Técnicas eficientes de filtrado y análisis de tráfico para la monitorización continua de redes de comunicaciones

[1]J.J. Ruiz, [2]E. Magaña, [3]J.Aracil, [4]J.Villadangos
Departamento de Automática y Computación
Universidad Pública de Navarra
Campus de Arrosadía s/n, 31006 Pamplona
Telf: 948168904, Fax: 948169281
Email: [1]jr18315@zurron.upna.es, {[2]eduardo.magana, [3]javier.aracil, [4]jesusv,}@upna.es

Abstract

*This paper presents an efficient traffic filtering and analysis architecture for network monitoring. Opposed to the usual network monitoring architectures that provide simultaneous filters as requested by managers (*packet filters*), we propose a different approach that aims at minimizing CPU load by avoiding unnecessary filter duplicates. Such architecture makes it possible to optimize several parallel filters execution and thus is suitable for continuous network monitoring in which it is necessary to keep track of hundreds of filters. This architecture has been implemented in a network-monitoring tool called PROMIS whose main features are detailed in this paper.*

1. Introducción

Las redes corporativas se han convertido en uno de los elementos estratégicos de cualquier compañía de medio y gran tamaño. Las tecnologías de red evolucionan rápidamente, haciendo que las pequeñas redes locales aisladas se conviertan en grandes sistemas de información multiservicio. Incluso se está produciendo la convergencia entre las redes telefónicas y de datos, tendiendo a un acceso común para voz (VoIP), datos, imagen y vídeo. Así, puesto que estas redes de datos constituyen una parte fundamental de cualquier actividad empresarial es preciso asegurar su disponibilidad mediante un sistema de monitorización de red [3][5].

Estos sistemas de monitorización de red disponen de elementos activos que recogen el tráfico de la red en modo promiscuo (sondas) y su funcionalidad se agrupa en tres grandes grupos: realización de capturas de tráfico, programación de alarmas y recogida de estadísticas como, por ejemplo, número de paquetes IP que produce una máquina. Para realizar estas funciones es necesario disponer de filtros de paquetes en las sondas que monitorizan la red. Esta necesidad viene dada por el altísimo grado de concurrencia de este tipo de sistemas, en el entorno de alta velocidad y gran volumen de tráfico generado que presentan las actuales redes de comunicaciones. Así, un sistema de monitorización típico puede estar recogiendo estadísticas específicas como los paquetes/sg entre dos máquinas a nivel IP, mientras simultáneamente se verifican situaciones de alarma y se realizan las capturas programadas. En este escenario es imprescindible disponer de un filtro que sólo deje pasar los paquetes IP de la máquina que se pretende monitorizar, para realizar el cálculo de estadísticas a partir de estos paquetes solamente y no a partir del tráfico total en bruto, que cargaría innecesariamente al proceso de cálculo de estadísticas con un excesivo número de eventos.

Observamos entonces que la técnica de filtrado de paquetes determina en gran medida las prestaciones de un sistema de monitorización. En el caso de redes de medio/gran tamaño es práctica común la realización de un gran número de capturas, alarmas o recogida de estadísticas. Cada una de estas operaciones depende de un filtro para seleccionar los paquetes que le conciernen. Así, las técnicas de filtrado y análisis de tráfico constituyen un aspecto esencial en el diseño de sistemas de monitorización de red.

Precisamente, este artículo propone una arquitectura de filtrado y análisis de tráfico destinada a su utilización en un sistema de monitorización de redes de comunicaciones. La aplicación de la técnica propuesta resulta en la optimización de series de filtros simultáneos. La idea intuitiva de esta nueva técnica es la utilización de los niveles de los protocolos de red para la organización del filtrado, de forma que se aprovechan filtrados comunes en niveles inferiores aunque difieran los filtrados a niveles más altos. Como resultado se logran dos objetivos fundamentales en un sistema de monitorización de red. Por un lado, la flexibilidad necesaria en cuanto a la programación de filtros, capturas y recogida de estadísticas específicas. Por otro lado, la eficiencia necesaria para que el sistema permita la monitorización continua, sin interrupción, de redes de alta velocidad.

El resto del artículo se organiza de la siguiente forma: en la sección 2 se expone el estado del arte en sistemas de filtrado y la motivación para presentar uno nuevo. En la sección 3 se comentan los problemas típicos a los que se enfrenta una implementación de filtrado de protocolos y en la sección 4 se expone la arquitectura del sistema de monitorización PROMIS que hace uso de este núcleo

de filtrado. Las secciones 5 y 6 se centran en la especificación de la arquitectura de filtrado y rendimiento del sistema, para terminar en la sección 7 con las conclusiones.

2. Motivación

El filtrado de protocolos en un sistema de monitorización de redes de comunicaciones se puede realizar desde el kernel del sistema operativo o desde un proceso de usuario. En el primer caso el código es más difícil de desarrollar y mantener, mientras que en el segundo caso se produce una sobrecarga de procesado y una bajada del rendimiento, debido al incremento de cambios de contexto y llamadas al sistema que supone.

