DISEÑO DE UN ESQUEMA DE MARCACIÓN DE TIEMPOS DE LLEGADA DE PAQUETES DE PRUEBA EN UN ESTIMADOR DE ANCHO DE BANDA DISPONIBLE A TRAVÉS DE LA NETFPGA CON EL FIN DE REDUCIR SU ERROR DE ESTIMACIÓN.

NYDIA SUSANA SANDOVAL CARRERO

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA

FACULTAD DE INGENIERÍA DE SISTEMAS

MAESTRÍA EN TELEMÁTICA MODALIDAD INVESTIGACION

GRUPO DE INVESTIGACIÓN TECNOLOGÍAS DE INFORMACIÓN

LINEA DE INVESTIGACION TELEMATICA

BUCARAMANGA, Agosto 2015

DISEÑO DE UN ESQUEMA DE MARCACIÓN DE TIEMPOS DE LLEGADA DE PAQUETES DE PRUEBA EN UN ESTIMADOR DE ANCHO DE BANDA DISPONIBLE A TRAVÉS DE LA NETFPGA CON EL FIN DE REDUCIR SU ERROR DE ESTIMACIÓN.

Ing. NYDIA SUSANA SANDOVAL CARRERO

Trabajo de Grado para optar por el título de

Magíster en Telemática - Modalidad Investigación

DIRECTOR: PhD. CESAR DARIO GUERRERO SANTANDER

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA

FACULTAD DE INGENIERÍA DE SISTEMAS

MAESTRÍA EN TELEMÁTICA MODALIDAD INVESTIGACION

GRUPO DE INVESTIGACIÓN TECNOLOGÍAS DE INFORMACIÓN GTI

LINEA DE INVESTIGACION TELEMATICA

BUCARAMANGA, Agosto 2015

DEDICATORIA

Dedico este trabajo en primer lugar a Dios por todas sus bendiciones

A mis dos amores, mis hijas Tatiana y Deisy, por su paciencia y tiempo

A mi Padres, José y Susana por su amor

A mis hermanos, William y Leidy por su apoyo incondicional

Y mi hermoso sobrino Alejandro por alegrarnos la vida

GRACIAS!!!

AGRADECIMIENTOS

Después de mucho esfuerzo y trabajo llegó hora de Dar las Gracias

En primer Lugar a DIOS

Gracias al Director del trabajo de grado PhD Cesar Darío Guerrero

Gracias al grupo de investigación GTI, y Docentes de la Maestría

Gracias al asesor PhD Jhon Jairo Padilla

Gracias al MSc Diego Armando Reyes

Gracias a todos por su grandiosa colaboración para ver hoy culminado este trabajo.

CONTENIDO

pág.

LISTA DE FIGURAS

Pág

LISTA DE ECUACIONES

Pág

LISTA DE TABLAS

Pág

ANEXOS

Pág

ABREVIATURAS

ABW	*Available Bandwidth*	Ancho de banda disponible
DMA	*Direct Memory Access*	Acceso a memoria directo
FIFO	*First In, First Out*	Primero en entrar, primero en salir
FPGA	*Field Programmable Gate Array*	Arreglo de compuertas programable
HATS	*High Accuracy Timestamping System*	Sistema de marcado de tiempo de alta precisión
IP	*Internet Protocol*	Protocolo de Internet
MAC	*Media Access Control*	Control de acceso al medio
NIC	*Network interface Card*	Tarjeta de Red
PGM	*Probe Gap Model*	Modelo de separación de prueba
PRM	*Probe Rate Model*	Modelo de velocidad de prueba
RAM	*Random Access Memory*	Memoria de acceso aleatorio
RX	Receptor	
TCP	*Transmission Control Protocol*	Protocolo de control de transmisión
TX	Transmisor	
UDP	*User Datagram Protocol*	Protocolo de datagrama de usuario

RESUMEN

Existen en la literatura diversas herramientas que buscan estimar el ancho de banda disponible de extremo a extremo. Estas herramientas basan sus cálculos en las marcaciones de tiempo cuando los paquetes de prueba usados por las herramientas, llegan al receptor de la medición. Dado que esta marcación se realiza a nivel de software, existen diferentes fuentes de error principalmente asociadas a variaciones generadas por otros procesos que toman control del sistema operativo.

El propósito de esta investigación es diseñar un mecanismo que permita realizar el marcado de tiempo de los paquetes a nivel de hardware utilizando una tecnología denominada NetFPGA. Esta plataforma permite modificar su comportamiento por los arreglos lógicos programables que esta posee e interactuar con el software de estimación que para el caso de este proyecto es la herramienta TRACEBAND.

Como resultado de esta investigación, se plantean módulos para realizar el marcado de tiempo o *timestamp* utilizando la estructura HATS. Se diseñó un módulo de identificación de los paquetes de prueba usando los identificadores de tipo de protocolo y puerto destino. El almacenamiento de los *timestamp* se realiza en la RAM de la NetFPGA para luego ser leídos desde TRACEBAND. Adicionalmente, se describe la forma en que la NetFPGA y TRACEBAND deben comunicarse a través del módulo REGISTER IO de la NetFPGA y las llamadas IOCTL con sus funciones *readreg* y *writereg* en TRACENBAND.

ABSTRACT

There are various tools in the literature seeking to estimate the available bandwidth from end to end. These tools base their calculations on the marks of time when the test packets used by tools, arrive at the receiver measurement. Since this marking is done at the software level, there are several sources of error associated with variations mainly generated by other processes that take control of the operating system.

The purpose of this research is to design a mechanism for performing marking time of the packets at the hardware level using a technology called NetFPGA. This platform allows you to modify your behavior programmable logic arrays that this has and interact with the software estimation, this project is the TRACEBAND tool.

As a result of this investigation, they posed modules for marking time or timestamp using the HATS structure. An identification module test packets using identifiers protocol type and destination port was designed. Storing the timestamp is made in the RAM NetFPGA then be read from TRACEBAND. In addition, the way the NetFPGA and TRACEBAND must communicate through the REGISTER IO module NetFPGA and IOCTL calls with READREG and WRITEREG in TRACENBAND functions described.

Keywords: Bandwidth available, NetFPGA, Network, Timestamp

INTRODUCCIÓN

En la actualidad las diversas aplicaciones o servicios que se encuentran disponibles en la web llaman la atención de nuevos clientes que esperan que estos servicios no se vean afectados por las limitaciones de velocidad y capacidad en los enlaces de la red, así como el crecimiento continuo y complejidad de las redes, tanto en la configuración de las rutas como en la distribución del tráfico, hacen que la calidad de conexión sea difícil de predecir. Generando un interés en los investigadores en el área de las telecomunicaciones para el desarrollo de herramientas que permitan la medición de parámetros en la conectividad entre dos nodos terminales.

Para determinar el rendimiento de una conexión es necesario conocer parámetros como, el ancho de banda, la latencia, y el volumen de tráfico entre otros. La medición del ancho de banda es de relevante importancia, puesto que el ancho de banda es finito, no es gratuito, y su demanda sigue creciendo.

Existen tres métricas asociadas al ancho de banda la capacidad, el ancho de banda disponible y la capacidad de transferencia a granel. En este caso se enfocará el estudio a la estimación de ancho de banda disponible, este parámetro permite medir la capacidad no utilizada del canal, lo cual podría usarse para mejorar la utilización del mismo en servicios como descarga de video y audio.

Se han desarrollado diferentes herramientas para estimar el ancho de banda disponible y algunas de estas presentan limitaciones en cuanto la precisión, sobrecarga en la red generada por el envío de los paquetes de prueba, y en el tiempo de estimación que son largos para utilizar estas herramientas en tiempo real.

En este trabajo la medición de ancho de banda disponible se realiza mediante la integración de la herramienta TRACEBAND y la NetFPGA. La primera se encarga de realizar el cálculo del ancho de banda disponible basado en la dispersión del paquete en una ruta determinada y la segunda de realizar la marca de tiempo en el momento en que llega el paquete al receptor. Además se desarrolla un módulo de comunicación entre la NetFPGA y TRACEBAND para la transferencia de los datos de tiempo registrados.

1. PLANTEAMIENTO DEL PROBLEMA

En los últimos años se ha despertado un gran interés en los investigadores por el desarrollo de técnicas y herramientas de medida de diversas métricas sobre las prestaciones de la red de extremo a extremo. En particular, por las herramientas que permiten estimar el ancho de banda disponible (*available bandwidth ABW*) debido a la gran importancia en las diferentes aplicaciones como *la descarga* de video y audio tanto para operadores de redes como para usuarios finales. Por ejemplo, las herramientas de administración podrían monitorear con precisión la utilización de un enlace; los proveedores de servicio de internet podrían monitorear y verificar niveles de calidad de servicio; los protocolos de transporte podrían determinar la mejor tasa de transmisión según sea la cantidad de ancho de banda disponible en la red; los sistemas de detección de intrusos pudieran generar alertas basadas en un aumento inesperado en la utilización de la red.

Estas y otras aplicaciones requieren de una estimación del ancho de banda disponible de extremo a extremo, dado que no se tiene control sobre los nodos intermedios a través de los cuales se establece el canal de comunicación.

Se han desarrollado 16 herramientas de estimación de ancho de banda disponible, que de acuerdo a los trabajos de Guerrero & Labrador **(2006)**, en el que se comparan las herramientas *Pathload, IGI* y *Spruce* a través de los indicadores de evaluación: Error de estimación, tiempo de estimación, sobrecarga y fiabilidad, realizando 11760 experimentos, se concluye que *Pathload* presenta menor error de estimación, IGI tiene un tiempo de estimación más pequeño y *Spruce* presenta la menor sobrecarga en el cuello de botella del enlace de extremo a extremo.

Traceband, es una herramienta desarrollada por Guerrero y Labrador **(2009)** que basa la estimación en el modelo PGM, la cual al realizar estudios comparativos con *Pathload* y *Spruce,* presenta mejor rendimiento en cuanto a la baja sobrecarga similar a *Spruce* y presenta una precisión en la medición similar a la que presenta *Pathload*.

De lo anterior se puede decir que las herramientas requieren mejorar la precisión y fiabilidad en la estimación, usar un menor tiempo de medición, tener una baja sobrecarga en el canal, para poder ser utilizada en aplicaciones reales.

Para mejorar el rendimiento de la herramienta se debe tener en cuenta que estás han sido desarrolladas en la capa de aplicación del modelo OSI. El funcionamiento de manera general, implica que cada par o tren de paquetes se envía a través de la red al receptor donde la aplicación debe realizar una marca de tiempo a cada paquete en el momento de su llegada. Con esta marca de tiempo se puede estimar la dispersión entre paquetes que corresponde el retardo que sufre el paquete al interactuar con el tráfico de la red en los nodos intermedios. Esta dispersión se utiliza para la estimación actual de ancho de banda de acuerdo al modelo que utiliza cada herramienta.

En el proceso de generación de paquetes, los paquetes de prueba se envían con una dispersión inicial predeterminada, luego los paquetes pasan a la cola de los procesos del kernel como se muestra en la Figura 1 del equipo emisor, donde sufre un retardo antes de ser colocado en el canal de comunicación. Una vez llega el paquete al receptor, este pasa por los procesos internos del kernel antes de estar disponible para la aplicación, estos procesos generan retardos que influyen en el registro de tiempo de llegada, y la estimación que se realiza desde la aplicación depende de la dispersión que sufre el paquete en todo el recorrido.

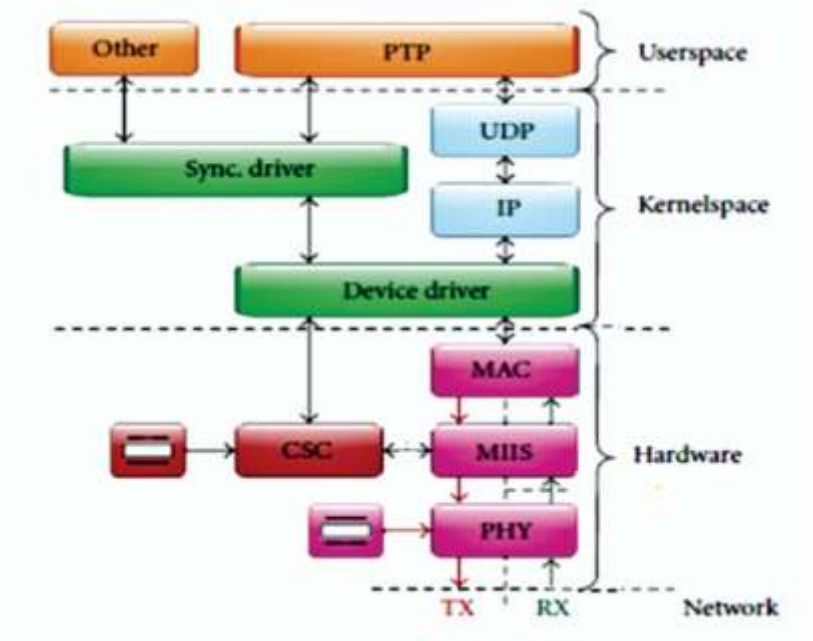
La conmutación entre procesos del Kernel impide que los tiempos de envío y recepción de los paquetes de prueba sean los dispuestos por la herramienta de estimación induciendo a errores difíciles de controlar que influyen en el error de estimación de la herramienta.

Para mejorar la calidad de la estimación del ancho de banda disponible es necesario que la herramienta tienda además a disminuir los errores de la estimación generados por los múltiples errores causados por los procesos del sistema operativo

Como se mencionó anteriormente la estimación se realiza en base al tiempo de retardo de sufre un paquete desde que es enviado hasta que se recibe por la aplicación, se pretende hacer que el proceso de la marcación del tiempo en la llegada de los paquetes se realice a nivel de hardware, en una tarjeta de red convencional no podría modificarse el hardware para que realice la función de marcación, por lo tanto, se hace necesario utilizar una plataforma como NetFPGA, que permita desde el hardware programar la marcación de los paquetes en el receptor antes que los paquetes pasen a la aplicación. En el presente trabajo se plantea la pregunta de investigación ¿Es posible mejorar la precisión de la Estimación de Ancho de Banda Disponible sí el marcado de tiempo en la recepción

de los paquetes se realiza desde la NetFPGA, con una baja sobrecarga en la red y un tiempo de intrusión mínimo?

Figura 1. Ruta de los paquetes de datos



Fuente **(Loschmidt, Exel, & Gadere, 2011)**

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Diseñar un esquema de marcación de tiempos de llegada de paquetes de prueba de un estimador de ancho de banda disponible a través de la NetFPGA con el fin de reducir su error de estimación.

2.2 OBJETIVOS ESPECIFICOS

- Realizar un estado del arte de desarrollos en NetFPGA orientados al registro de los tiempos de recepción en el paquetes
- Diseñar un módulo en la NetFPGA que permita la marcación de tiempo de llega de paquetes en el hardware para la estimación de ancho de banda disponible
- Ajustar un estimador de ancho de banda disponible para incluir la recepción de la información sobre marcación de tiempos de la NetFPGA
- Diseñar un esquema de comunicación de la NetFPGA con el estimador de ancho de banda disponible para la marcación de tiempos de llegada de paquetes

3. MARCO REFERENCIAL

En el proyecto “Diseño de un esquema de marcación de tiempos de llegada de paquetes de prueba de un estimador de ancho de banda disponible a través de la NetFPGA con el fin de reducir su error de estimación” se hizo necesario el estudio de los conceptos generales relacionados con la estimación de ancho de banda disponible en una red de datos, y el análisis del estado del arte de las herramientas desarrolladas para medir el ancho de banda disponible y el marcado de tiempo de los paquetes en la NetFPGA

3.1 MARCO CONCEPTUAL

En esta sección se abordara los conceptos relacionados con redes IP, Ancho de Banda, Mediciones en internet y los modelos de estimación de ancho de banda disponible.

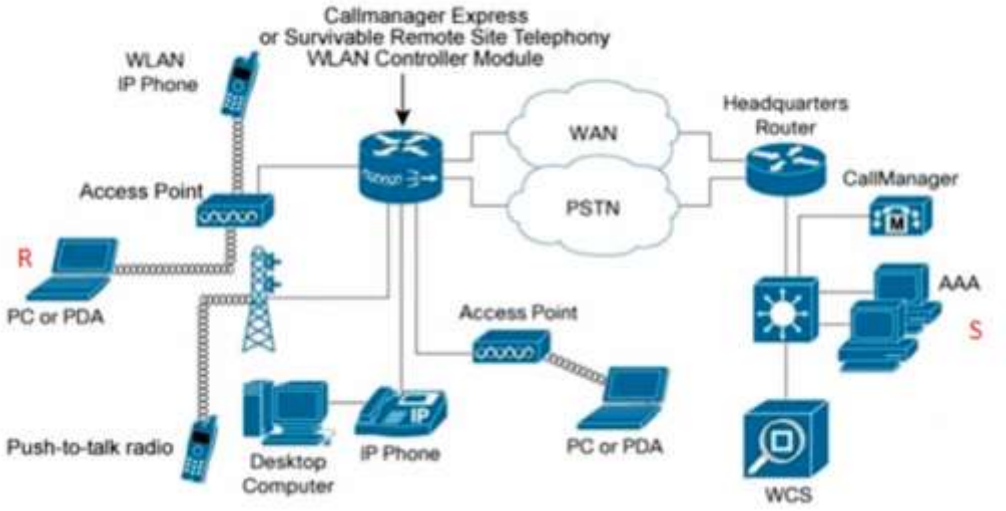
3.1.1 Redes IP

Una red IP está constituida esencialmente por equipos terminales (*host*) como laptops, servidores (web-host) y routers (envía la información desde la fuente al destino) (Tanenbaum, 2003). En entre estas entidades hay un conjunto de enlaces de conexión que operan a diferentes velocidades de bits. Por ejemplo en la Figura 2 se muestra una red con dispositivos finales como laptops, equipos residenciales, módems, routers.

