UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

Creación de registros NetFlow usando la plataforma NetFPGA

Álvaro García Salinas

Septiembre 2012

Creación de registros NetFlow usando la plataforma NetFPGA

AUTOR: Álvaro García Salinas

TUTOR: Víctor Moreno Martínez

High Performance Computing and Networking Group

Dpto. de Tecnología Electrónica y de las Comunicaciones

Escuela Politécnica Superior

UNIVERSIDAD AUTÓNOMA DE MADRID

Septiembre 2012

Álvaro García Salinas
Creación de registros NetFlow usando la plataforma NetFPGA

Álvaro García Salinas
Escuela Politécnica Superior. High Performance Computing and Networking Group

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Índice

AGRADECIMIENTOS

En primer lugar quería agradecerle a los dos tutores que he tenido toda la ayuda que me han prestado. A Víctor Moreno que aún cuando no sabía que iba a ser mi tutor ya estaba echándo horas conmigo, que siempre sacaba tiempo para atenderme por muy liado que estuviese y, sobre todo, porque siempre me iluminaba en esas horas oscuras en las que me planteaba ¿por qué habré elegido un proyecto hardware? A Víctor López, porque sus clases me hicieron decidirme por este proyecto, por hacerlo a mi medida, con los ingredientes que le había pedido (un poco de redes, un poco de hardware...) y porque también me ha rescatado en más de una ocasión. Es un placer trabajar con vosotros.

A todos mis compañeros del HPCN, que desde un primer momento me hicieron sentirme agusto y con ganas de ir a trabajar, con quien he vivido buenas experiencias dentro y fuera de la universidad. Especialmente a Jose Luis, David Muelas, David Madrigal, Jaime, Javier, Miguel y Pedro Santiago.

A Eloy Anguiano, autor del estilo LaTeX de este documento, por todas las consultas, la gran cantiad de mails intercambiados y, mucho antes de esto, por descubrirme el mundo del software libre.

A todos mis compañeros y amigos de la universidad, porque sin vosotros esto no hubiera merecido la pena. Sandra, por las innumerables horas de biblioteca y pasillos, por tantas penas y alegrías compartidas; Ana, porque discutiendo se entiende la gente; Pedro (claro que eres tú, ¿no ves que no estás en el HPCN?), por las ganas de escuchar y por todos los ratos de cafetería; Diego, por las experiencias vividas (en Oropesa y en Madrid); Joe, porque disfruto trabajando contigo; Patri, por tener siempre una sonrisa capaz de alegrarme el día; Miriam, aunque al principio me costase adaptarme un poco a ti, al final no te cambiaría, y porque contigo es imposible aburrirse; Pablo, por todas las conversaciones, charlas, cotilleos y cafetitos; Carmen, por meterme el gusanillo de LaTeX, lástima haber coincidido tan poco tiempo; Eva, por ser como eres; Marco, por las horas de trabajo compartido y por revolucionar las comidas; Stephan, por confiar en mi; Pilar, por una de las prácticas que más he disfrutado en la carrera; Javi por elegirme, enseñarme y motivarme; a todas las psicolocas (Ana, Loli, Laura, Paola, Vicky...), porque le dais color a la vida; y a todos los demás (no pocos) que me dejo en el tintero...

A Diego, Jose y Álex. Os conocí al principo de la carrera, a pesar de que antes ya veía a diario a alguno. Empezásteis como compañeros de prácticas y ahora sois grandes amigos. Con vosotros he vivido muy buenos momentos: de trabajo duro, de ser detallista, de

creatividad, de estrés (p.e. apagando ordenadores), de comernos el mundo, de cachondeo, de seriedad, de confidencialidad. Gracias por todos los proyectos que quedan atrás, por todos los que continúan y por todos los que están por delante.

A toda la gente que ha puesto su granito de arena en lo que soy, o directamente un cubo. Aunque os dedicaría un párrafo a cada uno no tengo demasiado espacio (y seguramente me pondría demasiado empalagoso) así que sencillamente gracias a todos: Rodrigo, Carlos, Sandra, Jose, Tomás, Miguel, Sergio, Isabel, Ana, Mariano, David de Bedoya, Amador, Ana y a todos los que sois importantes para mi...

A mi chica, por todas las horas que le ha dedicado a este proyecto (incluyendo las extras), ayudándome con la falta de sujetos, predicados, frases desordenadas e incoherentes (por ayudarme a darle sentido a frases como *"también entendiendo que uniendo incógnitas equilibramos resultados obtenidos"*). Por enseñarme nuevos puntos de vista, por hacer que el mundo sea un poco menos cuadriculado, por todo lo que hemos aprendido juntos, por todo lo que disfrutamos y por hacerme reír.

A mi familia, a la que tanto aprecio. No todo el mundo puede decir que tengo lo que yo tengo con vosotros. A todos mis tíos y primos dispersos desde Granada a Santander, pasando por Las Vegas, Talavera y Madrid (incluidos San Chinarro y Alcorcón). Especialmente a mis abuelos, con los que tanto he disfrutado y disfruto desde que era pequeño.

A mis padres. Los últimos en aparecer en estos agradecimientos, pero sin duda los más importantes. A vosotros os dedico este proyecto. Gracias por el apoyo que me habéis dado en estos años de carrera, y durante toda mi vida. Por alegraros con mis triunfos, sufrir con mis fracasos, por la infinita paciencia en los momentos de ansiedad y desesperación. A vosotros os debo lo que soy, gracias.

Abstract

Network traffic monitoring is an activity of paramount importance. It has proven useful in a number of management tasks, such as the performance evaluation of networks, the detection of anomalies and denial of service attacks (DoS), even the generation of the clients' invoices. Flow monitoring has become a widely mechanisms to measure the network behavior.

On the other hand, NetFPGA project has developed a framework which facilitates the usage of Network applications. This framework includes low cost NetFPGA card which explodes FPGAs goodness (speed, versatility, etc.).

This project covers the implementation and evaluation of a Netflow measurement system which applies sampling techniques like deterministic sampling or distributed sampling.

Key words:

NetFPGA, FPGA, network measurement, NetFlow, flow, packet sampling, deterministic sampling, distributed sampling, traffic monitoring.

Resumen

La monitorización del tráfico se ha convertido en una actividad de gran importancia. Su utilidad ha sido demostrada en multitud de tareas tales como la evaluación del rendimiento de la red, la detección de anomalías y ataques de denegación de servicio (DoS), e incluso en el proceso de tarificación. Actualmente una gran cantidad de sistemas de monitorización se basan en la extracción de registros NetFlow.

Por otra parte, el proyecto NetFPGA ha creado un *framework* para facilitar el uso de aplicaciones de red. Dicho *framework* incluye una tarjeta de bajo coste que explota las bondades de las FPGAs (velocidad, versatilidad,etc.).

En este proyecto se ha implementado y evaluado un sistema de monitorización NetFlow en la NetFPGA que aplica la técnica de muestreo determinista de paquetes así como la agregación de datos obtenidos en varios puntos de análisis distribuidos.

Palabras clave:

NetFPGA, FPGA, monitorización de red, NetFlow, flujo, muestreo de paquetes, muestreo determinista, muestreo distribuido.

1 Introducción

1.1 Motivación

La monitorización del tráfico se ha convertido en una herramienta de gran importancia para los proveedores de servicios de Internet (ISPs), así como un tema de estudio para la comunidad universitaria. Su utilidad ha sido demostrada en multitud de tareas tales como la evaluación del rendimiento de la red [1], la detección de anomalías y ataques de denegación de servicio [2], e incluso en el proceso de tarificación a los clientes [3].

La mayoría de los sistemas de monitorización actuales se basan en la extracción y compresión de la información de muchos paquetes en registros NetFlow o flujos [4]. Así, un flujo es una conexión de datos entre dos direcciones y puertos con un protocolo determinado.

No obstante, el incremento de la velocidad de transmisión de las redes actuales hace una tarea muy costosa el análisis paquete a paquete. Por esta razón se hace necesario el empleo de técnicas de muestreo de paquetes [5], en las que sólo se tiene en cuenta un subconjunto de paquetes del total del tráfico para la generación de los registros NetFlow.

Por otro lado, se hace desable el empleo de dispositivos de bajo coste que sean capaces de implementar los sistemas de monitorización NetFlow. Estos sistemas permiten una reducción de la inversión necesaria (Capex), tanto por la comunidad universitaria como por el ámbito empresarial.

La comunidad NetFPGA ha creado un *framework* para facilitar el desarrollo de aplicaciones de red usando FPGAs. Dicho proyecto permite la construcción de prototipos de alta velocidad, acelerados por hardware, a un bajo coste [6]. Además, si el sistema se encuentra en producción, permite la reconfiguración de la FPGA en pocos segundos, siendo posible introducir corrección de errores, mejoras o nuevas funcionalidades a nivel hardware.

El uso de la plataforma NetFPGA para el análisis del tráfico y la creación de registros NetFlow permite explotar las ventajas de ambos. Por ello, la motivación de este Proyecto Fin de Carrera es la implementación, evaluación y mejora de un sistema que permita obtener estadísticas de flujos mediante mecanismos de muestreo usando NetFPGA.

1.2 Objetivos

El objetivo de este Proyecto Fin de Carrera es la implementación y evaluación de la técnica de muestreo determinista usando la plataforma NetFPGA.

Se partirá de un proyecto disponible, creado por Martin Zadnik, que es capaz de analizar el tráfico y obtener estadísticas NetFlow, y se mejorará dicho proyecto para permitir el uso de técnicas de muestreo.

En concreto se trabajará para conseguir los siguientes objetivos:

- **Instalación y puesta en marcha de la plataforma NetFPGA de 1G:** se estudiará el entorno de trabajo de NetFPGA y se realizará la instalación y puesta en marcha del mismo en una máquina de la U.A.M. Se pretende tener un equipo con la NetFPGA operativa así como la documentación necesaria que permita repetir el trabajo en otras máquinas.
- **Instalación y validación del módulo NetFlow:** se instalará el módulo creado por Martin Zadnik y se realizarán una serie de pruebas para verificar su correcto funcionamiento. Se documentará todo el proceso así como los problemas tratados y las soluciones dadas.
- **Implementación del muestreo sobre el módulo NetFlow:** se modificará el módulo NetFlow de forma que sea posible aplicar muestreo en el proceso de extracción de los flujos. Dicho muestreo deberá ser reconfigurable de forma rápida y sencilla.
- **Validación y análisis de los resultados obtenidos:** se realizarán una serie de pruebas con trazas experimentales con el fin de verificar que el muestreo se realiza de forma correcta.
- **Aplicación y evaluación de técnicas de muestreo distribuido:** se estudiará el uso de técnicas de muestreo distribuido con el fin de obtener una mejora del error introducido por el proceso de muestreo.

1.3 Organización de la memoria

La memoria se divide en cinco capítulos:

Introducción: es el capítulo actual, en él se expone la motivación del proyecto, los objetivos que se pretenden alcanzar y la organización de la memoria.

Estado del arte: en este capítulo se hace una descripción de NetFlow, se explican sus conceptos fundamentales, nomenclatura, aplicaciones, y funcionamiento. A continuación se presentan dos técnicas de muestreo. Por último se describe la plataforma NetFPGA, las diferentes aplicaciones de la misma y se hace una introducción al módulo NetFlow de dicha plataforma.

Desarrollo: el tercer capítulo tiene dos partes.

En la primera se profundiza en la plataforma NetFPGA explicando de forma detallada su funcionamiento, cómo se estructura un proyecto y las herramientas de trabajo proporcionadas a los desarrolladores.

La segunda parte del capítulo trata sobre el módulo NetFlow. También se divide en dos partes. Primero se hace una descripción general del módulo NetFlow con el fin de dar al lector una idea general de su funcionamiento. Posteriormente se hace un estudio más profundo de cada uno de los submódulos que lo conforman y se describe el proceso de implementación del muestreo.

Pruebas y resultados: el capítulo se divide en tres partes.

Primero se exponen los problemas encontrados durante la instalación del proyecto así como las soluciones aportadas.

A continuación se realizan una serie de pruebas a la NetFPGA con el fin de evaluar tanto el funcionamiento del módulo NetFlow como del muestreo implementado.

Por último se realiza un experimento usando muestreo distribuido.

Conclusiones y trabajo futuro: el último capítulo hace un resumen del proyecto y muestra las conclusiones extraídas. Además se proponen nuevas líneas de trabajo que pueden realizarse a partir de las mismas.

2

Estado del arte

2.1 NetFlow

NetFlow es un protocolo desarrollado por Cisco que se basa en la extracción de flujos para caracterizar el tráfico IP. En un principio Cisco lo implementó en sus routers como un sistema para reducir la carga de conmutación de los mismos. Posteriormente se descubrió que los flujos eran una buena herramienta de recolección de datos y monitorización de la red. En consecuencia se creó un protocolo que permitía a los routers exportar la información de los flujos [7].

NetFlow facilita soluciones a problemas muy comunes a todos los ISPs [8]:

- Analizar nuevas aplicaciones y su impacto en la red.
- Evaluación de la utilización de los recursos y detección de los *cuellos de botella*.
- Evaluación del impacto de cambios en la red.
- Detección de tráfico no autorizado.
- Detección de anomalías y vulnerabillidades de seguridad.
- Evaluación de la calidad del servicio (QoS).

Un flujo es una secuencia unidireccional de paquetes que comparten las IPs origen y destino, los puertos origen y destino y el protocolo. Se denomina quintupla a este conjunto de campos que establece un flujo de datos entre un punto origen y destino (figura 2.1).

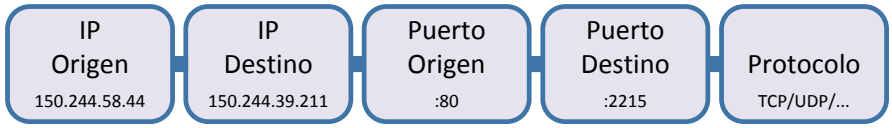

Figura 2.1: Un flujo queda definido por su quintupla.

Un registro NetFlow para un flujo dado almacena, además de la quintupla, el número total de paquetes que forman el flujo, número de bytes de dichos paquetes, los índices de las interfaces de entrada y salida, las marcas de tiempo de inicio y finalización del flujo y el ToS (Type of Service) [9].

Los flujos son generados por un router o un dispositivo de red con capacidad para analizar el tráfico. Cuando llega un paquete, el dispositivo identifica el flujo al que pertenece, consulta una tabla con los flujos activos y, dependiendo de si el flujo está en la tabla, actualiza el registro del flujo o crea uno nuevo (figura 2.2).

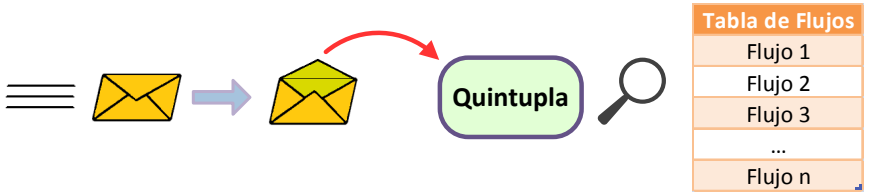

Figura 2.2: Cuando llega un paquete se extrae su quintupla, se busca en la tabla de flujos y se crea un registro nuevo o se actualiza uno existente.

Arquitectura de la red

Una red que usa NetFlow como herramienta de monitorización se compone, principalmente, de tres elemenos (que pueden estar separados o en el mismo equipo). El primero de ellos es el generador de flujos, un router o dispositivo que analiza el tráfico, genera registros NetFlow y los exporta. El segundo es un colector, como su propio nombre indica, colecta los flujos que se envían desde uno o varios generadores y los almacena [9]. Por último, un analizador de flujos accede a los datos almacenados por el colector y extrae las métricas útiles (figura 2.3).

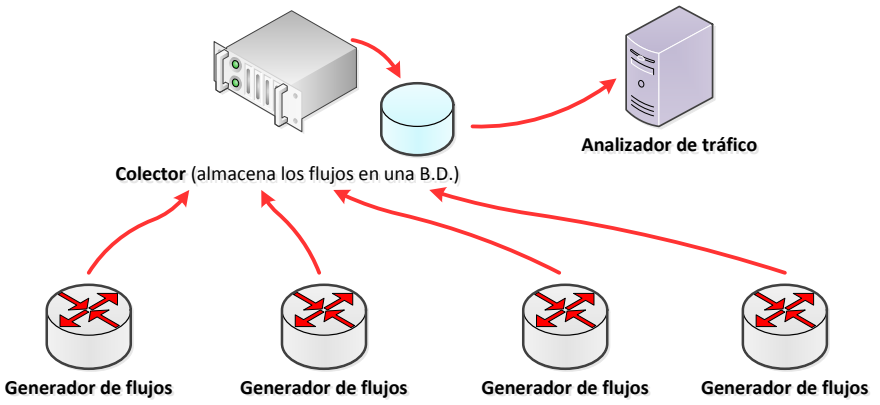

Figura 2.3: Arquitectura de la red.

2.2 Muestreo

La intensidad del tráfico en las redes de hoy en día es tan alta que realizar un análisis de cada paquete resulta una tarea compleja y costosa. Esto se debe a que los equipos comerciales actuales no son capaces de leer un paquete y generar o actualizar los registros NetFlow correspondientes antes de la llegada del siguiente.

Una solución aplicada para este problema, tanto por Cisco como por la comunidad investigadora, es el empleo de técnicas de muestreo, de forma que sólo se capture y analice un porcentaje del tráfico.

El muestreo consiste en capturar y analizar una parte de una secuencia de paquetes recibidos. Existen dos opciones básicas para la implementación del muestreo:

- **Muestreo aleatorio:** se asigna una probabilidad que determina si el paquete que llega será muestreado o descartado [5] (es decir, para cada paquete que entra se "*tira un dado*"). Por ejemplo, si se desea muestrear un paquete de cada diez, la probabilidad de que cada uno de ellos sea capturado es $p = 0.1$. En una secuencia de paquetes numerados por su orden de llegada, los paquetes capturados pueden ser 3, 17, 32, 40, etc.
- **Muestreo determinista:** se fija un intervalo que marca cada cuantos paquetes se seleccionará uno. Aplicado al ejemplo anterior, los paquetes capturados son 1, 11, 21, 31, 41,etc.

En el estudio *"Observations on cisco sampled netflow"* de B.Y. Choi y S. Bhattacharyya (2005) [5] se realiza un análisis teórico y práctico comparando ambos muestreos. Su conclusión es que la precisión de ambos tipos de muestreo es similar, siempre y cuando los paquetes lleguen de forma aleatoria al router, es decir, siempre que no exista periodicidad en los paquetes de cada flujo.

Estos resultados, sumados al hecho de que el muestreo implementado por Cisco en sus routers es determinista [10], ha llevado a la decisión de aplicar este tipo de muestreo en la realización del proyecto.

2.3 NetFPGA

2.3.1 Descripción general

El objetivo del proyecto es la implementación y evaluación de la técnica de muestreo determinista usando la plataforma NetFPGA. La elección de la NetFPGA se debe a que es una plataforma Open Source que permite la construcción de prototipos de alta velocidad, acelerados por hardware, a un bajo coste [6].

El proyecto NetFPGA nace en la Universidad de Standford debido a la necesidad de una herramienta de enseñanza en la asignatura de sistemas de redes. Los profesores del departamento de ingeniería eléctrica se dieron cuenta de que los estudiantes ganaban experiencia en las capas tres y superiores del modelo OSI (capa de red, capa de transporte), pero sus conocimientos en las capas física y de enlace eran más limitados [11].

La NetFPGA original permitía a los estudiantes el diseño de hardware real de red, usando herramientas estándar de la industria como Verilog, y su implementación en una red en condiciones de servicio.

Posteriormente se mejoró su diseño y fue adoptada por la comunidad investigadora como un medio de trabajo. Actualmente existen multitud de proyectos creados por grupos de investigación de todo el mundo (Washington, Pisa, Corea, Brasil, el Cairo) así como por importantes empresas (Ericcson, Xilinx, Microsoft) [12]. Ejemplo de estos proyectos son una tarjeta de red con cuatro interfaces Ethernet [13], proyecto OpenFlow con soporte MPLS [14] o un sistema de monitorización pasiva de alto rendimiento [15].