Una solución de equilibrio que se utiliza normalmente en sistemas UNIX es el *packet filter*, que permite una programación de filtros desde el nivel de usuario con una parte del kernel que filtra paquetes según los criterios especificados al nivel de usuario, resultando el proceso eficiente. Esta parte del kernel es la encargada de demultiplexar paquetes. El *packet filter* aísla el kernel de los detalles de implementación de los protocolos. La interfaz que provee el *packet filter* otorga facilidades para transmisión y recepción de paquetes, e información y control de estado. Algunos ejemplos de *packet filter* son el BPF de sistemas BSD[1] y el CSPF [2], que fue la primera tentativa. Otros sistemas que permiten el acceso de los procesos de usuario a los paquetes en el nivel de enlace son el DLPI de Solaris y el Snoop de sistemas Irix.

A continuación se presentan las características de estos sistemas de *packet filter*:

- CSPF(CMU/ /Standford Packet Filter): utiliza un árbol de expresiones booleanas, en el que los nodos representan operaciones booleanas y las hojas campos del paquete a comparar. Sus limitaciones son que no puede tratar cabeceras de tamaño variable y trabaja a 16bits.
- BPF (BSD Packet Filter): realiza el filtrado a nivel de kernel según unos filtros definidos a nivel de usuario y copia solo la parte del paquete solicitada. El filtrado está basado en registros y proporciona un buffer en el que se van almacenando los paquetes recibidos junto con su timestamp y longitud, disminuyendo el número de llamadas al sistema que ha de realizar la aplicación. El filtrado se basa en un grafo dirigido en el que cada nodo representa un campo del paquete, según el resultado del test se atraviesa una rama o la otra y sólo hay dos hojas que representan los valores true/false para todo el filtro.

Ambos filtros son equivalentes computacionalmente, pero el BPF está optimizado para la arquitectura de máquinas basadas en registros.

Estos *packet filter,* aunque proporcionan una herramienta útil para el análisis parcial del tráfico que circula por la red, no son de aplicación en el caso de monitorización de redes de medio/gran tamaño. En el caso de análisis de redes que nos ocupa en este artículo se pretende monitorizar la red de forma global, lo que supone miles de filtros simultáneos y procesado de todos los paquetes de la red. En estos entornos, con enlaces de alta velocidad, un *packet filter* produce una carga excesiva del kernel por los posibles filtros duplicados, que conlleva la pérdida de paquetes y, por tanto, fallos graves de monitorización.

Para ilustrar este fenómeno de duplicación definimos *captura* como un conjunto de filtros de los diferentes campos de cada trama (por ejemplo, paquetes con una dirección IP destino determinada por el puerto TCP 80 –WWW-), como se muestra en la Fig. 1. Se definen tres capturas que comparten algunos filtrados intermedios: por ejemplo, la segunda captura supone comprobar en los paquetes si poseen cierta dirección MAC_1 origen, cualquier dirección MAC destino, cierta dirección IP_3 origen y que además sea un paquete TCP. Se observa como la tercera captura comparte el filtro de dirección IP_3 origen con la segunda captura, entre otros filtros comunes. Así, el diseño de estas capturas se puede enfocar de dos maneras distintas:

- Comprobar que cada paquete satisfaga cada uno de los filtros de cada captura, de forma independiente entre cada captura, como indica la Fig. 2. Esta es la técnica de filtrado de los *packet filter* CSPF y BPF, que resultaría en la aplicación de 12 filtros.
- O bien, hacer que se compruebe que cada paquete satisfaga cada uno de los filtros, pero de forma que si una serie de capturas tienen en común un mismo filtro, esta comprobación sea común para todas esas capturas. De esta forma, la comprobación de que un paquete cumple o no un filtro determinado se realiza una sola vez para todo ese conjunto de capturas, como se indica en la Fig. 3. Resultan 6 filtros, la mitad que en los anteriores.

Captura1	MAC_1 origen	MAC_2 destino	IP_3 origen	IP_4 destino	UDP
Captura2	MAC_1 origen	IP_3 origen	TCP		
Captura3	MAC_2 destino	IP_3 origen	IP_4 destino	UDP	

Fig. 1: Capturas propuestas

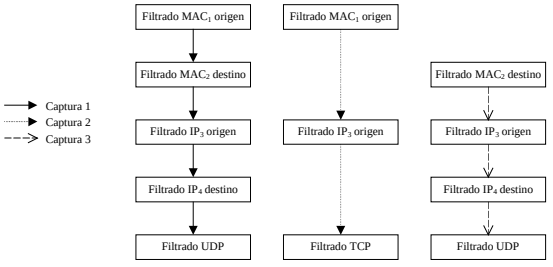

Fig. 2: Capturas por filtrado mediante CSPF y BPF

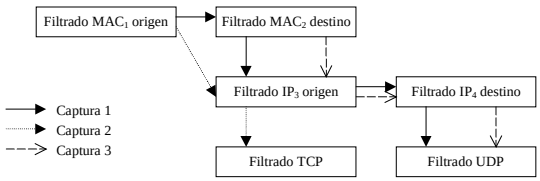

Fig. 3: Capturas por filtrado en árbol

Comparando ambas estrategias se observa que la segunda opción minimiza el número de comprobaciones para cada paquete y un mismo número de filtrados. Por ejemplo, si se quiere realizar un cierto número X de capturas, de forma que cada una de ellas tengan los mismos N filtros, es decir, repetimos X veces la misma captura. En la primera estrategia se tiene una cifra de NxX comprobaciones en los filtros para cada paquete, mientras que en la segunda estrategia se tienen únicamente N comprobaciones.

