Universidad Autónoma de Madrid

Escuela Politécnica Superior

Máster conjunto en Ingeniería Informática e Investigación e Innovación en Tecnologías de la Información y las Comunicaciones

TRABAJO DE FIN DE MÁSTER

ESTUDIO DE CAPTURA Y ALMACENAMIENTO DE TRÁFICO EN REDES FÍSICAS Y VIRTUALES MULTI-GIGABIT

Rafael Leira Osuna
Tutor: Iván González Martínez

19/06/2015

ESTUDIO DE CAPTURA Y ALMACENAMIENTO DE TRÁFICO EN REDES FÍSICAS Y VIRTUALES MULTI-GIGABIT

Autor: Rafael Leira Osuna
Tutor: Iván González Martínez

High Performance Computing and Networking
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid

19/06/2015

Agradecimientos

Son muchas a las personas a las que debería agradecer y poco el espacio en donde escribir. A Juan Sidrach, por completar y mejorar esta plantilla en LaTeX, así como a uno de sus creadores originales, Diego Hernando, el cual me ha apoyado siempre a lo largo de carrera y el Máster. A Iván González, por haber sido un magnífico y comprensivo tutor y por ayudarme siempre en todo lo que ha podido. Gracias a el y a la ayuda de Victor Moreno, he podido finalizar con éxito este Trabajo Fin de Máster. Al equipo de handbe: Diego Hernando, Carlos Asensio y Paloma Domínguez, con los cuales he pasado muchas horas llevando a cabo nuestro proyecto y que también me han apoyado de múltiples formas. A Raúl Martín, porque sin esos viajes en coche y nuestras conversaciones en el gimnasio, el año hubiese sido mucho más aburrido.

Y en general a todos los compañeros del grupo de investigación HPCN, ya que en mayor o menor medida han puesto su granito de arena a lo largo del Máster. Menciono especialmente a: Rubén García-Valcárcel, David Muelas, Mario Ruiz, José F. Zazo, Isabel García y a Sergio López Buedo. De la misma manera agradezco el apoyo a todas y cada una de las personas que he conocido a lo largo de la carrera y del Máster, a mis amigos de toda la vida, a familiares y a Bea que de una u otra forma han estado conmigo a lo largo de este doble Máster haciéndolo más ameno y finalmente llevándome a escribir este TFM, y con él, a terminar mi doble Máster.

Abstract

Abstract — Study and analyze a high speed network (≥10Gbps) is a challenge in terms of the amount of data to be processed and the data rate itself. As a result, the networking capture tools are usually very complex. Those tools also have to be continuously adapted to new technology and higher data rates. To meet those requirements, each capture tool implements its own formats and way to capture that difficulties its interoperability. In order to solve this problem, it is necessary to develop a capture tool that stores and works with network data in a well-known format. Standard formats, like *PCAP*, allow different applications to work together easly, even in a paralel way. In the same way, common formats frees network analyzing tools from the underlying network.

Typically, expensive dedicated servers are used to capture, store and process network data at high speed rates. However, this is changing due to the proliferation of *cloud computing* and the greatly improved performance virtualization technology. This trend makes difficult to find bare-metal servers or even network equipment in some environments. Therefore, it is becoming more and more important to evaluate the performance and feasibility of capture and process network data on virtual environments. To achieve that, a capture and store tool has been developed. The tool can work at 10 Gbps thanks to *Intel DPDK* capture technology. A technology, that also can work in both bare-metal and virtual environments. In this work, different methods and capture tools are compared. In the same way, different virtualization methods provided by *KVM* are evaluated. While running applications in virtual machines have a small overhead compared with the bare-metal version, results show that performance in virtual environment is really close to bare-metal environment. However, those results can only be reached using the correct configuration and the latest advantages of the state-of-the-art hardware devices.

Key words — Virtual network functions, packet capture, virtual machines, Intel DPDK, HPCAP

Resumen

Resumen — Estudiar y analizar el comportamiento de una red a alta velocidad ($\geq$10 Gbps) supone un reto constante a medida que aumenta la velocidad de las redes de comunicaciones debido a la gran cantidad de datos que se generan a diario y al propio hecho de procesar información a tales velocidades. Por estos motivos, las herramientas encargadas de la captura de datos son complejas y se encuentran, por lo general, en constante adaptación a las nuevas tecnologías y velocidades, lo que dificulta considerablemente su integración directa con otras aplicaciones de motorización o análisis de datos. Por ello es necesario que estas herramientas sean capaces de capturar y almacenar los datos en un formato estándar en el que otras herramientas puedan trabajar a posteriori o incluso en paralelo, con los datos de red independientemente de la tecnología de captura utilizada.

Típicamente, este proceso de captura, almacenamiento y procesamiento de datos a alta velocidad se ha realizado en máquinas dedicadas. No obstante, debido a la proliferación del *cloud computing* y a la gran mejora en rendimiento de la tecnología de virtualización, esto está cambiando, pudiéndose llegar al caso en el que sea raro disponer de una máquina física en la que realizar estos procesos. Por ello, evaluar la viabilidad de realizar estos procesos de tan alto rendimiento dentro de entornos virtuales comienza a cobrar importancia. Dentro de este contexto, se ha desarrollado una herramienta de captura y almacenamiento en disco a 10 Gbps mediante la tecnología de captura *Intel DPDK*, con la capacidad de funcionar tanto en entornos físicos como virtuales. Del mismo modo, en este trabajo se presentan y se comparan diferentes métodos y herramientas de captura, así como los diferentes métodos de virtualización de componentes que ofrece *KVM*. A pesar de que el uso de máquinas virtuales impone un sobrecoste computacional a cualquier aplicación, los resultados obtenidos muestran que el rendimiento en entornos virtuales se asemeja mucho al rendimiento en entornos sin virtualización, siempre y cuando se utilice la configuración adecuada que exprima las capacidades de los dispositivos actuales.

Palabras clave — Funciones de red virtuales, captura de paquetes, máquinas virtuales, Intel DPDK, HPCAP

Glosario

cache Se suele llamar cache a una pequeña memoria de alta velocidad, que comunica una memoria lenta de tamaño muy superior y otro dispositivo. Por el principio de localidad y de temporalidad, almacenar los datos en una caché permite a, por ejemplo, microprocesadores, tener un acceso a los datos mucho más rápidamente. 8

CAPEX El término inglés CAPEX, se refiere fundamentalmente a los gastos iniciales (o de actualización si es el caso), que son necesarios para poner en marcha un producto, o construir algo. 1

Cloud Computing Se define la computación en la nube, como un conjunto de servicios que se encuentran en internet. Dichos servicios suelen estar descentralizados y repartidos en localizaciones y elementos físicos a los que usualmente no se tiene acceso. 1, 14

core Entendemos como *core*, como la unidad de procesamiento más básica de la que dispone nuestro sistema. Entendemos igualmente, que un *core*, solo puede ejecutar un único hilo o proceso simultáneamente sin que su rendimiento se vea perjudicado. 7, 24, 25, 31, 33, 34

CPD Un centro de datos (o Data Center), suele estar formado por varias habitaciones habilitadas para almacenar entre decenas y centenas de equipos. Estas salas se encuentran climatizadas, de forma que se mantenga una temperatura adecuada para el funcionamiento de los diferentes equipos. Estos centros de datos, suelen ofrecer a sus clientes luz y acceso a Internet a cambio de una cuota por cada U consumida. 14

CPU La tan conocida Unidad de procesamiento central, o CPU, se encarga de realizar la mayor parte de las tareas que realiza un ordenador de hoy en día. En las últimas arquitecturas, es también la encargada de coordinar diferentes dispositivos como la memoria o el bus PCIe. 2, 3, 6, 8, 24

HPTL La librería HPTL partió de una primera implementación en mi trabajo fin de grado [1]. Debido a la aparición de problemas de mantenibilidad y de configurabilidad, se decidió externalizar su uso como librería y publicarla en github [2]. 18

HugePage Un ordenador convencional divide la memoria del sistema en bloques alineados e indivisibles denominados páginas. Cada una de estas páginas pueda ser asignada a un proceso diferente o ser escrita a disco. De forma tradicional, estas páginas han tenido un tamaño de 4 KB, no obstante, manejar páginas tan pequeñas puede llegar a suponer una enorme sobrecarga para el sistema operativo. Por este motivo, los nuevos sistemas soportan páginas de 2 MB o incluso 1 GB. A estas dos últimas, se las conoce por el nombre de HugePages. 21, 24, 25

hypervisor Se denomina hypervisor a la capa virtual que conecta una máquina virtual con los diferentes dispositivos físicos de la máquina anfitriona. Este elemento, también es el encargado de la paravirtualización y de la virtualización completa de dispositivos. 11, 12

KVM La tecnología KVM [3] es una de las más populares en cuanto a virtualización se refiere, junto con su mayor competidor XEN. Esta tecnología se encuentra presente en el Kernel de Linux (en formato de módulo) y ofrece desde este nivel de ejecución la capacidad de virtualizar diferentes máquinas y dispositivos. Gracias a la tecnología actual de los procesadores (VT-X/AMD-V), esta virtualización puede llegar a ser transparente para la máquina virtualizada, además de ofrecer un aislamiento entre las diferentes máquinas virtuales y la propia máquina física. 3

monitorización activa La monitorización activa consiste en la realización de una serie de medidas de una red, en donde dos o más nodos de la misma, intercambian un conjunto de paquetes. A partir de estos paquetes, se obtienen medidas como ancho de banda, jitter, latencia o cantidad de paquetes perdidos. 5

monitorización pasiva La monitorización pasiva consiste en la captura de tráfico en uno o varios puntos de la red. Analizando el tráfico capturado es posible obtener multitud de información, incluidos problemas a niveles de aplicación. 5, 6, 13

MTU En el contexto de las redes de comunicaciones, se considera MTU al tamaño máximo que puede tener un paquete en un determinado enlace. Normalmente esta medida viene dada por una restricción del medio físico subyacente, no obstante, si no hubiese una limitación en la longitud de los paquetes, una aplicación podría abusar y emitir un paquete de tamaño infinito que colapsase indefinidamente la red. 21

multicore Dentro de este contexto, la palabra *multicore* hace referencia a los sitemas con más de una unidad de cómputo, ya sea porque hay más de un procesador completo, o se trate de un procesador con más de un *core*. 7

NUMA La arquitectura NUMA se basa en la descentralización de los recursos de un ordenador. De esta forma, cada procesador tiene su propia memoria y sus propios dispositivos. Si un procesador necesita acceder a los recursos de otro, debe pedirselos al procesador propietario de los mismos, el cual deberá interrumpir alguno de sus procesos para atender la petición. Estas peticiones, se realizan utilizando enlaces de altísima velocidad (como QPI), pero igualmente suelen ocasionar una degradación de rendimiento. 7

one-copy Hablamos de one-copy, cuando una aplicación no requiere realizar una única copia de los datos (Normalmente de la memoria del kernel a la de usuario) para trabajar con ellos. 8, 9, 27, 28, 31, 37, 38

OPPEX El término inglés OPPEX, se refiere fundamentalmente a los gastos que tiene un producto tras su contratación o compra. 1

Passthrough Se denomina Passthrough, al mecanismo que tienen las máquinas virtuales para apoderarse por completo de un determinado dispositivo, "desconectándolo" de la máquina anfitriona. De esta forma, el sistema operativo invitado, es capaz de utilizar los drivers de forma nativa con el dispositivo y en principio con perdida de rendimiento 0. 3, 20, 25, 29–32

RAID 0 Se llama comunmente RAID a un grupo de discos duros que actúan en conjunto mostrándose al sistema operativo como un único disco duro. Un raid 0, es un caso específico en el cual todos los discos que forman el RAID, se encuentran al mismo nivel, es decir, no existe ningún disco que replique datos. 17

SR-IOV También conocida como *Single Root I/O Virtualization*, SR-IOV [4] es una tecnología de virtualización que permite a un único dispositivo PCIe, mostrarse como diversas funciones virtuales (VF). Cada una de estas VF, puede conectarse a una máquina virtual, de forma que la compartición del dispositivo sea realizada por el propio hardware. Esto, minimiza el coste computacional de la máquina anfitriona, así como de las diversas máquinas virtuales que utilizan el dispositivo. 2, 3, 20, 29, 32, 33, 37

U Bajo el contexto de los equipos enracables, se denomina una U a la unidad básica de espacio que ocupa un equipo en un rack. 14

vanilla Se denomina a un elemento vanilla, cuando dicho elemento se encuentra en su estado original, sin modificaciones. 9, 11, 13, 21, 24, 27, 33

VirtIO La tecnología VirtIO [5], forma una parte importante de la tecnología de virtualización de KVM. El concepto de VirtIO, abarca desde los drivers de la máquina virtual, hasta los propios dispositivos virtuales generados por KVM. Un dispositivo VirtIO es, en esencia, una paravirtualización del dispositivo original. 3, 10, 12, 16, 24, 29, 31, 34

zero-copy Hablamos de zero-copy, cuando una aplicación no necesita realizar ninguna copia de los datos (Normalmente de la memoria del kernel a la de usuario) para trabajar con ellos. 7–10, 28, 31, 37

Acrónimos

CAPEX CAPital EXpenditures. 1, *Glosario:* CAPEX

CPD Centro de procesamiento de datos. 14, 29, *Glosario:* CPD

DPDK Data Plane Development Kit. 3, 10, 11, 13, 16, 18, 20, 21, 23–25, 27, 28, 32–34, 38

DPI Deep Packet Inspection. 6, 7, 10

FPGA Field Programmable Gate Array. 6, 21, 38

Gbps Gigabit per second. 1, 3, 7–11, 13, 16, 17, 20–23, 27, 28, 31, 33–36

GPU Graphic processing unit. 6, 8

HPTL High Performance Timing Library. 18, *Glosario:* HPTL

ISP Internet Service Provider. 1, 23

KVM Kernel-based Virtual Machine. 3, 12, 23, 24, 29, *Glosario:* KVM

MPE Multicast Promiscuous Enable. 34

MTU Maximum Transmission Unit. 21, *Glosario:* MTU

NFV Virtual Network Function. 2–4, 10, 11, 15, 16, 24, 25, 32–35, 37

NIC Network Interface Card. 7, 10, 13, 18, 20, 21, 24, 25, 27, 28, 31, 33, 34, 37

NUMA Non-Uniform Memory Access. 7, 20, 35, *Glosario:* NUMA

OPPEX OPerational EXpenditures. 1, *Glosario:* OPPEX

OS Operative System. 11

QoS Quality of Service. 6

SDN Software Defined Network. 2

SSD Solid-state drive. 16, 17

VF Virtual Function. 2, 12, 24, 33, 38

VM Virtual Machine. 1, 2, 11, 12, 14, 15, 24, 29–31, 33–36

Índice general

Índice de tablas

Índice de figuras

1

Introducción

Internet, es conocida como la red de redes. Cada año, más y más dispositivos se une a esta gran y colosal red. Una red, que desde hace mucho dejó de estar formada por solo ordenadores caseros y servidores, para ser un conjunto muy heterogéneo de dispositivos (como móviles [6], tablets e incluso neveras o relojes). Este gran aumento en la cantidad de dispositivos que conforman la red, ha sido parejo al aumento de la velocidad de la red, gracias a una gran cantidad de avances tecnológicos. Si bien, ahora es fácil poder disponer de un enlace simétrico a 100 o incluso a 300 Mbps en nuestros hogares, no podemos obviar que estos enlaces deben acabar por ser agregados en algún punto, convirtiéndose así en enlaces de 10, 40 o incluso 100 Gigabit por segundo (Gbps).

Aunque los enlaces de 100 Gbps comienzan a proliferan en los grandes proveedores de servicios de internet (ISPs), aun son complicados de encontrar en otras grandes empresas. No obstante, si que es posible encontrar algunos enlaces e incluso subredes enteras a 10 Gbps. Mantener en funcionamiento redes de alta velocidad ($\geq$10 Gbps) requiere de una infraestructura de red con un alto gasto de capital inical (CAPEX) y un alto gasto de capital para operar (OPPEX). Dicha infraestructura, debe ser además monitorizada con regularidad para asegurar el correcto funcionamiento.

Toda la infraestructura de red así como el equipamiento de monitorización requieren, de forma tradicional, grandes y potentes máquinas dedicadas a realizar las típicas tareas de *enrutado*, *captura* o *análisis de tráfico*, entre otras. No obstante, este tipo de necesidades no es nuevo. La mayoría de los servidores del mundo funcionaban en caros servidores dedicados, hasta la llegada del Cloud Computing. El Cloud Computing virtualiza los recursos clásicos de computación permitiendo a una compañía descentralizar de forma sencilla y barata sus servidores a lo largo del mundo, así como amoldarse en capacidad de cómputo a la demanda de sus clientes. Esto, inevitablemente minimiza el CAPEX y el OPPEX que una empresa debe afrontar para operar.

Internamente los diversos sistemas de Cloud Computing utilizan máquinas virtuales (VMs) para ofrecer sus servicios. Estas máquinas virtuales, deben a su vez conectarse entre sí y al resto de dispositivos dando lugar a la aparición redes virtuales. Sin embargo, la virtualización completa los dispositivos de red, no ofrece una escalabilidad en velocidad y tamaño fácilmente, o no al

menos, sin un alto coste y consumo de procesamiento CPU.

Para evitar la virtualización completa de dispositivos hardware, los fabricantes de microprocesadores desarrollaron la tecnología virtualización: VT-X (Intel) y AMD-V (Amd). Dicha tecnología permite crear un nivel de aislamiento entre la máquina anfitriona y las máquinas virtuales que hospeda. Aunque esta tecnología supuso un gran avance, una gran máquina capaz de ejecutar diversas máquinas virtuales necesitaría tantos recursos (tarjetas de red, tarjetas gráficas, tarjetas raid, etc) como máquinas virtuales ejecutase. Esto, a pesar de suponer un avance y en cierta medida un ahorro, seguía siendo una solución cara pues no siempre una máquina virtual tenía porque exprimir al máximo un dispositivo hardware completo.

Para solventar el problema y poder compartir dispositivos hardware entre diferentes máquinas virtuales, se diseñó la tecnología SR-IOV [4]. Los dispositivos PCI compatibles con SR-IOV permiten representar un único dispositivo, como un conjunto de elementos PCI denominados funciones virtuales (VFs). Cada VF tiene sus propios recursos virtuales (proporcionados por el hardware anfitrion) al igual que su propio espacio de memoria en el sistema, lo que lo convierte a efectos prácticos en un dispositivo PCI independiente. Cada uno de estos recursos virtuales proporcionados por las VFs, son además gestionados por el dispositivo PCI físico. De esta forma, cada VF puede ser asociada a una máquina individual diferente, siendo el propio dispositivo hardware el encargado de la compartición y balanceado de los recursos.

Gracias a estos avances, el mundo de las redes de comunicaciones ha podido introducir el concepto de función de red virtual (NFV). Siendo una NFV, una simple VF de una tarjeta de red de alta velocidad. Estas NFVs, representan adaptadores de red de la misma velocidad que el adaptador físico, actuando este último como si de un switch se tratase con una (o varias) salidas al exterior. El uso de esta tecnología, permite crear fácilmente conexiones de alta velocidad y rendimiento entre máquinas virtuales. No obstante, hasta donde llega mi conocimiento, actualmente no existe ninguna herramienta capaz de realizar una captura entre dos (o más) NFVs.

Este tipo de tecnología, es muy novedoso, lo que conlleva una serie de ventajas e inconvenientes. La peor parte radica en la falta de medidas empíricas de rendimiento de estos sistemas de virtualización. Recordemos, que procesos como la monitorización de tráfico a altas velocidades supone un coste computacional elevado. Valorar si una VM es capaz de realizar esas tareas y bajo que condiciones, es fundamental para que futuros arquitectos de redes tomen decisiones a la hora de implementar arquitecturas de red virtuales, o también conocidas como redes definidas por software (SDNs).

1.1. Objetivos

Los objetivo principal de este trabajo de este doble trabajo fin de máster es realizar una comparativa entre las diferentes tecnologías de captura, así como de los diferentes métodos de virtualización. Para ello:

- Se ha construido con la tecnología Intel Data Plane Development Kit (DPDK), una herramienta de captura a 10 Gbps capaz de:
  - Capturar en cualquier entorno (virtual o físico) la mayor cantidad posible de paquetes, sin realizar ninguna tarea con ellos. De esta forma, se pretende evaluar la tasa máxima de captura en ambas situaciones.
  - Almacenar los paquetes capturados en diferentes ficheros PCAP, de forma que cualquier otra herramienta pueda funcionar con ellos.
- Se ha utilizado la herramienta anterior, y compararla con herramientas de captura y/o almacenado a disco actuales, dentro de sus respectivos entornos de funcionamiento.
- Se han evaluado las diferentes técnologías de virtualización ofrecidas por una máquina virtual basada en el Kernel (KVM): VirtIO, Passthrough y SR-IOV.

1.2. Fases de realización

Para poder realizar las mediciones adecuadas, fue necesario iniciar con el desarrollo de una herramienta ligera de captura con Intel DPDK. De esta forma, es posible asegurar que se mide la tasa de recepción en el mejor escenario posible: en el que la CPU se encuentra ociosa. Una vez completada la herramienta, era posible empezar a probar diferentes combinaciones y elementos, siendo la primera de ellas, la física. Se idearon un conjunto de scripts con los cuales se podía realizar la experimentación explotando los recursos de almacenamiento (mediante herramientas como dd) y los recursos de captura (como la herramienta en DPDK).

Tras disponer de las pruebas más básicas, se comenzó el desarrollo de una aplicación de captura a disco realizada de nuevo en el framework de Intel DPDK. A pesar de parecer una tarea sencilla, está compuesta por: la captura de paquetes (copia hasta el host), procesamiento (obtener el tiempo, etc), formateo del fichero PCAP, y almacenamiento de paquetes de 64 byte a 10 Gbps. Esto significa, que todas esas subtareas deben realizarse en menos de 67 ns, lo que supone un verdadero reto. Construir una aplicación capaz de superar las restricciones de rendimiento llevo bastante tiempo.

Una vez todas las herramientas estuvieron realizadas, se procedió a la comparación con las diversas herramientas de captura existentes (ver capítulo 2) en un entorno físico. Conocidos los rendimientos en los casos más habituales, se procedió a instalar y crear diversas máquinas virtuales sobre las que realizar las pruebas. Tal y como se cuenta en la sección 4.1, se han probado multitud de configuraciones de los diferentes dispositivos virtuales, lo cual acarreó un tiempo no despreciable. Cabe destacar que la generación de NFVs, supone una tarea computacional no despreciable.

Finalmente, se repitieron las mismas pruebas de los entornos físicos en los diferentes y diversos entornos virtuales, para aquellos motores de captura y almacenamiento que soportaban la virtualización.

1.3. Estructura del documento

El documento se ha intentado dividir de forma que permita a un lector con pocos conocimientos profundizar en el campo de la virtualización y sea capaz de entender y aprovechar los conceptos descubiertos de una forma fácil.