Existen dos diseños de la NetFPGA, ambos incluyen todos los recursos, lógica, memorias e interfaces Ethernet necesarios para construir un dispositivo de red como un switch, router, dispositivo de seguridad, etc. Ambos diseños soportan conexiones full-duplex y son capaces de procesar cada paquete en pocos ciclos de reloj.

NetFPGA 1G

Incluye una FPGA Xilinx Virtex-II Pro 50, cuatro puertos Gigabit Ethernet de 1 Gbps, memoria SRAM de 4.5 MBytes, memoria DD2R2 DRAM de 64 Mbytes, conector estándar PCI y dos conectores SATA [16]. Es el modelo empleado en este proyecto. En el capítulo 3.1.1 se analizará detalladamente.

NetFPGA 10G

La NetFPGA 10G consta de cuatro interfaces ópticas de 10 Gbps con conectores SFP, tarjeta PCI Express, FPGA Xilinx Virtex-5 TX240T, 27 MBytes de memoria SRAM, 288 MBytes RLDRAM-II y un conector de expansión de banda ancha para tarjetas esclavas [17]. Su lanzamiento ha sido reciente y cuenta con un número menor de proyectos desarrollados.

2.3.2 Módulo NetFlow de la NetFPGA

El módulo NetFlow es uno de los proyectos disponibles para la NetFPGA. Se trata de un módulo que recibe paquetes por las interfaces Ethernet, extrae los flujos de dichos paquetes y los exporta en formato NetFlow a un colector. Por lo tanto, su función es la de generador de flujos NetFlow (apartado 2.1).

Fue desarrollado en la universidad de Brno, República Checa, por Martin Zadnik en los años 2009 y 2010. Es un proyecto Open Source que contiene todo el código en Verilog, scripts de configuración, scripts de inicialización y documentación según lo establecido por la comunidad NetFPGA. Los procesos de medida y exportación de flujos están completamente implementados en la tarjeta NetFPGA, mientras que los procesos de configruación y colección de flujos se ejecutan como software de usuario en el equipo anfitrión [18].

Las características del módulo NetFlow son:

- Capacidad de obtención de medidas en redes de 1 Gbps.
- Medida de flujos IPv4.
- Memoria para 4000 flujos concurrentes.
- TimeStamp con una precisión de milisegundos sincronizado con el equipo anfitrión.
- Exporta registros de flujos usando el prococolo NetFlow v5.

A lo largo del proyecto se realiazará un análisis mucho más detallado del módulo NetFlow así como de sus características, limitaciones y funcionamiento.

3

DESARROLLO

3.1 NetFPGA 1G

La plataforma de desarrollo NetFPGA de 1 Gbps es el sistema hardware elegido para el desarrollo del proyecto. Es una plataforma Open Source de bajo coste que integra una FPGA y cuatro interfaces Ethernet de 1 Gbps conectadas a ella.

La plataforma NetFPGA se compone de la propia tarjeta y de una serie de paquetes denominados NFPs (NetFPGA Packages).

3.1.1 Tarjeta NetFPGA

La NetFPGA implementa todo el *datapath* en hardware. El sistema es capaz de procesar paquetes a tasa de línea de 1 Gbps con una latencia de tan solo unos pocos ciclos de reloj. La tarjeta, mostrada en la figura 3.2, incluye todos los recursos, memoria e interfaces necesarias para construir un switch, router y/o un dispositivo de seguridad, etc.

Los componentes integrados en la tarjeta NetFPGA son:

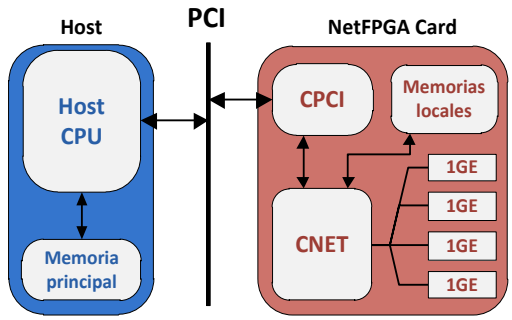

Figura 3.1: La NetFPGA se conecta a un PC mediante el PCI

- Una **tarjeta PCI** estándar que se puede conectar a una ranura PCI de un ordenador de sobremesa u otro dispositivo. Permite reconfigurar la Virtex-II Pro, y por lo tanto

la NetFPGA, directamente desde el PC, sin la necesidad de un cable JTAG. Además proporciona una forma de comunicación entre el PC y la FPGA mediante el acceso a registros y a memoria (Ver figura 3.1).

- Una **Spartan 2**, denominada CPCI (Control del bus PCI). Como su propio nombre indica, es la encargada de controlar la comunicación entre la Virtex II y el bus PCI. Es una FPGA de menor capacidad transparente para el desarrollador ya que, una vez configurada la memoria flash, no es necesario preocuparse por ella.
- Una FPGA modelo **Virtex-II Pro 50** a la que se denominada CNET (chip de Control de la NetFPGA). Mediante la programación de esta FPGA se consigue que la plataforma NetFPGA sea reconfigurable. Cargando en ella distintos *bitfiles* la NetFPGA se comporta como una tarjeta de red, un router, etc.
- Cuatro puertos **Gigabit Ethernet**.
- Memoria **SRAM** de 4,5 MBytes.
- Memoria **DDR2** de 64 MBytes.

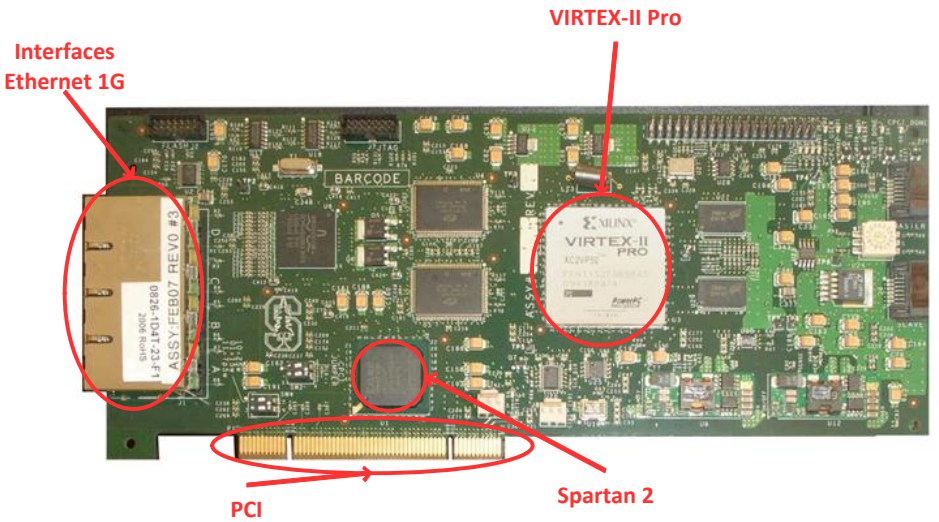

Figura 3.2: La NetFPGA se conecta a un PC mediante el PCI

3.1.2 NetFPGA packages

El otro elemento que conforma la plataforma NetFPGA es el conjunto de paquetes NFPs. Un NFP (NetFPGA Package) es un conjunto de ficheros con los códigos fuente escritos en lenguajes hardware y software (VHDL, Verilog, Perl, etc.) necesarios para implementar alguna funcionalidad en la NetFPGA. A través de estos paquetes la plataforma proporciona a

investigadores y desarrolladores una herramienta de trabajo.

Un ejemplo de paquete NFP es el Reference Router [13]. Éste contiene el conjunto de fuentes que permiten configurar la NetFPGA para que funcione como un router.

Existen tres formas en que los desarrolladores pueden usar los paquetes NFPs que se explicarán usando el ejemplo anterior:

- Modificar el software sin necesidad de tocar el hardware. Por ejemplo se puede configurar el hardware con el router por defecto y modificar el software para implementar un nuevo protocolo.
- Comenzar con el diseño inicial e implementar alguna mejora o nueva funcionalidad. Por ejemplo mejorar el rendimiento del router. Este es el modelo seguido en el desarrollo de este proyecto.
- Crear un paquete nuevo con todo el diseño hardware y software. El diseñador puede usar módulos de la librearía oficial NFP para crear nuevas funcionalidades o puede comenzar completamente desde cero.

3.1.3 Jerarquía de ficheros

Cualquier proyecto realizado en Verilog o en VHDL se estructura en módulos. Un módulo es una "caja" con un conjunto de entradas y salidas que realiza una determinada función. A su vez un módulo puede contener otros en su interior.

Los tres módulos principales en cualquier proyecto de la NetFPGA de 1 Gbps son el *Top*, el *Core* y el *User Data Path*. En el *Top* están instanciadas todas las entradas y salidas con el resto del hardware de la NetFPGA. Dentro del *Top* se encuentra el *Core* con toda la lógica que simplifica la interfaz con las memorias, interfaces Ethernet, etc. Por último, dentro del *Core* se instancia el *User Data Path*. Se trata de un módulo preparado para que el desarrollador implemente toda la lógica. En la mayoría de los proyectos es en él donde se realiza todo el trabajo (ver figura 3.3).

Top

Módulo principal del proyecto. Instancia el *Core* y todas las señales de entrada y salida de la FPGA con sus respectivos *buffers*. Estas señales se conectan con el resto del hardware de la NetFPGA (memorias, interfaces Ethernet, CPCI...). A continuación se presenta un listado con las más importantes:

Interfaces Ethernet: Señales de control y transmisión de datos de las cuatro interfaces

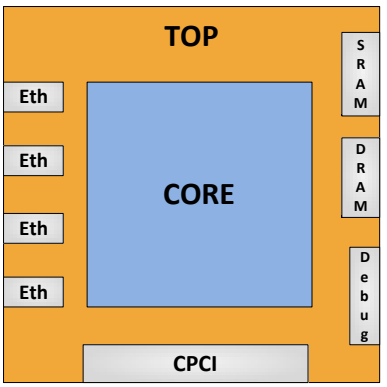

Figura 3.3: Módulo TOP. Instancia todas las entradas y salidas

Ethernet de 1 Gbps.

CPCI: FPGA de control del PCI.

Señales de reloj: La NetFPGA trabaja con una frecuencia de 62.5 MHz. A su vez, dentro del *Top*, se instancian varios DCMs (Digital Clock Managers) encargados de manipular las señales del reloj.

Memorias: Conexión con las memorias DRAM y SRAM.

Debug bus: la FPGA integra una serie de pines preparados para depurar los programas. Mediante el uso de un programa llamado Chip Scope es posible obtener el valor de las señales internas de la FPGA.

Pines de configuración: se conectan con la CPCI. Usando una configuración en cadena es posible programar la FPGA directamente desde el PC.

Core

En el *Core* se encuentra el *User Data Path* así como una serie de módulos, a los que denominaremos *wrappers*. Estos *wrappers* se encargan de simplificar las interfaces con el resto de los componentes (memorias, ethernet, etc.) haciendo que múltiples señales de control y transmisión se transformen en unas pocas, más intuitivas y fáciles de manejar (figura 3.4).

User Data Path

El *User Data Path* es un módulo preparado para que los desarrolladores puedan implementar en él sus proyectos de una forma fácil y cómoda. Contiene todos los módulos instanciados por los usuarios. Implementa las conexiones con las colas de entrada y salida de las interfaces Ethernet así como las señales de control y datos de las memorias y resto de componentes.

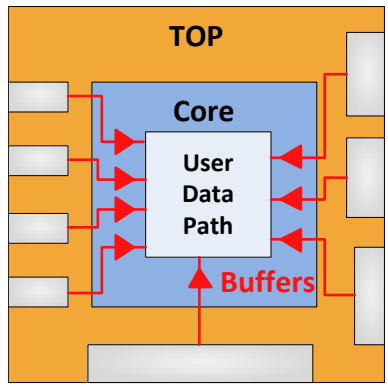

Figura 3.4: Módulo Core. Instancia los buffers y el User Data Path

Las colas de entrada están conectadas al *input arbiter*, que se encarga de servir los paquetes al siguiente módulo. El *input arbiter* conecta con la lógica implementada por desarrolladores, a la que se denominará lógica del usuario, y ésta conecta con las colas de salida (ver figura 3.5). Dependiendo de las funciones que realice la lógica del usuario la NetFPGA se comportará de una forma u otra. A continuación se presentan dos ejemplos posibles de lógica de usuario que cambian el funcionamiento de la NetFPGA.

1.– Router: si se introduce una lógica en la que se especifique una tabla de rutas y se encargue de redirigir los paquetes de entrada por las interfaces de salida en función de dicha tabla de rutas se tendría un router básico. Si además se hace que dicha tabla de rutas cambie de forma dinámica se tendría un router con mejores prestaciones.

2.– NIC: otro modo de funcionamiento es hacer que los paquetes que llegan de las interfaces Ethernet se transmitan a la CPU a través del PCI y viceversa. Con este diseño se tendría una tarjeta de red (figura 3.5).

3.1.4 Comunicación con la NetFGA, registros

La plataforma NetFPGA ofrece una forma fácil y cómoda de leer y escribir registros en la FPGA. Proporciona un módulo, denominado *Generic Regs* el cual permite instanciar dos tipos de registros:

1.– Software: registros en los que se escribe desde el PC y de los que se lee desde la FPGA.

2.– Hardware: registros en los que se escribe desde la FPGA y de los que se lee desde el PC.

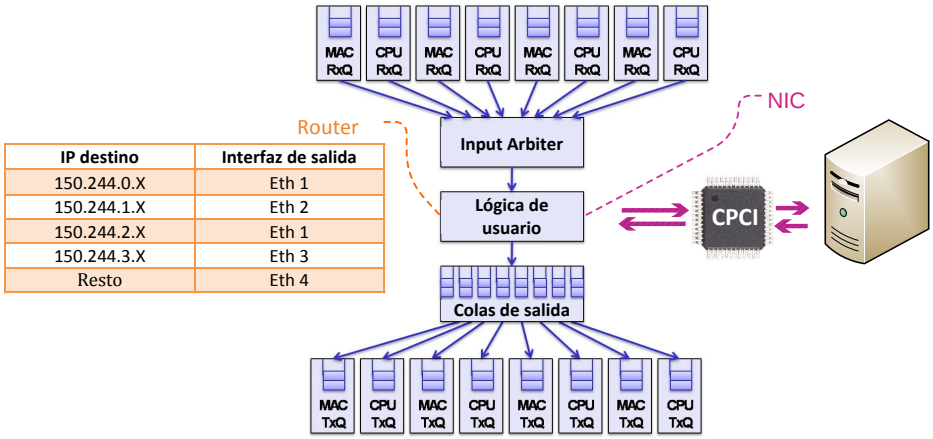

Figura 3.5: El User Data Path es el módulo en el que los desarrolladores implementan la lógica deseada.

Usando estos registros es posible leer y escribir datos en la NetFPGA desde el PC utilizando un driver apropiado que se comunica con la NetFPGA a través del PCI. Además la plataforma proporciona una serie de librerías en C con las funciones para acceder a dichos registros. Se usarán a lo largo de todo el proyecto con utilidades muy diferentes (habilitar módulos, configurar IPs, obtener estadísticas de la NetFGA, etc.).

3.2 Módulo Netflow

3.2.1 Descripción

El objetivo de este proyecto es la obtención de estadísticas de red mediante la extracción de flujos NetFlow. La plataforma NetFPGA proporciona un módulo, creado por Martin Zadnik, capaz de de exportar flujos NetFlow versión 5 [13]. Dicho módulo constituye el punto de partida para el proyecto.

El módulo NetFlow es capaz de recibir paquetes por una interfaz Ethernet, extraer la información de los flujos concernientes a dichos paquetes, formar paquetes de tipo UDP y exportarlos a un colector que se encuentre a la escucha (Ver figura 3.6). Los procesos de extracción de los flujos, formación de los paquetes UDP y su exportación están completamente implementados en la tarjeta NetFPGA. La configuración de la tarjeta, así como el proceso de colección de paquetes, se realizan mediante software desde un PC anfitrión.

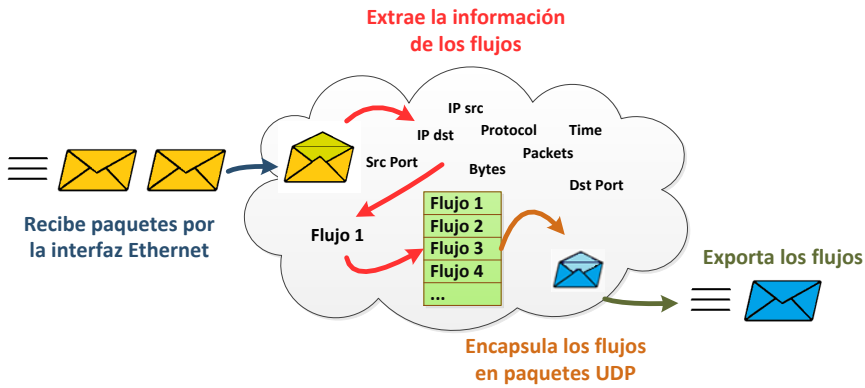

Figura 3.6: Funcionamiento del módulo NetFlow.

Las características del módulo NetFlow (anteriormente descritas en el apartado 2.3.2) son:

- Capacidad de obtención de medidas en redes de 1 Gbps.
- Medida de flujos IPv4.
- Memoria para 4000 flujos concurrentes.
- TimeStamp con una precisión de milisegundos sincronizado con el equipo anfitrión.
- Exporta registros de flujos usando el prococolo NetFlow v5.

3.2.2 Visión general del módulo netflow

El módulo NetFlow se trata del módulo principal del proyecto. Se encuentra implementado dentro del *User Data Path* (como ya se comentó en la sección 3.1.3). Su función es recibir los paquetes IPv4, extraer la información de los flujos y exportarla.

El módulo principal se encuentra dividido en una serie de módulos, cada uno con diferente funcionalidad. Para obtener una visión general del funcionamiendo del proyecto se hará una introducción de cada uno de los módulos. En la sección 3.2.3 se realiza un análisis más detallado de cada uno de ellos.

Pipeline

El procesamiento de la información dentro del módulo NetFlow se realiza mediante un pipeline. La entrada de este pipeline son los paquetes recibidos y su salida son paquetes CFLOW, paquetes de tipo UDP con la información de los flujos extraídos. Estos paquetes están preparados para ser exportados a un colector. La configuración del pipeline se muestra en la figura 3.7.

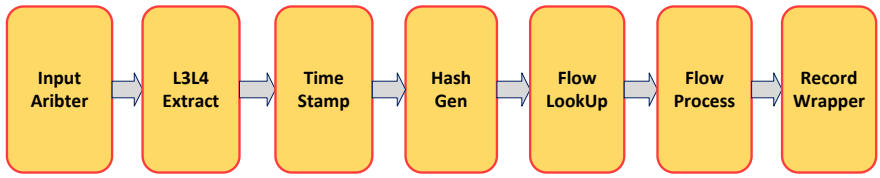

Figura 3.7: El procesamiento de la información se realiza mediante un pipeline

La información se transmite gracias a un protocolo interno de comunicación, protocolo equivalente a las FIFOs de Xilinx o al nuevo bus AXI Stream, que utiliza distintas señales (bus de datos, bus de control, señales de habilitación de lectura y escritura...).

Input arbiter

Tiene una entrada con una cola asociada por cada interfaz y una única salida. Siguiendo un algoritmo Round-Robin, recorre cada cola de entrada para servir un paquete de cada una de ellas en cada iteración. (Ver figura 3.8). Se trata de un módulo proporcionado por las librerías estándar de la NetFPGA.

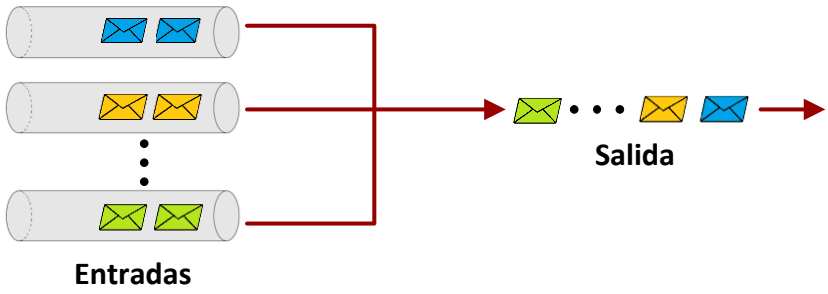

Figura 3.8: El input arbiter sirve un paquete de cada entrada en cada iteración.