La técnica propuesta se basa en la segunda estrategia y se ha seleccionado una disposición de los diferentes filtros en forma de una estructura en árbol, de modo que se identifica cada filtro con un nodo del árbol. Cada nodo representa una comprobación de cierto campo de determinado protocolo, observándose la estructura lógica de niveles de protocolos en el diseño del árbol. Si distintos filtrados comparten etapas de filtrado comunes, compartirán los nodos del árbol evitando la realización de filtrados duplicados como en los otros sistemas de *packet filter*.

3. Problemas de Sistemas de Filtrado y Análisis de Tráfico de Red

Todos los sistemas de captura de paquetes, ya sean basados en *packet filter* o al nivel de usuario, muestran una serie de problemas [4] o limitaciones entre los que destacan los que se citan a continuación:

- La precisión del Timestamp (marca temporal con la que se ha recibido el paquete): depende del sistema sobre el que funcione, del orden de centésimas de segundos para un PC, o de microsegundos para una SUN Sparc. En todo caso, se garantiza que cada paquete recibido posee un timestamp mayor o igual que el paquete anterior.
- Pérdidas de paquetes: se puede producir a dos niveles. En el ámbito de la tarjeta de red, cuando no se hacen lecturas sobre la tarjeta a suficiente velocidad y esta ha de sobreescribir los paquetes recibidos en su buffer, normalmente de tamaño pequeño, del orden de Kbytes. El otro ámbito se produce en el buffer intermedio, cuando el proceso de filtrado y análisis no es lo suficientemente rápido.
- Generación de paquetes fantasma, por fallos de la implementación del soporte de red. Por ejemplo, en tasas alta de carga se ha detectado como en Linux se capturan paquetes de tamaño menor al mínimo permitido en una Ethernet (64 bytes). En realidad, estos paquetes no existen sobre la red y son creados por el driver de la tarjeta de red cuando se intenta mandar paquetes a la red desde la misma máquina, estando la tarjeta en modo promiscuo.

Estos aspectos se han tenido en cuenta en la implementación de la técnica de filtrado propuesta, evitando o minimizando sus efectos sobre los resultados finales. Dicha técnica de filtrado es el núcleo del sistema de monitorización distribuido PROMIS [7], diseñado por los autores de este trabajo, que se presenta brevemente en el siguiente apartado.

4. Sistema PROMIS

El sistema de monitorización PROMIS está compuesto de sondas y consolas. Las sondas están implementadas en PCs con Linux como sistema operativo, y son las encargadas del filtrado y análisis de tráfico de la red, además de servir la información procesada a la consola. Se coloca una sonda por cada segmento de red que se desee monitorizar. La consola provee la interfaz de usuario, a través del cual el gestor de la red puede acceder a la información recogida por las sondas. Esta información puede ir desde gráficos de monitorización en tiempo real de un determinado parámetro de tráfico hasta la recepción de capturas, pasando por la recepción de alarmas ante problemas detectados en las redes. Los parámetros que se pueden obtener pueden ser bytes/sg, paquetes/sg o utilización media, globales o filtrados por máquina, protocolo, servicio y/o aplicación. Además se lleva cuenta de las "top N" máquinas que más tráfico generan y la matriz de tráfico entre ellas. Se posee un mecanismo flexible de definición de alarmas en función de umbrales de determinado parámetro, y también se pueden definir capturas de parámetros para analizarlas con posterioridad. El sistema se completa con una serie de informes accesibles desde la Web con los parámetros resumidos de evolución de la red a lo largo de la última hora, día, semana y mes.

5. Arquitectura de filtrado

5.1. Arquitectura de la sonda

El procesado de cada paquete o trama que se recibe por la red se realiza mediante el software instalado en la sonda conectada a esa red. El software está desarrollado en lenguaje C, sobre sistema operativo Linux, y comprende cinco procesos concurrentes y dos memorias compartidas:

- Un proceso "*lector*", que se encarga de recoger los paquetes de la red, y el proceso "*master*", que se encarga del análisis de dichos paquetes. Además existen tres procesos "*servidores*", que son los encargados de comunicar la sonda con la consola y de suministrarle los datos y parámetros que solicite.
- Las dos zonas de memoria reservada se utilizan para el intercambio de datos entre los diferentes procesos. Una de ellas se implementa para el

paso de los paquetes que recoge el proceso "*lector*" hacia el proceso "*master",* actuando como buffer intermedio que permite almacenar paquetes en periodos de pico de tráfico, y de esta forma evitar la pérdida de paquetes comentada en el apartado 3. La otra memoria compartida es la utilizada por el proceso "*master"* para almacenar los parámetros ya analizados para que los procesos servidores puedan disponer de ellos.

El proceso "*master*" es el que realiza el filtrado y análisis del tráfico propiamente dicho, y es en la zona de memoria de este proceso donde se localiza el árbol de filtrado. El árbol de filtrado, aunque funciona internamente en este proceso, accede a la segunda memoria compartida para almacenar los parámetros calculados y hacerlos así accesibles a los procesos servidores.