En el capítulo 2, se presentan las diferentes tecnologías de virtualización con un mayor detalle que el aportado en la introducción. De igual modo, se presentan los diferentes métodos de captura y recepción que se han utilizado posteriormente en las pruebas, así como su relevancia dentro del mercado. También se comentan y se remarcan algunos ejemplo de uso de NFV en el mundo de las redes de comunicaciones, y que han dado lugar a trabajos de gran relevancia.

En el capítulo 3, se presenta la arquitectura de las herramientas desarrolladas, mientras que en el capítulo 4 se presenta el desarrollo de las pruebas y las comparativas realizadas entre ellas. En este capítulo, en la sección 4.1 se introduce el entorno de desarrollo sobre el que se operarán las pruebas, tanto a nivel hardware (subsección 4.1.1) como a nivel software (subsección 4.1.2). En la sección 4.3 se presenta la definición de las pruebas realizadas, así como los primeros resultados en un entorno físico, mientras que en la sección 4.4 se presentan los resultados para las pruebas en entornos virtuales, así como los problemas y conclusiones obtenidas en las diferentes pruebas.

Finalmente, en el capítulo 5 se cierra el documento mostrando las lecciones aprendidas así como las futuras lineas de trabajo.

2

Estado del Arte

Hoy en día, resulta complicado imaginar Internet como una red exclusiva de clientes y servidores. El fenómeno "Internet of Things" [7] ha supuesto un cambio radical a hora de percibir que es Internet. Del mismo modo, que resulta difícil imaginarse a alguien sin un smartphone [6] o sin acceso a Internet [8]. Este crecimiento ha supuesto un equivalente aumento tanto en la velocidad como en tamaño de las redes internas de las empresas.

Para mantener una red de cierto tamaño en un buen estado, es necesario monitorizarla [9]. Comunmente se habla de 2 tipos de monitorización: Activa y Pasiva. Cada tipo de monitorización ofrece una serie de ventajas e inconventientes. La monitorización activa, permite a un administrador de red conocer el estado y la calidad de una red, pues este tipo de mediciones ofrece métricas como ancho de banda disponible, el jitter de la red (El cual es muy importante a la hora de transmitir contenido multimedia [10–12]) o la latencia de la red entre dos extremos. No obstante, la monitorización activa afecta de mayor o menor forma al estado de una red, lo que puede perjudicar a la medida y por supuesto a los usuarios de la propia red. Estos efectos perjudiciales, pueden verse ampliados si la monitorización no se realiza mediante técnicas adecuadas [13].

Por otro lado, la monitorización pasiva no sufre de los efectos perjudiciales de la monitorización activa. Aunque este tipo de monitorización no es capaz de medir el jitter entre dos puntos de la red, ni tampoco la latencia a la que se ven sometidas determinadas comunicaciones, si ofrece otras muchas métricas relevantes. Para realizar este tipo de monitorización en una enlace de alta velocidad, es necesario un equipo de elevadas prestaciones para capturar el tráfico [14–19].

Al disponer de todo el tráfico de un enlace, es posible obtener métricas de este, como pueden ser el ancho de banda del propio enlace, o de una determinada aplicación o usuario, la detección de intrusiones o incluso detección de malfuncionamiento de determinadas aplicaciones. Aunque las posibilidades de la monitorización pasiva son muchas, tratar con toda esta cantidad de tráfico también supone una gran carga computacional. Por ello, existe multitud de investigación acerca de como escoger el tráfico que debe ser posteriormente almacenado o procesado. Una de las aproximaciones más conocidas para segmentar el tráfico, es lo que se llama *packet sampling* [20]. Este tipo de algoritmos intentan seleccionar el tráfico más relevante, de forma que sea posible

obtener información estadística de ellos y a su vez reducir de manera drástica el número de paquetes por segundo que deben ser procesados. No obstante, utilizar *packet sampling* tiene un cierto impacto en la medida [21], además de limitar el número de métricas interesantes que pueden obtenerse a partir de la monitorización pasiva.

Otros métodos de selección de tráfico, se basan en la selección por protocolo. Dado que existen miles de protocolos, es posible que de cara a realizar una monitorización solo nos preocupen algunos en concreto. Aunque identificar un protocolo a nivel de red o transporte es una tarea trivial, no lo es a la hora de identificar un protocolo a nivel de aplicación [10]. Por este motivo, existen 3 formas diferentes de clasificar tráfico: por puerto, mediante clasificación estadística o mediante Deep Packet Inspection (DPI). La clásica clasificación por puerto, nos brinda velocidad en la clasificación, pero no es útil al a hora de clasificar protocolos sin puertos conocidos, detectar atacantes o errores de configuración. La clasificación estadística por otro lado, nos brinda un compromiso entre fiabilidad y velocidad, lo que ha propiciado una elevada cantidad de investigación [22–25]. No obstante, la clasificación estadística requiere disponer de una muestra preliminar del tráfico a partir de la cual obtener un modelo que utilizar posteriormente. De igual modo, es complicado confiar en este tipo de métricas si se desea aplicar algún tipo de reacción activa como denegar a un usuario el acceso a un recurso, a pesar de que la métrica acierte en el 99 % de los casos. En el caso de DPI, es necesario aplicar una gran cantidad de cálculo para obtener el resultado [26]. Por ello, se han desarrollado aplicaciones de DPI que recurren a diversos coprocesadores como unidades de procesamiento gráfico (GPUs) [10, 27] o dispositivos de hardware programable (FPGAs) [28–31]. La propia detección de protocolos, aporta suficiente información como para la construcción de firewalls [32, 33], detección de intrusiones [27], corrección de Quality of Service (QoS) [10] entre otras muchas aplicaciones.

En redes de altas velocidades, en muchos casos resulta poco verosímil trabajar a nivel de paquete y frecuentemente resulta poco interesante. Por este motivo, la extracción de la información de flujo con su correspondiente análisis se vuelve fundamental [34]. Aunque generar la información de cada uno de los flujos supone procesar igualmente cada uno de los paquetes, resulta un proceso computacionalmente más barato que una clasificación DPI. Este procesado de flujos tiene una gran cantidad de desarrollo por detrás ya que facilita el análisis posterior de una red. El tratamiento de flujos es un tema extenso, desde los estándar de NetFlow [35, 36] e IPFIX [37, 38], hasta el desarrollo de herramientas para CPU [12, 34, 39, 40] o para diversos coprocesadores [41].

En definitiva, para poder analizar la información de los flujos de una red, obtener la información referente a los protocolos o simplemente transmitir y recibir paquetes a alta velocidad, es necesario un conjunto de herramientas capaz de capturar y emitir el tráfico exprimiendo todo el ancho de banda del enlace.

2.1. Herramientas de captura

Realizar un proceso de captura a alta velocidad en un ordanador convencional, supone atravesar toda la pila de red de un sistema operativo, lo que de forma inevitable nos supone una degradación del rendimiento y en el caso de redes de ≥10 Gbps, un impedimento a la hora de capturar todo el tráfico. Para solventar este problema, se han desarrollado multitud de herramientas y hardware dedicado. Dos claros ejemplos de hardware dedicado son los procesadores de red Tilera [42] y las FPGAs, en especial con su popular proyecto NetFPGA [14]. No obstante, trabajar con este tipo de harware supone un alto coste de programación, pues los sistema tienen una complejidad elevada. Por el otro lado, estos sistemas permiten realizar algunas tareas que en software serían mucho más complejas o incluso inviables. Algunos de los trabajos más interesantes en estas plataformas son: Procesamiento DPI a 200 Gbps sobre una Tilera [43], captura y almacenamiento a disco a 10 Gbps [44], emisión de tráfico a 10 Gbps con precisión de ns [44], obtención de estadística de flujos [41] o marcado temporal de paquetes con precisión GPS [45, 46].

No obstante, para la mayoría de aplicaciones de red, no es necesario recurrir al hardware dedicado. Por ello, se han desarrollado multitud de herramientas que intentan exprimir al máximo el hardware disponible. Aunque cada una de las aplicaciones toma una serie de decisiones diferentes, en su conjunto todas intentan reducir el impacto de la pila de red, así como el número de interrupciones del procesador. A continuación se muestran algunas de las herramientas más populares y potentes en captura y procesado de red, así como su funcionamiento y características.

2.1.1. PF_RING DNA

PF_Ring [47, 48] es un motor y un framework de captura para las tarjetas Intel de 1 y 10 Gbps. Este motor, está diseñado para explotar sistemas multicore, de forma que el tráfico de una única Interfaz de red (NIC), se divida de forma equitativa entre diferentes colas de paquetes. Cada una de estas colas, debe ser gestionada por un único core. De esta forma, cada core, es el encargado de solicitar las copias de la memoria de la tarjeta hasta la memoria principal, así como de atender a las interrupciones de la tarjeta de red asociadas a la cola que está atendiendo. De esta forma, el motor permite paralelizar el proceso de captura adquiriendo un enorme speedup. No obstante, hay que tener en cuenta la arquitectura de la máquina subyacente, pues una tarjeta de red está conectada a un único procesador. Dado que nos encontramos en una arquitectura de Acceso a Memoria No-Uniforme (NUMA), instanciar más colas que cores tiene el procesador, supondrá que el otro procesador se verá obligado a procesar también tráfico. Esto causará que el segundo procesador interrumpirá al primero para poder acceder a la tarjeta, pudiendo perjudicar el rendimiento.

Otro detalle de la implementación de *PF_Ring*, se encuentra en las copias de paquetes. Realizar una copia en memoria tiene un coste que se vuelve no despreciable a la hora de hablar de 10 Gbps. Por este motivo, *PF_Ring* sigue la política de zero-copy, de forma que una vez llegan los paquetes a la tarjeta, *PF_Ring* copia el contenido de los paquetes a la memoria principal y mapea dicha memoria al usuario. De esta forma, queda evitada la copia entre la zona de memoria del usuario y el espacio del módulo del kernel. Es importante destacar que *PF_Ring* proporciona una API en formato estándar de facto: *libPCAP*. En la figura 2.1 se muestra el flujo de datos que siguen los paquetes hasta llegar a una aplicación de usuario.

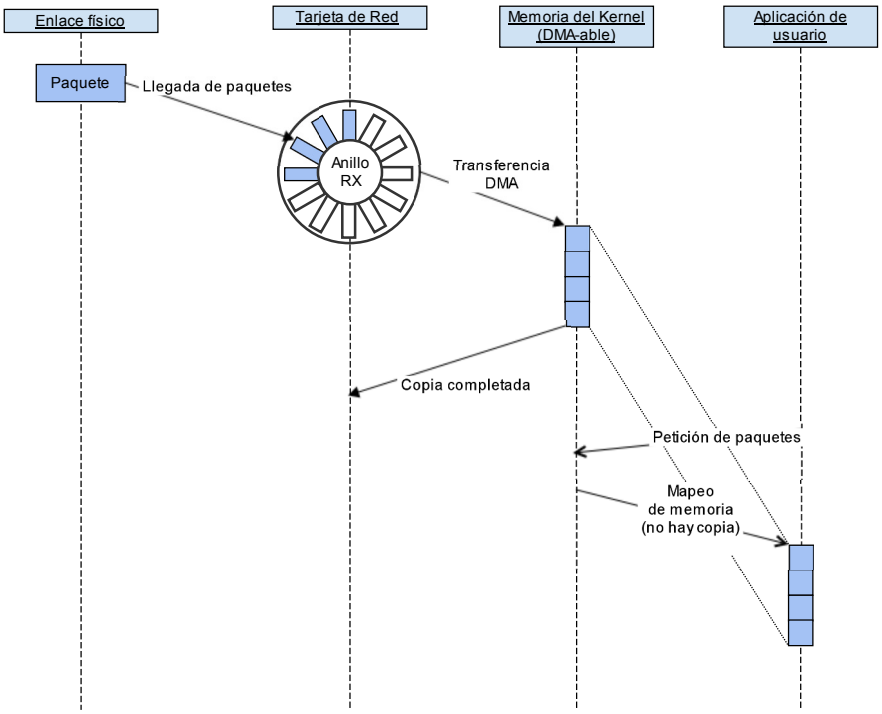

Figura 2.1: Flujo de paquetes en PF_Ring

2.1.2. PacketShader

A pesar de que *PacketShader* [15] fue diseñado inicialmente para ser un router, con la ayuda de una GPU, el núcleo de este sistema puede ser extrapolado a otras aplicaciones. Este motor, cuenta con capacidad de trabajar a tasas de varias decenas de Gbps, ya que al igual que *PF_Ring*, explota los beneficios del paralelismo, así como una reducción en el número de copias. No obstante, cabe destacar en que este sistema utiliza el método de one-copy en vez de zero-copy, lo que puede llegar a degradar el rendimiento del motor de captura.

Algunas de las optimizaciones de *PacketShader* es el control de cache. Toda la memoria se encuentra alineada de forma que las caches de la CPU sufran lo menos posible, optimizándose así el rendimiento. Como desventaja, cabe indicar que el motor no pide datos nuevos a la tarjeta hasta que el usuario no lo solicita, pudiéndose así perder paquetes si el usuario no hace las llamadas en el tiempo adecuado. A diferencia con *PF_Ring*, *PacketShader* no provee una API de uso estándar, por lo que un posible desarrollador usuario se vería obligado a reescribir parte de su aplicación de red. En la figura 2.2 se muestra el flujo de datos que siguen los paquetes hasta llegar a una aplicación de usuario.

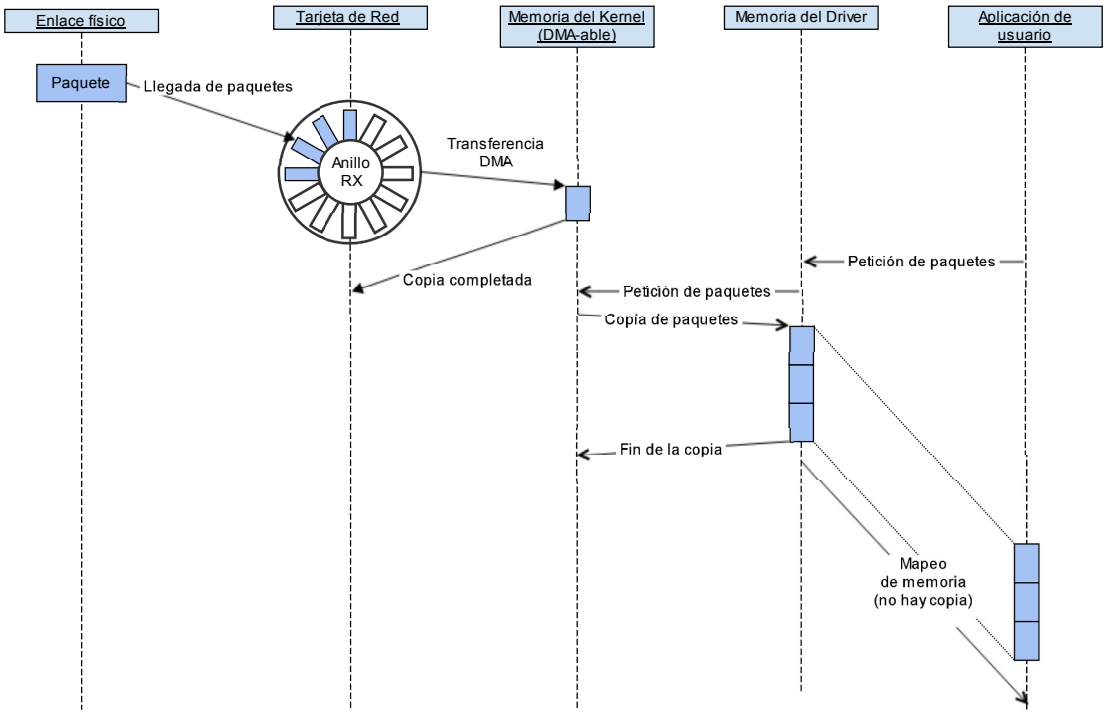

Figura 2.2: Flujo de paquetes en PacketShader

2.1.3. HPCAP

El driver *HPCAP* fue desarrollado en el grupo de investigación HPCN [16]. Este driver, es una extensión mejorada del driver oficial de las tarjetas Intel de 10 Gbps, por lo que solo es compatible con este tipo de tarjetas. Al ser una extensión, este motor de captura conserva en gran medida parte de la funcionalidad original del driver, permitiendo a las interfaces funcionar un modo vanilla o en modo *HPCAP* indistintamente. Al contrario que los otros motores de captura presentados anteriormente, *HPCAP* tiene la capacidad de capturar la mayor parte del tráfico con una única cola de recepción.

Al igual que *PacketShader*, *HPCAP* se basa en la filosofía de one-copy. Aunque a primera vista está claro que la filosofía del zero-copy proporciona una copia menos y mejora el rendimiento, esta técnica supone mantener en memoria los paquetes hasta que el último de los procesos haya terminado con él. Dado que la zona de memoria en la que puede escribir la tarjeta es pequeña, si el tiempo total de proceso de un paquete es elevado, la tarjeta se quedará sin espacio donde copiar los nuevos paquetes viéndose obligada a descartar algunos de ellos. En el caso de la filosofía one-copy, es posible mantener un buffer intermedio de gran tamaño en donde almacenar temporalmente los paquetes recibidos. De esta forma, es posible a su vez crear pipelines de proceso de paquetes mucho mayores y mantener un cierto aislamiento de las aplicaciones de procesado de red y la captura en sí, elemento del que no dispone la implementación de *PacketShader*. Como cierta desventaja, desear utilizar el motor de captura *HPCAP* en modo *HPCAP*, requiere utilizar una API proporcionada por el propio motor obligando a un desarrollador a reescribir su programa de procesado de red. En la figura 2.3 se muestra el flujo de datos que siguen los paquetes hasta llegar a una aplicación de usuario.

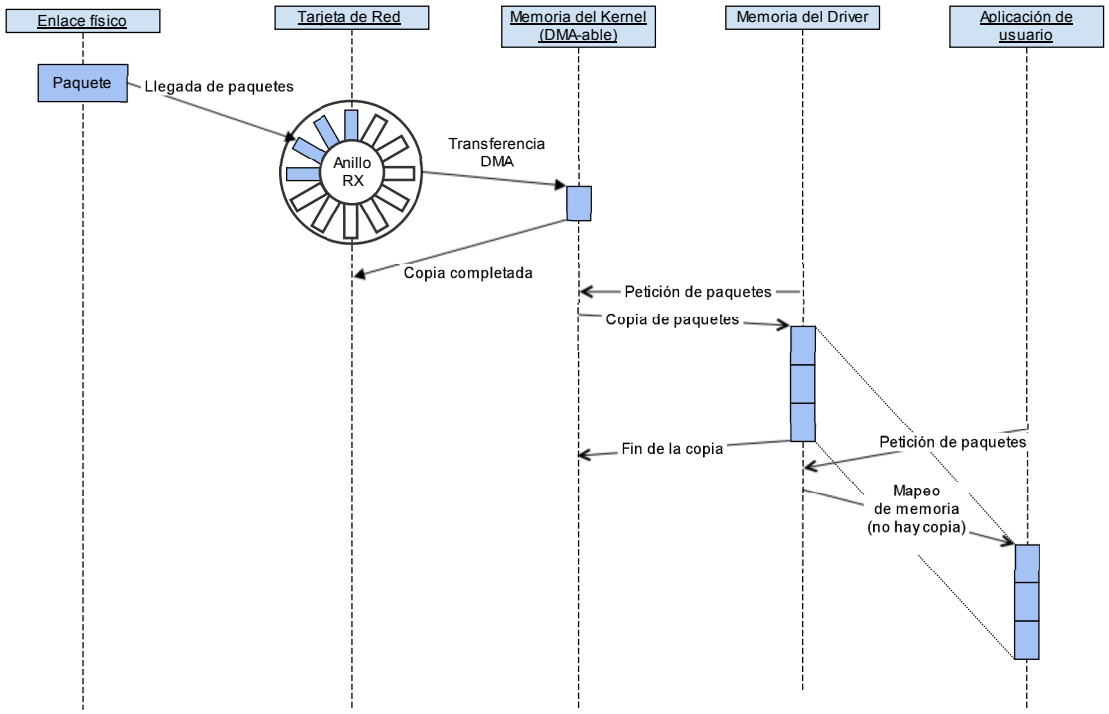

Figura 2.3: Flujo de paquetes en Hpcap

2.1.4. Intel DPDK

Intel DPDK, es una filosofía, así como un motor de captura [19, 49]. Se trata de uno de los sistemas de captura mas novedosos, y por ello, cuenta con gran cantidad de las filosofías que se han ido desarrollando previamente. La filosofía de Intel DPDK, se basa en explotar la tecnología multicore de los sistemas actuales, de forma que uno o varios hilos, sean los encargados de recibir y transmitir los datos por una determinada NIC. En paralelo, otros hilos serán los encargados de procesar dicho tráfico (por ejemplo enrutar, o realizar DPI), y si es oportuno, enviar de nuevo los paquetes a los hilos que manejan las transmisiones. Todo esto, de una manera relativamente transparente mediante la utilización de anillos (Colas circulares) para conectar los diferentes hilos entre sí. Un ejemplo de este tipo de arquitectura puede verse en la figura 2.4.

A pesar de ser un motor de captura (y transmisión) desarrollado por Intel, DPDK da soporte a multitud de tarjetas de red no fabricadas por la propia compañía. Algunas de las tarjetas soportadas es la ellas, son las tarjetas Ethernet a 40 Gbps Mellanox o las tarjetas *enic* de *Cisco*. Del mismo modo que DPDK da soporte a diversas tarjetas, tambien da soporte a elementos virtualizados como a las propias NFVs de sus NICs, o las tarjetas paravirtualizadas con VirtIO. Intel DPDK evita utilizar el espacio del Kernel de linux para maniobrar (evitando así copias intermedias). Por este motivo, su diseño se basa en la construcción de pequeños drivers a nivel de usuario (mediante el módulo *UIO* de *Linux*), lo que permite a un desarrollador utilizar la API de DPDK sin tener que preocuparse por los problemas de trabajar a nivel de Kernel. Esto mismo, llevó a DPDK a optar por la filosofía de zero-copy, la cual, puede llegar a ser perjudicial si el pipeline de proceso se llegase a alargar demasiado.

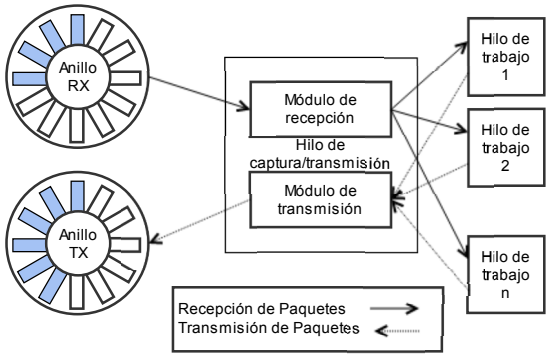

Figura 2.4: Flujo de paquetes en una aplicación Intel DPDK

2.1.5. Otras herramientas y conclusiones

En este trabajo, se ha buscado enfocarse en una fusión entre el mundo de las redes de comunicaciones y la virtualización. Aunque existen un mayor número de sistemas de captura (como NetMap [50–52] o PFQ [53]) que los mencionados anteriormente, pocos son capaces de adaptarse al mundo virtual y soportar NFVs. De las herramientas mencionadas anteriormente, solo Intel DPDK, y *HPCAPvf* (Ver anexo A) son capaces de trabajar con ellas. En el trabajo [54] se exponen y detallan las herramientas de captura comentadas anteriormente. De igual forma, se realiza una comparativa de algunas aplicaciones desarrolladas con dichas herramientas, así como su rendimiento.

