REVISTA DIGITAL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES
TELEM@TICA
Revista Telem@tica. Vol. 14. No. 2, mayo-agosto, 2015, p. 13-22 ISSN 1729-3804

PF_RING: SOLUCIÓN DE SOFTWARE LIBRE Y CODIGO ABIERTO PARA LA CAPTURA DE PAQUETES EN REDES DE ALTA VELOCIDAD

Donnie DeFreitas Ortega[1]

Ingeniero Informático
[1]e-mail: defreitas.ortega@gmail.com

RESUMEN

En menos de dos décadas la velocidad de las redes de datos ha incrementado en un factor de cien. Este volumen de datos tiende a abrumar las técnicas tradicionales de monitoreo de sistemas y puede resultar en la pérdida de paquetes en nodos sobrecargados. Todo análisis y/o decisión subsecuente por parte de sistemas de seguridad como los sistemas de detección de intrusos por ejemplo, resulta comprometida. Este artículo propone utilizar herramientas de software libre para agilizar la captura de paquetes en sistemas Linux y aplicable a redes de alta velocidad. Se introduce el módulo PF_RING como variante al tradicional método de captura del núcleo de Linux, exponiendo la base teórica sobre la cual funciona este módulo.

PALABRAS CLAVES: Captura de paquetes, Linux, PF_RING, redes de alta velocidad.

ABSTRACT

In less than two decades the speed at which we can connect to the Internet has increased by a factor of one hundred. Legacy techniques for systems surveillance tend to be overwhelmed and present packet lost at overloaded nodes. Any subsequent decision by security systems like Intrusion Detection Systems for example, is compromised. This article proposes the use of open source software to increase the capture rate of Linux based systems in high speed networks. The PF_RING module is introduced as a variant to the legacy packet capture method and presents the theoretic basis upon which the module was implemented.

KEYWORDS: *High speed networks, Linux, packet capture, PF_RING module.*

INTRODUCCIÓN

Hace aproximadamente dos décadas proveedores de servicio de Internet ofrecían acceso a velocidades máximas en el orden de los MB/s. Actualmente este valor se ha incrementado en un factor de 100, con conexiones entre uno a diez GB/s. Esto resulta en respuesta a la incesante demanda de ancho de banda por sus usuarios. No solo la cantidad de contenido sino la cantidad de usuarios se ha incrementado exponencialmente.

Cada vez más la red de datos se integra en la vida del ser humano [1, 2], y cada vez más la supervisión de estos sistemas es más importante. Pero los estudios en los sistemas de supervisión no van al mismo ritmo que otras innovaciones que por naturaleza humana son más interesantes. Actualmente los métodos tradicionales de captura de paquetes para un posterior análisis se ven fácilmente abrumados con la cantidad de tráfico que generan las redes Giga Ethernet. Esto conduce a la pérdida de paquetes que no pueden ser procesados por nodos sobrecargados y por tanto, compromete cualquier análisis o decisión subsecuente.

Existen soluciones a este problema, pero la mayoría son propietarias [3, 4] y requieren de dispositivos especializados. Esto implica costos de adquisición y operación significativamente elevados, y desde la perspectiva cubana pueden surgir complicaciones económico-políticas debido al embargo económico estadounidense. Además, el uso de soluciones propietarias restringe el desarrollo futuro del sistema ligándolo a la solución adquirida y en caso de que la proveedora de la solución deje de funcionar se pierde todo el soporte por el cual se pagó incrementando el riesgo de la inversión.

Debido a esto, este estudio se centra en el uso de herramientas libres, reduciendo hasta cero el costo de adquisición. Vale señalar que a pesar del ahorro económico, la solución propuesta en este artículo requiere de un periodo de aprendizaje y asimilación mayor que en el caso de las soluciones propietarias.