L3L4 Extract

Extrae de la cabecera de cada paquete los bytes útiles para obtener la información de los flujos (IPs, puertos, protocolo, número de bytes, etc.). Descarta el resto de la información y los paquetes que no se correspondan con los protocolos soportados (Ver figura 3.9).

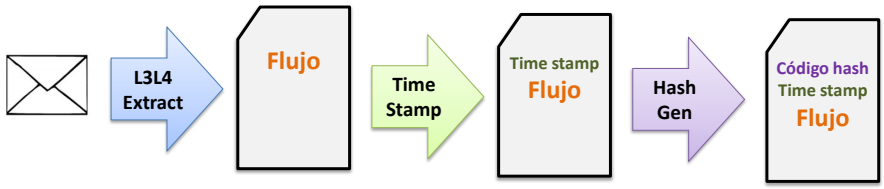

Figura 3.9: Módulos L3L4 Extract, Time Stamp y Hash generator.

Time Stamp

Inserta una marca de tiempo relativa al tiempo de llegada del primer paquete. Este módulo posee un contador de 32 bits cuya precisión es de milisegundos. No se producirá overflow hasta que no hayan transcurrido 49 días y 17 horas (cifra que puede ser mejorada fácilmente aumentando el contador a 64 bits). La velocidad del contador se puede ajustar mediante un valor que cuenta el número de ciclos de reloj en un milisegundo (Ver figura 3.9).

Hash Generator

Genera e inserta un código hash de 64 bits. Para generar dicho código se usan bits pertenecientes a las IPs origen y destino, puertos origen y destino, interfaz de entrada y protocolo. La probabilidad de colisión de dos flujos, es decir, la probabilidad de que dos flujos diferentes tengan el mismo código hash, es aproximadamente de $3.81 \cdot 10^{-6}$ (calculada en el apartado 3.2.3).

Flow Look Up

Controla los flujos que se encuentran presentes en memoria de flujos. Recibe el código hash del flujo que se está procesando y devuelve el valor de la posición de memoria donde se encuentra dicho flujo (previamente almacenado en una tabla). Tiene una capacidad para almacenar 4096 flujos concurrentes. Si llega un paquete de un flujo que no se encuentra en la tabla y ésta está llena, el paquete se descarta (Ver figura 3.10).

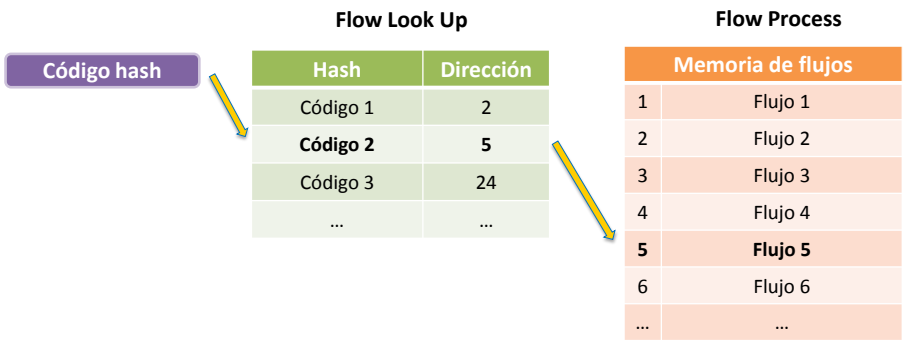

Figura 3.10: Módulos Flow Look Up y Flow Process. Almacenan la información de los flujos.

Flow Process

Controla la creación, actualización o expiración de los flujos. Se encuentra conectado con una memoria RAM implementada mediante blockRAMs, bloques de memoria dentro de la FPGA, en la que se almacenan los registros completos de cada flujo activo.

Ejecuta dos procesos en paralelo, el primero encargado de crear o actualizar los flujos y el segundo de controlar el tiempo de expiración de cada uno. Cuando un flujo expira se elimina de la memoria y pasa al siguiente módulo (Ver figura 3.10).

Record Wrapper

Recibe los flujos expirados por el módulo *Flow Process* y los almacena en un buffer. Cuando el buffer contiene 15 flujos o han transcurrido más de 20 ms desde el flujo expirado más antiguo todos los flujos son encapsulados en un paquete tipo NetFlow que se envía a la cola de salida de la interfaz seleccionada (Ver figura 3.11).

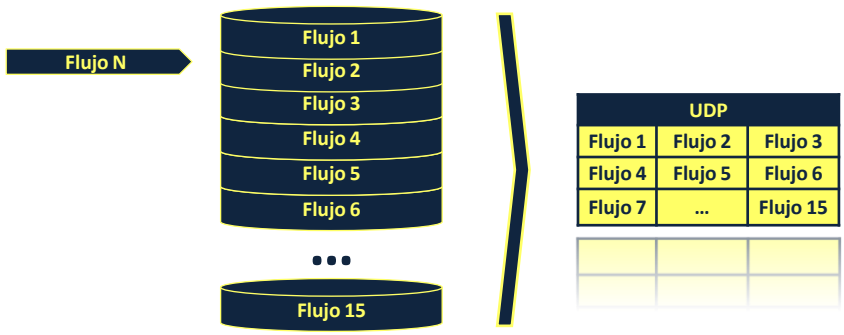

Figura 3.11: Record wrapper. Empaqueta los flujos en paquetes UDP.

3.2.3 Jerarquía de ficheros

En este apartado se explicará de forma detallada el funcionamiento de cada uno de los módulos así como el flujo que sigue la información desde que se reciben los paquetes hasta que se exportan los flujos.

User Data Path

El *User Data Path* tiene ocho entradas, cada una de ellas conectada a cada una de las colas de entrada. Cuatro de estas colas se corresponden con las cuatro interfaces físicas Ethernet y las otras cuatro con las interfaces software.

Las interfaces software son las interfaces Ethernet que se comunican con el equipo host a través del PCI. Es decir, si se desea enviar un paquete a otro PC a través del cable Ethernet éste se enviará por una interfaz física. Si se desea enviar un paquete al PC para recibirlo a través de un sniffer (como el Tcpdump) se hará a través de una interfaz software.

Como ya se ha explicado anteriormente, las colas de entrada están conectadas al *Input Arbiter*, módulo genérico de la plataforma. El *Input Arbiter* se encarga de recibir los paquetes y servirlos al *L3L4Extract*.

L3L4 Extract

Dentro del módulo NetFlow la información se transmite en serie en bloques de 64 bits. En esta sección y en las siguientes se detallará el camino que siguen estos los bloques a lo largo del pipeline (también mostrado en el apéndice A).

El *L3L4Extract* recibe todo el paquete dividido en bloques de 64 bits. Contiene una máquina de estados, mostrada en la figura 3.12, que se encarga de procesar cada paquete, extraer la información útil y pasarla al siguiente módulo.

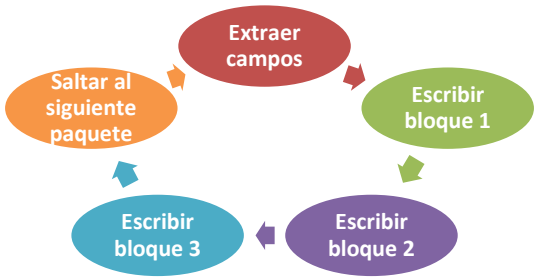

Figura 3.12: Máquina de estados del L3L4Extract

En un primer momento la máquina se encuentra en el estado *Saltar al siguiente paquete*. Cuando se recibe el primer bloque del siguiente paquete, identificado mediante una señal de control proveniente del Input Arbiter, se pasa al estado *Extraer campos*. En este momento se activa otra máquina de estados, más sencilla, que se encarga de recorrer los siete primeros bloques y extraer la información útil. En la tabla 3.1 se puede observar los campos extraídos de cada bloque.

Bloque	Datos extraídos
1	Interfaz
3	Versión IP, ToS
4	Número de bytes, ttl, protocolo
5	IP origen, IP destino (primeros 16 bits)
6	IP destino (útimos 16 bits), puerto origen, puerto destino
7	Flags

Tabla 3.1: Campos extraídos en el módulo L3L4Extract

Cuando la extracción se ha completado se comprueba que el paquete sea IPv4 y que su protocolo sea TCP, UDP o ICMP. Si cumple con estas características se pasará al estado *Escribir bloque 1*, de lo contrario el paquete se descartará y se pasará al estado *Salta al siguiente paquete*.

Los tres estados de escritura se encargan de formar los tres bloques que se transmitirán

al módulo *Time Stamp*. La figura 3.13 muestra los bloques recibidos y cómo se encapsula la información en los bloques de salida.

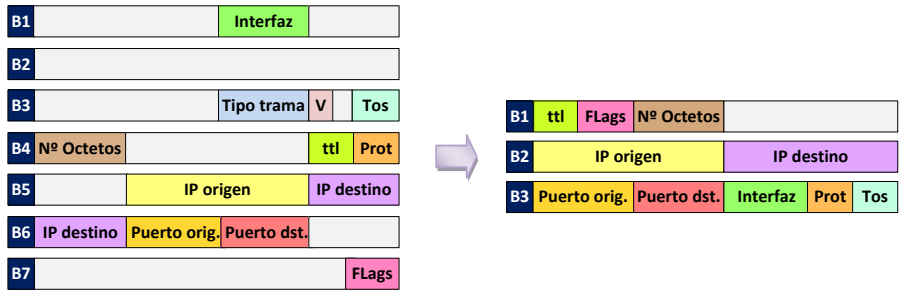

Figura 3.13: Transmisión de bloques en el módulo L3L4Extract

Registros

El módulo *L3L4Extract* implementa cuatro registros.

- Reg sw: Registro software para habilitar el módulo. Escribiendo en este registro desde el PC se puede habilitar o deshabilitar el módulo y, puesto que es el primero en el pipeline, todo el módulo NetFlow.
- Cnt discarted: Contador de paquetes descartados (paquetes que no sean TCP, UDP o ICMP IPv4).
- Cnt accepted: Contador de paquetes aceptados.
- Cnt total: Número total de paquetes vistos por el módulo.

Time Stamp

El módulo *Time Stamp*, como su propio nombre indica, se encarga de insertar una marca de tiempo en cada paquete. Esta marca de tiempo se corresponde con el número de milisegundos desde el comienzo del proceso de captura.

El módulo usa dos contadores, el primero es un contador de ciclos de reloj. Conociendo la velocidad del reloj de la FPGA (62,5MHz) es posible calcular el número de ciclos en un milisegundo. Mediante un registro software se configura el contador para que se produzca un pulso cada milisegundo. El segundo contador se incrementa cada vez que se produce pulso en el primero, obteniendo así un contador de milisegundos (figura 3.14). Estos contadores se reinician al comienzo del proceso de captura.

Figura 3.14: Mediante el contador de ciclos de reloj se configura el contador de milisegundos

Cada vez que llegan los tres bloques de un paquete, el módulo *Time Stamp* envía al siguiente módulo un bloque con la marca de tiempo y los tres bloques de datos. Puesto que el contador de milisegundos es de 32 bits no se producirá overflow hasta que no hayan transcurrido 49 días y 17 horas.

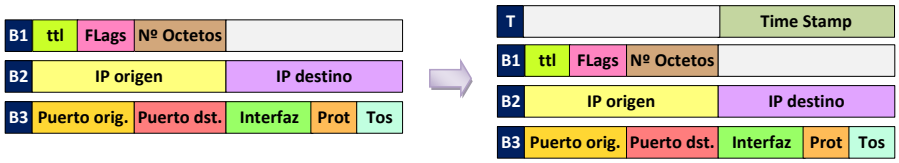

Figura 3.15: Tranmsión de bloques en el módulo Time Stamp

Registros

El módulo *Time Stamp* implementa dos registros.

- Sw num tick: Registro software para configurar el contador de ciclos de reloj.
- Cnt timestamp: Registro hardware con el valor del contador de milisegundos.

Hash Generator

El módulo *Hash Generator* genera un código hash de 64 bits que posteriormente se usará para direccionar y almacenar los flujos.

El módulo recibe los cuatro bloques del *Time Stamp* (tres de datos y uno con la marca de tiempo) y extrae los siguientes campos:

IP origen	IP destino
Puerto origen	Puerto destino
Interfaz de entrada	Protocolo

Tabla 3.2: Campos empleados para calcular el código hash

Una vez extraídos aplica un código CRC de 64 bits para obtener el código hash. Este código se transmite al siguiente módulo seguido por el Time Stamp y los tres bloques de datos (Ver figura 3.16).

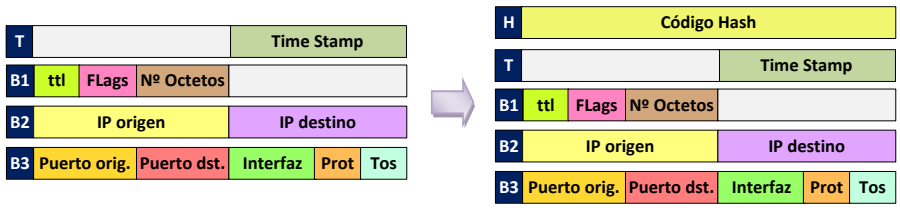

Figura 3.16: Transmisión de los bloques en el módulo hash

Probabilidad de colisión

Martin Zadnik, autor del módulo NetFlow, define la probabilidad de que colisión (probabilidad de que dos flujos diferentes generen el mismo valor hash) como:

$$p_{colision}(n) = 1 - \frac{m!}{m^n(m-n)!} \approx 1 - e^{-\frac{n^2}{2m}} \approx 10^{-13} \tag{3.1}$$

Donde el número total de valores hash es $m = 2^{48}$ y el número de posiciones de memoria $n = 2^{12} = 4096$. Como se verá más adelante, aunque el código hash tenga 64 bits, sólo son útiles 48. Para realizar este cálculo se asume que cada valor de hash puede ir en cualquier posición de memoria, sin embargo, el cálculo podría mejorarse para una caché asociativa de K vías.

Se ha realizado un análisis tanto del módulo como de la fórmula y se ha encontrado que la probabilidad de colisión es mayor ya que el número de bits útiles del código hash no es 48 si no 41.

La probabilidad de colisión de dos flujos [19] (probabilidad de que a dos valores hash

diferentes se les asigne la misma posición de memoria) viene dada por:

$$p_{colision}(m,n) = 1 - p_{no\ colision}(m,n) \tag{3.2}$$

Donde m es el número de posibles valores hash y n el número de posiciones memoria. Para rellenar la tabla de forma que no colisione ningún flujo se puede elegir el primero entre m posibles, el segundo entre $m-1$, etc.

$$p_{no\ colision}(m,n) = \frac{m}{m}\frac{m-1}{m}...\frac{m-(n-1)}{m} = \prod_{k=0}^{n-1}(1-\frac{k}{m}) \tag{3.3}$$

Por otro lado, usando el desarrollo de Taylor alrededor de x=0, se tiene que:

$$e^x = \sum_{k=0}^{\infty}\frac{x^k}{k!} \approx \frac{x^0}{0!} + \frac{x^1}{1!} = 1 + x \tag{3.4}$$

Tomando $x = -\frac{k}{m}$ se tiene que $1 - \frac{k}{m} \approx e^{-\frac{k}{m}}$. Sustituyendo en la ecuación 3.3:

$$p_{no\ colision}(m,n) = \prod_{k=0}^{n-1} e^{-\frac{k}{m}} = e^{\sum_{k=0}^{n-1} -\frac{k}{m}} = e^{-\frac{1}{m}\sum_{k=0}^{n-1} k} = e^{-\frac{n(n-1)}{2m}} \tag{3.5}$$

Si $1 << n^2 << 2m$ entonces:

$$p_{colision}(m,n) \approx 1 - e^{-\frac{n^2}{2m}} \tag{3.6}$$

Volviendo a aplicar la aproximación de Taylor descrita en la ecuación 3.4 y tomando $x = -\frac{n^2}{2m}$ se obtiene

$$p_{no\ colision}(m,n) \approx 1 - e^{-\frac{n^2}{2m}} \approx 1 - (1 - \frac{n^2}{2m}) = \frac{n^2}{2m} \tag{3.7}$$

En conclusión

$$p_{no\ colision}(m,n) \approx \frac{n^2}{2m} \approx 3.81 \cdot 10^{-6} \tag{3.8}$$

Para $m = 2^{41}$ y $n = 4096$. Como se puede comprobar este valor es mucho mayor al propuesto por Martin Zadnik. De hecho, para $m = 2^{48}$ y $n = 4096$ da $2.98 \cdot 10^{-8}$.

Registros

El módulo sólo implementa un registro software, empleado para escribir la semilla del código CRC-64.

FLow Look Up

El módulo *Flow Look Up* almacena la información de los flujos que se encuentran activos en memoria y su posición. La entrada es el código hash y la salida la dirección de memoria principal donde se encuentra el flujo.

Implementa, mediante blockRAMs, una memoria asociativa de ocho vías. El módulo parte el código hash en dos trozos de 32 bits. De los primeros 32 bits usa nueve para direccionar una posición de memoria. Debido a la ocupación de la FPGA no es posible usar los 32 bits.

Cada dirección de memoria contiene ocho bloques de 32 bits. Los últimos 32 bits del código hash se comparan con cada uno de los bloques y si uno de ellos contiene el cógido significa que el flujo está presente en memoria. En total hay $2^9 \cdot 2^3 = 2^{12}$ bloques de memoria.

La dirección del flujo en memoria principal se calcula como la concatenación de los nueve bits de la posición de memoria y los tres bits que indican el bloque (para direccionar ocho bloques se utilizan tres bits).

En total se usan 12 bits para direccionar los flujos en memoria principal lo que limita el número de flujos concurrentes a $2^{12} = 4096$. Si un flujo no se encuentra presente en memoria y hay bloques libres se usará uno de estos bloques para almacenar dicho flujo. Cada bloque contiene un bit de validez para indicar si está libre.

Si la memoria está llena el flujo se descarta. Este hecho dará lugar a una *pérdida de flujos* que no serán capturados por la tarjeta. En el siguiente capítulo se analizarán los efectos de esta pérdida.

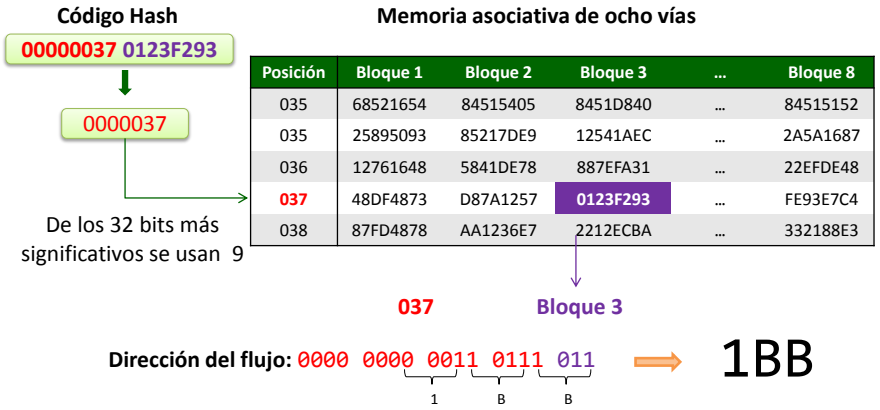

Figura 3.17: Memoria asociativa de ocho vías.

Registros

El módulo Flow Look Up implementa tres registros.

- Sw debug: Registro software usado para depurar.
- Cnt discartds: Contador de paquetes descartados (debido a que la memoria está llena).
- Cnt items: Contador de flujos capturados.

Flow Process

El módulo *Flow Process* almacena la información de los flujos activos. Los datos almacenados de cada flujo se presentan en la tabla 3.3. El modulo implementa una memoria RAM con una capacidad para 4096 flujos.