Finalmente, cabe destacar a la olvidada compañía Mellanox. Los drivers vanilla de sus tarjetas de 40 Gbps, pueden facilitar a una aplicación cualquiera realizar transmisiones a nivel tcp a más de 20 Gbps, sin necesitar de ningún cambio en la lógica de la aplicación. Aunque este tipo de tarjetas suelen ser olvidadas en las comparativas, pueden generar NFVs de forma natural y trabajar en entornos virtuales, ofreciendo una clara ventaja frente al resto de herramientas. Aunque, en este trabajo no se han tenido en cuenta este tipo de tarjetas, se espera realizar un profundo estudio de las mismas en un futuro.

2.2. Tecnologías de virtualización

Tal y como se describe en [55], existen 3 formas fundamentales de virtualización (Ver fig. 2.5):

1. **Virtualización Nativa**: Al arrancar el ordenador, se inicia por defecto un hypervisor. Este hypervisor, a su vez inicia los diferentes sistemas operativos, quedando una fina capa entre el hardware físico y las diferentes VMs. Las herramientas más conocidas que utilizan este sistema son: *VMware* [56] y *Xen* [57].
2. **Virtualización a nivel de Sistema Operativo (OS)**: Este sistema de virtualización, inicia una versión ligera del sistema operativo que no llega a arrancar un espacio de usuario. Sobre esta capa, se construyen contenedores virtuales (hypervisorses) que iniciarán a su vez las diferentes VMs. Las herramienta más conocida que utiliza este sistema es: *OpenVZ* [58].
3. **Virtualización a nivel de usuario**: En este tipo de virtualización, cada VM es ejecutada

como cualquier otra aplicación a nivel de usuario. Esto conlleva una cierta sobrecarga en la ejecución de las aplicaciones, pero también ofrece un mayor aislamiento. Las herramientas más conocidas que utilizan este sistema son: *Virtual Box* [59] y *KVM* [3].

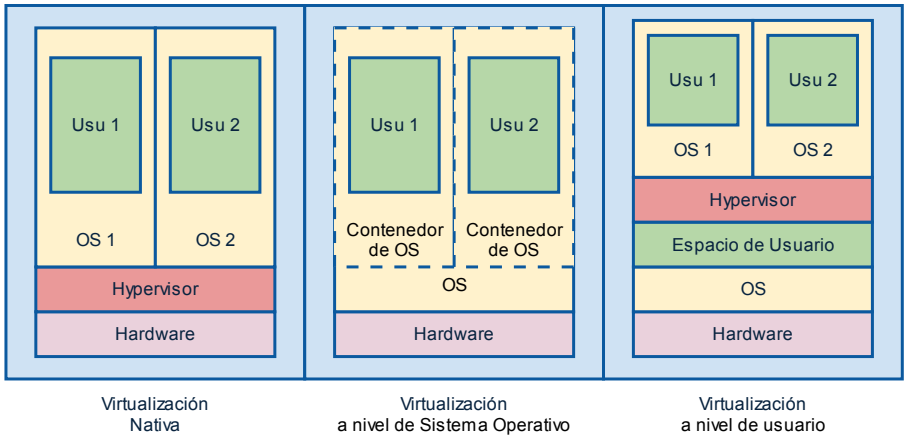

Figura 2.5: Métodos fundamentales de virtualización

Cada uno de estos sistemas, de virtualización, ofrece una serie de posibilidades y configuraciones muy amplio. Abarcar todos y cada uno de ellos en busca de la mejor configuración supone un trabajo arduo y, en principio, fuera del alcance de este trabajo. Por ello, y dado que la mayoría de la investigación se vuelca en el uso de *KVM* como sistema de virtualización, ha sido escogido como hypervisor.

El hypervisor *KVM*, ofrece diferentes métodos a la hora de gestionar los dispositivos virtuales de las VMs. Cada uno de estos dispositivos (como la pantalla, ratón, disco duro o tarjeta de red) puede estar conectado a la máquina virtual de diferentes formas y mediante diferentes niveles de virtualización. *KVM* ofrece los siguientes modos o métodos de virtualización:

- **Virtualización completa**: *KVM* puede crear un dispositivo completamente virtual y simularlo como si fuese un dispositivo real. No obstante, esta virtualización tiene sus límites pues solo es capaz de simular un número limitado de dispositivos. Entre los dispositivos simulables se incluyen tarjetas de red de Intel, dispositivos gráficos o discos duros.
- **Paravirtualización**: *KVM* ofrece un importante y potente conjunto de herramientas de paravirtualización conocidos como VirtIO. Este estándar de creación de dispositivos y drivers, ofrece una serie de facilidades para que los desarrolladores puedan construirse sus propios elementos virtuales. Además, los desarrolladores de la tecnología de VirtIO presumen de ofrecer un rendimiento muy similar al de un dispositivo físico [5].
- **Passthrough**: El método de passthrough, permite des-asociar completamente un dispositivo del sistema operativo de la máquina anfitriona para asociarlo a una determinada VM, cediéndola el control completo sobre dicho dispositivo. Este método, es también usado para transferir VFs a las máquinas virtuales, dado que tanto el sistema operativo, como los diferentes módulos PCI, ven a cada VF como un dispositivo independiente.

3

Diseño y desarrollo de la aplicación

En este trabajo fin de Máster, se ha decidido comparar los sistemas de captura de tráfico *PF_RING*, *HPCAP*, *Intel DPDK* y el driver vanilla de Intel (*ixgbe*). De cara a comparar estos sistemas es necesario disponer de algún tipo de herramienta que haga uso de susodicho sistema para medir de forma precisa los paquetes recibidos y perdidos sin ningún tipo de proceso. Mediante esta métrica, es posible determinar el comportamiento de un motor de captura en un entorno muy favorable, proporcionando una cota de rendimiento o fiabilidad del sistema máxima. Por otro lado, es importante evaluar la capacidad de estos motores para trabajar en conjunto con otros elementos. Dado que resulta interesante almacenar el tráfico capturado para realizar posteriores análisis, se ha decidido evaluar la capacidad de estos motores almacenando a tasa de linea en disco.

No obstante, *Intel DPDK* no ofrece una herramienta simple para medir la tasa de recepción en una NIC, y al igual que *PF_RING*, no existe ninguna aplicación que utilicen estos motores de captura y sean capaces de almacenar a disco a tasa de linea (o al menos hasta donde llega mi conocimiento). Para poder desarrollar y diseñar estos sistemas de captura y almacenado, es de vital importancia conocer cuales serán los escenarios en los que se podrían utilizar.

3.1. Escenarios de captura

Cuando se piensa en un entorno de captura y almacenamiento (y en principio de procesado y análisis) de red, surge de forma natural el concepto de monitorización pasiva. Es extraño imaginar en un contexto real a un único servidor sirviendo peticiones a 10 o incluso a más Gbps. No obstante, parece razonable pensar que varios cientos o incluso miles de nodos se encuentren atendiendo miles o millones de peticiones a tasas mas bajas que, por otro lado, agregadas pueden llegar a superar fácilmente los 10 o incluso 40 Gbps en ciertas compañías.

Bajo esta clásica idea, podemos imaginarnos un sistema (Ver Fig. 3.1) en el que una gran cantidad de máquinas se encuentran conectadas entre sí mediante uno o varios switches o routers.

Dado que estos sistemas suelen acceder de una forma u otra a Internet, no es de extrañarse que en algún punto de la red se encuentre al menos una salida de alta velocidad hacia el resto de Internet. Dado que este enlace supone una vulnerabilidad para la red, suele encontrarse protegido por un cortafuegos. No obstante, una habilidoso atacante podría llegar a franquear el cortafuegos y entrar de una forma u otra en la red. Dado que trabajar con una elevada cantidad de dispositivos en la red, causa un inevitable aumento en la probabilidad de fallo o ataque en alguno de ellos, parece sensato disponer de algún sistema de monitorización que al menos vigile que la red se encuentra en un estado adecuado.

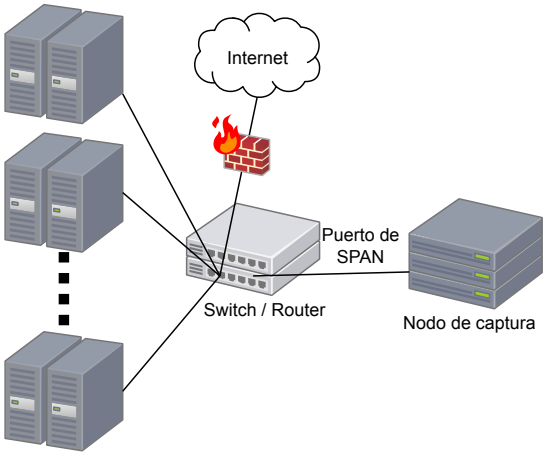

Figura 3.1: Arquitectura de captura tradicional

No obstante, añadir un nuevo equipo a una red, suele suponer la implantación de un equipo grande (4U) y de elevado coste dentro del Centro de procesamiento de datos (CPD) en donde se encuentran los equipos y la red que desean ser monitorizados. También hay que tener en cuenta, que añadir un nuevo equipo supone una serie de riesgos de seguridad para la empresa propietaria de la red y de los equipos y que en ciertos casos, puede no ser asumible.

Dejando estos problemas de lado, es importante tener en cuenta que esta visión de como funciona un CPD y por ende, esta definición de la arquitectura de red es muy diferente a la visión tradicional. Actualmente, y desde la popularización del Cloud Computing, resulta difícil de imaginar un dispositivo dedicado en exclusiva a una única tarea. De forma habitual, una aplicación (como puede ser un servidor web o una base de datos) no explota completamente los recursos de la máquina en la que se ejecuta. Por este motivo, los CPDs actuales(Ver Fig. 3.2) disponen de grandes y potentes máquinas que permiten ejecutar multitud de pequeñas máquinas virtuales dedicadas a tareas muy concretas. Esta división en pequeñas tareas o servicios viene de la mano con el auge de los sistemas distribuidos (como Hadoop [60, 61]), cuyo objetivo es la resolución de un gran problema a base de fragmentarlo en pequeños trozos y procesarlos en paralelo.

La aparición de la virtualización, al igual que aumenta la rentabilidad de los servidores, hace surgir nuevos problemas que en un escenario tradicional no se presentaban. Dentro de un único servidor, están presentes varios sistemas operativos, así como sus respectivas aplicaciones. Comunicar las diferentes máquinas virtuales con el exterior supone una decisión crítica a la hora de definir cada una de las VM. Virtualizar las tarjetas de red con *Passthrough* permite explotar las tarjetas de red al máximo, por otro lado, esto requeriría una tarjeta de red completa por cada una de las máquinas virtuales que fuesen a ejecutarse en la máquina física, además de un aumento en el número de cables y en la capacidad de los switches para interconectarlas.

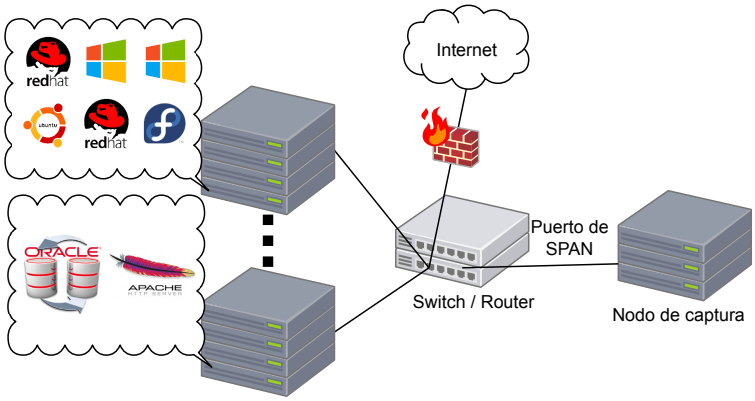

Figura 3.2: Arquitectura de captura en un sistema con virtualización

Dada que una de las bases de las máquinas virtuales es la compartición de recursos, estas máquinas virtuales suelen utilizar virtualizaciones completas de las tarjetas de red, para virtualizaciones de las mismas, o en su defecto, NFVs. Esta compartición de las interfaces de red, provoca que la tarjeta de red actué como un switch virtual entre las diferentes VMs que la comparten. Aunque esto, a priori, no parece suponer un problema, hay que tener en cuenta que el objetivo es monitorizar el tráfico de una determinada red. Todo el tráfico interno entre máquinas virtuales no llega nunca a salir al nodo de captura de la figura 3.2, perdiéndose información potencialmente relevante para la motorización de la red.

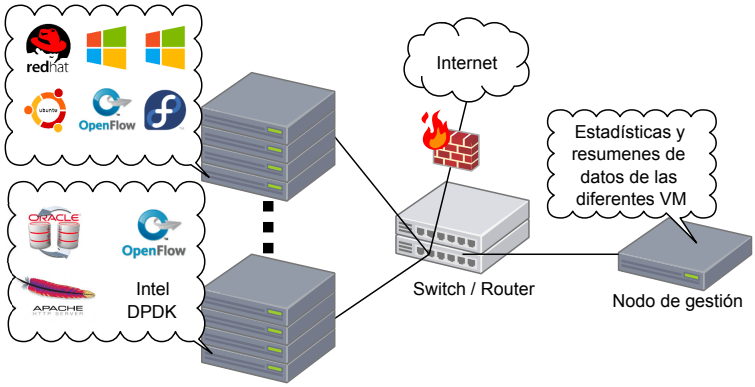

Figura 3.3: Arquitectura de un sistema de captura completamente virtualizado

La única forma de solventar este problema, supone la inserción de un elemento en las máquinas anfitrionas que monitorice dicho tráfico. La forma mas simple de hacerlo, radica en insertar una máquina virtual en cada máquina física, de forma que recoja y procese el tráfico del resto de VMs de su anfitriona. En caso de lograr posicionar una VM en cada nodo físico a monitorizar, la necesidad de disponer de una máquina de captura desaparece completamente. No obstante, al tener un conjunto de VMs monitorizando fragmentos de la red, aparece el concepto de agente (o nodo) de gestión de monitorización. Este nodo, que bien puede ser un elemento físico u otra VM, se encargaría de recoger la información capturada y procesada en las distintas VMs de monitorización y centralizar los resultados bajo un único punto de acceso. La figura 3.3 muestra una representación gráfica del escenario previamente descrito. Aunque está presente el elemento *OpenFlow* en el dibujo, podría ser sustituido por cualquier otro agente de monitorización y captura de tráfico.

3.2. Sistema de captura diseñado

A raíz de los escenarios descritos previamente, es posible definir los requisitos que requiere nuestra aplicación de captura de red. En el peor de los casos, esta aplicación debe ser capaz de capturar y almacenar el tráfico a tasa de linea, es decir: 10 Gbps. Del mismo modo, la aplicación debe ser capaz de trabajar tanto en entornos físicos como en entornos virtuales (Ya sea mediante dispositivos VirtIO, como NFVs). Para lograrlo, dicha herramienta, debe utilizar la API de Intel DPDK que implementa esta compatibilidad con elementos virtuales. DPDK también ofrece una cierta interoperatividad con diferentes tarjetas de red, no limitando la aplicación a un único fabricante.

Para crear una aplicación con Intel DPDK, hay que tener en cuenta su filosofía orientada a la construcción de pipelines y explotación de paralelismo. El funcionamiento de comunicación entre diferentes elementos en una aplicación DPDK se basa en la utilización de anillos. Cada anillo, está diseñado como una cola circular que almacena punteros a descriptores de paquetes. Cada descriptor de paquete, a su vez almacena el tamaño del paquete y un puntero al contenido del mismo. De esta forma, al mover un paquete entre diferentes hilos, no es necesario copiar el contenido del paquete, optimizándose los accesos y escrituras de memoria.

Con el objetivo inicial en mente, de evaluar DPDK en el mejor caso posible (sin procesamiento), se ha desarrollado una aplicación con único hilo. Dicha aplicación captura todos y cada uno de los paquetes y libera los recursos asociados tan pronto es posible. Tras haber recibido un determinado número de paquetes, imprime por pantalla uno valor estimado de ancho de banda, así como cantidad de paquetes recibidos y perdidos (ya sea por errores en los paquetes u otros efectos). Dado que el funcionamiento en detalle de la API es muy conocido [19] y ha sido estudiado previamente en trabajos anteriores [1, 10], considero que no es necesario entrar en detalles implementativos de la aplicación. En la figura 3.4 se muestra una representación gráfica del flujo de datos en la aplicación, mientras que en [62] puede descargarse la aplicación desarrollada.

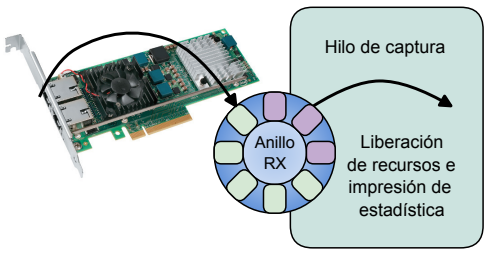

Figura 3.4: Arquitectura de un sistema de captura con DPDK

Una vez se ha desarrollado una versión preliminar capaz de capturar todos los paquetes a alta velocidad, es posible ampliar la aplicación y almacenar a disco. Aunque parezca una tarea sencilla, existe una serie de restricciones que deben tenerse presentes si se desea escribir a alta velocidad. Dichas restricciones vienen impuestas directamente por la tecnología de almacenamiento utilizada, que en este caso son discos duros.

Fundamentalmente pueden encontrarse 2 tipos de discos duros: Discos duros mecánicos y Discos de estado sólido (SSDs). Cada tipo, tiene sus propias ventajas e inconvenientes. La unidad básica de escritura y lectura de un disco duro es el *bloque* (típicamente con tamaños del ordern de KiloBytes o pocos MegaBytes). Si una aplicación desea leer o escribir un único

byte en un disco duro, independientemente del sistema de ficheros que se esté utilizando, el sistema operativo deberá leer y/o escribir el bloque completo. Los discos duros mecánicos se basan utilizan un conjunto de discos magnéticos para almacenar los datos. Dichos discos, deben girar a alta velocidad para ofrecer una velocidad de lectura razonable, así como un conjunto de agujas que se desplacen por el disco leyendo y escribiendo los diferentes bloques. Dichos bloques, se encuentran consecutivos, de forma que leer o escribir una gran cantidad de datos de forma consecutiva, suponga un movimiento muy sutil de las agujas y por tanto un rendimiento muy bueno. Por otro lado, leer o escribir en bloques disjuntos puede obligar a la aguja a moverse llegando a perderse del orden de milisegundos en el proceso.

En cambio, los discos duros SSD, en cambio, utilizan memoria flash para almacenar los datos. Esta memoria, no tiene problemas de localidad, por lo que acceder a bloques muy esparcidos no supone una degradación del rendimiento. Esta tecnología, tambien ofrece unas mayores tasas tanto de lectura y escritura (unos 500 MBps, comparados con unos 150 MBps de los discos duros mecánicos). A cambio de estas ventajas, cada bloque de almacenamiento de un SSD tiene limitados el número de escrituras posibles. De cara a realizar escrituras de forma periódica y dado el coste que supone un disco SSD, parece más sensato utilizar discos duros mecánicos como método de almacenamiento.

No obstante, dado que el objetivo es capturar a 10 Gbps, es necesario utilizar un conjunto de discos. Gracias a las pruebas que se comentan en la sección 4.1.1, se ha determinado que gracias a un RAID 0 formado por 9 discos, es posible escribir a 10 Gbps. No obstante, alcanzar estas velocidades sigue sin ser una tarea del todo trivial. Al crear un RAID 0, el tamaño de bloque del disco RAID se convierte en el tamaño de bloque de la suma de tamaños de bloque de cada uno de los discos que lo forman. Realizar escrituras más pequeñas que este tamaño, degradaría el rendimiento de escritura, por lo que hay que tenerlo presente.

Los sistemas operativos actuales, realizan una división entre la memoria que pertenece al usuario y la memoria que pertenece al núcleo (o Kernel) del sistema operativo. A la hora de realizar multitud de tareas, esta división acarrea problemas de rendimiento y el acceso a disco no es una excepción. Al solicitar una escritura de forma habitual en una aplicación de usuario, la información solicitada es copiada a una región específica del Kernel, y es desde esa memoria desde donde se transferirá al RAID. A la hora de trabajar a altas velocidades, esta copia intermedia impide alcanzar la tasa deseada. No obstante, es posible abrir un fichero en modo “Direct”, es decir, que los datos sean copiados directamente desde la región de memoria del usuario, hasta el fichero destino. Tunear y refinar esta escritura en disco desde una aplicación propia lleva tiempo y en caso de tratarse de un RAID 0, depende del número y propiedades de los discos que lo conforman. Recordemos, que la herramienta debe producir un fichero estándar de los paquetes que ha capturado. El formato estándar para este cometido se denomina PCAP. Un fichero PCAP clásico está compuesto por una pequeña cabecera inicial, y una sucesión de estructuras paquete. Cada una de estas estructuras está formado a su vez por una cabecera en la que figura el tamaño del paquete, el tamaño capturado del paquete, el momento en el que se capturó el paquete y finalmente, el contenido capturado del propio paquete. Debido a que la mayoría de herramientas que procesan ficheros en este formato trabajan adecuadamente con ficheros grandes, la herramienta de captura debe trocear y generar una sucesión de ficheros de tamaño más o menos constante. De forma ideal, este tamaño debe ser de unos 2 GBs como máximo.