5.2. Arbol de filtrado

La técnica de filtrado y análisis propuesta se basa en una estructura en árbol. Se organiza por niveles de profundidad, actuando todos los nodos del mismo nivel sobre el mismo campo de la trama recibida, es decir, actúan en la misma posición del paquete (*offset*). Conforme se desciende por el árbol, recorriendo la trama campo a campo, se comprueba el contenido de los campos. La información obtenida será filtrada por los nodos del nivel correspondiente a ese campo. Cuando el filtrado sea positivo para un determinado nodo se puede seguir descendiendo por el árbol a través de ese nodo.

Dada la disposición en árbol, se provee de una absoluta libertad a la hora de configurar el conjunto de filtros de una determinada captura. Además, se tiene la suficiente flexibilidad como para poder actualizar el árbol en tiempo de ejecución, pudiendo añadir nuevos nodos-filtros mientras se siguen procesando tramas recogidas de la red. En cada uno de los nodos-filtros es posible definir una serie de nodos-parámetro, que contendrán la información necesaria para tomar los parámetros requeridos en esa captura. Estos nodos sólo son recorridos y actualizados en caso de que se cumpla el filtro al que están asociados.

Como se puede observar en la Fig. 4, se distingue entre filtrado obligatorio y filtrado opcional en los distintos niveles. El primer caso se refiere a que en un nivel determinado se obliga a un paquete a que cumpla uno, y sólo uno, de los filtros definidos en ese nivel. El segundo hace referencia a que existen niveles en los que se definen estructuras nodo-filtro que no tienen función de filtrado, con lo que se puede pasar directamente al nivel inferior. De esta forma es posible la combinación de conjuntos de filtros en los que no se tiene que especificar un filtro por cada uno de los niveles.

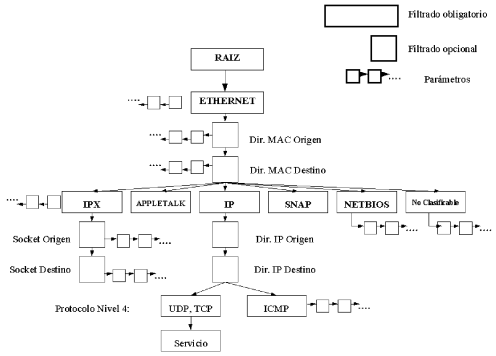

Fig. 4: Ejemplo de árbol de filtrado sobre red Ethernet

La necesidad de fijar niveles de filtrado obligatorio se debe a que es necesario obtener forzosamente información del campo de la trama que se está chequeando en ese momento para poder seguir descendiendo por los nodos del árbol. Por ejemplo, es necesario comprobar el campo situado a partir del byte 13 de una trama Ethernet para saber qué protocolo encapsula la trama. El esquema del árbol de la Fig. 4 representa un conjunto de filtrados que se pueden realizar sobre una red Ethernet. Para optimizar la flexibilidad del árbol, no se ha puesto el nodo-filtro Ethernet como raíz de la estructura, sino que se ha añadido por encima de este nodo un nivel más. Esto permite que bajo el nodo raíz se puedan disponer varios nodos, que coincidirían con diferentes protocolos de nivel de enlace, como podrían ser Token Ring, Token Bus, ATM, Fast Ethernet, Gigabit Ethernet... De esta forma, la misma sonda se podría conectar en diferentes redes sin necesidad de modificar la estructura del árbol, y por tanto, sin modificar el software.

Para implementar el árbol se ha definido una estructura de datos única, de forma que todos los nodos del árbol, sin importar el nivel en el que se encuentren, se configuren mediante la misma estructura. Esto puede producir un menor rendimiento en el uso de la memoria del sistema, ya que existen nodos cuyos parámetros no ocupan completamente la zona de memoria asignada de los campos donde se almacenan. Pero el hecho de utilizar este tipo de estructura única simplifica y agiliza, en cuanto a velocidad, la construcción, borrado y recorrido del árbol de filtrado.

Mediante la ayuda de cuadrados, que indican cada nodo, y flechas, que indican el enlace de nodos (en términos de memoria) mediante punteros, en la Fig. 5 se representa la forma del árbol que queda con la utilización de la estructura de nodo anterior. Sin embargo, existen cuatro niveles en el árbol Ethernet donde la disposición de nodos comentada no se satisface. El cambio de estructura se debe a la búsqueda de una disposición alternativa que optimice la velocidad de recorrido del árbol en los niveles de dirección MAC origen, MAC destino, IP origen e IP destino.

La estructura de parámetro es la encargada de llevar los contadores de los bytes o paquetes que cumplen el filtro asociado y que también se representa en la Fig. 5. Como en las estructuras de nodo, se intenta que con un solo tipo de estructura se puedan implementar todos los parámetros definidos en el sistema. La estructura de parámetro puede ir unida a una estructura de nodo o a otra estructura de parámetro, como puede observarse en la Fig. 5.

5.2.1. Posible estructura del árbol de filtrado en una red Ethernet

El descenso de nivel por el árbol coincide con un desplazamiento por los diferentes campos que conforman la trama Ethernet.