El requisito de esta investigación es proponer una alternativa de bajo costo para reducir la carga de procesamiento que conlleva la tarea de captura sobre el sistema, haciendo que un equipo de propósito general pueda ser un punto de monitoreo de red eficiente en redes de alta velocidad. Por esta razón el enfoque es sobre la captura de paquetes en redes de alta velocidad desde un sistema Linux. Específicamente, el módulo PF_RING, creado y mantenido por Luca Deri y su equipo en ntop [5], cómo este resulta en una mejora al método tradicional de captura de Linux (tcpdump).

CAPTURA DE PAQUETES CON PF_RING

De manera tradicional cuando un paquete llega a un sistema Linux es procesado por el controlador del NIC (Network Interface Card) por el núcleo del sistema y si es destinado a dicho sistema comienza el recorrido mostrado en la Figura 1 hasta llegar a la aplicación de usuario que le pertenezca. En redes de alta velocidad, donde el tráfico llega al orden de los GB/s, este proceso se convierte en un punto de embotellamiento para las aplicaciones de análisis y seguridad de la red que deben procesar el paquete, analizarlo y categorizarlo sin interferir en el funcionamiento de la red.

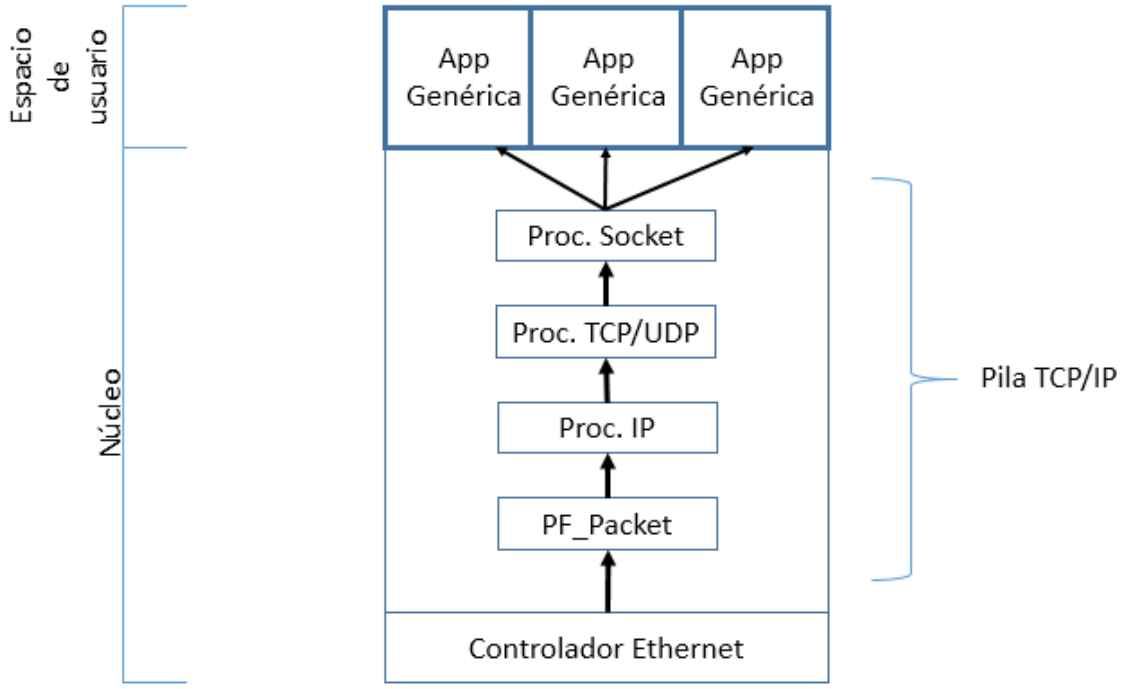

Figura 1 Recorrido de un paquete de red por la suite de protocolos de Internet del sistema [6].

Los estudios de Luca Deri y su equipo [7, 8, 9] han mostrado la ineficiencia de la captura de paquetes por las herramientas tradicionales de código abierto como tcpdump debido al recorrido del paquete a través de la pila de protocolos TCP/IP. Por esto, han desarrollado un *framework* llamado PF_RING [10] de libre distribución que soluciona el problema del embotellamiento creando una pila circular de copia directa de paquetes entre el núcleo del sistema y las aplicaciones a nivel de usuario.

