Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE CARRERA

TÍTULO DEL TFC: Experimentación con relojes de alta precisión para la monitorización de tráfico en redes de ordenadores

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Jordi Soler Camí

DIRECTORES: David Rincón Rivera

FECHA: 7 de mayo de 2012

Título: Experimentación con relojes de alta precisión para la monitorización de tráfico en redes de ordenadores

Autor: Jordi Soler Camí

Directores: David Rincón Rivera

Fecha: 7 de mayo de 2012

Resumen

Los operadores de redes de comunicaciones necesitan realizar una monitorización muy precisa del tráfico transportado. Ya sea para mantener la calidad necesaria en servicios en tiempo real (como por ejemplo una videoconferencia), o para poder monitorizar procesos y operaciones con alta precisión (como la necesaria en el caso del control de tráfico aéreo) necesitamos utilizar relojes de altas prestaciones que nos permitan realizar esta monitorización de la forma más precisa posible.

En la primera parte de este trabajo se exponen esas necesidades, los problemas que representan y se presentan dos posibles soluciones (una basada en hardware y otra en software) para conseguir una monitorización de precisión. A continuación se analizan con detalle las soluciones propuestas y se explica cómo ponerlas en funcionamiento.

Las soluciones propuestas se evalúan mediante unos escenarios de prueba con los que se pretende analizar cuál de las dos soluciones nos ofrece una mejor precisión en la monitorización del tráfico de red, así como determinar otras ventajas que nos ofrecen estas soluciones y que pueden servir para otros posibles estudios relacionados con el tema tratado en este documento.

Finalmente, se presentan las conclusiones a las que hemos llegado. Por lo que hemos podido ver a raíz de las pruebas realizadas las tarjetas hardware ofrecen mejores resultados que la solución software, pese a ello, la solución software supone igualmente una mejora en la monitorización de tráfico y además es una solución mucho más barata que la solución basada en hardware. Además esta solución software ha demostrado tener unas cualidades que pueden explotarse en otros ámbitos.

Title: Experiments with high accuracy clocks applied to traffic monitoring in computer networks

Author: Jordi Soler Camí

Director: David Rincón Rivera

Date: May, 7th 2012

Overview

The operators of communcations networks need to accurately monitoring their traffic, due to several reasons. For example, it is important to maintain the necessary quality in real-time services (for example a videoconference) or to monitor industrial processes and operations with high precision (air traffic control, for example). In any of these cases, we need high performance clocks to perform this monitoring as accurately as possible.

In the first part of this document we describe these needs, the problems they represent and two possible solutions (one based on hardware and other based on software) to obtain an accurate monitoring clock. Then we analyze in detail the proposed solutions and describe how to put them into operation.

The proposed solutions are evaluated by means of testbeds, in order to analyze which of the two solutions gives us a better accuracy in monitoring the network traffic, and determine other advantages offered by these solutions that can serve to other possible studies related to the subject matter in this document.

Finally, we present the conclusions we have reached. The main conclusion is that the hardware solution give better results than the software solution. Despite this, and compare to the usual scenario (no high-precision clocks), the software solution offers an important improvement in the monitoring of traffic and it's a much cheaper solution than the hardware-based solution. Furthermore, this software solution has demonstrated qualities that can be exploited in other applications.

Gracias David por tu ayuda y tu paciencia.

También he de agradecer a mi familia,
amigos y especialmente a mi pareja
Ester, por todo el tiempo que no
les he podido dedicar a ellos.

ÍNDICE

INTRODUCCIÓN

Gracias los avances tecnológicos y al aumento de la demanda de servicios el uso de las comunicaciones a través de internet ha crecido de una forma exponencial en los últimos años. Cada vez son más las empresas que comunican sus distintas sedes y oficinas a través de la red. Ya es prácticamente imposible encontrar un hogar en nuestro país en el que no se disponga de acceso a internet. Además los proveedores de internet (ISP) cada vez ofrecen conexiones de más velocidad para los usuarios, que a su vez siguen demandando mayores velocidades.

Por estos motivos, la cantidad de datos que ha de transmitirse por las redes troncales es cada vez mayor. Además muchos de estos datos corresponden cada vez más a servicios en tiempo real (videoconferencias, juegos en red, comunicaciones industriales, monitorización[1], seguridad…). Para poder ofrecer todos estos servicios con la calidad necesaria es muy importante que la sincronización entre los equipos sea lo más precisa posible, necesitando para ello relojes de altas prestaciones.

Mediante una buena sincronización podemos mejorar los parámetros de QoS (*Quality of service*), garantizando así la transmisión de cierta cantidad de información en un tiempo dado. También podemos obtener un buen control de uso de parámetros para así poder detectar y solucionar los problemas que se pueden presentar por falta de recursos. Por otro lado las operadoras pueden realizar un buen *accounting* de los recursos empleados por sus clientes.

Para poder llevar esto a cabo es necesario, en primer lugar, realizar una monitorización muy precisa del tráfico que se pretende controlar. Para poder llevar a cabo esta monitorización necesitaremos equipos capaces de realizar capturas de tráfico de alta velocidad lo más precisas posibles, capturando el tiempo de llegada o emisión de los paquetes IP o tramas de nivel de enlace.

Dadas las velocidades de las redes troncales y la cantidad de datos que se envían por ellas, es necesario que las máquinas que se encargan de realizar la monitorización del tráfico dispongan también de relojes de altas prestaciones, capaces de detectar y capturar el mayor numero de paquetes que pasa por ellas.

Pese a que existen arquitecturas hardware especializadas en solucionar estos problemas (por ejemplo, las tarjetas **DAG** (*Data Acquisition and Generation*) de Endace), son muy caras y no se pueden desplegar de manera generalizada. Si se quiere hacer con equipos de bajo coste (PCs dedicados) surge un reto técnico, ya que el reloj físico del que disponen los equipos de hoy en día (TSC o Time System Clock) dista mucho de ser un reloj de precisión. Internamente

[1] Por ejemplo, los datos de los radares aeronáuticos se están transmitiendo sobre una red IP privada con el protocolo RTP, y hay restricciones importantes en el tiempo máximo en que estos datos tienen que llegar desde el radar a las consolas de los controladores aéreos (del orden de unos 100 milisegundos, que a una velocidad de 800 Km/h – 222 m/s- supone que el avión recorre algunas decenas de metros).

estos relojes presentan problemas de precisión (frecuencia) y de estabilidad en la deriva del reloj, dando lugar a una precisión del orden de ±10 ppm[2].

Por otro lado no hemos de olvidar que la mejora de la precisión del reloj de una máquina no solo mejora las capacidades de esta para realizar una buena monitorización del tráfico, sino que, además, mejorará también las capacidades del equipo en cuanto a emisión de datos se refiere. Al igual que un buen reloj puede realizar una buena marcación del tráfico entrante, también puede mejorar la estabilidad de los datos enviados. Y dado que en ciertos estudios de rendimiento de equipos y protocolos de red se necesita generar tráfico para estresar los equipos o valorar su respuesta, la disponibilidad de un buen reloj que gobierne la generación de tráfico se convierte en una necesidad.

Motivaciones

Originalmente el proyecto seguía la estela de un PFC realizado en la EETAC [1] donde se estudiaron algunas de las capacidades de las tarjetas capturadoras DAG. El objetivo del PFC fue el desarrollo de software de captura y análisis de tráfico para estas tarjetas, y el presente proyecto debía volver a poner en marcha las tarjetas y avanzar más en el estudio

Las DAG 4.3 GE de Endace son unas tarjetas capturadoras que disponen de dos puertos Gigabit Ethernet cada una, con altas prestaciones en cuanto a precisión del reloj y captura a *wire speed*[3]. Estos puertos son accesibles mediante dos módulos SFP (Mini-GBICS) donde se pueden conectar *transceivers* ópticos o de cobre. Los *transceivers* ópticos pueden ser 1000BaseSX (longitud de onda 850 nm) o 1000BASELX (longitud de onda 1310 nm), estos suelen tener conectores ópticos tipo LC. Los *transceivers* de cable, par trenzado o cobre son los 1000Base-T cuyos conectores son RJ-45.

Desafortunadamente, los *transceivers* utilizados en el proyecto original no estaban disponibles, por lo que se tuvo que realizar una búsqueda de nuevos *transceivers* así como realizar su compra. Debido al retraso temporal que eso conllevaba decidimos dar un nuevo enfoque al proyecto, buscando otras posibles soluciones para la obtención de una monitorización de tráfico precisa. Por ello estudiamos **Radclock** [2], una nueva arquitectura de sincronización que combina el reloj software (poco preciso) de los PCs con los ajustes que se reciben desde relojes de alta precisión (atómicos, GPS) recibidos desde Internet.

[2] Partes por millón. Si un reloj de 10 MHz tiene una tolerancia en frecuencia de +/-10 ppm, significa que su frecuencia real se mueve en un margen de +/- 100 Hz respecto a la frecuencia central de 10 MHz

[3] Se denomina así a la capacidad de capturar todas las tramas Ethernet incluso al 100% de la capacidad del enlace. Una tarjeta de red “normal” para PC suele tener un límite del 70-80% de la capacidad del enlace (700-800 Mbit/s para el caso de Gigabit Ethernet).

Objetivos

El objetivo principal de este proyecto es el estudio de diferentes posibles soluciones que ofrezcan relojes de altas prestaciones para poder realizar capturas de tráfico de alta velocidad con la máxima precisión posible.

En este caso, hemos decido estudiar dos posibles soluciones de diferente tipo, una basada en hardware y otra de tipo software. Por un lado estudiaremos las tarjetas capturadoras de tráfico *DAG card 4.3 GE* de Endace. Estas tarjetas nos ofrecen medio hardware capaz de realizar capturas de tráfico de altas prestaciones. Por otro lado, estudiaremos una posible solución software, el reloj RADclock. Este software nos ofrece la posibilidad de mejor la precisión y la estabilidad del reloj físico de la máquina donde lo ejecutemos.

El resto de este documento está organizado como se describe a continuación:

En el **capítulo 1**, ampliaremos la información sobre la monitorización de tráfico, analizaremos los problemas que se presentan y sus posibles soluciones. En el **capítulo 2** realizaremos un pequeño resumen de las tarjetas DAG basado en los informes de PFCs anteriores. Además explicaremos como poder utilizar estas tarjetas. A continuación, en el **capítulo 3**, pasaremos a analizar el software RADclock, su instalación y su puesta en marcha. En el **capítulo 4** analizaremos las pruebas realizadas con las dos soluciones y compararemos los resultados. El documento finaliza presentando las conclusiones a las que hemos llegado, así como los posibles estudios a realizar en futuros proyectos.

CAPÍTULO 1. MONITORIZACIÓN DE TRÁFICO

1.1. Importancia del reloj en la monitorización de tráfico

Para la correcta realización de una monitorización de tráfico hay una serie de cuestiones que tenemos que tener en cuenta. Por un lado, necesitamos un marcado temporal preciso, así como minimizar el jitter[4] para obtener una buena calidad en las transmisiones de tiempo real. Por otro lado, los proveedores necesitan realizar un control preciso de sus servicios para poder cumplir la QoS acordada en los SLA (*Service Level Agreement*)[5] con sus clientes.

A la hora de realizar capturas de tráfico para su análisis se nos presentan diferentes opciones. Una de ellas es la captura de los paquetes completos, pudiendo así realizar chequeos de la integridad de los datos si es necesario. Otra posible opción es capturar solo las cabeceras de los paquetes (o las cabeceras más una pequeña parte de los datos) puesto que con los datos incluidos en ellas ya tenemos lo suficiente como para analizar el paquete (*timestamps*, direcciones origen y destino, tipo de paquete…).

Independientemente de la opción que utilicemos es de vital importancia para el correcto análisis de los paquetes que el marcado temporal de los mismos sea lo más precisa posible. Cuando se captura un paquete, lo primero que se hace es asociarle una marca temporal, indicando en qué momento exacto ha llegado (*timestamp*). Cuanto más precisa sea esta medida podremos calcular con más precisión el tiempo de llegada entre paquetes (*delta time*). Debido a las variaciones que puede sufrir el *round-trip time* (RTT)[6] de distintos paquetes este tiempo entre paquetes puede variar.

El retardo y su estabilidad son especialmente importantes en transmisiones en tiempo real de flujos multimedia, donde un gran retardo o su variación (jitter) pueden dar lugar a interrupciones de la comunicación. Por ejemplo, la variación del retardo puede producir interrupciones en la transmisión de voz por Telefonía IP. Otro ejemplo: en el caso de la transmisión por IP de datos de control de tráfico aéreo es muy importante que el radar de control reciba la posición de los distintos aviones de una forma extremadamente rápida, periódica y precisa para monitorizar donde esta cada avión en todo momento.

[4] El *jitter* se define como la variación del retardo extremo a extremo respecto a su media. Es decir, si los cuatro primeros paquetes de una transmisión tardan 15, 18, 15 y 16 ms en ir desde el origen a su destino, y siendo el retardo medio 16 ms, el jitter de cada paquete sería -1 ms, + 2 ms, -1 ms, y 0 ms, respectivamente.

[5] El SLA es un contrato escrito entre un proveedor de servicio y su cliente cuyo objetivo es llegar a un acuerdo para fijar la calidad de dicho servicio.

[6] El RTT es el tiempo transcurrido desde que se envía un paquete hasta que llega la respuesta a dicho paquete.

1.2. Problemas

A la hora de conseguir esta precisión temporal surgen diversos problemas. Por un lado tenemos que mejorar de alguna manera la calidad del reloj local de la máquina. La falta de precisión y una deriva del reloj muy grande pueden acarrear que la marcación del paquete sea incorrecta.

Además cuando realizamos capturas de tráfico no solo interviene la tarjeta NIC[7] a nivel físico, también intervienen el sistema operativo (SO) y la aplicación con la que realicemos la captura.

Figura 1.1. Capas que intervienen en la captura de tráfico.

Cuando un paquete llega a la tarjeta NIC esta debe leer el paquete entero, comprobarlo y copiarlo al PC. Esto produce una interrupción en el sistema operativo. Cuando la CPU esté libre, atenderá la interrupción y marcará el paquete con el *clock* del sistema. Esta latencia adicional puede variar decenas de milisegundos dependiendo del equipo utilizado, falseando así el tiempo real de recepción del paquete. Cuanta más alta sea la velocidad de la transmisión más nos afectara este retardo adicional.

1.3. Posibles soluciones

Para solucionar estos problemas nos encontramos con dos posibles frentes de actuación. Por un lado podemos intentar mejorar la calidad del reloj que utilizamos, o bien modificar el software utilizado para conseguir un mejor retardo en el procesado del paquete.

Para conseguir esto hemos decidido investigar diferentes soluciones. Por un lado realizaremos pruebas de captura con las tarjetas DAG de Endace, que nos ofrecen una solución hardware para realizar capturas de tráfico con mucha precisión. Otra posible solución que estudiaremos es utilizar un software con el que implementar un reloj de altas prestaciones que nos permita realizar capturas con precisión a través de tarjetas NIC normales.

[7] NIC (*network interface card*), periférico que permite la comunicación entre equipos conectados entre sí o a una misma red.

1.3.1. DAG card 4.3 GE

La principal característica que nos hizo estudiar esta solución es su capacidad de marcado temporal. En la **figura 1.2** podemos observar la arquitectura por capas de la tarjeta. En ella vemos que el *DAG universal Clock Kit* (DUCK) se encuentra en las primeras capas realizando así un marcado hardware. Este reloj es el que utiliza la DAG para realizar el marcado temporal, y lo hace tal y como comienza a recibir el paquete, antes de comenzar su procesado. Como ya hemos comentado antes, esto supone una mejora ya que al capturar con una tarjeta NIC normal el marcado del paquete se realiza después de recibir y procesar el paquete entero, añadiendo así un retardo de procesado.

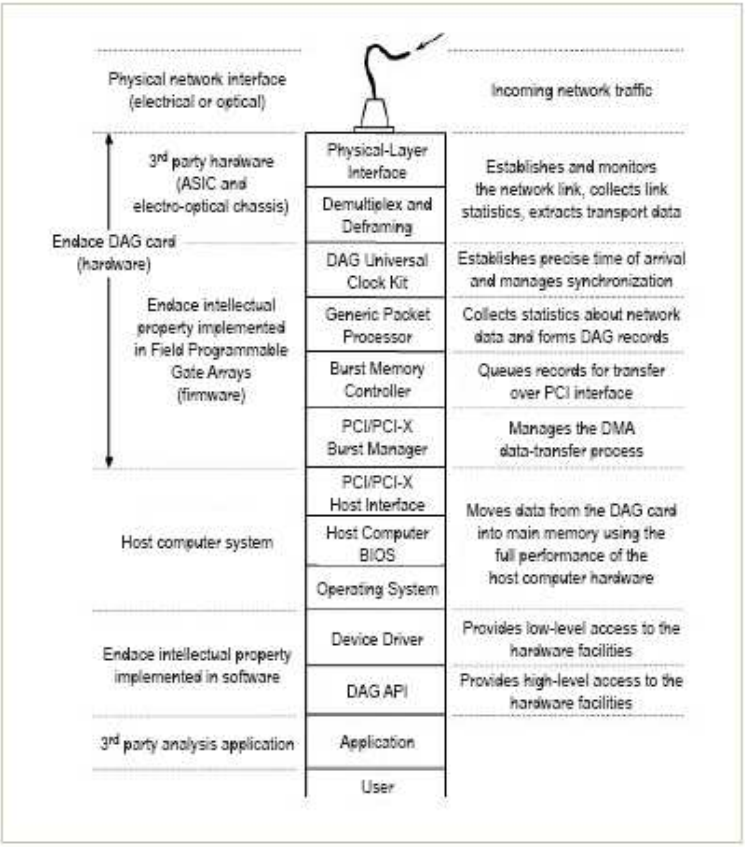

Figura 1.2. Arquitectura de las capas de la tarjeta DAG [Imagen extraída de *Endace Dag Technology Brief*, pág. 3].

En la **figura 1.3** podemos observar que a diferencia de las tarjetas NIC normales, la tarjeta DAG realiza el marcado durante el preámbulo de la trama sin producir ninguna interrupción. Además este marcado se realiza con el DUCK, de manera que en ningún momento dependemos del reloj de la máquina.

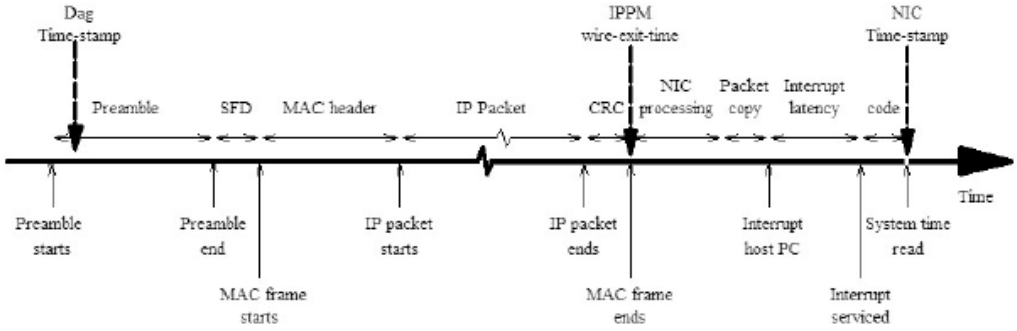

Figura 1.3. Momento de marcado en tarjetas DAG y tarjetas NIC [Imagen extraída de *Endace Dag Time-Stamping Whitepaper*, pág. 2].

Este reloj DUCK interno tiene una precisión de 100ns. Además este sistema puede mantener una sincronización del reloj de la tarjeta con otra referencia externa como, por ejemplo, un reloj GPS.

1.3.2. Radclock

Como ya hemos comentado, uno de los problemas de las máquinas de las que disponemos (PCs normalmente) es que disponen de un reloj local de muy baja precisión, con derivas de hasta 1 segundo por día (11,6 ppm). Estos relojes son los que marcan la base temporal a la hora de enviar o capturar paquetes. Y por ello una posible solución es mejorar sus prestaciones.

Existen diferentes protocolos relacionados con la sincronización, por ejemplo NTP (*Network Time Protocol*), que buscan mejorar indirectamente las prestaciones de estos relojes corrigiéndolos a partir de las señales de relojes de más alta precisión. NTP es un protocolo utilizado en Internet para sincronizar los relojes a través de redes IP con latencia variable. Es transportado sobre UDP utilizando el puerto 123.

NTP fue diseñado por *Dave Mills* de la universidad de *Delaware* en 1985. Utiliza el algoritmo de Marzullo con el estándar de tiempo UTC (*Universal Time Coordinated*). La versión que se utiliza actualmente (NTPv4) puede mantener dos relojes sincronizados con una diferencia máxima de 10ms a través de internet. Esta diferencia puede disminuir hasta los 200ns en redes locales con bajas latencias [3].

NTP define un sistema de jerarquías de relojes en estratos (o *stratum*):

- Estrato 0: GPS, relojes atómicos (cesio, rubidio). Normalmente estos relojes son externos a la red.
- Estrato 1: Máquinas cuyo reloj esta sincronizado a uno de estrato 0. Normalmente estas maquinas ejercen como servidores NTP.
- Estrato 2: Máquinas cuyo reloj esta sincronizado con uno de estrato 1.
- Y sucesivamente hasta llegar a los 256 estratos, pero normalmente no se utilizan relojes con un estrato mayor a 2 o 3.

Los *timestamps* utilizados por este protocolo consisten en un segundo de 32 bits mas una parte fraccional de 32 bits. Con esto se consigue una escala de 2^{32} segundos (136 años), con una resolución teórica de 2^{-32} segundos (0.233ns). Podemos encontrar más detalles de NTP en [4], [5] y [6]

Normalmente en los sistemas Unix este protocolo se implementa mediante el ***daemon* ntpd**. Este es un proceso que puede ejecutarse de forma continua a nivel de usuario. El reloj RADclock representa una alternativa para el *daemon* ntpd. Pese a trabajar de una manera muy similar, nos ofrece una serie de ventajas que explicaremos en el capítulo 3.

CAPÍTULO 2. DAG CARD 4.3 GE

2.1. Introducción

A raíz de los estudios realizados en TFCs anteriores [1], podemos asegurar que las tarjetas de captura pasiva DAG GE 4.3 nos ofrecen la posibilidad de realizar capturas de tráfico precisas con una mínima carga para el procesador.

En este proyecto las utilizaremos para realizar capturas de tráfico generado por nosotros. Analizaremos la precisión de los *timestamp* y el jitter de estas capturas para comparar los resultados obtenidos en las soluciones propuestas.

2.2. ¿Por qué utilizar las tarjetas DAG?

Además de lo expuesto en el capitulo anterior hay otras características de estas tarjetas que queremos destacar. A diferencia de las tarjetas NIC habituales que fácilmente que pueden verse sobrecargadas con el tráfico de una red (incluso si es pequeña), las tarjetas DAG permiten una monitorización eficiente del tráfico de red. Para ello, utilizan la memoria del PC evitando interrupciones y liberando así la carga a la CPU. Además debido a su diseño ofrecen una gran precisión en los *timestamps* y en su sincronización.

Por otro lado, la capacidad de la tarjeta para capturar a tasa máxima va ligada a las capacidades del PC donde esta está instalada. Es muy importante la velocidad del disco duro, así como tener en cuenta que al estar conectadas al bus PCI-X solo pueden trabajar hasta 4 Gbps. Además la tarjeta dispone de un buffer FIFO donde almacenar unos cuantos milisegundos de tráfico que le permiten soportar ráfagas de paquetes pequeños a tasa máxima.

El formato utilizado por las tarjetas para realizar las capturas ofrece la posibilidad de almacenar el *timestamp* con una precisión superior que las herramientas que utilicen la librería **libpcap**. Las DAG nos ofrecen una resolución de hasta 15 nanosegundos frente a los milisegundos de libpcap. A diferencia de libpcap estas tarjetas realizan el *timestamp* al principio del paquete y no al final de la recepción del mismo (eliminando así el tiempo de procesado del paquete).

Para poder trabajar a estas velocidades el driver de la tarjeta DAG reserva un espacio continuo de la memoria física del PC que se utiliza a modo de buffer circular:

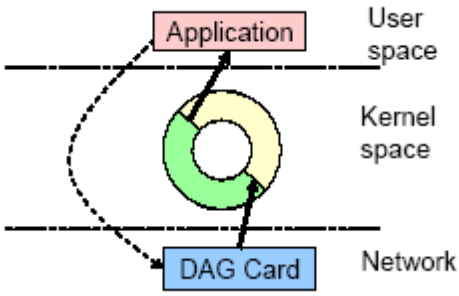

Figura 2.1. Arquitectura software DAG [Imagen extraída de [1]].

Cuando se inicia una captura el driver realiza dos mapeados de memoria, uno para crear un puntero al espacio de memoria en el espacio de direcciones de la aplicación, y otro para llevar el espacio de entrada/salida de la tarjeta a la aplicación. Así la aplicación que realiza la captura tiene un control completo de la tarjeta sin necesidad de pasar por el kernel del sistema operativo. Gracias a este modo de funcionamiento la tarjeta DAG supone una carga mínima para la CPU.

2.2.1. Captura de tráfico

Las tarjetas DAG nos permiten capturar todos los paquetes que pasan por sus puertos. Además, al no tratarse de tarjetas NIC convencionales, no responderán a mensajes de resolución de direcciones (por ejemplo ARP), ni a ningún protocolo que requiera de mensajes de respuesta (por ejemplo ICMP).

En nuestro caso tendremos que capturar un flujo de tráfico cuyo destinatario no será la DAG. Por ello necesitamos introducir en los escenarios **splitters** o bien **switches con funcionalidad de *mirror*** en medio del enlace para poder capturar una copia de los paquetes y sacarla por el puerto donde se conecta la DAG. Nosotros hemos optado por la segunda opción, utilizando para las pruebas realizadas con la DAG un switch SMC 8624T (cedido por la fundación i2Cat) por su capacidad para realizar *port mirroring*.

Figura 2.2. Switch SMC 8624T

2.3. Instalación

Ahora expondremos las necesidades de la tarjeta DAG para su instalación, así como los pasos a seguir para poner la tarjeta en marcha.