Con todas estas ideas en mente, y partiendo del programa inicial presentado en la figura 3.4, es posible describir la arquitectura final de la herramienta, la cual está formada por 3 hilos:

- **Hilo de captura**: Al igual que en la primera versión, un primer hilo se encarga de recoger los paquetes desde el anillo de recepción de la NIC de captura, sin realizar ningún cálculo con ellos, los inserta a alta velocidad en un nuevo anillo software.
- **Hilo de construcción PCAP**: El segundo hilo es el encargado de realizar la mayor parte del procesamiento de paquetes. El trabajo de este hilo, consiste en la construcción de ficheros PCAP en memoria. Para lograrlo, cuenta con un array de buffers de 1 GB (por defecto de tamaño 4), en donde se almacenará en formato binario el fichero PCAP completo. Dado que el tamaño de un fichero puede ser superior al de un buffer, este hilo debe indicar que buffers representan un nuevo fichero y cuales representan la continuación del buffer predecesor. Por cada buffer de comienzo de fichero, el hilo escribe una cabecera de formato PCAP. A continuación comienza a extraer bloques de paquetes del anillo software mencionado anteriormente. Cada uno de estos paquetes, es marcado temporalmente mediante una librería de tiempo de alto rendimiento (HPTL) [2], de forma que pueda construirse la cabecera del paquete. Tanto la cabecera, como el contenido del paquete son copiados al buffer. Cuando el último paquete del último buffer que forma un fichero debe ser escrito, puede dejar algunos bytes libres en los que no quepa dl posible siguiente paquete. En estos casos, el hilo anota en el buffer, que el fichero debe ser truncado tantos bytes como hayan quedado en desuso. Una vez se ha completado el buffer, el hilo lo libera y comienza a trabajar con el siguiente buffer disponible.
- **Hilo de volcado a disco**: El tercer hilo, es el encargado de volcar el contenido de los buffers intermedios a disco. Dado que el proceso de escritura es bloqueante, es necesario disponer de un hilo encargado únicamente a este proceso. Gracias al funcionamiento de DPDK, los buffers en los que escribe el segundo hilo se encuentran contiguos y alineados en memoria, de forma que el Kernel es capaz de escribirlo en el fichero destino a muy bajo coste. Este hilo, a su vez, es el encargado de la creación y truncado de ficheros, mediante las pautas indicadas por el hilo número 2. Tras terminar el volcado de un buffer, el hilo lo libera para que pueda ser reutilizado cuanto antes.

Gráficamente, es posible ver la arquitectura y el flujo de los paquetes en la figura 3.5.

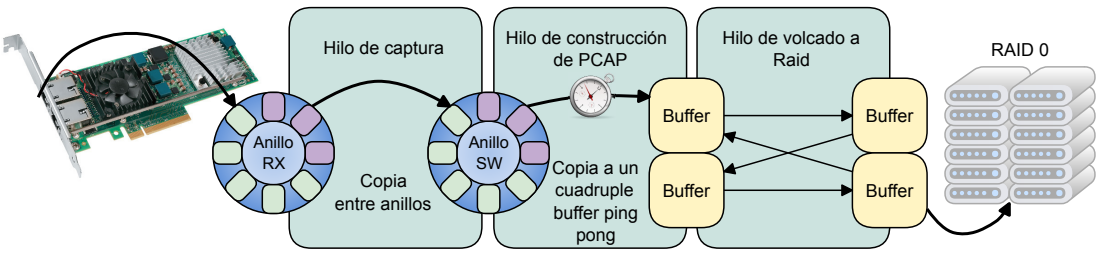

Figura 3.5: Arquitectura de un sistema de captura y almacenamiento con DPDK

4

Pruebas

El objetivo de este capítulo es mostrar tanto las pruebas realizadas como la metodología seguida para obtener los resultados. Para poder entrar en detalle, en las pruebas, es necesario hacer un repaso de los dispositivos hardware y software que se encuentran disponibles. Una vez conocidos, es posible definir una metodología de pruebas, así como una descripción de las mismas en los diferentes entornos.

4.1. Entorno de Desarrollo

Escoger un entorno de desarrollo y pruebas adecuado, supone realizar multitud de comparativas y pruebas entre los diferentes elementos disponibles, tanto a nivel software, como a nivel hardware. Poner a prueba todos y cada uno de los posibles elementos supone un reto en cuanto a coste monetario como en tiempo, y dado que tanto el presupuesto como el tiempo de realización del doble trabajo fin de máster son finitos, se ha partido del siguiente equipamiento facilitado por el grupo de investigación HPCN.

4.1.1. Equipamiento de prueba utilizado

El equipamiento de prueba utilizado costa de 3 elementos fundamentales: Captura de tráfico, emisión de tráfico y el tráfico emitido como tal. Todos estos elementos son de vital importancia a la hora de realizar una comparativa, pues tan importante es el proceso de captura, como que los datos recibidos hayan sido emitidos correctamente. Estos datos, por otro lado, deben representar algún tráfico significativo para las pruebas, ya sea un caso extremo, o un caso realista. Cada una de estas partes es explicada con detalle a continuación:

Equipo de captura y desarrollo

El equipo de captura y desarrollo es conocido internamente por el nombre de *Nrg*. Esta sonda de captura, está compuesta por una arquitectura de tipo NUMA con dos procesadores *Intel Xeon E5-2630* a una velocidad de 2.6 Ghz cada uno. Ambos procesadores se encuentran conectados a 2 tarjetas de memoria de 8 GB cada una, haciendo un total de 32 GB para todo el sistema. Este equipo cuenta con diversos PCIe generación 3 que permiten transferencias de datos de hasta casi un 1 Gbps por cada linea PCIe. El equipo cuenta con los siguientes dispositivos PCI:

- **Tarjeta gráfica Nvidia Tesla K40C**: A pesar de no haber sido utilizada la tarjeta gráfica a lo largo de las pruebas o el desarrollo, parece interesante mencionar su presencia en el bus PCIe.
- **Tarjeta de red Ethernet Mellanox MT27500 (ConnectX-3)**: Esta tarjeta es capaz de establecer velocidades de enlace de entre 40 y 56 Gbps. Aunque inicialmente se pretendía probar esta tarjeta, Intel DPDK no publicó los drivers para explotar esta tarjeta hasta la versión 2.0, la cual fue publicada en abril de 2015. De igual modo, no disponemos de ningún emisor de tráfico fiable capaz de saturar este enlace, ni tampoco la capacidad de realizar el almacenamiento a disco a esta tasa de red sin realizar algún tipo de filtrado de tráfico. Hasta donde llega mi conocimiento, no existe (aparte del nuevo driver de Intel DPDK y el driver nativo) ninguna aplicación similar contra la que comparar resultados. Por todo esto, el uso de esta tarjeta ha sido descartado al encontrarse fuera del marco de este trabajo. Esta tarjeta conecta la máquina de captura con la máquina llamada *Onelab3*.
- **Tarjeta de red Ethernet Intel I350**: Esta tarjeta dispone de dos NICs a 1 Gbps cada una. Dado la máquina de captura se encuentra inaccesible físicamente, se ha utilizado esta tarjeta como medio para el acceso remoto, así como el acceso a diferentes recursos y paquetes de Internet.
- **Tarjeta de red Ethernet Intel 82599ES**: Esta tarjeta dispone de dos NICs SFP+ a 10 Gbps cada una. El chipset *82599ES* es compatible con la mayoría de capturadores de bajo coste actuales, por lo que la convierte en el dispositivo predilecto de cara a realizar una comparativa de rendimiento. Además, soporta diferentes modos de virtualización (Passthrough y SR-IOV), permitiendo de esta forma alcanzar los objetivos planteados en este trabajo. Las dos interfaces de la tarjeta se encuentran conectadas al emisor de tráfico (llamado *Dagda*), el cual será explicado más adelante.
- **Controladora MegaRAID SAS-3 3108**: Dadas las limitaciones del chasis de la sonda de captura, la controladora RAID es capaz de gestionar hasta un máximo de 12 discos duros. Estos discos duros, pueden conformar desde un único Raid 0, hasta un conjunto de diferentes tipos de Raid.

Una vez observados los diferentes dispositivos disponibles, parece interesante realizar un inciso en la controladora raid. Si bien puede gestionar hasta 12 discos duros, el precio de estos no es en absoluto despreciable. En este caso, se parte de discos mecánicos *Hitachi HUA 72303* (especiales para servidores) con 3 TB de capacidad cada uno. El precio actual aproximado de estos discos duros oscila entorno a los *310€*, elevando el coste de almacenamiento a más de *3700€*. Por este motivo, determinar cual es la cantidad de discos duros necesarios para cada escenario es de vital importancia de cara a reducir el coste de la sonda de captura.

Equipo de emisión de tráfico

Existen multitud de herramientas de emisión de tráfico. Una de las herramientas clásicas es TCPReplay. Dicha herramienta parte de un fichero estándar *PCAP* y lo transmite por una determinada interfaz de red. TCPReplay, proporciona cierto valor añadido, ya que puede regular la velocidad de transmisión entre otras muchas cosas. No obstante, esta herramienta al igual que la mayoría de emisores de tráfico clásicos (Como pktgen, etc), no es capaz de emitir tráfico a 10 Gbps, aun si este se encuentra en memoria. Esto se debe a que las herramientas de transmisión y generación de tráfico utilizan la pila de red completa del Kernel y los drivers vanilla de las tarjetas de red.

De cara a solventar el problema de la generación y transmisión de tráfico, se han construido multitud de herramientas. Los generadores de tráfico software se basan en la reutilización de los drivers optimizados de captura como DPDK o *PacketShader*. No obstante, los emisores de tráfico software presentan serias limitaciones. En el caso del emisor de *PacketShader* [15], se presentan ciertas irregularidades en la tasa de tráfico: emisión a ráfagas, problemas en los contenidos de los paquetes y en general problemas de estabilidad. De cara a hacer una comparativa de rendimiento y tasa de captura, estas irregularidades complicarían y harían fluctuar las medidas, por lo que este emisor fue descartado. Por otro lado, Intel DPDK proporciona una herramienta de emisión de tráfico bastante sofísticada conocida como *Pktgen-DPDK* [63], la cual, poco a poco, se está convirtiendo en una conocida aplicación. La herramienta *Pktgen* es capaz de saturar fácilmente 4 enlaces a 10 Gbps mediante tráfico sintético. Para conseguirlo, reserva las estructuras de los paquetes en memoria y las envía a cada una de las NIC. Esta herramienta, aunque útil para probar casos extremos, se ve perjudicada si lo que se desea es enviar tráfico almacenado previamente en un fichero. Debido al sobrecoste de las estructuras de los paquetes[1]que utiliza DPDK, no es posible almacenar más que unos pocos centenares de miles de paquetes por GigaByte. Aunque este número pueda aparentar ser muy grande, recordemos que en una red de 10 Gbps, se pueden llegar a mandar hasta casi 15 millones de paquetes por segundo (en caso de paquete mínimo), por lo que esta cantidad de paquetes representaría menos de una décima parte de segundo del tráfico del enlace.

En el otro lado, se encuentran los generadores hardware basados en FPGAs. Dentro del grupo de investigación HPCN ha sido desarrollado un potente y versátil generador de tráfico a 10 Gbps [44]. Este generador, aporta un nivel de control en la transmisión de las tramas muy preciso, permitiendo simular, tanto cualquier situación de una red real, como casos extremos incluso a nivel físico. Para lograr estas características, este sistema utiliza un formato especial de fichero denominado *simple*. Este fichero es similar al formato PCAP, salvo porque almacena la cantidad de ceros que existen a nivel físico entre dos paquetes, lo que permite tener un elevado control y precisión durante la emisión. Dichos ficheros se encuentran en un Raid 0, de forma que un driver intermedio es capaz de acceder de forma eficiente a ellos y copiarlos en HugePages. Estas páginas son transferidas a la FPGA a través del bus PCIe. La FPGA emite el tráfico por una o varias de las diferentes NIC que posee. Como contrapartida, aunque la FPGA dispone de hasta 4 interfaces de 10 Gbps, solo es capaz de recibir hasta 10 Gbps por PCIe, por lo que como mucho podrá enviar el mismo tráfico simultáneamente por las diferentes NICs. En la figura 4.1 se muestra un esquema del funcionamiento del generador y emisor de tráfico hardware.

Tras escoger el generador de tráfico hardware, se definió un interconexionado de las diferentes

[1]Cada estructura de DPDK almacena diversa información referente a un paquete. De cara a mantener la coalescencia de las cachés, estas estructuras cuentan con 2048 Bytes (La potencia de 2 más próxima a la Unidad Máxima de Transmisión (MTU) de la red) para almacenar el payload del paquete.

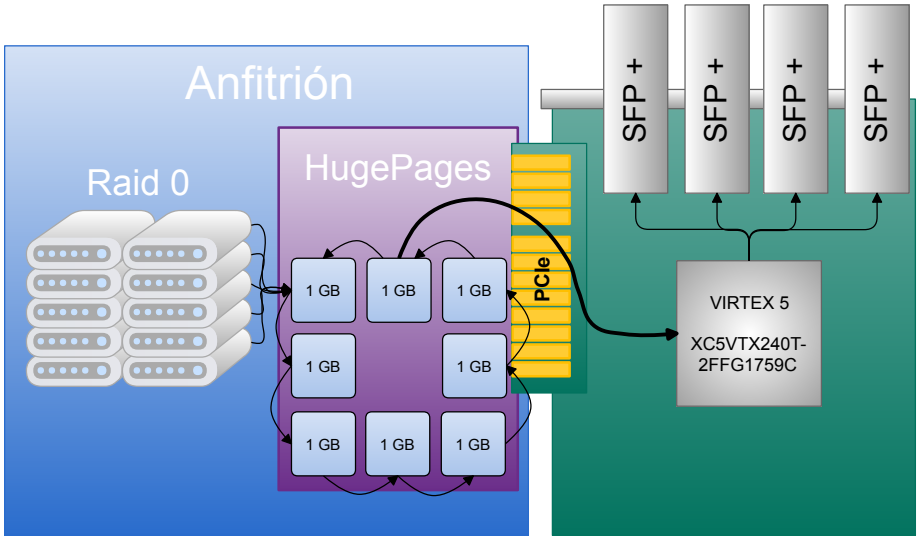

Figura 4.1: Arquitectura del emisor de tráfico

máquinas que formarán parte de las diversas pruebas. La máquina encargada de la transmisión de tráfico (llamada *Dagda*) se conecta mediante 2 interfaces de 10 Gbps a la máquina capturadora de tráfico (llamada *Nrg*). La máquina de captura se encuentra a su vez conectada por un enlace de 40 Gbps con una máquina llamda *Onelab3*. No obstante, y como se ha mencionado anteriormente, este enlace queda en desuso debido a la falta de software necesario para explotarlo adecuadamente. En la figura 4.2 se muestra gráficamente este interconexionado.

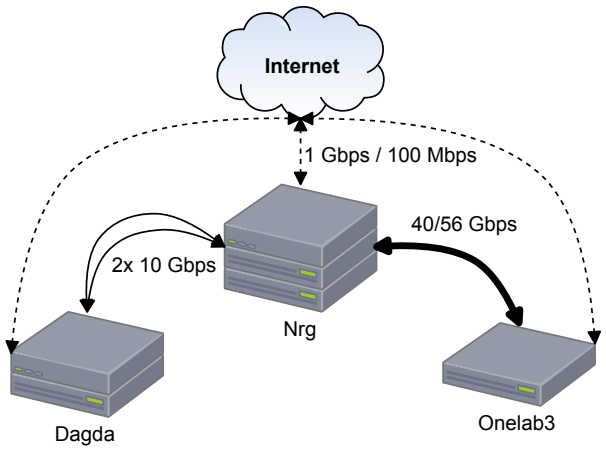

Figura 4.2: Conexionado del equipo de captura y desarrollo

Tráfico emitido

Una vez se ha decidido tanto el funcionamiento de la sonda de captura, como el hardware que se utilizará, es necesario definir el tráfico con el que se realizarán las evaluaciones, pruebas y comparativas. Para poder tener medidas útiles, es necesario poner al equipo de captura al límite, forzando y poniendo el sistema en el peor caso posible. No obstante, como se comentó en la introducción, resulta complicado encontrar enlaces completamente saturados, por lo que resulta interesante contar con tráfico representativo real.

Con el objetivo de cubrir ambos escenarios, se han utilizado los siguientes tipos de tráfico:

1. **Tráfico extremo**: Dentro del protocolo Ethernet, existen dos casos peores posibles: Un enlace en la que solo hay paquetes de 64 bytes y supone una tasa de 14.88 Millones de paquetes por segundo, o un enlace, en la que solo hay paquetes de tamaño 1500 y unos 800 mil paquetes por segundo. En el primer caso, tan solo un 76 % de los bytes transmitidos[2] en el enlace son almacenados, mientras que en el segundo caso se almacenan entorno al 98 % de los bytes. Mientras que el primer caso requiere una mayor necesidad de computo para procesar un gigantesco número de paquetes, el segundo caso requiere de una mayor capacidad de escritura a disco. Las herramientas proporcionadas por el generador de tráfico hardware escogido, son capaces de generar ambos escenarios extremos sin la necesidad de construir una traza a medida para la prueba.

2. **Tráfico Real**: Para poder obtener a tráfico real representativo a alta velocidad, es necesario recurrir a redes de grandes empresas o grandes nodos de interconexión de algún ISP. Esto causa, que la naturaleza de este tipo de tráfico suela ser confidencial o deba mantener ciertos requisitos de privacidad, por lo que, es complejo acceder a este tipo de muestras de tráfico. Dentro de este contexto, la organización Caida, captura tráfico en nodos que interconectan grandes ciudades de Estados Unidos. Tras realizar un proceso de anonimización[3], la organización Caida publica estas capturas de tráfico para su posterior uso por investigadores. Dentro de las trazas disponibles, se ha escogido una muestra unos 7 minutos de duración entre la ciudad de Seattle y la ciudad de Chicago el día 1 de octubre de 2014 [64]. Aunque esta traza está capturada en un enlace a 10 Gbps, la velocidad efectiva del tráfico ronda los 5 Gbps con un tamaño medio de paquete de unos 965 Bytes. De cara a realizar las pruebas y estresar un poco el sistema, se ha decidido acelerar transmisión de esta traza, reduciendo el tiempo entre los paquetes al mínimo posible.

4.1.2. Software utilizado

Encontrar el software apropiado para realizar un desarrollo es una tarea tediosa. Dado que el objetivo de este trabajo fin de máster es la realización de un motor de captura y almacenamiento de tráfico con Intel DPDK y la realización de una comparativa entre los diferentes métodos de captura en diferentes entornos de ejecución, es necesario plantear un entorno software aceptable para un posible cliente. En el mundo empresarial, predomina el uso de las distribuciones de linux basadas en red hat o en suse. No obstante, la tecnología de virtualización, requiere de los últimos avances para poder ser explotada al máximo. Con esto en mente, la distribución de linux que mejor cumple estas condiciones es Fedora. Por este motivo, en la sonde captura se ha instalado un Fedora 20.

La mayoría de los entornos de virtualización de los que disponen las grandes compañías son: KVM [3], XEN [57] o la versión profesional de VMWare [56]. De cara a un presupuesto limitado, descartamos la opción de utilizar VMWare desde el principio. La decisión entre los hypervisores KVM y XEN es algo más compleja pues son ambos sistemas muy utilizados, ambos son gratuitos y ambos se encuentran en auge. No obstante, se aprecia a la comunidad investigadora más enfocada

[2]Los paquetes Ethernet tienen diversos campos, como el interframe gap o el preludio, que deben ser transmitidos pero no aportan ninguna información relevante. Por este motivo, dichos campos nunca son transmitidos por la tarjeta hacia el ordenador anfitrión y no pueden ser almacenados. En caso de que los paquetes sean muy pequeños, estos bytes "ocultos" se hacen relevantes.

[3]El proceso de anonimización consiste en truncar los paquetes eliminando la capa de aplicación.

en el entorno de KVM, así como una gran cantidad de esfuerzo por parte de la comunidad de KVM en el desarrollo de elementos y técnicas avanzadas de virtualización como VirtIO. Por estos motivos, se ha escogido KVM como método de virtualización. Como optimización, dado que varios de los motores de captura hacen uso de las HugePages, se ha configurado el sistema de virtualización KVM para que reserve la memoria de las máquinas virtuales en HugePage con el objetivo de que el rendimiento en las VMs no se vea excesivamente afectado por problemas de paginación o problemas de cache [65]. Las máquinas virtuales, a su vez, proveen HugePage a los diferentes motores que se ejecutan en ellas. Cada una de las máquinas virtuales utilizadas está formada por 3 cores virtuales, mapeados y reservados en 3 cores físicos. Cada máquina virtual cuenta a su vez con 5 GB de memoria RAM.

Una vez que tenemos los más pilares básicos de nuestro sistema de captura, es necesario hacer un pequeño énfasis en lo que a funciones virtuales se refiere. Tal y como se ha explicado en capítulos anteriores, las funciones virtuales son a todos los efectos un dispositivo PCIe más. No obstante, las tarjetas Intel ofrecen diversas formas de crear y gestionar sus NFVs. Por un lado, el driver vanilla *ixgbe*, permite indicar a la tarjeta que cree un número determinado de NFVs, de una forma muy sencilla. No obstante, este tipo de funcionamiento delega todo el control de la función virtual a la tarjeta física, sin que la CPU intervenga en ningún caso en el trasiego de paquetes. Por otro lado, el driver proporcionado por DPDK, proporciona su propia forma de gestionar las funciones virtuales. Este paradigma, rompe ligeramente el concepto de VF, ya que DPDK requiere del uso de un determinado programa, llamado *testpmd*, que de forma activa ayuda a la tarjeta a manejar las diferentes funciones virtuales. No obstante, este método obliga a la CPU a trabajar, consumiendo, como mínimo, un core completo. De cara a evaluar que método es el mejor, se han tenido en cuenta ambas aproximaciones y se detallan los resultados de la comparativa en la sección 4.4.3.

Dentro del software de captura de tráfico se ha decidido realizar una comparativa entre los siguientes motores: DPDK, *HPCAP*, *PF_RING*, y el driver vanilla *ixgbe* mediante el programa de captura simple basado en *libPCAP* [66].El resto de motores de captura han sido descartados para las pruebas, dado que su funcionamiento es muy similar entre sí, y ninguno de ellos es capaz de operar con NFV. Los motores de captura han sido explicados previamente en el capítulo 2.

4.2. Metodología de las pruebas

Realizar todas y cada una de las pruebas que se describen en las siguientes secciones, supone un trabajo tedioso y en un principio muy manual. Si bien son importantes los resultados de las pruebas, es de igual importancia mantener un cierto protocolo a la hora de realizarlas asegurando su repetibilidad así como almacenar los resultados de forma organizada. Para llevar esto a cabo, se realizó un documento interno que incluían los pasos a la hora de lanzar una prueba en los diferentes escenarios, así como los parámetros recomendados para cada una de las diferentes aplicaciones.

Cada una de las aplicaciones de captura tiene su propia forma de representar las diferentes estadísticas relacionadas con la NIC que está utilizando para capturar (como paquetes recibidos, paquetes perdidos, paquetes con errores, etc). De cara a normalizar estos datos, se realizaron diferentes scripts encargados de interpretar la salida de cada una de las aplicaciones y convertirlas a un formato único. El formato planteado se basa en un simple fichero de texto de 4 columnas separadas por tabulaciones. Cada línea de este fichero, representa un determinado instante de

tiempo en el que se consultaron las estadísticas de una única NIC. En caso de que la aplicación esté gestionando más de una interfaz de red, se generan tantos ficheros como NICs utilice.

El fichero de estadísticas de una NIC consta de las siguientes columnas:

1. Momento en formato unix en el que se realizó la medida.
2. Velocidad en Gigabits por segundo entre el momento anterior y el actual.
3. Paquetes recibidos entre el momento anterior y el actual.
4. Paquetes perdidos entre el momento anterior y el actual.

