas de tráfico real observado para las aplicaciones. Se ha conseguido mejorar los porcentajes de identificación sin que por ello aumente la tasa de falsos positivos, es decir, aquellos paquetes que se identifican para una aplicación cuando en realidad pertenecen a otra.

Area	Protocolo	Clientes
RFCs	HTTP	Firefox, Explorer, Opera, Netscape...
	DNS	
	FTP	Clientes FTP
P2P	eDonkey	eDonkey2000, eMule, LMule, Lphant, Shareaza, xMule, iMesh…
	BitTorrent	AllPeers, ABC, Azureus, BitComet, BitTornado, BitTorrent, Lphant, Shareaza, Tribler, µTorrent…
	gNutella	BearShare, Gnucleus, Grokster, KCeasy, LimeWire, Morpheus…
	Napster	Napigator, OpenNap, WinMX…
	Ares	Ares Galaxy, FileCroc, KCeasy…
	Fasttrack	giFT, Grokster, iMesh, Kazaa, KCeasy, Mammoth, mlMac…
Otros	DHCPv6	
	Cisco	
	SMB	
	NBNS	
	Spanning Tree	
	SSDP	
	RPC	

Tabla 4 – Protocolos soportados por XePI

En la Tabla 5 se presentan las expresiones regulares finales asociadas a cada protocolo y que han dado los mejores resultados. En el siguiente apartado se presentarán el análisis comparativo con L7-filter.

VII. EVALUACIÓN DE LA IDENTIFICACIÓN

Tras presentar los protocolos que el sistema es capaz de detectar, queda comprobar el correcto funcionamiento de esa identificación. La única fuente posible de comparación es el proyecto L7-filter, por lo que se ha realizado un estudio del funcionamiento de 8 expresiones regulares respecto a las que se pueden encontrar en el proyecto L7-filter.

En la Figura 8 se presenta una gráfica que muestra el porcentaje de identificación de las expresiones regulares de XePI y de las de L7-filter aplicadas paquete a paquete sin aplicar el historial de flujos. En dicha gráfica se puede ver como las expresiones de XePI detectan una cantidad similar y muchas veces superior a la que detectan las expresiones de L7-filter. En algunos casos esa mejora es significativa con respecto a L7-filter, como en el caso de Ares o gNutella.

Sin embargo, como ya se ha comentado, XePI tiene una segunda oportunidad de detección de los paquetes basada en el historial de flujos. Si un paquete no verifica la expresión regular y ha habido otros paquetes de ese mismo flujo que sí han sido identificados, se supone la misma aplicación para todos los paquetes del flujo, siempre que lleguen dentro del tiempo de permanencia del último paquete del flujo detectado directamente por la expresión regular. Esta mejora no la posee L7-filter que únicamente se basa en aplicar expresiones regulares paquete a paquete. Si incluimos esta

funcionalidad, los paquetes que detecta XePI son más que los que detecta L7, como se puede comprobar en la Figura 9.