2.3.1. Preparación del software

El software y drivers para las tarjetas DAG de Endace puede funcionar en cualquier Linux con kernel 2.4.x o 2.6.x. No es necesaria ninguna modificación del kernel para su instalación.

La instalación requiere que el entorno de desarrollo en C esté disponible así como los paquetes tar, gcc, *make* y los *headers files* del kernel que estemos utilizando.

Además hemos de asegurarnos que los paquetes flex, byacc, bison, readline, readline-dev y libghc6-mmap-dev estén instalados.

En nuestro caso hemos instalado las tarjetas en Debian 3.1 con kernel 2.4.27. Hemos decidido utilizar esta versión de Linux porque hemos tenido problemas de compatibilidad al intentar instalar los drivers para la DAG de que disponíamos en SO más modernos. Todos los paquetes necesarios que no fueron instalados junto con el sistema operativo han sido instalados mediante el comando `aptitude install`

2.3.2. Instalación de la tarjeta

Primero debemos descomprimir el juego de drivers (en este caso el 2.5.5) en la carpeta `/usr/local`.
Esto nos crea una carpeta llamada `dag-2.5.5`.

Creamos un symlink a esta tarjeta con el comando:

```
ln –s dag-2.5.5 dag
```

Entramos en el directorio y ejecutamos los siguientes comandos:

```
./ config
make depend
make
make install
```

Con esto instalamos los drivers de la tarjeta así como la librería libdag. Esta se instala por defecto en `/usr/local/lib`. Para asegurarnos que el sistema coge esta librería la añadimos al fichero `/etc/ld.so.conf` y ejecutamos `ldconfig`.
En los anexos 2.1 a 2.4 podemos encontrar los logs obtenidos en los diferentes pasos de la instalación.

Por último tenemos que añadir el modulo dagmem para que el sistema reserve el espacio requerido en la memoria RAM al activar la tarjeta.
Para ellos hemos añadido lo siguiente al fichero `/etc/modules`:

```
dagmem dsize=512M
```

Reiniciamos la máquina y ya podemos empezar a utilizar la tarjeta DAG.

2.4. Puesta en marcha

Una vez esta la tarjeta DAG instalada en nuestra máquina podemos comenzar a realizar capturas con ella. Para ello aun debemos realizar unos pasos previos.

2.4.1. Preparación de los drivers

Para poder realizar capturas de tráfico con la tarjeta hemos de realizar primero unos pasos. Para ello hemos utilizado de guía los informes de [1].

La tarjeta DAG incluye un conjunto de utilidades para la realización de las operaciones básicas para su funcionamiento (inicialización de parámetros, transmisión, captura…). Estas se encuentran en el directorio `/usr/local/dag-2.5.5/tools`. Para poder utilizarlas hay que cargar el modulo dag mediante el comando `dagload`.

A continuación ejecutamos el comando `dagroom`. Este comando sirve para actualizar y cambiar el firmware de la tarjeta. Lo ejecutamos con la opción `–p` para habilitar la transmisión y recepción de tráfico.

Finalmente se ejecuta el comando `dagfour default` para utilizar la configuración por defecto de la tarjeta.

Ahora la tarjeta ya está lista para realizar capturas de tráfico.

2.4.2. Captura de tráfico

Para capturar el tráfico hemos utilizado una herramienta incluida en el software de la tarjeta. La función que hemos utilizado es: `dagsnap`. Con ella la tarjeta realiza una captura de tráfico ininterrumpida. Ejecutamos el siguiente comando:

```
dagsnap –d /dev/dag0 –o captura.erf
```

El resultado de la captura lo escribe en un fichero de salida que indicamos con la opción -o. Este fichero lo analizamos con el analizador de protocolos Wireshark [7] para comprobar los resultados.

2.4.3. Resultados de la captura

El fichero .erf obtenido contiene todos los paquetes recibidos por la tarjeta durante el tiempo que hemos tenido activa la captura. Este fichero está en formato binario y para poder analizarlo lo hemos de pasar a un formato legible. Esto lo realizamos mediante la función `dagbits`:

```
dagbits dge time jitter delta print -f captura.erf >
captura.asc
```

Esta función comprueba e imprime cada *timestamp*, CRC, cabeceras ATM, *payloads*, delta dime y el jitter del fichero .erf y lo pasa a un fichero ASCII. El fichero que obtenemos tiene el siguiente formato:

```
print 1: file offset 0x0
ts=0x4f60b52d39e0c2c0 2012-03-14 15:11:41.2260858 UTC
type: ERF Ethernet
dserror=0 rxerror=0 trunc=0 vlen=1 iface=0 rlen=64 lctr=0
wlen=573
etype=0x86dd
dst=33:33:00:00:00:0c src=00:1f:d0:25:6e:f4
60 00 00 00 02 03 11 01 fe 80 00 00 00 00 00 00
00 6c 8b 0f e7 f5 39 44 ff 02 00 00 00 00 00 00

dge 1: dge content test failed
print 1: file offset 0x0
ts=0x4f60b52d39e0c2c0 2012-03-14 15:11:41.2260858 UTC
type: ERF Ethernet
dserror=0 rxerror=0 trunc=0 vlen=1 iface=0 rlen=64 lctr=0
wlen=573
etype=0x86dd
dst=33:33:00:00:00:0c src=00:1f:d0:25:6e:f4
60 00 00 00 02 03 11 01 fe 80 00 00 00 00 00 00
00 6c 8b 0f e7 f5 39 44 ff 02 00 00 00 00 00 00
…
```

Podemos ver el *timestamp*, el tipo *(ERF Ethernet)* y los *flags (*ningún error interno, ningún error en el enlace, ningún truncamiento, paquetes de longitud variable y uso de la interface 0*)*. También vemos la longitud total de la información transferida al bus PCI *(rlen),* ninguna pérdida *(lctr)* y el tamaño del paquete junto con su cabecera *(wlen).* Por último se nos muestra los datos útiles del paquete.

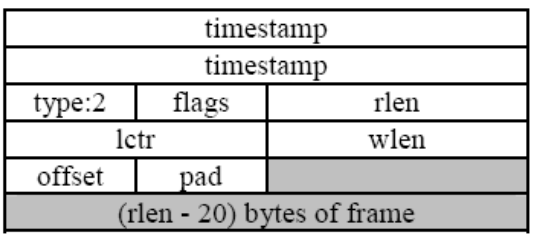

Fig. 2.3. Formato paquete ERF [Imagen extraída del Manual de Usuario de la tarjeta DAG Card GE 4.3 de Endace]

CAPÍTULO 3. RADCLOCK

3.1. Introducción

RADclock [2] (antiguamente conocido como “TSCclock”) es un proyecto iniciado por Darryl Veitch en la Universidad de Melbourne. Durante los últimos 10 años han estudiado diferentes formas de mejorar el reloj TSC de PCs mediante el estudio de algoritmos y su implementación vía software. Actualmente este proyecto sigue abierto y siguen desarrollando nuevas versiones del software (la última versión estable (0.3.2) es de diciembre del 2011 y hay una versión 0.4.0 en fase beta). Este sistema es compatible con servidores NTP y puede funcionar en paralelo al *daemon* ntpd.

Este proyecto busca reemplazar el protocolo de sincronización NTP por uno casi tan preciso como la sincronización por GPS (precisiones de 1 a 0,1 ppm) pero más barato y realizado mediante la combinación del reloj (impreciso, ±10 ppm) del PC y un software especifico. Para ello RADclock se basa en un nuevo principio que distingue los relojes diferenciales y los absolutos.

- *Difference clock* (reloj diferencial): mide con precisión el tiempo entre 2 sucesos.
- *Absolute clock* (reloj absoluto): mide la referencia temporal absoluta de un momento determinado (ahora son las XX:XX:XX:xxxx)

Ambos relojes son accesibles mediante una API (*Application Programming Interface*) especifica: *libradclock*.

3.2. ¿Por qué utilizar RADclock?

Como ya se ha comentado, a la hora de realizar capturas de tráfico nos interesa poder realizar-las de la manera más rápida posible. Para ello necesitamos un medio que nos permita capturar el máximo de paquetes con la mayor precisión posible. Como hemos comentado anteriormente una posible solución para esto es la utilización de un software capaz de implementar un reloj de altas prestaciones que nos permita mejorar la precisión de las capturas realizadas con tarjetas NIC normales y corrientes.

RADclock nos ofrece una solución de este tipo. Según su creador [2] este software emplea una serie de algoritmos que, basándose en los datos obtenidos del dialogo NTP con un servidor de *stratum 1*, realiza una serie de estimaciones. Para ello, RADclock utiliza los valores temporales de Ta, Tb, Te y Tf obtenidos en los diferentes paquetes NTP:

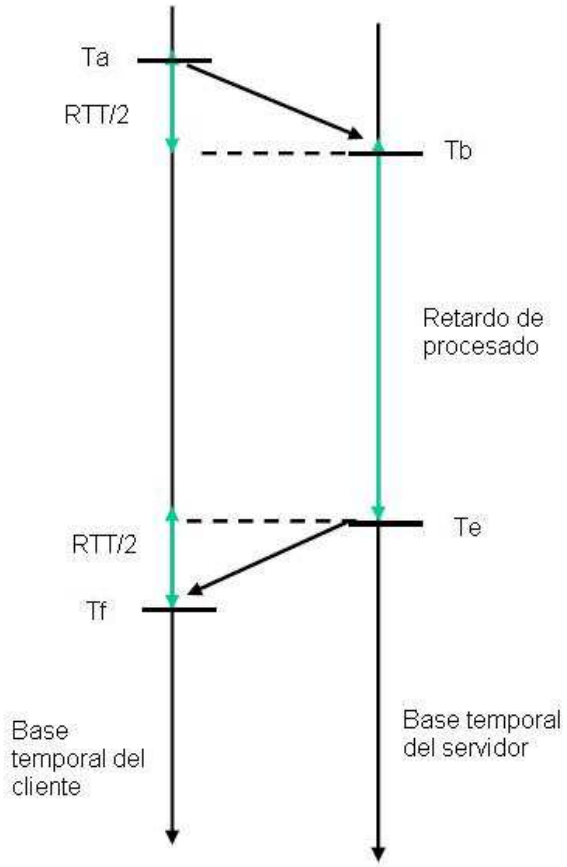

Figura 3.1. Ejemplo de dialogo NTP (figura extraída de [2])

$$Tf\_corr = Te + RTTest/2 \qquad \textbf{(2.1)}$$

donde la estimación del RTT se calcula mediante la expresión:

$$RTTest = (Tf - Ta) - (Te - Tb) \qquad \textbf{(2.2)}$$

A partir de estos valores temporales RADclock realiza una serie de cálculos para realizar predicciones de esos mismos valores para el siguiente par de paquetes petición NTP – respuesta NTP (a lo que a partir de ahora llamaremos *stamps*). Después, a raíz de los mismos valores temporales del próximo par de paquetes y de la predicción realizada, RADclock calcula el error que se ha producido en su predicción.

Poco a poco RADclock utiliza estos valores temporales (tanto los extraídos de los paquetes NTP, como las previsiones calculadas y su error) para estimar la deriva del reloj ($\pm\Delta t$) propio de nuestra máquina así como del reloj del servidor NTP con el que hemos establecido del diálogo. Además de este nuevo valor calculado también calcula el error producido entre predicción y predicción.

Mediante este método de predicciones y cálculo de errores RADclock genera un reloj de alta precisión, basado en el reloj del servidor NTP. Este reloj se utiliza para sincronizar el TSC de la máquina donde hemos ejecutado RADclock.

La precisión de estas predicciones depende del tiempo que el diálogo NTP se mantenga, de manera que cuanto más tiempo dure la conexión más fiables serán las previsiones, obteniendo así un TSC más estable y preciso, llegando a la máxima precisión tras aproximadamente dos meses de conexión. Por otro lado el autor [2] nos asegura que frente a una caída del dialogo NTP, RADclock, nos ofrece una deriva del TSC más estable que sin él, y además que recuperará la estabilidad obtenida en cuanto se recupere la conexión con el servidor NTP.

La **figura 3.2** nos muestra el comportamiento de RADclock y ntpd frente a una caída de la conexión con el servidor durante dos horas. Mientras que ntpd tarda casi otras dos horas en recuperar la estabilidad, RADclock la recupera en cuestión de segundos tras la reconexión con el servidor. Además durante ese periodo de dos horas el reloj gobernado por RADclock pierde su estabilidad poco a poco, mientras que el controlado por ntpd se vuelve muy inestable de manera inmediata.

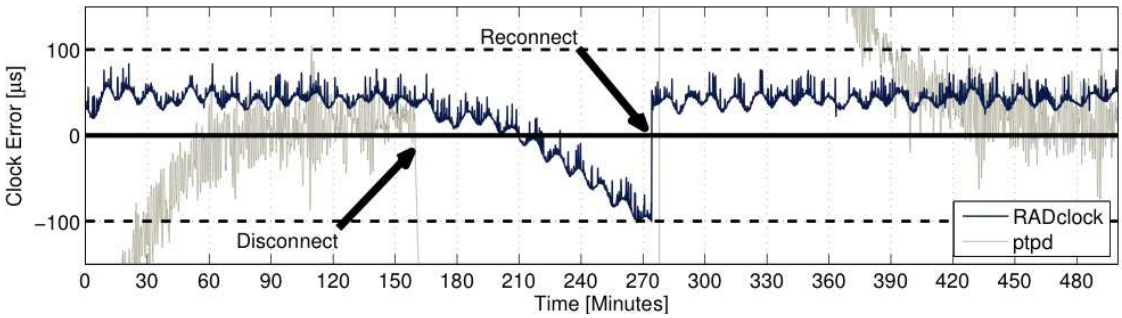

Figura 3.2. Ejemplo de caída de la conexión con el servidor NTP utilizando RADclock y ntpd (Imagen extraída de [2])

De esta manera el reloj software RADclock sincroniza el TSC propio de la maquina con el reloj del servidor NTP con el que se comunica para obtener una precisión del orden de 0,1 ppm [2].

En el momento en el que comenzamos a investigar sobre RADclock la última versión estable era la 0.3.1. Por ello hemos decidido probar esta versión de RADclock mediante una serie de pruebas de transmisión comprobar que precisión nos permite obtener.

3.3. Preparación e instalación del kernel

El reloj RADclock requiere para su funcionamiento el uso de un kernel determinado modificado. En [2], podemos encontrar diferentes versiones del software en las que se detalla sobre qué kernels pueden funcionar.

A continuación explicaremos los pasos a realizar para preparar el kernel de una máquina en la que queremos instalar RADclock.

3.3.1. Modificación del Kernel

Primero descomprimimos el software de RADclock (por ejemplo en `/tmp`).

Descomprimimos (en `/usr/src`) el kernel que queramos utilizar (dentro de la lista de posibles kernels que soportan RADclock). En este caso hemos decidido utilizar el kernel 2.6.32 por ser el más moderno de los que la versión de RADclock empleada permite utilizar.

Una vez descomprimido el archivo tendremos un directorio con el siguiente nombre: `linux-2.6.32`. Creamos un enlace simbólico:

```
ln -s linux-2.6.32 linux
```

Desde el directorio `/usr/src/linux` parchemos el kernel con el siguiente comando:

```
patch –p1 < /tmp/radclock-0.3.1/kernel-
patches/linux/2.6.32/*
```

Ahora el kernel está preparado para soportar RADclock.

3.3.2. Compilación del Kernel

Primero creamos el archivo de configuración del kernel mediante alguno de los siguientes comandos:

`make menuconfig`	(menú tipo consola)
`make gconfig`	(menú tipo ventana [Gtk], funciona mejor en un escritorio Gnome)
`make xconfig`	(menú tipo ventana [Qt], funciona mejor en un escritorio KDE)

Nos aseguramos que todas las opciones (64-bit o 32-bit kernel, *processor family*, por ejemplo) sean las correctas para esa máquina. También debemos asegurarnos que la opción “*Kernel support for the RADclock”*, en la sección “*DeviceDrivers -> RADclock Driver”*, está marcada con una “y”. En caso contrario el kernel no tendrá el soporte para RADclock instalado. Esta opción es la modificación al kernel realizada en el paso anterior. En los anexos 3.5 y 3.6 del DVD que acompaña este TFC podemos encontrar los archivos de configuración del kernel de las máquinas utilizadas.

A continuación compilamos el kernel:

```
make
make modules
su
make modules_install
make bzImage
make install
```

Estos comandos crean una serie de archivos en `/boot`:

```
System.map-2.6.32
vmlinuz-2.6.32
```

Creamos la imagen initrd:

```
cd /boot
mkinitrd –o initrd.img-2.6.32 2.6.32
```

Modificamos el gestor de arranque (Grub en este caso) para añadir el nuevo kernel:

```
nano /boot/grub/menu.lst
```

En él creamos una nueva entrada:

```
title 2.6.32 --   → (Nombre del kernel)
root           (hd0,2)→ (dispositivo donde está instalado)
kernel          /boot/vmlinuz-2.6.32
initrd          /boot/initrd.img-2.6.32
```

Reiniciamos la maquina. Si todo ha funcionado correctamente podremos arrancar con el nuevo kernel.

3.4. Puesta en marcha de RADclock

Llegados a este punto solo tenemos que instalar y configurar RADclock.

3.4.1. Instalación

Una vez compilado el kernel procedemos con la instalación del RADclock.
Primero nos aseguramos de que las librerías libpcap y libnl están instaladas. También debemos asegurarnos de que el soporte en el kernel está funcionando:

```
cd /tmp/radclock-0.3.1
./configure
```

Debemos obtener la siguiente salida:

```
configure: ----------------------------------------
configure: RADClock version 0.3.1
configure: Found target arch: x86_64
configure: Compiling for linux
configure: Detected kernel RADClock extensions: yes
configure: Detected kernel access channel: yes
configure: get_counter syscall number: 299
configure: get_counter_latency syscall number: 300
configure: ----------------------------------------
```

Procedemos con la instalación del *daemon* RADclock:

```
make
sudo make install
```

Creamos el *start-up script* en `/etc/init.d/radclock` (ver anexo 3.1) y añadimos la siguiente configuración en `/etc/logrotate.conf`:

```
/var/log/radclock.log {
    missingok
    size 100k
    compress
    create 0644 root root
    rotate 7
    postrotate
    kill -USR1 `cat /var/run/radclock/radclock.pid`
    endscript
}
```

RADclock está listo para ser configurado.

3.4.2. Configuración

Lo primero que hacemos es comprobar la lista de fuentes de *clock* disponibles:

```
cat
/sys/devices/system/clocksource/clocksource0/available_cloc
ksource
```

Obtenemos la siguiente lista:

```
tsc  hpet  acpi_pm
```

Siguiendo las recomendaciones de [2] escogemos el tsc:

```
echo “tsc” >
/sys/devices/system/clocksource/clocksource0/current_clocksource
```

Ejecutamos el script `/etc/init.d/radclock`. Esto nos crea un archivo de configuración en `/etc/radclock.conf` (ver anexo 3.2), cuyos principales parámetros configurables son:

- **verbose_level** → para controlar la cantidad y tipo de mensajes que radclock imprimirá por pantalla.
- **synchronisation_type** → tipo de funcionamiento:
  - Piggy (*piggybacking*) → funciona simultáneamente a otro *daemon* NTP (contra el mismo servidor NTP). En este modo no podemos utilizar RADclock para servir a clientes NTP ni para ajustar el reloj del sistema debido a que será el otro *daemon* NTP el que se encargue de ello. Solo utilizaremos el reloj RADclock como referencia para otros programas.
  - NTP →RADclock puede funcionar por si solo como cliente NTP utilizando el protocolo NTP (substituyendo al ntpd).
- **polling_period** → tiempo en segundos entre peticiones al servidor.
  - En modo *piggybacking* el autor [2] recomienda que sea el mismo que en el otro *daemon* NTP.
  - A menor tiempo mejor sincronización.
  - En general y según los resultados experimentales del autor [2] un valor de 16 segundos ofrece equilibrio entre buena precisión y carga al servidor.
- **hostname** → nombre o IP de la máquina local.
- **time_server** → servidor NTP al que apuntamos (nombre o IP).
- **ipc_server** → permite que otros programas utilicen el reloj de RADclock.
- **ntp_server** → permite utilizar RADclock como servidor NTP de 2º nivel (*stratum* 2) para clientes remotos.
- **adjust_system_clock** → permite que RADclock ajuste el reloj del sistema (no compatible con el método "*piggybacking*".
- **virtual_machine** → para ejecutar RADclock en una maquina virtual (xen o vbox):
  - xen-master: radclock crea tiempo para todos los sistemas *Xen*.
  - xen-slave: radclock obtiene tiempo de un *Xen master*.
  - vbox-master: crea tiempo para todos los sistemas *Virtual Box*.
  - vbox-slave: radclock obtiene tiempo de un *Virtual Box master*.
- **temperature_quality** → para indicar la calidad del entorno (temperatura y calidad del hardware).
- **network_device** → interface de red a utilizar.

Los campos "**hostname**" y "**time_server**" tienen que tener un valor para que el sistema pueda funcionar.

Estos son los ficheros de entrada y salida de datos:

- **sync_input_pcap** → para "reproducir" un dialogo NTP anterior (requiere un archivo pcap producido con RADclock)
- **sync_input_ascii** → para "reproducir" un dialogo NTP anterior (requiere un archivo producido con RADclock)
- **sync_output_pcap** → dialogo NTP en formato pcap modificado
- **sync_output_ascii** → dialogo NTP en formato ASCII
- **clock_output_ascii** → valores temporales del dialogo (tiempos, RTT, valores RADclock, error…)

3.4.3. Primera prueba de funcionamiento

Una vez instalado y configurado podemos encontrar el ejecutable de RADclock en la carpeta `/usr/local/bin/`. Desde ella lo ejecutamos:

```
radclock
```

Al ejecutarlo nos imprime por pantalla una serie de mensaje de información que luego podemos encontrar en el fichero radclock.log (sin los mensajes tipo *Sync* y *Control*, ver el anexo 3.3).

Al parar el *daemon* (mediante el comando `kill -KILLALL`) termina de imprimir el resto de mensajes tipo *Info*.

Estos mensajes de información nos muestran los valores de los diferentes parámetros configurados y nos informan de posibles errores durante la ejecución.

Los mensajes tipo *Sync* y *Control* del archivo `radclock.log` muestran información del dialogo NTP establecido con el servidor (*timestamps*, RTT, errores calculado por RADclock, etc.)

3.4.4. Registros de control (logs)

En estos archivos de LOG encontramos primero una lista que indica el valor que representa cada columna.

3.4.4.1. sync_output_ascii

Genera un documento de texto ASCII, que contiene una línea de datos por cada par de mensajes NTP petición y respuesta (o *stamp*):

```
% column 1 - Ta [vcount]
% column 2 - Tb [sec]
% column 3 - Te [sec]
% column 4 - Tf [vcount]
% column 5 - OUT src port
61072663064468 1317134300.671778658 1317134300.672025862
61072668269633 47504

61078663316873 1317134302.671772562 1317134302.672019684
61078668451163 47504

61084663592535 1317134304.671867949 1317134304.672114824
61084668631553 47504

61090663867403 1317134306.672217538 1317134306.672449999
61090669561133 47504

61096664161058 1317134308.672058758 1317134308.672288511
61096668991230 47504
…
```

- El primer valor muestra el tiempo de salida del host de la petición NTP. Utiliza una variable propia de RADclock (struct vcount) creada a partir de la estructura "***uint64_t***" (unsigned long int) de la librería stdint.h. Este tiempo se calcula trasladando los segundos del tiempo en NTP a los 32 bits de mayor peso y el valor de la fracción de segundos a los 32 bits de menor peso.
- La columna número 2 muestra el tiempo de llegado al servidor NTP en el formato utilizado por NTP (32 bits para los segundos, y 32 más para fracciones de segundos).
- La tercera columna muestra el tiempo de salida de la respuesta del servidor, también en formato NTP.
- La cuarta columna (que aparece en otra línea por problemas de espacio) es el tiempo de llegada de la respuesta al host. También con la estructura vcount.
- La última columna es el puerto utilizado por el host para el dialogo NTP.

3.4.4.2. sync_output_pcap

Genera un documento en formato .pcap modificado, substituyendo la cabecera Ethernet por una cabecera propia (sll) donde guarda solo la información que necesita. Con este documento podemos observar el diálogo NTP mediante cualquier analizador de protocolos (Ethereal, Wireshark, SuperAgent, etc).

3.4.4.3. clock_output_ascii

Al igual que sync_output_ascii genera una entrada de datos por cada *stamp*. Pero estos datos no son solo del dialogo NTP en sí, sino que además incluye datos del reloj.

```
% NTP packet filtering run with:
%
% column 1 - Tb
% column 2 - Tf
% column 3 - RTT
% ---------------------------------------
% column 4 - phat
% column 5 - plocal
% column 6 - C
% column 7 - thetahat
% ---------------------------------------
% columns 8--10 - RTThat, RTThat_new, RTThat_sh
% columns 11--17 - th_naive, minET, minET_last,
RADclockout, RADclockin, errTa, errTf
% columns 18--22 - perr, plocalerr, wsum, best_Tf, clock
status
%
1317134300.671778658 61072668269633 5205165 3.333303552e-10
3.333303552e-10 1317113943.29930636939 0.0000000000 5205165
0 0 0 0 0 1317134300.67177865840 1317134300.67351369793
0.0000000000 0.0000000000 0 0 0 61072668269633 19

1317134302.671772562 61078668451163 5134290 3.333202571e-10
3.333202571e-10 1317113943.91602380457 0.0007438916 5134290
0 0 0.000743891552 0 0 1317134302.67178433272
1317134302.67349569558 -0.0007321203 0.0007321203 1.18121e-
05 0 1 61078668451163 7

1317134304.671867949 61084668631553 5039018 3.333283998e-10
3.333283998e-10 1317113943.41868064424 0.0007179167 5039018
0 0 0.000695054841 0 0 1317134304.67184661736
1317134304.67352626519 -0.0007392483 0.0006935246 1.38451e-
05 0 1.88015 61084668631553 7
…
```

- Los 2 primeros valores son los mismos que el 2 y 4 de la salida anteriormente explicada (sync_output_ascii).
- El tercer valor es el RTT del par medido en vcount.
- El valor de phat es una estimación del periodo del oscilador del servidor NTP.
- plocal es una estimación del periodo del oscilador local.
- El valor temporal C es una constante de alineamiento con el reloj de origen. Es un desplazamiento temporal para corregir el error de fase.
- El siguiente valor (thethahat) corresponde a la deriva del reloj (±Δt).
- RTThat es una estimación del RTT mínimo.