La arquitectura de PF_RING que se muestra en la Figura 2 está compuesta por los siguientes bloques:

- El módulo acelerado del núcleo que brinda la copia a bajo nivel de paquetes a la pila circular de PF_RING.
- El API de PF_RING que brinda un soporte transparente para las aplicaciones en la capa de usuario.
- (Opcional) Controladores especializados, compatibles con PF_RING que permiten agilizar aún más la captura de paquetes al copiar los paquetes del driver a PF_RING sin pasar por las estructuras de datos del núcleo.

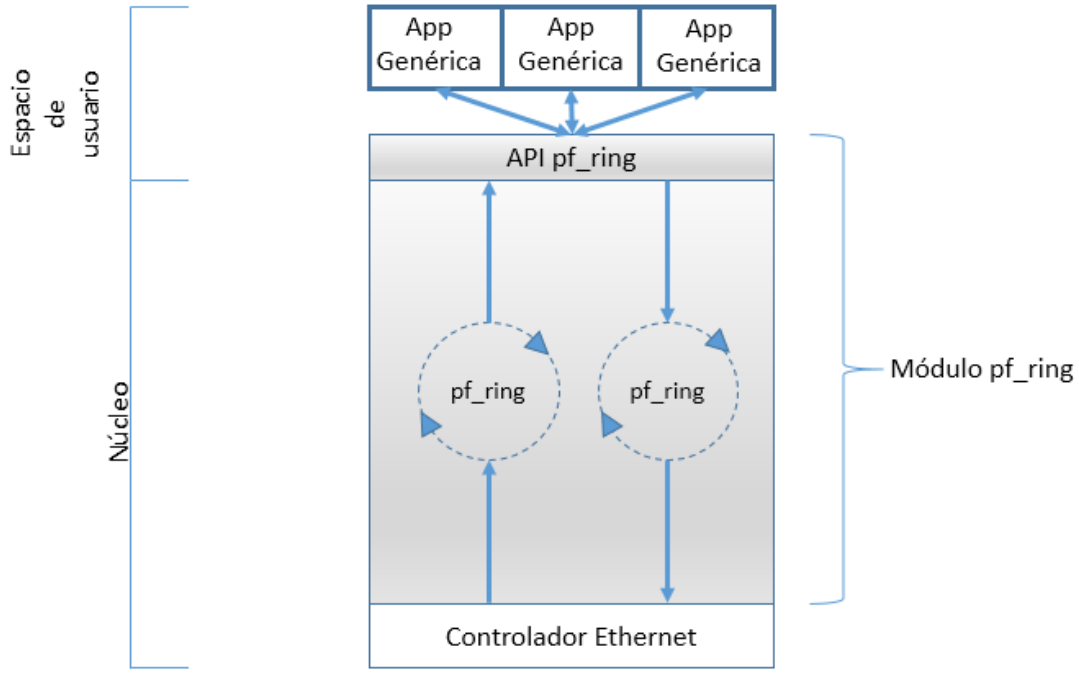

Figura 2 Arquitectura del módulo PF_RING y su integración al sistema [10].

PF_RING implementa un nuevo tipo de socket por el cual las aplicaciones en el entorno de usuario pueden comunicarse con el módulo del núcleo de PF_RING. Las aplicaciones pueden obtener lo que se conoce como *handles* (gestor de interrupciones en español) y realizar llamadas al interfaz de programación (API) de PF_RING. Cualquier *handle* puede ser ligado a una interface de red física, o a una cola de recepción (solo en adaptadores de red que soporten multi-cola), o a la interface virtual '*any*' lo cual acepta los paquetes entrantes/salientes en todas las interfaces del sistema.