Protocolo	Tipo	Expresión regular
HTTP	TCP	^[\x20-\x7e]*http/([01]\.[0-9])[\x09-\x0d-~] *(connection:|content-type:|content-length:|date:)
SMB	TCP	(^.{4,4}|\xff)\xffSMB
	UDP	^.{4,4}\xffSMB
Ares	TCP	^\x03\xff[\x5a\x5d]..\x05
	UDP	^\xe9[\x60\x61\x70\x75\x76\x80-\x83]| ^\x0d\xff{17,17}|ares\x20[0-9]
Napster	TCP	^(.[\x02\x06][!-~]+ [!-~]+ [0-9][0-9]?[0-9] ?[0-9]?[0-9]?"[\x09-\x0d -~]+" ([0-9] |10)|1(send|get)[!-~]+ "[\x09-\x0d -~]+")
RPC	TCP	\xff{3,3}[\x01\xff]\xff{3,3}[\x01\x02\x03 \x04]\xff\x01\x86[\xa0\xa3\xa4\xa5 \xab\xb5]\xff{3,3}[\x01\x02\x03\x04]
	UDP	\xff{3,3}[\x01\xff]\xff{3,3}[\x01\x02\x03 \x04]\xff\x01\x86[\xa0\xa3\xa4\xa5 \xab\xb5]\xff{3,3}[\x01\x02\x03\x04]
FastTrack	TCP	^get (/.download/[~]*|/.supernode[~]|/.status[~] |/.network[~]*|/.files|/.hash=[0-9a-f]*/[~]*) http /1.1|user-agent:kazaa|x-kazaa(-username|-network|-ip|-supernodeip|-xferid|-xferuid|tag)|^give[0-9][0-9] [0-9][0-9][0-9][0-9][0-9][0-9]?[0-9]?[0-9]?
NBNS	UDP	^.{2,2}\x01\x10.\x01.*\x20\x46
DNS	UDP	^.{5,5}[\x01\x02].{6,6}[\x01-\x3f][a-z0-9][(\x01-\x3f)a-z]*[\x02-\x06][a-z]{2,3}.{2,2}[\x01\x1c]\xFF
eDonkey	TCP	^[\xe3\xc5].{2,2}\xff{2,2}[\x01\x02\x05\x0a\x14-\x16\x18-\x1c\x20\x21\x38\x40-\x43\x46-\x52\x54-\x59\x60\x81\x82\x85-\x87\x8b\x8e\x92\x93\xa4]
	UDP	^\xe3[\x0c-\x16\x21\x24\x94\x96-\x9c\x9e\xa0-\xa4]
gNutella	TCP	^(gnutella[\x20\x2f]|get /uri-res)
	UDP	^(((.|\x0a){16,16}[\x01\x31\x41\x40\xff \x80\x81]|\x01\xff)|(GND))
bitTorrent	TCP	^\x13bittorrent protocol
	UDP	d1:[ar]d2:id20
SSDP	UDP	NOTIFY\x20.\x20http/([01]\.[0-9])[\x09-\x0d -~] *HOST[\x09-\x0d -~]*CACHE-CONTROL[\x09 -\x0d -~]*LOCATION[\x09-\x0d -~]*SERVER
FTP	TCP	^[\x09-\x0d -~]*ftp
ARP		^\xff\x01\x08\xff..\xff[\x01\x02]
Spanning Tree		^\x42\x42.\xff{3,3}
CISCO LOOP		^\xff\xff[\xff\x01-\x0a]\xff{43,43}
CISCO CDP		^\xaa\xaa.*cisco
DHCPv6		^.{40,40}\x02\x23\x02\x23

Tabla 5 – Expresiones regulares desarrolladas

En el caso de DNS, la mejora no es tal debido a que en las consultas de DNS se intercambian 2 paquetes habitualmente (la versión UDP) y por tanto el historial de flujos no aporta una mejoría significativa. En el caso de FTP se están considerando únicamente las conexiones de control y no las de datos. El sistema se podría extender para reconocer las conexiones de datos asociadas a una de control, pero en este caso sería suficiente con identificar las conexiones según las direcciones IP y puertos, porque los puertos de la conexión de datos se notifican expresamente en la conexión de control. No sería necesario un análisis de firmas como tal.

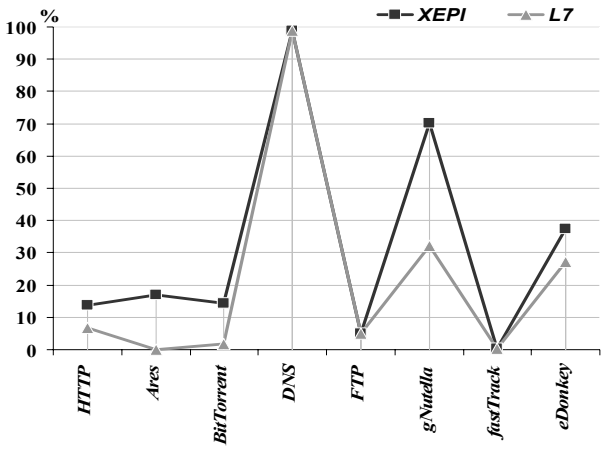

Figura 8 – Comparativa XePI-L7 de los paquetes detectados por las expresiones regulares de los principales protocolos