- RTThat_new es una estimación del RTT para la próxima ventana.
- th_naive es una estimación menos precisa de la deriva (thethahat).
- Los valores minET y minET_last son el error en la estimación de la deriva.
- RADclockout es el valor temporal del RADclock en el instante de salida del paquete NTP (en formato NTP). (Corresponde a Ta)
- RADclockin es el valor temporal del RADclock en el instante de recepción de la respuesta del servidor (en formato NTP). (Corresponde a Tf)
- A continuación, errTA y errTf nos indican el error cometido en la previsión de los tiempos Ta y Tf.
- Los valores perr y plocalerr son la estimación del error total de phat y plocal.
- El tiempo best_Tf es el Tf con menor error obtenido hasta el momento (con menor error en la previsión).
- Finalmente con *clock* status nos indica el modo de funcionamiento de RADclock en ese momento.

La máquina de estados se explica más detalladamente a continuación.

3.4.5. Máquina de estados

RADclock dispone de una máquina de estados que indica las funciones que está realizando en cada momento. Esta máquina dispone de una serie de *flags*:

- 0x00000001 (1): **STARAD_UNSYC**. RADclock aun no está sincronizado (justo al empezar, aun sin dialogo con el servidor NTP).
- 0x00000002 (2): **STARAD_WARMUP**. Fase inicial, la estimación del error es poco fiable. Este *flag* se mantiene activo en los primeros 480 *stamps*.
- 0x00000004 (4): **STARAD_KCLOCK**. Reloj RADclock ya está sincronizado y es fiable. Tras los primeros 25 *stamps* se activa este *flag*.
- 0x00000008 (8): **STARAD_SYSCLOCK**. Indica que el TSC es muy preciso (bien ajustado por las predicciones de RADclock).
- 0x00000010 (16): **STARAD_STARVING**. Falta de entrada de datos de calidad (aun no tenemos ningún valor temporal enviado por un reloj GPS). Este *flag* está activo al inicio del dialogo y después de una caída larga de la conexión con el servidor NTP.
- 0x00000020 (32): **STARAD_PERIOD_QUALITY**. La estimación del periodo de RADclock es “pobre” (mucho error en la estimación).

- 0x00000040 (64): **STARAD_PERIOD_SANITY**. Se están realizando correcciones del periodo de forma consecutiva.
- 0x00000080 (128): **STARAD_OFFSET_QUALITY**. La estimación del *offset* de RADclock es “pobre” (mucho error en la estimación).

  0x00000100 (256): **STARAD_OFFSET_SANITY**. Se están realizando correcciones del *offset* de forma consecutiva.

En el anexo 3.4 podemos encontrar más información sobre la máquina de estados de RADclock.

3.4.6. Modos de funcionamiento

A través de los parámetros del archivo del archivo **radclock.conf** podemos configurar RADclock para que funcione de 2 maneras:

- *Modo piggybacking*:

  Como ya se ha comentado anteriormente en este modo el RADclock no se encarga de realizar el dialogo NTP con el servidor ni de ajustar el TSC. En este caso RADclock genera un reloj software a partir de un diálogo NTP creado por otro *daemon* (por ejemplo ntpd). Basándose en este diálogo RADclock realiza sus predicciones y ajustes ofreciéndonos un reloj software que podemos utilizar desde otras aplicaciones en las que utilicemos su API.

- Modo NTP:

  En este modo es el propio RADclock el que genera el diálogo NTP y también se encarga de ajustar el TSC. El reloj software también se puede utilizar desde otras aplicaciones y además en este modo de funcionamiento podemos utilizar la máquina donde está funcionando RADclock como un servidor NTP de *stratum* 2.

- Modo “RADclocks en cadena”

  Es un modo creado *ad hoc* durante este TFC. Hemos realizado una prueba en la cual se han configurado RADclock para dos máquinas, con la primera (servgenmonII) funcionando en modo NTP y la segunda (servgenmonI) en modo *Piggybacking* utilizando como servidor NTP el RADclock de la primera máquina. Otra opción sería configurar la segunda máquina también en modo NTP contra el RADclock de la primera.

CAPÍTULO 4. PRUEBAS REALIZADAS

En este capítulo analizaremos algunas de las distintas pruebas realizadas con las dos soluciones que estudiamos en este TFC. En estas pruebas comprobaremos los *timestamp* obtenidos al realizar capturas con las DAG y con las NIC de máquinas en las que se ejecuta RADclock. A raíz de los resultados observaremos un histograma del tiempo entre paquetes, así como la variación del jitter a lo largo de la transmisión. Para calcular el jitter usamos la expresión definida para el protocolo RTP [8].

4.1. Cálculo del Jitter

RTP (*Real-time Transport Protocol*) es un protocolo de transporte en tiempo real que añade algunos campos a UDP para poder hacer transmisiones de información audiovisual. Desde nuestro punto de vista lo más interesante es que RTP añade marcas temporales en cada paquete, que corresponden al momento de muestreo del audio o vídeo que se transporta. La marca o *timestamp* está referida a una base de tiempos que suele coincidir con la frecuencia de muestreo de la señal audiovisual (por ejemplo, 8000 muestras/segundo en el caso de audio telefónico de ancho de banda de 4 KHz), y por tanto, depende del códec concreto.

RTP suele ir acompañado por otro protocolo, RTCP (*Real-Time Control Protocol*), que envía periódicamente informes de cómo va la transmisión, informando de aspectos como cuantas pérdidas de paquete se han producido, cuantos bytes se están enviando, y una estimación del jitter de la transmisión. En principio el jitter debería calcularse como la varianza de la variable aleatoria “tiempo entre llegada de paquetes” (delta), de la siguiente manera:

Dado una serie X_i de N valores y su media, la desviación estándar σ se calcula según la siguiente expresión:

$$\overline{X} = \frac{\sum_{i=1}^{N} X_i}{N} \qquad \textbf{(4.1)}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2} \qquad \textbf{(4.2)}$$

Sin embargo cualquier expresión que involucre productos y raíces cuadradas es costosa desde el punto de vista computacional, y si queremos trabajar a tasas del orden de miles de paquetes por segundo, como puede pasar en una transmisión audiovisual, el coste de cálculo puede ser prohibitivo (especialmente si tenemos en cuenta que el cálculo del jitter no es una

operación crítica, sino una información más ofrecida por RTP/RTCP). Por ello el RFC 3550 define la siguiente aproximación:

Dado el paquete i-ésimo, S_i es el *timestamp* RTP y R_i es su tiempo de llegada expresado en unidades de *timestamp* RTP, entonces para 2 paquetes i y j, el delta time D puede expresarse como:

$$D_{(i,j)} = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i) \quad \textbf{(4.3)}$$

El jitter (J) acumulado debe ser calculado de forma continua para cada paquete i recibido de esa fuente, utilizando la diferencia D de ese paquete con el anterior (i-1) en orden de llegada de acuerdo con la expresión:

$$J_{(i)} = J_{(i-1)} + (|D_{(i-1,i)}| - J_{(i-1)})/16 \quad \textbf{(4.4)}$$

Es decir, calculamos el jitter de cada paquete teniendo en cuenta el jitter del paquete anterior y una pequeña parte (1/16) de la diferencia entre el *delta time* del paquete actual y el jitter del anterior.

4.2. Características de las pruebas

Para generar el tráfico hemos decidido utilizar MGEN [9] por ser un software de uso sencillo que nos permite enviar tráfico de distintos tipos sobre diferentes protocolos. En nuestro caso, el tráfico que generaremos será periódico, sobre UDP[8] para todos los escenarios, estudiando varios casos. Por un lado utilizamos paquetes de 576 bytes y de 1500 bytes (MTU Ethernet[9]). Por otro lado, en todos los casos se ha realizado la transmisión a dos velocidades distintas. Primero enviamos a una velocidad tal que suponga la utilización del 10% del enlace y después a una velocidad que corresponda al 50% de la capacidad del enlace. De esta manera comprobaremos si afecta (y si lo hace, en qué medida) el tamaño de los paquetes así como la ocupación del enlace.

Con estas pruebas también estudiaremos como afecta RADclock a la hora de enviar datos. Para ello hemos enviado tráfico de tres formas distintas: desde una máquina sin control alguno sobre su TSC, desde la misma máquina sincronizada mediante ntpd y finalmente la máquina sincronizada mediante RADclock. En todos los casos, en las máquinas donde se estaba ejecutando RADclock este se encontraba con su máquina de estados variando entre los estados 4 y 132 (ver anexo 3.4)

Las capturas realizadas en emisión (en todos los casos) y en recepción (en el caso de las realizadas con RADclock) se han efectuado mediante el comando `dumpcap` [10] que es la herramienta que utiliza wireshark para realizar capturas, pero ejecutada desde comando (sin interfaz gráfica).

[8] Para controlar con precisión el envío de paquetes tenemos que utilizar siempre UDP y nunca TCP, ya que con este último no podemos controlar desde el sistema operativo al 100% el momento de salida de los paquetes, debido al algoritmo de Nagle [11].

[9] *Maximum Transfer Unit*, este es el máximo tamaño de datagrama que se puede enviar [12, 13].

4.3. Resultados DAG

A continuación presentamos algunos de los resultados obtenidos en las pruebas realizadas con las tarjetas DAG. En ellas han intervenido tres PCs, servgenmonI, servgenmonII y radclockserv (anexo 4.1) y el switch SMC 8624T. ServgenmonII y radclockserv han sido utilizados para generar el tráfico con MGEN y es donde ejecutamos RADclock o ntpd. Desde servgenmonI realizamos la captura con la DAG. Puesto que no se trata de tarjetas NIC no podemos generar el tráfico directamente hacia ellas, por ello, el tráfico generado se envía a servgenmonI. Para poder monitorizar ese tráfico hemos configurado el switch para que realice un *port mirroring* y envíe por el puerto donde está conectada la DAG todo lo que envía a servgenmonI.

En este apartado estudiaremos solo la capacidad de captura de la DAG, dejando el estudio de cómo afecta RADclock a la emisión de datos para el apartado 4.2 Concretamente analizaremos las pruebas realizadas con radclockserver como emisor.

4.3.1. Escenario 1

En este escenario radclokserv está funcionando con su propio TSC, sin la ayuda de ninguna fuente externa.

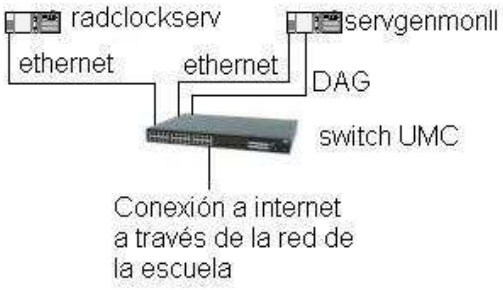

Figura 4.1. Escenario 1 DAG

Analizaremos la prueba realizada con un 10% de la ocupación del enlace (en este caso eso equivale a 10 Mbps) con paquetes de 1500 Bytes, lo que supone 833.3 paquetes/s = 10 Mbit/s / (1500 x 8 bits). Se han enviado 20834 paquetes durante 25 segundos. Con estas características radclockserver envía (`PERIODIC [833 1458]`[10]):

[10] Este es el comando utilizado en MGEN para indicar el tráfico que queremos enviar. PERIODIC indica el tipo de tráfico que queremos enviar y entre corchetes indicamos el número de paquetes por segundo y su tamaño.

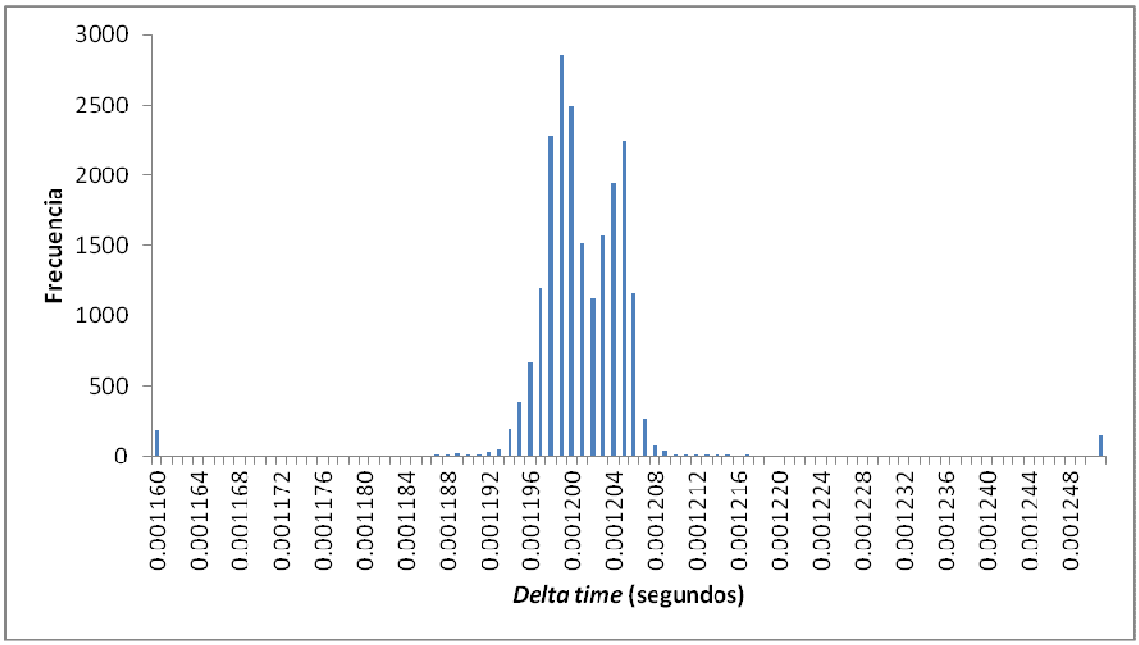

Fig. 4.2 Histograma

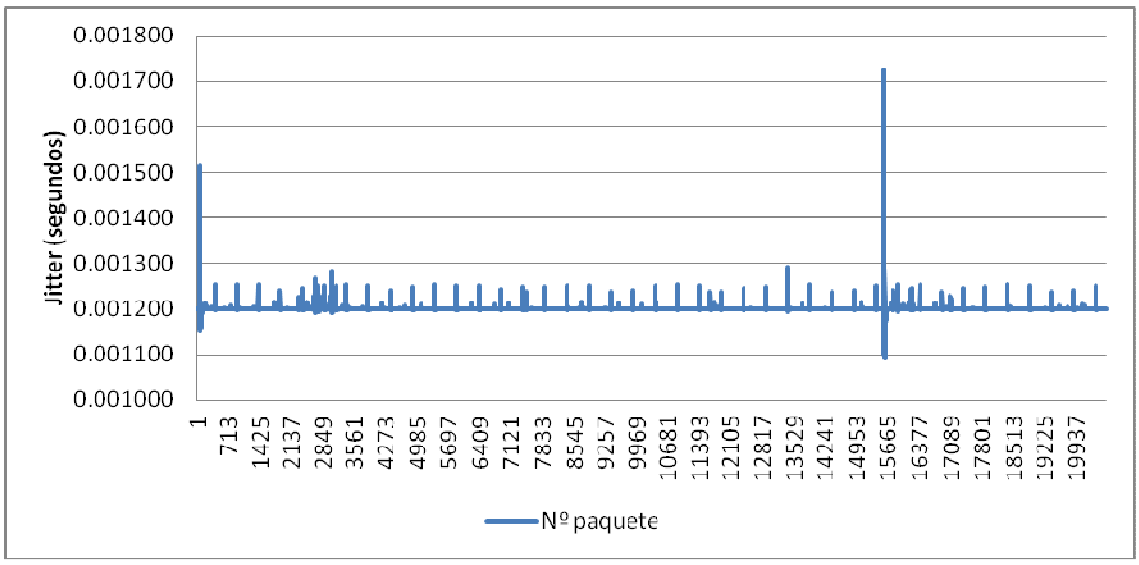

Fig. 4.3 Jitter acumulado

En estas gráficas podemos observar que los paquetes han sido enviados con un *delta time* medio de 1200µs tal y como esperábamos s (1500 x 8 bits / 10 Mbit/s = 1.2 ms). El jitter se ha mantenido bastante estable a lo largo de la transmisión, con variaciones pequeñas (de 40 a 80 µs) frecuentes y algunas mayores (de 0.3 a 0.5 ms), estas variaciones pueden ser causadas por la propia máquina (si el SO tiene que atender otras interrupciones), por la red (si hay otros tráficos circulando en ella ajenos al nuestro) o incluso por el switch (si por el pasa una gran cantidad de tráfico)

Y la tarjeta DAG captura lo siguiente:

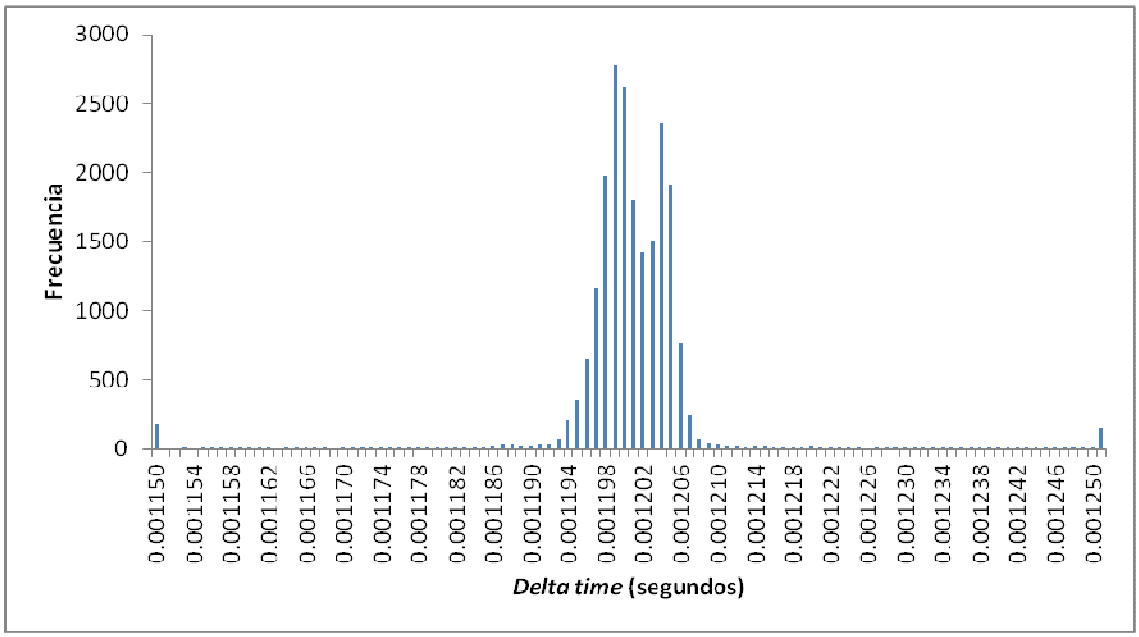

Fig. 4.4 Histograma

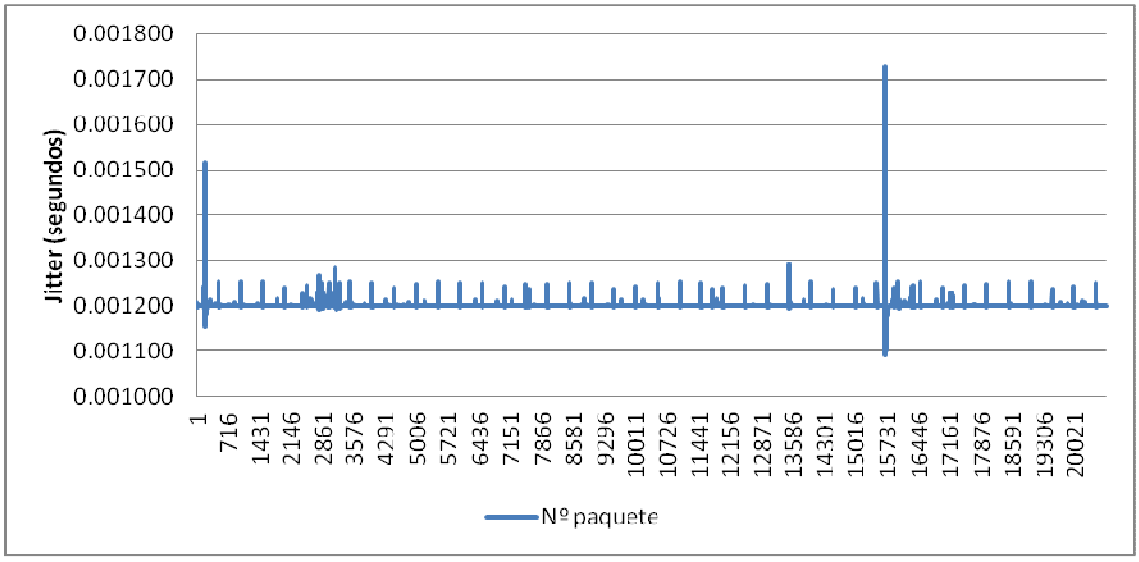

Fig. 4.5 Jitter acumulado

Como podemos ver en las gráficas, los datos obtenidos por el monitor de tráfico son prácticamente idénticos a los capturados por el emisor, lo cual nos indica que la tarjeta DAG ha capturado los paquetes sin añadir prácticamente ningún tipo de retardo de procesado tal y como esperábamos. Y dada la transparencia, el switch SMC queda descartado como fuente de jitter.

4.3.2. Escenario 2

Para este caso hemos activado el *daemon* ntpd en radclockserv utilizando el reloj RADclock de servgenmonII como servidor de referencia. Este a su vez utiliza como referencia el servidor ntp.cesca.cat al que accedemos a través de internet.

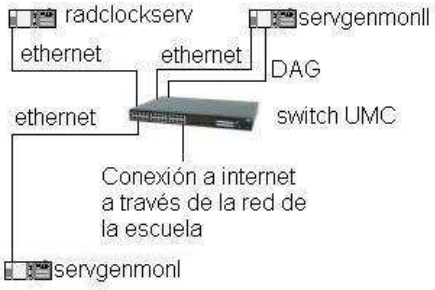

Fig. 4.6 Escenario 2 DAG

En esta ocasión, analizaremos la prueba realizada con paquetes de 1500 bytes, pero en este caso con una utilización del 50% del enlace (4166 paquetes/s durante 6 segundos, con un total de 24996 paquetes enviados). En este caso radclockserv envía (`PERIODIC [4166 1458]`):

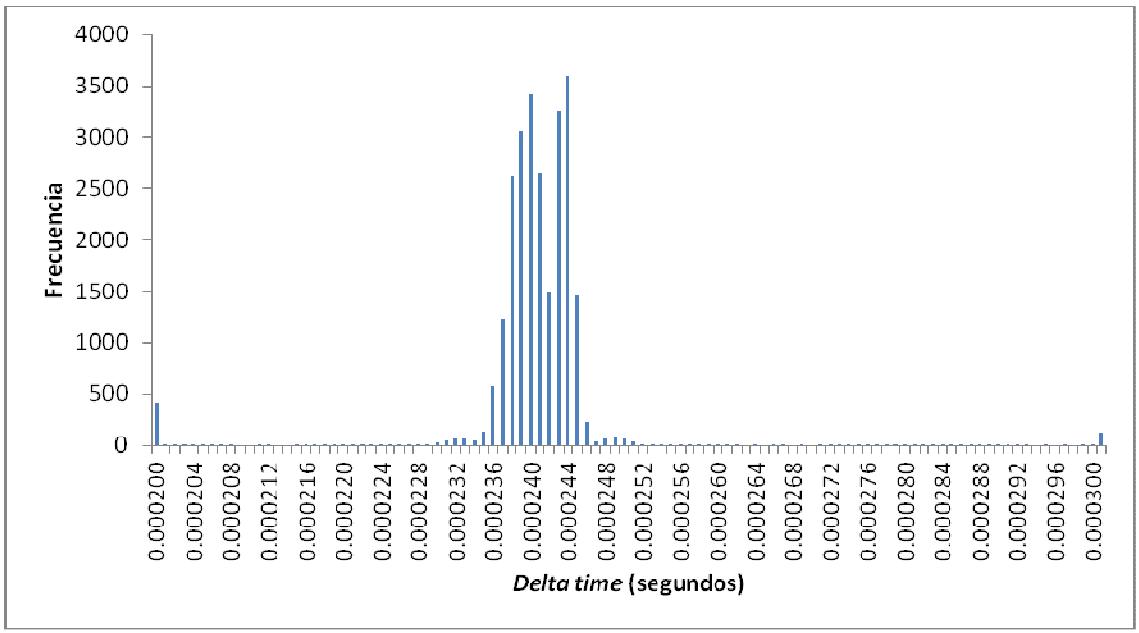

Fig. 4.7 Histograma

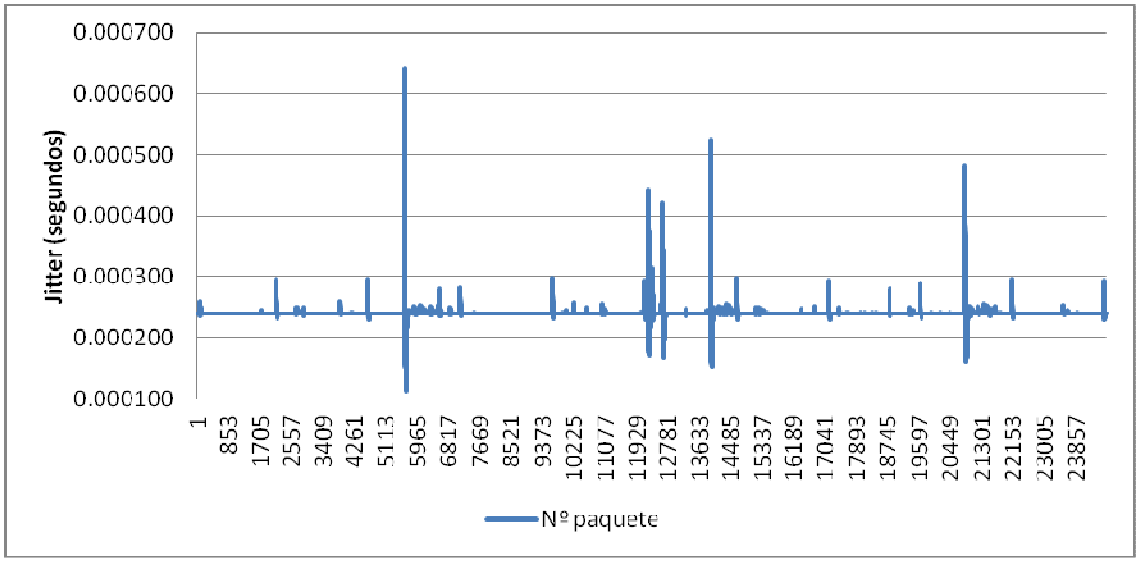

Fig. 4.8 Jitter acumulado

De nuevo los paquetes han sido enviados con el tiempo entre paquetes que se esperaba (alrededor de 240 µs. Podemos observar que el jitter ha sido muy estable, pese a tener más variaciones de 0.2 a 0.5 ms ha sufrido menos variaciones en general que en el escenario anterior. Esto puede ser debido a la sincronización mediante NTP.