Como se mencionó anteriormente, los paquetes se leen de un anillo de memoria que se reserva en creación. Paquetes entrantes se copian por el módulo del núcleo al anillo y leídas por las aplicaciones en el espacio de usuario. No hay reservación ni liberación de espacio en memoria por cada paquete. Cuando un paquete es leído, el espacio del anillo usado para el paquete recién leído será usado para acomodar nuevos paquetes. Esto implica que aplicaciones que desean mantener un registro de los paquetes debe almacenar los paquetes leídos ya que PF_RING no los preserva.

PF_RING soporta tanto los métodos tradicionales de filtros de paquetes Berkley (BPF) [11], basados en las librerías pcap (como *tcpdump*) y dos filtros adicionales (filtros tipo *wildcard* y tipo *precise* o preciso) que brindan un amplio rango de opciones para desarrolladores. Los filtros son evaluados por el módulo PF_RING y por tanto en el núcleo del sistema. Algunos adaptadores modernos como aquellos basados en la serie Intel 82599 o de Silicom Reditector NICs soportan filtros en el dispositivo que también son soportados por PF_RING. Los filtros en PF_RING (excepto los filtros por dispositivos) pueden tener una acción asociada, indicándole a PF_RING la acción a tomar cuando ocurre una coincidencia. Las acciones pueden ser de:

- Pasar o no el paquete a la aplicación en espacio de usuario.
- Dejar de procesar el filtro.
- Reflejar el paquete.

En PF_RING, reflejar un paquete es la habilidad de transmitir (sin modificación) el paquete que coincide con el filtro hacia una interfaz de red. Toda la operación de reflejar el paquete ocurre dentro del módulo PF_RING y la única solicitud hacia el espacio de usuario es la especificación del filtro.

AGRUPAMIENTO Y BALANCEO DE CARGA

El *framework* de PF_RING puede agilizar aún más las aplicaciones de captura y/o análisis de paquetes gracias a la implementación de dos mecanismos, agrupamiento (o *clustering*) y balanceo de carga (o *load balancing*). Estos mecanismos permiten que aplicaciones procesen sólo una parte del flujo de paquetes mientras que el resto sea direccionado hacia otros clústeres. Esto quiere decir que distintas instancias de Snort [11] (por ejemplo) pueden ser asociadas a diferentes clústeres (mediante la función API pfring_set_cluster) y juntas analizar todo el tráfico. Los paquetes son distribuidos por los clústeres de dos maneras, por flujo, que es la opción por defecto, o round-robin. La opción de por flujo agrupa en un clúster todo el tráfico donde coincide el protocolo, IP (fuente y destino), y puerto (fuente y destino). En caso de seleccionar la opción de round-robin todas las aplicaciones reciben la misma cantidad de paquetes. Por ejemplo, en la figura 3, si PF_RING crea su clúster por flujo, cada clúster (Clúster 1, 2,…, n) almacena un subconjunto lógico de tráfico de la red para ser analizado por una instancia de Snort.

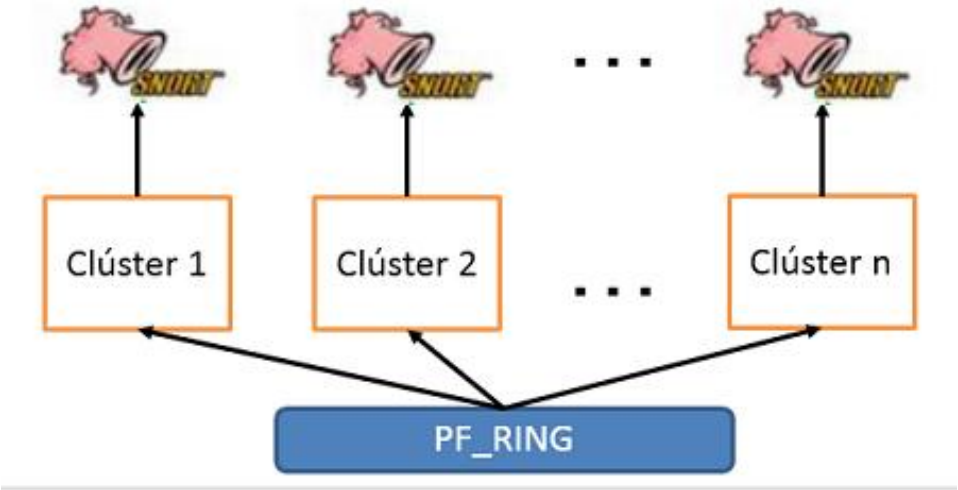

Figura 3 Mecanismo de clustering de PF_RING con Snort.