Para poder mantener una organización entre las diferentes pruebas, estos ficheros son almacenados en un árbol de carpetas. Este árbol se subdivide en nombre de la prueba, tráfico utilizado, motor de captura utilizado y marca temporal del inicio de la prueba. Dado que estos ficheros representan una serie temporal, que aporta cierta información de depuración, se ha realizado un nuevo script que resume los diferentes ficheros asociados a cada prueba, generándose así un pequeño informe de resultados de cada una de las pruebas realizadas.

La complejidad de los capturadores de tráfico es elevada. Por ello, estas aplicaciones cuentan con multitud de parámetros que los permiten acomodarse a diversos entornos. Estos parámetros engloban desde el core en el que se ejecuta un determinado componente del capturador, hasta el tamaño de los descriptores de paquete o la longitud de las colas de recepción. Para que la comparativa entre los diferentes motores de captura sea justa, es necesario repetir multitud de veces determinadas pruebas en busca de los parámetros adecuados. Del mismo modo, es necesario que todas las pruebas se ejecuten en las mismas condiciones, evitando fluctuaciones debido a otros efectos colaterales de haber ejecutado anteriormente una prueba con otro capturador (como elementos cacheados o cambios colaterales en el funcionamiento de algún dispositivo). Por este motivo, se desarrollaron dos protocolos de realización de pruebas, una para entornos físicos y virtuales mediante Passthrough y otro para entornos virtuales con NFVs:

- **Procedimiento en entorno físico y virtual con Passthrough:**
  1. Se reinicia la máquina (física o virtual) con la configuración adecuada (HugePages si son necesarias, etc).
  2. Se resetea el generador de tráfico.
  3. Se instancia el driver y programa de captura.
  4. Se inicia la transmisión de tráfico por parte del generador.
  5. Al terminar la ejecución del programa de transmisión, se cierra el programa de captura y se almacenan los resultados.
- **Procedimiento en entorno virtual con NFV:**
  1. Se instancia el driver encargado de la generación de NFV y se configura.
  2. Se reinicia la máquina virtual con la configuración adecuada a la prueba.
  3. (Si procede) se inicia el programa *testpmd* de Intel DPDK en el anfitrión.
  4. Se resetea el generador de tráfico.
  5. Se instancia el driver en la máquina virtual y programa de captura.

6. (Si procede) se configura el programa *testpmd*.
7. Se inicia la transmisión de tráfico por parte del generador.
8. Al terminar la ejecución del programa de transmisión, se cierra el programa de captura y se almacenan las estadísticas tanto de la máquina virtual como de la máquina física.

4.3. Pruebas en entorno físico

El régimen clásico de funcionamiento de las múltiples herramientas de red es el entorno nativo o físico. Dado que estas herramientas usualmente requieren de una gran cantidad de procesamiento así como de ancho de banda en la memoria para funcionar a altas (e incluso no tan altas) velocidades, hasta hace poco era impensable llevar a estas herramientas a un entorno virtual, salvo para realizar tareas con muy poca cantidad de datos. Por este motivo, el entorno físico es el punto de partida en la realización de cualquier comparativa de herramientas de red, lo que incluye a los capturadores de red.

La arquitectura del entorno físico es simple. Una tarjeta de red y una controladora Raid, mediante sus correspondientes drivers, se conectan a un motor de captura. De esta forma, el trabajo del motor de captura se resume en una copia de los datos que llegan desde la red hasta el Raid de alta velocidad. En un principio, esta tarjeta de red, se encontraría conectada a un switch o a un router de la red que deseamos monitorizar, el cual, nos duplica todo el tráfico que circula por la misma mediante un puerto de SPAN. Dado que en este caso, no se ha podido acceder a una red de alta velocidad, se ha simulado este enlace mediante el generador hardware comentado previamente.

Si el servidor de captura es suficientemente potente, es posible que en paralelo se encuentren en ejecución otras aplicaciones en nuestra sonda de captura. Estas aplicaciones pueden encargarse de leer los datos almacenados en el Raid y sacar algún tipo de análisis, o incluso, ejecutar algún tipo de servicio que no esté directamente relacionado con la captura como un servidor web o una máquina virtual completa. Esta arquitectura puede verse en la figura 4.3.

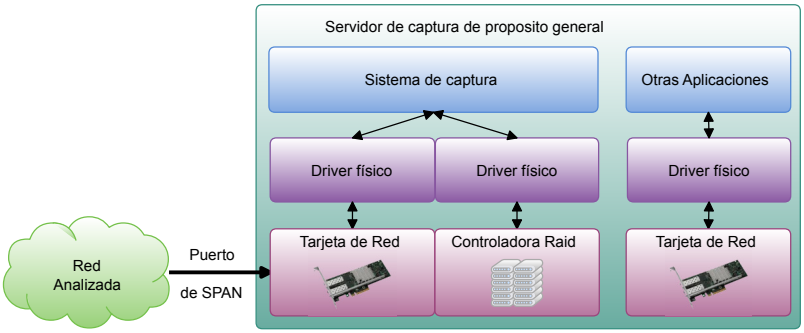

Figura 4.3: Arquitectura de captura clásica

Un sistema de captura y almacenado, como su nombre indica, se divide en dos partes: captura y almacenado. Ambos sistemas utilizan y explotan al máximo el ancho de banda de la memoria y el ancho de banda de los diferentes buses de comunicaciones. Por ello, es necesario realizar una comparativa que mida el caso mejor de ambos procesos por separado.

De cara a hacer una prueba de rendimiento del sistema de almacenado (Raid 0), se ha reutilizado un script que permite, dinámicamente, cambiar el número de discos que conforman

el Raid de captura. Ya que los motores de captura de los que se dispone, almacenan sucesivos ficheros de 2 GB, este script se encarga también de realizar sucesivas escrituras de ficheros de 2 GB mediante la herramienta de *GNU*, *dd*. Tal y como se ha comentado anteriormente, para poder escribir a esta velocidad en un raid es necesario saltarse las copias intermedias que realiza de forma natural el kernel de *Linux*. Para lograrlo, existe el modo de escritura *Direct*, que permite al kernel copiar directamente desde la memoria del usuario a nuestro raid de captura.

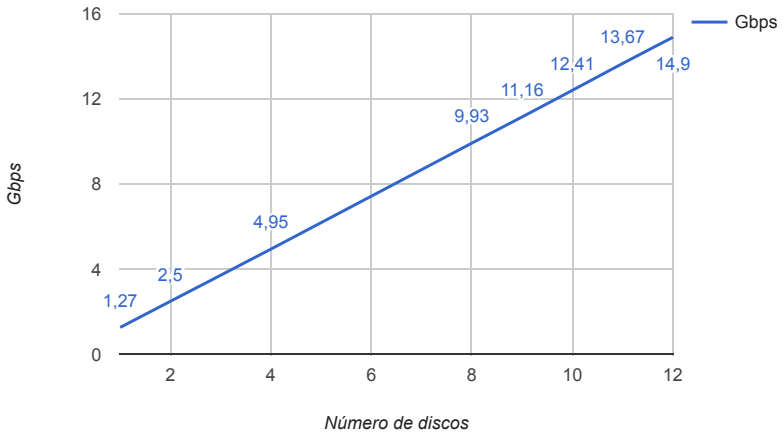

Figura 4.4: Rendimiento de escritura en Raid 0 sin virtualización

Tal y como puede observarse en la figura 4.4, la tasa de escritura a disco parece crecer de forma lineal con el número de discos que conforman el raid. A raíz de estos resultados, parece viable crear un raid de tan solo 9 discos para las pruebas, pues en media ofrece una tasa de escritura de más de los 10 Gbps a los que se recibe el tráfico. No obstante, y para mayor seguridad, se han realizado varios cientos de prueba de escritura para poder obtener un intervalo de confianza del rendimiento que proporciona el Raid con 9 discos. Dicho intervalo se encuentra entre 11,09 y 11,23 Gbps, lo que nos asegura en cierta medida que el rendimiento será suficiente para almacenar todo el tráfico.

Una vez se ha medido el rendimiento del raid de almacenamiento el caso óptimo, es posible comenzar a realizar las primeras pruebas de captura con los diferentes motores. Dado que en esta primera prueba, se busca encontrar las limitaciones de los capturadores de tráfico, se han utilizado unas pequeñas aplicaciones que tan solo reciben el tráfico y no realizan ningún procesado sobre los mismos. Mientras que los motores *HPCAP* y *PF_RING* proporcionan sus propias herramientas básicas de testeo, se ha realizado una herramienta propia en DPDK para realizar este tipo de mediciones. De igual modo, se ha reutilizado una pequeña aplicación realizada en *libpcap* para probar la tasa de captura del driver vanilla *ixgbe*.

Los resultados de esta primera prueba, pueden observarse en la tabla 4.1. Estos resultados, resultan en cierta medida sorprendentes pues el driver vanilla de Intel ha mejorado en gran medida con los años, pudiendo incluso capturar 10 Gbps sin perdidas cuando se enfrenta a un tráfico realista. También cabe destacar, el rendimiento del motor de captura *PF_RING*. A pesar de obtener un buen rendimiento al enfrentarse a una única NIC, el rendimiento decrece y comienza perder paquetes cuando debe enfrentarse a 2 interfaces de red simultáneamente. El rendimiento de *HPCAP*, parece bastante razonable si recordamos su filosfía de one-copy. Finalmente, en esta

tabla se puede observar como la tecnología de captura más novedosa sale ganando en esta prueba sin llegar a perder ni un solo paquete en ninguna de los casos.

Motor de captura	**% de paquetes procesados**			
	1 NIC		**2 NICs**	
	64 Bytes	**CAIDA**	**64 Bytes**	**CAIDA**
ixgbe	2.7	100	3.7	93.55
PF_RING	100	100	76.1	100
HPCAP	97.9	100	97.8	100
DPDK	100	100	100	100

Tabla 4.1: Porcentaje de paquetes capturados en un escenario sin virtualización ni almacenamiento de paquetes.

Una vez se han obtenido los resultados de las primeras pruebas, es posible comenzar a realizar las pruebas conjuntas de captura y almacenamiento a disco. Cabe recordar, que en estas pruebas pueden empezar a surgir cuellos de botella en los accesos a memoria, así como en los accesos a los buses PCIe. Tal y como se muestra en la tabla 4.2, el rendimiento con respecto a la tabla 4.1 decrece en cierta medida. Un detalle a tener en cuenta es el descenso en el rendimiento de la aplicación desarrollada en DPDK. Al igual que el zero-copy ayuda a mejorar el rendimiento, si el pipeline de proceso tiene demasiada latencia, se comienza a perder paquetes. Por otro lado, en una filosofía one-copy, como la de *HPCAP*, se independiza mucho mejor el procesamiento sobre los paquetes (en este caso copia a disco) del proceso de captura. Por este motivo, la tasa de paquetes capturados utilizando *HPCAP* se ve muy poco afectada cuando se añade el proceso de almacenado. Mientras que en la implementación de captura sobre DPDK, se pierde más de un 4 % de los paquetes, frente a la implementación de solo captura que no perdía ningún paquete. Finalmente, cabe mencionar que las pruebas sobre captura y almacenamiento se han realizado utilizando una única NIC. Dado que las pruebas del raid indican que solo es capaz de almacenar hasta 10 Gbps, carece de sentido realizar una prueba que claramente va a estar acotada por esta cifra.

Motor de captura	**% de paquetes procesados**	
	64 Bytes	**CAIDA**
ixgbe	10,3	91.9
HPCAP	95.7	100
DPDK	95.8	100

Tabla 4.2: Porcentaje de paquetes capturados con almacenamiento y sin virtualización.

4.4. Pruebas en entornos virtuales

Una vez se dispone de los primeros resultados en los entornos físicos es posible iniciar con las pruebas dentro de diferentes entornos y configuraciones virtuales. Es fácil suponer que los nuevos resultados que se obtendrán dentro de un entorno virtualizado, sufrirán algún tipo de degradación pues inevitablemente existe una pequeña sobrecarga en el sistema debido a la ejecución simultanea de diversos sistemas operativos.

Llegados a este punto, es interesante recordar la motivación de la virtualización. La virtualización en redes de comunicaciones suele venir por temas de escalabildiad, o problemas

a la hora de insertar una nueva máquina en un CPD. Esto significa, que en un principio la máquina virtual dedicada captura, va a compartir con una alta probabilidad la maquina física en la que se encuentra. Esto hace plantearse diferentes métodos para compartir los recursos y operar dentro de estas VMs. Por ello, en esta sección se pretende mostrar, no solo las diferentes combinaciones de virtualización de los dispositivos, sino también, una comparativa en cuanto ventajas y desventajas que suponen cada uno de estos métodos.

4.4.1. Usando Paravirtualización: VIRTIO

Antes de que los sistemas de virtualización contasen con la tecnología necesaria para realizar Passthrough o SR-IOV, se recurría a la virtualización completa del dispositivo y a la paravirtualización. Dado que el objetivo es alcanzar altas tasas de captura y almacenamiento, puede parecer poco producente dedicar esfuerzo a probar esta clase de tecnología que requiere un gran esfuerzo por parte del hypervisor a la hora de realizar el interconexionado entre el mundo físico y el mundo virtual.

Dado que existe un gran esfuerzo en optimizar el sistema de paravirtualización de KVM con los módulos VirtIO, se ha decidido darle una oportunidad. A diferencia de un sistema de virtualziación completo, una paravirtualización construye un pequeño hardware virtual que permite compartir un determinado dispostivo hardware entre la máquina física y diversas máquinas virtuales. Este pequeño hardware virtual, es lo suficientemente ligero como para no suponer una elevada perdida de rendimiento pero lo suficientemente completo como para proporcionar, en principio, todas las funcionalidades del hardware original. Este hardware virtual, requiere a su vez de drivers especiales en las máquinas virtuales.

Teniendo en cuenta que los dispositivos VirtIO se construyen como una aplicación más que utiliza el hardware subyacente, parece que la aproximación de utilizar VirtIO con la tarjeta de red, pierde sentido. Si bien, el driver *ixgbe* es incapaz de ofrecer un rendimiento consistente e independiente del procesamiento, añadir una nueva capa de procesamiento solo empeorará el rendimiento. Por este motivo, se ha descartado utilizar VirtIO con las tarjetas de red.

Por otro lado, la idea de poder compartir un raid entre diferentes máquinas virtuales parece atractiva. No obstante, resulta complicado encontrar tarjetas Raid con soporte *sriov*, y las que lo soportan no suelen ser capaces de asociar un raid a más de una función virtual, impidiendo así su compartición. Por otro lado, los driver VirtIO permiten compartir un cualquier dispositivo de bloques entre diferentes máquinas virtuales. Es importante mencionar, que dado que este tipo de compartición se hace a nivel de dispositivo de bloques, si varias máquinas virtuales escriben a la vez sobre un sistema de ficheros, pueden llegar a darse problemas e incluso a corromperse el sistema de ficheros perdiendo toda la información del Raid. A pesar de este riesgo, este modelo sigue resultando atractivo en temas de monitorización, en donde una máquina virtual escribe a disco los datos de red y otras máquinas virtuales los leen únicamente para realizar diferentes tipos de análisis. Para analizar el rendimiento de los driver VirtIO se ha realizado el mismo conjunto de pruebas de escritura en el raid que en la sección anterior cuyos resultados pueden observarse en la figura 4.5. Si miramos con detalle los resultados obtenidos, podremos observar que el rendimiento del raid no ha caído, sino que ha aumentado muy ligeramente con respecto a la versión física. Aunque pueda parecer extraño, esto es debido al funcionamiento de VirtIO. Para suplir la sobrecarga de un minidriver, VirtIO se asegura de realizar las escrituras en bloques de tamaños óptimos, buffereando las diferentes peticiones de escritura. Por este motivo, y de forma extraordinaria, las escrituras a disco mediante VirtIO aparentan ser ligeramente más eficientes.

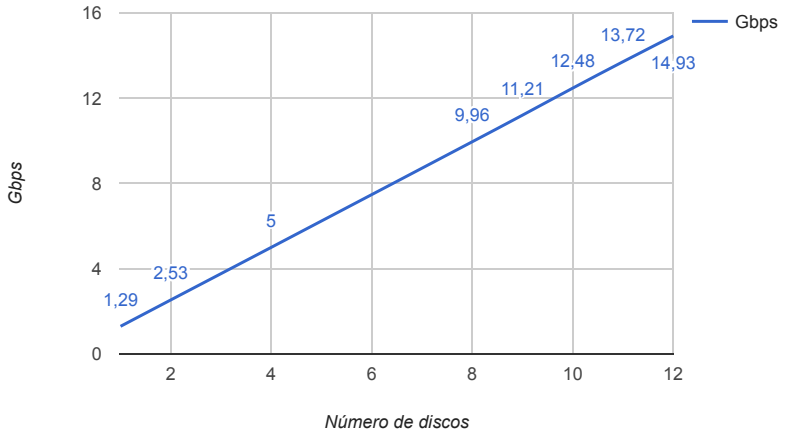

Figura 4.5: Rendimiento de escritura en Raid 0 utilizando VirtIO

4.4.2. Usando Passthrough

Dentro de los modelos de virtualización clásicos, nos encontramos con el modelo de Passthrough. Este sistema de virtualización consiste en transferir todo el control sobre un determinado dispositivo a la máquina virtual. De esta forma, una VM, es capaz de utilizar los driver nativos para susodicho dispositivo mitigando enormemente los efectos de la virtualización.

A pesar de que en términos de rendimiento la tecnología de Passthrough se encuentra aventajada, esta forma de virtualización no permite que los recursos se compartan. Recordemos que una de las premisas de la virtualización es la compartición de recursos, ya que se asume que una única máquina virtual usualmente no explota al 100 % todos los recursos de los que dispone. En la figura 4.6 se muestra una arquitectura en la cual existen diversas máquinas virtuales. No obstante, aunque las aplicaciones dentro de una máquina virtual pueden compartir el harware, las aplicaciones de otras VMs requieren sus propios dispositivos.

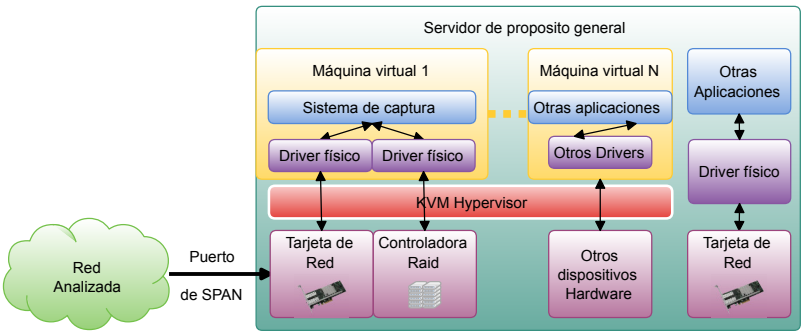

Figura 4.6: Arquitectura de captura en un escenario con Passthrough

Bajo este paradigma de virtualización, se han vuelto a repetir las pruebas de escritura en el raid. Dado que la máquina virtual gestiona la controladora raid de la misma forma que lo haría una máquina real, el rendimiento se ve muy poco alterado con el rendimiento medido en un entorno real. Los resultados de estas pruebas pueden verse en la figura 4.7.

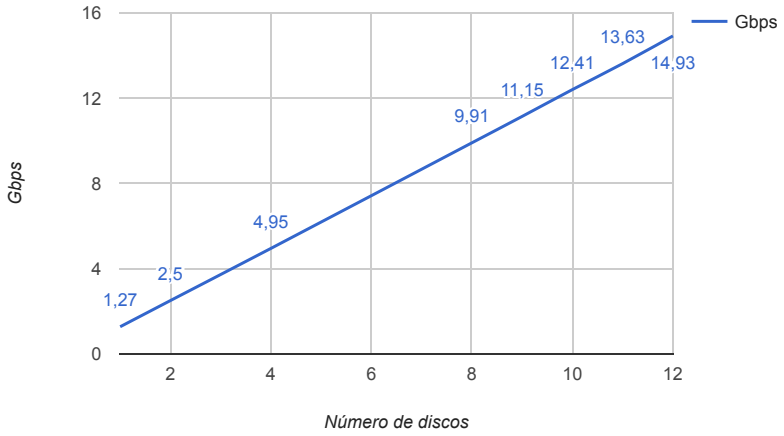

Figura 4.7: Rendimiento de escritura en Raid 0 utilizando Passthrough

Una de las ventajas que aportan las tarjetas de red de 10 Gbps de Intel es la representación de cada NIC como un dispositivo PCIe independiente. Esto permite que cada interfaz de la tarjeta de red pueda ser utilizada por una máquina diferente, ya sea esta física o virtual. Con el objetivo de observar como afecta dividir el tráfico entre un entorno físico y un entorno virtual, se han realizado pruebas en las siguientes 3 arquitecturas: 1 NIC en una única máquina virtual, 2 NICs en una única máquina virtual y 2 NICs (1+1) repartidas entre una máquina virtual y su sistema operativo anfitrión. Los resultados de la prueba de solo captura pueden verse en la taba 4.3.

Motor de captura	**% de paquetes procesados**					
	1 NIC		**2 NICs**		**1+1 NICs**	
	64 Bytes	**CAIDA**	**64 Bytes**	**CAIDA**	**64 Bytes**	**CAIDA**
ixgbe	1.9	62.7	2.5	41.8	6.2	83.7
PF_RING	99.8	100	75.2	100	99.7	100
HPCAP	85.2	100	33.6	99.9	89.9	100
DPDK	100	100	100	100	100	100

Tabla 4.3: Porcentaje de paquetes capturados mediante Passthrough sin almacenamiento de paquetes.

En los resultados mostrados por la tabla anterior, se aprecia una pequeña degradación en el rendimiento cuando una única máquina virtual gestiona más de 1 NIC. Es probable que este efecto se deba a que cada una de las VMs tiene un número de recursos limitado inferior al del sistema operativo anfitrión. Si los motores de captura requieren más recursos (ya sean cores o memoria) y no disponen de ellos, inevitablemente el rendimiento se ve afectado.

Finalmente, en la tabla 4.4, se muestran los resultados de las pruebas de realizar almacenamiento y captura simultáneamente sobre una única máquina virtual. Esta tabla muestra a su vez el rendimiento de captura del sistema tanto cuando el raid se encuentra conectado mediante VirtIO, como cuando se encuentra conectado mediante Passthrough. Es interesante observar nuevamente el efecto de las filosofías one-copy y zero-copy. A pesar de que la escritura mediante VirtIO mejora ligeramente el rendimiento del raid, las copias intermedias causan un breve aumento en la latencia y duración del pipeline de proceso. Mientras que el motor de captura

Motor de captura	% de paquetes procesados			
	Raid en Passthrough		Raid en VirtIO	
	64 Bytes	CAIDA	64 Bytes	CAIDA
HPCAP	82.2	100	82.8	100
DPDK	97.6	100	95.3	100

Tabla 4.4: Porcentaje de paquetes almacenados en un escenario con Passthrough.

DPDK se ve afectado negativamente por el aumento de este retardo, el motor de captura *HPCAP* tiene una mayor independencia y se ve beneficiado por el aumento de velocidad medio.