Y la tarjeta DAG captura lo siguiente:

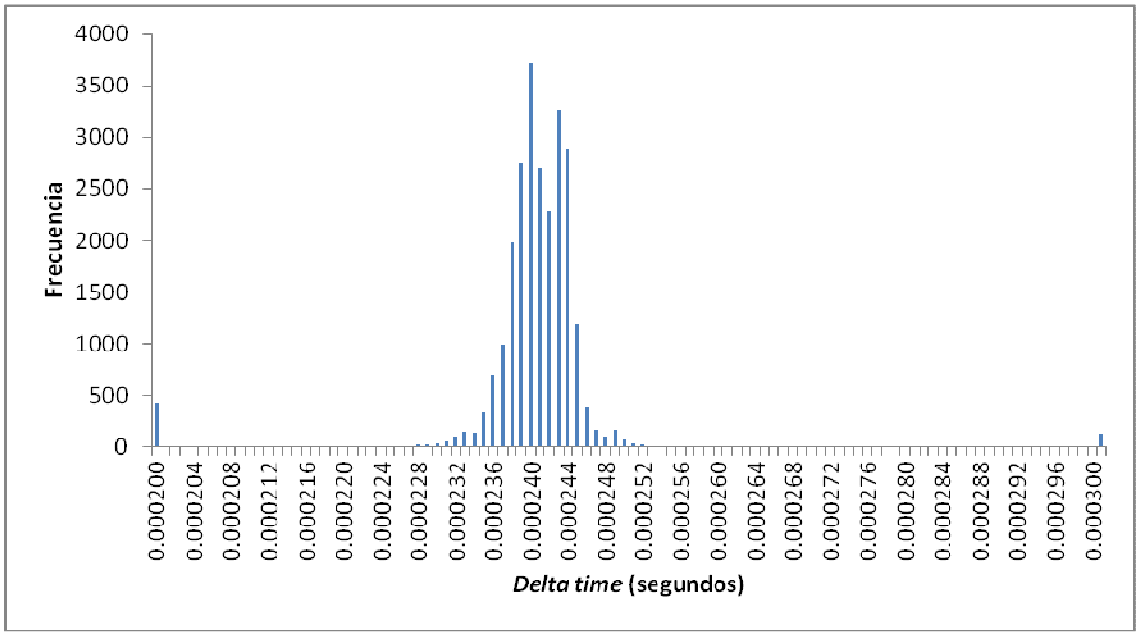

Fig. 4.9 Histograma

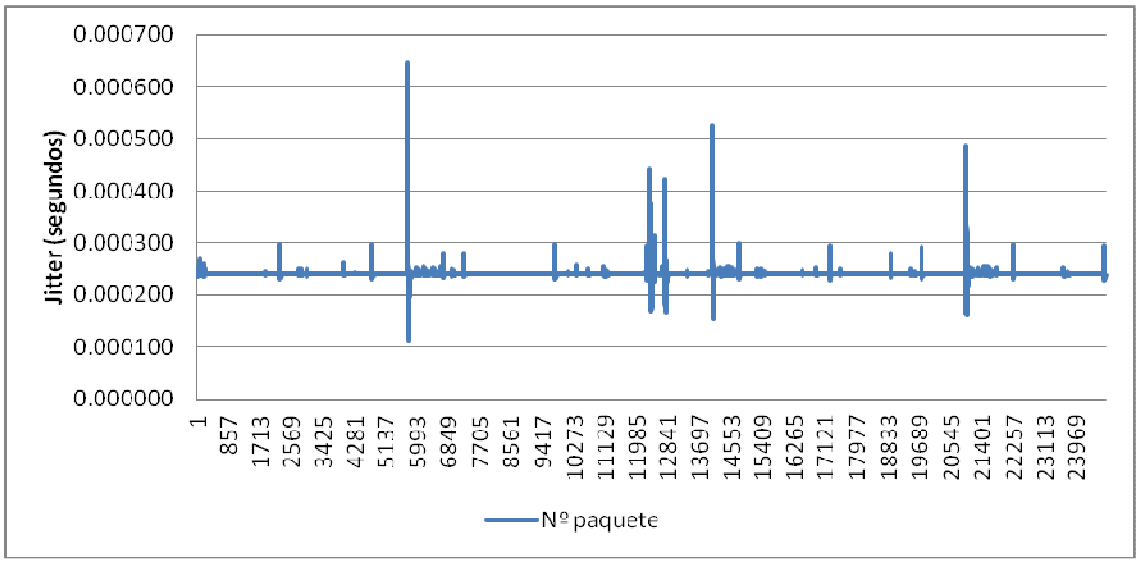

Fig. 4.10 Jitter acumulado

Como podemos observar en las gráficas, una vez más el tiempo entre paquetes capturados por la DAG no ha variado sensiblemente respecto al origen.

4.3.3. Escenario 3

Finalmente, en este último escenario se ha puesto en marcha el reloj RADclock en radclockserv. El servidor temporal utilizado es el reloj de *stratum1* de ntp.cesca.cat al que accedemos a través de internet.

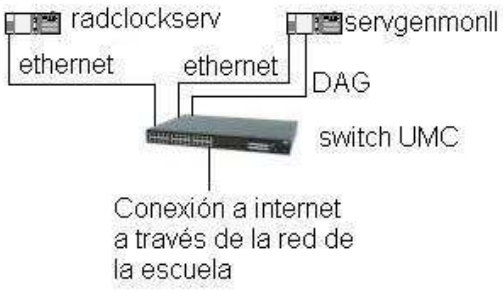

Figura 4.11. Escenario 3 DAG

En este caso mostraremos los resultados obtenidos en la prueba realiza al 10% de la utilización del enlace (10 Mbps) y paquetes de 576 Bytes (2170 paquetes por segundo durante 10 segundos, para un total de 21700 paquetes). Esta vez radclockserv envía (`PERIODIC [2170 534]`):

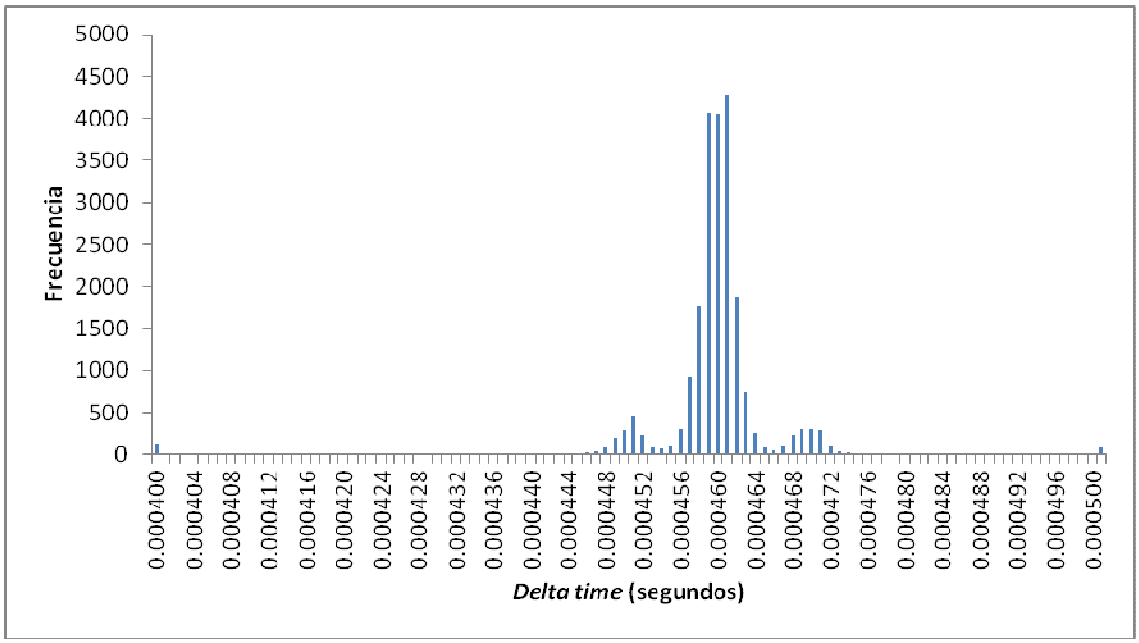

Fig. 4.12 Histograma

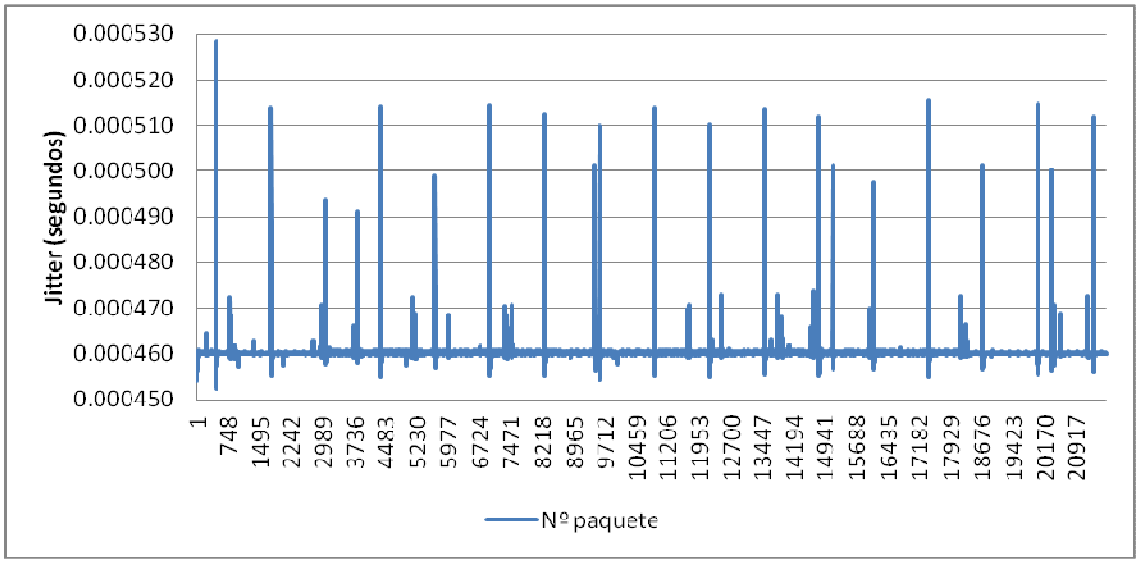

Fig. 4.13 Jitter acumulado

Las gráficas muestran que, una vez más, el tiempo entre paquetes ha sido el esperado (alrededor de los 460 µs). Además en este caso podemos ver que el jitter ha sido muy estable. Pese a tener muchas variaciones a lo largo del tiempo, ninguna supera los 70 µs mientras que en otros caso llegaba a valores de casi 500 µs.

Y servgenmonI recibe:

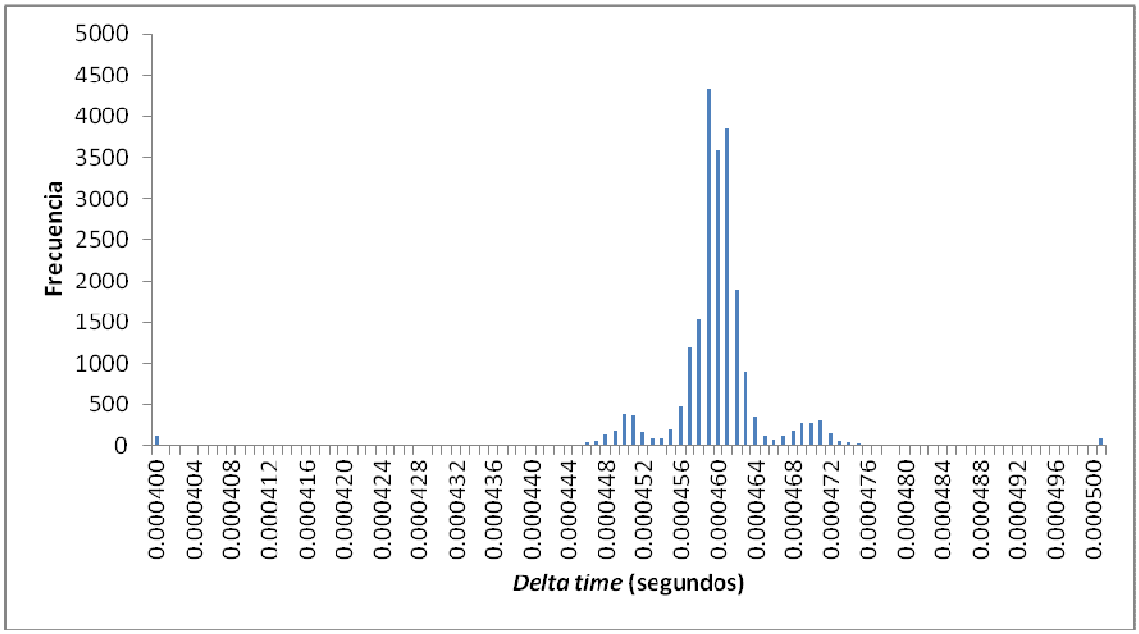

Fig. 4.14 Histograma

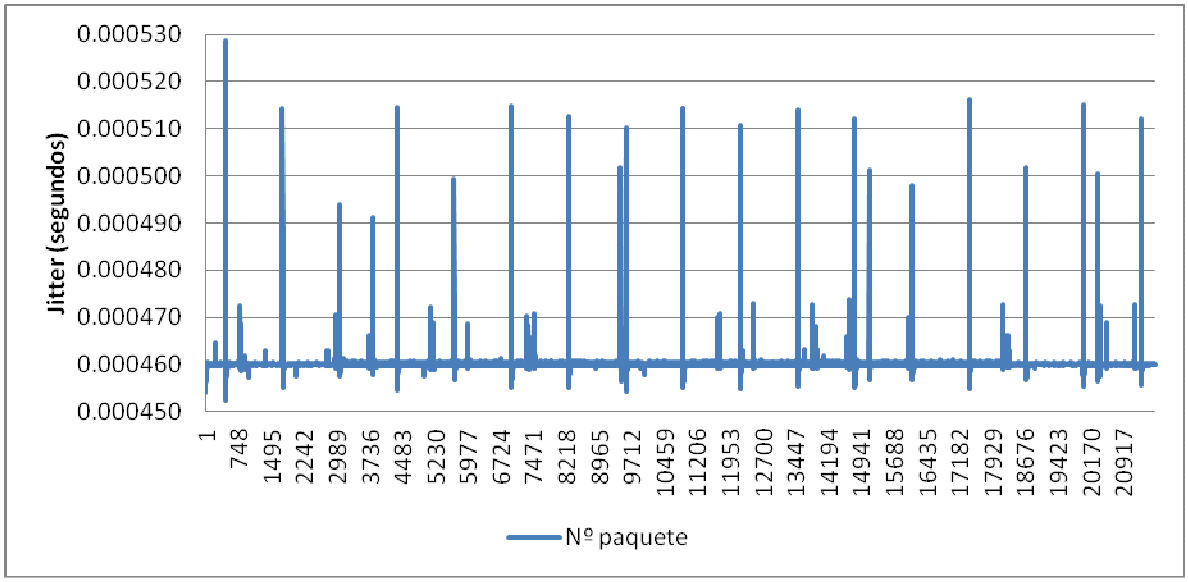

Fig. 4.15 Jitter acumulado

Los valores de delta time obtenidos por la DAG han variado un poco respecto a los obtenidos por la máquina emisora, lo cual muestra una pequeña diferencia en el reloj, pero pese a ello podemos afirmar que el tiempo entre paquetes no se ha visto afectado por el procesado de la DAG.

4.4. Resultados RADclock

En este apartado presentamos algunos de los resultados obtenidos en las pruebas realizadas con el software RADclock. En ellas han intervenido tres PCs, servgenmonI, servgenmonII y radclockserv (anexo 4.1) y el switch ZyXEL GS1100-16 (más sencillo que el SMC, y sin capacidad de *port mirroring*, que no es necesaria en este caso). ServgenmonI y radclockserv han sido utilizados para generar el tráfico con MGEN y es donde ejecutamos RADclock o ntpd. El tráfico que generan tiene como destino servgenmonII donde tenemos activado el reloj RADclock sincronizado con el servidor de *stratum 1* ntp.cesca.cat.

En este apartado, además de ver la capacidad de captura de una máquina con RADclock y una NIC convencional nos fijaremos en cómo ha afectado el uso de este software en la emisión de los datos.

En los apartados siguientes mostraremos los resultados obtenidos en las pruebas realizadas con servgenmonI enviando paquetes de 1500 bytes a servgenmonII a una velocidad de 500 Mbps que corresponde a una ocupación del 50% del enlace (`PERIODIC [41666 1458]`), obteniendo 41666 paquetes por segundo durante 1 segundo. Hemos decidido utilizar estas pruebas por ser aquellas que muestran mejor los resultados obtenidos.

4.4.1. Escenario 1

En este escenario servgenmonI funciona con su propio TSC, sin la ayuda de ninguna fuente externa.

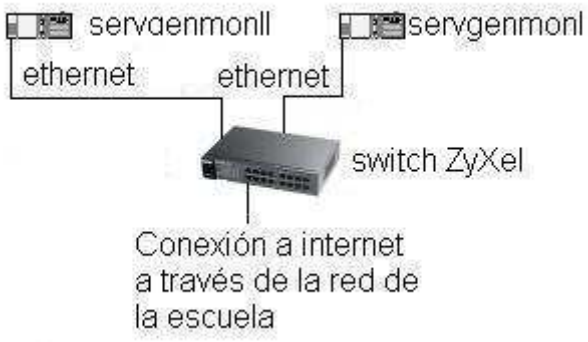

Fig. 4.16 Escenario 1 RADclock

La captura realizada en emisión es la siguiente:

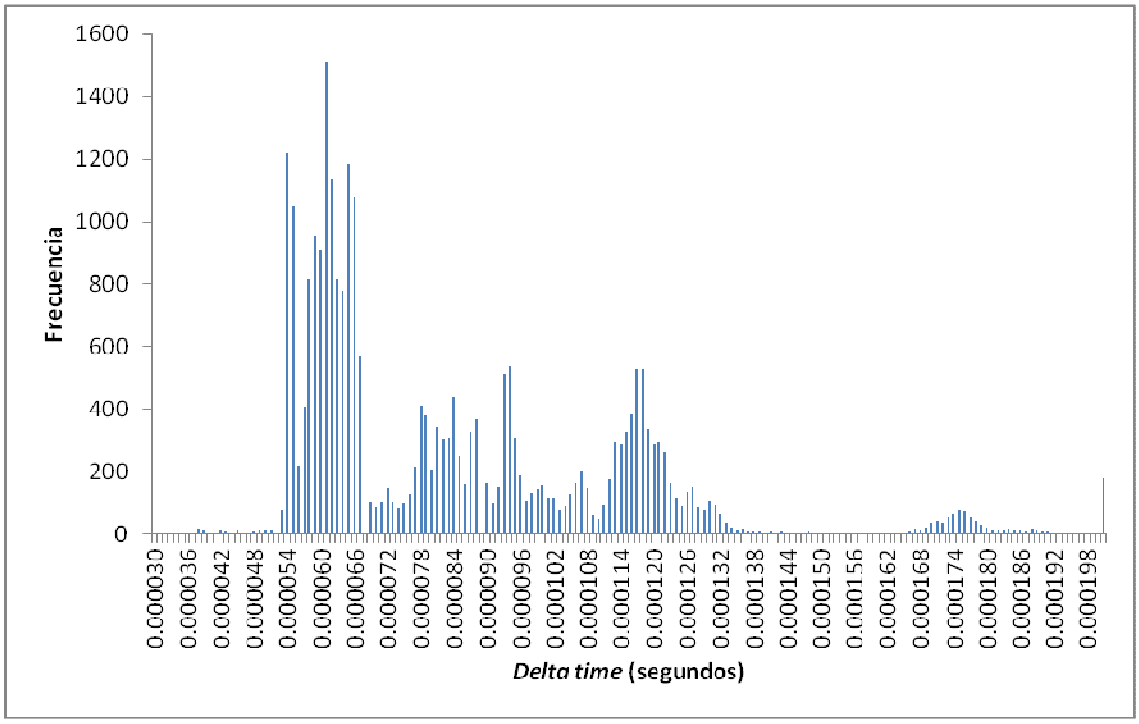

Fig. 4.17 Histograma

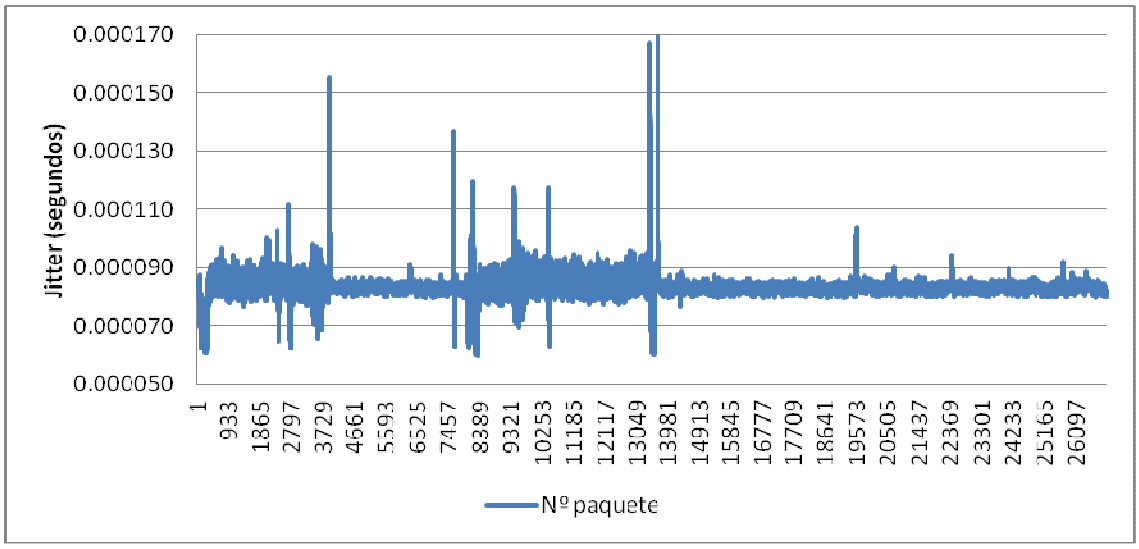

Fig. 4.18 Jitter acumulado

En este caso podemos ver que a esta velocidad y con tamaño de paquetes MGEN no consigue enviar los paquetes de una forma periódica, sino que, el tiempo entre paquetes varía mucho, y es superior al esperado. Esto seguramente será a causa del software empleado, que al tener que generar muchas interrupciones en el SO se desestabiliza.

Como podemos ver también, el jitter es bastante inestable, pese a mantener un valor concreto durante la mayor parte de la transmisión hay algunos picos

(máx +90 µs) y zonas donde el jitter varia constantemente (de la variación normal de varios µs a ±10 µs aproximadamente).