IP origen	IP destino	Puerto origen
Puerto destino	Protocolo	Tiempo de llegada del primer paquete
Tiempo de llegada del último paquete	Número de paquetes	Número de octetos
Ttl	Flags	ToS

Tabla 3.3: Campos almacenados de cada flujo

Ejecuta dos procesos en paralelo, el primero encargado de crear y almacenar los flujos y el segundo de eliminarlos.

Proceso de creación y actualización de flujos

Cuando se recibe un paquete del módulo *Flow Look Up* se accede a la posición de memoria indicada para dicho flujo. Se lee toda la información del flujo. Mediante un bit de validez se identifica si la posición estaba ocupada y se realiza una de las siguientes operaciones:

- Crear: se inicializa el flujo con la información del paquete recibido.
- Actualizar: se actualiza la información del flujo (se suman los bytes del paquete recibido con los ya almacenados, se actualiza el tiempo del último paquete, etc.).

Proceso de eliminación de flujos

Existen dos tiempos, y dos registros asociados a cada tiempo, que marcan la condición para que un flujo expire:

- Tiempo de inactividad: se refiere al tiempo transcurrido desde la llegada del último paquete. Si un flujo no ha recibido paquetes en un tiempo superior al valor indicado en el registro *Inactive Time Out*, por ejemplo 10 segundos, el flujo expira.
- Tiempo de actividad: se refiere al tiempo desde que el flujo fue creado. El flujo se elimina si el tiempo desde que se creó es superior al valor indicado por el registro *Active Time Out*, por ejemplo 30 segundos, aunque éste siga recibiendo paquetes.

El proceso de eliminación de flujos recorre la memoria continuamente calculando el tiempo de actividad e inactividad de cada flujo y comprobando si el flujo debe expirar. Cuando un flujo expira se realizan tres acciones:

1.– Se manda un comando al módulo *Flow Look Up* para que este lo elimine de su tabla.

2.– El *Flow Look Up* borra el índice de dicho flujo de su memoria y devuelve un comando al *Flow Process*.

3.– El *Flow Process* borra el flujo de memoria y lo manda al siguiente módulo.

Registros

El módulo *Flow Process* implementa cinco registros:

- Reg inactive timeout: registro software usado para configurar el tiempo de inactividad.
- Reg active timeout: registro software para configurar el tiempo de actividad.
- Cnt delete: registro hardware para contar los flujos eliminados.
- Cnt update: registro hardware para contar los flujos actualizados.

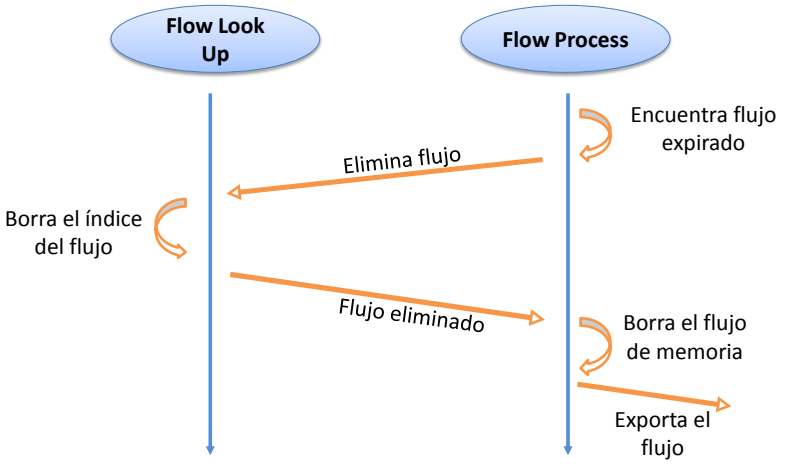

Figura 3.18: Proceso de expiración y eliminación de flujos

- Cnt new: registro hardware para contar los flujos nuevos.

Record Wrapper

El módulo *Record Wrapper* recibe los flujos expirados del *Flow Process* y los almacena en un buffer. Cuando han llegado 15 flujos o han transcurrido más de 20 ms desde la llegada del primero, el módulo encapsula todos los flujos almacenados en el buffer en un paquete UDP siguiendo el protocolo NetFlow V5 [20].

Al comenzar el proceso de captura el módulo almacena la fecha y hora del equipo al que está conectada la NetFPGA. Como se comentó en el apartado 3.2.3, las marcas de tiempo de cada flujo son relativas a dicha hora (número de milisegundos transcurridos desde el comienzo del proceso de captura). De esta forma es posible obtener los tiempos de cada flujo en formato absoluto (año, mes, día, hora, minutos, segundos y milisegundos).

Por último se envía el paquete UDP a una de las colas de salida, las cuales están conectadas a las interfaces Ethernet (ver figura 3.19). Este módulo conecta con las colas de salida genéricas, al igual que el *L3L4Extract* lo hacía con el *Input Arbiter*. De esta forma se completa el ciclo completo de recepción de paquetes, extracción de flujos y su exportación.

Es importante resaltar que en el *Record Wrapper* se configuran todos los campos necesarios para la exportación de paquetes. Es aquí donde se escribe la IP del equipo host, IP destino, puertos, direcciones MAC, etc.

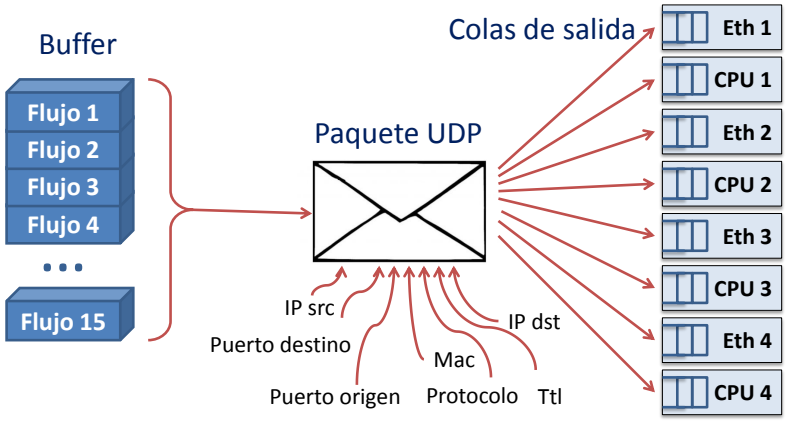

Figura 3.19: Record Wrapper. Forma los paquetes UDP y los envía a las colas de salida

Registros

Los registros implementados por el *Record Wrapper* son:

- Sw debud: registro software usado para depurar el módulo.
- Hw debug: registro hardware usado para depurar el módulo.
- Cnt epoch time: registro con el tiempo real del equipo host.
- Cnt records fifo: registro hardware con el número máximo de flujos permitidos en el buffer.
- Cnt total regs: registro hardware con el número total de flujos exportados.

3.2.4 Memorias

Las memorias RAM usadas en módulo NetFlow se instancian dentro de la propia FPGA mediante blockRAMs. El proyecto incluye su propio módulo genérico que permite implementar memorias de diferentes tamaños. Estas memorias son las implementadas en los módulos *Flow Look Up* y *Flow Process*.

El módulo de memoria recibe como parámetros el número de posiciones y el tamaño de cada posición. Implementa un vector de registros con tantos registros como direcciones. Tiene dos entradas síncronas, una de ellas para leer y escribir y la otra sólo para escribir (figura 3.20).

El tamaño de las memorias es una limitación importante en el proyecto ya que, si son

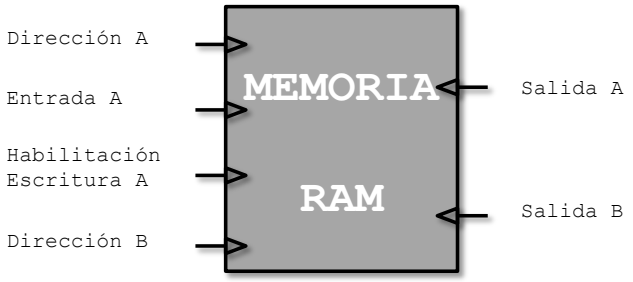

Figura 3.20: Bloque de memoria RAM

demasiado grandes, pueden ocupar demasiado espacio en la FPGA o no caber en ella. Como ya se ha visto, existen memorias RAM externas accesibles desde la FPGA. Sin embargo su acceso es más lento por lo que se reduce el rendimiento y la velocidad de trabajo (alcanzando menores tasas de línea).

3.2.5 Añadir registros

Una tarea importante realizada antes de implementar el muestreo es añadir nuevos registros. Recordemos que un registro proporciona una herramienta de comunicación entre el PC y la FPGA. Posteriormente estos registros se usarán para diferentes tareas: configurar la tasa de muestreo de forma dinámica, depurar el código, obtener estadísticas, etc.

Para añadir nuevos registros dentro de un módulo los pasos a realizar son:

1.– Modificar la instanciación del *Generic Regs* (ver apartado 3.1.4) dentro del módulo. Cuando se instancia el *Generic Regs* se fijan dos parámetros:

- Número de registros hardware y software.
- Nombre de los registros.

2.– Modificar la declaración de los bloques de memoria. Dentro del proyecto NetFlow existe un archivo llamado *netflow defines* con la definición de los bloques de memoria empleados por los todos los módulos. Cada módulo tiene un bloque de memoria asociado. De cada bloque se define:

- El tamaño (número de registros de 32 bits).
- Posición de memoria en la que se encuentra el bloque. No es necesario modificar este parámetro para añadir registros.
- Nombre de cada registro.

Después de realizar estos pasos ya es posible acceder a los nuevos registros. Para ello la plataforma NetFPGA proporciona una serie de librerías en C y Perl con funciones para acceder a ellos (figura 3.21). En concreto se han usado dos funciones de la librería Perl:

- Lectura:
  `my $srcip = nf_regread('nf2c0', NFLOW_RECORDWRAPPER_SRC_IP_REG());`
  Que recibe como parámetros el dispositivo donde se desea escribir `('nf2c0')` y la posición de memoria o registro `(NFLOW_FLOWPROC_INACTIVE_TIMEOUT_REG)` que se desea leer.
- Escritura:
  `nf_regwrite('nf2c0', NFLOW_TIMESTAMP_INCREMENT_REG()`
  `$default_ticksperms);`
  Que recibe como parámetros el dispositivo donde se desea escribir `('nf2c0')`, la posición de memoria o registro `( NFLOW_TIMESTAMP_INCREMENT_REG)`, y el valor que se desea asignar `($default_ticksperms)`.

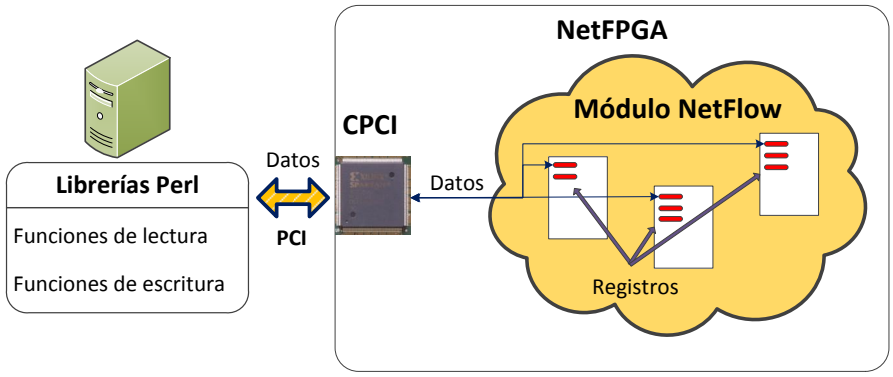

Figura 3.21: La plataforma proporciona librerías en Perl y en C para comunicarse con la NetFPGA.

NOTA: Para una descripción más detallada de este proceso ver el anexo B.

3.2.6 Implementar muestreo

La implementación del muestreo ha sido uno de los objetivos principales del proyecto. Para realizar esta tarea de han definido dos parámetros:

Tasa de muestreo o frecuencia de muestreo: es un número entero N que marca cada cuán-

tos paquetes se seleccionará uno para ser muestreado (1 de cada 10, uno de cada 100).

Offset: una vez fijada la tasa de muestreo debe fijarse qué paquete seleccionar de la secuencia de N recibidos. El offset es importante ya que si se despliegan varias tarjetas en la misma red cada una debe muestrear paquetes diferentes.

Por lo tanto el muestreo queda definido por estos parámetros. Un muestreo con una frecuencia de 10 y un offset 3 es aquel que muestrea el tercer paquete de cada diez recibidos.

El primer paso para implementar el muestreo es decidir el punto donde realizarlo. Tras realizar un estudio del módulo NetFlow (detallado en apartados anteriores) parece que el punto más adecuado es donde se realiza la extracción de los campos de los paquetes, en el módulo *L3L4Extract* (apartado 3.2.3).

El segundo paso es introducir dos registros que se usarán para configurar el muestreo, el `reg_sample_rate` y el `reg_sample_offset` y modificar los scripts de configuración para acceder a ellos.

Dentro de este módulo existe una máquina de estados (figura 3.12) que extrae los campos de cada paquete, comprueba si es un paquete TCP, UDP o ICMP y forma los bloques que se transmiten al siguiente módulo. Recordemos esta máquina de estados:

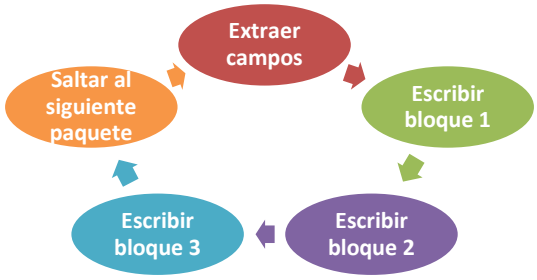

Figura 3.22: Máquina de estados del L3L4Extract

Cuando se recibe un paquete nuevo se pasa del estado *Saltar al siguiente paquete* al *Extraer campos* y se incrementa un contador de paquetes.

```
if (total_en) begin
    cnt_total = cnt_total + 1;
end
```

Código 3.1: Contador de paquetes

Se ha añadido un segundo contador para controlar el número de paquete dentro de la secuencia recibida que se reinicia cuando se alcanza la tasa de muestreo (indicada por el registro `reg_sample_rate`):

```
if (total_en) begin
    cnt_total = cnt_total + 1;
    sample_cnt = sample_cnt +1;
    if (sample_cnt == reg_sample_rate+1)
    sample_cnt = h1;end;
```

Código 3.2: Contador del paquete en una secuencia

Después se extraen los campos útiles y se comprueban dos condiciones:

1.– Que el paquete sea IPv4.

2.– Que el paquete sea TCP, UCP o ICMP.

```
if (l3proto_ok && l4proto_ok)
    nfsm_state = WRITEFIELD0;
else
    nfsm_state = SKIPTONEXTPACKET;
```

Código 3.3: Selección de un paquete

Aquí es donde se añade la condición para muestrear. Si el offset del paquete coincide con el offset de muestreo el paquete se muestrea, si no el paquete se descarta.

```
if (l3proto_ok && l4proto_ok && sample_cnt == reg_sample_offset)
    nfsm_state = WRITEFIELD0;
else
    nfsm_state = SKIPTONEXTPACKET;
```

Código 3.4: Muestreo de un paquete

4

Pruebas y resultados

4.1 Instalación y puesta en marcha

Durante el desarrollo del proyecto uno de los principales problemas, en tiempo y en esfuerzo, ha sido la instalación y puesta en marcha de la NetFPGA y, en especial, del módulo NetFlow.

Con el fin de facilitar una guía en caso de que se desease trabajar con dicho módulo, se ha creído conveniente incluir un apartado en el que se describa el proceso de instalación, así como las soluciones aportadas a los problemas encontrados.

4.1.1 Fedora y NetFPGA v2.2

El equipo necesario para instalar la NetFPGA es un PC con una ranura PCI libre y la propia tarjeta.

En la primera instalación se eligió el sistema operativo Fedora. Es posible descargar un live CD desde el portal web de la NetFPGA de una distribución llamada *Fedora Core 13 with NetFPGA* [21] que proporciona todas las librerías y drivers de la versión 2.2 de la NetFPGA.

Después de instalar Fedora, hay que programar la memoria flash de la FPGA denominada CPCI mediante una interfaz JTAG. Esto puede realizarse desde cualquier equipo que disponga de la herramienta Imact de Xilinx. Posteriormente, es posible conectar la NetFPGA al PCI y probar los primeros diseños.

Por otro lado, es necesario descargar e instalar el paquete del módulo NetFlow siguiendo el proceso de instalación descrito en la guía oficial [18].

No obstante, no fue posible realizar correctamente la instalación del módulo Netflow debido a su incompatibilidad con la versión 2.2 de la NetFPGA. Dicha incompatibilidad se debe a que la versión 2.2 de la NetFPGA instalada difiere en el sistema de archivos, especificamente en la localización de los directorios, con la versión 1.2 usada por Martin Zadnik, autor del módulo NetFlow.

Para solucionar este problema se decidió instalar el sistema operativo y la versión usada por Martin Zadnik: CentOS y NetFPGA v1.2, respectivamente.

4.1.2 CentOS y NetFPGA v1.2

La versión del sistema operativo usado ha sido CentOS 5.7 de 32 bits. Para instalar la NetFPGA se debe descargar la versión 1.2 de los repositorios [22] y seguir los pasos mostrados en el tutorial [23].

De nuevo, para la instalación del módulo NetFlow, basta con seguir las instrucciones mostradas en [18].

Durante la instalación se han encontrado dos problemas a desctacar:

- Java: la versión de Java no es compatible por lo que se debe desinstalar la versión de Java nativa en CentOS e instalar la versión 6 revisión 6 del entorno de trabajo y del entorno para desarrolladores [24] (jre-6u6 y jdk-6u6 respectivamente). Este error sólo afecta en caso de que vayan a usar las aplicaciones disponibles con intefaz gráfica. En el módulo NetFlow este error no afecta al proyecto.
- PCI: el script para programar el CPCI incluido en la NetFPGA v1.2 no funciona para el módulo NetFlow. Ha sido necesario ponerse en contacto con el autor del módulo, quien ha facilitado dicho script.

4.1.3 Self test y regression test

La plataforma NetFPGA proporciona una serie de herramientas útiles para comprobar el correcto funcionamiento de la tarjeta y de un determinado proyecto.

- **Self test:** se trata de una serie de tests que comprueban que todo el hardware de la NetFPGA funcione correctamente. Verifican el acceso a memoria, la comunicación con la interfaz SATA, etc. Al pasar los test se ha obtenido un error en el acceso a memoria DRAM, sin embargo este error no afecta al módulo NetFlow puesto que no usa la memoria RAM externa.
- **Regression test:** se trata de una serie de tests específicos para cada proyecto que comprueban su correcto funcionamiento. Todos los regression test del módulo NetFlow se han realizado con éxito.

4.1.4 Primera prueba

Como punto final de la instalación es necesario realizar una pequeña prueba para comprobar que es posible exportar flujos de forma correcta. Para ello se realizan lo siguientes pasos:

1.– Conectar una interfaz Ethernet de la NetFPGA a la tarjeta de red.

2.– Configurar la NetFPGA para que comience el proceso de captura y exportación de flujos (apéndice C).

3.– Comenzar a capturar los paquetes recibidos por la tarjeta de red con el WireShark.

4.– Ejecutar un *tcpreplay* para mandar tráfico de la tarjeta de red a la NetFPGA.

De esta forma la NetFPGA recibe los paquetes enviados por la tarjeta de red, extrae la información de los flujos y los exporta encapsulados en paquetes CFLOW, paquetes de tipo UDP, por la misma interfaz que los recibe (figura 4.1).