4.4.3. Usando SR-IOV

La tecnología SR-IOV ofrece multitud de ventajas frente a los otros métodos de virtualización, no sin por supuesto, traer nuevos inconvenientes. La mayor ventaja que ofrece SR-IOV es la compartición de recursos, al igual que lo es su mayor desventaja, pues se crea un cuello de botella. Cuando hablamos de SR-IOV en redes de comunicaciones, comúnmente estamos hablando de NFV, es decir funciones virtuales de una determinada interfaz de red. Recordemos, que para que una tarjeta de red pueda crear diversas NFVs, es necesario que la tarjeta física actúe a su vez como un switch entre las diferentes NFVs. Esto aporta cierto valor añadido, pues mejora la comunicación interna entre las diferentes máquinas virtuales. Como contraparte, da un mayor trabajo a la tarjeta de red.

Dentro de este contexto, surge una nueva arquitectura de captura de tráfico. Dado que existen NFVs en nuestro escenario, la aplicación de captura comparte las interfaces con otras aplicaciones de red (como puede ser un servidor web apache, una base de datos, o cualquier aplicación que utilice la red). Todas estas aplicaciones, tanto en máquinas máquinas virtuales, como en el propio anfitrión, utilizan un driver virtual para explotar un segmento de los recursos físicos de la tarjeta de red. En la figura 4.8 se presenta una arquitectura de captura en la que hay presentes diferentes elementos todos ellos conectados mediante SR-IOV y drivers virtuales. Nótese, que siguen existiendo elementos (la controladora Raid) que siguen siendo virtualizados con Passthrough pues no disponen de SR-IOV.

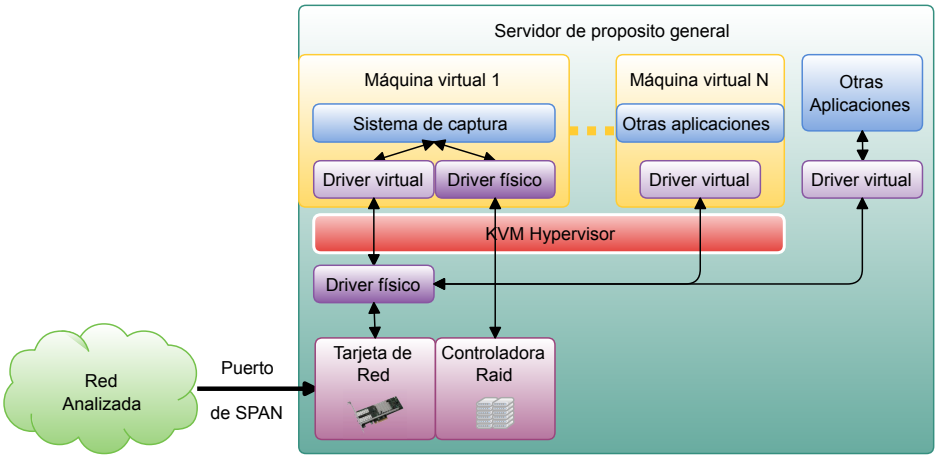

Figura 4.8: Arquitectura de captura en un escenario con SRIOV y NFVs

Comparativa de construcción de NFVs

Para que un dispositivo genere funciones virtuales, es necesaria la presencia de un driver en la máquina anfitriona que configure el dispositivo físico (número de VFs, tamaños de las colas, reparto de interrupciones, etc). Para el caso de las tarjetas de Intel de 10 Gbps, el driver vanilla ofrece esta funcionalidad de una manera muy cómoda. Este driver cumple con la filosofía de SR-IOV de delegar en la tarjeta todo el proceso de reparto de interrupciones y paquetes entre las diferentes NFV.

No obstante, Intel ofrece una segunda forma de gestionar estas NFVs. Los drivers de DPDK permiten a su vez generar estas funciones virtuales, pero a diferencia del driver *ixgbe*, es necesario ejecutar un programa sobre DPDK para que estas NFV reciban tráfico. El programa, conocido como *testpmd*, ayuda activamente a la tarjeta liberándola en parte de la lógica que supone realizar de switch y la copia de paquetes entre la cola de recepción física y las colas asociadas a cada función virtual. A su vez este programa permite configurar elementos a muy bajo nivel de la tarjeta, que en el driver *ixgbe* supondría modificar el código del propio driver.

En la tabla 4.5 se muestra una comparativa entre los diferentes métodos de generación de NFV. Claramente, parece que el sistema de generación mediante DPDK supera por mucho en rendimiento al driver vanilla. Por otro lado, para que la aplicación *testpmd* funcione a pleno rendimiento, es necesario reservarla un core completo por cada función virtual, al igual que otro core para ejecutar la interfaz. Esta limitación obliga a la larga a disponer de un gran número de cores que tan solo copian paquetes entre colas físicas y virtuales.

Motor de captura	**% de paquetes procesados**	
	Gen. Ixgbe	**Gen. DPDK**
ixgbevf	>0.1	1
HPCAPvf	36.2	82.7
DPDK	37.6	100

Tabla 4.5: Porcentaje de paquetes capturados en función del método de generación de NFV. El tráfico utilizado estaba formado por paquetes de tamaño mínimo (64Bytes).

Pruebas de recepción con NFVs

De cara a realizar pruebas con NFVs, es necesario mencionar algunos detalles implementativos. Cada NFV, a diferencia de su hermana física, tiene una única cola de recepción y transmisión. Por temas de diseño, las NFVs tampoco cuentan con contadores de paquetes perdidos, o erróneos, por lo que obtener estos datos es necesario inferirlos de las estadísticas que ofrecen las funciones físicas. Por este motivo, no se han podido medir con exactitud las pérdidas en los diferentes escenarios con NFV, no obstante, se han obtenido datos bastante aproximados.

Como se ha mencionado anteriormente, la tarjeta física, es la encargada en última instancia de gestionar hacia que NFV se envía cada uno de los paquetes recibidos. Para determinarlo, cada NFV posee su propia dirección MAC virtual. Con el objetivo de probar la escalabilidad en función de máquinas virtuales y NFVs por NIC física, se han utilizado 3 máquinas virtuales con dos 2 cores virtuales cada una. Cada VM cuenta con una NFV con MAC única. Para medir el rendimiento, estas máquinas virtuales ejecutan en paralelo la versión de solo captura desarrollada en DPDK. Para realizar las pruebas se han creado 3 trazas de tráfico sinténtico. La primera traza contiene paquetes de tamaño 64 Bytes, todos ellos con una única dirección MAC. La segunda

traza, contiene 2 direcciones MAC en una proporcion del 50 % cada una. La tercera traza, contiene 3 direcciones MAC, en una proporcion del 33.3 % cada una.

Los resultados obtenidos de las pruebas figura en la tabla 4.6. En esta tabla, se pueden observar ligeras pérdidas pues no se llegan a alcanzar los 10 Gbps. No obstante, el rendimiento parece escalar y no se ve aparentemente afecto por el número de NFVs que hay en el sistema o el número de máquinas virtuales en ejecución.

Número de MACs emitidas	**% de paquetes procesados**	
	Por cada VM	**En total**
1	99.5	99.5
2	49.7	99.4
3	33.2	99.5

Tabla 4.6: Porcentaje de paquetes capturados en función del número de VMs y de direcciones MAC destino

NFVs con captura promíscua

No obstante, el método nativo de funcionamiento de las NFV, no nos sirve para poder construir una sonda de tráfico virtual, pues en un principio solo seríamos capaces de capturar el tráfico destinado a nuestra función virtual. Si el objetivo de nuestro sistema es ofrecer un servicio por una determinada interfaz y a la vez capturar por esta, podemos recurrir a la bandera Multicast Promiscuous Enable (MPE) [67]. Esta bandera puede ser configurada a través del programa *testpmd*, e indica a la tarjeta que reenvíe todos aquellos paquetes sin destino conocido a una determinada NFV. Nótese, que esta aproximación no nos reenvía aquellos paquetes destinados a una dirección MAC asociada a otra NFV de nuestro sistema. No obstante, esta aproximación es sencilla y permite a la tarjeta funcionar a alta velocidad, además de ser completamente válida si mantenemos la arquitectura del típico puerto de SPAN.

En la tabla 4.7 se muestran los resultados de solo captura. En este escenario, se pretende probar el rendimiento que ofrecen las NFVs bajo condiciones habituales. Para ello, se realizan 3 pruebas simples: Una única VM conectada a una NFV, una única VM conectada a dos NFV (cada una correspondiente a una NIC diferente) y finalmente, 2 VMs conectadas a 2 NFV generadas a partir de la misma NIC. Los resultados de la tabla son en parte similares a otras pruebas ya realizadas, aunque caben destacar algunos resultados. Para el caso de *HPCAPfv*, al gestionar 2 NFVs en una máquina virtual con pocos recursos, se ve saturado rápidamente y el rendimiento cae por falta de cores. En el caso de utilizar dos máquinas virtuales en paralelo compartiendo NIC, se observa que tanto en el motor DPDK como en el motor de captura*HPCAPvf*, tienen resultados similares. Esta última prueba de doble NFV y doble VM, se realizó mediante una traza de paquete mínimo con la MAC destino a broadcast, de forma que todos las NFVs recibieran todos y cada uno de los paquetes. Si tenemos esto en cuenta, aunque el tráfico que llega por la interfaz se recibe a 10 Gbps, la tarjeta debe enviar datos a las diferentes NFVs a 20 Gbps. Con esto en mente, el 51 % o el 44.1 % de 20 Gbps, se asemeja mucho a valores obtenidos anteriormente.

En la tabla 4.8 se muestran los resultados de captura y almacenamiento cuando solo hay una única NFV y una única VM en ejecución. Es interesante apreciar como la sobrecarga de la virtualización con VirtIO y NFV incrementa lo suficiente la latencia del pipeline de captura de la herramienta de DPDK como para hacerla perder una enorme cantidad de paquetes.

Motor de captura	% de paquetes procesados					
	1 NFV		2 NFVs		(1+1) 2NFVs/2VMs	
	64 Bytes	CAIDA	64 Bytes	CAIDA	64 Bytes	CAIDA
ixgbePvf	1	34.2	1	42.4	0.9	14.8
HPCAPvf	82.7	100	13.0	100	44.1	82.7
DPDK	100	100	100	100	50.8	83.0

Tabla 4.7: Porcentaje de paquetes capturados y no almacenados en un escenario con SRIOV y flag MPE.

Motor de captura	% de paquetes procesados			
	Raid en Passthrough		Raid en VirtIO	
	64 Bytes	CAIDA	64 Bytes	CAIDA
HPCAPvf	82.6	100	82.3	100
DPDK	94.2	100	68.2	100

Tabla 4.8: Porcentaje de paquetes almacenados en un escenario con SRIOV y flag MPE.

NFVs con captura global

En las secciones anteriores de NFVs, se ha hablado acerca de como funciona el reparto de paquetes entre las diferentes funciones y de como es posible capturar tráfico que no va destinado a ninguna función virtual. No obstante, uno de los objetivos de este trabajo es evaluar el proceso de monitorización en redes virtuales. Para lograrlo, el datasheet de las tarjetas de Intel [67] nos proporciona información acerca del concepto de *Port Mirroring*. A través de una serie de registros, es posible indicar a la tarjeta de red, que copie todos los paquetes que entren o salgan de una cola a otra cola. Dado que cada NFV, tiene asociada una única cola, es posible forzar la copia del contenido de una NFV a otra NFV. Toda esta configuración de *mirroring* puede ser configurada a través de la aplicación *testpmd*.

La idea principal para realizar esta prueba consiste en utilizar las 3 máquinas virtuales mencionadas al inicio de esta subsección como consumidoras de tráfico, mientras que una cuarta máquina virtual, con la configuración estándar utilizada hasta ahora(y descrita en la sección 4.1.2), monitorice el tráfico entre dichas VMs. Dado que el equipo de pruebas tiene arquitectura NUMA y los recursos de las 4 VMs exceden los recursos de un procesador, es necesario distribuir las diferentes VMs entre los procesadores. Dado que la localización de las VM puede afectar al rendimiento, se ha decidido agrupar las 3 VMs consumidoras en un mismo procesador, pues comparten funcionalidad similar, y la VM de monitorización en otro procesador.

En la tabla 4.9 se muestra un breve resumen de las pruebas realizadas. Por un lado, los resultados muestran una tasa acotada de Gbps que pueden ser capturados mediante este método. Dado que el *Port Mirroring* consiste en duplicar todos y cada uno de los paquetes, el hecho de que el rendimiento decrezca es razonable, pues en definitiva se están recibiendo al menos el doble Gbps. Un dato curioso, son los resultados de rendimiento. Si recordamos la arquitectura de la sonda de prueba, el tarjeta de red se encuentra conectada en el nodo NUMA. Dado que este nodo es el más cercano a la tarjeta, parece razonable ejecutar la VM en este nodo. No obstante, cuando la VM es capaz de capturar mucho más tráfico cuando se encuentra en el nodo NUMA 1. Este efecto ha de tenerse en cuenta, pues el *Port Mirroring*, no redirige los paquetes a la VM de captura, salvo que dicho paquete ya haya sido atendido por la VM adecuada. Por ello, situar las máquinas consumidoras en el nodo más cercano a la tarjeta favorece la cantidad de paquetes

capturados mediante la técnica del *Port Mirroring*.

Motor de captura	**Nodo Numa de captura**	**% de paquetes procesados**	
		64 Bytes	**CAIDA**
HPCAPvf	0	30.7	48.0
	1	47.4	76.7
DPDK	0	22.8	38.4
	1	50.4	76.8

Tabla 4.9: Porcentaje de paquetes almacenados en un escenario con SRIOV y Port Mirroring. Captura del tráfico interno de una red virtual formada por 3 VMs. El raid de almacenamiento se encuentra configurado con VirtIO

Sistema distribuido de monitorización

Aunque las pruebas de rendimiento mostradas en la tabla 4.9 parece que no son excepcionales, debemos recordar que encontrar un único servidor capaz de saturar un enlace a 10 Gbps resulta complicado incluso si sus servicios no sufren de ningún tipo de virtualización. Por otro lado, los servicios de virtualización se nutren de utilizar grandes cantidades de máquinas virtuales relativamente pequeñas.

A partir de este concepto, nace la idea de un sistema de monitorización y captura de tráfico distribuido. El coste de añadir una pequeña máquina virtual encargada de recolectar el tráfico de las demás VMs con las que comparte anfitrión, es muy bajo. Dado que todas estas máquinas virtuales tienen una vía de comunicación entre sí, basta con añadir una nueva VM que de forma transparente sea capaz de realizar un análisis con los datos capturados de cada una de las mini-sondas de captura virtuales. No obstante, construir y experimentar con este tipo de sistema distribuido queda fuera del alcance de este trabajo, aunque será considerada su realización como trabajo futuro

5

Conclusiones y Trabajo futuro

A lo largo de este trabajo se ha explorado el estado actual tanto de los sistemas de captura como de los sistemas de virtualización. En base a los sistemas y equipamiento disponibles se han realizado multitud de pruebas para comparar las diferentes aplicaciones y entornos. Estos experimentos han desvelado los regímenes de funcionamiento de los diferentes métodos de virtualización y aplicaciones de captura de red. Para sorpresa de muchos, la mayoría de la tecnología de virtualización actual ofrece un rendimiento equiparable a los entornos nativos de funcionamiento. De esta forma, es posible migrar los sistemas de captura a alta velocidad a los entornos virtuales. Vistos los resultados, es probable que sistemas de monitorización específicos también puedan ser migrados a entornos virtuales sin verse afectados.

En la realización de las pruebas, se han descubierto y descubierto nuevas formas de realizar una monitorización de redes virtuales que hasta ahora no habían sido propuestas o alcanzadas. Gracias a las pruebas de SR-IOV, se descubrieron dos posibles aplicaciones nuevas para la monitorización en entornos virtuales. Mediante las NFVs, es posible capturar a un subconjunto de NFVs que utilicen la misma NIC. Aunque este escenario no permita alcanzar tasas extremadamente altas, hasta donde llega mi conocimiento, no existía ninguna herramienta capaz de capturar el tráfico interno entre dos máquinas virtuales que utilizasen NFV. A raíz de las pruebas, también han surgido nuevas controversias. La mayoría de los capturadores o emisores de tráfico software se aferran a la filosofía del zero-copy. No obstante, las pruebas han mostrado que utilizar de forma adecuada una filosofía one-copy permite crear un cierto nivel de aislamiento entre el sistema de captura y el sistema de procesamiento de paquetes. Con este método, que a priori parece más lento, se puede reducir el número de estructuras de paquetes, e incluso aumentar la tasa.

Otra de las ideas surgidas a raíz de este trabajo, se trata de la monitorización distribuida. Partiendo de la suposición de disponer de múltiples máquinas virtuales que comparten NICs, es posible instanciar nuevas máquinas virtuales que monitoricen dichas NICs a través de NFVs. Dichas máquinas virtuales se coordinarían con otra máquina virtual maestra y de forma distribuida podrían llegar a realizar análisis de red complejos en muy poco tiempo. Dado que los paradigmas distribuidos están tomando fuerza, parece una línea de investigación interesante por

la que continuar en un futuro cercano.

A partir de este punto, se espera que los experimentos realizados ayuden a la comunidad a explotar la funcionalidad de los diferentes métodos de virtualización sin tener que enfrentarse a realizar un extenso análisis de cada uno de los elementos probados. Para lograr este objetivo, se han hecho públicos 2 dos de los tres elementos desarrollados: La librería hptl [2] y la aplicación de solo captura sobre DPDK [62]. Dentro del marco de publicar el trabajo desarrollado, se está trabajando actualmente en la realización de una versión one-copy del capturador y almacenador a disco de DPDK, con la esperanza de minimizar o eliminar las perdidas cuando el procesamiento del tráfico tiene mucha latencia. Del mismo modo que se ha hecho con las herramientas construidas, se ha estado trabajando en la realización de dos artículos. El primero de ellos se encuentra aceptado en el congreso *High Performance Computing and Communications* (HPCC 2015, Core B, ver anexo A). El segundo artículo se encuentra a la espera de ser enviado a una revista Q1. También se ha planteado realizar una extensión de este trabajo, añadiendo un estudio de VFs sobre FPGAs y controladoras raid.

Bibliografía

[1] R. Leira Osuna, "Clasificación de flujos en 10 gbps ethernet mediante intel dpdk y gpus," B.S. Thesis, Universidad Autónoma de Madrid, 2013, http://ir.ii.uam.es/~fdiez/TFGs/gestion/leidos/2013/20130611RafaelLeiraOsuna.pdf, [16 Junio 2015].

[2] ——, "High Performance Timing Library (HPTL)," 2015, https://github.com/ralequi/hptimelib, [16 Junio 2015].

[3] KVM, "Kernel Virtual Machine start page," 2015, http://www.linux-kvm.org/page/Main_Page, [15 Junio 2015].

[4] Pcie sr-iov official specifications. [Online]. Available: https://www.pcisig.com/specifications/iov/single_root/

[5] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[6] Global smartphone penetration 2014. [Online]. Available: https://ondeviceresearch.com/blog/global-smartphone-penetration-2014

[7] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610001568

[8] Crecimiento de internet entre 1995 y 2014. [Online]. Available: http://www.internetworldstats.com/emarketing.htm

[9] J. R. de Santiago and J. A. Rico, "Proactive measurement techniques for network monitoring in heterogeneous environments," Ph.D. dissertation, Universidad Autónoma de Madrid, 2013.

[10] R. Leira Osuna, P. Gomez Nieto, I. Gonzalez, and J. Lopez de Vergara, "Multimedia flow classification at 10 gbps using acceleration techniques on commodity hardware," in *4th IEEE Technical CoSponsored International Conference on Smart Communications in Network Technologies 2013 (SaCoNeT 2013)*, Paris, France, Jun. 2013.

[11] D. Hernando-Loeda, J. E. López de Vergara, J. Aracil, D. Madrigal, and F. Mata, "Measuring mpeg frame loss rate to evaluate the quality of experience in iptv services," *Quality of Experience Engineering for Customer Added Value Services*, pp. 31–51, 2013.

[12] J. L. García-Dorado, P. M. Santiago del Río, J. Ramos, D. Muelas, V. Moreno, J. E. López de Vergara, and J. Aracil, "Low-cost and high-performance: Voip monitoring and full-data retention at multi-gb/s rates using commodity hardware," *International Journal of Network Management*, vol. 24, no. 3, pp. 181–199, 2014.

[13] J. Ramos, P. S. del Río, J. Aracil, and J. L. de Vergara, "On the effect of concurrent applications in bandwidth measurement speedometers," *Computer Networks*, vol. 55, no. 6, pp. 1435 – 1453, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610003683

[14] "Netfpga." [Online]. Available: http://netfpga.org/

[15] S. Han, K. Jang, K. Park, and S. Moon, "Packetshader: a gpu-accelerated software router," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 195–206, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1851275.1851207

[16] V. Moreno, "Development and evaluation of a low-cost scalable architecture for network traffic capture and storage for 10Gbps networks," Master's thesis, Universidad Autónoma de Madrid, 2012, http://www.ii.uam.es/~vmoreno/Publications/morenoTFM2012.pdf, [1 August 2014].

[17] L. Rizzo, M. Carbone, and G. Catalli, "Transparent acceleration of software packet forwarding using netmap," in *INFOCOM, 2012 Proceedings IEEE*, 2012, pp. 2471–2479.

[18] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi, "On multi—gigabit packet capturing with multi—core commodity hardware," in *Proceedings of the 13th international conference on Passive and Active Measurement*, ser. PAM'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 64–73. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28537-0_7

[19] Intel, "Intel Data Plane Development Kit (Intel DPDK) Release Notes," 2015, http://www.dpdk.org/doc/guides/rel_notes/, [03 Abril 2015].

[20] M. Kodialam and T. Lakshman, "Detecting network intrusions via sampling: a game theoretic approach," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3, March 2003, pp. 1880–1889 vol.3.

[21] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 159–164. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177101

[22] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 5–16, Jan. 2007.

[23] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core," in *Proceedings of the 16th international conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 883–892.

[24] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, ser. MineNet '06. New York, NY, USA: ACM, 2006, pp. 281–286.

[25] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proceedings of the 2006 ACM CoNEXT conference*, ser. CoNEXT '06. New York, NY, USA: ACM, 2006, pp. 6:1–6:12.

[26] N. Cascarano, A. Este, F. Gringoli, F. Risso, and L. Salgarelli, "An experimental evaluation of the computational cost of a dpi traffic classifier," in *Proceedings of the 28th IEEE conference on Global telecommunications*, ser. GLOBECOM'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1132–1139.

[27] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. Markatos, and S. Ioannidis, "Gnort: High performance network intrusion detection using graphics processors," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, R. Lippmann, E. Kirda, and A. Trachtenberg, Eds. Springer Berlin Heidelberg, 2008, vol. 5230, pp. 116–134.

[28] B. Hutchings, R. Franklin, and D. Carver, "Assisting network intrusion detection with reconfigurable hardware," in *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, 2002, pp. 111–120.

[29] C. Clark and D. Schimmel, "Scalable pattern matching for high speed networks," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, 2004, pp. 249–257.