En recepción servgenmonII captura lo siguiente:

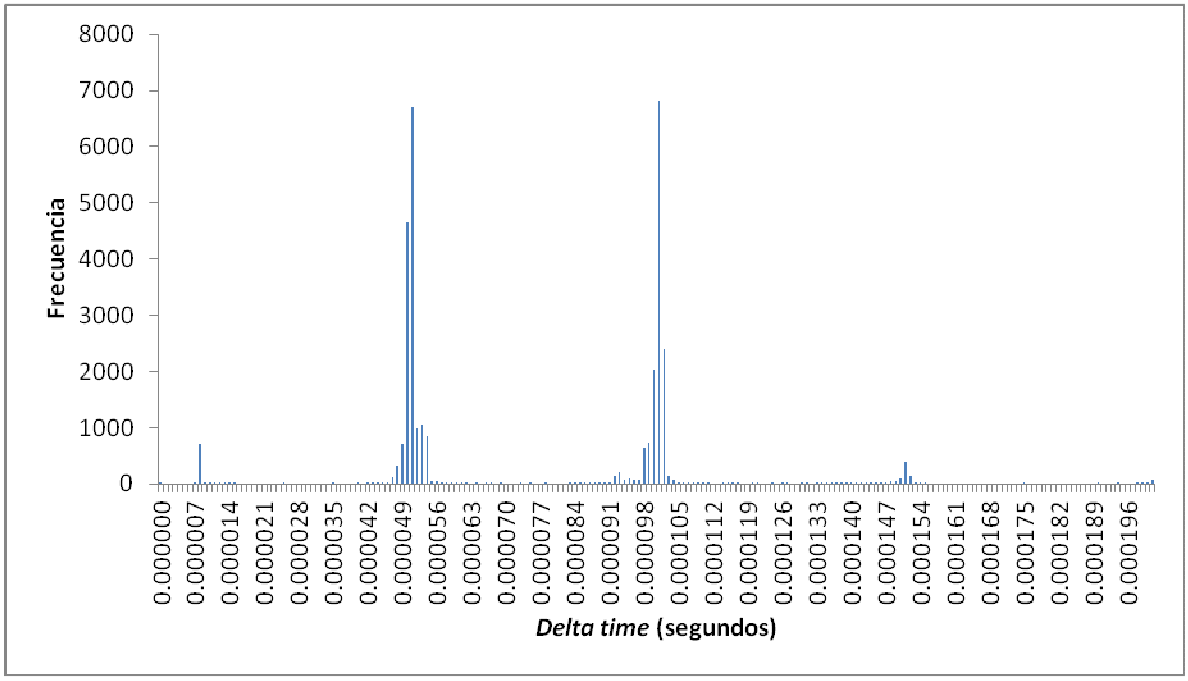

Fig. 4.19 Histograma

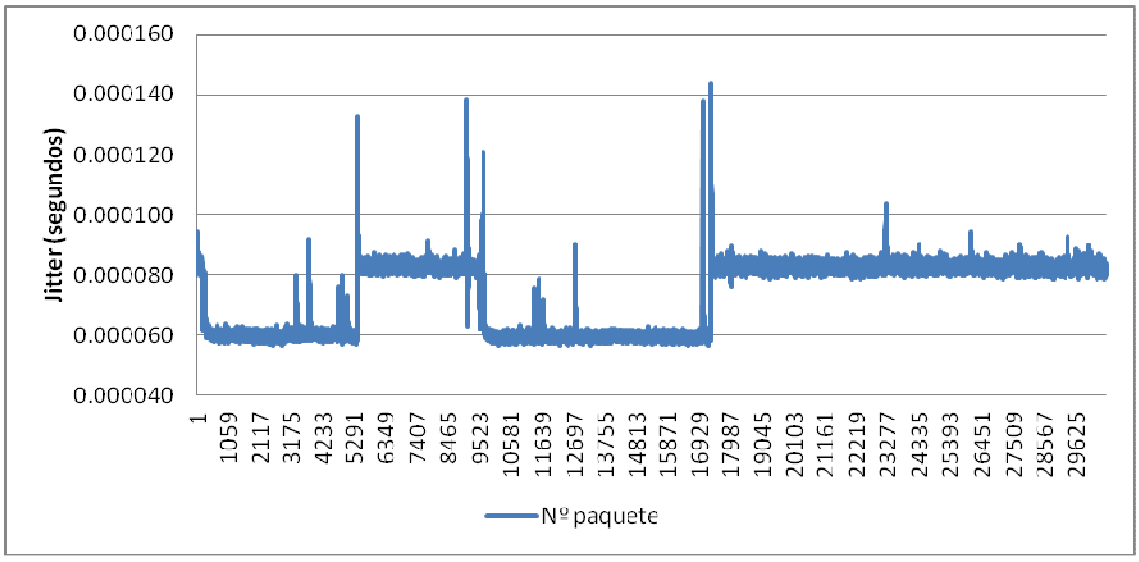

Fig. 4.20 Jitter acumulado

En estas gráficas podemos observar que el tiempo entre paquetes calculado según los *timestamps* ha variado respecto al emitido, ha pasado a estar dividido en varios “modos” que están comprendidos entre el tiempo entre paquetes máximo y mínimo detectado en emisión. Esto seguramente es debido a que pese a que RADclock mejora la precisón de el marcado, la tarjeta NIC, al capturar de la forma habitual (generando interrupciones que son atendidas

cuando la CPU esta libre) no envía los paquetes a la aplicación tal y como le llegan, y por lo tanto el marcado temporal no es del todo correcto.

Adicionalmente, podemos observar que los momentos de peor estabilidad del jitter en emisión se estabilizan ligeramente en recepción, pese a no tener el valor esperado, la variación durante esos instantes es menor. Esto puede ser debido a la mayor precisión del reloj en recepción. Estamos seguros de que el culpable de esto no es el switch, ya que algunas de las preubas realizadas se repitieron cambiando el switchs por el mismo utilizado para las pruebas con DAG (del cual estamos seguros que no introduce jitter; véase sección 4.3.1) y el resultado no varió en ningun caso.

4.4.2. Escenario 2

En este escenario hemos activado el *daemon* ntpd en servgenmonI utilizando el reloj RADclock de servgenmonII como servidor de referencia.

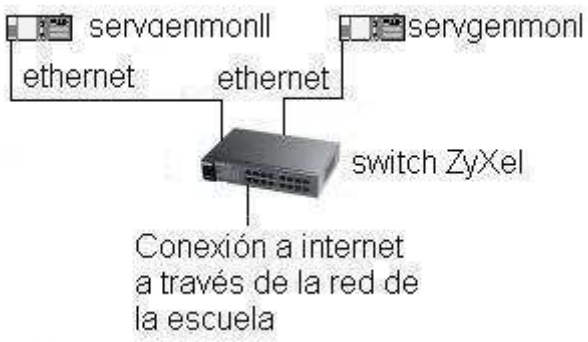

Fig. 4.21 Escenario 2 RADclock

Esto es lo que servgenmonI envía:

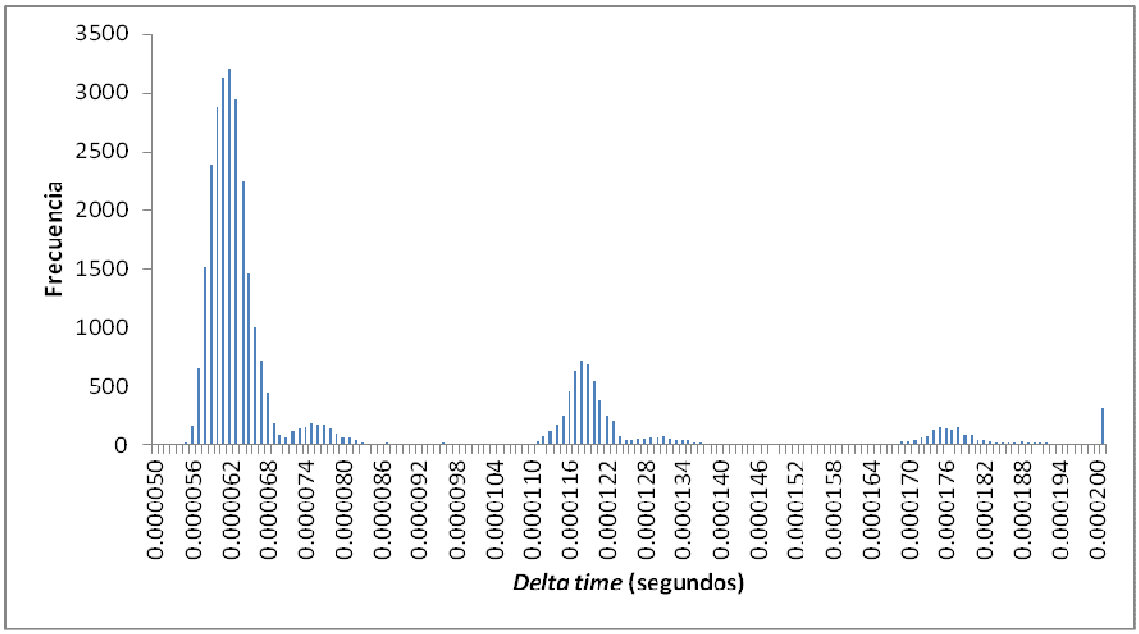

Fig. 4.22 Histograma

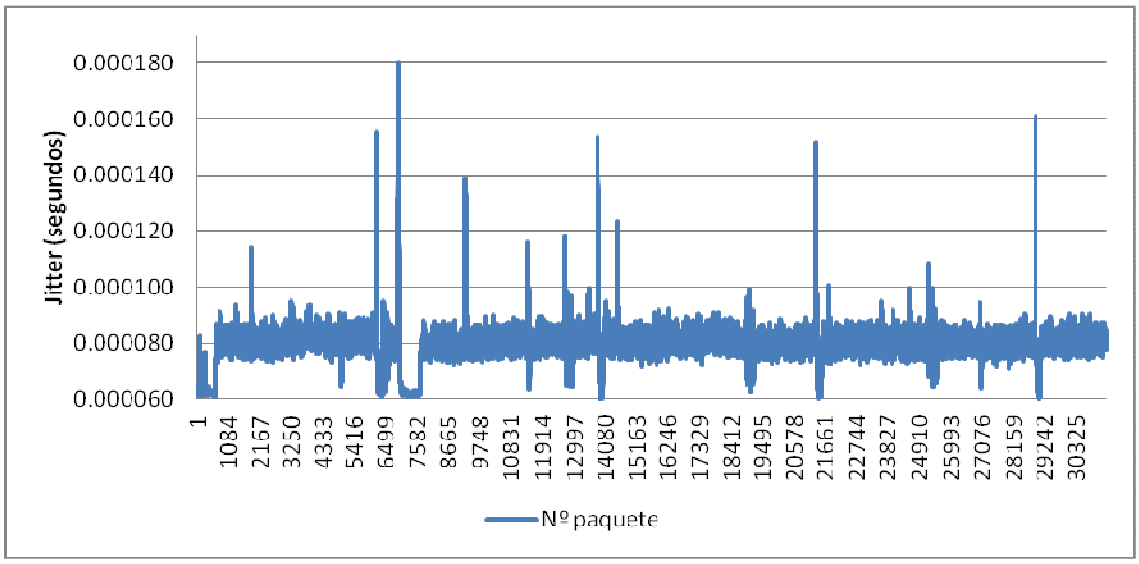

Fig. 4.23 Jitter acumulado

En este caso, la máquina tampoco ha sido capaz de enviar los paquetes con la periodicidad que esperábamos. De nuevo el *delta time* ha variado respecto al esperado (un paquete cada 24 µs) pasando a tener tres "modos" diferenciados (aproximadamente en 8, 50 y 100 µs) . Por otro lado el jitter se ha mantenido bastante estable a lo largo de toda la transmisión, con una variación constante de unos ±10 µs, con algunos picos de hasta 0.1 ms.

En este caso servgenmonII recibe:

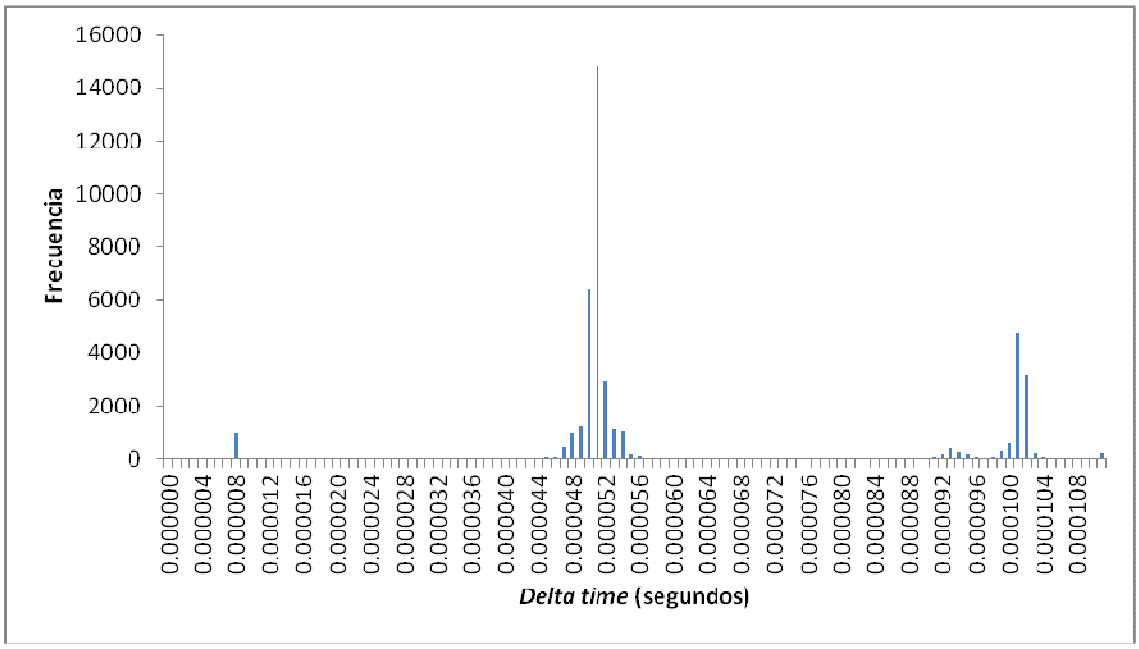

Fig. 4.24 Histograma

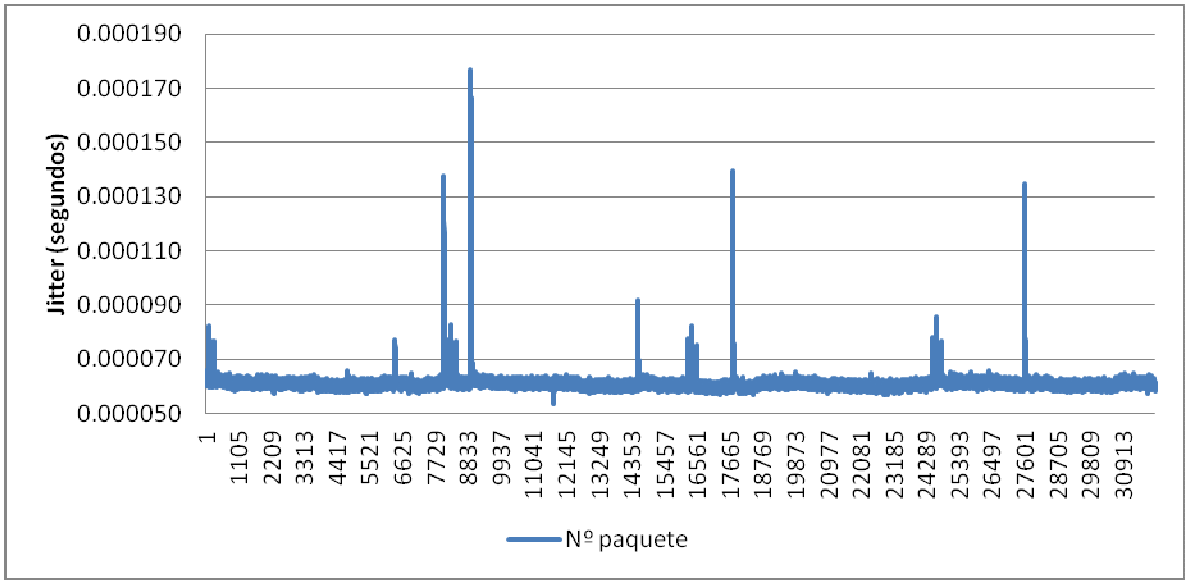

Fig. 4.25 Jitter acumulado

De nuevo, los delta time obtenidos en recepción se han esparcido en varios modos, pero en este caso coinciden con los detectados en emisión. Pese a ello podemos apreciar que la distrubución ha variado y por lo tanto, en realidad, la captura si se ha visto afectado por el método de captura utilizado. Por otra parte el jitter calculado ha sido más estable en recepción que en emisión, los picos de mayor tamaño se mantienen, pero la variación constante disminuye drásticamente (±4 µs aproximadamente)

4.4.3. Escenario 3

Finalmente, en este último escenario se ha puesto en marcha el reloj RADclock en radclockserv. El servidor temporal utilizado es reloj de *stratum1* de ntp.cesca.cat al que accedemos a través de internet.

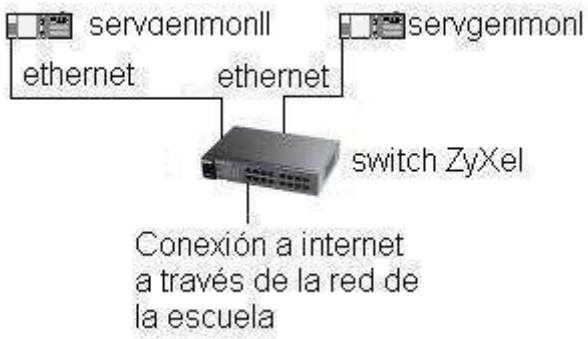

Fig. 4.26 Escenario 3 RADclock

En este escenario, los valores temporales capturados por la máquina que emite los datos dan como resultado estas gráficas:

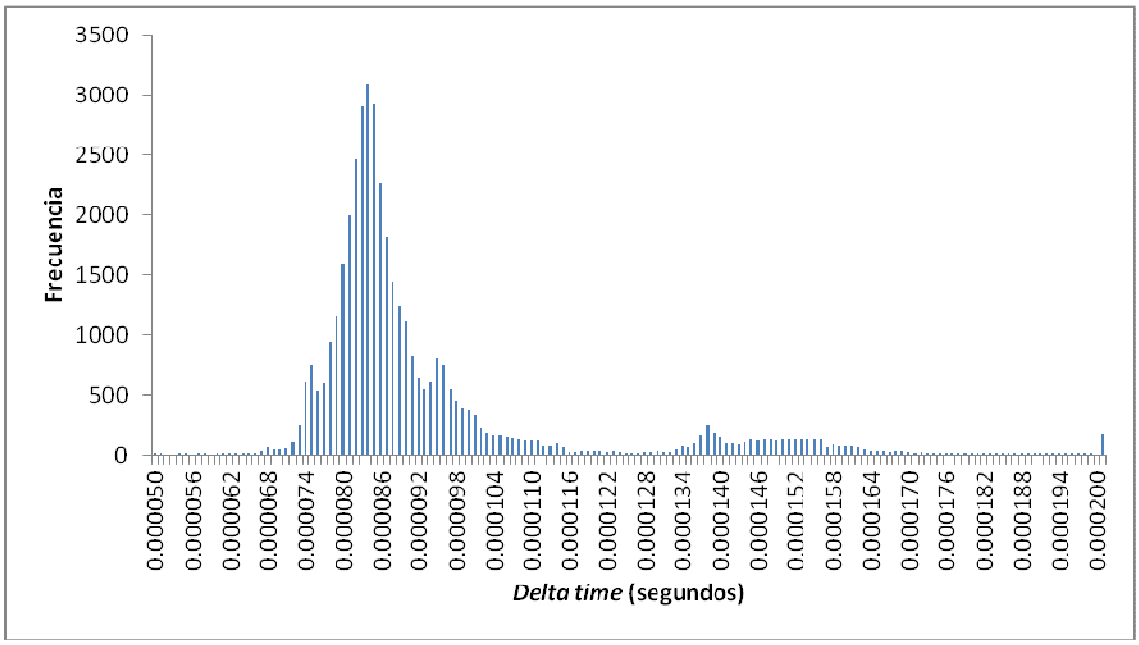

Fig. 4.27 Histograma

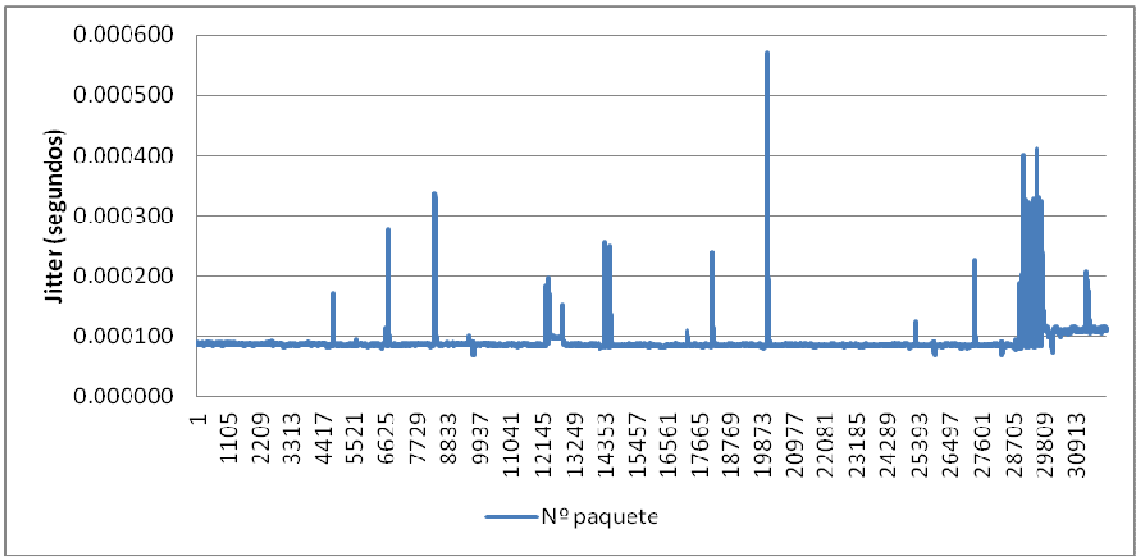

Fig. 4.28 Jitter acumulado

Una vez más el tiempo entre paquetes no ha sido el calculado, aun y así, ha sido bastante periódico con una minoría de paquetes que se envían con *delta time* superior al obtenido por la mayoría. El jitter también se ha mantenido muy estable a lo largo de la transmisión, con muy pocos picos y una sola variación grave al final de la transmisión.

Esta vez servgenmonII recibe:

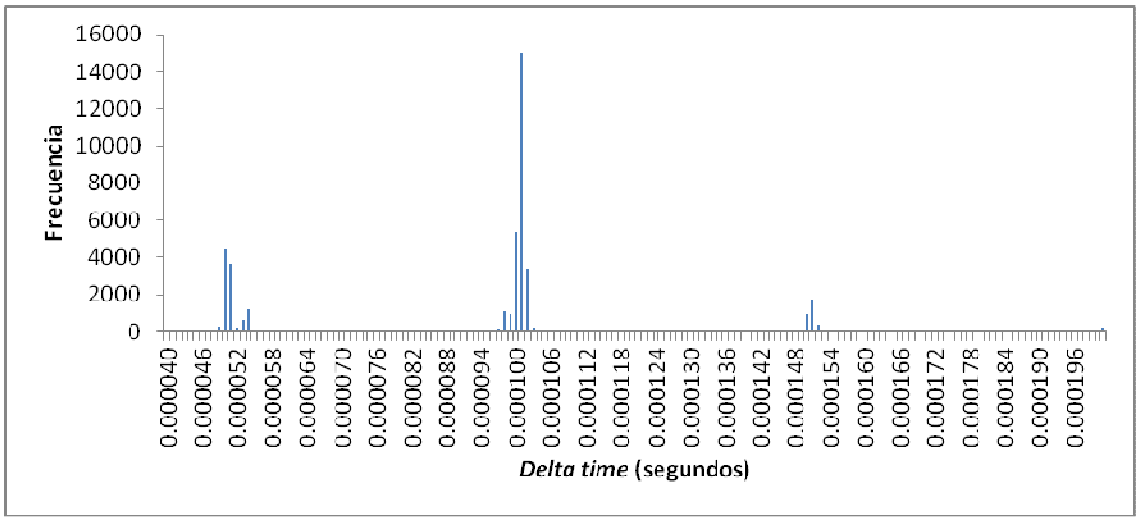

Fig. 4.29 Histograma

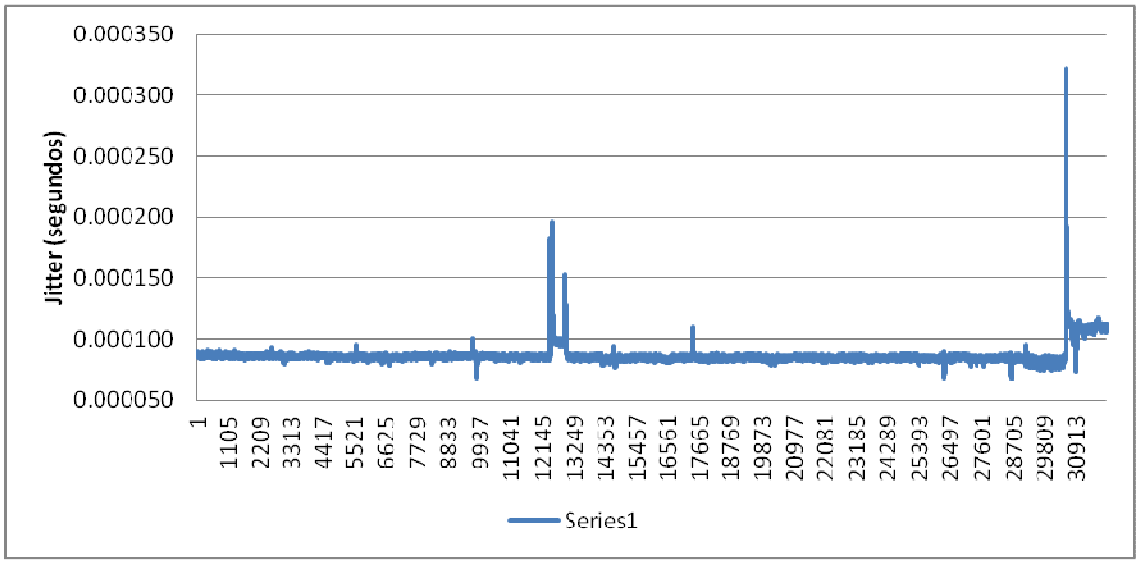

Fig. 4.30 Jitter acumulado

De nuevo, el tiempo entre paquetes se ha visto afectado por el metodo de captura empleado.. Por otra parte el jitter ha sido aun más estable que en emisión, han desparecido los picos puntuales y solo se han mantenido las variaciones que han afectado a varios paquetes consecutivos.

4.5. Otros resultados a destacar

En este apartado comentaremos otros aspectos de los resultados obtenidos en las pruebas realizadas. Aquí comentaremos como afectó el cambio en el tamaño del paquete y el aumento de la ocupación del enlace en algunos de los resultados obtenidos.