Todos los datos que se envían desde emisor al receptor son encapsulados en paquetes. Cada paquete contiene una dirección origen y una dirección destino

además de la información que se desea transmitir. Sin embargo, en Internet el proceso de encapsulación se realiza en varios pasos, donde cada paso agrega una cabecera adicional al paquete, con información acerca de la transmisión entre el emisor y el receptor, para el paquete existente. (Ver Figura 3.)

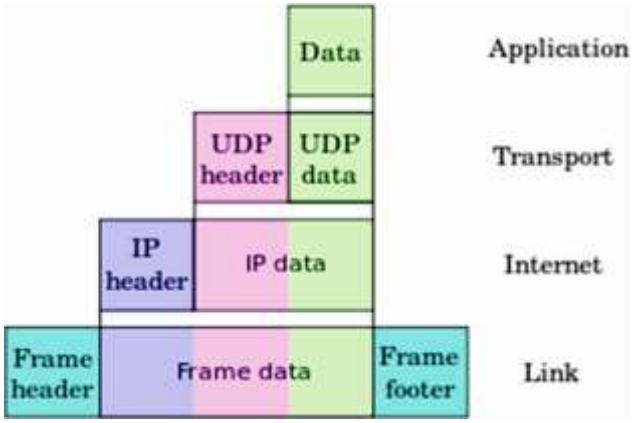
Figura 2. Red heterogénea



Fuente: (CONASA Comunicaciones Navarra S.A, s.f.)

El primer paso consiste en añadir una cabecera de transporte para los datos de la capa de aplicación. La capa de transporte contiene, entre otra información los datos para que los procesos de comunicación sobre el origen y el equipo de destino se pueda realizar. A continuación, se añade una cabecera IP al paquete. Esto corresponde a la capa de red. La cabecera IP contiene información sobre la fuente y los hosts de destino, siendo esta la información necesaria para que los datos pueden ser enviados correctamente a través de la red. La capa inferior, la capa de enlace, añade una cabecera de enlace. Este encabezado contiene información sobre la manera de enviar el paquete en un vínculo físico real entre dos ordenadores. Dependiendo de las características del enlace la cabecera de capa de enlace puede variar.

Figura 3. Encabezado de datos en el modelo TCP/IP



Fuente: (WIKIPEDIA a, 2013)

Cuando un paquete atraviesa una red IP de S a R como se muestra en la Figura 2 El paquete se encontrará con varios routers en el camino. Un router es una máquina que examina la dirección de destino del paquete IP y, a continuación reenvía el paquete al siguiente router, un salto más cerca del destino. Esto se repite hasta que el paquete IP ha atravesado todo el camino hasta R.

Los paquetes de diferentes enlaces pueden alcanzar el router al mismo tiempo. En tales casos los paquetes se encolan en la cola de entrada. Una disciplina de cola para un enrutador común es la disciplina de primero en entrar, primero en salir (FIFO). Es decir, un paquete P debe esperar a que todos los paquetes delante de él salgan de la cola.

Dado que un enrutador puede descartar los paquetes IP cuando una cola del enrutador se llena, el remitente no puede obtener garantías de que un paquete llegará a su destino. Este tipo de redes se denominan redes de mejor esfuerzo. La red hace todo lo posible para enviar todos los paquetes, pero no puede prometer nada.

Dado al servicio de mejor esfuerzo, los anfitriones finales deben hacer frente a la pérdida de paquetes. Esto se hace mediante la retransmisión de paquetes perdidos. La capa IP no proporciona mecanismos de retransmisión, en su lugar un conjunto de protocolos de transporte se han desarrollado (la capa de transporte en la Figura 3). Un protocolo de transporte ampliamente utilizado es TCP, que crea una conexión entre el emisor y el receptor. TCP se ocupa de la pérdida de paquetes, los paquetes corruptos, los paquetes fuera de orden y así sucesivamente.

UDP es otro protocolo de transporte de Internet. UDP sólo proporciona una interfaz entre la capa de red y el usuario destino y en el proceso de envío y recepción de ordenador. Es decir, UDP no proporciona funciones similares a TCP tales como la retransmisión de paquetes y paquetes fuera de orden.

3.1.2 Estimación de parámetros de rendimiento sobre internet

Una vez se tiene compresión sobre el funcionamiento global de las redes IP, se analizan los tipos de mediciones que tienen lugar en estas redes.

Al evaluar el rendimiento de una red, se han propuesto una serie de parámetros como la perdida de paquetes, tiempo promedio de transmisión de un paquete, capacidad del enlace entre otros, para ello se han desarrollado una variedad de aplicaciones basadas en los métodos de medición pasivos o activos las cuales se describen a continuación.

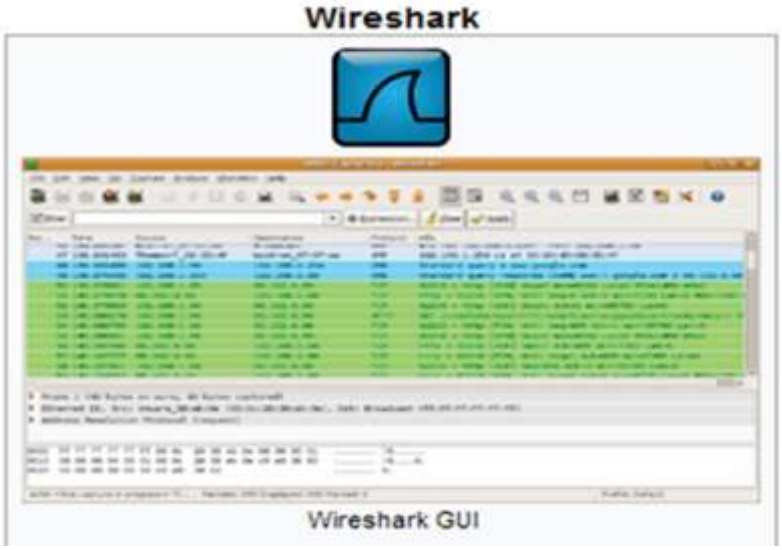
3.1.2.1 Mediciones Pasivas.

Los métodos y herramientas de medición pasivos actúan como observadores dentro de una red y por lo general no interfieran con el resto del tráfico. (Hu & Steenkiste, Evaluation and Characterization of Available Bandwidth Probing Techniques, 2003) Estos métodos a menudo también requieren control y privilegios de administración de la infraestructura de red subyacente (es decir, el acceso a los routers y servidores de la red)

Un ejemplo de una herramienta de medición pasiva es WIRESHARK. Wireshark es un software que "entiende" la estructura de los diferentes protocolos de red. Por lo tanto, es capaz de mostrar la encapsulación y los campos junto con sus significados de diferentes paquetes especificados por diferentes protocolos de red. Wireshark utiliza un formato *.pcap para capturar paquetes, por lo que sólo puede capturar los paquetes en los tipos de redes que pcap apoya. (WIKIPEDIA b, 2013)

Los datos pueden ser capturados de una conexión de red activa o leer desde un archivo que registra los paquetes ya capturados. Además, los datos pueden ser leídos desde un número de tipos de red, incluyendo Ethernet , IEEE 802.11 , PPP , y de bucle de retorno.

Figura 4. Interfaz de Wireshark



Fuente (WIKIPEDIA b, 2013)

Los métodos de medición de pasivos son de gran alcance en su contexto, pero por lo general solo puede ser utilizada por personas que tienen privilegios de administrador de red. Por otra parte, los métodos pasivos no dan pleno conocimiento de lo que sucede en una ruta de extremo a extremo, entre dos usuarios finales. En su lugar, estos métodos ofrecen una fotografía del estado de la red en un momento dado y el enlace dentro de la red como se muestra en la Figura 4.

3.1.2.2 Mediciones Activas

Los métodos activos inyectan tráfico llamado tráfico de sondeo en la red desde una fuente de tráfico y se mide la influencia que este tráfico tiene sobre la red en un receptor de tráfico sonda. Por lo tanto, los métodos de medición activos afectan el tráfico de la red en sí, al contrario del método de medición pasiva. Obsérvese que los métodos de medición activos sólo tienen acceso a dos ordenadores, uno la fuente de tráfico y dos el receptor de tráfico. Tales métodos son llamados métodos de medición de extremo a extremo.

Las herramientas de sondeo activo se caracterizan por generar tráfico y basar sus cálculos en las mediciones efectuadas sobre el tráfico que generan, permiten realizar mediciones en tiempo real, y su desventaja se presenta en la agregación de tráfico a la red el cual puede generar congestión sobre la misma.

El sondeo es el elemento básico de todos los métodos de medición de activos, esto se realiza mediante la inyección de un conjunto de paquetes de prueba con una separación o dispersión predefinida. A continuación, se describen dos esquemas básicos de sondeo: el par de paquetes y el tren de paquetes de sondeo. El esquema de sondeo más básico es dividir los paquetes de sondeo de dos en dos, cada par tiene una dispersión predefinida que corresponde a la tasa de prueba. En lugar de utilizar pares de paquetes de sondeo muchos métodos utilizan los trenes de paquetes de prueba. La dispersión entre los paquetes de sondeo en el interior del tren puede ser, por ejemplo igual o exponencialmente decreciente a la tasa de prueba.

Hay una diferencia fundamental entre el uso de pares de paquetes y el tren de paquetes de sondeo. En los sistemas de sondeo de tren de paquetes los retrasos de varios de paquetes de las sondas pueden ser dependientes entre sí, lo cual no es el caso en los sistemas de sondeo de pares de paquetes.

3.1.3 Ancho de banda

En una red para determinar el rendimiento de una conexión es necesario conocer parámetros como, el ancho de banda, la latencia, y el volumen de tráfico entre otros. (Obara, Koseki, & Selin, 2012)

El ancho de banda es una medida de velocidad que se define como la cantidad de bits transmitidos en un canal de comunicaciones por unidad de tiempo. Hay tres métricas asociadas con el ancho de banda: capacidad, capacidad de transferencia a granel y ancho de banda disponible. La capacidad de un enlace es el ancho de banda máximo o la máxima cantidad de bits que se pueden transmitir al enlace por unidad de tiempo. Por el contrario, el ancho de banda disponible (ABW) de extremo a extremo (end-to-end) es la mínima capacidad no utilizada de todos los enlaces en una ruta de comunicación entre dos nodos extremos como se observa en la **Figura 5**.

En una definición más estricta, la capacidad de un enlace de extremo a extremo, en una ruta de H enlaces es definida como la mínima capacidad entre dos nodos.

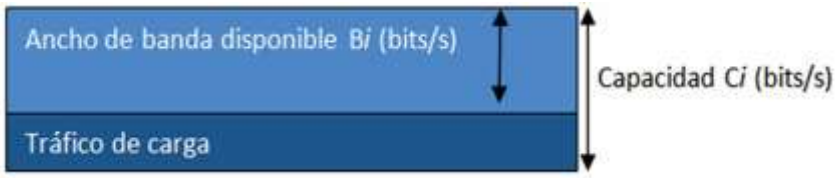
Esto es:

Ecuación 1 Capacidad entre dos nodos

$$C = \min_{i=1...H} C_i$$

Donde C es la capacidad del enlace i.

Figura 5. Capacidad de un canal de comunicaciones



Y el ancho de banda disponible definida por Jain, **(Jain M. &., 2002)** se basa en la observación de que la transmisión de bits (un bit puede ser 0 o 1) en un enlace es un proceso discreto. O bien el enlace está ocupado enviando un bit o el enlace está inactivo.

La utilización promedio de un enlace i durante un periodo de tiempo $(\boldsymbol{t, t+\tau})$, está dada por la Ecuación 2:

Ecuación 2. Utilización del enlace i durante el tiempo τ

$$u_i(t, t+\tau) = \frac{1}{\tau}\int_{t}^{t+\tau} u_i(t)$$

Conociendo la utilización $\boldsymbol{u_i(t, t+\tau)}$, el ancho de banda disponible $\boldsymbol{ab\_disp}$ en el enlace $\boldsymbol{i}$ estaría dado por la Ecuación 3 :

Ecuación 3 Ancho de banda disponible de un enlace $\boldsymbol{i}$

$$ab\_disp_i(t, t+\tau) = ab\_disp_i^{\tau}(t) = C_i[1 - u(t, t+\tau)]$$

Donde $\boldsymbol{C_i}$ es la capacidad del enlace $\boldsymbol{i}$. De ésta manera, el ancho de banda disponible de extremo a extremo en una ruta compuesta por H enlaces, está definido por la Ecuación 4:

Ecuación 4 Ancho de banda disponible de extremo a extremo a lo largo de H enlaces.

$$\boldsymbol{ab\_disp(t, t+\tau) = ab\_disp^{\tau}(t) = \min_{i=1\ldots H}\{ab\_disp_i(t, t+\tau)\}}$$

La utilización de un enlace depende del tráfico cruzado (*cross Traffic)*. *Cross Traffic* es el tráfico regular que fluye a través de la red (es decir, todo el tráfico, sin incluir el tráfico de la sonda utilizados en la medición). Muchos parámetros controlan el comportamiento del *Cross Traffic*, algunos se describen a continuación:

La intensidad de distribución. La dispersión entre los paquetes de tráfico cruzado puede ser aproximada a una función de probabilidad, tal como rectangular (muy suave), Poisson o Pareto (por ráfagas). Dependiendo de la fuente de tráfico cruzado la intensidad de distribución puede variar.

Distribución de tamaño de paquetes. Dependiendo de la fuente de tráfico cruzado el tamaño de los paquetes variará. Por ejemplo, una transferencia de un archivo de gran tamaño utiliza paquetes grandes mientras que los paquetes de aplicación de voz sobre IP utilizan paquetes pequeños.

Longitud de flujo. El flujo de tráfico cruzado entre un emisor y un receptor específico puede variar de los flujos de tráfico web cortos (medido en segundos) hasta largos flujos correspondientes a una descarga de archivos (medido en horas). La agregación de tráfico cruzado en los enlaces dentro de la red central está constituida por varios flujos de tráfico. Un mayor número de flujos de tráfico cruzado da un

mayor grado de agregación. Por lo general, un alto grado de agregación da una distribución de la intensidad de tráfico cruzado general más suave.

3.1.3.1 Medición de ancho de banda disponible (*Available Bandwidth Estimations ABW*).

Al estimar el ancho de banda disponible se realiza a través de mediciones activas además requiere de operaciones estadísticas en los valores de dispersión. Por tratarse de mediciones activas requiere aplicar en primer lugar uno de los dos tipos de sondeo, a través de pares de paquetes o trenes de paquetes de prueba.

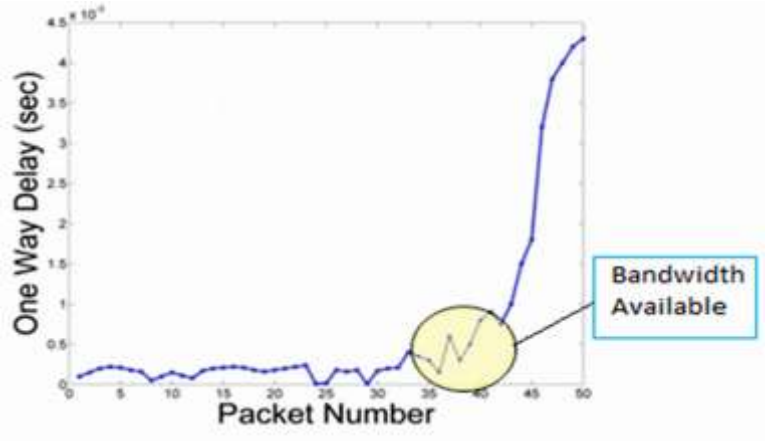
La inyección del conjunto de paquetes de prueba se realiza con una separación o dispersión predefinida. La dispersión es inversamente proporcional a la velocidad de envió de los paquetes (medida en bits por segundo). Cuando los paquetes de sondeo atraviesan la red, la dispersión predefinida puede cambiar. Esto se debe al tráfico que compite por el canal causando el llamado efecto de espaciamiento entre los paquetes en el enlace de menor capacidad o cuello de botella.

Las principales modelos de estimación de ancho de banda disponible son modelo de velocidad de prueba (PRM del inglés *Probe Rate Model*) y modelo brecha de prueba (PGM del inglés *Probe Gap Model*) (Strauss, Katabi, & Kaashoek, 2003).