[30] J. Moscola, J. Lockwood, R. Loui, and M. Pachos, "Implementation of a content-scanning module for an internet firewall," in *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*, 2003, pp. 31–38.

[31] H.-J. Jung, Z. Baker, and V. Prasanna, "Performance of fpga implementation of bit-split architecture for intrusion detection systems," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, pp. 8 pp.–.

[32] J. A. A. Sáez, "Procesamiento mediante gpu en netfilter (linux)," Master's thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2012.

[33] l7-filter. [Online]. Available: http://l7-filter.clearfoundation.com

[34] P. S. del Río, "Internet traffic classification for high-performance and off-the-shelf systems," Ph.D. dissertation, Universidad Autónoma de Madrid, May 2013. [Online]. Available: http://arantxa.ii.uam.es/~psantiago/Publications/dissertation.pdf

[35] Claise, Benoit, "Cisco systems netflow services export version 9," RFC 3954, Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt

[36] "Netflow." [Online]. Available: http://www.cisco.com/go/netflow

[37] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–1, 2014.

[38] Claise, Benoit, "Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information," RFC 5101, Jan. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5101.txt

[39] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[40] M.-S. Kim, Y. J. Won, and J. W. Hong, "Characteristic analysis of Internet traffic from the perspective of flows," *Computer Communications*, vol. 29, no. 10, pp. 1639–1652, 2006.

[41] M. Forconesi, G. Sutter, S. López-Buedo, and C. Sisterna, "Clasificación de flujos de comunicación en redes de 10 gbps con fpgas," in *Jornadas SARTECO 2012*, 2012. [Online]. Available: http://www.jornadassarteco.org/js2012/papers/paper_64.pdf

[42] Tilera, "Tilera Network Processor," 2015, http://www.tilera.com/, [14 Junio 2015].

[43] Procera, "Procera Networks and Tilera Unleash 200Gbps Deep Packet Inspection Solution," 2015, http://www.proceranetworks.com/recent-press-releases/1203-procera-networks-and-tilera-unleash-200gbps-deep-packet-inspection-solution.html, [15 Junio 2015].

[44] J. F. Zazo, M. Forconesi, S. Lopez-Buedo, G. Sutter, and J. Aracil, "Tnt10g: A high-accuracy 10 gbe traffic player and recorder for multi-terabyte traces," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*. IEEE, 2014, pp. 1–6.

[45] J. J. Garnica, V. Moreno, I. Gonzalez, S. Lopez-Buedo, F. J. Gomez-Arribas, and J. Aracil, "Argos: A gps time-synchronized network interface card based on netfpga," in *2nd North American NetFPGA Developers Workshop*, 2010.

[46] U. Kaemmerer and M. Hortig, "Method and system for recording, synchronizing and analysing data by means of analysis devices which are spatially distributed in a communication network," May 12 2015, uS Patent 9,031,055.

[47] L. Rizzo, L. Deri, and A. Cardigliano, "10 Gbit/s line rate packet processing using commodity hardware: survey and new proposals," 2012, online: http://luca.ntop.org/10g.pdf [1 August 2014]. [Online]. Available: http://luca.ntop.org/10g.pdf

[48] ntop, "Libzero for DNA," 2014, www.ntop.org/products/pf_ring/libzero-for-dna/, [1 August 2014].

[49] "Models for packet processing on multi-core systems," White Paper, Intel, Dec. 2008. [Online]. Available: http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/ia-multicore-packet-processing-paper.html

[50] L. Rizzo, "netmap: a novel framework for fast packet I/O," in *Proceedings of USENIX Annual Technical Conference*, 2012.

[51] ——, "Revisiting network I/O APIs: the netmap framework," *Communications of the ACM*, vol. 55, no. 3, pp. 45–51, 2012.

[52] ——, "Portable packet processing modules for OS kernels," *IEEE Network*, vol. 28, no. 2, pp. 6–11, 2014.

[53] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi, "On multi-gigabit packet capturing with multi-core commodity hardware," in *Proceedings of Passive and Active Measurement Conference*, 2012.

[54] V. Moreno, J. Ramos, P. Santiago del Rio, J. Garcia-Dorado, F. Gomez-Arribas, and J. Aracil, "Commodity packet capture engines: tutorial, cookbook and applicability," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–1, 2015.

[55] D. Schlosser, M. Duelli, and S. Goll, "Performance comparison of hardware virtualization platforms," in *NETWORKING 2011*. Springer, 2011, pp. 393–405.

[56] VMware, "VMware start page," 2015, http://www.vmware.com/es, [15 Junio 2015].

[57] X. Project, "Xen Project start page," 2015, http://www.xenproject.org/, [15 Junio 2015].

[58] OpenVz, "OpenVz wiki page," 2015, https://openvz.org/Main_Page, [15 Junio 2015].

[59] VirtualBox, "VirtualBox start page," 2015, https://www.virtualbox.org/, [15 Junio 2015].

[60] "Apache Hadoop," [Disponible: http://hadoop.apache.org]. [Online]. Available: http://hadoop.apache.org

[61] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[62] R. Leira Osuna, "Intel Data Plane Development Kit (DPDK) Speedometer (Bandwidth consumed monitor)," 2015, https://github.com/hpcn-uam/iDPDK-Speedometer, [16 Junio 2015].

[63] K. Wiles, "Traffic generator powered by DPDK (pktgen-dpdk)," 2015, http://dpdk.org/browse/apps/pktgen-dpdk/, [22 Junio 2015].

[64] C. Walsworth, E. Aben, k. Claffy, and D. Andersen, "The CAIDA anonymized Internet traces 2014 dataset," http://www.caida.org/data/passive/passive_2014_dataset.xml, [1 October 2014].

[65] A. Kudryavtsev, V. Koshelev, and A. Avetisyan, "Prospects for virtualization of high-performance x64 systems," *Programming and Computer Software*, vol. 39, no. 6, pp. 285–294, 2013. [Online]. Available: http://dx.doi.org/10.1134/S0361768813060042

[66] TCPDump and Libpcap, "TCPDump," 2015, http://www.tcpdump.org/, [24 Junio 2015].

[67] Intel, "82599 10 Gbe controller datasheet," 2012, http://www.intel.com/content/www/us/en/ethernet-controllers/82599-10-gbe-controller-datasheet.html, [1 August 2014].

Apéndices

A

Towards high-performance network processing in virtualized environments

El artículo publicado se titula " Towards high-performance network processing in virtualized environments " y ha sido enviado al congreso HPCC, el cual está catalogado como **CORE B**. A continuación, se muestra la carta de aceptación, las revisiones y finalmente el artículo en sí.

A.1. Email de aceptación

```
Fecha:  6 de junio de 2015, 21:29:31 CEST
De:  HPCC 2015 <hpcc2015@easychair.org>
Para:  Víctor Moreno <victor.moreno@uam.es
Asunto: HPCC 2015 notification for paper 51

Dear Víctor,

Thank you for your contribution to IEEE HPCC 2015.

Congratulations! Your paper #51, titled as "Towards high-performance network
processing in virtualized environments", has been officially accepted as a
full paper for publication and presentation in IEEE HPCC 2015.

Please check reviewers' comments, and prepare your final version based on IEEE
conference proceeding format. The size of your final paper should be up to 6
pages complimentary, or 12 pages with the over length charge.

The completed reviews are attached below.

In order to publish your article, please prepare your final camera ready version
```

according to the requirements on IEEE HPCC website.

We will send you the detailed instruction about your final submission in another mail later.

Please prepare your trip (visa, air ticket, hotel) in advance. See you at New York City.

Best,
PC Chairs of IEEE HPCC 2015

A.2. Revisiones

A.2.1. Revisor 1

OVERALL EVALUATION: 0 (borderline paper)
REVIEWER'S CONFIDENCE: 4 (high)

---------- REVIEW ----------
This paper presents a study on some of the existing network processing techniques in different configurations. By comparing the package capturing capabilities and performance, the authors claim that they provide the audience a series of guidelines for network application deployment with low cost, high performance, space efficiency, and low power consumption.

My major concern about the paper is its contribution. A large body of the paper is devoted to presenting a preliminary investigation of several well known technologies (PF_RING, Intel DPDK, HPCAP, etc.), omiting the detailed introduction to author's own framework HPCAPvf. The authors present their experimental results for various configurations and techniques. However, I feel there is a lack of the experimental details and methodology. To make their results more convincing, the authors should include a methodology section dedicated to elaborate their experimental approach. Too much space is taken to introduce the results, system structures and tradeoffs for basic, well know concepts such as huge pages, package handling in virtualized network, etc. Figure 4 and 5 take too much space. The author should use more space to introduce the details and techniques specific to this work. Regarding the presented results, why the authors use percentage of captured packages as a "perf!ormance" indicator of the compared techniques? I think this metric should be used to indicate the accuracy, not the performance.

The paper is not well written. There are many typos and mistakes. Just to name a few: In the sentence "traditional end-user network applications retrieves packets individually by means of system calls, which in Linux systems involves at least two context switches.", retrieves should be retrieve, involves should be involve. This sentence is long and reads awkward: "Those tests were made using the same hardware as previously mentioned and using KVM for creating and managing the VMs, and accordingly to the bare-metal scenario, the table depicts each capture engine's performance for

the worst case scenario and in an average scenario.", and accordingly should be according. There are many other places that the author should do a proofreading.

A.2.2. Revisor 2

OVERALL EVALUATION: 2 (accept)
REVIEWER'S CONFIDENCE: 2 (low)

---------- REVIEW ----------
This paper evaluates the performance of several well-known high-performance capture engines on virtual machines. In addition, the paper presents an open-source version of the HPCAP for virtual environments. The experimental results presented in this paper are valuable for network operators that want to employ network virtual functions.

The paper is well written and easy to understand even for a non-expert in this field. The experimental results are based on a realistic benchmark and show insightful results.

A.3. Artículo

Se adjunta el artículo completo en la siguiente página.

Towards high-performance network processing in virtualized environments

Victor Moreno, Rafael Leira, Ivan Gonzalez and Francisco J. Gomez-Arribas
High Performance Computing and Networking research group,
Universidad Autónoma de Madrid, Spain
{victor.moreno, rafael.leira, ivan.gonzalez, francisco.gomez}@uam.es

***Abstract*—Nowadays, network operators' amenities are populated with a huge amount of proprietary hardware devices for carrying out their core tasks. Moreover, those networks must include additional hardware appliances as they host more and more services and applications offered by third-party actors. Thus, such an infrastructure reduces the profitability, as including new hardware boxes in the network becomes increasingly harder in terms of space, cooling and power consumption. In a context in which virtualization has become a ubiquitous technique, industry and academia has turned an eye to Network Function Virtualization (NFV) to mitigate those effects and maximize potential earnings. NFV aims to transform future network architectures by exploiting standard IT virtualization technology to consolidate a variety of network processing elements onto standard commodity servers. On this work, we assess the feasibility of moving high-performance network processing tasks to a virtualized environment. For such purpose, we analyse the possible configurations that allow feeding the network traffic to applications running inside virtual machines. For each configuration, we compare the usage of different high-performance packet capture engines on virtual machines, namely PF_RING, Intel DPDK and HPCAP. Specifically, we obtain the performance bounds for the primary task, packet sniffing, for physical, virtual and mixed configurations. We also developed HPCAPvf, a counterpart of HPCAP for virtual environments, and made it available under a GPL license.**

Keywords*—*Virtual network functions, packet capture, virtual machines.

I. Introduction

Over the past decades, the use of the Internet has rapidly grown due to the emergence of new services and applications. The amount of services and applications available to end-users makes it necessary for those services' providers to deploy quality-assessment policies in order to distinguish their product among the rest. In this scenario, network processing and analysis becomes a central task that has to deal with humongous amounts of data at high-speed rates. Obviously, service providers must be able to accomplish such a challenging task using processing elements capable of reaching the required rates while keeping the cost as low as possible for the sake of profitability.

To deal with such amount data, specialized hardware (HW) solutions have been developed, which are traditionally based on the used of Field-Programmable Gate Arrays (FPGAs) or Network Processors. These solutions answer the high-performance needs for specific network monitoring tasks, e.g. routing or classifying traffic on multi-Gb/s links [1], [2]. However, those solutions imply high investments: such hardware's elevated cost rises capital expenditures (CAPEX), while operational expenditures (OPEX) are increased due to the difficulty of their operation, maintenance and evolution. Furthermore, HW life cycles become shorter as technology and services evolve, which prevents new earnings and limits innovation. Those drawbacks turn specialized HW solutions into a non-desirable option for large-scale network processing.

As an alternative to mitigate those negative effects, academia and industry turned an eye to the use of commodity hardware in conjunction with open source software [3]. Such combination is referred as an off-the-shelf system in the literature. The advantages of those systems lay in the ubiquity of those components, which makes it easy and affordable to acquire and replace them and consequently reduces CAPEX. Those systems are not necessarily cheap, but their wide-range of application allows their price to benefit from large-scale economies and makes it possible to achieve great degrees of experimentation. Additionally, such systems offer extensive and high-quality support, thus reducing OPEX. However, the increased demands for network processing capacity would be translated into a big number of machines even though if off-the-shelf systems were used. Such an amount of machines means high expenses in terms of power consumption and physical space. Moreover, the presence of commodity servers from different vendors empowers the appearance of interoperability issues. All those drawbacks damage the profitability that networked service providers may experience.

On the other hand, there has been an increasing trend regarding the use of virtualization for computational purposes. Such trend has been empowered by the inherent advantages provided by virtualization solutions [4]. In this light, network operators and service providers have been working during the last years on the development of the concept of Network Function Virtualization (NFV). This new paradigm aims to unify the environments where network applications

shall run by means of adding a virtualization layer on top of which network applications may run. This novel philosophy also allows merging independent network applications using unique hardware equipment. Consequently, the application on NFV can increase the benefits obtained by network service providers by (i) reducing their equipment investment by acquiring large-scale manufacturers' products and by reducing the amount of physical machines required, which also entails cuts in power consumption; (ii) speeding up network applications maturation cycle as all applications are developed in an unified environment; (iii) easing maintenance procedures and expenditures as testability is radically enhanced; (iv) opening the network applications' market for small companies and academia by minimizing risk and thus encouraging innovation; (v) the possibility to rapidly adjust network applications and resources based on specific clients requirements.

The development of this novel NFV philosophy has been favoured by other trending technologies such as Cloud Computing and Software Defined Networking (SDN). In the case of Cloud Computing, NFV can easily benefit from all the research carried out on virtualization management [5]. Furthermore, NFV is to reside inside Cloud providers' networks in order to carry out all the network-related management tasks. On the other hand, NFV is complementary to SDN but not dependent on it and vice-versa [6]. However, NFV enhances SDN as it provides the infrastructure on top of which SDN software can run.

However, in order to make the most of NFV, mechanisms that allow obtaining maximum network processing throughput in such virtual environments must be developed. In a bare-metal scenario, researchers have recently focused on developing high-performance packet capture engines [3]. This work evaluates the feasibility of applying such capture engines in NFV-based environments for the primal network task: packet capture. Specifically, we evaluate different virtualization alternatives, namely PCI-passthrough and Network Virtual Functions (NVF) available on contemporary systems. For each capture engine and environment, we measure the performance bounds, so researchers and practitioners may benefit from our results in order to build their high-performance NFV-based applications.

II. Network processing on bare-metal scenarios

In the recent years both the research community and industry have paid attention to the use of off-the-shelf for network processing purposes [7]. Those systems offer interesting features that allow reducing CAPEX and OPEX when building a system on top of them. In terms of network processing, off-the-shelf systems have traditionally relied on the use of the corresponding NIC vendor's driver plus a standardized network stack. Such approach is characterized by a great degree of flexibility, as the network stack provides independent layers that allow distributing the traffic to the corresponding final applications. Nevertheless, the performance obtained by such approaches is poor: every incoming packet must traverse a set of layers, which is translated into additional copies, resource re-allocations at processing time. Thus, this flexibility the standard solution offers limits its applicability in high-speed scenarios.

Consequently, if off-the-shelf systems are to be applied in high-speed networked scenarios, they have to be carefully planned and tuned. In this light is how high-performance packet capture engines were born [3]. Solutions such as PF_RING, PacketShader, netmap, PFQ, Intel DPDK or HPCAP were created as high-performance counterparts for the traditional network driver-plus-stack alternative. All those solution are based on some of the following ideas (see [3], [8] for a detailed discussion):

- *Pre-allocation and re-use of memory*: traditional solutions allocate a set of structures and buffers for each received packet. Those resources are also released once the packet is delivered to upper layers. Such technique is a very demanding task and may be optimized by pre-allocating pools of resources for re-using them along time.
- *Memory mapping*: this technique makes it possible for high-level applications to map buffers and data structures allocated at driver-level. Consequently, the number of copies experimented by incoming packets is reduced.
- *Use of parallel direct paths*: modern NICs support have multiple parallel reception queues and to distribute incoming traffic among such queues using RSS (Receive Side Scaling) mechanisms. However, the use of contemporary network stacks becomes a bottleneck serializing all the traffic at one single point for delivering it to upper layers. In this light, high-performance network solution must avoid serialization point for really exploiting NIC's parallel nature. Note that the use of parallel paths may cause incoming packet reordering [9], so it must be carefully planned.
- *Batch processing*: traditional end-user network applications retrieves packets individually by means of system calls, which in Linux systems involves at least two context switches. In order to mitigate this effect, some capture engines pretend processing several packets with a single system call. Solutions such as PacketShader, netmap or Intel DPDK group packets into batches, and so all packets conforming the batch are handled in the same system call. Note that applying such technique, in spite of its performance improve-

ment, may entail side effects such as latency increase and inaccurate timestamping [10]. For this reason, solutions such as HPCAP propose a byte-stream oriented approach.

- *Prefetching*: this technique consists in preloading memory locations in processors' caches in a predictive way so that it can be quickly accessed in a near future. Its application reduces the number of cache misses in capture process and thus leverages performance.
- *Affinity planning*: contemporary server feature NUMA architectures, where applications performance is highly influenced by the NUMA node or processor it is executed on. Thus, the affinity of all threads and processes involved in a capture system must be carefully scheduled. Such schedule must also take into account the connectivity of the processors to the PCI slot the NICs are connected to.

All the above-mentioned solutions enable network managers to deploy high-performance network applications on top of them. Table I shows the performance level obtained for packet capture in a full-saturated 10 Gb/s link for the worst-case scenario (64 byte packets) and an average scenario (a CAIDA from a ISP's backbone link between Chicago and Seattle obtained the 19^{th} June 2014, with an average packet size of 965 bytes [11]). We have compared the performance of the standard `ixgbe` plus network stack solution compared to PF_RING, Intel DPDK and HPCAP. We chose PF_RING as an archetype for the previously mentioned capture engines, and Intel DPDK and HPCAP for being the ones supporting virtual environments. Importantly, HPCAP has been developed by the authors as capture engine focused not only on high-performance packet capture rates, but also on accurate packet timestamping [10] and on building a multi-granular multi-purpose monitoring framework [12].

The tests have been carried out using a server with two Xeon E5-2630 processors running at 2.6 Ghz and 32 GB of DDR3 RAM. The NIC used was an Intel 82599 card connected to a PCIe Gen3 slot. The operating system in use is a Linux Fedora 20 with a 3.14.7 kernel. The results obtained show that all four capture engines are capable of capturing 100% of the packets when replying the CAIDA trace at top-speed. In the worst-case scenario, `ixgbe` captures only 2.7% of the incoming packets, while PF_RING and Intel DPDK capture 100% of them, and HPCAP captures 97.9% of the packets. Note that the performance degradation experienced by HPCAP is due to the driver timestamping each incoming packet.

Nevertheless, the application of the existing packet capture engines has been limited in the literature to the bare-metal case. That is, a physical server to which the NIC is connected through a PCIe slot.

Configuration	Bare-metal		PCI pass-through	
Traffic	64 byte	CAIDA	64 byte	CAIDA
ixgbe	2.7 %	100 %	1.9 %	62.7 %
PF_RING	100 %	100 %	100 %	100 %
DPDK	100 %	100 %	100 %	100 %
HPCAP	97.9 %	100 %	82.5 %	100 %

TABLE I: Percentage of packets captured for different traffic patterns in a fully-saturated 10 Gb/s link obtained by different solutions in a bare-metal and PCI pass-through configuration

This configuration corresponds to the leftmost NIC shown on Fig. 1. Consequently, the application of the high-performance packet capture engines has not been evaluated in NFV environments yet.

III. Network processing in virtualized environments

When using virtual machines (VMs) for computing intensive applications, the performance obtained by the target application is usually damaged. For this reason, if we desire to obtain maximum performance, the creation and schedule of each VM must be carefully made. For example, the amount of cores of the VM should be such that allows the VM to be executed inside a single NUMA node in the physical server, and those virtual cores should be configured to be mapped to independent physical cores. Another determinant factor for VM's performance is optimizing how the VM accesses the physical system's memory. Contemporary VM managers allow attaching the VM's memory to certain NUMA node, which will increase overall memory access latency. However, studies such as [13] point out the importance of using Linux's huge pages for allocating the VM's memory chunk so the amount of page misses is reduced and performance is enhanced.

By using Linux huge pages both the packet capture engines and the network applications built on top of them running on a VM would experience a performance increase. To confirm this effect, we empirically tested the impact on the percentage of captured packets when using different capture approaches from VMs running over huge pages and not (those experiments were made using the PCI pass-through technique, see subsection III-B). We experienced a slight performance improvement when running them on top of huge pages. Note that those were simple experiments in which only packet capture were made, but by using Linux huge pages network applications built on top of those VMs would also see their performance boosted regardless the capture procedure used. All the VMs used along the performance experiments presented in this paper were created taking these facts into account. Regarding the operating system running inside the VMs, we used the same version as in the physical server, a Linux Fedora 20 with a 3.14.7 kernel. The choice of the operating systems to be used

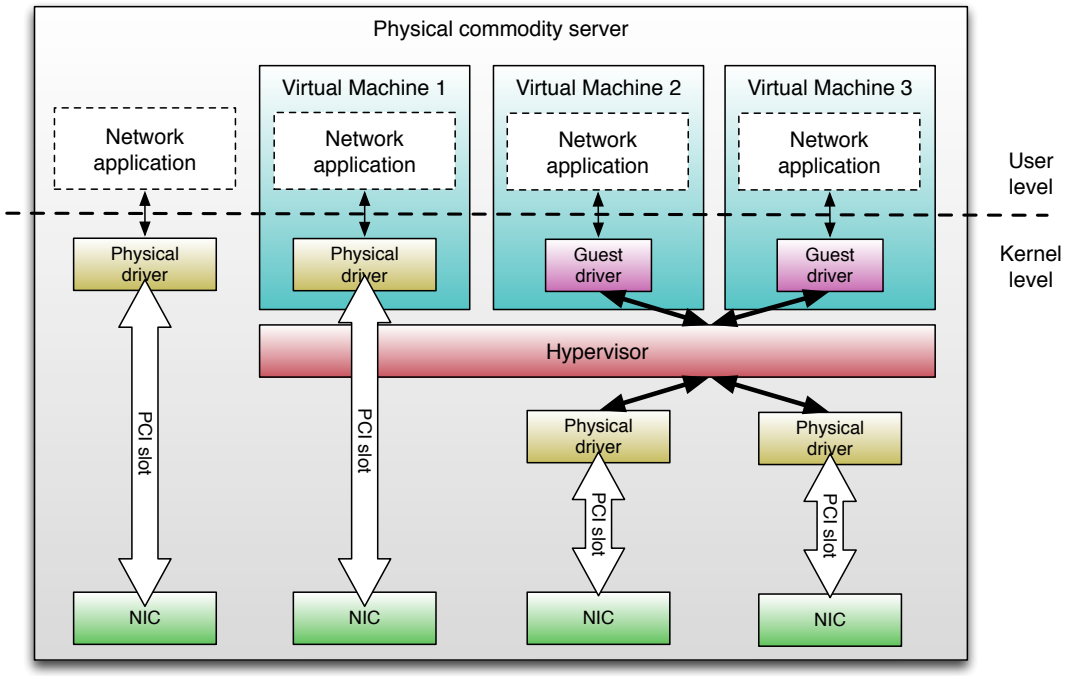

Fig. 1: Using a network device in bare-metal (leftmost), PCI pass-through (VM1) and emulated NIC (VM2, VM3)

both in the physical and virtual machines is relevant, as not all combinations support the use of Linux huge pages both in the physical server for creating the VM as mentioned and inside the VM. Note that some of the capture engines used such as PF_RING and Intel DPDK make use of those huge pages, so this requirement is nontrivial.