En cuanto al tamaño de paquete se refiere, podemos decir que no ha afectado demasiado a las pruebas realizadas. La herramienta utilizada para generar el tráfico (MGEN) no ha sido todo lo precisa que esperábamos, e independientemente del tamaño del paquete ha tenido problemas para mantener la periodicidad. Además cuanto mayor era la velocidad a la que transmitíamos los paquetes peor era la respuesta de MGEN:

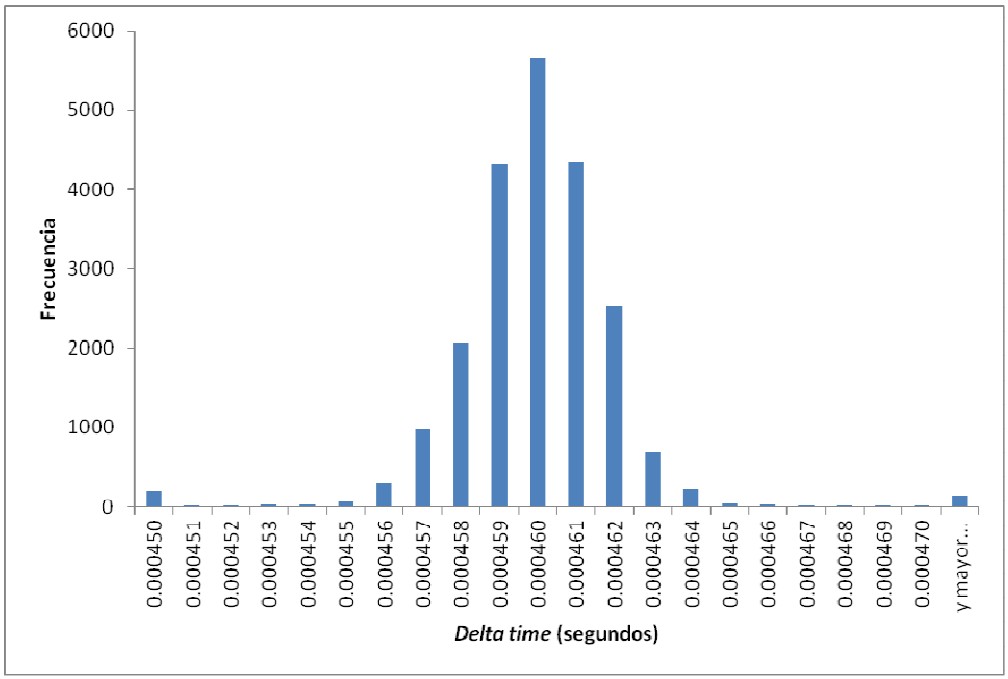

Fig. 4.31 radclockserv enviando tráfico de 576 bytes a 10 Mbps

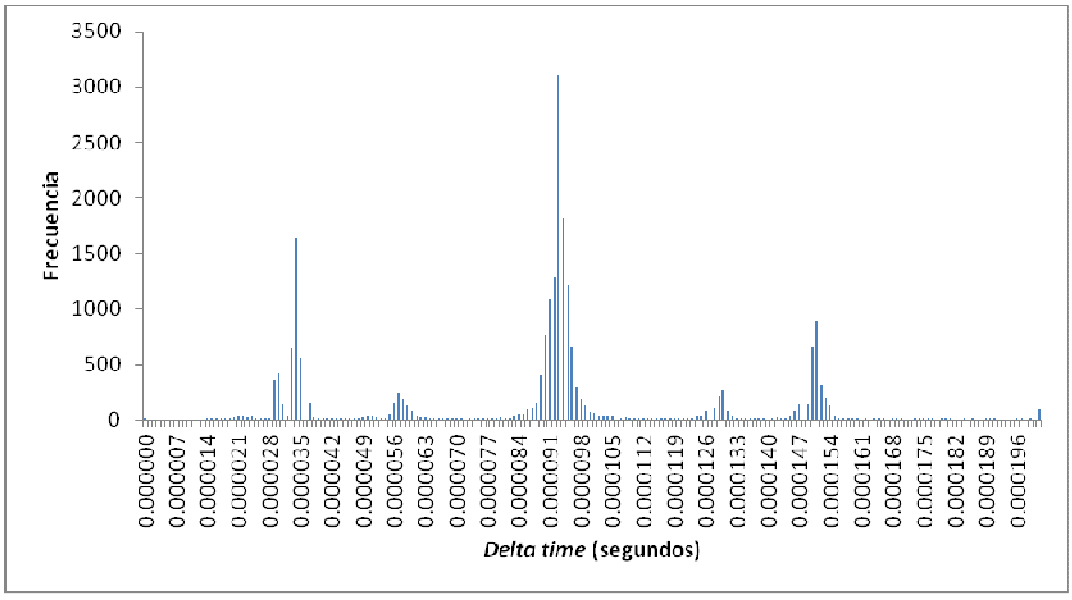

Fig. 4.32 radclockserv enviando tráfico de 576 bytes a 50 Mbps

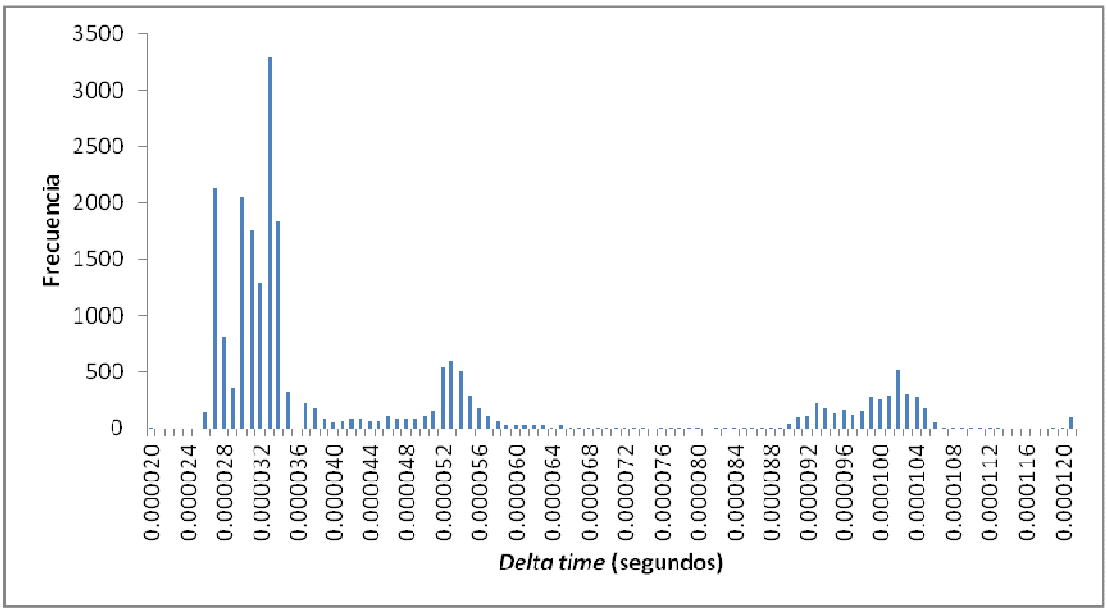

Fig. 4.33 servgenmonII enviando tráfico de 576 bytes a 100 Mbps

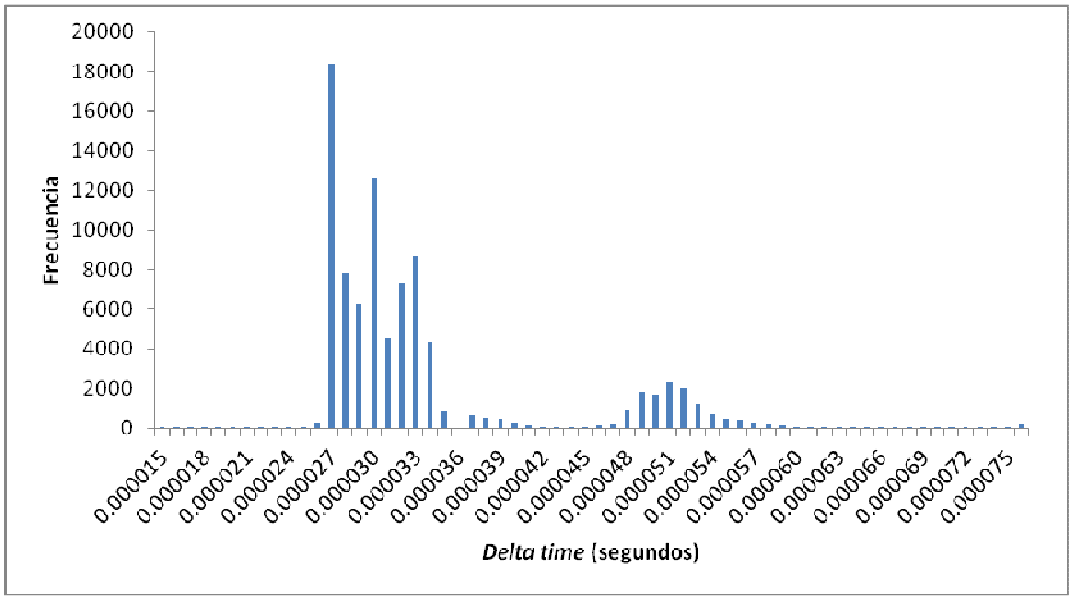

Fig. 4.34 servgenmonII enviando tráfico de 576 bytes a 500 Mbps

Como podemos observar en estas 4 gráficas, cuanto mayor es la velocidad peor es la periodicidad conseguida con MGEN.

A parte de eso, la ocupación del canal tampoco ha afectado mucho a las pruebas realizadas, no hemos encontrado ninguna clase de problema en captura (ni con DAG ni con RADclock) al aumentar la velocidad de transmisión.

Por otro lado, la utilización de RADclock en una máquina más moderna servgenmonI y servgenmonII) o en una máquina antigua (radclockserv) o no parece tener ninguna variación sobre su funcionamiento. En los dos casos hemos obtenido el mismo rendimiento.

CONCLUSIONES Y LÍNEAS FUTURAS

5.1. Objetivos alcanzados

Durante el desarrollo de este TFC hemos vuelto a poner en marcha las tarjetas DAG de Endace obteniendo así conocimientos sobre su puesta en marcha: qué *transceivers* necesita, SO y software necesarios para su instalación y como capturar tráfico. Una vez finalizada la instalación las hemos utilizado para realizar capturas de tráfico y así comprobar sus capacidades.

Otra tarea que se ha realizado es la preparación de una máquina para la instalación del software RADclock aprendiendo así a modificar y compilar un kernel concreto en un sistema operativo Linux. Una vez preparado el equipo hemos pasado a la instalación del propio RADclock y a su puesta en marcha. Durante este proceso hemos aprendido mucho sobre el funcionamiento de RADclock y hemos explorado algunas de las distintas posibilidades que nos ofrece.

Una vez preparadas las 2 soluciones propuestas, hemos llevado a cabo una serie de pruebas para comprobar las ventajas que nos ofrece cada una de ellas. Estas pruebas nos han permitido ver el comportamiento de las dos soluciones para capturar tráfico. Además también nos han servido para ver el comportamiento de una máquina con su reloj sincronizado mediante RADclock a la hora de generar tráfico periódico.

A lo largo de la realización de este estudio nos hemos enfrentado a diferentes dificultades. Por un lado encontramos problemas de compatibilidad de los drivers de los que disponíamos para la tarjeta con sistemas operativos modernos teniendo así que instalarlas en un SO más antiguo. Por otro lado se ha tenido que aprender a compilar correctamente un kernel en Linux, o cual era nuevo para el autor de este trabajo.

5.2. Valoración de los objetivos

A raíz de los resultados obtenidos en las pruebas podemos llegar a varias conclusiones que en parte han sido satisfactorias.

Por un lado hemos visto que el reloj controlado por software RADclock mejora las capturas realizadas con NIC convencionales, pero no ofrece una solución mejor que la DAG a la hora de realizar capturas de tráfico. Pese a mejorar la precisión del reloj, el hecho de capturar a través de tarjetas NIC normales y corrientes, generando interrupciones en el SO y realizando el marcado temporal a nivel de aplicación hace que la precisión de las capturas no sea la adecuada para la monitorización de tráfico, ya que el tiempo de llegada entre paquetes se ha visto afectado por el paso por el SO (interrupciones) lo cual puede perjudicar a transmisiones en tiempo real.

En cambio, las tarjetas DAG han demostrado ser una muy buena solución para la monitorización de tráfico. Debido a su modo de funcionamiento (prácticamente ajeno al resto de la máquina donde están funcionando), el marcado temporal de los paquetes se realiza de forma precisa, sin afectar a las características del tráfico que capturamos. Por ello, podemos llegar a la conclusión que para llevar a cabo una monitorización precisa es mejor utilizar esta solución.

Por otra parte hemos visto que RADclock nos ofrece una serie de características que pueden ser explotadas. Hemos podido comprobar que el reloj de una máquina con este sistema instalado gana en estabilidad. A lo largo de las pruebas realizadas se ha comprobado como la utilización de RADclock en la máquina que generaba el tráfico mejoraba drásticamente el jitter obtenido (más que con el *daemon* ntpd). Debido a esto podemos afirmar que, pese a que RADclock no ofrece una buena solución para la monitorización, puede ofrecernos un reloj estable que nos sea útil para otros ámbitos.

5.3. Líneas futuras

A partir de lo desarrollado en este TFC se abren varias líneas de investigación. Por un lado las tarjetas DAG vuelven a estar en funcionamiento, lo cual nos permite seguir indagando en sus capacidades. Siguiendo la línea de TFC anteriores [1], se puede desarrollar el software de emisión y captura de tráfico creado en ellos, así como implementar nuevos programas.

Por otro lado el software RADclock podría implementarse en sistemas donde se requiera transmitir datos con un jitter estable, como sistemas audiovisuales, o comunicaciones industriales para servicios en tiempo real (por ejemplo radares aeronáuticos). Además la API de RADclock nos ofrece la posibilidad de utilizar las funciones de este reloj en programas implementados por nosotros mismos, lo cual nos permitiría crear programas propios que utilicen un reloj muy estable.

También se puede investigar cómo conseguir que la precisión del reloj fuese útil a la hora de monitorizar tráfico con tarjetas NIC, desarrollando software capaz de aprovechar la precisión de RADclock y que a su vez disminuya el impacto que el paso por el kernel supone para la marcación temporal. Para ello las principales opciones a investigar serían las que se mencionan a continuación:

En primer lugar, tendríamos que tener presentes las librerías libpcap (que permite capturar paquetes en formato pcap) y time.h (para acceder al TSC de la máquina). Después tendríamos que ver que funciones podemos aprovechar de la librería libradclock (que es la propia de RADclock y nos permite utilizar el reloj de alta precisión que este implementa). Por último, sería interesante estudiar el posible uso de las funciones de af_packet.c [14], con el que podemos conseguir una mejora en el paso por el SO.

5.4. Cuestiones éticas y medioambientales

Los resultados obtenidos en este TFC no presentan en sí mismos una amenaza para el medioambiente. Por otra parte las máquinas necesarias para llevar a cabo una correcta monitorización de tráfico suponen un consumo energético, teniendo en cuenta que en el proceso pueden intervenir múltiples dispositivos (PCs, *switchs*, *routers*, pantallas…) que, además, suelen estar ubicados en salas que necesitan sistemas de refrigeración este consumo puede ser muy elevado. . Por lo tanto habría que tomar medidas de obtención de energía limpias como puede ser la solar para alimentar los dispositivos menos potentes y sistemas de ahorro de energía para los que más consumen. Además habría que regular las emisiones de los sistemas de refrigeración mediante catalizadores u otros dispositivos reguladores.

Por otro lado existen soluciones con las que disminuir el consumo energético de los equipos de comunicaciones estudiados en otros TFC's. Como podemos ver en [15] estos sistemas presentan un modo de funcionamiento en el que los enlaces solo están activos durante e tiempo de transmisiónestrictamente necesario. Por ejemplo *Energy-Efficient Ethernet 802.3az* utiliza un sistema mediante el cual, mientras un enlace no se utiliza este pasa a un estado "dormido". En este estado, al no transmitir ni recibir nada, perdemos la sincronización de los relojes de transmisión[11], y al volver a levantar ese enlace para enviar datos, antes de comenzar el envió tiene que enviar una serie de peticiones para indicar que el enlace vuelve a estar activo y volver a sincronizarlo. Esto supone un problema, ya que el tiempo en el que se tarda en restablecer un enlace "dormido" acarrea un tiempo adicional que puede afectar al jitter ya que el paquete tiene que esperar a que el puerto se active antes de ser transmitido. Si inmediatamente después otro paquete está preparado para ser transmitido, se encadenan las transmisiones, por lo que no todos los paquetes sufren el mismo jitter Por lo tanto sería interesante evaluar estas soluciones mediante un estudio que cuantifique en qué medida nos puede afectar esa variación del jitter.

En cuanto a las cuestiones éticas, siempre que hablamos de monitorización de tráfico nos toparemos con tramas pertenecientes a ISP. En este aspecto siempre hemos de procurar proteger la intimidad de los propietarios de los datos, procurando que los datos en circulen de forma protegida. Para este propósito se pueden utilizar herramientas de anonimización de trazas o de "sanitización" [16] mediante las cuales se sustituyan los datos privados (direcciones IP, *payload* de paquetes SMTP, etc.), por otros valores que sigan manteniendo la semántica del tráfico pero oculten los datos comprometidos.

[11] Gigabit Ethernet transmite datos de relleno cuando no hay actividad útil, para mantener sincronizados los relojes de muestreo de la seña en emisor y receptor.

GLOSARIO

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
CPU	Central Processing Unit
CRC	Comprobación de Redundancia Cíclica
DAG	Data Acquisition and Generation
DUCK	DAG universal Clock Kit
GPS	Global Positioning System
IP	Internet Protocol
ISP	Internet Service Provider
NIC	Network Interface Card
NTP	Network Time Protocol
QoS	Quality of service
RADclock	Robust Absolute and Difference Clock
RAM	Random access memory
RTCP	Real time control protocol)
RTT	Round-trip delay time
SLA	Service Level Agreement
SMTP	Simple Mail Transfer Protocol
SO	Sistema Operativo
TSC	Time System Clock
UDP	User Datagram Protocol
UTC	Universal Time Coordinated

BIBLIOGRAFIA

[1] Pérez, I. (Julio 2006), "Analizador/Generador Gigabit Ethernet de altas prestaciones". Director: D. Rincón Rivera. Trabajo Fin de Carrera, Departamento de Ingeniería Telemática, Escola Politècnica Superior de Castelldefels.

[2] "RADclock", [En línea], http://www.synclab.org/radclock/ [Última consulta el 27 de septiembre de 2011]

[3] "NTP" [En línea], http://es.wikipedia.org/wiki/Network_Time_Protocol/ [Ultima consulta el 22 de abril de 2012]

[4] D.L. Mills (Septiembre 1985), RFC956, "Algorithms for Synchronizing Network Clocks**",** http://tools.ietf.org/html/rfc956

[5] D.L. Mills (Septiembre 1985), RFC958, "Network Time Protocol (NTP)", http://tools.ietf.org/html/rfc958

[6] D.L. Mills (Marzo 1992), RFC1305, "Network Time Protocol (Version 3) Specification, Implementation and Analysis", http://tools.ietf.org/html/rfc1305

[7] "Wireshark", [En línea] http://www.wireshark.org/ [Ultima consulta noviembre 2011]

[8] H. Schulzrinne (Julio 2003), RFC3550 "RTP: A Transport Protocol for Real-Time Applications", http://tools.ietf.org/html/rfc3550

[9] "MGEN", [en línea] http://cs.itd.nrl.navy.mil/work/mgen/ [Ultima consulta noviembre 2011]

[10] "dumpcap", [en línea] http://www.wireshark.org/docs/man-pages/dumpcap.html

[11] "Algoritmo de Nagle En línea], http://es.wikipedia.org/wiki/Algoritmo_de_Nagle [Ultima consulta el 22 de abril de 2012]

[12] J. Postel (Noviembre 1983), RFC879 "The TCP Maximum Segment Size and Related Topics", http://tools.ietf.org/html/rfc879

[13] J. Mogul (Noviembre 1990), RFC1191 "Path MTU Discovery", http://tools.ietf.org/html/rfc1191

[14] Ross Biro, af_packet.c, [En línea] http://lxr.linux.no/#linux+v3.3.3/net/packet/af_packet.c [Ultima consulta julio 2011]

[15] Ezponda, J. (Septiembre 2011), “Estudi i simulacions de l’estàndard Energy-Efficient Ethernet”. Director: D. Rincón Rivera. Trabajo Fin de Carrera, Departamento de Ingeniería Telemática, Escola Politècnica Superior de Castelldefels.

[16] Bishop, M; Bhumiratana, B; Crawford, R; y Levitt, K. (Junio 2004) *“How to sanitize data?”* 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2004. WET ICE 2004, pp 217- 222

ANEXOS

Anexo 2.1 log al ejecutar ./config

Durante la instalación de los drivers de la tarjeta DAG este es el log que obtenemos al ejecutar correctamente el comando `./config`

```
checking for flex... flex
checking for flex 2.4 or higher... yes
checking for bison... bison
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking for flex... flex
checking for yywrap in -lfl... yes
checking lex output file root... lex.yy
checking whether yytext is a pointer... yes
checking for bison... bison -y
checking for ldconfig... /sbin/ldconfig
checking how to run the C preprocessor... gcc -E
checking for egrep... grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for long... yes
checking size of long... 4
checking which drivers, tools and libraries to build ... ok
checking byteswap.h usability... yes
checking byteswap.h presence... yes
checking for byteswap.h... yes
checking readline/readline.h usability... yes
checking readline/readline.h presence... yes
checking for readline/readline.h... yes
checking for initscr in -lreadline... yes
checking for initscr in -lncurses... yes
checking for kernel source ... /usr/src/linux
checking for kernel config ... /usr/src/linux/.config
checking whether remap_pfn_range() is available... no
```

```
checking whether remap_page_range() takes 5 parameters... no
checking for kgcc... gcc
checking for gcc-2.95... gcc
checking for a BSD-compatible install... /usr/bin/install -c
configure: creating ./config.status
config.status: creating Makefile
config.status: creating arm/Makefile
config.status: creating xilinx/Makefile
config.status: creating doc/Makefile
config.status: creating include/Makefile
config.status: creating lib/Makefile
config.status: creating lib/dag37t_api/Makefile
config.status: creating drv/divdi3/Makefile
config.status: creating drv/linux/Makefile
config.status: creating tools/Makefile
config.status: creating filtering/Makefile
config.status: creating tools/dagconvert/Makefile
config.status: creating tools/dagfwddemo/Makefile
config.status: creating tools/daggen/Makefile
config.status: creating filtering/filter_loader/Makefile
config.status: creating filtering/snort_compiler/Makefile
config.status: creating filtering/tcpdump_compiler/Makefile
config.status: creating include/config.h
config.status: include/config.h is unchanged
```

Anexo 2.2 log al ejecutar make depend

Al hacer el comando `make depend` para instalar los drivers de la DAG obtenemos este resultado:

```
make[1]: Entering directory `/usr/local/dag-2.5.5/arm'
make[1]: Nothing to be done for `depend'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/arm'
make[1]: Entering directory `/usr/local/dag-2.5.5/xilinx'
make[1]: Nothing to be done for `depend'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/xilinx'
make[1]: Entering directory `/usr/local/dag-2.5.5/doc'
make[1]: Nothing to be done for `depend'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/doc'
make[1]: Entering directory `/usr/local/dag-2.5.5/include'
make[1]: Nothing to be done for `depend'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/include'
make[1]: Entering directory `/usr/local/dag-2.5.5/lib'
flex -Pdagopt  -t dagopts.l > dagopts.c
Creating file dagversion.c
sed -e 's/.*/const char* const kDagReleaseVersion = "&";/'
../VERSION > ./dagversion.c
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-2.5.5/lib'
make[1]: Entering directory `/usr/local/dag-
2.5.5/lib/dag37t_api'
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-2.5.5/lib/dag37t_api'
make[1]: Entering directory `/usr/local/dag-2.5.5/drv/divdi3'
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-2.5.5/drv/divdi3'
make[1]: Entering directory `/usr/local/dag-2.5.5/drv/linux'
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-2.5.5/drv/linux'
make[1]: Entering directory `/usr/local/dag-2.5.5/tools'
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-2.5.5/tools'
make[1]: Entering directory `/usr/local/dag-2.5.5/filtering'
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-2.5.5/filtering'
make[1]: Entering directory `/usr/local/dag-
2.5.5/tools/dagconvert'
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-
2.5.5/tools/dagconvert'
make[1]: Entering directory `/usr/local/dag-
2.5.5/tools/dagfwddemo'
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-
2.5.5/tools/dagfwddemo'
make[1]: Entering directory `/usr/local/dag-2.5.5/tools/daggen'
bison -y -d -v -o gram.c gram.y
flex  -t lex.l > lex.c
rebuilding .depend ... done
```

```
make[1]: Leaving directory `/usr/local/dag-2.5.5/tools/daggen'
make[1]: Entering directory `/usr/local/dag-
2.5.5/filtering/filter_loader'
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-
2.5.5/filtering/filter_loader'
make[1]: Entering directory `/usr/local/dag-
2.5.5/filtering/snort_compiler'
bison -y -d -v -o src/parser.c src/parser.y
flex  -t src/scanner.l > src/scanner.c
rebuilding .depend ... done
make[1]: Leaving directory `/usr/local/dag-
2.5.5/filtering/snort_compiler'
make[1]: Entering directory `/usr/local/dag-
2.5.5/filtering/tcpdump_compiler'
flex  -t src/scanner.l > src/scanner.c
bison -y -d -v -o src/parser.c src/parser.y
rebuilding .depend ... done
make[1]:        Leaving        directory        `/usr/local/dag-
2.5.5/filtering/tcpdump_compiler'
```