- Nivel 0: La raíz del árbol. Este nodo no realiza ningún tipo de filtrado y tan sólo sirve como soporte de todos los niveles que se creen bajo él.
- Nivel 1: Filtrado de tipo de red LAN. Desde este nivel se pueden especificar las diferentes ramas para cada uno de los protocolos Ethernet, Token Ring, ATM... Este nodo es de filtrado obligatorio, ya que hay que saber que protocolo de nivel de enlace se está soportando.

A partir de este nivel la configuración del árbol se ha implementado únicamente para pilas de protocolos sobre Ethernet.

- Nivel 2: Filtrado de dirección MAC origen. Cada nodo de este nivel, exceptuando si existe el nodo de "no filtrado", tendrá en su identificador una dirección MAC. Este nivel es de filtrado opcional, con lo que se puede incluir el nodo de "no filtrado".
- Nivel 3: Filtrado de dirección MAC destino. Igual que el anterior, pero chequeando el campo de la dirección destino de la trama.
- Nivel 4: Filtrado por protocolo de red. Cada uno de los nodos indica un protocolo de red. En su función de filtrado existe un algoritmo que chequea la estructura de la trama, identificando el tipo de protocolo definido a continuación en la trama. Dependiendo del protocolo identificado, el puntero que recorre la trama se desplaza un *offset* determinado para situarse en el primer campo de la cabecera de red.

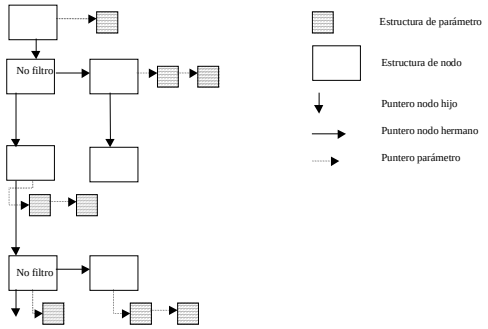

Fig. 5: Esquema de árbol

Dado que es necesario el conocimiento de dicho protocolo, este nivel es de filtrado obligatorio.

- Nivel 5: A partir de este nivel hay que distinguir entre los posibles protocolos de nivel de red que se hayan definido. Por ejemplo, en:
  1. Protocolo IP: Filtrado de dirección IP origen. En cada nodo se pone una dirección IP diferente. En este nivel también se admite el nodo de "no filtrado", dado que el nivel es de filtrado opcional.
  2. Protocolo IPX: Filtrado de punto de acceso al servicio origen. Al igual que el protocolo IP es de filtrado opcional.
- Nivel 6: También hay que distinguir entre protocolos:
  1. Protocolo IP: Filtrado de dirección IP destino, con las mismas propiedades que el nivel anterior.
  2. Protocolo IPX: Filtrado de punto de acceso a servicio destino. Al igual que el protocolo IP es de filtrado opcional.

  Este nivel permite realizar filtrados para propósito de filtrado de túneles.
- Nivel 7: A partir de este nivel se podría implementar, por ejemplo, el protocolo IP. En este nivel se distingue entre protocolos TCP y UDP, y los paquetes ICMP que circulan por la red.
- Nivel 8: Para aquellos paquetes TCP y UDP se filtra el puerto por el que se manda, es decir, se filtra el servicio.

Este es un ejemplo de árbol para una sonda sobre redes Ethernet. Se comprueba que la topología utilizada permite una gran flexibilidad para incrementar, ya sea el número de niveles de profundidad, ya sea el número de nodos dentro de un mismo nivel. Además, dado que cada nodo es independiente de sus hermanos y sólo depende de su nodo padre, es posible la realización de subárboles a partir de cualquier nodo, teniendo estos nuevos árboles la configuración que se desee.

5.2.2 Creación y borrado de nodos en el árbol de filtrado

El proceso master es el encargado, además de analizar las tramas tomadas de la red, de mantener actualizado el árbol de filtrado. Si se desea crear una nueva captura, los procesos servidores proporcionan toda la información de dicha captura al proceso master a través de la Memoria Compartida 2.

El sistema de creación de nodos es el mismo para todos los niveles, al igual que el sistema de creación de parámetros, de tal forma que el algoritmo utilizado no depende de la información que contenga la directriz suministrada por los procesos servidores ni del nivel en el que se vaya a situar el nuevo nodo. El descenso por el árbol se realiza mediante llamadas a sí misma de la función anterior, de modo recursivo. Dicha función es capaz de decidir si es posible la creación de un nodo de "no filtrado" en un determinado nivel, así como decidir cuándo se llega

al punto de comenzar a introducir nodos de parámetro.

Sin embargo, para optimizar la velocidad de recorrido del árbol existen cuatro excepciones a la regla anterior. Dichas excepciones se sitúan en el nivel 2 y 3 del árbol Ethernet (que coinciden con los niveles de filtrado de direcciones MAC), y el nodo 5 y 6 del árbol Ethernet con protocolo de red IP (que coinciden con los niveles de filtrado de direcciones IP). En este nivel la estructura cambia, de forma que se realiza una discriminación previa del identificador de nodo para almacenarlo de una forma más ordenada, con el fin de disminuir el tiempo de búsqueda horizontal del nodo en el nivel. En estos cuatro niveles la estructura queda de la forma que indica la Fig. 6.