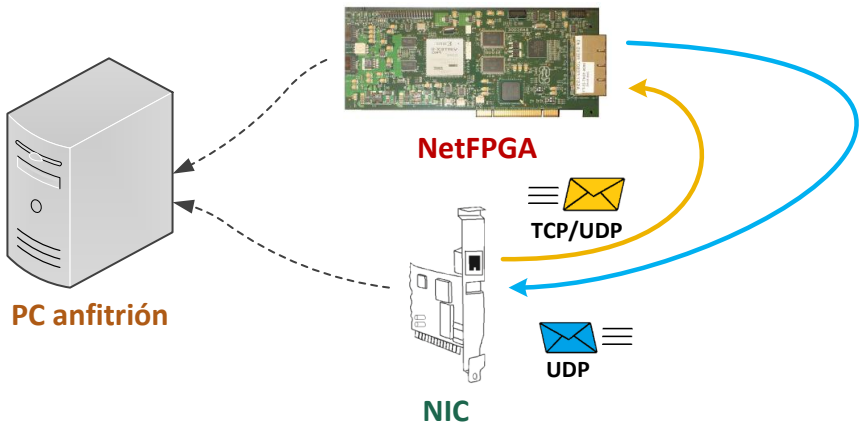

Figura 4.1: Esquema del proceso de captura.

4.2 Validación del proyecto

El objetivo principal del proyecto es la extracción de flujos, usando la NetFPGA, tanto con muestreo como sin él. Los siguientes apartados describen la metodología seguida y las pruebas realizadas con el fin de alcanzar dos objetivos:

- **Validación del módulo NetFlow:** se realizarán una serie de pruebas para comprobar el correcto funcionamiento del módulo NetFlow, es decir, para verificar que los flujos extraídos por la NetFPGA se corresponden con los flujos de la traza enviada.
- **Validación del muestreo:** después de validar el módulo NetFlow y de implementar el muestreo es necesario comprobar que este proceso se realice de forma correcta.

4.2.1 Metodología seguida

En este punto del proyecto la NetFPGA y el módulo NetFlow se encuentran instalados y en funcionamiento. Es posible enviar una traza a la NetFPGA y obtener sus flujos. No obstante, aún no se ha comprobado que la extracción de los flujos sea correcta.

Por lo tanto, el problema a resolver para la validación del módulo NetFlow es comprobar que los flujos obtenidos se corresponden con el tráfico escaneado. Para realizar dicha tarea se plantean tres pasos:

1.– Obtener la información de los flujos producidos por la NetFPGA de forma que sea posible procesarlos y trabajar con ellos. Para realizar esta tarea se han usado las *flow tools* [9].

2.– Obtener la información de los flujos con una herramienta diferente a la NetFPGA, sabiendo que dicha herramienta extrae los flujos de forma correcta. Para realizar esta tarea se ha usado el *flow process*, herramienta desarrollada por el personal del HPCN de la U.A.M.

3.– Comparar los flujos extraídos por la NetFPGA con los flujos extraídos por el *flow process*.

En referencia al paso 1, el proceso de generación y exportación de flujos en la NetFPGA puede realizarse, principalmente, en dos escenarios:

- Insertar una sonda en una red real y capturar el tráfico generado en tiempo real.
- Mandar a la tarjeta una traza *pcap* usando *tcpreplay*. En este caso la generación de flujos también se realiza en tiempo real, a pesar de que se trabaje en un entorno experimental, ya que el *tcpreplay* replica las mismas condiciones en las que se ha realizado la captura

de la traza. Ésta es la opción elegida para realizar las pruebas por dos razones:

- Se puede replicar siempre la misma traza y controlar las condiciones de la captura.
- Los costes del equipo utilizado son menores al poderse hacer las pruebas desde el propio PC donde está instalada la NetFPGA.

Paso 1, presentación de los resultados usando las Flow tools

El proceso de generación y exportación de flujos, descrito en el apartado 4.1.4, da como resultado la obtención de paquetes CFLOW en la interfaz de la tarjeta de red.

Las herramientas utilizadas para colectar los paquetes CFLOW y obtener la información de los flujos en un formato adecuado para su posterior procesamiento son las *Flow-Tools* [9]. Se trata de un conjunto de herramientas que permiten capturar, filtrar, imprimir y analizar registros de flujos exportados en formato NetFlow. En concreto, se han usado tres:

Flow-capture: captura los paquetes CFLOW y almacena la información en disco. Recibe como parámetro el directorio donde guarda la infromación de los flujos en ficheros de formato propio. Se encarga de gestionar dichos ficheros para que no ocupen demasiado espacio en disco.

Flow-cat: recibe como parámetro varios ficheros de salida del *flow-capture* y los concatena dejándolos perparados para el *flow-print*.

Flow-print: recibe los ficheros creados por *flow-capture* (si sólo se imprime un fichero) o por *flow-cat* (si se imprimen varios ficheros) e imprime los flujos en un fichero de texto. Permite elegir varios formatos de salida de forma que sólo imprime los datos deseados. A continuación se muestra un ejemplo de archivo de salida, donde cada línea se corresonde con un flujo:

StarTime	EndTime	SrcIP	SrcPort	DstIPaddress	DstPort	Prot	Pkts	Octets
0716.17:41:39.10	0716.17:41:40.507	138.212.186.208	2716	200.134.144.159	1025	17	23	1251
0716.17:41:39.21	0716.17:41:41.207	183.218.162.154	1080	138.212.186.208	2116	17	53	2238
0716.17:41:39.25	0716.17:41:42.160	138.212.186.208	2716	183.218.162.154	1080	17	3	0481
0716.17:41:39.18	0716.17:41:39.185	200.132.100.176	1117	138.212.185.148	83	6	71	0140

Paso 2, extracción de flujos usando Flow process

La herramienta utilizada para extraer los flujos de la traza pcap y poder compararlos con los extraídos por la NetFPGA es el *flow process*.

Desarrollado por el HPCN (High Performance Computing and Networking) de la U.A.M., el *flow process* es capaz (entre otras funcionalidades) de procesar un archivo pcap, extraer

los flujos e imprimirlos en un formato similar al generado por el *flow-print*. A continuación se muestra un fragmento de código de salida del *flow process*:

src	dst	sport	dport	proto	pkts	bytes	first	latest	duration
150.24.125.52	150.244.212.26	68	67	p17	1	328	1341247657.4	1341247657.4	0.0
138.22.23.176	138.212.184.14	1099	0139	p06	1	040	1341247657.8	1341247657.8	0.0
19.132.78.208	138.212.184.93	7960	7960	p17	1	055	1341247657.8	1341247657.8	0.0
215.132.19.31	138.212.16.191	0080	1220	p06	1	448	1341247657.8	1341247657.8	0.0
138.22.115.53	138.212.21.164	0139	2133	p06	2	300	1341247657.8	1341247658.1	0.3
138.22.104.17	139.45.177.202	3801	6069	p06	1	040	1341247657.8	1341247657.8	0.0

El *flow process* no incluye la funcionalidad de muestreo por lo que ha sido necesario modificar el programa para incluir dicha funcionalidad y adaptarla a la forma de trabajar de la NetFPGA (apartado 3.2.6), es decir, permitir configurar el muestreo mediante tasa de muestreo y *offset*.

Paso 3, comparación de los resultados mediante Awk

El lenguaje de programación elegido para procesar los archivos de salida generados por la NetFPGA y el *flow process* es Awk [25]. Dichos archivos de salida son documentos de texto con miles de líneas, todas ellas siguiendo el mismo formato. Awk permite procesar estos documentos de forma rápida y eficiente.

Durante la realización del proyecto se ha creado una colección de scripts en awk con diferentes funcionalidades: comparar los archivos de salida y dar una medida de su similitud, obtener estadísticas, agregar flujos, etc. Aunque en esta memoria sólo se dará una pequeña descripción de estos scripts, los resultados mostrados en los siguientes apartados han sido obtenidos mediante su uso.

A continuación se describe el esquema de funcionamiento de los principales scripts awk utilizados (ilustrado en la figura 4.2):

1.– Se lanza el programa pasándole como parámetros los archivos generados por la NetFPGA y el *flow process*.

2.– Se leen estos archivos y se crean dos tablas con los flujos generados por la NetFPGA y el *flow process* respectivamente.

3.– Se procesan dichas tablas realizando algún trabajo con ellas, por ejemplo comparar los flujos.

4.– Se imprime el resultado en un fichero de texto, por ejemplo el número de flujos correctos.

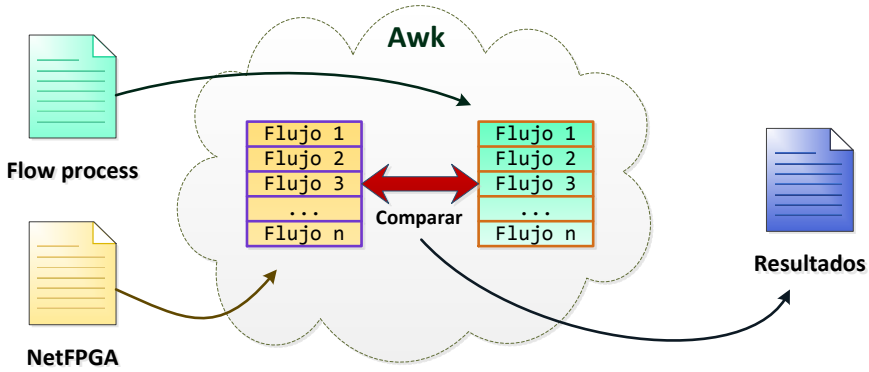

Figura 4.2: Esquema de funcionamiento de los scripts awk.

4.2.2 Elección de una traza

La elección de los datos para realizar los experimentos es fundamental. Para la realización de las pruebas se ha decidido escoger una traza pública fácilmente accesible por la comunidad investigadora.

SimpleWeb [26] es un sitio web creado por el DACS group (Design and Analysis of Communication Systems) de la universidad de Twente, Alemania. En él se pueden encontrar numerosas trazas capturadas en diversas redes con diferentes características.

La NetFPGA dispone de cuatro interfaces Ethernet de 1 Gbps, no obsante y como se verá en los siguientes apartados, el módulo NetFlow no es capaz de alcanzar dicha velocidad de trabajo debido a las limitaciones de memoria. Ya que el objetivo del proyecto es trabajar en redes de alta velocidad aplicando un muestreo a los paquetes analizados, se necesita una traza cuya velocidad sea elevada para el módulo. Aunque esto suponga unas pérdidas iniciales altas, **se pretende verificar la hipótesis de que estas pérdidas no serán tan significativas después de realizar el muestreo**.

En la traza elegida se ha capturado el tráfico en un enlace Ethernet de 300 Mbps (3 troncales de 100 Mbps) que conecta una red de la zona residencial de una universidad con el núcleo de la red de la propia universidad. En la zona residencial se conectan unos 2000 estudiantes, cada uno con un enlace Ethernet de 100 Mbps. La propia red de la zona residencial consiste en enlaces de 100 y 300 Mbps conectados a varios switches, dependiendo del nivel de agregación. El enlace utilizado tiene una media de carga del 60%. Las medidas fueron tomadas en julio de 2002 [27].

4.3 Pruebas y resultados

4.3.1 Validación del módulo NetFlow

En esta sección se realizarán una serie de pruebas para comprobar el correcto funcionamiento del módulo NetFlow, es decir, para verificar que los flujos extraídos por la NetFPGA se corresponden con los flujos de la traza enviada.

Pérdidas y limitación de la velocidad

El módulo NetFlow sobre la NetFPGA introduce una serie de pérdidas debido a que no es capaz de procesar todos los paquetes y flujos recibidos. Como ya se comentó en el apartado 3.2.3, estas péridas son debidas a que el módulo sólo es capaz de almacenar 4096 flujos concurrentes. Si llega un paquete de un flujo que no está presente en memoria y la memoria está llena el paquete se descarta.

En los siguientes apartados se caracterizan y cuantifican las pérdidas producidas durante la realización de las pruebas.

Por otro lado, se ha detectado una limitación de la velocidad de envío de los paquetes a la NetFPGA. La herramienta utilizada para replicar las trazas y mandarlas a la NetFPGA es el *tcpreplay*. Se ha comprobado que el equipo que usa el *tcpreplay* no es capaz de procesar la traza y enviar los paquetes a la misma velocidad con la que se han capturado.

Este efecto produce una pérdidad del sincronismo entre la traza original y la traza enviada a la NetFPGA. Ya que el *flow process* procesa la traza original, dicha pérdida de sincronismo produce que los flujos a comparar sean diferentes.

Para solucionar el problema se ha utilizado un TAP. Un TAP es un dispositivo hardware que se introduce en un enlace Ethernet entre dos máquinas y replica el tráfico recibido en un sentido (o ambos) a un tercer dispositivo que se encuentra a la escucha.

La configuración usada se muestra en la figura 4.3. Desde el PC origen se ejecuta el *tcpreplay* y se manda la traza al TAP. El TAP copia la traza y la envía tanto a la NetFPGA como a la tarjeta de red del PC destino. La NetFPGA exporta los flujos al PC origen a través del TAP mientras que el PC destino ejecuta un *tcpdump*. El *tcpdump* crea una nueva traza con los mismos paquetes que la traza original pero modifica los tiempos de llegada de los paquetes de forma que se corresponden con los tiempos de llegada a la NetFPGA.

Por último se ejecuta el *flow process* sobre la traza capturada por el PC destino. De esta forma los tiempos extraídos por la NetFPGA y el *flow process* son los mismos y se soluciona

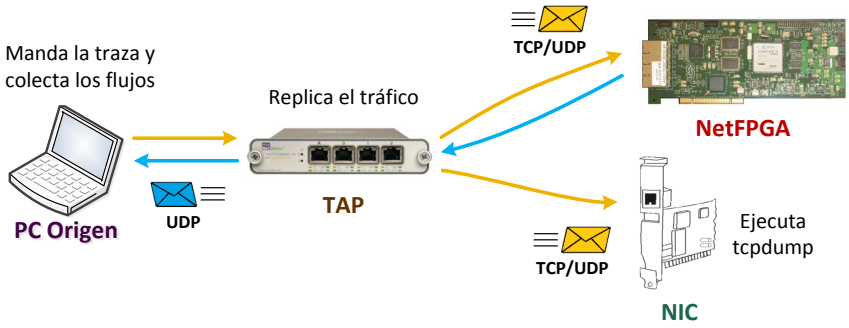

Figura 4.3: Esquema del proceso de captura incluyendo el TAP. La NetFPGA y la NIC se encuentran en el mismo equipo.

el problema de pérdida del sincronismo.

Cuantificación de las pérdidas

El objetivo de esta prueba es comprobar el correcto funcionamiento del módulo NetFlow. Se han seleccionado trazas de diferentes tamaños, formadas por conjuntos de paquetes de la traza de prueba, y se han extraído sus flujos.

Los *timeout* seleccionados han sido: tres segundos para el *inactive timeout* (un flujo expira si han transcurrido tres segundos sin recibir ningún paquete) y no se ha fijado ningún *active timeout*, de forma que un flujo no expira mientras estén llegando paquetes por mucho tiempo que esté activo.

Usando un script awk se procesan los flujos extrayendo su tiempo de inicio, tiempo de finalización, número de paqutes y número de bytes. Los tiempos se guardan en un formato relativo al comienzo de la traza, es decir, se almacena el número de segundos desde la llegada del primer paquete. De esta forma se pueden comparar los tiempos de la NetFPGA y del *flow process*, inicialmente en diferente formato.

Por último, en el mismo script awk se comparan los flujos. Dos flujos se identifican como el mismo si coinciden en su quintupla, su tiempo de inicio y su tiempo de finalización. Debido a las pérdidas producidas por la NetFPGA se ha introducido un margen de error para los tiempos de inicio y finalización, de forma que, si la diferencia entre los tiempos de un flujo extraídos por la NetFPGA y por el *flow process* no superan dicho margen, estos flujos se consideran el mismo (Ver 4.4).

Los parámetros extraídos por el script awk como resultado de la comparación son:

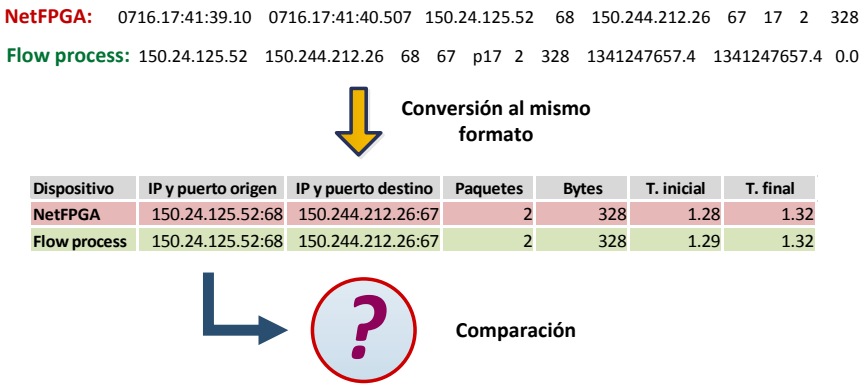

Figura 4.4: Para poder comparar dos flujos primero se convierten al mismo formato.

Total *flow process*: número total de flujos extraídos por el *flow process*.

Total NetFPGA: número total de flujos extraídos por la NetFPGA.

Idénticos: número de flujos que coinciden en todos los campos extraídos.

***Flow process* con pérdidas:** número de flujos que coinciden tanto en tiempo de inicio como de finalización pero se ha perdido algún paquete en el *flow process* por lo que no coincide su número de paquetes ni de bytes.

NetFPGA con pérdidas: número de flujos que coinciden en tiempo de inicio y de finalización pero se ha perdido algún paquete en la NetFPGA por lo que no coincide su número de paquetes ni de bytes.

Perdidos por *flow process*: número de flujos que se han capturado en la NetFPGA pero no en el *flow process*

Perdidos por NetFPGA: número de flujos que se han capturado en el *flow process* pero no en la NetFPGA.

Resultados validación del módulo NetFlow

Traza con cien paquetes

La primera traza utilizada contiene pocos paquetes con el objetivo de evitar las pérdidas. Para ello se han seleccionado cien paquetes de la traza inicial, de forma que la mayoría de ellos pertenecen a flujos diferentes.

Los resultados obtenidos son:

Parámetro	Flujos	Porcentaje
Total *flow process*	76	100%
Total NetFPGA	75	98.69%
Idénticos	75	98.69%
Flow process con pérdidas	0	0%
NetFPGA con pérdidas	0	0%
Perdidos por *flow process*	1	1.31%
Perdidos por NetFPGA	0	0%

Tabla 4.1: Resultados de la traza con cien paquetes, el 100% de los flujos se corresponde con los flujos vistos por el flow process.

La primera diferencia que encontramos en los parámetros registrados para esta traza es el número total de flujos extraídos: el *flow process* a detectado un flujo que no ha sido detectado por la NetFPGA.

Para estudiar este fenómeno se obtiene la quintupla de dicho flujo: `150.244.255.252 150.244.212.26 68 67 p17`. Si se presta atención a las direcciones IP, se detecta que éstas se corresponden con IPs de la propia U.A.M. De hecho, son las IPs asignadas a los PCs origen y destino desde donde se están realizando las pruebas.

Recordemos que para realizar las pruebas se envía una traza (original) a la NetFPGA pero se realiza una copia de ella (réplica) usando un TAP y el *tcpdump*. Abriendo ambas trazas se detecta que el paquete que ha originado dicho flujo está en la traza réplica pero no en la original. Esto es debido a que el PC origen no sólo manda los paquetes contenidos en la traza de prueba si no que además inserta otros paquetes, por ejemplo peticiones DNS [1].

Estos paquetes también se capturan con la NetFPGA, no obstante, algunos de ellos (una minoría) se pierden en el proceso de captura. El orden de ejecución del proceso de captura es:

1.– Comienza el proceso de captura con la NetFPGA.

2.– Se lanza el *tcpdump* en el equipo destino.

3.– Se lanza el *tcpreplay* en el equipo origen.

4.– Finaliza el envío de la traza.

5.– Se para el proceso de captura con el *tcpdump*.

6.– Se para el proceso de captura con la NetFPGA.

Al cominezo y al final del proceso de captura, entre los pasos 1-2 y 5-6, no se está ejecutando el *tcpdump*, sin embargo, la NetFPGA está capturando paquetes. En estos puntos los paquetes generados por el PC origen (que no se encuentran en la traza original) son captura-

[1] Se ha intentado evitar la captura de estos paquetes pero no ha sido posible.

dos por la NetFPGA pero no por el *tcpdump*.