Anexo 2.3 log al ejecutar make

Este es el resultado de ejecutar con éxito el comando `make` en la instalación de los drivers de la DAG.

```
make[1]: Entering directory `/usr/local/dag-2.5.5/arm'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/arm'
make[1]: Entering directory `/usr/local/dag-2.5.5/xilinx'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/xilinx'
make[1]: Entering directory `/usr/local/dag-2.5.5/doc'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/doc'
make[1]: Entering directory `/usr/local/dag-2.5.5/include'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/include'
make[1]: Entering directory `/usr/local/dag-2.5.5/lib'
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagapi.o dagapi.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagclarg.o dagclarg.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagcrc.o dagcrc.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagimg.o dagimg.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagload.o dagload.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagname.o dagname.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagpci.o dagpci.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dag_platform_freebsd.o dag_platform_freebsd.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dag_platform_linux.o dag_platform_linux.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dag_platform_macosx.o dag_platform_macosx.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dag_platform_solaris.o dag_platform_solaris.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dag_platform_win32.o dag_platform_win32.c
```

```
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagreg.o dagreg.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dag_romutil.o dag_romutil.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagutil.o dagutil.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagopts.o dagopts.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../include -fPIC   -c -o
dagversion.o dagversion.c
Creating standard library libdag.a
rm -f libdag.a
ar -cq libdag.a dagapi.o dagclarg.o dagcrc.o dagimg.o dagload.o
dagname.o dagpci.o dag_platform_freebsd.o dag_platform_linux.o
dag_platform_macosx.o dag_platform_solaris.o
dag_platform_win32.o dagreg.o dag_romutil.o dagutil.o dagopts.o
dagversion.o
ranlib libdag.a
Creating shared library libdag.so.1.7
rm -f libdag.so.1.7
cc -shared -fPIC -Wl,-soname,libdag.so.1 -o libdag.so.1.7
dagapi.o dagclarg.o dagcrc.o dagimg.o dagload.o dagname.o
dagpci.o dag_platform_freebsd.o dag_platform_linux.o
dag_platform_macosx.o dag_platform_solaris.o
dag_platform_win32.o dagreg.o dag_romutil.o dagutil.o dagopts.o
dagversion.o
rm -f libdag.so.1
rm -f libdag.so
ln -s libdag.so.1.7 libdag.so.1
ln -s libdag.so.1.7 libdag.so
make[1]: Leaving directory `/usr/local/dag-2.5.5/lib'
make[1]: Entering directory `/usr/local/dag-
2.5.5/lib/dag37t_api'
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o cfgwcmd.o cfgwcmd.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o configmgr.o configmgr.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o conncmd.o conncmd.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o d37t_host_i2c.o d37t_host_i2c.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o d37t_host_msg.o d37t_host_msg.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o dag37t_api.o dag37t_api.c
```

```
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o dagchanio.o dagchanio.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o dag_ima_config_api.o dag_ima_config_api.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o dagmapper.o dagmapper.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o iomgr.o iomgr.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o maskcmd.o maskcmd.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o moduleio.o moduleio.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -I../../include -fPIC   -c
-o ../dagswid.o ../dagswid.c
Creating standard library libdag37t.a
rm -f libdag37t.a
ar -cq libdag37t.a cfgwcmd.o configmgr.o conncmd.o
d37t_host_i2c.o d37t_host_msg.o dag37t_api.o dagchanio.o
dag_ima_config_api.o dagmapper.o iomgr.o maskcmd.o moduleio.o
../dagswid.o
ranlib libdag37t.a
Creating shared library libdag37t.so.1.0
rm -f libdag37t.so.1.0
cc -shared -fPic -Wl,-soname,libdag37t.so.1 -o libdag37t.so.1.0
cfgwcmd.o configmgr.o conncmd.o d37t_host_i2c.o d37t_host_msg.o
dag37t_api.o dagchanio.o dag_ima_config_api.o dagmapper.o
iomgr.o maskcmd.o moduleio.o ../dagswid.o
rm -f libdag37t.so.1
rm -f libdag37t.so
ln -s libdag37t.so.1.0 libdag37t.so.1
ln -s libdag37t.so.1.0 libdag37t.so
make[1]: Leaving directory `/usr/local/dag-2.5.5/lib/dag37t_api'
make[1]: Entering directory `/usr/local/dag-2.5.5/drv/divdi3'
gcc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-I/usr/src/linux/include/asm-i386/mach-default -
I/usr/src/linux/include -D__KERNEL__ -DMODULE -fomit-frame-
pointer -DDIAGNOSTIC   -DMODVERSIONS -include
/usr/src/linux/include/linux/modversions.h   -c -o divdi3_exp.o
divdi3_exp.c
gcc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-I/usr/src/linux/include/asm-i386/mach-default -
I/usr/src/linux/include -D__KERNEL__ -DMODULE -fomit-frame-
pointer -DDIAGNOSTIC   -DMODVERSIONS -include
/usr/src/linux/include/linux/modversions.h   -c -o divdi3_impl.o
divdi3_impl.c
ld -r divdi3_exp.o divdi3_impl.o -o divdi3
make[1]: Leaving directory `/usr/local/dag-2.5.5/drv/divdi3'
make[1]: Entering directory `/usr/local/dag-2.5.5/drv/linux'
```

```
gcc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-I/usr/src/linux/include/asm-i386/mach-default -
I/usr/src/linux/include -I../../include -D__KERNEL__ -DMODULE -
fomit-frame-pointer -DDIAGNOSTIC   -DMODVERSIONS -include
/usr/src/linux/include/linux/modversions.h   -c -o dag_main.o
dag_main.c
gcc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-I/usr/src/linux/include/asm-i386/mach-default -
I/usr/src/linux/include -I../../include -D__KERNEL__ -DMODULE -
fomit-frame-pointer -DDIAGNOSTIC   -DMODVERSIONS -include
/usr/src/linux/include/linux/modversions.h   -c -o dagpci.o
../../lib/dagpci.c
gcc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-I/usr/src/linux/include/asm-i386/mach-default -
I/usr/src/linux/include -I../../include -D__KERNEL__ -DMODULE -
fomit-frame-pointer -DDIAGNOSTIC   -DMODVERSIONS -include
/usr/src/linux/include/linux/modversions.h   -c -o dagduck.o
../common/dagduck.c
gcc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-I/usr/src/linux/include/asm-i386/mach-default -
I/usr/src/linux/include -I../../include -D__KERNEL__ -DMODULE -
fomit-frame-pointer -DDIAGNOSTIC   -DMODVERSIONS -include
/usr/src/linux/include/linux/modversions.h   -c -o dagmon.o
../common/dagmon.c
gcc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-I/usr/src/linux/include/asm-i386/mach-default -
I/usr/src/linux/include -I../../include -D__KERNEL__ -DMODULE -
fomit-frame-pointer -DDIAGNOSTIC   -DMODVERSIONS -include
/usr/src/linux/include/linux/modversions.h   -c -o dagreg.o
../../lib/dagreg.c
gcc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-I/usr/src/linux/include/asm-i386/mach-default -
I/usr/src/linux/include -I../../include -D__KERNEL__ -DMODULE -
fomit-frame-pointer -DDIAGNOSTIC   -DMODVERSIONS -include
/usr/src/linux/include/linux/modversions.h   -c -o dagversion.o
../../lib/dagversion.c
ld -r dag_main.o dagpci.o dagduck.o dagmon.o dagreg.o
dagversion.o -o dag
gcc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-I/usr/src/linux/include/asm-i386/mach-default -
I/usr/src/linux/include -I../../include -D__KERNEL__ -DMODULE -
fomit-frame-pointer -DDIAGNOSTIC   -DMODVERSIONS -include
/usr/src/linux/include/linux/modversions.h   -c -o dagmem_main.o
dagmem_main.c
ld -r dagmem_main.o dagpci.o -o dagmem
make[1]: Leaving directory `/usr/local/dag-2.5.5/drv/linux'
make[1]: Entering directory `/usr/local/dag-2.5.5/tools'
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagreset.o dagreset.c
cc   dagreset.o  -lreadline -lncurses -L../lib -ldag -o dagreset
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o daginf.o daginf.c
cc   daginf.o  -lreadline -lncurses -L../lib -ldag -o daginf
```

```
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagld.o dagld.c
cc   dagld.o  -lreadline -lncurses -L../lib -ldag -o dagld
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP -c -o dagrom.o dagrom.c
cc dagrom.o -lreadline -lncurses -L../lib -ldag -
L../lib/dag37t_api -ldag37t -lpthread -o dagrom
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagsnap.o dagsnap.c
cc   dagsnap.o  -lreadline -lncurses -L../lib -ldag -o dagsnap
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagbits.o dagbits.c
cc   dagbits.o  -lreadline -lncurses -L../lib -ldag -o dagbits
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagthree.o dagthree.c
cc   dagthree.o  -lreadline -lncurses -L../lib -ldag -o dagthree
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagfour.o dagfour.c
cc   dagfour.o  -lreadline -lncurses -L../lib -ldag -o dagfour
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagsix.o dagsix.c
cc   dagsix.o  -lreadline -lncurses -L../lib -ldag -o dagsix
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagclock.o dagclock.c
cc   dagclock.o  -lreadline -lncurses -L../lib -ldag -o dagclock
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagsar.o dagsar.c
cc   dagsar.o  -lreadline -lncurses -L../lib -ldag -o dagsar
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagvcam.o dagvcam.c
cc   dagvcam.o  -lreadline -lncurses -L../lib -ldag -o dagvcam
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP -c -o dagchan.o dagchan.c
cc dagchan.o -lreadline -lncurses -L../lib -ldag -
L../lib/dag37t_api -ldag37t -lpthread -o dagchan
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagflood.o dagflood.c
cc   dagflood.o  -lreadline -lncurses -L../lib -ldag -o dagflood
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP -c -o dagmap.o dagmap.c
cc dagmap.o -lreadline -lncurses -L../lib -ldag -
L../lib/dag37t_api -ldag37t -lpthread -o dagmap
```

```
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagpps.o dagpps.c
cc   dagpps.o  -lreadline -lncurses -L../lib -ldag -o dagpps
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP -c -o dagwatchdog.o dagwatchdog.c
cc dagwatchdog.o -lreadline -lncurses -L../lib -ldag -lpthread -
o dagwatchdog
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP -c -o dagswid.o dagswid.c
cc dagswid.o -lreadline -lncurses -L../lib -ldag -
L../lib/dag37t_api -ldag37t -lpthread -o dagswid
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP -c -o dagimademo.o dagimademo.c
cc dagimademo.o -lreadline -lncurses -L../lib -ldag -
L../lib/dag37t_api -ldag37t -lpthread -o dagimademo
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../include -DBYTESWAP   -c -o dagbug.o dagbug.c
cc   dagbug.o  -lreadline -lncurses -L../lib -ldag -o dagbug
make[1]: Leaving directory `/usr/local/dag-2.5.5/tools'
make[1]: Entering directory `/usr/local/dag-2.5.5/filtering'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/filtering'
make[1]: Entering directory `/usr/local/dag-
2.5.5/tools/dagconvert'
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
dagconvert.o dagconvert.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
dagio.o dagio.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
erfio.o erfio.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
legacyio.o legacyio.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
nullio.o nullio.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
pcapio.o pcapio.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
prtio.o prtio.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
utils.o utils.c
cc   dagconvert.o dagio.o erfio.o legacyio.o nullio.o pcapio.o
prtio.o utils.o  -L../../lib -ldag  -lpcap -o dagconvert
```

```
make[1]: Leaving directory `/usr/local/dag-
2.5.5/tools/dagconvert'
make[1]: Entering directory `/usr/local/dag-
2.5.5/tools/dagfwddemo'
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
dagfwddemo.o dagfwddemo.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE64_SOURCE -I../../include    -c -o
inline_filter.o inline_filter.c
cc   dagfwddemo.o inline_filter.o  -L../../lib -
L../../lib/dag37t_api -ldag37t -ldag  -lpcap -lpthread -o
dagfwddemo
make[1]: Leaving directory `/usr/local/dag-
2.5.5/tools/dagfwddemo'
make[1]: Entering directory `/usr/local/dag-2.5.5/tools/daggen'
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o cmd.o
cmd.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o
cmdline.o cmdline.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o
dagcrc.o dagcrc.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o
daggen.o daggen.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o
dagrandom.o dagrandom.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o
list.o list.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o
packet.o packet.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o
toerf.o toerf.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o
gram.o gram.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strict-aliasing -
I../../include -I/usr/src/linux/include -DBYTESWAP   -c -o lex.o
lex.c
```

```
cc -L../../lib -lm -Wall -Wimplicit -Wformat -O2 -DNDEBUG -
DHAVE_CONFIG_H -pipe -D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-
strict-aliasing -I../../include -I/usr/src/linux/include -
DBYTESWAP -o daggen cmd.o cmdline.o dagcrc.o daggen.o
dagrandom.o list.o packet.o toerf.o gram.o lex.o
make[1]: Leaving directory `/usr/local/dag-2.5.5/tools/daggen'
make[1]: Entering directory `/usr/local/dag-
2.5.5/filtering/filter_loader'
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/main.o src/main.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/settings.o src/settings.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o ../common/dag_snort.o ../common/dag_snort.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o ../common/dagipf.o ../common/dagipf.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o ../common/filters.o ../common/filters.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o ../common/negation_tree.o ../common/negation_tree.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o ../common/rdtsc.o ../common/rdtsc.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o ../common/utilities.o ../common/utilities.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include -
L../../lib -ldag -o filter_loader src/main.o src/settings.o
../common/dag_snort.o ../common/dagipf.o ../common/filters.o
../common/negation_tree.o ../common/rdtsc.o
../common/utilities.o ../../lib/dagreg.o
make[1]: Leaving directory `/usr/local/dag-
2.5.5/filtering/filter_loader'
make[1]: Entering directory `/usr/local/dag-
2.5.5/filtering/snort_compiler'
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/parser.o src/parser.c
```

```
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/rule_lib.o src/rule_lib.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/rules.o src/rules.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/scanner.o src/scanner.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/settings.o src/settings.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/snort_filters.o src/snort_filters.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/main.o src/main.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include -
L../../lib -ldag -o snort_compiler src/parser.o src/rule_lib.o
src/rules.o src/scanner.o src/settings.o src/snort_filters.o
src/main.o ../common/filters.o ../common/negation_tree.o
../common/utilities.o
make[1]: Leaving directory `/usr/local/dag-
2.5.5/filtering/snort_compiler'
make[1]: Entering directory `/usr/local/dag-
2.5.5/filtering/tcpdump_compiler'
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/main.o src/main.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/settings.o src/settings.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/parse_tree_node.o src/parse_tree_node.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/scanner.o src/scanner.c
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include   -c -
o src/parser.o src/parser.c
```

```
cc -Wall -Wimplicit -Wformat -O2 -DNDEBUG -DHAVE_CONFIG_H -pipe
-D_GNU_SOURCE -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -I../common -I./src -I../../include -
L../../lib -ldag -o tcpdump_compiler src/main.o src/settings.o
src/parse_tree_node.o src/scanner.o src/parser.o
../common/filters.o ../common/negation_tree.o
../common/utilities.o
make[1]: Leaving directory `/usr/local/dag-
2.5.5/filtering/tcpdump_compiler'

Compilation succeeded!
To complete the installation, make install
```

Anexo 2.4 log al ejecutar make install

Finalmente cuando terminamos de instalar los drivers de la DAG mediante `make install` este es el resultado que muestra por pantalla:

```
make[1]: Entering directory `/usr/local/dag-2.5.5/arm'
installing firmware in /usr/local/share/dag/arm
/usr/bin/install -c -d /usr/local/share/dag/arm
/usr/bin/install -c dag3atm-erf.b /usr/local/share/dag/arm
make[1]: Leaving directory `/usr/local/dag-2.5.5/arm'
make[1]: Entering directory `/usr/local/dag-2.5.5/xilinx'
removing old links/images in xilinx
rm -f dag43gepcix-erf.bit
rm -f dag43pcix-erf.bit
rm -f dag38pcix-erf.bit
rm -f dag38pp-erf.bit
creating new links in xilinx
ln -s dag43gepcix-terf.bit dag43gepcix-erf.bit
ln -s dag43pcix-terf.bit dag43pcix-erf.bit
ln -s dag38pcix-terf.bit dag38pcix-erf.bit
ln -s dag38pp-terf.bit dag38pp-erf.bit
installing firmware in /usr/local/share/dag/xilinx
/usr/bin/install -c -d /usr/local/share/dag/xilinx
/usr/bin/install -c dag35pci-erf.bit dag35pp-erf.bit dag36atm-
erf.bit dag36epci-erf.bit dag36epp-erf.bit dag36gepp-erf.bit
dag36pci-erf.bit dag37gepci-terf.bit dag37tpci-atm-erf.bit
dag37tpci-hdlc-erf.bit dag38cp-aal5.bit dag38pcix-aal5.bit
dag38pcix-erf.bit dag38pcix-terf.bit dag38pp-erf.bit dag38pp-
terf.bit dag423eth-erf.bit dag43cp-aal5.bit dag43gepcix-erf.bit
dag43gepcix-terf.bit dag43pcix-aal5.bit dag43pcix-erf.bit
dag43pcix-terf.bit dag61pcix-erf.bit dag61rx-erf.bit dag62pcix-
erf.bit dag62rx-erf.bit /usr/local/share/dag/xilinx
removing old links/images in /usr/local/share/dag/xilinx
rm -f /usr/local/share/dag/xilinx/dag43gepcix-erf.bit
rm -f /usr/local/share/dag/xilinx/dag43pcix-erf.bit
rm -f /usr/local/share/dag/xilinx/dag38pcix-erf.bit
rm -f /usr/local/share/dag/xilinx/dag38pp-erf.bit
creating new links in /usr/local/share/dag/xilinx
cd /usr/local/share/dag/xilinx/; ln -s dag43gepcix-terf.bit
dag43gepcix-erf.bit
cd /usr/local/share/dag/xilinx/; ln -s dag43pcix-terf.bit
dag43pcix-erf.bit
cd /usr/local/share/dag/xilinx/; ln -s dag38pcix-terf.bit
dag38pcix-erf.bit
cd /usr/local/share/dag/xilinx/; ln -s dag38pp-terf.bit dag38pp-
erf.bit
make[1]: Leaving directory `/usr/local/dag-2.5.5/xilinx'
make[1]: Entering directory `/usr/local/dag-2.5.5/doc'
installing docs in /usr/local/share/doc/dag
/usr/bin/install -c -d /usr/local/share/doc/dag
/usr/bin/install -c EDM01.05-01_DAG_4.3GE_Card_User_Manual.pdf
EDM01.05-03_DAG_6.2S_Card_User_Manual.pdf EDM01.05-
04_DAG_3.6GE_Card_User_Manual.pdf EDM01.05-
05_DAG_3.6E_Card_User_Manual.pdf EDM01.05-
```

```
07_DAG_3.7G_Card_User_Manual.pdf EDM01.05-
09_DAG_3.8S_Card_User_Manual.pdf EDM01.05-
10_DAG_4.3S_Card_User_Manual.pdf EDM01.05-
11_DAG_6.1S_Card_User_Manual.pdf EDM01.05-
12_DAG_3.7T_Card_User_Manual.pdf EDM01.05-
13_DAG_3.5S_Card_User_Manual.pdf EDM01.05-
14_DAG_3.6D_Card_User_Manual.pdf EDM01.05-
16_DAG_4.2GE_Card_User_Manual.pdf EDM02.05-
01_Coprocessor_IP_Filter_User_Manual.pdf EDM02.05-
03_Coprocessor_ATM_Reassembler_User_Manual.pdf EDM02.05-
10_Endace_Linux_Windows_Release_Notes_2.5.5.pdf EDM04.05-
01_Linux_FreeBSD_Installation_Manual.pdf EDM04.05-
03_dagflood_User_Manual.pdf EDM04.05-
04_dagfwddemo_Program_User_Manual.pdf EDM04.05-
06_daggen_User_Manual.pdf EDM05.05-01_TDS2_TDS6_User_Manual.pdf
Licence.pdf /usr/local/share/doc/dag
make[1]: Leaving directory `/usr/local/dag-2.5.5/doc'
make[1]: Entering directory `/usr/local/dag-2.5.5/include'
installing headers in /usr/local/include/
/usr/bin/install -c -d /usr/local/include
/usr/bin/install -c dagapi.h dagcrc.h daginf.h dag37t_api.h
dagnew.h dagpbm.h dagreg.h dagutil.h dagclarg.h dag_platform.h
dag_platform_linux.h dag_platform_freebsd.h
dag_platform_solaris.h dag_platform_win32.h
dag_platform_macosx.h /usr/local/include
make[1]: Leaving directory `/usr/local/dag-2.5.5/include'
make[1]: Entering directory `/usr/local/dag-2.5.5/lib'
installing libraries in /usr/local/lib/
/usr/bin/install -c -d /usr/local/lib
/usr/bin/install -c libdag.a libdag.so.1.7 /usr/local/lib
# remove any existing links
rm -f /usr/local/lib/libdag.so.1
rm -f /usr/local/lib/libdag.so
# make the new links
cd /usr/local/lib; ln -s libdag.so.1.7  libdag.so.1
cd /usr/local/lib; ln -s libdag.so.1.7  libdag.so
/sbin/ldconfig
make[1]: Leaving directory `/usr/local/dag-2.5.5/lib'
make[1]: Entering directory `/usr/local/dag-
2.5.5/lib/dag37t_api'
installing libraries in /usr/local/lib/
/usr/bin/install -c -d /usr/local/lib
/usr/bin/install -c libdag37t.a libdag37t.so.1.0 /usr/local/lib
# remove any existing links
rm -f /usr/local/lib/libdag37t.so.1
rm -f /usr/local/lib/libdag37t.so
# make the new links
cd /usr/local/lib; ln -s libdag37t.so.1.0  libdag37t.so.1
cd /usr/local/lib; ln -s libdag37t.so.1.0  libdag37t.so
/sbin/ldconfig
make[1]: Leaving directory `/usr/local/dag-2.5.5/lib/dag37t_api'
make[1]: Entering directory `/usr/local/dag-2.5.5/drv/divdi3'
/usr/bin/install -c -d /lib/modules/`uname -r`/extra
/usr/bin/install -c divdi3 /lib/modules/`uname -r`/extra
depmod -a >/dev/null
make[1]: Leaving directory `/usr/local/dag-2.5.5/drv/divdi3'
```

```
make[1]: Entering directory `/usr/local/dag-2.5.5/drv/linux'
/usr/bin/install -c -d /lib/modules/`uname -r`/endace
/usr/bin/install -c dag dagmem /lib/modules/`uname -r`/endace
/usr/bin/install -c -d /usr/local/bin
/usr/bin/install -c dagload /usr/local/bin
depmod -aeu >/dev/null
====>>> Dont forget to update /etc/modules
make[1]: Leaving directory `/usr/local/dag-2.5.5/drv/linux'
make[1]: Entering directory `/usr/local/dag-2.5.5/tools'
/usr/bin/install -c -d /usr/local/bin
list='dagreset daginf dagld dagrom dagsnap dagbits dagthree
dagfour dagsix dagclock dagsar dagvcam dagchan dagflood dagmap
dagpps dagwatchdog dagswid dagimademo dagbug'; for item in
$list; do /usr/bin/install -c $item /usr/local/bin; done
make[1]: Leaving directory `/usr/local/dag-2.5.5/tools'
make[1]: Entering directory `/usr/local/dag-2.5.5/filtering'
make[1]: Nothing to be done for `install'.
make[1]: Leaving directory `/usr/local/dag-2.5.5/filtering'
make[1]: Entering directory `/usr/local/dag-
2.5.5/tools/dagconvert'
/usr/bin/install -c dagconvert /usr/local/bin
make[1]: Leaving directory `/usr/local/dag-
2.5.5/tools/dagconvert'
make[1]: Entering directory `/usr/local/dag-
2.5.5/tools/dagfwddemo'
/usr/bin/install -c dagfwddemo /usr/local/bin
make[1]: Leaving directory `/usr/local/dag-
2.5.5/tools/dagfwddemo'
make[1]: Entering directory `/usr/local/dag-2.5.5/tools/daggen'
/usr/bin/install -c daggen /usr/local/bin
make[1]: Leaving directory `/usr/local/dag-2.5.5/tools/daggen'
make[1]: Entering directory `/usr/local/dag-
2.5.5/filtering/filter_loader'
/usr/bin/install -c filter_loader /usr/local/bin
make[1]: Leaving directory `/usr/local/dag-
2.5.5/filtering/filter_loader'
make[1]: Entering directory `/usr/local/dag-
2.5.5/filtering/snort_compiler'
/usr/bin/install -c snort_compiler /usr/local/bin
make[1]: Leaving directory `/usr/local/dag-
2.5.5/filtering/snort_compiler'
make[1]: Entering directory `/usr/local/dag-
2.5.5/filtering/tcpdump_compiler'
/usr/bin/install -c tcpdump_compiler /usr/local/bin
make[1]: Leaving directory `/usr/local/dag-
2.5.5/filtering/tcpdump_compiler'

To complete the library installation you can either
  1) Add /usr/local/lib to /etc/ld.so.conf and run ldconfig, OR
  2) Add /usr/local/lib to LD_LIBRARY_PATH
     e.g add the following line to your .bash_profile
     export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

Anexo 3.1 start-up script

Este es el *script* necesario para poner RADclock en marcha tras su instalación.

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          radclock
# Required-Start:    mountvirtfs ifupdown $local_fs
# Default-Start:     S
# Default-Stop:      0 6
### END INIT INFO

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=`which radclock`
NAME=radclock
DESC="the $NAME daemon"
DAEMON_OPTS="-d"

test -x $DAEMON || exit 0

# Include radclock defaults if available
if [ -f /etc/default/radclock ] ; then
        . /etc/default/radclock
fi

set -e

case "$1" in
  start)
        echo -n "Starting $DESC: "
        start-stop-daemon --start --quiet --pidfile
/var/run/radclock/$NAME.pid \
                --exec $DAEMON -- $DAEMON_OPTS
        echo "done."
        ;;
  stop)
        echo -n "Stopping $DESC: "
        start-stop-daemon --stop --quiet --retry
TERM/2/TERM/forever/TERM \
                --pidfile /var/run/radclock/$NAME.pid --exec $DAEMON
        echo "done."
        ;;
  reload)
        echo "Reloading $DESC configuration files."
        start-stop-daemon --stop --signal 1 --quiet --pidfile \
                /var/run/radclock/$NAME.pid --exec $DAEMON
  ;;
  restart)
    echo -n "Restarting $DESC: "
        start-stop-daemon --stop --quiet --retry
TERM/2/TERM/forever/TERM \
                --pidfile /var/run/radclock/$NAME.pid --exec $DAEMON
        sleep 2
        start-stop-daemon --start --quiet --pidfile \
                /var/run/radclock/$NAME.pid --exec $DAEMON --
$DAEMON_OPTS
        echo "done."
        ;;
  status)
        echo -n "$DESC status:  "
```

```
        if [ ! -r "/var/run/radclock/$NAME.pid" ]; then
                echo "is not running."
                exit 3
        fi
        if read pid < "/var/run/radclock/$NAME.pid" && ps -p "$pid" >
/dev/null 2>&1; then
                echo "is running with pid $pid."
                exit 0
        else
                echo "is not running but /var/run/radclock/$NAME.pid
exists."
                exit 1
        fi
        ;;
  *)
        N=/etc/init.d/$NAME
        echo "Usage: $N {start|stop|restart|reload|status}" >&2
        exit 1
        ;;
esac

exit 0
```