En estos niveles se realiza una clasificación previa de los nodos, almacenándolos en uno de los 256 grupos, dependiendo del valor numérico del último byte significativo del identificador (se utiliza el último byte por que es el que más varianza sufre). De esta forma, la comprobación de si el nodo está ya creado se realiza entre un número menor de nodos. Dentro de cada subnivel horizontal, el ordenamiento es igual que en el resto de los niveles, y el árbol que se crea bajo este nodo sigue el criterio general anteriormente comentado, sin las cuatro excepciones.

La necesidad de estas cuatro excepciones está en la generación de la matriz de tráfico y el listado de máquinas presentes en la red de forma dinámica. Para realizarlos es necesario mantener información continua del tráfico entre todo par de máquinas, es decir, se posee un filtro definido por cada combinación de máquina origen y destino. Al recorrer el árbol, si esta gran cantidad de información estuviera almacenada de forma "estándar", el coste de tiempo necesario en la generación dinámica de la matriz y los listados haría que el proceso master dedicase mucho tiempo al análisis de cada paquete. El cambio del algoritmo de almacenamiento de nodos optimiza la velocidad de análisis de cada paquete en estos niveles.

Si se desea borrar una captura, el proceso servidor proporciona al proceso master la misma información que le había suministrado en la creación de dicha captura a través de la Memoria Compartida 2.

5.2.3. Recorrido del árbol de filtrado

El recorrido del árbol se realiza de un modo recursivo "mixto". Esto quiere decir que dentro de la función de recorrido, que se llama a sí misma para descender por el árbol de filtrado, existen cuatro excepciones, donde el recorrido no se realiza de forma recursiva sino de forma directamente implementada en la misma función.

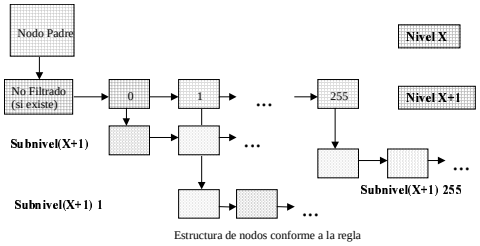

Fig. 6: Estructura de los niveles no estándar

Estas cuatro excepciones coinciden con las cuatro excepciones de la creación del árbol de capturas, y como aquéllas, se han desarrollado para optimizar la velocidad de recorrido del árbol.

El algoritmo de recorrido sobre el árbol Ethernet sigue unas reglas generales:

1. Si la función de filtrado devuelve un valor verdadero, la siguiente acción es seguir el recorrido del árbol por el nodo hijo, si existe. Si no existe, se sigue el recorrido por el puntero a la estructura de parámetro.
2. Como el algoritmo es recursivo, cuando la función a la que ha invocado el nodo en el que se encuentre en ese momento termina su ejecución, se sigue el recorrido del árbol por el puntero a la estructura de parámetro.
3. En cada nivel el paquete cumplirá la función de filtrado de un solo nodo (si se llega a cumplir la función de alguno). El recorrido horizontal se continúa hasta que se encuentre un nodo que cumpla uno de los filtros. Si está definido el nodo de "no filtrado", que todos paquetes cumplen siempre, es posible que se pueda cumplir la función de otro nodo más en ese mismo nivel. Debido a esta posible circunstancia, se fuerza a que tras la vuelta al nodo de "no filtrado", se recorra el nivel en el que se encuentre situado de forma horizontal para comprobar si hay o no otro nodo en el que el paquete cumpla la función de filtrado.

Las tres reglas anteriores se pueden comprender observando la Fig. 7, donde la numeración indica el orden de recorrido en el árbol.

La necesidad de creación y mantenimiento dinámico de la matriz de tráfico y el listado de direcciones MAC e IP de la red ha hecho necesario que se puedan generar nodos–filtros de forma automática, es decir, sin necesidad de que un proceso servidor haga una petición de creación de captura. De esta forma, con cada paquete que se captura de la red, se realiza un proceso de comparación de la información de sus campos de dirección con los identificadores de los nodos de los niveles 2, 3, 5 y 6. Como la información contenida en estos niveles (en forma de estructuras de nodo) puede llegar a tener un gran volumen, en términos de memoria, si se sigue el algoritmo "estándar", que es recursivo, las tareas que debe realizar el sistema operativo cuando realiza los cambios de contexto hace que se pierda una gran cantidad de tiempo y memoria.

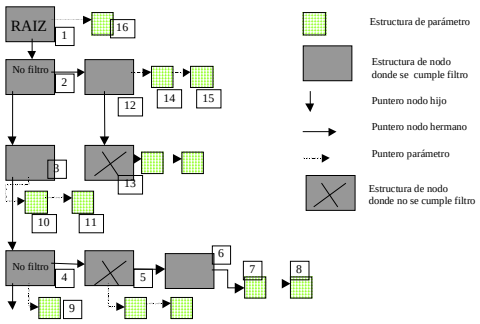

Fig.7: Esquema del recorrido del árbol

Por ello, para el chequeo y actualización de estos niveles se utiliza un algoritmo lineal, cuyo comportamiento y resultados son equivalentes a los que se obtendrían con el algoritmo recursivo, pero sin el malgaste de tiempo y memoria debidos a los cambios de contexto. No se utiliza un algoritmo lineal en todo el recorrido del árbol porque la estructura del filtrado es muy flexible y el algoritmo lineal es muy rígido, y no puede moldearse a todas las posibles combinaciones de filtros que se puedan dar.