Por un lado, tenemos la opción de que preservar la lógica de la comunicación con el agrupamiento por flujo y la aplicación recibirá un subconjunto de todos los paquetes. Por el otro lado, si hay un flujo en

particular que consume la mayor parte del ancho de banda, repartir esta carga equitativamente por todas las instancias de Snort puede ser una mejor opción.

CONTROLADORES COMPATIBLES CON PF_RING

Como se mencionó previamente, PF_RING puede funcionar sobre los controladores estándar o con controladores especializados (disponibles con la descarga de PF_RING). El módulo del núcleo de PF_RING no cambia, pero dependiendo del driver utilizado cambian algunas funcionalidades y el rendimiento varía.

Los controladores compatibles con PF_RING pueden ser encontrados en la dirección PF_RING/driver/PF_RING-aware y son diseñados para mejorar la captura de paquetes al pasar directamente los paquetes hacia el anillo de PF_RING sin pasar por los mecanismos estándar de gestión de paquetes de Linux. Estos controladores permiten el uso del transparent_mode con opciones uno o dos. Además del uso de controladores compatibles, es posible utilizar diferentes tipos de controladores para incrementar aún más la captura de paquetes.

La primera familia de controladores es TNAPI (o Threaded NAPI), que permite pasar de manera más eficiente los paquetes hacia el módulo PF_RING mediante hilos del núcleo activados directamente por el TNAPI driver. Estos controladores están diseñados para mejorar la captura, pero no son capaces de transmitir paquetes (el camino de transmisión se deshabilita).

Para los casos donde se requiere de la máxima velocidad de captura con un 0% de utilización del procesador para copiar los paquetes al huésped (mecanismo NAPI no es usado) es posible utilizar un tipo de driver llamado DNA (Direct NIC Access). DNA permite leer directamente del interfaz de red saltándose simultáneamente el núcleo de Linux y el módulo PF_RING en modo de zero-copia. Con DNA, tanto la recepción como la transmisión son permitidas, pero como el núcleo es saltado algunas funcionalidades de PF_RING no están disponibles, como el filtrado de paquetes (tanto, BPF como de PF_RING).

A partir de PF_RING 6.0 se soporta una nueva familia de controladores, son los controladores compatibles con PF_RING y con soporte ZC (zero-copia). Estos controladores son compatibles estándar con soporte para la nueva librería PF_RING ZC. Estos controladores pueden ser usados de forma estándar en Linux, o en modo transparente, o en modo de salto del núcleo por completo con la librería ZC.

En modo transparente, PF_RING puede operar con tres opciones. El modo por defecto es ‘0’, en esta opción los paquetes son recibidos por el interfaz estándar de Linux y hacen el recorrido tradicional (Figura 1). En este modo no se utiliza el módulo núcleo de PF_RING, pero si dispone del API, permitiendo funcionalidades como el agrupamiento y balanceo de carga para optimizar la captura. La opción ‘1’ indica que los paquetes son copiados hacia el módulo núcleo de PF_RING (Figura 2) y también por la ruta tradicional. Si el modo es ‘2’ entonces los paquetes solo son copiados al módulo PF_RING (tcpdump no vera ningún tráfico).