Anexo 3.2 radclock.conf

Esta es la configuración utilizada para las primeras pruebas de funcionamiento de RADclock:

```
##
## This is the default configuration file for the RADclock
##

#---------------------------------------------------------------------
-------#
# Package version. Do not modify this line.
radclock_version = 0.3.1
#---------------------------------------------------------------------
-------#


# Verbosity level of the radclock daemon.
#     quiet : only errors and warnings are logged
#     normal: adaptive logging of events
#     high  : include debug messages
verbose_level = normal




#---------------------------------------------------------------------
-------#
# Synchronisation Client parameters
#---------------------------------------------------------------------
-------#

# Specify the type of underlying synchronisation used.
# Note that piggybacking requires an NTP daemon running and is then
# incompatible with the RADclock serving clients over the network or
# adjusting the system clock. Piggybacking disables these functions.
#     piggy   : piggybacking on running NTP daemon
#     ntp     : RADclock uses NTP protocol
#     ieee1588: RADclock uses IEEE 1588 protocol - NOT IMPLEMENTED YET
#     pps     : RADclock listens to PPS - NOT IMPLEMENTED YET
synchronisation_type = ntp

# The polling period specifies the time interval at which
# the requests for time are sent to the server (value in seconds)
polling_period = 16

# Hostname or IP address (uses lookup name resolution).
# Automatic detection will be attempted if not specified.
hostname = servgenmonII

# Time server answering the requests from this client.
# Can be a host name or an IP address (uses lookup name resolution).
time_server = ntp.cesca.cat




#---------------------------------------------------------------------
-------#
```

```
# Synchronisation Server parameters
#-------------------------------------------------------------------
-------#

# IPC server.
# Serves time to the kernel and other processes.
#     on : Start service - makes the RADclock available to other
programs
#     off: Stop service  - useful when replaying traces
ipc_server = on

# NTP server.
# Serves time to radclock clients over the network.
#     on : runs a NTP server for remote clients
#     off: no server running
ntp_server = off

# System clock.
# Let the RADclock adjust the system clock, make sure no other
synchronisation
# daemon is running, especially the NTP daemon.
# Note that this feature relies on standard kernel calls to adjust the
time and
# is completely different from the IPC server provided. This feature
is essentially
# provided to maintain system time for non critical operations. If you
care about
# synchronisation, turn the IPC server on and use the libradclock API
# Also note that system clock time causality may break since the
system clock will
# be set on RADclock restart. The system clock will tick monotically
afterwards.
#     on : adjust the system clock
#     off: does not adjust the system clock (to use with NTP
piggybacking)
adjust_system_clock = on



#-------------------------------------------------------------------
-------#
# Virtual Machine Environment parameters
#-------------------------------------------------------------------
-------#

# The role of the radclock in a virtual machine environment (if any).
# It is usually better to give the master role to the radclock
instance
# running in the host system, and the slave roles to the radclock
running
# in the guest system.
# Possible values are:
#     none        : no virtual machine environment
#     xen-master  : radclock creates time for all Xen systems
#     xen-slave   : radclock gets its time from a Xen master
#     vbox-master : radclock creates time for all Virtual Box systems
#     vbox-slave  : radclock gets its time from a Virtual Box master
virtual_machine_mode = none
```

```
#-------------------------------------------------------------------------
-------#
# Environment and Tuning parameters
#-------------------------------------------------------------------------
-------#

# Temperature environment and hardware quality.
# Keywods accepted are: poor, good, excellent.
# This setting overrides temperature and hardware expert mode (default
behavior).
temperature_quality = good

# EXPERIMENTAL.
# Temperature environment and hardware quality - EXPERT.
# This settings are over-written by the temperature_quality keyword.
#ts_limit = 0.000015000
#skm_scale = 1024.000000000
#rate_error_bound = 0.000000500
#best_skm_rate = 0.000000200
#offset_ratio = 6
#plocal_quality = 0.000000800

# Specify if p_local should be started.
# Possible values are:
#     on
#     off
#     restart
start_local_phat = on

# For a quick start, the initial value of the period of the counter
(in seconds).
init_period_estimate = 1e-09

# Estimation of the asym within the host (in seconds).
host_asymmetry = 0.000000

# Estimation of the network asym (in seconds).
network_asymmetry = 0.000000




#-------------------------------------------------------------------------
-------#
# Input / Output parameters
#-------------------------------------------------------------------------
-------#

# Network interface.
# Specify a different interface (xl0, eth0, ...)
# If none, the RADclock will lookup for a default one.
network_device = eth0

# Synchronisation data input file (modified pcap format).
# Replay mode requires a file produced by the RADclock.
#sync_input_pcap = sync_input.pcap

# Synchronisation data input file (ascii format).
# Replay mode requires a file produced by the RADclock.
```

```
#sync_input_ascii = sync_input.ascii

# Synchronisation data output file (modified pcap format).
sync_output_pcap = sync_output.pcap

# Synchronisation data output file (ascii format).
sync_output_ascii = sync_output.ascii

# Internal clock data output file (ascii format).
clock_output_ascii = clock_output.ascii
```

Anexo 3.3 radclock.log

Este es un ejemplo de la información que podemos encontrar en el archivo radclock.log después de terminar la conexión con el servidor de forma manual.

```
-RADclock Init-: Info:      RADclock - configuration summary
-RADclock Init-: Info:      radclock version     : 0.3.1
-RADclock Init-: Info:      Configuration file   : radclock.conf
-RADclock Init-: Info:      Log file             :
-RADclock Init-: Info:      Verbose level        : normal
-RADclock Init-: Info:      Client sync          : ntp
-RADclock Init-: Info:      Server IPC           : on
-RADclock Init-: Info:      Server NTP           : off
-RADclock Init-: Info:      Adjust system clock  : on
-RADclock Init-: Info:      Virtual Machine mode : none
-RADclock Init-: Info:      Polling period       : 16
-RADclock Init-: Info:      TSLIMIT              : 0.000015000
-RADclock Init-: Info:      SKM_SCALE            : 1024.000000000
-RADclock Init-: Info:      RateErrBound         : 0.000000500
-RADclock Init-: Info:      BestSKMrate          : 0.000000200
-RADclock Init-: Info:      offset_ratio         : 6
-RADclock Init-: Info:      plocal_quality       : 0.000000800
-RADclock Init-: Info:      Using plocal         : on
-RADclock Init-: Info:      Initial phat         : 3.3333e-10
-RADclock Init-: Info:      Host asymmetry       : 0.000000
-RADclock Init-: Info:      Network asymmetry    : 0.000000
-RADclock Init-: Info:      Host name            : servgenmonII
-RADclock Init-: Info:      Time server          : ntp.cesca.cat
-RADclock Init-: Info:      Interface            : eth0
-RADclock Init-: Info:      pcap sync input      :
-RADclock Init-: Info:      ascii sync input     :
-RADclock Init-: Info:      pcap sync output     : sync_output.pcap
-RADclock Init-: Info:      ascii sync output    : sync_output.ascii
-RADclock Init-: Info:      ascii clock output   : clock_output.ascii
-RADclock Init-: Info:      Feed-Forward kernel support detected
(version: 1)
-RADclock Init-: Info:      registered get_vcounter syscall at 299
-RADclock Init-: Info:      registered get_vcounter_latency syscall at
300
-RADclock Init-: Info:      Clocksource used is tsc
-RADclock Init-: Info:      Initialising radclock_get_vcounter with
syscall.
-RADclock Init-: Info:      Creating live input source
-RADclock Init-: Info:      Found address 127.0.0.2
-RADclock Init-: Info:      Found IP address 147.83.118.240 based on
interface eth0
-RADclock Init-: Info:      Using host name servgenmonII and address
147.83.118.240
-RADclock Init-: Info:      Opening device eth0
-RADclock Init-: Info:      Packet filter: (src host 147.83.118.240
and dst port 123 and dst host ntp.cesca.cat) or (src host
ntp.cesca.cat and dst host 147.83.118.240 and src port 123)
-RADclock Init-: Info:      Reading from live interface eth0,
linktype: EN10MB
-RADclock Init-: Info:      Capture mode is SYSCLOCK
-RADclock Init-: Info:      Backed up existing output file:
sync_output.pcap
-RADclock Init-: Info:      Backed up existing output file:
sync_output.ascii
```

```
-RADclock Init-: Info:      Backed up existing output file:
clock_output.ascii
Jan 01 17:57:51: Info:      Global data generic netlink id is 19
Jan 01 17:57:51: Info:      Feed-Forward Kernel initialised
Jan 01 17:57:51: Info:      Starting trigger thread
Jan 01 17:57:51: Info:      Starting data processing thread
Jan 01 17:57:51: Info:      Starting IPC thread
Jan 01 17:57:51: Info:      Starting fixedpoint thread
Jan 01 17:57:51: Info:      IPC thread initialised.
Jan 01 17:57:52: Warning:   New NTP server info on packet 2:
Jan 01 17:57:52: Warning:   SERVER - IP: 84.88.0.106, STRATUM: 1, TTL:
59, ID: PPS, MODE: 4, LEAPINFO: 0
Jan 01 17:57:52: Warning:   HOST   - IP: 147.83.118.240, DST_PORT:
47504
Jan 01 17:57:52: Sync:      Initialising RADclock synchronization
Jan 01 17:57:52: Control:   Adjusting Warmup window
Jan 01 17:57:52: Control:   Warmup window smaller than algo windows,
increasing from 100 to 480 stamps
Jan 01 17:57:52: Control:   Warmup Adjustment Complete
Jan 01 17:57:52: Control:   Machine Parameters:  TSLIMIT: 1.5e-05,
SKM_SCALE: 1024, RateErrBOUND: 5e-07, BestSKMrate: 2e-07
Jan 01 17:57:52: Control:   Network Parameters:  poll_period: 16,
h_win: 37800
Jan 01 17:57:52: Control:   Windows (in pkts):   warmup: 480, history:
37800, shift: 100 (thres =  150 [mus]), plocal: 320, offset: 64 (SKM
scale is 64)
Jan 01 17:57:52: Control:   Error thresholds :   phat:  Ep 0.045 [ms],
Ep_qual 0.1 [PPM], plocal:  Eplocal_qual 0.8 [PPM]
Jan 01 17:57:52: Control:                        offset:  Eoffset 0.09
[ms], Eoffset_qual 0.3 [ms]
Jan 01 17:57:52: Control:   Sanity Levels:       phat  1.5e-06, plocal
1.5e-06, offset: absolute:    1.5 [ms], rate:  0.01 [ms]/[sec] (  0.16
[ms]/[stamp])
Jan 01 17:57:52: Sync:      i=0: Beginning Warmup Phase. Stamp read
check: 61072663064468  1317134300.6717786584  1317134300.6720258624
61072668269633
Jan 01 17:57:52: Sync:      i=0: Assuming 1Ghz oscillator, 1st
vcounter stamp is 0.707 [days] (1017.9 [min]) since reset, RTT is
5.205 [ms], SD 247.204 [mus]
Jan 01 17:57:52: Sync:      i=0: After initialisation:
(far,near)=(0,0), phat = 3.333303552e-10, perr =     0, C =
1317113943.2993
Sep 27 16:38:20: Control:   i=0: NTPserver stamp 1317134300.672026,
RAD - NTPserver = 1.488 [ms], RTT/2 = 0.000 [ms]
Sep 27 16:38:20: Control:   i=0: Clock Error Bound (cur,avg,std)
0.000000 0.000000 0.000000 [ms]
Sep 27 16:38:22: Sync:      i=1: phat update (far,near)=(0,1), phat =
3.333202571e-10, rel diff = -3.029546764e-05, perr = 0, C =
1317113943.9160
Sep 27 16:38:22: Sync:      i=1: th_naive = 0.744 [ms], thetahat =
0.744 [ms], wsum = 1.00000, minET = 0.00000 [ms], RTThat/2 = 0.856
[ms]
Sep 27 16:38:22: Control:   i=1: NTPserver stamp 1317134302.672020,
RAD - NTPserver = 0.732 [ms], RTT/2 = 0.856 [ms]
Sep 27 16:38:22: Control:   i=1: Clock Error Bound (cur,avg,std)
0.000000 0.000000 0.000000 [ms]
Sep 27 16:38:24: Sync:      i=2: phat update (far,near)=(0,2), phat =
3.333283998e-10, rel diff = 2.442835981e-05, perr = 1.181207262e-05, C
= 1317113943.4187
```

```
Sep 27 16:38:24: Sync:      i=2: th_naive = 0.695 [ms], thetahat =
0.718 [ms], wsum = 1.88015, minET = 0.00000 [ms], RTThat/2 = 0.840
[ms]
Sep 27 16:38:24: Control:   i=2: NTPserver stamp 1317134304.672115,
RAD - NTPserver = 0.694 [ms], RTT/2 = 0.840 [ms]
Sep 27 16:38:24: Control:   i=2: Clock Error Bound (cur,avg,std)
0.000000 0.000000 0.000000 [ms]
Sep 27 16:38:26: Sync:      i=3: phat update (far,near)=(0,3), phat =
3.333379129e-10, rel diff = 2.853896957e-05, perr = 1.384506935e-05, C
= 1317113942.8376
Sep 27 16:38:26: Sync:      i=3: th_naive = 0.581 [ms], thetahat =
0.718 [ms], wsum = 1.88011, minET = 0.00040 [ms], RTThat/2 = 0.840
[ms]
Sep 27 16:38:26: Control:   i=3: NTPserver stamp 1317134306.672450,
RAD - NTPserver = 0.696 [ms], RTT/2 = 0.840 [ms]
Sep 27 16:38:26: Control:   i=3: Clock Error Bound (cur,avg,std)
0.000400 0.000133 0.000231 [ms]
Sep 27 16:38:28: Sync:      i=4: phat update (far,near)=(1,4), phat =
3.333359334e-10, rel diff = -5.938298859e-06, perr = 4.560056878e-05,
C = 1317113942.9585
Sep 27 16:38:28: Sync:      i=4: th_naive = 0.711 [ms], thetahat =
0.711 [ms], wsum = 1.81504, minET = 0.00000 [ms], RTThat/2 = 0.805
[ms]
Sep 27 16:38:28: Control:   i=4: NTPserver stamp 1317134308.672289,
RAD - NTPserver = 0.690 [ms], RTT/2 = 0.805 [ms]
Sep 27 16:38:28: Control:   i=4: Clock Error Bound (cur,avg,std)
0.000000 0.000100 0.000200 [ms]
Sep 27 16:38:30: Sync:      i=5: th_naive = 0.885 [ms], thetahat =
0.711 [ms], wsum = 1.80850, minET = 0.00040 [ms], RTThat/2 = 0.805
[ms]
Sep 27 16:38:30: Control:   i=5: NTPserver stamp 1317134310.672470,
RAD - NTPserver = 1.089 [ms], RTT/2 = 0.805 [ms]
Sep 27 16:38:30: Control:   i=5: Clock Error Bound (cur,avg,std)
0.000400 0.000160 0.000219 [ms]
Sep 27 16:38:32: Sync:      i=6: phat update (far,near)=(1,5), phat =
3.333286714e-10, rel diff = -2.178658751e-05, perr = 1.689477036e-05,
C = 1317113943.4022
Sep 27 16:38:32: Sync:      i=6: th_naive = 0.916 [ms], thetahat =
0.711 [ms], wsum = 1.80197, minET = 0.00080 [ms], RTThat/2 = 0.805
[ms]
Sep 27 16:38:32: Control:   i=6: NTPserver stamp 1317134312.672482,
RAD - NTPserver = 1.105 [ms], RTT/2 = 0.805 [ms]
Sep 27 16:38:32: Control:   i=6: Clock Error Bound (cur,avg,std)
0.000800 0.000267 0.000327 [ms]
Sep 27 16:38:34: Sync:      i=7: phat update (far,near)=(1,7), phat =
3.333347e-10, rel diff = 1.808578401e-05, perr = 7.053005277e-05, C =
1317113943.0338
Sep 27 16:38:34: Sync:      i=7: th_naive = 0.704 [ms], thetahat =
0.706 [ms], wsum = 1.36793, minET = 0.00000 [ms], RTThat/2 = 0.758
[ms]
Sep 27 16:38:34: Control:   System clock set to 1317134314.673522
[sec]
Sep 27 16:38:34: Control:   i=7: NTPserver stamp 1317134314.672524,
RAD - NTPserver = 0.635 [ms], RTT/2 = 0.758 [ms]
Sep 27 16:38:34: Control:   i=7: Clock Error Bound (cur,avg,std)
0.000000 0.000229 0.000315 [ms]
Sep 27 16:38:36: Sync:      i=8: phat update (far,near)=(2,7), phat =
3.333343315e-10, rel diff = -1.105469286e-06, perr = 1.628521076e-05,
C = 1317113943.0564
```

```
Sep 27 16:38:36: Sync:      i=8: th_naive = 0.776 [ms], thetahat =
0.706 [ms], wsum = 1.36754, minET = 0.00040 [ms], RTThat/2 = 0.758
[ms]
Sep 27 16:38:36: Control:   i=8: NTPserver stamp 1317134316.672663,
RAD - NTPserver = 0.812 [ms], RTT/2 = 0.758 [ms]
Sep 27 16:38:36: Control:   i=8: Clock Error Bound (cur,avg,std)
0.000400 0.000250 0.000298 [ms]
Sep 27 16:38:52: Sync:      i=9: th_naive = 0.741 [ms], thetahat =
0.718 [ms], wsum = 2.01846, minET = 0.00360 [ms], RTThat/2 = 0.758
[ms]
Sep 27 16:38:56: Info:      Breaking current capture loop for rehash
Sep 27 16:38:56: Info:      Send killing signal to threads. Wait for
stop message.
Sep 27 16:38:56: Info:      Thread IPC server is terminating.
Sep 27 16:38:56: Info:      IPC thread is dead.
Sep 27 16:39:02: Info:      Thread fixedpoint is terminating.
Sep 27 16:39:08: Info:      Thread trigger is terminating.
Sep 27 16:39:08: Info:      Thread data processing is terminating.
Sep 27 16:39:08: Info:      Trigger thread is dead.
Sep 27 16:39:08: Info:      Kernel fixedpoint thread is dead.
Sep 27 16:39:08: Info:      Data processing thread is dead.
Sep 27 16:39:08: Info:      Threads are dead.
Sep 27 16:39:08: Info:      20 NTP packets captured
Sep 27 16:39:08: Info:      0 missed NTP packets
Sep 27 16:39:08: Info:      10 valid timestamp tuples extracted
Sep 27 16:39:08: Info:      Last estimate of the clock source period:
3.333343315e-10
Sep 27 16:39:08: Info:      RADclock stopped
```

Anexo 3.4 Máquina de estados de RADclock

En este anexo explicamos lo que hemos podido ver de la máquina de estados de RADclock. Esta máquina de estados no está explicada por el autor en ningún manual, y por lo tanto solo aparecen los estados y las transiciones de estados que hemos podido observar en las diferentes pruebas realizadas con RADclock.

Los estados que hemos podido observar son:

3 --> STARAD_UNSYNC (no sincronizado) + STRAD_WARMUP (en fase de *warm-up*). El reloj RADclock aun no ha sido sincronizado con el reloj GPS.

4 --> STRAD_KCLOCK (RADclock *time* es fiable). Estimaciones de error creíbles.

6 --> STRAD_WARMUP (en fase de *warm-up*) + STRAD_KCLOCK (RADclock *time* es fiable). Aun estamos en la fase de *warm-up*, pero ya se considera que el reloj esta correctamente sincronizado.

7 --> STARAD_UNSYNC (no sincronizado) + STRAD_WARMUP (en fase de *warm-up*) + STRAD_KCLOCK (RADclock time es fiable). El reloj RADclock aun no ha sido sincronizado con el reloj GPS pero ya obtiene datos fiables de un servidor NTP.

14 --> STRAD_WARMUP (en fase de *warm-up*) + STRAD_KCLOCK (RADclock *time* es fiable) + STARAD_SYSCLOCK (*System clock* bastante exacto (ajustado por RADclock). El TSC ha sido bastante exacto respecto a las previsiones calculadas por RADclock.

15 --> STARAD_UNSYNC (no sincronizado) + STRAD_WARMUP (en fase de *warm-up*) + STRAD_KCLOCK (RADclock *time* es fiable) + STARAD_SYSCLOCK (*System clock* bastante exacto (ajustado por RADclock). Igual que el anterior, pero antes de los primeros 25 *stamps*.

19 --> STARAD_UNSYNC (no sincronizado) + STRAD_WARMUP (en fase de *warm-up*) + STRAD_STARVING (falta entrada de datos de calidad). Este es el estado inicial en la mayoría de las pruebas realizadas. Es el obtenido después de la primera respuesta NTP.

36 --> STRAD_KCLOCK (RADclock *time* es fiable) + STARAD_PERIOD_QUALITY (mala calidad de la estimación periodo de RADclock: por encima del umbral). El reloj RADclock está sincronizado, pero las estimaciones del periodo no han sido buenas.

132 --> STRAD_KCLOCK (RADclock *time* es fiable) + STARAD_OFFSET_QUALITY (mala estimación del error de *offset*: por encima del umbral). El reloj RADclock está sincronizado, pero las estimaciones del offset no han sido buenas.

135 --> STARAD_UNSYNC (no sincronizado) + STRAD_WARMUP (en fase de *warm-up*) + STRAD_KCLOCK (RADclock *time* es fiable) + STARAD_OFFSET_QUALITY (mala estimación del error de *offset*: por encima del umbral).

140 --> STARAD_KCLOCK (RADclock *time* es fiable) + STARAD_SYSCLOCK (*System clock* bastante exacto (ajustado por RADclock) + STARAD_OFFSET_QUALITY (mala estimación del error de *offset*: por encima del umbral).

148 --> STARAD_KCLOCK (RADclock *time* es fiable) + STRAD_STARVING (falta entrada de datos de calidad) + STARAD_OFFSET_QUALITY (mala estimación del error de *offset*: por encima del umbral). Este estado aparece después de una caída larga de la conexión con el servidor NTP. Nos indica que tiene que volver a recibir datos de calidad.

164 --> STRAD_KCLOCK (RADclock *time* es fiable) + STARAD_PERIOD_QUALITY (mala calidad de la estimación periodo de RADclock: por encima del umbral) + STARAD_OFFSET_QUALITY (mala estimación del error de *offset*: por encima del umbral).

263 --> STARAD_UNSYNC (no sincronizado) + STRAD_WARMUP (en fase de *warm-up*) + STRAD_KCLOCK (RADclock *time* es fiable) + STARAD_OFFSET_SANITY (activa chequeos de la actualización del *offset*). El *flag* 0x00000100 nos indica que se están realizando correcciones en el *offset*.

391 --> STARAD_UNSYNC (no sincronizado) + STRAD_WARMUP (en fase de *warm-up*) + STRAD_KCLOCK (RADclock *time* es fiable) + STARAD_OFFSET_QUALITY (mala estimación del error de *offset*: por encima del umbral) + STARAD_OFFSET_SANITY (activa chequeos de la actualización del *offset*). El *flag* 0x00000100 nos indica que se están realizando correcciones en el *offset*.

En este diagrama podemos ver las transiciones observadas a lo largo de las pruebas de funcionamiento realizadas:

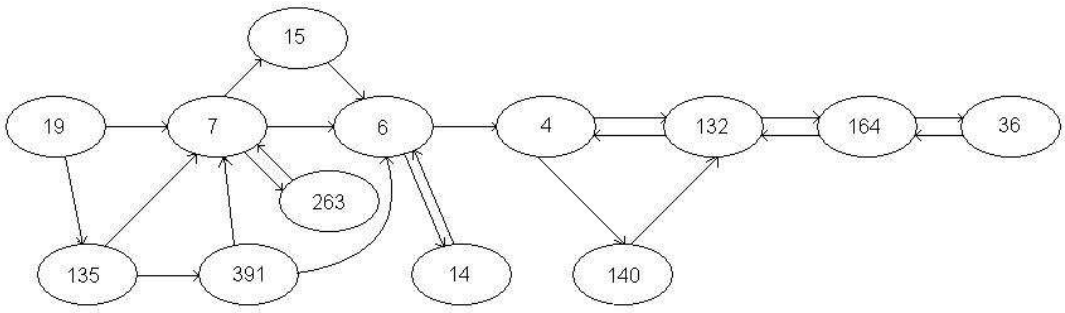

Fig.A3.1 Diagrama de estado de RADclock

Anexo 3.5 configuración del kernel en servgenmonI y servgenmonII

Este anexo se encuentra en el DVD que incluye este TFC.

Anexo 3.6 configuración del kernel en radclockserv

Este anexo se encuentra en el DVD que incluye este TFC.

Anexo 4.1 Máquinas empleadas en el laboratorio.

A lo largo del TFC hemos hecho referencia a los tres PC's utilizados para las pruebas: servgenmonI, servgenmonII y radclocserv. En este anexo detallamos cuáles son sus características:

- servgenmonI y servgenmonII:
  - Hardware
    - Placa base Supermicro X6DHE-XG2: Chipset Intel E7520
    - Microprocesador Xeon 3.0 GHz Nocona
    - Memoria Ram 2x 1024 Mbytes DDR2 ECC
    - HarDsc 120 Gbytes SERIAL ATA 72000 RPM
    - Tarjeta grafica Nvidia PCI Express de 258 Mbytes
    - Endace Dag 4.3GE PCI-X 133MHz
  - Software
    - Para pruebas RADclock
      - SUSE Linux 11.2 x64
      - Kernel 2.6.32
      - Gnome 2.28.2
      - RADclock 0.3.1
    - Para pruebas MGEN
      - Debian 3.1 x64
      - Kernel 2.4.27
      - Dag-2.5.5
- Radclockserv
  - Hardware
    - Microprocesador Intel Pentium 4 2.4GHz
    - Memoria RAM 256 Mbytes
    - nVidia GeForce4 MX 440
    - Intel EtherExpress PRO/100 S Desktop Adapter
  - Software
    - SUSE Linux 11.2 x32
    - Kernel 2.6.32
    - Gnome 2.28.2
    - RADclock 0.3.1

Anexo 4.2 Pruebas realizadas con DAG

Este anexo se encuentra en el DVD que incluye este TFC.

Anexo 4.3 Pruebas realizadas con RADclock

Este anexo se encuentra en el DVD que incluye este TFC.