6. Rendimiento de la arquitectura

Se han realizado una serie de experiencias para comprobar el rendimiento de la arquitectura descrita en una red de área local Ethernet 10BaseT. La sonda está instalada sobre un PC 133 MHz, 32 MB RAM y 1.5GB de disco duro, con una tarjeta de red 3COM 3c509 ISA. En la red de área local existen 39 máquinas conectadas.

En la Fig. 8 se observa el porcentaje de uso de CPU de la aplicación en función del número de estructuras-nodo que se van creando en el árbol de filtrado para el peor caso, que se ha generado con las condiciones que se explican a continuación. Dado que en la red en la que se toman las medidas no existe una cantidad suficiente de máquinas para crear una gran cantidad de nodos en el árbol, se ha implementado un algoritmo que permite cambiar la dirección MAC de una máquina determinada a una arbitraria. De esta forma se simula la existencia de miles de máquinas en la red.

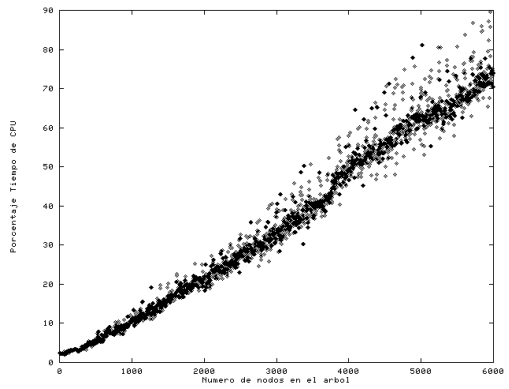

Fig. 8: Porcentaje de uso de CPU en función del número de nodos para el peor caso

Se realiza una secuencia de transmisión a cada uno de estas máquinas “virtuales” de 1000 paquetes a una velocidad media de 3 Mbps. Dado que en el experimento se mantiene la dirección MAC origen constante y lo que se cambia es la dirección MAC destino, en el proceso de actualización de la matriz de tráfico estos nodos de filtrado de MAC no hacen uso del algoritmo de optimización del recorrido del árbol sino que utilizan el estándar.

Por tanto, todos los nodos generados de esta forma aparecen en el mismo subnivel del árbol y su recorrido no está optimizado. La razón de no haber aplicado la optimización es que en el funcionamiento ordinario de una red una máquina se comunica con un número reducido de máquinas, por lo que este efecto no sería apreciable.

De esta forma se observa cómo el porcentaje de uso de CPU se incrementa de forma lineal conforme se eleva el número de nodos en el árbol de filtrado. Esto se debe a que se obliga al nuevo paquete que llega de la red a realizar el recorrido de mayor longitud, con lo que con cada nuevo paquete se incrementa dicha longitud. Este es, por tanto, el peor caso que no puede ocurrir en una red ordinaria. Se observa que para una cantidad de nodos en el árbol cercana a los 6000, se llega a acaparar prácticamente todo el uso de CPU. Cuando se llega a este punto, el tiempo de análisis de cada paquete es superior al tiempo entre llegadas medio de nuevos paquetes, con lo que se comienzan a almacenar estos últimos en la Memoria Compartida I, pudiéndose llegar al punto de que ésta se sature y se comiencen a perder paquetes.

En la Fig. 9 se aprecia el caso que podría corresponder a una red con una carga de tráfico más real. De nuevo, existen miles de máquinas pero ahora el tráfico de una máquina se produce con un número limitado de máquinas destino, en concreto 40, con una carga media de 3 Mbps. Se aprecia como ahora el porcentaje de uso de CPU es asumible, no superando el 10% para 6000 nodos-filtro. Los resultados de la optimización introducida en la estructura del árbol se comprueban con esta experiencia.

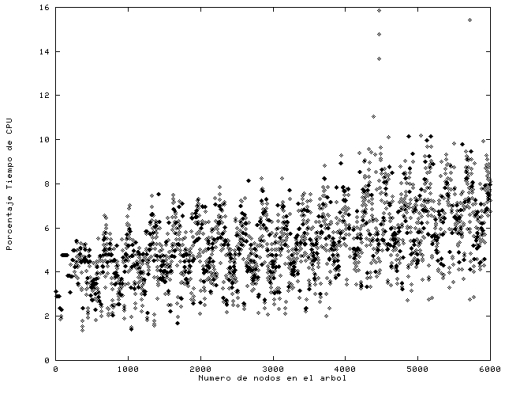

Fig. 9: Porcentaje de uso de CPU en función del número de nodos para una situación de carga de red real

Ante un aumento significativo del número de nodos – filtro en el árbol el incremento de uso de CPU es levemente apreciable (con un aumento de 5000 filtros sólo se aumenta aproximadamente un 3% de uso de CPU).