La diferencia de paquetes capturados explica la diferencia entre los flujos vistos por la NetFPGA y por el *flow process*, y en consecuencia, la pérdida de flujos detectada en el *flow process*.

El otro resultado destacable en esta prueba es que **todos los flujos capturados por la NetFPGA son idénticos a los capturados por el *flow process*** por lo que se deduce que la NetFPGA extrae los flujos de forma correcta siempre y cuando no haya pérdidas (dos flujos son idénticos son idénticos si tienen la misma quintupla, el mismo número de paquetes y bytes, y el mismo tiempo de inicio y finalización).

Traza de 15 mil paquetes

En la siguiente prueba se ha seleccionado una traza con un número mayor de paquetes y de flujos pero lo suficientemente pequeña para que las pérdidas sean casi nulas. El archivo pcap de la traza de prueba seleccionada tiene aproximadamente 46 millones de paquetes de los cuales se han usado los 15 mil primeros.

Los resultados obtenidos se muestran en la tabla 4.2.

Parámetro	**Flujos**	**Porcentaje**
Total *flow process*	2161	100%
Total NetFPGA	2137	98.89%
Idénticos	2129	98.52%
Flow process con pérdidas	0	0%
NetFPGA con pérdidas	7	0.32%
Perdidos por *flow process*	1	0,05%
Perdidos por NetFPGA	25	1.11%

Tabla 4.2: Resultados de la traza de 15 mil paquetes.

Se resaltan tres resultados: los flujos idénticos, los flujos con pérdidas y los flujos perdidos.

El 98.52% de los flujos son idénticos. Con este resultado, debido a que en esta prueba se ha empleado un número mayor de flujos, se da por correcto el funcionamiento del módulo NetFlow, salvo por los efectos de las pérdidas. Es decir, el módulo NetFlow extrae de forma correcta la información de los flujos aunque se produce una pérdida de paquetes.

También se observa que el 0.32% de los flujos han perdido algún paquete, detectando aquí el primer efecto de las pérdidas. De los flujos con pérdidas se han obtenido el número de paquetes y bytes totales y el número de bytes y paquetes perdidos (tabla 4.3).

Por último, se pueden ver los primeros flujos perdidos en la NetFPGA, el 1.16%. Además, hay un flujo perdido por el *flow process*, igual que en la traza anterior.

Parámetro	Totales	Perdidos	Porcentaje
Paquetes	105	19	18.09%
Bytes	135557	14080	10.38%

Tabla 4.3: Paquetes y bytes perdidos en los flujos con pérdidas, traza de 15 mil paquetes.

Traza de siete millones de paquetes

En esta prueba se ha seleccionado una traza con un número de flujos elevados. Se han usado aproximadamente siete millones de paquetes de la traza de prueba.

Se ha eliminado el parámetro *flow process* con pérdidas debido a que siempre es el 0%. Los resultados son:

Parámetro	Flujos	Porcentaje
Total *flow process*	142640	100%
Total NetFPGA	82681	57.96%
Idénticos	31605	21.94%
NetFPGA con pérdidas	49670	34.48%
Perdidos por *flow process*	1406	0.98%
Perdidos por NetFPGA	61365	42.6%

Tabla 4.4: Resultados de la traza de siete millones de paquetes.

En esta prueba se ven los efectos de las pérdidas. El 22.16% de los flujos son correctos, el 34.4% tienen pérdidas y el 42.46% se han perdido. Debido a las limitaciones de memoria del módulo NetFlow las pérdidas son elevadas. Dentro de los fines de este proyecto final de carrera no se incluye el objetivo de mejorar estas pérdidas, por lo tanto sólo se ha realizado una cuantificación de las mismas.

Los paquetes y bytes perdidos en los flujos con pérdidas son:

Parámetro	Totales	Perdidos	Porcentaje
Paquetes	5383135	1021126	18.96%
Bytes	3794288066	587577753	15.48%

Tabla 4.5: Paquetes y bytes perdidos en los flujos con pérdidas, traza de 500 Mbits.

Traza de 46 millones de paquetes

Por último se ha pasado la prueba con la traza completa. Los resutlados obtenidos se muestran en la tabla 4.6.

El número de flujos idénticos es menor que en la prueba anterior, 15.47% frente al 21.94%. Esta disminución de flujos conlleva el consecuente aumento de flujos con pérdidas, 37.74%

Parámetro	Flujos	Porcentaje
Total *flow process*	869710	100%
Total NetFPGA	480877	55.29%
Idénticos	136306	15.47%
NetFPGA con pérdidas	332719	37.74%
Perdidos por *flow process*	11852	1.34%
Perdidos por NetFPGA	400685	45.45%

Tabla 4.6: Resultados de la traza de 46 millones de paquetes.

frente al 34.48%, y perdidos, 45.45% frente al 42.6%. Aunque en esta traza se han obtenido los peores resultados, el aumento de las pérdidas frente a la traza de siete millones de paquetes no es muy grande.

Los paquetes y bytes perdidos en los flujos con pérdidas son:

Parámetro	Totales	Perdidos	Porcentaje
Paquetes	41412649	29349715960	42.01%
Bytes	17400891	9292514470	31.66%

Tabla 4.7: Paquetes y bytes perdidos en los flujos con pérdidas, traza de 46 millones de paquetes.

En la traza de 46 millones de paquetes se duplican los valores de paquetes y bytes perdidos en los flujos con pérdidas en comparación con la traza de siete millones de paquetes.

4.3.2 Validación del muestreo

Después de implementar el muestreo se han realizado una serie de pruebas para comprobar que el muestreo y la posterior extracción de flujos se realiza de forma correcta.

Descripción de las pruebas

El diseño de las pruebas es muy similar al llevado a cabo para la validación del módulo NetFlow (apartado 4.3.1).

Se elige una traza y se envía a la NetFPGA para extraer sus flujos aplicando un muestreo de paquetes. A su vez, la traza original se copia usando un TAP obteniendo así una traza réplica (como se vió en la figura 4.3). A continuación se extraen los flujos de la traza réplica usando el *flow process* con la misma tasa de muestreo y *offset*.

Una vez obtenidos los flujos extraídos por la NetFPGA y el *flow process* se comparan con el mismo script awk usado en la validación del módulo NetFlow.

Trazas utilizadas y parámetros extraídos

Debido al diseño de las pruebas, si se pierde un paquete cambia el *offset* y los paquetes muestreados a partir de dicho punto ya no son los mismos. Por ello, se han usado trazas en las que no se pierde ningún paquete para realizar la validación del muestreo.

Dos de las trazas utilizadas son las mismas que en la valicación del módulo NetFlow: trazas de cien y 15 mil paquetes. Además se ha introducido una nueva traza con mil paquetes.

El script awk usado extrae los parámetros explicados en el apartado 4.3.1. En esta prueba se mostrarán el número total de flujos vistos tanto por la NetFPGA como por el *flow process* y el número total de flujos correctos.

Resultados validación del muestreo

En la primera prueba se han utilizado las dos trazas con menos paquetes (trazas de cien y mil paquetes) para observar más fácilmente los flujos extraídos.

Recordemos que, al realizar el proceso de captura, se generan flujos nuevos en el *flow process* que no aparecen en la NetFPGA. Dichos flujos sólo se encuentran al principio y al final de la captura, de modo que, para evitar capturarlos al comienzo de la traza, se ha ajustado el *offset* del *flow process* para que ambos dispositivos empiecen a muestrear en el mismo paquete.

Se han aplicado diferentes tasas de muestreo a las trazas seleccionadas. Los resutlados obtenidos se muestran en la tabla 4.8.

Traza	Muestreo	NetFPGA	*Flow process*	Idénticos	Idénticos(%)
100 paquetes	10	10	10	10	**100**
1000 paquetes	2	336	338	336	**99.4**
1000 paquetes	3	244	244	244	**100**
1000 paquetes	5	162	162	162	**100**
1000 paquetes	10	86	87	86	**98.85**
1000 paquetes	15	58	58	58	**100**
1000 paquetes	25	37	37	37	**100**
1000 paquetes	100	10	10	10	**100**

Tabla 4.8: Resultados del muestreo de la trazas con cien paquetes y mil paquetes. El parámetro muestreo es la tasa de muestreo. Los parámetros NetFPGA y flow process son el número total de flujos vistos por cada unos de ellos respectivamente.

En traza de mil paquetes, para las tasas de muestreo de dos y diez, se han detectado tres flujos perdidos por la NetFPGA. Dichos flujos se generan debido a los paquetes insertados en el proceso de captura que son vistos por el *flow process* pero no por la NetFPGA. Como ya se ha comentado, ajustando el *offset* en el *flow process* se consiguen eliminar los paquetes

insertados al principio de la traza, no obstante, no ocurre lo mismo con los paquetes insertados al final. Si se desprecian estos tres flujos se puede ver que el proceso de muestreo y extracción de flujos se realiza de forma correcta.

Se ha creído conveniente repetir el experimento analizando una traza con mayor número de paquetes. Para esta segunda prueba, los resultados de la traza de 15 mil paquetes pueden observarse en la tabla 4.9.

Traza	Muestreo	NetFPGA	*Flow process*	Idénticos	Idénticos(%)
15 mil paquetes	10	825	826	825	**99.88**
15 mil paquetes	25	420	420	420	**100**
15 mil paquetes	100	134	134	134	**100**

Tabla 4.9: Resultados del muestreo de la traza de 15 mil paquetes.

Los resultados obtenidos son los mismos que para las trazas de cien y mil paquetes, es decir, **el proceso de muestreo y extracción de flujos se realiza de forma correcta**. Se valida así el funcionamiento del muestreo implementado en el módulo NetFlow.

4.3.3 Repercusión de las pérdidas

Durante la validación del módulo NetFlow se han cuantificado las pérdidas producidas por las limitaciones de memoria en la FPGA.

En el apartado 4.2.2 se planteó la hipótesis de que al aplicar el muestreo las repercusiones de dichas pérdidas son menores. Para comprobar esta hipótesis se ha realizado una prueba. Se ha creado un script en bash que lanza el proceso de captura en la NetFPGA, replica una traza y almacena el número de paquetes perdidos. El script repite el proceso para todas las tasas de muestreo entre uno y cien, almacenando el número de paquetes perdidos para cada una de ellas.

Se han sometido a esta prueba las trazas de siete millones de paquetes y la traza original de 46 millones de paquetes. En la figura 4.5 se representa el número de paquetes perdidos frente a la tasa de muestreo para cada una de las trazas. En ella se puede observar que las pérdidas decrecen muy rápidamente según aumenta la tasa de muestreo, especialmente para las tasas entre uno y diez.

Con el fin de apreciar mejor las pérdidas, en la figura 4.6 se muestran a partir de una tasa de muestreo de veinte. Para la tasa de 30, en la traza de 46 millones de paquetes, las pérdidas son menores que el $6,6 \cdot 10^{-3}$ % de los paquetes, mientras que sin muestrear son el 4.13 % . **Se confirma la hipótesis** de que al aplicar el muestreo las repercusiones de las pérdidas son mucho menores.

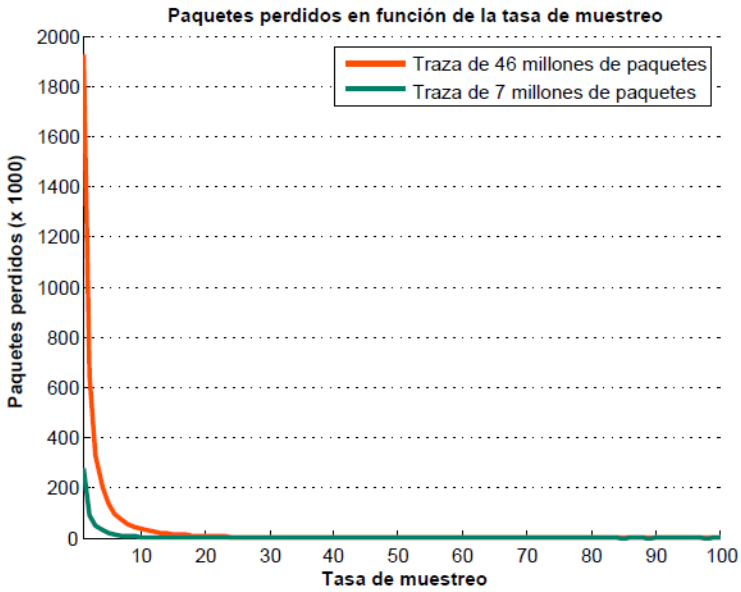

Figura 4.5: Paquetes perdidos en función de la tasa de muestreo.

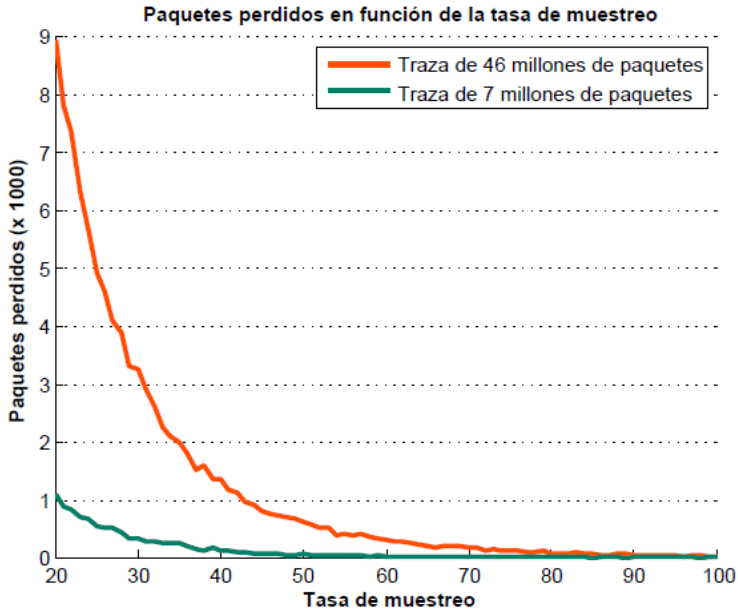

Figura 4.6: Paquetes perdidos para una tasa de muestreo superior a veinte.

4.3.4 Muestreo distribuido

A lo largo del capítulo se ha expuesto que las pérdidas suponen el principal inconveniente del módulo NetFlow. Se ha demostrado que al aplicar muestreo las repercusiones de las mismas no son tan severas. Recordemos que las pérdidas se producen porque el módulo no es capaz de almacenar más de 4096 flujos concurrentes.

Se han propuesto dos soluciones posibles a este problema aunque cada una de ellas implica unas limitaciones:

- Bajar la velocidad a la que se envía la traza: disminuye el número de flujos concurrentes, sin embargo, se contrapone al objetivo de alcanzar la mayor velocidad de trabajo posible dentro de las limitaciones.
- Aplicar muestreo: se capturan menos paquetes y, en consecuencia, menos flujos. Ésta solución es mejor que la anterior aunque introduce un error de medida propio del muestreo.

Como último experimento evaluará la aplicación de muestreo distribuido. Consiste en desplegar varias tarjetas NetFPGA en diferentes puntos de la red por los que el tráfico sea muy similar. Cada NetFPGA aplicará un muestreo y enviará la información a un colector. El colector se encargará de agregar los flujos que caracterizarán el tráfico de la red (ver figura 4.7).

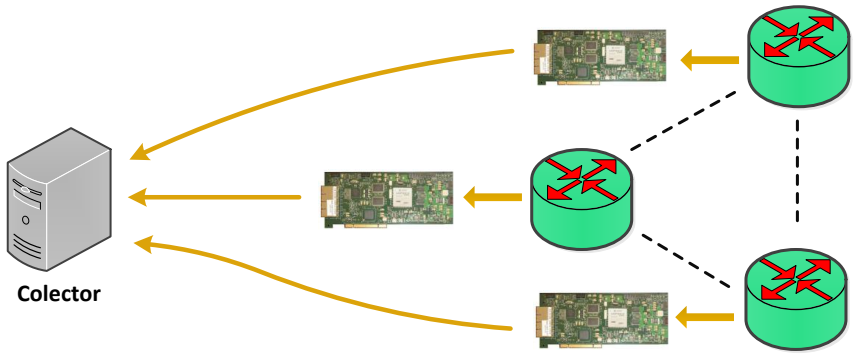

Figura 4.7: Muestreo distribuido. El colector agrega los flujos.

Con esta solución se pretende reducir aún más los efectos de las perdidas de forma que sea posible capturar flujos en redes de mayor velocidad o aplicar un muestreo menor.

Justificación del muestreo distribuido

El objetivo de la extracción de flujos de una red es la caractirzación del tráfico dentro de la misma. La traza elegida en las pruebas de este proyecto se corresponde con una arquitectura bastante común, en la cual se unen dos subredes a través de un troncal de gran capacidad. En contreto, en esta red, se une la zona residencial de una universidad con el *core* de la propia universidad (figura 4.8).

Para caraterizar el tráfico de dicha red, se aplicará un muestreo en cada uno de los bordes del troncal que los une y se enviarán los flujos resultantes a un colector encargado de agregarlos.

Se pretende comprobar si los resutlados de muestrear en dos puntos con una tasa de muestreo λ y diferente *offset* es equivalente a muestrar en un punto con una tasa de muestreo de 2λ. Es decir, ¿es equivalente muestrear un paquete de cada 25 a muestrear un paquete de cada 50 en dos puntos diferentes?

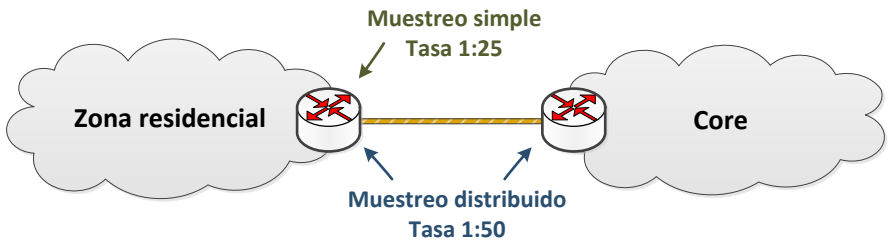

Figura 4.8: Muestreo distribuido aplicado a la red de la traza de prueba.

Por último se ampliará el experimento hipotetizando el caso de que hubiese varios saltos intermedios en el troncal, de forma que sea posible muestrear en tres y cuatro puntos diferentes, es decir con varios routers intermedios (4.9).

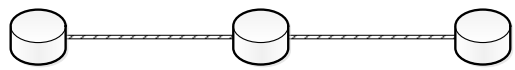

Figura 4.9: Red con varios saltos intermedios.

Agregación del muestreo

Es necesario definir una política de agregación de flujos ya que los resultados finales dependerán en gran medida de ella. Si dos NetFPGA detectan el mismo flujo muestreando diferentes paquetes, el colector encargado de realizar la agregación debe definir cómo será el flujo final resultado de unificar ambos.

En los experimentos realizados se ha trabajado con los siguientes campos de cada flujo: número de paquetes, número de bytes, tiempo de inicio y tiempo de finalización. La forma de agreción de cada uno de ellos es diferente:

- Tiempo de inicio: el tiempo de inicio del flujo resultado será el menor tiempo de inicio de los flujos agregados.
- Tiemo de finalización: el tiempo de finalización del flujo resultado será el mayor tiempo de finalización de los flujos agregados.
- Número de paquetes y número de bytes: el número de paquetes y número de bytes finales será la suma del número de paquetes y bytes de los flujos agregados, dividido por el número de puntos totales en los que se ha muestreado y multiplicado por la tasa de muestreo.

$$N_{final}(paquetes\ o\ bytes) = tasa\ de\ muestreo \cdot \frac{Suma(paquetes\ o\ bytes)}{N_{puntos\ muestreados\ totales}} \tag{4.1}$$

En la tabla 4.10 se muestrea un ejemplo de flujo agregado donde la tasa de muestreo es de diez. Se ha muestreado en tres puntos diferentes pero el flujo sólo ha sido detectado por dos de ellos.

Flujo	Tiempo inicio	Tiempo finalización	Paquetes	Bytes
Flujo 1	1.2	3.2	5	2000
Flujo 2	1	3.1	4	1000
Unificado	1	3.2	30	10000

Tabla 4.10: Ejemplo de flujo agregado. Se ha muestreado en tres puntos diferentes con una tasa de muestreo de diez

Comparación de los resultados

En este experimento se han obtenido los flujos de la traza de prueba simulando dos escenarios:

- Muestreo simple.