3.1.3.2 Modelo de Velocidad de Paquetes PRM

El modelo PRM es también conocido como congestión autoinducida en un solo camino. El funcionamiento radica en inyectar en una red un conjunto de paquetes de sondeo (por ejemplo, en pares o en trenes) con una tasa de sonda inicial predefinida. Si la velocidad de la sonda inicial es mayor que el ancho de banda disponible, los paquetes de sondeo se pondrán en cola uno después del otro en el cuello de botella y por lo tanto causar congestión. En tal caso, la dispersión media entre los paquetes de sondeo se incrementará, lo que equivale a una disminución en la tasa de sonda recibido. Sin embargo, si el tipo de sonda inicial es igual a la tasa de la sonda recibida se supone que los paquetes no tienen que pasar por cola y por lo tanto la ruta de extremo a extremo no está congestionada. Es decir, la tasa de sonda inicial es menor que el ancho de banda disponible. (Ver Figura 6)

Figura 6. Modelo PRM



Fuente (Guerrero & Labrador, 2009)

3.1.3.3 Modelo de Separación de Paquetes PGM

Cuando se usa el modelo de separación de paquetes la topología de la red debe ser conocida hasta cierto punto, por lo general hasta el enlace de cuello de botella (es decir, el enlace que determina el ancho de banda disponible) debe ser conocida. Además, sólo puede haber un enlace cuello de botella entre el emisor y el receptor de prueba.

El modelo PGM resuelve la Ecuación 5

Ecuación 5 Ancho de banda disponible en PGM

$$ab\_disp = Cbl - Ri\left(\frac{Cbl}{Ro} - 1\right)$$

Donde Cbl es la capacidad del cuello de botella, que es conocida, R_i es la velocidad inicial de prueba, *Ro* es la velocidad medida. Para obtener los valores R_i y R_o, la ruta de extremo a extremo se sondea utilizando diferentes tipos de pruebas iníciales. Para cada tren de prueba o par se puede hacer una estimación del ancho de banda disponible.

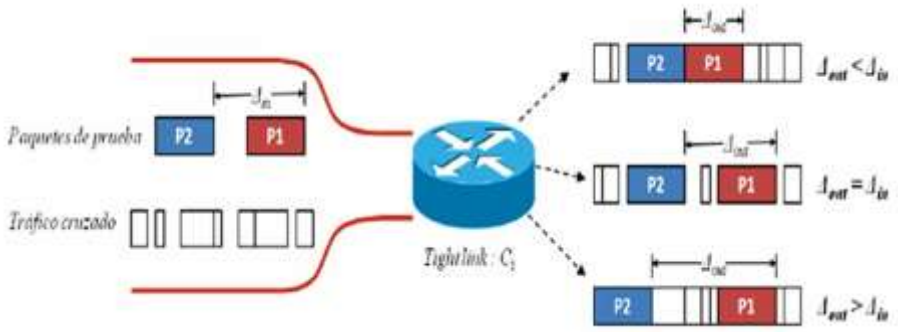
En términos generales, el comportamiento de un par de paquetes cuando salen del *tight link* se muestra en la Figura 7.

Si dos paquetes consecutivos son enviados hacia un enlace, ellos llegan al nodo con cierta separación de tiempo entre ellos (Δin). Después de interactuar en la cola del *tight link* con el tráfico cruzado que viene de diferentes fuentes, el par de paquetes de prueba saldrá del enrutador con una nueva separación de tiempo (Δout). La diferencia entre ellos $\Delta out - \Delta in$ es la dispersión del par de paquetes.

La dispersión del par de paquetes puede ser negativa, positiva o igual a cero. Tal como se ve en la Figura 7, un valor negativo ($\Delta out < \Delta in$) ocurre cuando el primer paquete encuentra paquetes de tráfico cruzado en la cola y es seguido por el segundo paquete.

Un valor positivo ($\Delta out > \Delta in$) ocurre cuando se insertan paquetes de tráfico cruzado entre el par de paquetes de prueba en la cola. Finalmente, un valor de cero ($\Delta out = \Delta in$) ocurre cuando el enlace no tiene suficiente tráfico cruzado para afectar la separación inicial de los paquetes.

Figura 7. Interacción de un par de paquetes de prueba en el Tight link



Fuente: (Guerrero C. , 2013)

3.1.4 Plataforma NetFPGA

La plataforma NetFPGA (Ver Figura 8) desarrollada por la Universidad de Stanford, es un dispositivo de bajo costo. Permite a los investigadores e instructores construir prototipos de sistemas de redes de alta velocidad, acelerados por hardware. La

plataforma puede ser utilizada en el aula para enseñar a los estudiantes como construir conmutadores Ethernet y routers IP (del inglés *Prototcol Internet)* además puede ser utilizada por los investigadores para crear prototipos de servicios avanzados para las redes de próxima generación. (Digital Design Engenier's Source, 2000)

Figura 8. Tarjeta NetFPGA

Fuente **(Digital Design Engenier's Source, 2000)**

3.1.4.1 Descripción de la plataforma NetFPGA

La tarjeta NetFPGA 1G mostrada en la Figura 8 incluye una FPGA Xilinx Virtex-II Pro 50, cuatro puertos Gigabit Ethernet de 1 Gbps de velocidad, una memoria SRAM de 4.5MBytes, una memoria DD2R2 DRAM de 64 MBytes, un conector estándar PCI y dos conectores SATA. Tiene una interfaz PCI estándar que le permite conectarse a un PC de escritorio o servidor.

Una FPGA (del inglés *Field Programmable Gate Array*) es un Arreglo de compuertas programables en campo, el cual, es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada 'in situ' mediante un lenguaje de descripción especializado. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip (Quintans, Lago, Menéndez, & Mandado, 2006).

La NetFPGA proporciona un camino de datos acelerados por hardware directamente conectada a los cuatro puertos Gigabit Ethernet y con múltiples bancos de memoria local instalados en la tarjeta. Uno de los lenguajes de programación de la tarjeta NetFPGA es Verilog, es cual es un lenguaje de descripción de hardware (*Hardware Description Languaje*, HDL) utilizado para describir sistemas digitales. Verilog permite la descripción de los sistemas usando

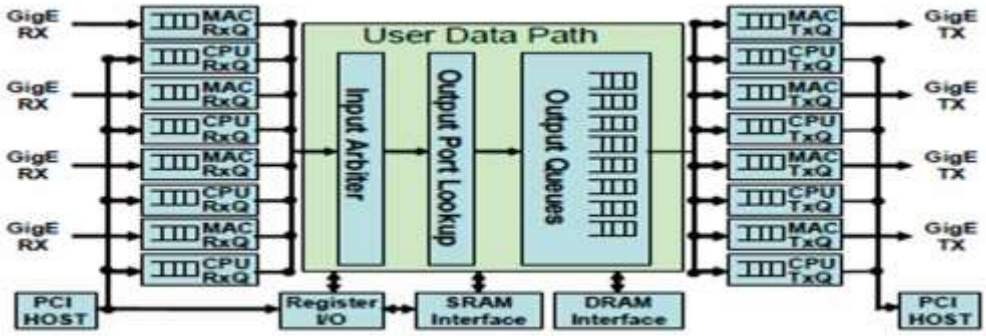
vistas de comportamiento o estructurales, no físicas. Y soporta como niveles de abstracción el nivel de puerta (GATE), nivel de transferencia de registros RTL y el nivel de comportamiento.

Para el desarrollo de aplicaciones con la NETFPGA se cuenta con diseños de referencia que permiten la configuración de la misma como tarjeta de red (NIC), Router IPv4, entre otras aplicaciones que se encuentran en www.netfpga.org. Las ventajas de utilizar la NetFPGA son la descarga de un procesador de ciertas tareas como el *timestamping*, el procesador principal ubicado en el PC puede utilizar DMA (Direct Memory Access) para leer/escribir registros y memorias de la tarjeta NetFPGA y NetFPGA provee un camino de datos acelerado por Hardware a través de los cuatro puertos GigaEthernet. La estructura de directorios de la NetFPGA se divide en tres, bin, lib, y proyect. El directorio bin contiene las secuencias de comandos para ejecutar las simulaciones y configuraciones del entorno. El directorio lib contiene los módulos comunes que son necesarios para la simulación, síntesis, y de diseño, en los lenguajes de programación C, Verilog, Makefiles y PERL; y en proyect se encuentran los proyectos de usuario y los diseños de referencia bajo los archivos src, synth, sw y include **(NetFPGA, 2013)**.

3.1.4.2 Diseño de referencia de la NetFPGA de la plataforma NetFPGA

Uno de los diseños de referencia de la NetFPGA es la NIC o tarjeta de red, para lo cual se realiza la división del hardware a través de módulos como se muestra en la Figura 9. Los módulos que conforman la NIC son el modulo cola de RX (recepción) de MAC *(Medium Access Control*), módulo de Cola de Rx de CPU *(Central Processing Unit*), modulo Cola de TX (Transmisión) de MAC, modulo Cola de TX de CPU, modulo Arbitro de entrada (input Arbiter), módulo busqueda de puerto de salida(Output Port Lookup), módulo de salida de colas (output Queues), modulo interface PCI (*Peripheral Component Interconnect*) Host, módulo de registros I/O, modulo interface SRAM y el módulo de interface DRAM.

Figura 9. Diseño de referencia NIC de la NetFPGA



Fuente: (Covington, Gibb, McKeown, & Lockwood, 2009)

La primera etapa del recorrido, consta de 8 colas que son llamadas las colas RX. Estas colas reciben paquetes desde los puertos IO tales como los puertos Ethernet y el puerto de DMA y proporcionan una interfaz unificada (AXI) para el resto del sistema. Este diseño tiene 4 colas Rx de 1G Ethernet y 1 cola CPU DMA sobre PCIe (OPED RX). Está se maneja con las 4 colas virtuales DMA. Los paquetes que llegan a la cola RX de CPU DMA virtual son paquetes que han sido enviados por el software de interfaz.

En el *USER DATA PATH (*camino de datos principal), el primer módulo que un paquete pasa a través de él, es el árbitro de entrada. El árbitro decide de cual cola Rx recibe el paquete, saca el paquete de esa cola Rx y se lo entrega al siguiente módulo (Output port lookup) para dar servicio a otro. El módulo *Output port lookup* se encarga de decidir por cual puerto el paquete sale. Después se toma la decisión, el paquete se entrega, al módulo de Output Queues el cual almacena el paquete en las colas de salida correspondientes al puerto de salida hasta que la cola TX está lista para aceptar el paquete para la transmisión. Para cada uno de estos módulos, hay un conjunto de registros que mantienen la información sobre estado de acceso y señales de control de ajuste para el módulo.

3.1.4.3 Análisis de la plataforma NetFPGA

La NetFPGA es una plataforma tipo hardware, que permite la investigación de nuevos protocolos de comunicaciones en las redes de datos, al permitir la programación de estos en la FPGA Virtex II pro de Xillins. Además cuenta con cuatro

memorias, dos de tipo SRAM y dos SDRAM que permiten el almacenamiento temporal de los paquetes para su posterior procesamiento. En el lado izquierdo de la plataforma como lo muestra la Figura 10, hay un transceptor de cuatro puertos de la capa física (PHY), que permite a la plataforma enviar y recibir paquetes a través de cuatro cables estándar Ethernet de par trenzado. En el lado derecho de la tarjeta, contiene en la plataforma dos ATA (SATA) conectores Seriales que permiten múltiples NetFPGAs dentro de un sistema de intercambio de datos a alta velocidad sin necesidad de utilizar el bus PCI. El reloj de núcleo NetFPGA funciona a 125 MHz, lo que significa que cada ciclo de reloj dura 8ns. La FPGA maneja directamente toda la conmutación en la ruta de datos, el enrutamiento y las operaciones de procesamiento de tramas Ethernet y paquetes de Internet, dejando el software para manejar las funciones de ruta de control (Zhou, Cong, Lu, Deng, & Li, 2010). La NetFPGA encaja en un PC host a través de una ranura PCI.

Figura 10. Tarjeta NetFPGA

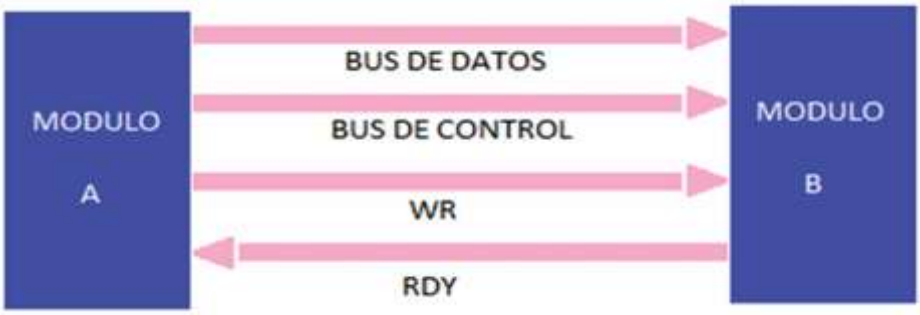
Fuente: (NetFPGA, 2013)

El funcionamiento de la tarjeta NETFPGA es a través de módulos como se muestra en la Figura 11, en el User Data Path se pueden insertar módulos al programar nuevas funciones en lenguaje veriloj, para lo cual se debe tener en cuenta la forma de comunicarse entre módulos. La comunicación entre módulos es a través de los buses de Datos, control, habilitación de escritura (WR) y lectura (RDY) como se muestra en la Figura 11. El bus de datos tiene un tamaño de 64bits, el bus de control se forma por una trama de 1byte, mientras para la habilitación de la lectura y escritura se usa un bit. En el módulo A, el bus de datos, el de control y la habilitación de escritura son salidas y tiene como entrada el bit de habilitación de lectura, para

el módulo B el bus de datos, el de control, la habilitación de escritura son entradas y la salida es la habilitación de lectura.

Todos los módulo comparten una señal de sincronización, por lo que se dispone en una entrada de la señal de reloj (*clock*) y otra para el reset la cual reinicia todos los valores de los registros utilizados en la programación del módulo. Las salidas de un módulo que se conectan a las entradas de otro módulo se programan en el USER_DATA_PATH.

Figura 11. Interconexión entre módulos



En la Tabla 1 se muestra la interfaz de la ruta interna de datos entre los módulos desde la perspectiva del módulo de recepción. El paquete de datos e información de control se envían a través de los buses de datos y control, respectivamente, cuando la señal en WR se afirma. Es responsabilidad del envío de los datos al módulo sólo cuando se afirma la señal en RDY, y los nuevos datos e información de control se envían durante cada ciclo positivo de la señal de reloj.

Tabla 1 Señales en la ruta de datos entre módulos en la NETFPGA

Nombre de la señal	Dirección	Descripción
In_data [DATA_WIDTH:0]	Input	Input Data: usado para pasar a un módulo escribiendo las cabeceras de 64bits y los paquetes de datos
In_ctrl [CTRL_WIDTH:0]	Input	Input Control: utilizado para escribir en el módulo la información de control sobre el valor en el bus de datos

In_wr	Input	Input write: usado para indicar que los datos en los buses in_data e in_crtl fueron escritos en el bus
In_rdy	Output	Input Ready Se utiliza para indicar que el módulo está preparado para recibir nuevos datos

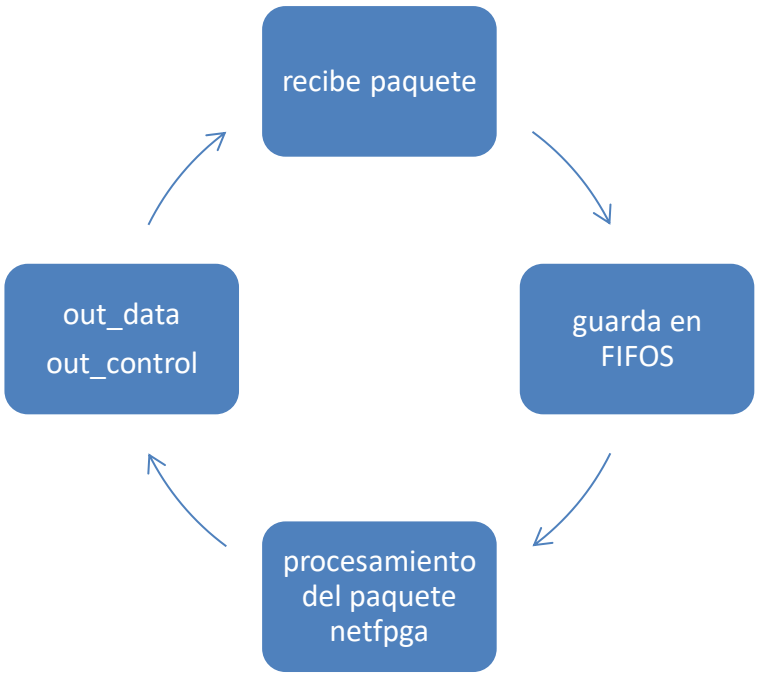
En este momento, una discusión de cómo los datos del paquete se manejan internamente en el pipeline de referencia es necesario. En el pipeline de referencia, cada paquete Ethernet se divide en paquetes de 64 bits y se transporta en todo el sistema a través del bus de datos in_data. Estos paquetes son procesados por cada módulo posterior en la ruta de datos antes de llegar a la salida como un paquete Ethernet completo en los puertos de salida Ethernet MAC. Además de dividir la salida en paquetes de 64bits internamente, la ruta de datos también es compatible con el uso de cabeceras internas. Estas cabeceras se identifican con valores distintos de cero en el bus in_crtl. El *Input Arbiter* anexa el primer encabezado con un valor asociado de 0xFF en el bus in_crtl. Esta cabecera especifica el puerto origen del paquete, el tamaño en bytes y palabras del paquete Ethernet y proporciona un campo para especificar después los puertos de salida para este paquete. Las cabeceras posteriores se añaden al paquete entre esta cabecera y los datos reales del paquete se identifican por un valor de 0x00 en el bus in_crtl. El valor de control para el último paquete interno del paquete Ethernet se establece en la ubicación del último byte válido en el paquete interno. A medida que cada paquete entra en un módulo, los datos y los valores de control se almacenan en el FIFO de entrada. El procesamiento del paquete de datos de 64bits en cada módulo se ilustra en la Figura 12, una vez el modulo está disponible para recibir el paquete, es guardado en la memoria o FIFOS, luego en el siguiente ciclo de reloj, se realiza del procesamiento que puede ser por ejemplo filtrado, timestamp, etc. Finalmente se genera el dato de salida y se guarda en out_data y out_ctrl.