Si se abre una interfaz en modo zc (por ejemplo pfcount –i zc:eth1) la interfaz resulta indisponible para la operación de redes normales ya que se pueden acceder de modo zero-copia (similar a su predecesor

DNA). Una vez que la aplicación utilizando el interfaz en zero-copia es cerrada entonces se resume el funcionamiento estándar de la red. Una interfaz en modo ZC brinda el mismo rendimiento que DNA.

APLICACIONES DE PF_RING

Como se mencionó previamente, PF_RING puede ser integrado con herramientas de análisis de tráfico como Snort, Bro [12], o Suricata [13]. Esto ha logrado potenciar significativamente la efectividad de estas aplicaciones, sobre todo en redes de alta velocidad. Algunos resultados realizados por Metaflows [14] utilizando Snort como un sistema de prevención de intrusos (IPS) y el conjunto de reglas del equipo de investigación de vulnerabilidades (VRT) [15], se muestran en la figura 4. El eje x representa el volumen de tráfico en Mbit/s y el eje vertical representa el porcentaje de paquetes que son reenviados (*forwarded*) por el sistema tras alunizarlos. El sistema utilizado como IPS era un Intel Core i7 con ocho núcleos y cuatro gigabytes de memoria. Se observa que gracias al balanceo de carga con PF_RING, cada núcleo es capaz de soportar aproximadamente 100 Mbit/s antes de presentar alguna pérdida de paquetes. Esto en comparación con la capacidad de 50 Mbit/s por un núcleo sin PF_RING.

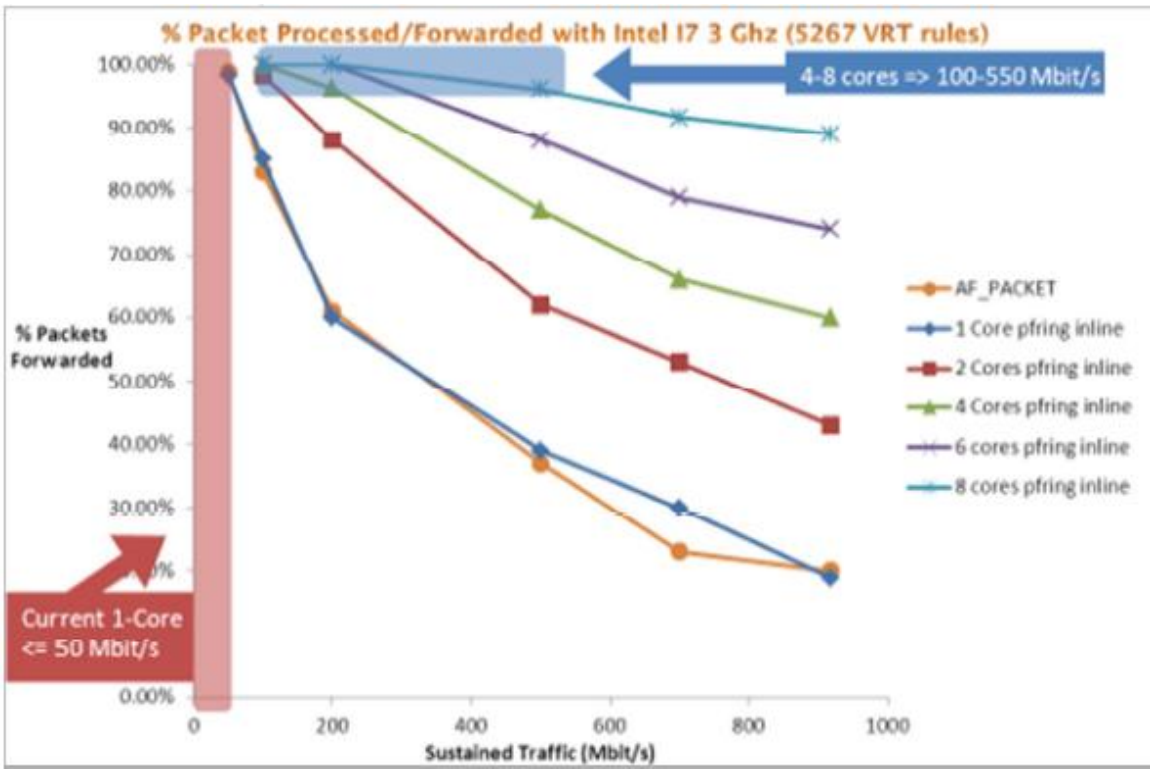

Figura 4 Resultados obtenidos por [14] al utilizar PF_RING con Snort en una red de alta velocidad.