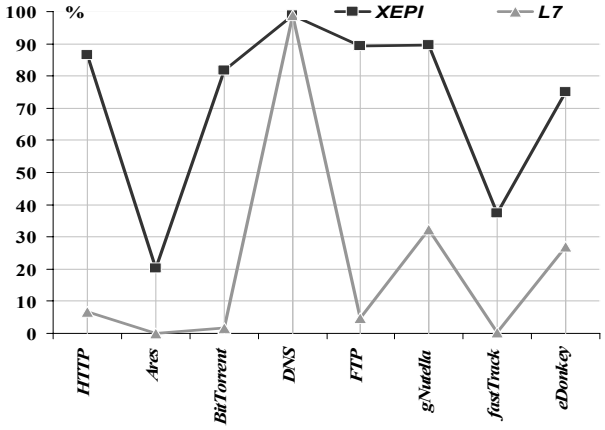

Figura 9 – Comparativa XePI-L7 de los paquetes detectados usando historial de flujo para XePI

En cuanto al tiempo de ejecución, se puede comprobar en la Figura 10 que tanto XePI como L7-filter se mueven en tiempos similares aun cuando la identificación de XePI es mucho mayor. En parte es debido a que las expresiones regulares de XePI son en general más rápidas que las de L7-filter, con lo que el coste extra de procesado del historial de flujo es perfectamente asumible en XePI. De hecho, como se ha comentado anteriormente, el factor determinante de la velocidad del sistema viene fijado casi exclusivamente por la evaluación de las expresiones regulares.

En cuanto a los falsos positivos, XePI y L7-filter ofrecen resultados similares, con falsos positivos siempre inferiores a un 0,2%, salvo en el caso del protocolo HTTP como se puede ver en la Figura 11. Para HTTP XePI tiene una tasa superior al 3,5% de falsos positivos pero que no se considera importante comparado con el porcentaje de identificación que consigue frente a L7-filter. La expresión regular de L7-filter para HTTP es más estricta, con lo que la identificación es menor y con ello también los falsos positivos. En todo caso, la expresión regular de HTTP para

XePI debería ser objeto de estudios posteriores más profundos con el fin de reducir la tasa de falsos positivos.

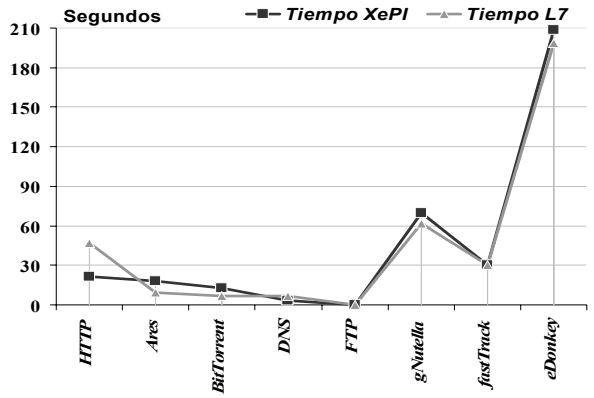

Figura 10 – Comparativa XePI-L7 del tiempo de procesado

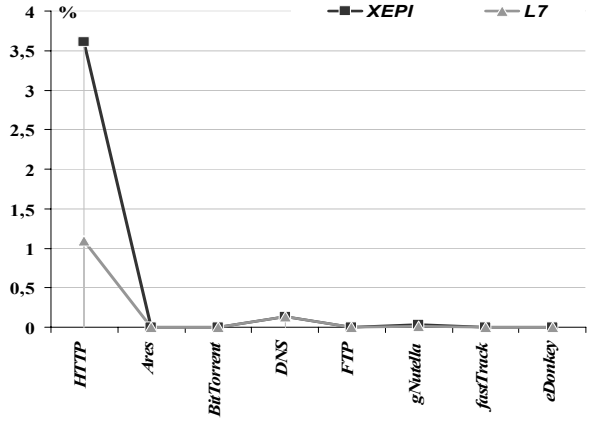

Figura 11 – Comparativa XePI-L7 de los falsos positivos

VIII. CONCLUSIONES

La identificación de tráfico basada en firmas es fuertemente dependiente de la calidad de las expresiones regulares utilizadas en la identificación. Calidad por una parte en cuanto a tasa de identificación que obtienen y también en cuanto a la tasa de falsos positivos que resultan. Por otro lado, la calidad también estará relacionada con el tiempo necesario para aplicar determinada expresión regular: no será útil una expresión muy lenta aunque identifique perfectamente el tráfico porque no sería operativa.