3.1.5 Marca de tiempo en paquetes de datos (*TIMESTAMP*)

El estampado de fecha es la asociación de un conjunto de datos con un valor de tiempo. En este contexto, el tiempo también puede incluir la fecha. El ejemplo más cercano puede estar en el uso de una computadora personal (PC). Cada vez que se crea o guarda un documento, el documento es automáticamente asignado un valor de fecha y hora. Este valor puede ser utilizado para localizar los documentos creados en una cierta fecha (el lunes pasado), los documentos creados dentro de

un cierto período de tiempo (último semestre de 1998), y el orden en el que se ha creado un conjunto de documentos como los mensajes de correo electrónico en la bandeja de entrada.

Figura 12. Proceso del paquete en la NetFPGA



Las trazas de paquetes de red se utilizan para investigar una serie de preguntas de investigación en el área de Internet. El *Timestamp* a la llegada de paquetes permite la investigación para conservar la noción de tiempo en el enlace de la red. Más específicamente, las marcas de tiempo permiten la llegada de un paquete dentro de la traza que se correlaciona con el otro, para calcular los indicadores de enlace, como las cifras de utilización o rendimiento de las aplicaciones observadas en este enlace, como el flujo de paquetes TCP, retardo y jitter. Otra importante utilización del *timestamp* es en las mediciones multipunto, la capacidad de correlacionar eventos en un eslabón de la red en relación con las observaciones en un enlace diferente.

Las tradicionales implementaciones de SNTP (del inglés *Simple Network Time Protocol*) se basan generalmente en el marcado de tiempo en los paquetes entrantes y salientes del nivel de aplicación. Un sistema de sellado de tiempo de

bajo nivel debe ser aplicado tanto en el cliente y el servidor de tiempo. Básicamente, hay tres métodos de sellado de tiempo diferentes de bajo nivel. En primer lugar, se puede implementar en un hardware el sellado tiempo en, o cerca del controlador Ethernet. En segundo lugar, se puede implementar en software sobre una rutina de servicio de interrupción (ISR). Tal ISR debe estar conectado a la señal de petición de interrupción Ethernet y tiene una prioridad superior en el hardware. La tercera posibilidad es la aplicación de software de marcado de tiempo en el controlador de Ethernet, que es controlado por el RTOS (del inglés *Real-Time Operating System*). Este controlador se conecta a la señal de petición de interrupción Ethernet con una prioridad normal de hardware.

3.2 ESTADO DEL ARTE

La revisión del estado del arte se realizó desde dos enfoques el primero relacionado con las herramientas de estimación de ancho de banda y el segundo enfoque está relacionado a los proyectos desarrollados con la plataforma NetFPGA donde se utiliza el marcado de tiempo (*Timestamp*) en los paquetes en una red de datos.

A continuación se describen las herramientas de estimación de ancho de banda disponible que aplican los modelos *PGM y PRM;* y los resultados de los trabajos realizados por Aceto y, Botta, **(2011)** y en el trabajo desarrollado por Guerrero & Labrador **(2006)** donde se comparan herramientas Pathload, Assolo del modelo PRM y Spruce, Traceband del modelo *PGM* y sus conclusiones se describen en la sección 3.2.2

3.2.1 Herramientas de estimación de ancho de banda disponible.

Para estimar el ancho de banda disponible de extremo a extremo sobre una ruta de red, se han desarrollado diferentes aplicaciones a nivel de software.

Las herramientas encontradas en la literatura que aplican el modelo PRM descrito en la sección 3.1.3.2 son Pathload (Jain M. &., 2002), PathChirp (Vinay Ribeiro, 2003), YAZ (Sommers, Barford, & Willinger, 2006), Nestet (Jin & Tierney, 2003), TOPP (Melander, Bjorkman, & Gunniengberg, 2000), ASSOLO (Goldoni & Rossi,

2009), y FEAT (Wang & Cheng, 2006) , aquellas que aplican el modelo PGM son IGI/PTR (Hu & Steenkiste, 2002), Spruce (Strauss J. , Katabi, Kaashoek, & Prabhakar, 2003), ProbeGap (Sharma, 2003), Delphi (Lao, Dovrolis., & Sanadidi, 2006), IGMPS (Ali & Lepage, 2007), Abing (Navratil & Cottrell, 2003) y Traceband (Guerrero & Labrador, 2009).

Las características más relevantes de las herramientas mencionadas anteriormente se describen en Tabla 2.

Tabla 2. Comparación entre las herramientas de ABW

Herramienta	Características	Limitación
Pathload	Basado en Self-Loading Periodic Streams. Tipo de paquete UDP. Sondeo Trenes de paquetes	Tiempo de convergencia largos. Afectado por tamaño de los paquetes de prueba y longitud del tren de paquetes
PathChirp	Congestión autoinducida. Espaciado exponencial entre paquetes (Chirp). No requiere sincronización de reloj en transmisor y receptor. Almacenamiento temporal de paquetes recibidos en CPU	El proceso de almacenamiento temporal introduce retrasos entre los paquetes que llegan a la capa de aplicación
YAZ	Basado en Self-Loading Periodic Streams. Paquetes UDP. Tren de paquetes. Pruebas de laboratorio	Solo puede Detectar la expansión o compresión del flujo de prueba
Nestet	Alta convergencia. Paquetes TCP o UDP.	Requiere usar todo el ancho de banda disponible para la medición
TOOP	Velocidad constante. Tren de Pares de paquetes. Espaciamiento variable entre pares. Regresión segmentada, puede calcular ABW aun en presencia de varios saltos congestionados	No hace uso de información de correlación de retardo obtenido a partir de los trenes de paquetes en el receptor
ASSOLO	Self-induced congestion y espaciado exponencial entre paquetes (Chirp). Sondea a varias	

	velocidades con un solo flujo, que es más preciso en el centro del flujo. Se ejecuta en sistema operativo en tiempo real	
FEAT	Velocidad variable para el flujo de paquetes de prueba. Un flujo de ojo de pez consiste en K paquetes de igual tamaño que se envían a una tasa de cambio a partir de L (límite inferior) a U (el límite superior).	
IGI/PTR	Asume un tren sondeo con tres posibilidades M paquetes de sondeo si las brechas se incrementan, K si no se han modificado, y N si se reducen.	Medición afectada por el tamaño del paquete y el número de paquetes de sondeo.
Spruce	Pares de paquetes, tráfico cruzada de distribución Poisson.	
Cprobe	Paquetes ICMP. Registra el tiempo entre la recepción del primer paquete y la recepción del último paquete. Reordena los paquetes a la llegada	Baja la precisión con aumento del tráfico cruzado
ProbeGap	Asume un único cuello de botella. Estima la fracción de tiempo que el enlace está libre	sobrestima el ABW cuando el tráfico cruzado es alto
Delphi	Asume un único cuello de botella, Chirp, multifractal	Precisión depende del uso del enlace de red
IGMPS	Tamaño de paquete variable que depende del tráfico cruzado en la red. Colas FIFO	Requiere acceso tanto al remitente como el receptor
Abing	Dispersión del par de paquetes y puede detectar los cambios en el ancho de banda causado por enrutamiento o congestiones	

Traceband	Realiza 10 sondeos, el primero envía 50 pares de paquetes UDP de 1498 bytes de longitud y las 9 restantes con 30 paquetes. Los tiempos entre paquetes son exponencialmente distribuidos.	

3.2.2 Estudios comparativos de las herramientas

Los estudios comparativos revelan aspectos diferenciadores de cada una de las herramientas cuando se ponen a prueba en diferentes escenarios de red y bajo diferentes tipos de tráfico cruzado de acuerdo con los diferentes autores ((Guerrero & Labrador, 2006), (Shriram, y otros, 2005), (Aceto, Botta, Pescapé, & D'Arienzo, 2011), y (Hu & Steenkiste, 2005)) La complejidad del problema ha llevado a varios autores a hacer consideraciones erradas que se ven reflejadas en el comportamiento de sus respectivas herramientas de estimación (Jain & Dovrolis, 2004) .

A continuación se muestran los trabajos más recientes que evalúan y comparan las herramientas más relevantes antes mencionadas.

En el trabajo "*Traceband: A fast, low overhead and accurate tool for available bandwidth estimation and monitoring*" desarrollado por Guerrero y Labrador (Guerrero & Labrador, 2009) se evalúa el rendimiento de la herramienta Traceband con Spruce y Pathload, en un ambiente controlado y en una red real utilizando diferentes patrones de tráfico. Los resultados comparativos muestran que la herramienta Traceband alcanza un mayor rendimiento que los otros en términos de tiempo de convergencia e intrusión y la misma exactitud de Pathload en todas las condiciones de prueba.

En cuanto a las comparaciones, realizadas por Goldoni en el trabajo "*End to end available bandwidth estimation tools, an experimental comparison*" (Goldoni & Schivi, 2010) los autores comparan varias herramientas bien conocidas en un banco de pruebas real con enlaces a 100 Mbps, y a una velocidad de bits constante (CBR) y un tráfico cruzado tipo Poisson. Los autores evalúan la exactitud, la intrusión, y el tiempo de convergencia de las herramientas. Reportando en los resultados, que la más alta precisión se proporciona por Pathload y YAZ, independientemente de la clase de tráfico cruzado. La intrusión y el tiempo de convergencia de Pathload y

YAZ son más significativos, especialmente durante las pruebas en una red de área amplia emulada. Entre herramientas restantes, Pathchirp, IGI/PTR, y Diettopp logran un rendimiento promedio en todas las situaciones, mientras que Assolo exhibe una alta precisión, baja intrusión, y el tiempo de convergencia es corto.

En la Tabla 3 se muestra el avance de las herramientas de ABW, siento esta un área de reciente interés por los investigadores como se puede observar en la columna de año, a partir del año 2000 se empiezan a desarrollar, en la columna Tipo SW/HW, donde SW denota software y HW Hardware, es evidente que estas herramientas han sido diseñadas de tipo Software y la herramienta ASSOLO siendo de este tipo, realiza el marcado de tiempo en los paquetes utilizando la prioridad del sistema operativo RTOS (Sistema operativo en tiempo real del inglés *Real Time Operative Sistem*), y el modelo PGM es utilizado 56% de las herramientas como base de la estimación de ancho de banda disponible.

De acuerdo con Tabla 3 se puede concluir que existe un campo de acción en el desarrollo a nivel de hardware de la estimación de ancho de banda disponible que aún no ha sido investigada y que es objeto de este trabajo.

Tabla 3. Síntesis estado del arte

HERRAMIENTA	AÑO, AUTOR	TIPO		MODELO
		SW	HW	
TOOP	2000	X		PRM
PATHLOAD	2002	X		PRM
ABING	2003	X		PGM
IGI/PTR	2003	X		PGM
NESTET	2003	X		PRM
PATHCHIRP	2003	X		PRM
SPRUCE	2003	X		PGM
FEAT	2006	X		PRM
PROBEGAB	2006	X		PGM
YAZ	2006	X		PRM
DELFHI	2007	X		PGM
IGMPS	2007	X		PGM
ASSOLO	2009	X		PRM
TRACEBAND	2010	X		PGM
CPROBE	2012	X		PGM

3.2.3 Descripción de TRACEBAND

Traceband es una herramienta de estimación de ancho de banda disponible desarrollada por Guerrero y Labarador (2009). Es una herramienta cliente-servidor escrita en *ANSI C* que utiliza Modelos Ocultos de Markov (*HMM*) para proporcionar estimaciones de *ABW* de forma rápida, continua y precisa, bajo el modelo de estimación *PGM*.

El cliente en *Traceband* ejecuta ciclos de diez estimaciones. En la primera estimación de la herramienta envía 50 pares de paquetes *UDP* de 1498 bytes de longitud. Las nueve estimaciones restantes se llevan a cabo con 30 pares de paquetes de cada una. Esta reducción es posible gracias a *HMM*, el cual es capaz de aprender la dinámica del *ABW* con una muestra inicial y mantener el modelo actualizado con muestras de tamaño reducido. Se encontró por experimentación que diez estimaciones fueron suficientes para mantener una buena precisión con bajo costo operativo. Con 50 pares de paquetes el modelo mostró los mejores resultados de la estimación.

Traceband utiliza diferentes valores para los tiempos entre los paquetes *intra-gap* e *inter-gap*. Los tiempos entre paquetes *intra-gap* hacen referencia al tiempo entre los dos paquetes de cada par de paquetes. El tiempo entre paquetes *intra-gap* o Δ_{in} se especifican en el remitente y se establece igual al tiempo de transmisión de un único paquete de sondeo en el *tigh link*. De ese modo, el par de paquetes será capaz de interactuar el tráfico cruzado en la cola del *tigh link*. Los tiempos entre paquetes *inter-gap* hacen referencia al tiempo entre pares de paquetes de prueba, es decir, el tiempo entre el segundo paquete de sondeo de par $i-1$ y el primer paquete de sondeo par i. Estos tiempos se obtienen utilizando la función *gettimeofday*, y sus valores se envían al receptor en la carga útil de los paquetes de prueba. Similar a Spruce [2], *Traceband* realiza un proceso de muestreo de tipo *Poisson* del ancho de banda disponible con tiempos *inter-gap* distribuidos exponencialmente. Con el fin de mantener la sobrecarga controlada y baja, el valor medio de tiempo *inter-gap* se calcula de manera que la sobrecarga máxima introducida por la herramienta sea del 5% o menos de la capacidad del *tigh link*.

En el lado del receptor, la herramienta recibe cada uno de los paquetes de prueba a nivel de *kernel*. Esto se realiza mediante el establecimiento de la opción *SO_TIMESTAMP*. Los paquetes se numeran para determinar qué paquetes pertenecen al mismo par y calcular el tiempo relativo de dispersión $\in$ entre ellos. El *HMM* se determina para cada par de paquetes de prueba.

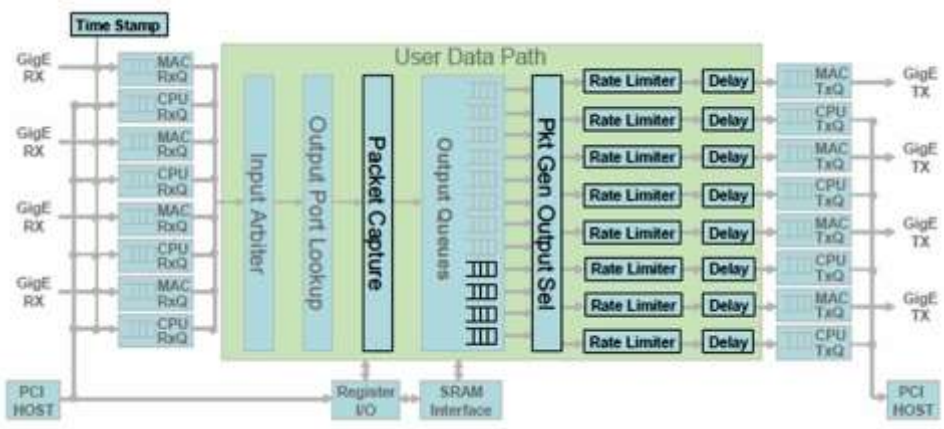
Los resultados experimentales demuestran que el modelo oculto de *Markov* permite a *Traceband* proporcionar estimaciones rápidas y precisas con una tasa de tráfico de prueba bajo. La herramienta también es capaz de reaccionar con rapidez y

precisión a las variaciones del tráfico cruzado, como los presentes en enlaces cargados con tráfico a ráfagas.

3.2.4 Utilización del módulo *TIMESTAMP* en aplicaciones con la NetFPGA

En el proyecto NetFPGA Packet Generator desarrollado por Covington, Gibb, McKeown, & Lockwood (2009) y NetFPGA based Precise Traffic Generation (PTG) desarrollado por Salmon, Ghobadi, Labrecque, Ganjali, & Steffan (2009) en los dos proyectos se implementan procesos de marcación de paquetes, este proceso se realiza operando la NetFPGA como una tarjeta de red estándar o NIC de referencia, donde su uso está orientado a garantizar la precisión a la hora de la captura de los paquetes recibidos. En la Figura 13 se puede apreciar la estructura utilizada que en el generador de paquetes, se utiliza los módulos básicos *input arbiter, output port lockup, output Queues* de la NetFPGA como tarjeta de red. Adicionalmente se puede observar el módulo *time Stamp*. Este módulo esta fuera de la ruta de datos de la red, fue añadido antes que los fifos MAC. Esto permite que los paquetes entrantes sean marcados en tiempo a medida que se reciben por el hardware. Esta marca de tiempo se utiliza más tarde durante la creación de un archivo PCAP que permite una precisión mucho mejor.