A. *VMs with emulated NICs*

In order to enable network applications running on top of NFV environments, we must understand the way a network device can be connected to a virtual machine (VM). The traditional scenario is composed by a set of virtual machines where an emulated network device is instantiated. In this configuration, incoming packets are captured by the NIC driver on the physical machine and traverse the system's network stack, and then delivered to the hypervisor's network module. Once acquired by the hypervisor, packets are delivered to the target VM depending on the virtual network configuration. This scenario is the one exhibited by VMs 2 and 3 on Fig. 1. Note that in this case the network driver used in the VM would be the one corresponding to the emulated device, and not the physical device that lies below. Such configuration implies at least two additional copies: from the physical server to the hypervisor and from the hypervisor to the virtual machine, which obviously degrades the performance achieved by the capture system.

Furthermore, the performance degradation experienced by network applications running in this configuration has pushed academia and industry to tune and optimize the hypervisor's packet handling policy. In this line, authors of [14] introduce VALE, which proposes a set of modification both on physical drivers and hypervisors in order to improve the network processing performance. By applying their proposals they leverage the system throughput: from 300 Mb/s using the conventional approach to nearly 1 Gb/s using VALE in the worst-case scenario (64 byte UDP packets); and from about 2.7 Gb/s to 4.4 Gb/s for TCP traffic. Although the results obtained by this approach are promising, the authors focus their attention on data exchange between VMs on the same physical server.

Note that both the traditional approach and VALE are base on the hypervisor as the central communication element. Consequently, the hypervisor becomes the bottleneck and a single point-of-failure for network processing tasks, limiting their applicability for network monitoring purposes. The use of an emulated NIC approach forces incoming packets to be processed twice: once when they arrive to the hypervisor, and again when they are transferred to their corresponding VM.

B. *VMs and PCI pass-through*

As an alternative to a hypervisor-centric approach, different hardware manufacturers developed a set of mechanism which enable direct connectivity from the

PCI adapter, referred as physical function, to the virtual machines providing near-native performance. The name given to those mechanisms depends of the underlying manufacturer: VT-d for Intel and ARM, Vi for AMD, but the technique is usually referred as PCI pass-through. This technique has been applied in several computational scenarios [15]. Applied to the network capture problem, this technology allows the VMs to map physical specific PCIe memory regions. Using this feature, VMs can operate as if they had the NIC physically connected to them, as shown by VM1 on Fig. 1. This implies that the drivers managing the NICs in the VMs are the same used to manage those NICs in the bare-metal scenario. Consequently, this allows network applications being executed in virtual machines to benefit from the high-performance packet capture solutions developed for bare-metal scenarios.

Table I shows the performance results in a fully-saturated link for the default `ixgbe` driver, PF_RING, Intel DPDK and HPCAP when executed in a VM with PCI pass-through compared to the previously mentioned bare-metal scenario. Those tests were made using the same hardware as previously mentioned and using KVM for creating and managing the VMs, and accordingly to the bare-metal scenario, the table depicts each capture engine's performance for the worst-case scenario and in an average scenario. The amount of packets captured by `ixgbe` falls from 2.7% under the bare-metal configuration to 1.9% using PCI pass-through in the worst-case scenario, and from 100% to 37.3% when replaying the previously mentioned CAIDA trace. On the other hand, PF_RING and Intel DPDK show no performance degradation when used via PCI pass-through, as they capture 100% of the packets on all scenarios. When it comes to HPCAP, packet capture performance is damaged when using PCI pass-through in the worst-case scenario, in which the amount of packets captured falls from 97.9% to 85.2%. The performance penalty experienced when introducing PCI pass-though can be blamed on the execution of the capture system in a virtual machine.

However, using PCI pass-through for instantiating network applications in different VMs has an inherent constraint: only one VM can make use of each network interface. That means that the scalability problem of instantiating more VMs for network processing purposes must be solved by adding additional NICs and probably more physical servers, as the amount of PCIe slots a server has is limited.

C. VMs attached to virtual functions

With the goal of promoting virtualization performance and interoperability the PCI Special Interest Group developed a series of standards, which they called Single-Root I/O Virtualization (SR-IOV). Those standards define the concept of virtual function (VF) as a lightweight image of the underlying physical PCI resource. Modern NIC such as the Intel 82599 use the concept of Virtual Machine Device Queues

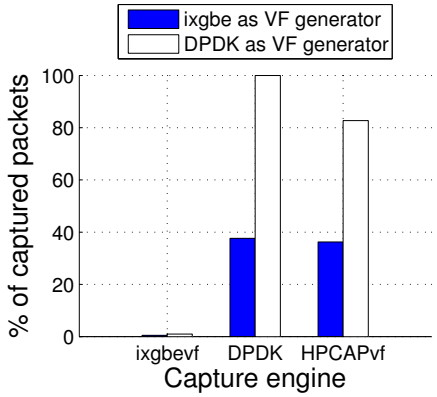

Fig. 2: Packet capture performance obtained when capturing from a 10 Gb/s link fully-saturated with 64-byte packets using virtualized alternatives for different virtual-function generators

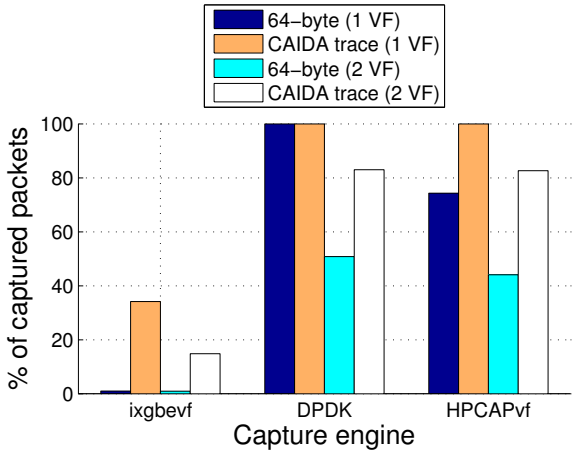

Fig. 3: Packet capture performance obtained by different virtualized alternatives when capturing different traffic in a fully-saturated 10 Gb/s link

(VMDq) to refere to the NIC's virtual functions. By using VMDqs, the traffic arriving to the physical network device can be distributed among a set of queues based on a set rules that can be configured at hardware. Each of those queues is attached to a virtual PCI device, or virtual function (VF), that can be mapped via PCI pass-through by a certain VM. This configuration is represented in Fig. 4. Note that this configuration allows connecting an arbitrary number of VMs to a single physical device. The amount of virtual functions generated per physical device is limited by the hardware device, being 32 for the Intel 82599 adapter.

It is worth remarking that if this approach is used, only VF-aware drivers can be used in the VMs, so they handle the peculiarities those devices have. This requirement limits the amount of capture engines avail-

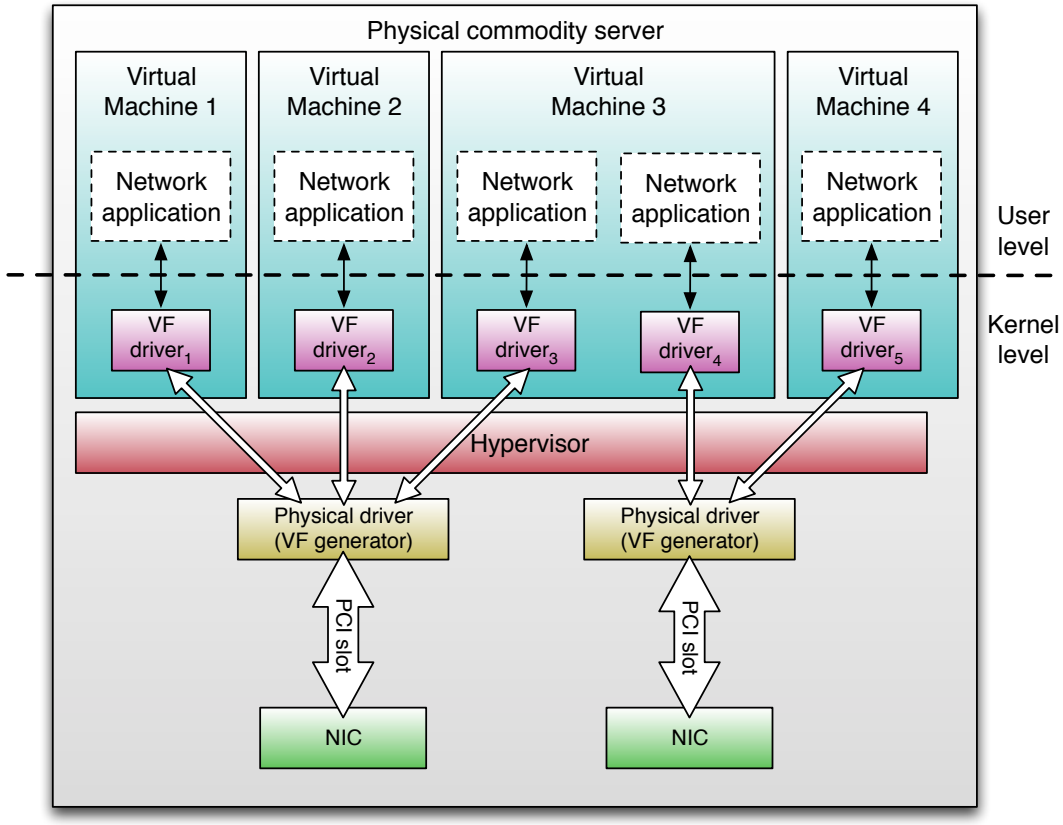

Fig. 4: Using a network device in a VM via virtual network functions

able. Specifically, Intel's DPDK has native support for working with VF, and they also supply a VF-aware counterpart to the `ixgbe` driver, named `ixgbevf`. Additionally, we developed a VF-aware version of HPCAP, that we named HPCAPvf, following all the design principles that guided HPCAP's design. Those three drivers have consequently been the only ones we have been able to test under this configuration.

On the other hand, the task of creating and managing the mentioned VF belong to the driver used in the physical server for managing the physical NIC. Above all the drivers previously mentioned capable of managing a physical NIC, only Intel's offer this feature. Consequently, when using VF only `ixgbe` and Intel DPDK are eligible. Importantly, the choice among those two drivers for generating the VF has an impact on the performance obtained by possible network applications running inside the VMs. Fig. 2 shows the effect of such choice when using different VF-aware for packet capturing in the worst-case scenario, that is a fully-saturated 10 Gb/s with 64-byte packets. Results show that using DPDK as VF generator improves the packet capture performance obtained from the VM side compared to the performance when using `ixgbe` for generating those VF. Specifically, when capturing packets in a VM using the `ixgbevf` driver using DPDK as VF generator raises the amount of packets captured from 0.5% to 1%. In a similar manner, when using DPDK and HPCAPvf in the VM side with `ixgbe` as VF generator, only 37.6% and 36.3% are respectively captured, but those ciphers are increased to 100% and 82.7% respectively when DPDK is used as VF generator.

When instantiating several VF through a single physical interface, the default traffic distribution policy is based on the MAC and IP addresses of each VM's interface the VF are connected to. Thus, each VF would only receive the traffic targeted to its corresponding VM. This would limit the use of this VF approach in scenarios where different network applications running on independent VMs need to be fed the same input traffic, which is the desirable case for scalable network monitoring. However, NICs such as Intel's 82599 supply a Mirroring and Multicast Promiscuous Enable (MPE) flags, which can be activated for each VF. By enabling those options, any VF could receive all the traffic traversing the physical device, or the traffic corresponding to a different VM, regardless it is targeted to its VM or not. Note that enabling those features in the Intel 82599 NIC implies a hardware-level packet replication, which minimizes the impact on the capture process' performance. De-

pending on the physical driver used to generate the NIC's VFs, activating the MPE may be done by tuning the driver's source code (that is the case when using `ixgbe` for generating the VFs) or by using a user-level application giving access to the NIC's registers (such as the `testpmd` application offered by Intel's DPDK).

Obviously, if several VMDqs are to be fed the same incoming packets, the physical driver will have to issue additional copies for each additional VMDq, and packet capture performance may be degraded. This effect is shown in the yellow and red bars of Fig. 3: adding a second VF to each physical device reduces the overall packet capture throughput obtained. When using `ixgbe` the amount of packets captured is 10% when using either one or two VFs in the worst-case scenario, but falls from 34.2% with one VF to 14.8% with two when replaying the CAIDA trace. Intel DPDK also suffer performance loss as it is capable of capturing all of the packets in both scenarios when only one VF is instantiated, but it captures 50.8% of the 64-byte packets and 83.0% of the CAIDA ones when two VFs are used. Finally, HPCAP's performance falls in both cases: from 74.3% to 44.1% in the worst case and from 100% to 82.7% for the CAIDA trace.

IV. Application scenarios

In the light of the discussion and results presented along section III, we strongly recommend the usage of the VF-based approach for processing network-data. However, if there is a primal need for performance, users may find useful to connect their NICs and their VMs via PCI pass-through. We discourage the use of an emulated NIC configuration, as this would imply unnecessary redundant computation and limits the capture performance being the hypervisor the system's bottleneck. By using the VF approach, not only a reasonably high performance is achieved but also the traffic targeted to each VM is isolated and we acquire the ability of redirecting which may be interesting in a number of scenarios. Note that the migration process from a traditional (emulated NIC) configuration to any of the other two approaches only implies driver modifications in the host server and changing the VMs' default network interface for the new one (be it a physical one via pass-through or a virtual one).

Finally, we have identified three different usage scenarios in which the VF-based alternative would apply, which are also depicted in Fig. 5:

1) Users may run on their VM one of the network-processing applications provided by Intel DPDK or HPCAP, as in the N^{th} VM in Fig. 5.
2) Users may create their own network processing application based on the existing APIs, just as in VM3 in Fig. 5.
3) Users already running a set of VMs with a set of legacy network-related applications needing to monitor the traffic directed to those legacy applications. In this scenario a new VM could be created, adding the proper HW rules for re-directing the desired traffic to the new VF. This new VM could use a high-performance VF-aware driver with a monitoring application. This is depicted by VMs 1,2 and N in Fig. 5. The dashed lines are used to remark that this new VM could be dynamically created or destroyed without affecting the rest of VMs already running in the same host server. Importantly, this scenario would not require any changes in the legacy network applications.

V. Conclusions

The results obtained along the experiments presented in this work assess the feasibility of migrating the usage of high-performance packet capture engines in virtualized environments. We have discussed the different configurations by which a network application can be used inside a virtual machine, and obtained the performance bounds for each of those configurations. Moreover, we have given a set of guidelines that allow exploiting the functionality of generating virtual network functions, enabling a set of interesting scenarios. Differently from PCI pass-through, the use of VF allows end-users to scale in the amount of network applications running on a single hardware, with the consequent saving in terms of space, cooling and power consumption.

We have presented a set of solutions available for use under GNU Linux and, in the case of HPCAP and Intel DPDK, accessible as free software. As an additional contribution, we have developed HPCAPvf, a VF-aware version of HPCAP, offering a set of interesting features and capabilities. HPCAPvf may be used in any Linux distribution with kernel version newer than or equal to 2.6.32 without modifying nor kernel nor hardware configuration. Furthermore, we have made available the source code of HPCAPvf under a GPL license[1]. This study paves the way for future works involving advanced network actions in the VM side, such as packet storage, at high-speed rates.

Acknowledgements

This work has been carried out with the support of the Spanish National I+D *Packtrack* project (TEC2012-33754) and Universidad Autonoma de Madrid's multidisciplinary project *Implementacion de Modelos Computacionales Masivamente Paralelos* (CEMU-2013-14).

References

[1] F. Yu, R.H. Katz, and T.V. Lakshman. Gigabit rate packet pattern-matching using TCAM. In *Proceedings of IEEE Conference on Network Protocols*, 2004.

[1] https://github.com/hpcn-uam/HPCAP

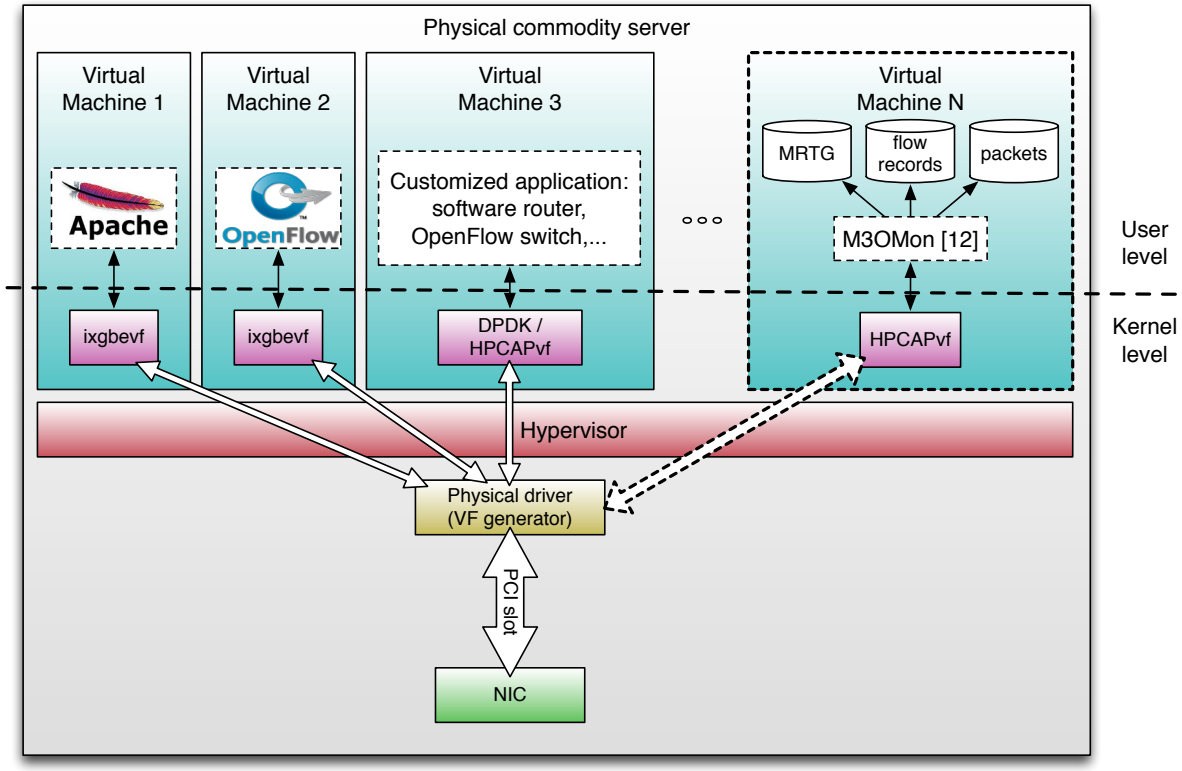

Fig. 5: Example usage cases

[2] M. Forconesi, G. Sutter, S. Lopez-Buedo, J.E. Lopez de Vergara, and J. Aracil. Bridging the gap between hardware and software open-source network developments. *IEEE Network*, 28(5), 2014.

[3] J.L. García-Dorado, F. Mata, J. Ramos, P.M. Santiago del Río, V. Moreno, and J. Aracil. High-performance network traffic processing systems using commodity hardware. In *Data Traffic Monitoring and Analysis*, volume 7754 of *Lecture Notes in Computer Science*, pages 3–27. Springer Berlin Heidelberg, 2013.

[4] AJ. Younge, R. Henschel, J.T. Brown, G. von Laszewski, J. Qiu, and G.C. Fox. Analysis of virtualization technologies for high performance computing environments. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 9–16, July 2011.

[5] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[6] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing software-defined networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 1–14, Berkeley, CA, USA, 2013. USENIX Association.

[7] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle. Comparing and improving current packet capturing solutions based on commodity hardware. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 206–217, New York, NY, USA, 2010. ACM.

[8] Intel. Intel Data Plane Development Kit (Intel DPDK) Release Notes, 2014. http://www.intel.com/content/dam/www/public/us/en/documents/release-notes/intel-dpdk-release-notes.pdf, [1 October 2014].

[9] W. Wenji, P. DeMar, and M. Crawford. Why can some advanced Ethernet NICs cause packet reordering? *IEEE Communications Letters*, 15(2):253–255, 2011.

[10] V. Moreno, P.M. Santiago del Rio, J. Ramos, J. Garnica, and J.L. Garcia-Dorado. Batch to the Future: Analyzing Timestamp Accuracy of High-Performance Packet I/O Engines. *Communications Letters, IEEE*, 16(11):1888 –1891, november 2012.

[11] C. Walsworth, E. Aben, k.c. Claffy, and D. Andersen. The CAIDA anonymized Internet traces 2014 dataset. http://www.caida.org/data/passive/passive_2014_dataset.xml, [1 October 2014].

[12] V. Moreno, P. M. Santiago del Río, J. Ramos, D. Muelas, J. L. García-Dorado, F. J. Gomez-Arribas, and J. Aracil. Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems. *International Journal of Network Management*, 24(4):221–234, 2014.

[13] A.O. Kudryavtsev, V.K. Koshelev, and A.I. Avetisyan. Prospects for virtualization of high-performance x64 systems. *Programming and Computer Software*, 39(6):285–294, 2013.

[14] L. Rizzo, G. Lettieri, and V. Maffione. Speeding up packet i/o in virtual machines. In *Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '13, pages 47–58, Piscataway, NJ, USA, 2013. IEEE Press.

[15] C.T. Yang, J.C. Liu, H.Y. Wang, and C.H. Hsu. Implementation of gpu virtualization using pci pass-through mechanism. *The Journal of Supercomputing*, 68(1):183–213, 2014.