En las tablas 1 y 2 se muestran los resultados obtenidos al observar el rendimiento de una puerta de enlace que cuenta con un procesador (CPU) Intel Core 2 Duo T7250 con cuatro Giga Bytes de memoria y sin la utilización de los controladores especializados de PF_RING. La función de este sistema era de servir como de puerta de enlace (*gateway*) entre dos áreas de red local (LAN), una cableada y otra inalámbrica. El experimento consiste en observar el incremento en el rendimiento de la puerta de

enlace mientras captura el tráfico bidireccional entre ambas LANs. De modo comparativo se utilizaron cuatro herramientas de captura de tráfico, *tcpdump*, *Wireshark*, *pfcount*, y *snort (*utilizando el conjunto de reglas de VRT de diciembre del 2014 para *snort)*. El tráfico se transmitió a una velocidad sostenida de 1 Mb/s, repitiendo el proceso posteriormente a 15Mb/s (velocidad que previamente fue observada como excesiva para herramientas de análisis con *snort*). Además se observó la diferencia entre el rendimiento de la puerta de enlace al utilizar solamente uno de sus procesadores o ambos disponibles.

Tabla 1 Incremento del rendimiento utilizando un CPU en la puerta de enlace.

Velocidad de transmisión (Mbps)	**Incremento en el rendimiento (desde reposo a modo captura) utilizando 1 CPU**			
	Tcpdump	***Wireshark***	***pfcount***	***snort***
1	7%	12%	<1%	48%
15	34%	42%	12%	84%

Tabla 2 Incremento del rendimiento utilizando dos CPU en la puerta de enlace.

Velocidad de transmisión (Mbps)	**Incremento en el rendimiento (desde reposo a modo captura) utilizando 2 CPU**			
	tcpdump	***Wireshark***	***pfcount***	***snort***
1	4%	7%	<1%	30%
15	20%	25%	5%	67%

Se pueden observar por estos resultados que herramientas de análisis como *snort*, experimentan una mejora significativa en la eficacia, aún sin utilizar los controladores especializados de PF_RING.

INSTALACIÓN DE PF_RING

PF_RING puede ser descargado directamente desde el sitio oficial (www.ntop.org) o por herramientas de colaboración como se explica en el sitio http://www.ntop.org/get-started/download. Al descargar PF_RING se adquieren los siguientes componentes:

- SDK (Software Development Kit) para el espacio de usuario.
- Una versión ampliada de la librería libpcap que de manera transparente utiliza PF_RING si está instalado y recae al funcionamiento normal en caso contrario.
- El módulo del núcleo de PF_RING.
- Una base de datos de controladores compatibles con PF_RING para diferentes NIC de varios distribuidores.

Al descomprimir el fichero en el directorio home/ se obtiene el directorio de PF_RING, y se puede compilar de la siguiente manera

$ cd ~/<directorio de pf_ring>

$ make

El módulo del núcleo de PF_RING requiere que el modulo linux-kernel-headers esté instalado antes de seguir. Si ya están instalados se continua con:

$ cd ~/<directorio de pf_ring>/kernel

$ make

Ahora antes de poder utilizar la aplicación de PF_RING hay que cargar el módulo del núcleo de esta manera

> # insmod ~/<directorio_de_pf_ring>/kernel/pf_ring.ko [transparent_mode=0|1|2] [min_num_slots=x] [enable_tx_capture=1|0] [enable_ip_defrag=1|0] [quick_mode=1|0]

Aparte de transparent_mode, que ya se explicó previamente, las opciones anteriores son, en orden de aparición:

1. **min_num_slots**: Indica la cantidad de ranuras que tendrá el anillo de PF_RING (valor por defecto es 4096)
2. **enable_tx_capture**: Valor por defecto es '1' el cual indica que se capturen los paquetes entrantes y salientes del sistema. Un valor de '0' indica que solo se capturen los paquetes entrantes.
3. **enable_ip_defrag**: Por defecto toma el valor de '1', el cual permite la desfragmentación de paquetes o '0' para desactivarlo. Solo el tráfico entrante es fragmentado.
4. **quick_mode**: Modo '1' acelera la captura, pero limita a no más que un socket PF_RING por interfaz de red.

PF_RING también viene con varios ejemplos de aplicaciones que utilizan la librería y pueden ser encontrados en el directorio <ruta a pf_ring>/userland/examples.

CONCLUSIONES.

El uso de PF_RING puede potenciar de manera significativa muchas aplicaciones de red que por su naturaleza requieren de un alto procesamiento. PF_RING hace que la captura de paquetes en Linux pueda ser realizada de manera barata y robusta a un costo económico menor que las soluciones propietarias disponibles. El módulo PF_RING puede convertir a un equipo de propósito general en un sistema de captura y monitoreo eficiente y eficaz; es compatible con las tarjetas de red más utilizadas en el mercado.

REFERENCIAS.

1. **CONTI, J, P:** "*The Internet of Things*", 2006, disponible en http://digital-library.theiet.org/content/journals/10.1049/ce_20060603
2. **CHUI M, ET AL:** "*The Internet of Things*", 2010 disponible en http://115.112.165.74:81/Krishna%20Akalamkam/digital%20marketing/articles/The%20Internet%20of%20Things_.pdf
3. **PAOLINI M** : *"DPI and Policy: Complementary tolos for network optimization and new revenue creation*". SENZA Fili Consulting 2013
4. **MOCHALSKI K, SCHULZE H**. : *Deep Packet Inspection: Technology, Aplications & Net Neutrality* [whitepaper]. 2009
5. **DERI, L; ANDREWS, R**. "*High-Speed Passive Packet Capture and Filtering*" [slides]. Proceedings of AIMS 2008, July
6. **RODRIGUEZ GONZALEZ, D** : **.**:"Diseño de escenario óptimo que permita implementar DPI" (en español) Tesis de Grado en Telecomunicaciones, Instituto Superior Politécnico José Antonio Echevarría, Cuba, 2012
7. **DERI, L:** *Open Source in network administration: the ntop Project* Open Source Conference, Athens 2008
8. **FUSCO F, DERI L:** ¨*High Speed Network Traffic Analysis with Commodity Multi-core Systems*¨ 2010
9. **DANELUTTO M, DERI L, DE SENSI D:** ¨*Network Monitoring on Multicores with Algorithmic Skeletons*¨ Computer Science Department, University of Pisa, Italy, 2012
10. **DERI, L :** . *PF_RING User's Manual: Linux High Speed Packet Capture*. Ntop.org Abril 2014
11. **ZHOU Z, ZHONGWEN C, TEICHENG Z:** *"The study on network intrusion detection system of Snort"* IEEE, 2010 disponible en http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5479341
12. **GUPTA S :** *"A Graphical User Interface Framework for detecting Intrusions using Bro IDS"* 2012 disponible en http://dspace.thapar.edu:8080/dspace/handle/10266/1891
13. **HOQUE MS, MUKIT M, BIKAS M, NASR A :** *"An implementation of intrusion detection system using genetic algorithm"* 2012 disponible en http://arxiv.org/abs/1204.1336
14. **RICCIULLI L, COVEL T :** *"Inline Snort Multiprocessing with PF_RING"* Septiembre 2011
15. **VALGENTI VC, SUN H, KIM MS :** *"Protecting Run-Time Filters for Network Intrusion Detection Systems"* IEEE 2014 disponible en http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6838655