Figura 13. Estructura de la NetFPGA utilizada en el generado de paquetes

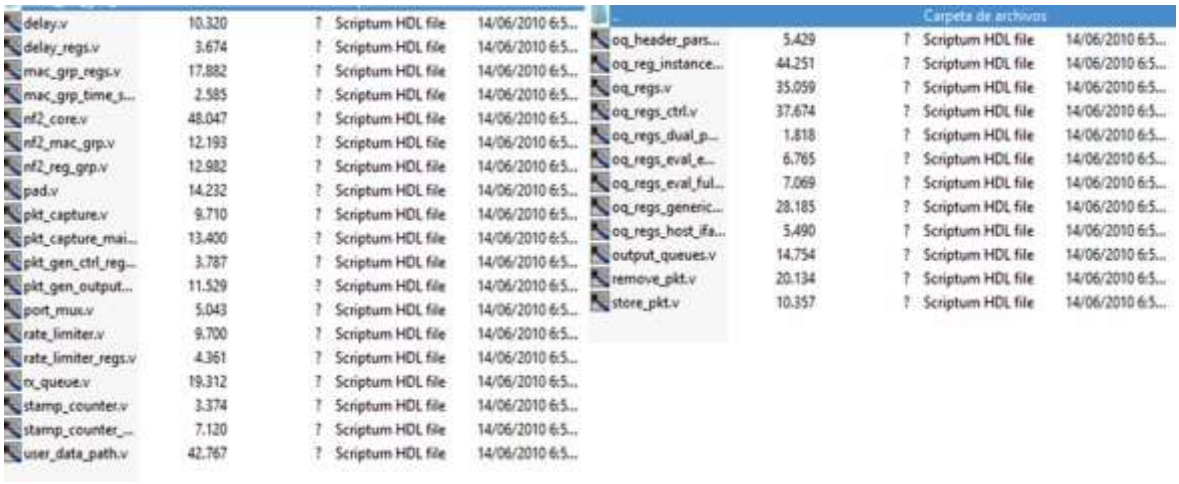


Fuente: (Covington, Gibb, McKeown, & Lockwood, 2009)

El proyecto, se encuentra en la wiki de la NetFPGA, al descargarlo se pueden observar las carpetas bitfiles y Projects y el archivo LICENSE. Al interior de la carpeta *projects*, se hallan las carpetas *include*, la cual contiene las librerías necesarias para la ejecución del proyecto, *regress*, el cual contiene los archivos para los test de prueba. En la carpeta SRC se hallan los archivos *.V los cuales corresponden a los programas escritos en Veriloj, en este caso se hallan 19 ficheros y una carpeta, esta carpeta contiene 12 ficheros como se muestra en la Figura 14

Al analizar los archivos, se encuentra que el timestamp es adicionado en el paquete antes de la última palabra del mismo, para adicionarla se verifica si el valor de in_fifo_ctrl es ‘hEF entonces se guarda la in_fifo_data, se coloca en out_data el valor del timestamp, y luego se pasa la última palabra. Como se altera la longitud del paquete esta se calcula nuevamente sumando 64bits correspondientes al tamaño del registro timestamp.

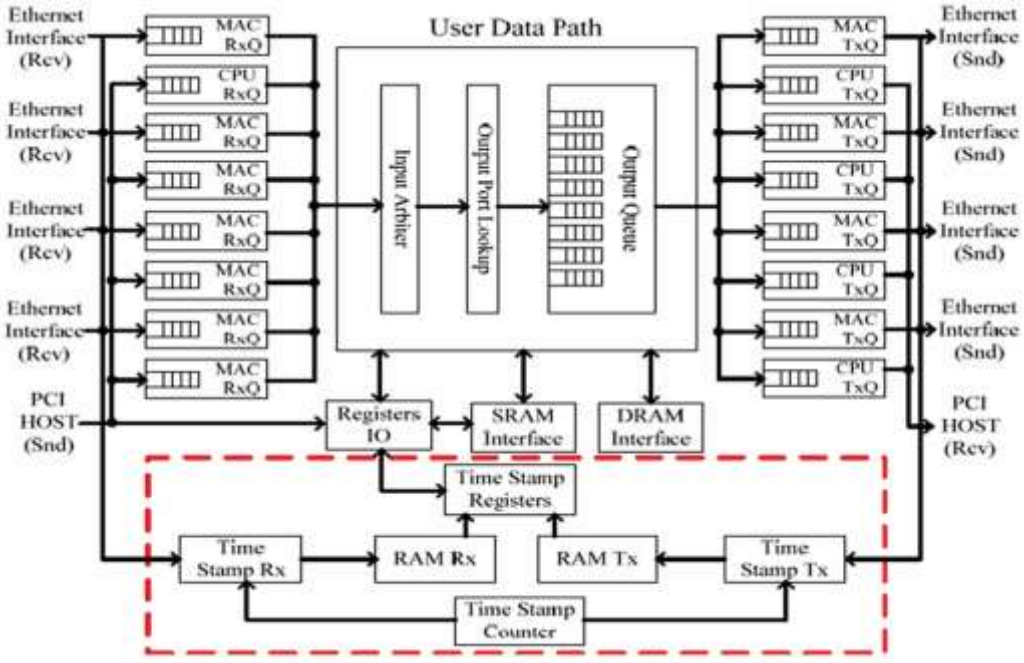
Figura 14. Programas en Veriloj del generador de paquetes



En el trabajo de investigación “***HATS: High Accuracy Timestamping System Based on NetFPGA***” desarrollado por Zhou, Cong, Lu, Deng y Li (2010) se realiza marcado de tiempo en hardware desde la NetFPGA. Los módulos utilizados en el *timestamping* se muestran en la Figura 15. En este caso, el módulo *timestamp* se encuentra por fuera del *User Data Path,* tiene una resolución de 8ns debido a que trabaja con el reloj de 125MHz de la NetFPGA. Adicionalmente puede realizar el marcado tanto en la transmisión como en la recepción, al utilizar un contador general, el cual está conectado a las interfaces Ethernet de transmisión y recepción de la tarjeta.

El contador, se incrementa con cada ciclo de reloj, y se inicia al conectarse la NetFPGA, cuenta con un registro de 64bits, de los cuales 48 son para el *timestamp* lo que permite un periodo de marcación de 26 días aproximadamente, y los siguientes 16bits son para identificar el paquete.

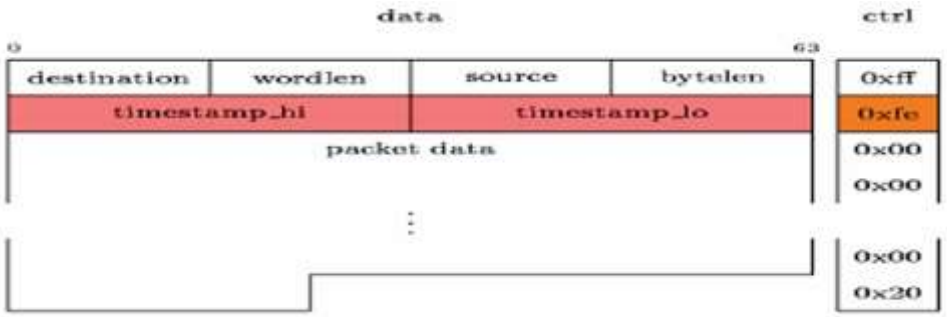
Figura 15. Estructura de la NetFPGA para HATS



Fuente: (Zhou, Cong, Lu, Deng, & Li, 2010)

Rnetprobe de desarrollado por Orosz, Tamas y Imrek (Orosz, Tamas, & Imrek, 2012), es un sistema de supervisión de red basada en NetFPGA compuesto de una aplicación en software que a su vez, se comunica con el hardware NetFPGA en el cual se realizan mediciones activas de extremo a extremo que permiten un mejor análisis para evaluar la calidad de servicio, en servicios sensibles al tiempo como *video stream*. Para cada paquete capturado se genera a nivel MAC una marca de tiempo que es proporcionado por el módulo de hardware FPGA mientras que otra marca de tiempo está etiquetada en el paquete como se muestra en la Figura 16. La segunda marca de tiempo está basada en software, por lo que refleja las características de procesamiento de paquetes en el sistema operativo.

Figura 16. Estructura del encabezado con la marca de tiempo



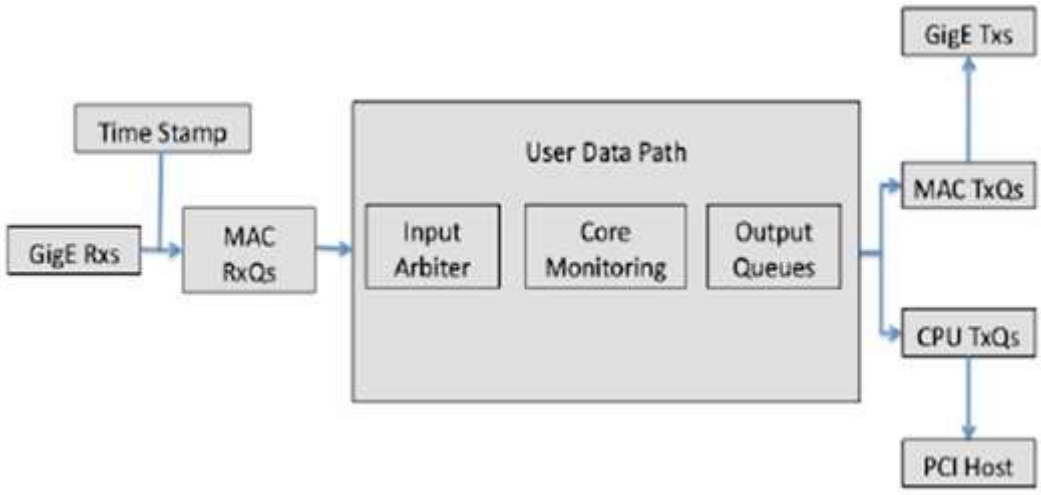
Fuente: (Orosz, Tamas, & Imrek, 2012)

Antichi (Antichi, Giordano, Miller, & Moore, 2012), presenta un pasivo, flexible y rentable sistema de control basado en la cooperación entre PC y NetFPGA para el monitoreo de paquetes como apoyo a los sistemas de detección de intrusiones. La NetFPGA se encarga de filtrar el tráfico y de marcar el tiempo de los paquetes de interés mientras que el software basado en libpcap[1] que ofrece en una interfaz de usuario y la selección de flujo se lleva a cabo por medio de comandos CLI (Command Line Interface). Mientras que todo el tráfico recibido se transmite a través de los puertos de red de la NetFPGA, sólo el tráfico que coincide con una regla de filtrado pasa al software del host.

En la Figura 17 se evidencia que realizan la marca de tiempo antes de que el paquete entre a los fifos MAC con el fin de minimizar el error de marca de tiempo. Las marcas de tiempo son la muestra de un contador de 64-bit, de funcionamiento libre impulsado por el reloj del sistema 125 MHz, lo que incrementa en 8 una vez cada 8ns. Al utilizar el reloj del sistema, el módulo de sello de tiempo puede hacerse síncrona.

[1] Al igual que PCAP, libpcap corresponde a un archivo en el cual se almacenan paquetes capturados en la red. En este caso es analizado por Antichi.

Figura 17. Ruta de datos con timestamp



Fuente: (Antichi, Giordano, Miller, & Moore, 2012)

En la Tabla 4 se realiza una comparación entre las diferentes formas de realizar el *timestamp,* se observa que tres de los cuatro proyectos utilizan el módulo *timestamp* antes de la entrada del paquete al *user data path*, lo que implica que no se generará retardos adicionales por el procesamiento del paquete, en cuando el timestamp que se guarda en el encabezado del paquete se requiere recalcular la longitud del paquete para que sea admitido en el bus PCI, lo cual genera un pequeño retardo y todos los proyectos utilizan la frecuencia del oscilador principal de la NetFPGA que es de 125Mhz con un ciclo de reloj de 8ns mientras que en el trabajo de Antichi (2012) incrementa el contador en ocho cada ciclo de reloj dando una resolución de 1ns.

Tabla 4. Resumen de utilización de TIMESTAMP en proyectos con la NetFPGA

PROYECTO	MARCA DE TIEMPO	MÓDULO	RESOLUCIÓN
Generador de paquetes (Covington, Gibb, McKeown, & Lockwood, 2009)	Registros de memoria SRAM	Fuera del User Data Path, no hay procesamiento del paquete para el marcado	8ns a una frecuencia de 125MHz
Rnetprobe (Orosz, Tamas, & Imrek, 2012)	En el encabezado del paquete	En el User Data Path. Se genera una cabecera que contiene el valor del	8ns a una frecuencia de 125MHz

		marcado de paquete, requiere de procesamiento del paquete.	
Monitoreo de Red (Antichi, Giordano, Miller, & Moore, 2012)	Registros de memoria	Fuera del User Data Path, no hay procesamiento del paquete para el marcado	1ns, incrementa el contador en 8 una vez cada 8ns con un reloj de 125MHz
HATS (Zhou, Cong, Lu, Deng, & Li, 2010)	Registros	Fuera del User Data Path	8ns a una frecuencia de 125MHz

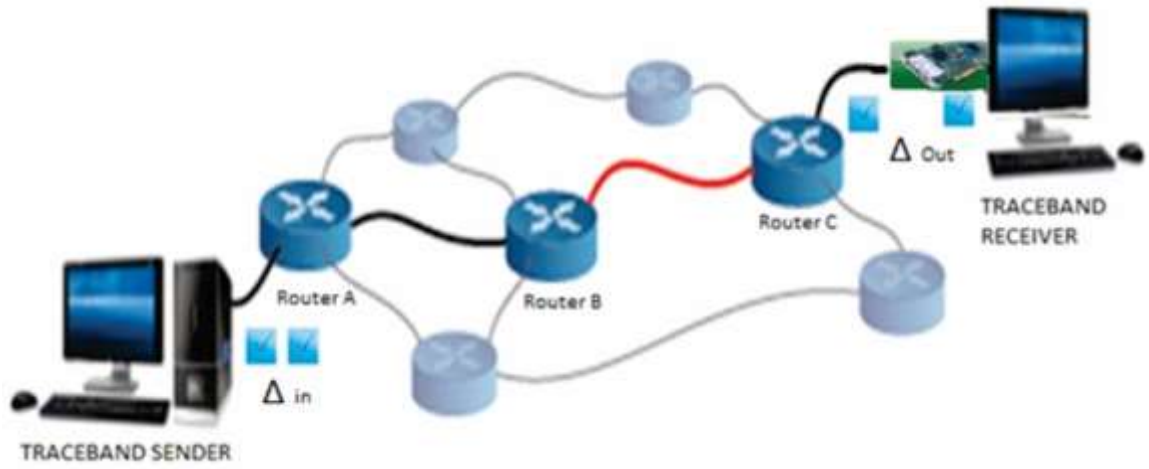
4. METODOLOGÍA

Este proyecto, se enmarca dentro –de la investigación– aplicada, desarrollada a través del método científico, basado en el diseño de una herramienta de estimación del ancho de banda disponible en una red de extremo a extremo.

En esta investigación se utiliza como base la herramienta de estimación de ancho de banda disponible TRACEBAND y la plataforma NetFPGA como se observa en la

Figura 18. TRACEBAND descrito en la sección 3.2.5, funciona mediante dos aplicaciones *sender* y *receiver.* La aplicación TRACEBAND sender se encarga del envió de los paquetes de prueba a un tiempo de separación Δin determinado. En el receptor, se colocará la NetFPGA como tarjeta de red, esta realizará las marcas de tiempo en cada paquete y calculará el tiempo de separación entre pares de paquetes de prueba Δout. Este resultado final Δout, es necesario para realizar la estimación de ABW por la aplicación TRACEBAND receiver.

Figura 18. Esquema general del estimador propuesto



En este proyecto, para cumplir con el diseño planteado anteriormente se establecen 3 fases de desarrollo, estas son

1. Análisis de TRACEBAND, en ella se analiza los algoritmos de TRACEBAND sender y traceband receiver, esto permite determinar en el momento en qué se hace el *timestamp* en el paquete de prueba y cuando se utiliza el resultado de la marcación para la estimación del ancho de banda disponible.
2. Diseño del módulo de *Timestamp* y filtrado en la NetFPGA, en esta fase, se determina donde se inserta el m~~o~~ódulo y sus características, es decir la frecuencia de reloj, la precisión en la marca de tiempo, registro del timestamp entre otras, para incluirlas en el módulo encargado de la marcación en hardware del paquete.
3. Diseño del módulo de comunicación TRACEBAND y NetFPGA, una vez se cuente con el módulo de marcación el siguiente paso es determinar el protocolo que se usará para hacerle llegar el paquete de prueba con la marca de tiempo realizada en la NetFPGA a la herramienta de estimación TRACEBAND.