- Muestreo distribuido más agregación.

Como resultado se obtienen dos archivos, uno con los flujos del muestreo simple y otro con los del distribuido. Para caracterizar cada uno de ellos se extraen las **funciónes densidad de probabilidad** de la duración de los flujos, número de paquetes y número de bytes. Además también se extraen dichas funciones para los flujos de la traza original, sin aplicar muestreo ni pérdidas.

A continuación se comparan las funciones de densidad de probabilidad de las trazas muestreadas con las de la traza sin muestrear. Para comparar cada una de las funciones de densidad de probabilidad de las trazas muestreadas con su correspondiente función de la traza sin muestrear se usa la *Distancia de Hellinger* (figura 4.10).

Distancia de Hellinger

La medida elegida para calcular la simulitud de dos funciones densisad de probabilidad es la Distancia de Hellinger. Se ha elegido esta distancia frente a otras alternativas evaluadas, como la divergencia de Kullback-Leibler, porque la distancia de Hellinger es, como su propio nombre indica, una distancia, y por lo tanto cumple propiedades deseables, como por ejemplo la simetría $d(p,q) = d(q,p)$.

Usada por D. Tammaro y S. Valentinni [28] en su estudio titulado *"Exploiting packet sampling measurements for traffic characterization and classification"*, la *Distancia de Hellinger*, definida en la ecuación 4.2, compara dos funciones densidad de probabilidad y da un valor acotado entre cero y uno donde los valores más cercanos a cero se corresponden a distribuciones con mayor similitud.

$$HD(p,q) = \sqrt{1 - \sum_{x \in X} \sqrt{p(x) \cdot q(x)}} \tag{4.2}$$

Siendo $p(x)$ y $q(x)$ las funciones densidad de probabilidad a comparar.

La *distancia de Hellinger* da un valor entre la similitud de dos distribuciones. Para poder comparar los resultados obtenidos con los dos tipos de muestreo se realizan los siguientes pasos:

1.– Calcular la *distancia de Hellinger* entre las distribuciones obtenidas aplicando muestreo simple y las distribuciones sin aplicar muestreo.

2.– Calcular la *distancia de Hellinger* entre las distribuciones obtenidas aplicando muestreo distribuido y las distribuciones sin aplicar muestreo.

3.– Comparar ambas distancias.

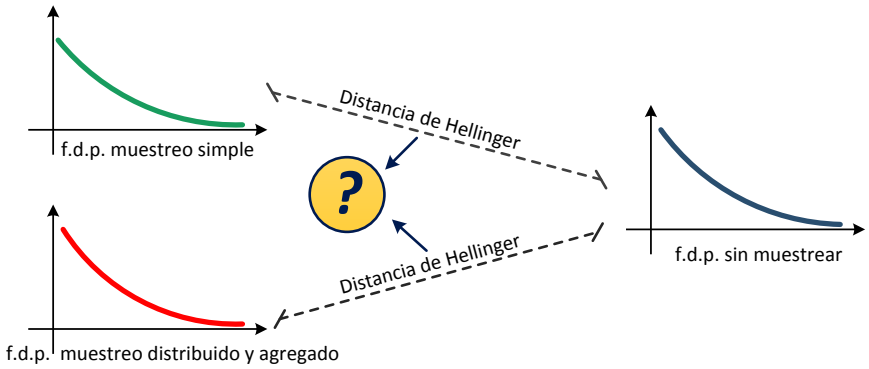

Figura 4.10: Comparación de dos f.d.p. usando la distancia de Helinger.

Resultados

Debido a que la traza de prueba proviene de un solo router y sólo se dispone de una NetFPGA, se ha simulado el muestreo distribuido. La simulación consiste en aplicar varios muestreos a la traza, con la misma tasa de muestreo y diferente *offset*, agregando cada uno de ellos como si correspondiese a un router diferente.

Se han realizado cinco repeticiones del experimento descrito, en las cuales se han comparado los muestreos mostrados en la tabla 4.11.

Muestreo simple	Muestreo distribuido
25	2 x 50
50	2 x 100
10	3 x 30
33	3 x 100
25	4 x 100

Tabla 4.11: Tasas de muestreo distribuido usadas en el experimento.

En la tabla 4.12 se muestra la *distancia de Hellinger* entre cada una de las funciones de densidad de probabilidad (f.d.p.) de cada muestreo y de la traza original.

Los resultados del experimento muestran una pequeña mejora para el muestreo distribuido respecto al simple debido a que las pérdida son menores para una tasa de muestreo mayor.

Se consigue de esta forma el objetivo final de este experimento que consiste en disminuir

	Distancia de Hellinger		
f.d.p. / **Muestreo**	**Duración**	**Número paquetes**	**Número bytes**
25	0.2383	0.1373	0.1164
2x50	0.2204	0.1234	0.0987
50	0.2814	0.1489	0.1242
2x100	0.2468	0.1320	0.1086
10	0.2286	0.2283	0.2707
3x30	0.1968	0.1699	0.2275
33	0.2435	0.1392	0.1158
3x100	0.2326	0.1207	0.0983
25	0.2383	0.1373	0.1164
4x100	0.2236	0.1114	0.0899

Tabla 4.12: Resultado de hallar la Distancia de Hellinger.

los efectos de las pérdidas aplicando muestreo distribuido. Esta mejora implica que, mientras sea posible aplicar muestreo distribuido, se obtendrán los siguientes beneficios:

1.– Se podrá **aumentar la velocidad de captura** de la NetFPGA por lo que será posible trabajar en redes con mayor velocidad.

2.– Se podrá **trabajar con tasas de muestreo menores**, disminuyendo así la imprecisión introducida por el muestreo. Por ejemplo, si la tasa mínima aceptable para la NetFPGA en una determinada red es de 30, muestreando en dos puntos de la red se obtendrá una tasa de muestreo final equivalente a 15.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

A lo largo del proyecto se ha implementado un sistema capaz de extraer registros NetFlow, aplicando la técnica de muestreo determinista, usando la plataforma NetFPGA. Además se ha realizado una evaluación de sus prestaciones y se han propuesto soluciones que permiten mejorar su rendimiento.

Se ha elegido la NetFPGA ya a que se trata de una plataforma de desarrollo de bajo coste apropiada tanto para el ámbito de investigación como para el empresarial. Permite la construcción de prototipos de alta velocidad, acelerados por hardware, fácilmente reconfigurables.

El punto de partida para el proyecto ha sido el módulo NetFlow implementado por Martin Zadnik. El proyecto, disponible para la plataforma NetFPGA, es capaz de extraer registros NetFlow y exportar la información de los flujos. Se ha hecho un estudio en profundidad del módulo y de cada uno de los submódulos que lo forman. La información se procesa y transmite a través de un pipeline con diferentes etapas:

1.– Se reciben los paquetes por una interfaz Ethernet y se extrae la información útil.

2.– Se inserta una marca de tiempo.

3.– Se genera un código hash que se usará para identificar el flujo.

4.– Se almacena el flujo en memoria.

5.– El flujo expira y se extrae de memoria.

6.– Se encapsula el flujo en un paquete UDP junto con otros flujos.

7.– Se exporta el paquete UDP a un colector usando la interfaz Ethernet.

Durante la instalación de la NetFPGA y del módulo NetFlow surgieron problemas en los que se invirtió tiempo y esfuerzo. No obstante, se ha generado la documentación necesaria para realizar la instalación de forma eficiente y se ha conseguido repetirla de forma correcta.

Se realizaron una series de pruebas para evaluar el módulo NetFlow y comprobar su correcto funcionamiento. Las herramientas usadas fueron las *flow tools* y el *flowprocess* (apartado 4.2.1) para la obtención de resultados, y scripts Awk para su comparación.

El principal problema encontrado fue la limitación de memoria. Debido a que la memoria

donde se alacenan los flujos se implementa como blockRAMs, el módulo sólo es capaz de almacenar 4096 flujos concurrentes. Cuando se recibe un paquete de un flujo que no se encuentra presente en memoria y ésta está llena, el paquete se descarta. Este comportamiento da lugar a una pérdida de paquetes y flujos que no serán registrados por la NetFPGA.

Para evaluar el impacto de las pérdidas se realizó una cuantificación de las mismas. En un enlace de 300 Mbps con una media de carga del 60 % se obtiene una pérdida del 45 % de los flujos.

Después de comprobar el correcto funcionamiento del módulo NetFlow y de cuantificar las pérdidas se implementó el muestreo. Para ello se modificaron algunos de los módulos de la NetFPGA así como los scripts de configuración de la misma (escritos en Perl y C) que se ejecutan desde el PC anfitrión.

Para comprobar el correcto funcionamiento del muestreo se realizaron las mismas pruebas que en la cuantificación de las pérdidas, haciendo algunas modificaciones. Debido a la inserción de paquetes no deseados y a una pérdida del sincronismo, se usó un TAP (elemento que recibe tráfico por una interfaz Ethernet y lo replica por dos interfaces de salida). Se modificó el *flow process*, añadiendo la funcionalidad de muestreo, y se usaron los mismos scripts awk.

Una vez comprobado el correcto funcionamiento del muestreo se planteó un experimento para comprobar la hipótesis de que, tras aplicar el muestreo, las repercusiones de las pérdidas son mucho menores. Para ello se midió el número de paquetes perdidos en función de la tasa de muestreo. Los resultados confirmaron la hipóteis, mostrando una reducción elevada de las pérdidas al aumentar la tasa de muestreo.

Por último se evaluó el uso de muestreo distribuido como una forma de reducir aún más los efectos de las péridas. Se hizo un experimento en el cual se comparaban los resultados de aplicar muestreo simple y muestreo distribuido más agragación a la misma traza. Se comprobó que el muestreo distribuido obtenía una ligera mejora respecto al simple debido a una reducción de las pérdidas. Con estos resultados se deduce que, al aplicar muestreo distribuido, será posible aumentar la velocidad de captura de la NetFPGA y trabjar con tasas de muestreo menores.

En conclusión, a lo largo de este Proyecto Fin de Carrera se ha conseguido instalar la plataforma NetFPGA con el módulo NetFlow, se ha visto que el mayor problema del mismo son las pérdidas de paquetes y flujos debido a las limitaciones de memoria. Se ha implementado un muestreo determinista como un método de reducir las pérdidas que permite trabajar en redes de mayor velocidad. Finalmente se han usado técnicas de muestreo distribuido y agregación del tráfico que permiten aumentar aún más la velocidad de trabajo.

5.2 Trabajo futuro

Como ya se ha visto a lo largo de todo el proyecto, la NetFPGA y el módulo NetFlow funcionan correctamente. Se ha conseguido crear un sistema por el cual es posible aplicar tanto un muestreo siemple, como un muestreo más agregación, de forma óbtima. No obstante, si se desea realizar un despligue para monitorizar redes en servicio es necesario realizar algunas mejoras.

En primer lugar conviene actualizar el módulo NetFlow a la última versión de la NetFPGA para que el proceso de instalación y puesta en marcha se puedan realizar fácilmente y de forma automática.

A continuación es necesario hacer un estudio más amplio de la NetFPGA y del módulo NetFlow, para evaluar su funcionamiento en diversos escenarios con diferentes características. Aunque se ha hecho una cuantificación de las pérdidas, éstas se han medido en un troncal con una velocidad de 300 Mbps. Sería conveniente cuantificarlas para diferentes velocidades de línea y distintas tasas de muestreo.

Por último, se podría introducir el uso de las memorias externas de la NetFPGA para almacenar los flujos activos, en vez de implementar la memoria como blockRAMs dentro de la FPGA. Con esta mejora se podría almancear un número mucho mayor de flujos concurrentes por lo que se reducirían las pérdidas de forma notable. No obstante debe recordarse que existe un compromiso entre la velocidad de trabajo y la memoria utilizada ya que la memorias externas a la FPGA son más lentas que los blockRAMs implementados dentro de la misma.

Otra línea de trabajo es modificar el tipo de muestreo aplicado en el módulo NetFlow. Una alternativa al muestreo de paquetes consiste en el muestreo de flujos [29]. Mediante el uso de la función hash, que identifica una quintupla de forma única, se puede muestrear un rango de flujos. De esta forma es posible aplicar técnicas de muesteo distribuido y agregación donde cada NetFPGA sólo muestree un conjunto de flujos.

El módulo NetFlow está implementado en la NetFPGA de 1 Gbps. Aunque se consiga optimizar el módulo y mejorar sus prestaciones, la velocidad máxima de trabajo que se puede alcanzar, limitada por las interfaces Ethernet, es de 1 Gbps. Para alcanzar mayores velocidades se debe crear el módulo NetFlow para la NetFPGA de 10 Gbps. Esta tarjeta está provista de interfaces de fibra óptica por lo que es posible trabajar a velocidades muy superiores. Además la FPGA integrada tiene mejores características así como el resto del hardware.

Bibliografía

[1] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational ip networks: methodology and experience," *IEEE/ACM Trans. Netw.*, vol. 9, pp. 265–280, June 2001. 1.1

[2] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," *SIGCOMM Comput. Commun. Rev.*, vol. 35, pp. 217–228, Aug. 2005. 1.1

[3] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. C. Diot, "Packet-level traffic measurements from the sprint ip backbone," *Netwrk. Mag. of Global Internetwkg.*, vol. 17, pp. 6–16, Nov. 2003. 1.1

[4] M.-S. Kim, Y. J. Won, and J. W. Hong, "Characteristic analysis of internet traffic from the perspective of flows," *Comput. Commun.*, vol. 29, no. 10, pp. 1639–1652, 2006. 1.1

[5] B.-Y. Choi and S. Bhattacharyya, "Observations on cisco sampled netflow," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, pp. 18–23, Dec. 2005. 1.1, 2.2

[6] "Introducción a la netfpga." `http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/LearnMore`. 1.1, 2.3.1

[7] F. G. de Rubalcava Blanca, "Diseño y análisis de técnicas para la mejora de la caracterización de los flujos de red utilizando muestreo distribuido," 2011. 2.1

[8] "Introduction to cisco ios netflow," *Cisco White Papers*, May 2012. 2.1

[9] S. Roming and M. Fullmer, "The osu flow-tools package and cisco netflow logs," pp. 291–304, 2000. 2.1, 2.1, 1, 4.2.1

[10] "Sampled netflow," *Cisco IOS Release 12.0 S*, May 2003. 2.2

[11] G. Watson, N. McKeown, and M. Casado, "Netfpga: A tool for network research and education," 2.3.1

[12] "Tabla de proyectos de la netfpga 1g." `http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/ProjectTable`. 2.3.1

[13] "Reference router." `http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/ReferenceRouterWalkthrough`. 2.3.1, 3.1.2, 3.2.1

[14] "Proyecto openflow con soporte mpls." `http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/ProjectsOpenFlowMPLSSwitch`. 2.3.1

[15] "High performance passive monitoring platform for netfpga." `http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/`

MonitoringSystem. 2.3.1

[16] “Especificaciones netfpga 1g.”
http://www.netfpga.org/php/specs.php. 2.3.1

[17] “Especificaciones netfpga 10g.”
https://github.com/NetFPGA/NetFPGA-10G-empty/wiki. 2.3.1

[18] “Módulo netflow.”
http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/NetFlowProbe. 2.3.2, 4.1.1, 4.1.2

[19] “Probabilidad de colisión hash.”
http://blog.shay.co/hash-collision-probability/. 3.2.3

[20] “Netflow services solutions guide,” *Cisco White Papers*. 3.2.3

[21] “Fedora core with netfpga live cd.”
http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/LiveDVDInstall. 4.1.1

[22] “Versiones antiguas de la netfpga.”
http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/Releases. 4.1.2

[23] “Tutorial de instalación de la netfpga v1.2.”
netfpga.org/netfpgawiki/index.php/GuideInstall_NetFPGA_Base_Package. 4.1.2

[24] “Java developers, oracle.”
http://www.oracle.com/technetwork/java/index.html. 4.1.2

[25] A. Robbins, *The GNU Awk user’s guide*. 2011.
http://www.gnu.org/software/gawk/manual/gawk.html. 4.2.1

[26] “Simpleweb.”
http://www.simpleweb.org/. 4.2.2

[27] “Trazas de simpleweb.”
http://www.simpleweb.org/wiki/Traces. 4.2.2

[28] D. Tammaro, S. Valenti, D. Rossi, and A. Pescapé, “Exploiting packet-sampling measurements for traffic characterization and classification,” *International Journal of Network Management*, pp. n/a–n/a, 2012. 4.3.4

[29] N. Hohn and D. Veitch, “Inverting sampled traffic,” *IEEE/ACM Trans. Netw.*, vol. 14, pp. 68–80, Feb. 2006. 5.2

LISTAS

Lista de códigos

Lista de figuras

Lista de tablas

ACRÓNIMOS

CNET ..... chip de Control de la NetFPGA

CPCI ..... Control del bus PCI

DCMs ..... Digital Clock Managers

FPGA ..... Field Programmable Gate Array

GBPS ..... Gigabits por segundo

HPCN ..... High Performance Computing and Networking

MBPS ..... Megabits por segundo

NETFPGA ..... Networking FPGA

NFP ..... NetFPGA Package

NFPs ..... NetFPGA Packages

NIC ..... Network Interface Card

PCI ..... Peripheral Component Interconnect

ToS ..... Type of Service

DEFINICIONES

blockRAMs

Bloques de memoria implementados dentro de la FPGA. Su velocidad de lectura y escritura es mayor que la de la memoria externa pero consumen recursos de la FPGA.

CFLOW

Paquete UDP que almacenan los registros NetFlow y que se puede exportar a un colector.

colector

Dispositivo (PC o servidor) ubicado en la red para recoger todos los registos NetFlow que son enviados desde los dispositivos de infraestructura (NetFPGA, routers o switches).

GLOSARIO

A

Diagrama de bloques

En la figura A.1 se muestra un diagrama con los bloques de información transmitidos a lo largo del pipeline. Son bloques de 64 bits que se transmiten gracias a un protocolo interno de comunicación que utiliza distintas señales (bus de datos, bus de control, señales de habilitación de lectura, escritura,etc.).