En este compromiso es sobre el que nace XePI como sistema diseñado para la identificación basada en firmas. Se han mejorado las expresiones regulares del proyecto L7-filter que se trata de las pocas fuentes disponibles de expresiones de suficiente calidad. La mejora ha venido por el aumento en la tasa identificación con valores de falsos positivos del mismo orden excepto para el caso de HTTP. También se ha mejorado en cuanto a los tiempos necesarios para aplicar las expresiones regulares simplificando la complejidad de buena parte de ellas.

El mecanismo de historial de flujos planteado permite mejorar aún más los resultados frente a propuestas que se limitan a aplicar expresiones regulares por paquete como L7-filter. El historial de flujos permite llevar estado de todos los flujos establecidos en una red compuestos por los paquetes que comparten la tupla {ip_origen, puerto_origen, ip_destino, puerto_destino} (conexiones TCP o flujos UDP). Esto permite identificar todos los paquetes de un flujo con tal de que sólo uno de los paquetes haya verificado una de las expresiones regulares asociadas a una aplicación. Para aplicaciones con flujos de duración importante se consiguen mejorías significativas en el porcentaje de paquetes identificados sin que por ello empeore la tasa de falsos positivos.

El desarrollo de expresiones regulares es un proceso de evolución continua, ya que constantemente se desarrollan nuevos protocolos o se mejoran los existentes. Además, la identificación de cada paquete es un proceso costoso y que cada vez va a resultar más difícil con la constante mejora en la velocidad de las redes. Es por ello que existen otras técnicas basados en heurísticos muy prometedoras pero que necesitan de estos métodos basados en firmas para generar los datos de referencia o entrenamiento del sistema con mayor precisión.

REFERENCIAS

[1] C. Fraleigh, et al. Packet-level traffic measurements from the Sprint IP backbone. IEEE Network, November/December 2003.

[2] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. SIGCOMM Computer Communications. Review Vol.35, No.4, pp. 229-240, Oct. 2005.

[3] L. Salgarelli, F. Gringoli and T. Karagiannis. Comparing traffic classifiers. SIGCOMM Computer. Communications Review Vol.37, No.3, pp.65-68, Jul. 2007.

[4] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service mapping for qos: a statistical signature-based approach to IP traffic classification. Proceedings of the 4th ACM SIGCOMM IMC'04 conference. pp.135–148, New York, NY, USA, 2004.

[5] T. Karagiannis, A. Broido, N. Brownlee,Kc Claffy, and M. Faloutsos. Is p2p dying or just hiding? IEEE Globecom 2004, Dallas, TX, USA, November 2004.

[6] A. W. Moore and K. Papagiannaki. Toward the Accurate Identification of Network Applications. Proceedings of Sixth Passive and Active Measurement Workshop (PAM 2005), Boston, MA, March/April 2005.

[7] S. Sen, O. Spatscheck, and D. Wang. Accurate, Scalable In-Network Identification of P2P Traffic using Application Signatures. Proceedings of the 13th International World Wide Web Conference, pp. 512-521, NY, USA, May 2004.

[8] Alok Madhukar and Carey Williamson. A Longitudinal Study of P2P Traffic Classification. 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2006, pp.179-188, 11-14 Sept. 2006

[9] J. Levandoski, E. Sommer and M. Strait. Application Layer Packet Classifier for Linux. http://l7-filter.sourceforge.net

[10] L. Karttunen, J-P. Chanod, G. Grefenstette, and A. Schiller. Regular expressions for language engineering. Natural Language Engineering, Vol.2, No.4, pp.305-238, 1996.

[11] John Maddock. Regular expression performance comparison. 2004. http://research.microsoft.com/projects/greta/regex_perf.html

[12] WinPcap: The Windows Packet Capture Library. 2008. http://www.winpcap.org

[13] Tcpdump/libpcap. 2007. http://www.tcpdump.org

[14] The Greta Regular Expression Template Archive. Microsoft 2007. http://research.microsoft.com/projects/greta

[15] John Maddock. Boost-Xpressive.2008. http://www.boost.org