5. RESULTADOS

En esté capitulo se discute los detalles pertinentes a la plataforma *NetFGPA* y la herramienta de *ABW - TRACEBAND*, y la solución implementada en esta tesis para la estimación de ABW. A continuación se encuentra la descripción de cada uno de los módulos diseñados en el proceso de investigación. En la primera sección se describen los diseños relacionados con la plataforma *NetFPGA,* y la siguiente se enfoca a los ajustes realizados en *TRACENBAND* para comunicarse con la *NetFPGA*

5.1 MÓDULOS *TIMESTAMP* Y COMUNICACIÓN EN LA *NetFPGA*

En el módulo *timestamp* se describe el marcado de tiempo realizado a los paquetes cuando llegan al receptor desde la *NetFPGA*.

5.1.1 Módulo *Timestamp*

El módulo *timestamp*, se puede desarrollar de dos formas de acuerdo a lo discutido en la sección 3.2.2. En la Tabla 5 se exponen las ventajas y desventajas más relevantes relacionadas con el módulo *Timestamp* de acuerdo a los proyectos anteriores mencionados. La primera donde el módulo se halla antes de los FIFOS MAC, el mayor inconveniente de esta es que se requiere de sincronización para la lectura de los registros guardados en la RAM y el acceso a esta desde la aplicación se hace de manera indirecta a través de los registros IO de la *NetFPGA*. La segunda opción requiere de mayor tiempo de procesamiento, puesto que se adiciona *el timestamp* como un paquete *NetFPGA* de 64 bits, la cual no permitiría mejorar la precisión en el estimador de ancho de banda disponible.

Tabla 5. Comparación entre los módulos de Timestamp

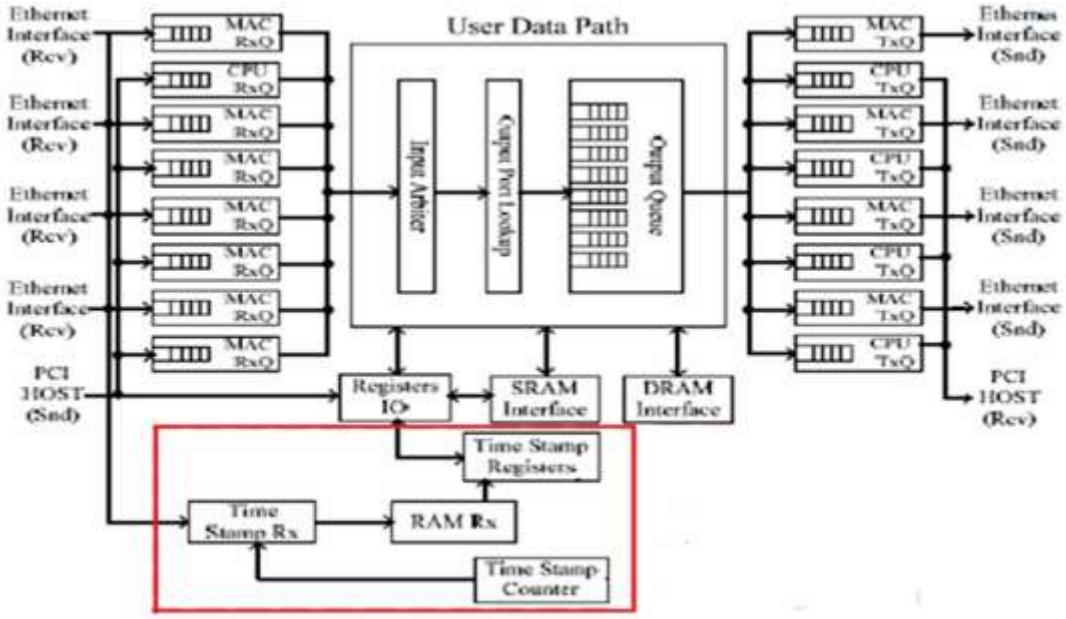
Módulo	Ventajas	Desventajas
Fuera del USER DATA PATH	No genera retardos adicionales dados por el procesamiento de los paquetes	Mayor uso de los recursos de la NetFGA como lo es la RAM en HAST
Dentro del USER DATA PATH	Información del timestamp agregado en la carga útil del paquete UDP Selección del tipo de paquetes al que se le realiza la marca de tiempo	Mayor Retardo por la creación de un módulo en la ruta de datos de la NetFPGA

Para este diseño, de acuerdo a lo anterior la mejor opción es la primera forma de realizar el *timestamp.* Para ello se va a usar dos registros de 32bits formando un registro de 64bits, de los cuales los 16 bits más significativos corresponden a un identificador de paquete y los 48 restantes a la marca de tiempo, al utilizar el reloj de núcleo de 125Mhz, se tiene una resolución de 8ns por ciclo de reloj, dando un tiempo de marcado de 78 horas aproximadamente, lo suficiente para realizar las pruebas de estimación de ancho de banda disponible.

En la Figura 19 se muestra el modelo a utilizar en este diseño basado en la estructura HATS descrita en la sección 3.2.2-, este módulo funciona de la siguiente manera, al detectar en las entradas de las interfaces Ethernet –(Recepción) la llegada de un paquete el módulo Time Stamp Rx se habilita y guarda el *timestamp* junto con el identificador del paquete, este último es el valor de un contador de paquetes, para obtener la marca de tiempo, el módulo se comunica con el modulo *Time Stamp Counter* y este le envía el valor del contador en ese momento–. El módulo *Time Stamp counter* comienza su conteo una vez se descargan los *bitfiles* a la NetFPGA y aumenta con cada ciclo de reloj. Al tener los dos valores *Id_pack y timestamp* se almacena en la RAM de la FPGA como un registro de 64bits. Para la lectura de la RAM es necesario un módulo que lea la RAM cada vez que la aplicación lo solicite, esto es, la aplicación se comunica con REGISTER IO le solicita

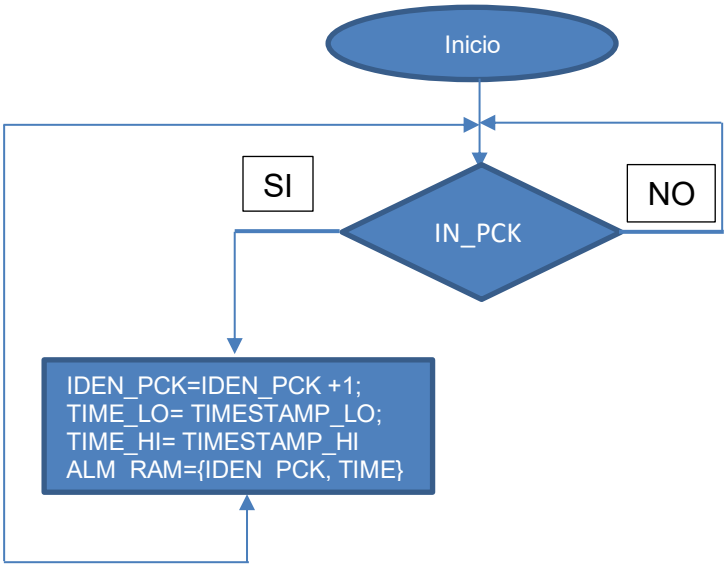
el envío de los timestamp después del tiempo de muestreo, REGISTER IO se enlaza con el módulo TIME STAMP REGISTER, este hace un barrido en las direcciones de la memoria y guarda el valor para pasarlo a REGISTER IO quien se encarga de pasarlo a la aplicación. Si se hace el barrido de todos los paquetes pueden tenerse en cuenta paquetes descartados en el módulo de IDENTIFICACION, para ello se realiza un conteo de los paquetes descartados y del número de paquete entrantes hasta ese momento almacenados en un registro de 64bits, donde estará formado por el registro de 32bit cnt_pck_desc es el número de paquetes descartados y el número de paquetes total. Al realizar la lectura de los *timestamp* se compara el identificador del paquete con el valor guardado, si estos no coinciden, el timestamp para al registro de REGISTER IO y de allí pasará a la aplicación.

Figura 19. Módulo Timestamp



En la Figura 20 se describe el comportamiento del módulo *Time Stamp Rx*, en este se observa que los datos que serán almacenados en la RAM dependen de si en la entrada o las interfaces Ethernet Rx se detecta un nuevo paquete, El *TIMESTAMP* es un registro tipo Wire que está conectado al módulo Time Stamp Counter, como los registros en la NetFPGA son de 32bits y el contador de 48 bits, se llamara timestamp_lo y timestamp_hi, el primero con los 32bit menos significativos del contador y el segundo con los 16bits más significativos del contador.

Figura 20. Diagrama de flujo del módulo Time Stamp RX



5.1.2.1 Código módulos encargados del *timestamp*

Modulo Time Stamp Counter: contador de tiempo

```
Module Time Stamp Counter(

Reg[48:0]          cnt_time,
Input              reset,
Input              clk,

Wire[31:0]         timestamp_lo;
Wire[31:0]         timestamp_hi;
Wire[1:0]          flag;
)
Time_Stamp_Counter#(
.flag              (flag),
.timestamp_lo  (timestamp_lo)
Timestamp_hi   (timestamp_hi)
)
        Always @(flag) begin
            If (flag==1)begin
```

```
            timestamp_lo=cnt_time[31:0];
            timestamp_hi=cnt_time[47: 32];
        end
    end

    Always @(posedge clk) begin
        if (reset )
            cnt_time<= 0;
            timestamp_lo<=0;
            timestamp_hi<=0;
        else begin
            cnt_time<= cnt_time + 1;
            end
    end
end //module
```

//este módulo es el encargado de guardar los valores de timestamp en RAM

Module Time_Stamp_Rx(

```
Localparameter   REG_ADDR_WIDTH=32;
Localparameter   DATA_WIDTH=32;
Input            ram_reg_req,
Output           ram_reg_ack,
Input            ram_reg_rd_wr_l,
Input[REG_ADDR_WIDTH-1:0]      ram_reg_addr,
Input[DATA_WIDTH -1:0]         ram_reg_data,
Output           disable_reset,
Input[3:0]       mac_rx
Wire[15:0]       ctrl_addr,
Wire[15:0]       ram_addr,
Wire[64:0]       ram_rd_data,
Wire[64:0]       ram_wr_data,

Wire[31:0]       timestamp_lo;
Wire[31:0]       timestamp_hi;
Wire             flag;
Wire             start
Reg[15:0]        cnt_pck,
Reg[15:0]        times_hi
Reg[31:0]        times_lo

Generic_reg #(
.REG_ADDR_WIDTH    (REG_ADDR_WIDTH),
.ram_addr          (ctrl_addr),
.ram_data          (ram_data),
.start             (start),
.clk               (clk),
```

```
.reset                    (reset),
.ram_wr_data              (ram_wr_data),
.ram_rd_data               (ram_rd_data),
)
Assign   Cnt_pck= 'h0;
Always @(mac_rx)
          Flag=1;
          Times_lo =timestamp_lo;
          Times_hi=timestamp_hi;
           Ram_addr=cnt_pck;
          Ram_wr_data={cnt_pck,times_hi,times_lo}
           Cnt_pck=cnt_pck+1'h0;

)
```

5.1.3 Módulo Identificación

El módulo identificación se hizo necesario para reconocer que paquete pertenece a los paquetes de prueba y que permita transferir a la aplicación los datos de *timestamp* correspondientes. El objetivo del módulo de identificación es reconocer que el paquete que ingresa a la ruta de datos de la *NetFPGA*, es un paquete UDP correspondiente a los paquetes de prueba enviados por *TRACEBAND*. En este momento es importante recordar cómo ésta conformado un paquete UDP/IP. -En la

Tabla 6 se muestra los campos de un paquete UDP, por ejemplo los primeros 16 bits corresponden a la dirección MAC origen (HIGH), los siguientes 48 bits contienen la dirección MAC destino, del 64 al 95 se tiene cinco campos: dirección MAC origen, identificación del protocolo de transporte si es UDP o TCP, y las banderas V, L y TOS.

Tabla 6. Encabezado de un paquete UDP en la NetFPGA

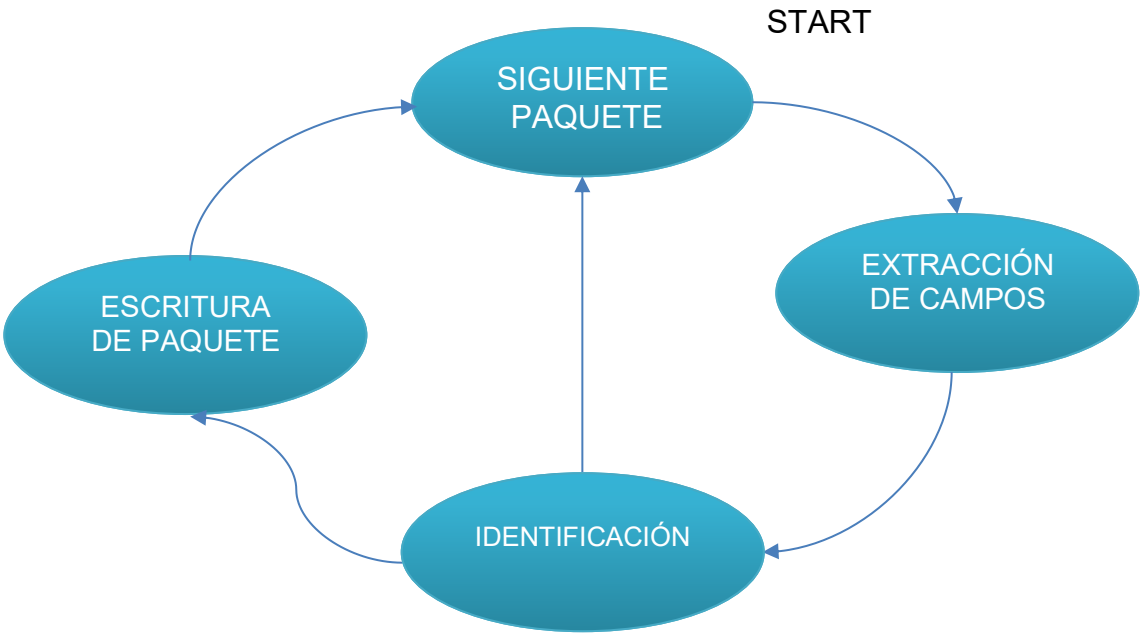
BITS	63-48	47-32	31-16	15-0
0	DIRECCIÓN MAC DESTINO			DIRECCION MAC ORIGEN HI
64	DIRECCION MAC ORIGEN LO		TIPO ETHERNET	V L TOS
128	LONGITUD TOTAL	ID	FLAGS	TTL PROT
192	CHKSUM	ORIGEN IP		DESTINO IP HI

256	DESTINO IP LO	PUERTO UDP ORIGEN	PUERTO UDP DESTINO	LONGITUD UDP
320	UDP CKSUM	EV	NUMERO DE SECUENCIA DE PAQUETE	

A continuación se detalla la máquina de estados del módulo identificación (ver Figura 21). Este módulo se basa en cinco estados, SIGUIENTE PAQUETE, EXTRACCION DE CAMPOS, IDENTIFICACION Y ESCRITURA DE PAQUETE. La función de SIGUIENTE PAQUETE es –anunciar la llegada de un nuevo paquete, una vez la FIFO contiene los datos y encabezados del paquete para ello requiere de verificar el estado de la FIFO y decirle al módulo anterior que está en proceso de lectura. –En el estado EXTRACCION DE CAMPOS se encarga de guardar los valores de los campos como direcciones IP destino y IP origen, Puertos origen y destino, tipo de protocolo que corresponden al encabezado del paquete. Para ello y de acuerdo a la

Tabla 6, se requiere de 6 ciclos de reloj que permitan hacer el barrido por el paquete. En el estado IDENTIFICACIÓN se verifica que el tipo de protocolo sea UDP esto es, que el valor de este campo sea 17, y que el puerto destino sea 3504, si esto es verdadero, se pasa a organizar de nuevo el paquete para pasarlo al siguiente módulo sino se pasa a recibir el siguiente paquete.

Figura 21. Máquina de estados



En la

Tabla 7 se pueden ver las variables involucradas en el proceso. La variable fsm_state es la encargada del cambio de estado en la máquina de estado, cnt_reset y cnt son variables que controlan el ciclo de extracción de campos, en este estado, se tienen en cuenta las variables in_fifo_data_rdy la cual indica que se está leyendo los datos de la fifo, e in_fifo_ctrl es la variable que se verifica para conocer el estado de la lectura de la fifo, cuando esta llega a 'hFF significa que es el última palabra, protocol_field es la encargada de guardar la información del protocolo y port_dtn del puerto destino permiten verificar sí el paquete corresponde a un paquete de prueba, se lleva la cuenta de los paquetes entrantes de prueba, a los cuales se les coloca un identificador que será usado en el momento de pasar el *timestamp* a la aplicación, y se cuentan también los paquetes que son descartados y se lleva la cuenta del total de los paquetes con las variables cnt_pck_prueba, cnt_pck_des y cnt_pck_total.

Tabla 7. Relación de estados y variables de entrada y salida

ESTADO ACTUAL	ESTADO SIGUIENTE	VARIABLES DE ENTRADA	VARIABLES DE SALIDA
Siguiente Paquete	Extracción De Campos	Fsm_state=0	Cnt_reset=1 Nfsm_state=1
Extracción de Campo	Identificación	In_fifo_data_rdy=1 In_fifo_ctrl=´hFF Cnt=1	Nfsm_state=2
Identificación	Escritura de paquete	Protocol_field=17 Port_dtn=3504	Nfsm_state=4
Escritura de paquete	Siguiente paquete	Out_rdy	Nfsm_state=0 Out_data
Identificación	Siguiente paquete	Protocol_field=?	Nfsm_state=0

5.1.3.1 Código en Verilog para el módulo de identificación.

El código que se muestra a continuación está basado en el proyecto NETFLOW encontrado en la WIKI de la NetFPGA (Zadnik , 2008).