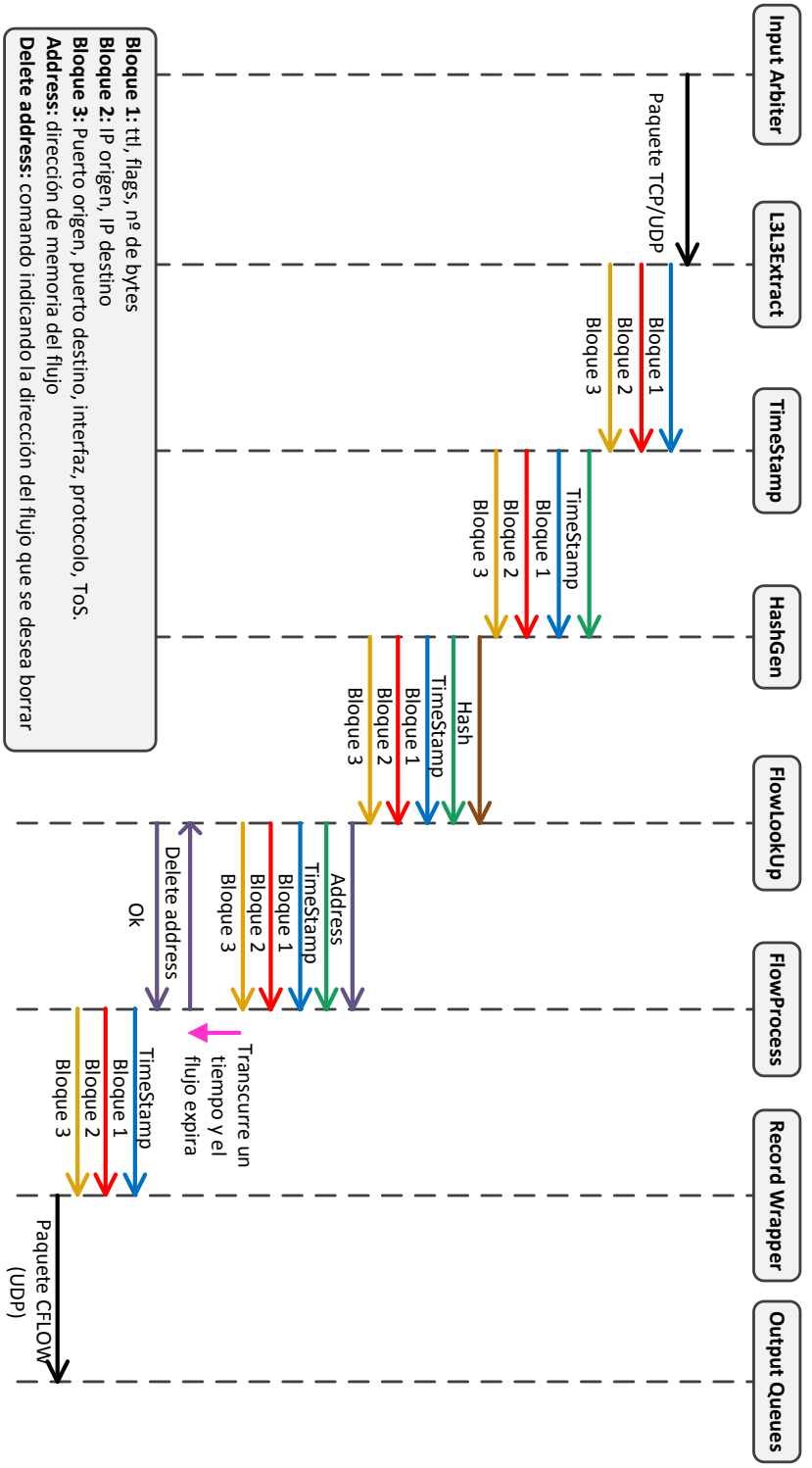

Figura A.1: Diagrama de los bloques transmitidos en el pipeline

B

Añadir registros paso a paso

Antes de implementar el muestreo es necesario añadir registros adicionales de comunicación con la NetFPGA. Posteriormente estos registros se usarán para configurar campos como la tasa de muestreo y proporcionarán una importante herramienta de depuración.

Como ya se explicó en el apartado 3.1.4 los registros se implementan mediante el módulo Generic Regs.

El código Verilog empleado para instanciar el módulo es:

```
1  generic_regs
2  #(
3      .UDP_REG_SRC_WIDTH (UDP_REG_SRC_WIDTH),
4      .TAG (`NETFLOW_L3L4EXTRACT_BLOCK_TAG),
5      .REG_ADDR_WIDTH (`NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH),
           // Width of block addresses -- eg. MODULE_REG_ADDR_WIDTH
6      .NUM_COUNTERS (0), // Number of counters
7      .NUM_SOFTWARE_REGS (4), // Number of sw regs
8      .NUM_HARDWARE_REGS (5) // Number of hw regs
9  ) module_regs (
10     .reg_req_in (reg_req_in),
11     .reg_ack_in (reg_ack_in),
12     .reg_rd_wr_L_in (reg_rd_wr_L_in),
13     .reg_addr_in (reg_addr_in),
14     .reg_data_in (reg_data_in),
15     .reg_src_in (reg_src_in),
16
17     .reg_req_out (reg_req_out),
18     .reg_ack_out (reg_ack_out),
19     .reg_rd_wr_L_out (reg_rd_wr_L_out),
20     .reg_addr_out (reg_addr_out),
21     .reg_data_out (reg_data_out),
22     .reg_src_out (reg_src_out),
23
24     // --- counters interface
25     .counter_updates (),
26     .counter_decrement(),
27
28     // --- SW regs interface
29     .software_regs ({reg_sample_ini, reg_sample_rate, reg_sw}),
30
31     // --- HW regs interface
32     .hardware_regs ({reg_sample_ctrl1, reg_sample_ctrl, cnt_discarded, cnt_accepted,
           cnt_total}),
```

Código B.1: Entradas y salidas del Generic Regs (I)

```
33          .clk (clk),
34          .reset (reset)
35      );
```

Código B.2: Entradas y salidas del Generic Regs (II)

En este código se distinguen tres bloques: hasta la línea 9 son señales de configuración, de la 10 a la 22 son las señales de control y de la 23 en adelante las señales de los registros.

Para añadir registros los pasos a seguir son los siguientes:

1.– Guardar el nombre de la etiqueta que hace referencia al módulo
.TAG (‘NETFLOW_L3L4EXTRACT_BLOCK_TAG),
Esta etiqueta se corresponde con la primera posición de memoria donde se encuentran los registros del módulo (en este caso se está usando como ejemplo el L3L4Extract). Se define en el fichero *netflow defines*, el cual tiene todas las direcciones de memoria usadas por el módulo.

2.– Indicar el número de registros del módulo, teniendo en cuenta los ya existentes y los nuevos.
.NUM_SOFTWARE_REGS (3), *//Number of sw regs*
.NUM_HARDWARE_REGS (5) *//Number of hw regs*
Debe recordarse que en un registro software se escribe desde el PC y se lee desde la FPGA y viceversa para uno hardware.

3.– Darle un nombre a los registros nuevos
.software_regs ({reg_sample_ini, reg_sample_rate, reg_sw}),
.hardware_regs ({reg_sample_ctrl1, reg_sample_ctrl, cnt_discarded,
cnt_accepted, cnt_total }),
El nombre de los registros puede ser cualquiera pero el orden en el que se definen los identifica. El orden es el inverso al que aparece en el fichero *netflow defines*.

4.– El fichero *netflow defines* se encuentra dentro del directorio include. En este fichero se definen cuatro elementos para cada bloque de memoria (un bloque de memoria contiene las posiciones de memoria a las que se accede desde un módulo).
Primero se define el bloque de memoria en sí, con dos constantes para indicar el tamaño, una para indicar el número de registros y la última para indicar la posición de memoria donde se encuentra:

```
1   // NFlow module block address
2   `define NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH
        `UDP_BLOCK_SIZE_64_REG_ADDR_WIDTH
3   `define NETFLOW_L3L4EXTRACT_BLOCK_ADDR_WIDTH
        `UDP_BLOCK_SIZE_64_BLOCK_ADDR_WIDTH
4   `define NETFLOW_L3L4EXTRACT_BLOCK_ADDR
        `NETFLOW_L3L4EXTRACT_BLOCK_ADDR_WIDTH'h4
5   `define NETFLOW_L3L4EXTRACT_BLOCK_TAG
        ({`UDP_BLOCK_SIZE_64_TAG,
        `NETFLOW_L3L4EXTRACT_BLOCK_ADDR})
```

Código B.3: Bloque de memoria definido en el fichero *netflow defines*.

Después se define el offset de cada registro en el bloque de memoria teniendo en cuenta que el orden en que aparezca lo identificará:

```
1   // L3L4EXTRACT registers
2   `define L3L4EXTRACT_ENABLE
        `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH'h0
3   `define L3L4EXTRACT_SWAP
        `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH'h1
4   `define L3L4EXTRACT_TOTAL
        `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH'h2
5   `define L3L4EXTRACT_ACCEPTED
        `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH'h3
6   `define L3L4EXTRACT_DISCARDED
        `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH'h4
```

Código B.4: Etiquetas para la posición de memoria.

A continuación se mapean los registros usando la primera posición del bloque de memoria y el offset dentro del bloque:

```
1   // modification for all registers: mapped with internal addr name
2   `define NFLOW_L3L4EXTRACT_ENABLE_REG
        (`UDP_BASE_ADDRESS |
        {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
        `L3L4EXTRACT_ENABLE})
3   `define NFLOW_L3L4EXTRACT_SWAP_REG
        (`UDP_BASE_ADDRESS |
        {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
        `L3L4EXTRACT_SWAP})
4   `define NFLOW_L3L4EXTRACT_TOTAL_REG
        (`UDP_BASE_ADDRESS |
        {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
        `L3L4EXTRACT_TOTAL})
5   `define NFLOW_L3L4EXTRACT_ACCEPTED_REG
        (`UDP_BASE_ADDRESS |
        {`NETFLOW_L3L4EXTRACT_BLOCdK_TAG,
        `L3L4EXTRACT_ACCEPTED})
6   `define NFLOW_L3L4EXTRACT_DISCARDED_REG
        (`UDP_BASE_ADDRESS |
        {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
        `L3L4EXTRACT_DISCARDED})
```

Código B.5: Mapeo de los registros del fichero *netflow defines* en memoria.

Y por último se escribe la función que llama a las librerías en C que leen y escriben en cada uno de los registros:

```
1   fwrite(c_reg_defines_fd, "#define␣
        NFLOW_L3L4EXTRACT_ENABLE_REG␣0x%07x\n\n",
        `NFLOW_L3L4EXTRACT_ENABLE_REG<<2); \
2   fwrite(c_reg_defines_fd, "#define␣
        NFLOW_L3L4EXTRACT_SWAP_REG␣0x%07x\n\n",
        `NFLOW_L3L4EXTRACT_SWAP_REG<<2); \
3   fwrite(c_reg_defines_fd, "#define␣
        NFLOW_L3L4EXTRACT_TOTAL_REG␣0x%07x\n\n",
        `NFLOW_L3L4EXTRACT_TOTAL_REG<<2); \
4   fwrite(c_reg_defines_fd, "#define␣
        NFLOW_L3L4EXTRACT_ACCEPTED_REG␣0x%07x\n\n",
        `NFLOW_L3L4EXTRACT_ACCEPTED_REG<<2); \
5   fwrite(c_reg_defines_fd, "#define␣
        NFLOW_L3L4EXTRACT_DISCARDED_REG␣0x%07x\n\n",
        `NFLOW_L3L4EXTRACT_DISCARDED_REG<<2); \
```

Código B.6: Funciones de acceso a memoria.

Es necesario modificar el fichero *netflow defines* de forma que se incluyan los registros nuevos. El fichero, después de añadir cuatro registros, queda de la siguiente manera (sólo se muestran las líneas de código del módulo *L3L4Extract*:

```
1        // NFlow module block address
2      `define NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH
           `UDP_BLOCK_SIZE_64_REG_ADDR_WIDTH
3      `define NETFLOW_L3L4EXTRACT_BLOCK_ADDR_WIDTH
           `UDP_BLOCK_SIZE_64_BLOCK_ADDR_WIDTH
4      `define NETFLOW_L3L4EXTRACT_BLOCK_ADDR
           `NETFLOW_L3L4EXTRACT_BLOCK_ADDR_WIDTH`h8
5      `define NETFLOW_L3L4EXTRACT_BLOCK_TAG
           ({`UDP_BLOCK_SIZE_64_TAG,
           `NETFLOW_L3L4EXTRACT_BLOCK_ADDR})
6
7      // L3L4EXTRACT registers
8      `define L3L4EXTRACT_ENABLE
           `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH`h0
9      `define L3L4EXTRACT_SWAP
           `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH`h1
10     `define L3L4EXTRACT_SAMPLE_RATE
           `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH`h2 //Nuevo
11     `define L3L4EXTRACT_SAMPLE_INI_VAL
           `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH`h3 //Nuevo
12     `define L3L4EXTRACT_TOTAL
           `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH`h4
13     `define L3L4EXTRACT_ACCEPTED
           `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH`h5
14     `define L3L4EXTRACT_DISCARDED
           `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH`h6
15     `define L3L4EXTRACT_SAMPLE_CTRL
           `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH`h7 //Nuevo
16     `define L3L4EXTRACT_SAMPLE_CTRL1
           `NETFLOW_L3L4EXTRACT_REG_ADDR_WIDTH`h8 //Nuevo
17
18     // modification for all registers: mapped with internal addr name
19     `define NFLOW_L3L4EXTRACT_ENABLE_REG
           (`UDP_BASE_ADDRESS |
           {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
           `L3L4EXTRACT_ENABLE})
20     `define NFLOW_L3L4EXTRACT_SWAP_REG
           (`UDP_BASE_ADDRESS |
           {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
           `L3L4EXTRACT_SWAP})
21     `define NFLOW_L3L4EXTRACT_SAMPLE_RATE_REG
           (`UDP_BASE_ADDRESS |
           {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
           `L3L4EXTRACT_SAMPLE_RATE}) //Nuevo
22     `define NFLOW_L3L4EXTRACT_SAMPLE_INI_VAL_REG
           (`UDP_BASE_ADDRESS |
           {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
           `L3L4EXTRACT_SAMPLE_INI_VAL}) //Nuevo
23     `define NFLOW_L3L4EXTRACT_TOTAL_REG
           (`UDP_BASE_ADDRESS |
           {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
           `L3L4EXTRACT_TOTAL})
24     `define NFLOW_L3L4EXTRACT_ACCEPTED_REG
           (`UDP_BASE_ADDRESS |
           {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
           `L3L4EXTRACT_ACCEPTED})
25     `define NFLOW_L3L4EXTRACT_DISCARDED_REG
           (`UDP_BASE_ADDRESS |
           {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
           `L3L4EXTRACT_DISCARDED})
```

Código B.7: Fichero *NetFlow defines* después de añadir nuevos registros (I).

```
26      `define NFLOW_L3L4EXTRACT_SAMPLE_CTRL_REG
            (`UDP_BASE_ADDRESS |
            {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
            `L3L4EXTRACT_SAMPLE_CTRL}) //Nuevo
27      `define NFLOW_L3L4EXTRACT_SAMPLE_CTRL1_REG
            (`UDP_BASE_ADDRESS |
            {`NETFLOW_L3L4EXTRACT_BLOCK_TAG,
            `L3L4EXTRACT_SAMPLE_CTRL1}) //Nuevo
28
29      `define PRINT_USER_REG_ADDRESSES \
30      fwrite(c_reg_defines_fd, "#define␣
            NFLOW_L3L4EXTRACT_ENABLE_REG␣0x%07x\n\n",
            `NFLOW_L3L4EXTRACT_ENABLE_REG<<2); \
31      fwrite(c_reg_defines_fd, "#define␣
            NFLOW_L3L4EXTRACT_SWAP_REG␣0x%07x\n\n",
            `NFLOW_L3L4EXTRACT_SWAP_REG<<2); \
32      fwrite(c_reg_defines_fd, "#define␣
            NFLOW_L3L4EXTRACT_SAMPLE_RATE_REG␣0x%07x\n\n",
            `NFLOW_L3L4EXTRACT_SAMPLE_RATE_REG<<2);
            /*Nuevo*/\
33      fwrite(c_reg_defines_fd, "#define␣
            NFLOW_L3L4EXTRACT_SAMPLE_INI_VAL_REG␣
            0x%07x\n\n",
            `NFLOW_L3L4EXTRACT_SAMPLE_INI_VAL_REG<<2);
            /*Nuevo*/ \
34      fwrite(c_reg_defines_fd, "#define␣
            NFLOW_L3L4EXTRACT_TOTAL_REG␣0x%07x\n\n",
            `NFLOW_L3L4EXTRACT_TOTAL_REG<<2); \
35      fwrite(c_reg_defines_fd, "#define␣
            NFLOW_L3L4EXTRACT_ACCEPTED_REG␣0x%07x\n\n",
            `NFLOW_L3L4EXTRACT_ACCEPTED_REG<<2); \
36      fwrite(c_reg_defines_fd, "#define␣
            NFLOW_L3L4EXTRACT_DISCARDED_REG␣0x%07x\n\n",
            `NFLOW_L3L4EXTRACT_DISCARDED_REG<<2); \
37      fwrite(c_reg_defines_fd, "#define␣
            NFLOW_L3L4EXTRACT_SAMPLE_CTRL_REG␣0x%07x\n\n",
            `NFLOW_L3L4EXTRACT_SAMPLE_CTRL_REG<<2);
            /*Nuevo*/ \
38      fwrite(c_reg_defines_fd, "#define␣
            NFLOW_L3L4EXTRACT_SAMPLE_CTRL1_REG␣0x%07x\n\n",
            `NFLOW_L3L4EXTRACT_SAMPLE_CTRL1_REG<<2);
            /*Nuevo*/ \
```

Código B.8: Fichero *NetFlow defines* después de añadir nuevos registros (II).

C

Configuración de la NetFPGA

El objetivo de este anexo es mostrar al lector la facilidad de uso del módulo NetFlow una vez se encuantra instalado y en correcto funcionamiento.

El módulo proporciona un script, escrito en perl, que permite configurar la NetFPGA. Usando este script es posible realizar tareas como configurar parámetros (IP y puerto destino del equipo colector), lanzar y parar el proceso de captura u obtener estadísticas de dicho proceso.

En concreto, los parámetros que se pueden configurar con el script son:

- Inactive timeout (-i): valor del inactive timeout en milisegundos.
- Active timeout (-a): valor del active timeout en milisegundos.
- IP y puerto del equipo colector (-c): IP y puerto del equipo al que se exportarán los paquetes CFLOW (paquetes UDP).
- IP y puerto del equipo anfitrión (-e): IP y puerto origen que se escribirá en los paquetes UDP.
- Interfaz de salida (-p): permite seleccionar una de las interfacez Ethernet de la NetFPGA por la que se exportarán los paquetes UDP.
- Usar la configuración por defecto (-d): configura los anteriores parámetros con su valor por defecto.
- Comenzar (-s): lanza el proceso de captura.
- Parar (-f): para el proceso de captura.
- Configuración (-conf): muestra la configuración actual de la NetFPGA.
- Estadísticas (-stat): muestra las estadísticas del proceso de captura (número de flujos capturados, número de paquetes descartados, etc).

A continuación se muestra un ejemplo de cómo se realiza el proceso de captura (el script se ejecuta desde consola):

1.– Configurar la NetFPGA y lanzar el proceso de captura:

```
./netflow.pl -i 10000 -a 30000 -c 192.168.1.2:9996
-e 192.168.0.1:1234 -p 0x10 -s
```

2.– Ver la configuración actual:

```
./netflow.pl -conf
```

Resultado:

```
NetFlow configuration:
====================
The NetFlow metering is:  Enabled
Bidirectional metering is:  Enabled
Active timeout is:  10 ms
Inactive timeout is:  30 ms
Source IP of NetFlow packet is:  192.168.0.1
Source port of NetFlow packet is:  1234
Destination IP of NetFlow packet is:  192.168.1.2
Destination port of NetFlow packet is:  9996
```

3.– Ver las estadísticas:

```
./netflow.pl -stat
```

Resultado:

```
 NetFlow statistics:
====================
Packets total:
Packets accepted:  14960
Packets discarded:  5
Packets rejected because no space available:  228
Number of active flows:  23
Total number of flows seen:  413
Total number of updates:  329
Total number of released flows:  283
```

4.– Parar el proceso de captura:

```
./netflow.pl -f
```

D

Presupuesto

1.– **Ejecución Material:**

- Compra de ordenador personal (software incluido) ..... 1000 €
- Compra de la tarjeta NetFPGA 1G ..... 478 €
- Disco duro portatil, capacidad 1 TB ..... 120 €
- Material de oficina ..... 100 €
- Total de ejecución material ..... 1698 €

2.– **Honorarios del proyecto**

- 1400 horas a 18 €/hora ..... 25200 €

3.– **Material fungible**

- Gastos de impresión ..... 200 €
- Encuadernación ..... 50 €

4.– **Subtotal del presupuesto**

- Subtotal Presupuesto ..... 27148 €

5.– **I.V.A. aplicable**

- 21 % Subtotal Presupuesto ..... 5701 €

6.– **Total presupuesto**

- Total presupuesto ..... 32849 €

E

Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un sistema de monitorización *netflow* usando la plataforma NetFPGA. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1.– La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2.– El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3.– En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4.– La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5.– Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6.– El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7.– Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que

sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8.– Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9.– Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10.– Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11.– Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondiera si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12.– Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13.– El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14.– Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15.– La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16.– La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17.– La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18.– Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19.– El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20.– Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21.– El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22.– Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23.– Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Pre-

supuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1.– La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2.– La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3.– Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4.– En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5.– En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6.– Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7.– Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8.– Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9.– Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10.– La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11.– La empresa consultora tendrá prioridad respecto a otras en la elaboración de los

proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12.– El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.