En la Fig. 10 se representa la evolución del tiempo de ocupación de CPU ante una variación lineal de la carga existente en la red. En este caso, para la generación de la carga de tráfico controlada se han utilizado unos programas cliente-servidor situados en dos máquinas con lo que todos los paquetes de la red proceden de una de estas dos máquinas.

Cuando se ha realizado la medida de esta experiencia, el árbol estaba formado por 1810 nodos. Se observa que el sistema evoluciona de forma aproximadamente lineal ante un aumento de la carga, sin que a altas cargas de la red el coste computacional sea excesivamente alto.

Sin embargo, se comprueba que el coste para una carga de 3 Mbps y aproximadamente 1800 nodos no es la misma entre la experiencia de la Fig. 8 y la Fig. 10: en la Fig. 8 se observa que se utiliza aproximadamente un 20% del tiempo de uso de CPU, mientras que en la Fig. 10 el tiempo de uso es de aproximadamente 6 - 8%. Esta diferencia es producida por la obligación que se tiene en la primera experiencia a realizar el recorrido de longitud máxima en el árbol. En cambio, en la segunda, al tener todos los paquetes con los que se genera el tráfico las mismas direcciones origen y destino, y no estar los nodos correspondientes a éstas en el extremo de los niveles, sino en posiciones intermedias, para analizar estos paquetes no se debe hacer un recorrido por el árbol tan extenso.

En cambio, las experiencias de la Fig. 9 y Fig. 10 representan situaciones similares de tráfico, y por tanto, coinciden los porcentajes de uso de CPU en ambas.

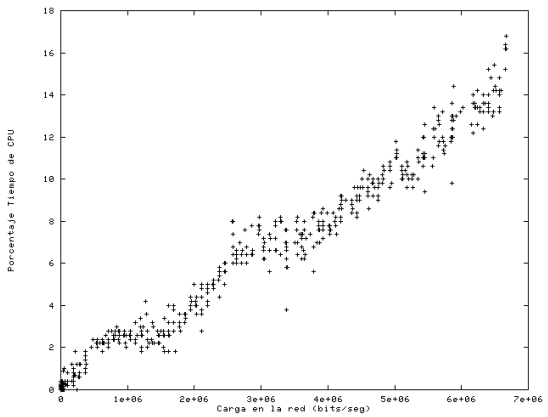

Fig. 10: Porcentaje de uso de CPU en función de la carga de la red

7. Conclusiones

Uno de los elementos fundamentales en el diseño de un sistema de monitorización continua de redes de comunicaciones es la forma en la que se capturan paquetes de la red, se filtran sus protocolos y se analizan. Se ha presentado una alternativa que trata de aprovechar los filtrados a niveles de protocolos comunes con el fin de minimizar el uso de CPU. Este punto de la duplicación de filtros es lo que hace a los *packet filter* poco apropiados para sustentar un sistema de monitorización de tráfico de red en el que se hace necesario llevar miles de filtros simultáneos.

En la arquitectura propuesta se ha optado por una estructura en árbol en la que cada nodo representa una comprobación de cierto campo de determinado protocolo. Si distintos filtrados comparten etapas de filtrado comunes, se comparten los nodos del árbol, evitando la realización de filtrados duplicados como en otros sistemas de *packet filter*. De esta forma se ha conseguido una arquitectura eficiente para el caso de sistemas que realicen miles de filtrados simultáneos, como puede ser el de un sistema de monitorización continua de red.

Para aumentar la flexibilidad de la estructura en árbol se han utilizado algoritmos de creación, borrado y recorrido basados en la recursividad. Sin embargo, para optimizar la velocidad de creación de ciertos filtros muy utilizados como son los de filtrado de máquinas, se ha introducido una estructura rígida, con un recorrido lineal, que agiliza el proceso de mantenimiento de la matriz de tráfico y los listados de direcciones.

Referencias

[1] Steven McCanne, Van Jacobson, *"The BSD Packet Filter: A New Architecture for User-level Packet Capture"*. Winter USENIX Conference Proceedings, San Diego, CA, Jan.1993.

[2] Jeffrey C. Mogul, *"The Packet Filter: An Efficient Mechanism for User-level Network Code*". Proceedings of 11th Symposium on Operating Systems Principles, Austin, TX, Nov.1987, ACM, pp-39-51

[3] David R. Boggs, Jeffrey C. Mogul, Christopher A. Kent, *"Measured Capacity of an Ethernet: Myths and Reality"*. SIGCOMM'88 proceedings, Standford, CA, Aug.1998, ACM

[4] Jeffrey C. Mogul, *"Efficient Use of Workstations for Passive Monitoring of Local Area Networks"*. SIGCOMM'90 proceedings, Philadelphia, PA, Sept.1990, ACM

[5] Vern Paxson, *"Automated Packet Trace Analysis of TCP Implementations",* ACM SIGCOMM'97, September 97, Cannes, France.

[6] *"The New Generation of Network Monitoring Systems"*, ATG's Communications & Networking Technology Guide Series, 1997

[7] E. Magaña, J. Aracil, J. Villadangos, *"PROMIS: A Reliable Real-time Network Management Tool for Wide Area Networks"*, Proceedings of IEEE Euromicro'98, Vasteras, Suecia, Agosto 1998.