Module Identificación

```
#(
    parameter DATA_WIDTH = 64,
    parameter CTRL_WIDTH = DATA_WIDTH/8,
    parameter UDP_REG_SRC_WIDTH = 2
  )
  (
    input  [DATA_WIDTH-1:0]          in_data,
    input  [CTRL_WIDTH-1:0]          in_ctrl,
    input                            in_wr,
    output                           in_rdy,
```

```
    output reg [DATA_WIDTH-1:0]       out_data,
    output reg [CTRL_WIDTH-1:0]       out_ctrl,
    output reg                        out_wr,
    input                             out_rdy,
// --- Interfaces tipo registro

    input                                     reg_req_in,
    input                                     reg_ack_in,
    input                                     reg_rd_wr_L_in,
    input  [`UDP_REG_ADDR_WIDTH-1:0]          reg_addr_in,
    input  [`CPCI_NF2_DATA_WIDTH-1:0]         reg_data_in,
    input  [UDP_REG_SRC_WIDTH-1:0]            reg_src_in,

    output                                    reg_req_out,
    output                                    reg_ack_out,
    output                                    reg_rd_wr_L_out,
    output [`UDP_REG_ADDR_WIDTH-1:0]          reg_addr_out,
    output [`CPCI_NF2_DATA_WIDTH-1:0]         reg_data_out,
    output [UDP_REG_SRC_WIDTH-1:0]            reg_src_out,

    // misc
    input                             reset,
    input                             clk
  );
// Estados posibles de la máquina de estados descrita anteriormente
  localparam SIGUIENTE_PAQUETE  = 0;
  localparam EXTRACCION         = 1;
  localparam IDENTIFICACION     = 2;
  localparam ESCRITURA          = 3;

// Instancia de FIFO
  wire [DATA_WIDTH-1:0]           in_fifo_data;
  wire [CTRL_WIDTH-1:0]           in_fifo_ctrl;
  wire                            in_fifo_nearly_full;
  wire                            in_fifo_empty;
  reg                             in_fifo_rd_en;


// Registros de interface con módulo time Stamp Register
wire [31:0]                       cnt_pck_prueba
wire [31:0]                       cnt_pck_desc
wire [31:0]                       cnt_pck_total
//Registros utilizados para almacenar los campos del encabezado UDP
  reg [31:0]                      ipsrc, ipdst;
  reg [3:0]                       ipversion;
  reg [15:0]                      src_port, dst_port;
  reg [7:0]                       protocol_type;
  reg [1:0]                       protocol_field;
  reg [15:0]                      packet_len_field;
```

```
  reg [7:0]                       tos;
  reg [7:0]                       ttl;
  reg [7:0]                       tcpflags;
  reg [15:0]                      doctets;


// Registros necesarios para la extracción de los campos
  reg [3:0]                       cnt;
  reg                             cnt_reset;


 // Registros de control del estado de la máquina
  reg [2:0]  fsm_state, nfsm_state;

// Registros auxiliares en los diferentes procesos
  reg       registers_ready;
  reg       allow_extract;
  reg       accepted;
  reg [31:0] cnt_pck_prueba;
  reg [31:0] cnt_pck_des;
  reg       total_en;
  reg [31:0] cnt_total;

// Paquete guardado en la FIFO

  fallthrough_small_fifo #(
    .WIDTH(CTRL_WIDTH+DATA_WIDTH),
    .MAX_DEPTH_BITS(2)
  ) input_fifo (
    .din            ({in_ctrl, in_data}),     // Entrada de datos
    .wr_en          (in_wr),                  //Habilitación de escritura
    .rd_en          (in_fifo_rd_en),          // Habilitación de lectura de la siguiente palabra
    .dout           ({in_fifo_ctrl, in_fifo_data}),
    .full           (),
    .nearly_full    (in_fifo_nearly_full),
    .empty          (in_fifo_empty),
    .reset          (reset),
    .clk          (clk)
  );

   generic_regs   Registros generales
  #(
    .UDP_REG_SRC_WIDTH   (UDP_REG_SRC_WIDTH),
    .REG_ADDR_WIDTH     (`ESTIMACION_IDENTIFICACION _REG_ADDR_WIDTH),
// Width of block addresses -- eg. MODULE_REG_ADDR_WIDTH

  ) module_regs (
    .reg_req_in                   (reg_req_in),
    .reg_ack_in                   (reg_ack_in),
```

```
    .reg_rd_wr_L_in          (reg_rd_wr_L_in),
    .reg_addr_in             (reg_addr_in),
    .reg_data_in             (reg_data_in),
    .reg_src_in              (reg_src_in),

    .reg_req_out             (reg_req_out),
    .reg_ack_out             (reg_ack_out),
    .reg_rd_wr_L_out         (reg_rd_wr_L_out),
    .reg_addr_out            (reg_addr_out),
    .reg_data_out            (reg_data_out),
    .reg_src_out             (reg_src_out),

    .cnt_pck_prueba          (cnt_pck_prueba),
    .cnt_pck_desc            (cnt_pck_desc),
    .cnt_pck_total           (cnt_pck_total),
    .clk                     (clk),
    .reset                   (reset)
  );

  assign in_rdy          = !in_fifo_nearly_full
  assign in_fifo_data_rdy = !in_fifo_empty;
```

Máquina de estados

```
always @(*) begin
    allow_extract  = 0;
    in_fifo_rd_en  = 0;
    cnt_reset      = 0;
    out_wr         = 0;
    out_ctrl       = 'h0;
    nfsm_state     = fsm_state;
    cnt_pck_prueba=0;
    cnt_pck_des=0;
    cnt_total=0;
    case (fsm_state)

      SIGUIENTE_PAQUETE: begin
       cnt_reset  = 1;
       if (in_fifo_data_rdy) begin
         if (in_fifo_ctrl == 'hFF)
           cnt_total=cnt_total+1;
           nfsm_state = EXTRACCION;
        else
          in_fifo_rd_en  = 1;
        end
      end

     EXTRACCION: begin
       if (in_fifo_data_rdy) begin
         allow_extract = 1;
```

```
            in_fifo_rd_en = 1;
            if (cnt == 5) begin
              nfsm_state=IDENTIFICACIÓN;
             end
        end

        IDENTIFICACION: begin
          if (protocol_ident==3) begin
                    if(port_ok==1)
                       nfsm_state = ESCRIBIR_PAQUETE;
                        else
                        nfsm_state = SIGUIENTE_PAQUETE;
                     end
           else
           nfsm_state= SIGUIENTE_PAQUETE;
          end

        ESCRITURA: begin
            out_wr=1'b0;
            out_ctrl='h0;
            for (i = 0; i < 5; i = i + 1) begin

          out_data=Word[i];
          end
          nfsm_nxt=SIGUIENTE_PAQUETE;
    end
end
```

Extracción de encabezado del paquete

```
always @(posedge clk) begin
     if (allow_extract && in_fifo_data_rdy) begin
        case (cnt)
          0: begin
               input_port          = in_fifo_data[31:16];
              Word[0]              = in_fifo_data[63:0]
            end
          1: begin
               Word[1]             = in_fifo_data[63:0];
             end
          2: begin
               packet_len_field  = in_fifo_data[31:16];
               ipversion           = in_fifo_data[15:12];
               tos                 = in_fifo_data[7:0];
               Word[2]             = in_fifo_data[63:0];
            end

          3: begin
               doctets                = in_fifo_data[63:48];
```

```
            ttl                 = in_fifo_data[15:8];
            protocol_field      = in_fifo_data[7:0];
            Word[3]             = in_fifo_data[63:0];
          end

        4: begin
            ipsrc           = in_fifo_data[47:16];
            ipdst[31:16]    = in_fifo_data[15:0];
            Word[4]         = in_fifo_data[63:0];
          end

        5: begin
            ipdst[15:0]     = in_fifo_data[63:48];
            src_port        = in_fifo_data[47:32];
            dst_port        = in_fifo_data[31:16];
           Word[5]          = in_fifo_data[63:0];
          end

        default:   ;
      endcase
    end
  end
```

// Registro del cambio de estado

```
always @(posedge clk) begin
    if (reset)
      fsm_state = SKIPTONEXTPACKET;
    else
      fsm_state = nfsm_state;
end
```

assign port_ok = (dst_port== 16'h0DB0)? 1:0;

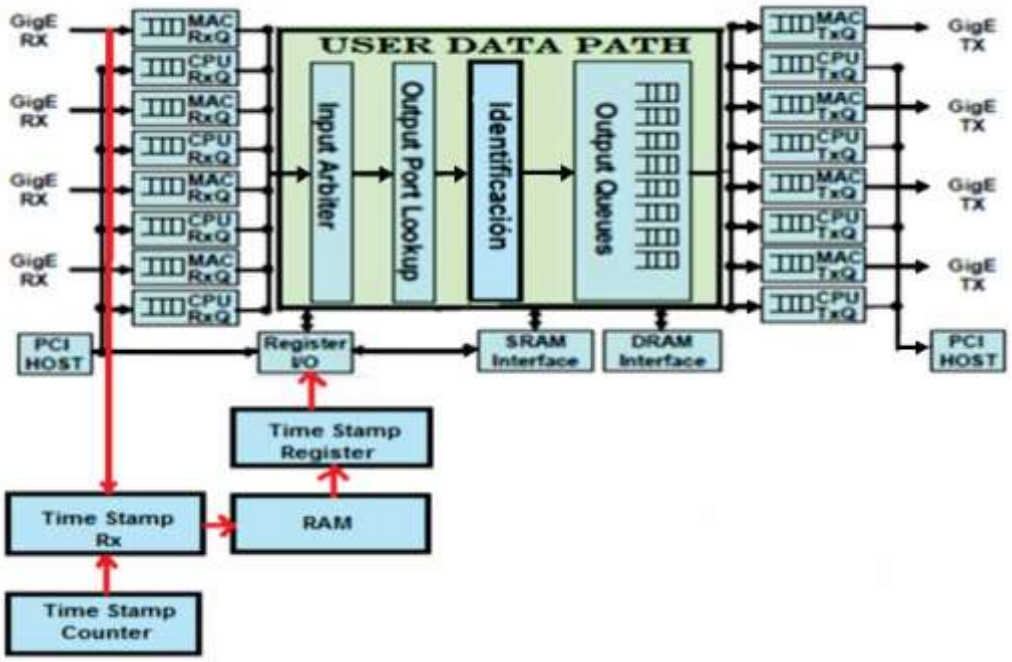
```
  always @(*) begin
     protocol_ident = 0;

    case (protocol_field)
      1: protocol_ident = 1; //ICMP
      6: protocol_ident = 2; //TCP
     17: protocol_ident = 3; //UDP
     default: protocol_ident = 0;
    endcase
  end

end //module
```

5.1.4 Integración de los módulo *Timestamp* e identificación

En la

Figura 22 se muestra la ruta de datos desde el interior de la NetFPGA hasta la aplicación en PC host diseñada en el desarrollo del presente trabajo. A continuación se describe la ruta del paquete en la NetFPGA. Inicialmente el paquete ingresa por el puerto Ethernet y el módulo Input Arbiter se encarga de direccionar los paquetes a los siguientes módulos, el módulo de identificación recibe el paquete que viene del Output Port Lookup, se realiza un filtrado de los paquetes que corresponden al protocolo UDP al puerto 3504. En el módulo Time Stamp Rx, se toman las marcas de tiempo de los paquetes y se halla la diferencia de llegada entre pares de paquetes de prueba en el módulo Time Stamp Register, y se pasa la información de los Δout a TRACEBAND a través de los REGISTROS IO

Figura 22. Diseño de la solución para la ruta de datos en la NetFPGA



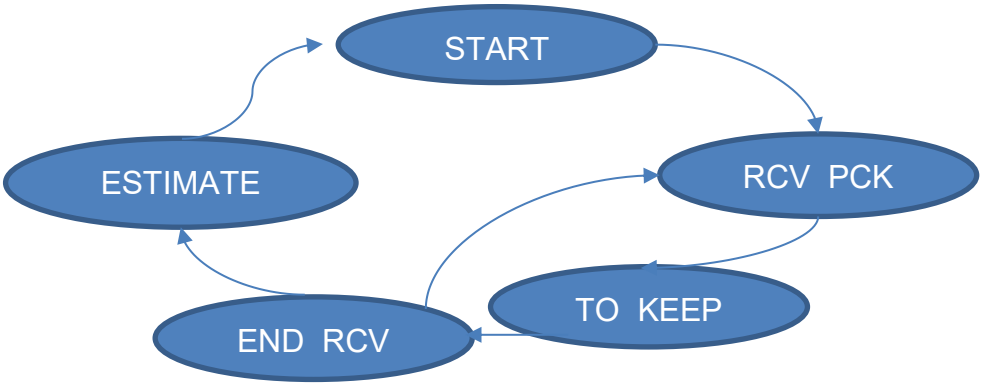
5.2 COMUNICACIÓN TRACEBAND CON LA NETFPGA

El propósito de este apartado es mostrar en primer lugar la forma en que Traceban_rcv funciona y los cambios que se deben hacer en su código para que establezca un enlace con la NetFPGA, específicamente para obtener el valor del de los paquetes prueba en la recepción.

5.2.1 Análisis de *TRACEBAND RECEIVER*

Al realizar una revisión del código de la aplicación de aplicación Traceband_rcv la cual se encarga de recibir los paquetes de prueba y el procesamiento de estos paquetes para la estimación de ancho de banda disponible. En la Figura 23, se muestra el proceso llevado a cabo por la aplicación para la estimación de ancho de banda disponible. Su funcionamiento se inicia llevando a las condiciones iniciales las variables vinculadas a cada estado. Al llegar un paquete, se realiza el marcado de tiempo y se guarda la información del paquete, luego se registra en una tabla los datos como número de paquete(pck_num), tamaño del paquete (pck_size), marca de tiempo de trasmisión en segundos y microsegundos (snd_time); y la marca de tiempo de recepción (rcv_time), se revisa si se finalizó la recepción de los paquetes, si es así, se procede a realizar la estimación de ancho de banda disponible con los datos guardados en la tabla y el resultado del Modelo Oculto de Markov, sino se procede a recibir el siguiente paquete hasta que llegue el último paquete de prueba cuyo tamaño es 0bytes. El código de Traceban_rcv se encuentra en el anexo 1.

Figura 23. Máquina de estados de Traceband_rcv



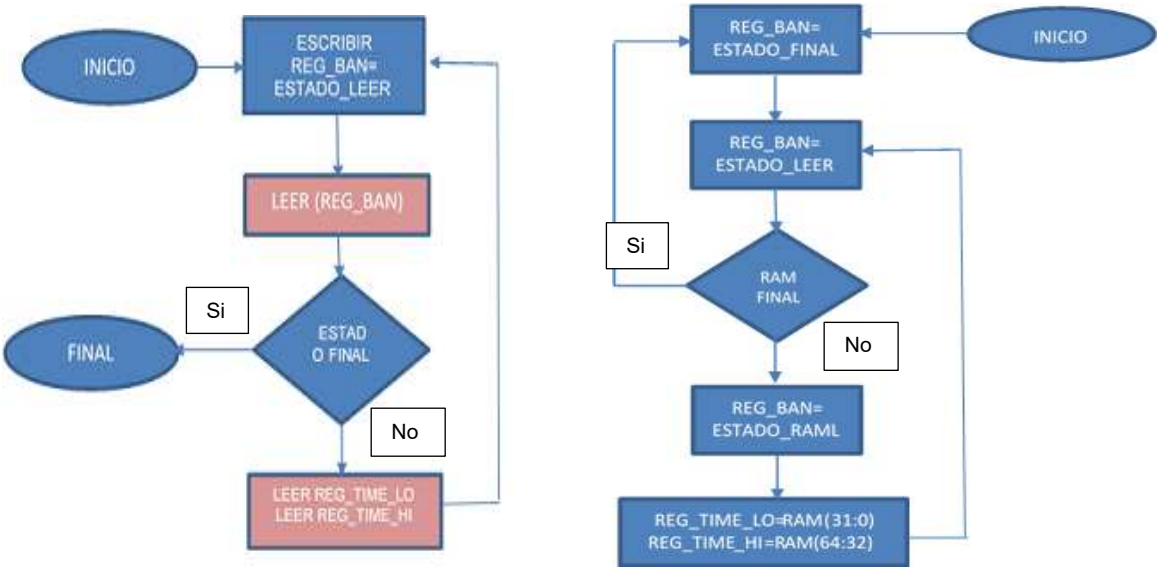
5.2.2 Comunicación entre Traceband_rcv y la NetFPGA

En el apartado anterior se menciona que en la aplicación Traceband_rcv en el estado RCV_PCK de la máquina de estados, al ingresar un paquete de prueba se realiza el *timestamp* con la función SCM_TIMESTAMP, y el resultado es guardado en la tabla Traceband_table[pck_cnt]. En este diseño se pretende realizar esta función en hardware desde la NetFPGA. El *Timestamp* se guarda en la RAM de la misma, y el acceso a esta desde la memoria no es directamente, sino a través de los registros IO de la misma. La RAM tiene una capacidad de 24576*64bits, lo que permite guardar 24575 datos, cuando se llega a la posición 24576, se reescribe la posición 0. Sin embargo el máximo número de paquetes utilizado por Traceband_snd es de 320; lo que es mucho menos que la capacidad de la RAM. Una vez se haya terminado de recibir los paquetes y en la RAM este registrado el tiempo en que llegaron dichos paquetes, la aplicación debe solicitar el envío de la información y se realiza el barrido de la RAM, registrando la información en la traceband_table[pck_con].rcv_time. Antes de guardar el *timestamp* en la tabla se debe realizar la conversión de nanosegundos a segundos y micro segundos puesto que en estas unidades de tiempo se halla snd_time.

En el trabajo desarrollado por Zhou (Zhou, Cong, Lu, Deng, & Li, 2010) incorporaron un mecanismo de sincronización para realizar la lectura de los *timestamp* de los paquetes recibidos como se muestra en la Figura 24. En la parte izquierda corresponde al mecanismo para la aplicación y en la parte derecha el mecanismo para el hardware.

La transferencia de la información es iniciada por la aplicación, ésta escribe en el registro bandera (REG_BAN) el valor del estado leer, este registro es evaluado por el hardware y mientras este en este estado, el procede a verificar si se leyó completamente la RAM, sino guarda en los registros reg_time_lo y reg_time_hi, de la marca de tiempo, el registro vuelve a cambiar de estado, cuando se ésta leyendo la RAM, este estado es leído por la aplicación guardando los valores y coloca el REG_BAN en Estado leer, cuando se ha terminado la lectura de la RAM se cambia el REG_BAN y tanto la aplicación como el hardware cierran el proceso hasta un nuevo llamado.

Figura 24. Mecanismo de sincronización para comunicación entre aplicación y NetFPGA



Fuente (Zhou, Cong, Lu, Deng, & Li, 2010)

En este diseño, basado en el mecanismo de sincronización descrito anteriormente, para que se pueda comunicar la aplicación con el hardware es necesario declarar el registro como registro software y para comunicar el hardware con la aplicación deberá usarse registros tipo hardware, en este caso se utilizaran tres tipo hardware (reg_ban1, timestamp_lo y timestamp_hi) y uno tipo software, (reg_ban2).

Además de realizar la lectura de la RAM debe identificarse en hardware si el *timestamp* guardado corresponde a un paquete rechazado en el módulo identificador de la NetFPGA, de ser así, el registro no se pasa a la aplicación y se continúa con la lectura de la siguiente posición de memoria.

El código en traceband_rcv para la comunicación con la NetFPGA, utiliza llamadas ioctl para hacer lecturas y escrituras de los registros del módulo REGISTER I/O. Las llamadas ioctl se envuelven en dos funciones simples READREG y WRITEREG. Una vez se haya terminado de recibir los paquetes de prueba, para ese momento, el número total de paquetes estará guardado en pkt_cnt y será las veces que se leerá la RAM. En el encabezado será necesario incluir las librerías para el manejo de los registros de la NetFPGA

Encabezado

```
#include <unistd.h>
```

```
 #include <net/if.h>
 #include "../common/reg_defines.h"
 #include "../common/nf2.h"
 #include "../common/nf2util.h"

  #define PATHLEN             80
  #define DEFAULT_IFACE  "nf2c0"
/* Variables Globales */
  static struct nf2device nf2;

void lectura();
void usage (void);

        int main(ESTADO_LEER, ESTADO_FINAL, i, REG_BAN2, REG_BAN1,
TIMES_LO, TIMES_HI)
        {
          nf2.device_name = DEFAULT_IFACE;
          // Abriendo interface
          if (check_iface(&nf2))
            {
              exit(1);
             }
          if (openDescriptor(&nf2))
            {
        exit(1);
            }
         lectura();
         closeDescriptor(&nf2);
        return 0;
         }
void lectura()
        {
          unsigned val;
            ESTADO_LEER=0;
         RAM_READ=1;
        writeReg(&nf2, REG_BAN2,& ESTADO_LEER );
           (for i=0 ; i < (*pck_cnt) ; i++)
                {
                   readReg(&nf2, REG_BAN1, &val);
                            if (val=RAM_READ)
                            { readReg(&nf2, TIMES_LO, TIMES_HI &valor_lo, valor_hi);
                            Con1=itoa(valor_lo);
                            Con2=itoa(valor_hi);
                            Con3=con2+con1;
                            I=atoi(con3);
                            timestamp.tv_s=(I*8)/1000000000;
                            timestamp.tv_usec=(I*8)%1000000000) ;
```

```
                    memcpy(&(traceband_table[*pck_cnt].rcv_time),      &rcv_time,
                    sizeof(struct timeval));
                        }
                }
            RETURN
        }
}
```

6. CONCLUSIONES

En este trabajo, se diseñaron los mecanismos para realizar la estimación de ancho de banda disponible utilizando la herramienta TRACEBAND y NetFPGA. Esta medición requiere del tiempo en que llegan los paquetes de prueba al receptor. Para dar mayor precisión a la estimación, se realizó el timestamp a nivel de hardware y a nivel de software el cálculo del ancho de banda disponible

En la NetFPGA se utilizaron los módulos Time_Stamp_Counter, Time_Stamp_Rx y Time_Stamp_Register, y el módulo Register IO, y la memoria RAM para el Timestamp. Estos módulos se encuentran fuera del *user_data_path* para no tener en cuenta el tiempo de procesamiento del paquete. El primer módulo mencionado es encargado de conteo de pulsos del reloj, a una frecuencia de 125Mhz, para una resolución de 8ns por ciclo en la NetFPGA de esta desde que se descargan los bitfile. En el módulo Time_Stamp_Rx se diseñó de tal manera que al ingresar un paquete por las interfaces Ethernet se registre el tiempo en que llegó tomando el tiempo del Time_Stamp_Counter y es almacenado en la memoria RAM de la NetFPGA. En el módulo Time_Stamp_Register se encarga de almacenar el número de paquetes descartados y su posición dentro de los paquetes totales entrantes, mientras se realiza el muestreo. Cuando tranceband_rcv lo requiere este se encarga de leer la RAM para pasar el timestamp solo de cada paquete de prueba

Adicionalmente, se diseñó un módulo llamado IDENTIFICACION, ubicado luego del módulo Output_Port_Lookup, del user data path el cual permite comprobar si el paquete entrante es un paquete de prueba, utilizando los parámetros puerto y protocolo del encabezado TCP/IP de los paquetes. Se lleva la cuenta de los paquetes de prueba, paquetes descartados y el total de los paquetes, esto se envía al módulo Time_Stamp_Register.

Para comunicar Traceband con la NetFPGA, se hizo necesario declarar tres registros de tipo hardware y uno de tipo software, los cuales permiten la transferencia de la información del timestamp y del estado del registro bandera, que

es el encargado de dar inicio a la lectura y estado del proceso de lectura de la RAM. En traceband_rcv se incluyen las librerías que permiten la identificación del dispositivo, la interface Ethernet de la NetFPGA. La utilización de las llamadas ioctl con el uso de las funciones writereg y readreg se realiza la escritura y lectura de los registros del módulo REGISTER IO respectivamente

ANEXOS

Anexos A Código de Traceband_rcv

```
#include "traceband_fn.h"
#include "hmm.h"

//----- Function Prototypes --------------------------------------
trace *dump_array(long, int*, int, trace *);
void dump_file(char *, trace *);
int prep_sockets(void);
void memory_dump(char *, int); // dump results in file_name

//----- Global variables-----------------------------------------
int traceband_sock; // Socket descriptor
int i, j; // For loops
char peer_actual[16];
struct sockaddr_in snd_echo; // Echo sender address
struct sockaddr_in rcv_echo; // Echo receiver address
int *symbol;
short *sign;
int *pck_cnt; // Number of packets received
int *pair_cnt; // Number of valid pairs received
traceband_pkt *traceband_table;

//=============================================================
//= Main program =
//=============================================================
int main(int argc, char *argv[])
{
static const int tmp_buff_size=2000; // Size of temporal buffer
unsigned int rcv_msg_size; // Size of received message
long pck_size; // Packet size as specified by sender
long path_capacity;
trace *trace_array;
trace *tb_array;
long pck_num=0; // Packet number as specified by sender
char ctrl[CMSG_SPACE(sizeof(struct timeval))];
```

```
char snd_msg[100]; // Message to send back
char *tmp_buff; // Temporal buffer for message received
struct msghdr rcv_msg; // Received message structure
struct iovec iov; // Data storage structure for I/O
struct cmsghdr *crcv_msg = (struct cmsghdr *)&ctrl;
struct timeval rcv_time, snd_time;
HMM hmm;
int N;
int M;
double state;
FILE *fp;
int seed;
if(argc<2) {
printf("usage : %s <trace file name>\n", argv[0]);
exit(0);
}

//Read hmm model (N, M, and B matrix) from file
fp = fopen("hmminitfile", "r");
if (fp == NULL) {
fprintf(stderr, "Error: File hmminitfile not found \n");
exit (1);
}
ReadHMM(fp, &hmm);
fclose(fp);

if (prep_sockets() != 0) DieWithError("Socket initialization failed");
fprintf("%s: waiting for data on UDP port %u\n\n", argv[0],SERVER_PORT);
symbol = (int *) ivector(1, MAX_NUMPCK);
sign = (short *) ivector(1, MAX_NUMPCK);
// Allocate memory to number packets from the sender
pck_cnt = malloc(sizeof(int));
assert(pck_cnt != NULL);
// Allocate memory to number valid pairs from the sender
pair_cnt = malloc(sizeof(int));
assert(pair_cnt != NULL);
// Allocate memory to store at most MAX_NUMPCK packets from the sender
traceband_table = malloc(sizeof(traceband_pkt) * MAX_NUMPCK);
assert(traceband_table != NULL);
// Allocate memory to store the packet from the sender
tmp_buff=malloc(tmp_buff_size);
assert(tmp_buff!=NULL);
// Allocate memory to store at most MAX_NUMPCK packets
tb_array = malloc(sizeof(trace) * MAX_NUMPCK);
assert(tb_array != NULL);
```

```
// fill out the array with zeros
bzero(traceband_table, sizeof(traceband_pkt) * MAX_NUMPCK);
bzero(&rcv_msg, sizeof(rcv_msg));
bzero(&rcv_time, sizeof(rcv_time));
bzero(ctrl, CMSG_SPACE(sizeof(struct timeval)));
bzero(tb_array, sizeof(trace) * MAX_NUMPCK);
iov.iov_base = tmp_buff; // base address of the data storage area
iov.iov_len = tmp_buff_size; // size of the data storage area in bytes
rcv_msg.msg_iov = &iov;
rcv_msg.msg_iovlen = 1;
rcv_msg.msg_namelen = sizeof(snd_echo);
rcv_msg.msg_name = &snd_echo;
rcv_msg.msg_control = (caddr_t) ctrl;
rcv_msg.msg_controllen = sizeof(ctrl);
srand((unsigned int)time((time_t *)NULL)); // to reset the seed
128 // Estimation of initial state probabilities:
hmm.pi = (double *) dvector(1, hmm.N);
for(i=1;i<=hmm.N;i++) hmm.pi[i]=rand_uniform(0,1); // random initial values
hmm.pi = normalize_vector(hmm.pi, hmm.N);
//// Estimation of initial state transition probabilities:
hmm.A = (double **) dmatrix(1, hmm.N, 1, hmm.N);
hmm.A = onestepTransMatrix(hmm.A, hmm.N);
//InitHMM(&hmm, hmm.N, hmm.M, seed);
*pck_cnt=0;
// Get and timestamp packets from the sender
while (TRUE)
{
if((rcv_msg_size=recvmsg(traceband_sock,&rcv_msg, 0)) < 0)
DieWithError("recv() failed");

if (crcv_msg->cmsg_level == SOL_SOCKET &&
crcv_msg->cmsg_type == SCM_TIMESTAMP &&
crcv_msg->cmsg_len == CMSG_LEN(sizeof(struct timeval)))
{ // Copy to avoid alignment problems
memcpy(&rcv_time,CMSG_DATA(crcv_msg),sizeof(rcv_time));
} else {
gettimeofday(&rcv_time, NULL);
fprintf (stderr, "didn't get timestamp data!\n");
}
pck_num = ntohl(*(((long*)tmp_buff)+0));
pck_size = ntohl(*(((long*)tmp_buff)+1));
snd_time.tv_sec = ntohl(*(((long*)tmp_buff)+2));
snd_time.tv_usec = ntohl(*(((long*)tmp_buff)+3));
if (pck_size == 0) break; // No more packets to receive
if (pck_num == 0) { // Ready to perform the estimation
```

```
path_capacity=pck_size; //weird but last received packet has the path capac
ity in
//the packet size field.
// Dump data into a matrix and calculate available bandwidth values
trace_array=dump_array(path_capacity, pck_cnt, hmm.M, tb_array);
PrintHMM(stdout, &hmm);
//state=mean(symbol, *pair_cnt);
state=hmmest(sign, symbol, &hmm, *pair_cnt);
// Dump calculated values into a file
dump_file(argv[1], trace_array);
sprintf(snd_msg, "AvBw: %.0f Mbps\t",
(((double)10/(double)hmm.N)*state)/10000000*path_capacity);
sendto(traceband_sock, snd_msg, (strlen(snd_msg)+1), 0,
(struct sockaddr *) &snd_echo, sizeof(snd_echo));
printf("Available Bandwidth: %.2f Mbps\n",
(((double)10/(double)hmm.N)*state)/10000000*path_capacity);
//printf("Available Bandwidth: %.2f bps\n",
((double)(10/hmm.N)*state)/10*path_capacity);
173 *pck_cnt=0;
174
continue;
}
if(*pck_cnt < MAX_NUMPCK){
traceband_table[*pck_cnt].num = pck_num;
traceband_table[*pck_cnt].size = pck_size;
memcpy(&(traceband_table[*pck_cnt].snd_time), &snd_time, sizeof(struct
timeval));
memcpy(&(traceband_table[*pck_cnt].rcv_time), &rcv_time, sizeof(struct timeval));
(*pck_cnt)++; // a new packet was received
}
} // end of while loop
FreeHMM(&hmm);
free_ivector(symbol, 1, MAX_NUMPCK);
close(traceband_sock);
return 0;
}

//----- Function Definitions --------------------------
//-----------------------------------------------------
//- Create and establish the UDP socket -
//-----------------------------------------------------

int prep_sockets()
{
struct protoent *udp;
```

```
int opt;
int one=1;

udp=getprotobyname("udp");
assert(udp != NULL);
//need accurate timings on recieved packets here
bzero((void *)&snd_echo, sizeof(struct sockaddr_in));
bzero((void *)&rcv_echo, sizeof(struct sockaddr_in));

traceband_sock=socket(PF_INET,SOCK_DGRAM,udp->p_proto);
if(traceband_sock < 0){
perror("socket3:");
exit(1);
}

rcv_echo.sin_family = AF_INET;
rcv_echo.sin_addr.s_addr = htonl(INADDR_ANY);
rcv_echo.sin_port = htons(SERVER_PORT);

if(bind(traceband_sock, (struct sockaddr*)&rcv_echo,
sizeof(rcv_echo)) < 0){
perror("udp bind 2");
exit(1);

/*set SO_TIMESTAMP option*/
if(setsockopt(traceband_sock, SOL_SOCKET, SO_TIMESTAMP,
&one, sizeof(one)) < 0){
perror("setsockopt(SO_TIMESTAMP) failed rcv_echo:");
}
return 0;
}

//-------------------------------------------------------
//- Dump data from memory to a file -
//-------------------------------------------------------
void dump_file(char *file_name, trace *dumparray)
{
FILE *disp; // file to store dispersions
int i;

// Open file for writting
if ((disp = fopen(file_name, "w+")) == NULL)
DieWithError("Unable to open the trace file\n");

for( i=0 ; i < (*pair_cnt) ; i++ ) // start with the second captured packet
```

```
{
fprintf(disp, "%5.2d %8.2f %d %d %d\n", dumparray[i].timestamp,
dumparray[i].av_bw,
dumparray[i].delta_in, dumparray[i].delta_out, symbol[i+1]);
}
fflush(disp);
fclose(disp);
}

//-------------------------------------------------------
//- Dump packet data from memory to an array -
//-------------------------------------------------------
trace* dump_array(long capacity, int *cnt, int M, trace* traceband_array)
{
//const int arraysize = &cnt;
//trace *traceband_array;
double delta_in, delta_out, jitter, strain, avg, sum=0;
int i;
*pair_cnt=0;
bzero(traceband_array, sizeof(trace) * MAX_NUMPCK);
for( i=1 ; i < *cnt ; i++ ) // start with the second captured packet
{
// if (two consecutive packets) && (second one is odd))
if ((traceband_table[i].num-traceband_table[i-1].num == 1)
&&(traceband_table[i].num /2*2 == traceband_table[i].num))
{
delta_in = timeval_diff(&(traceband_table[i].snd_time),
&(traceband_table[i-1].snd_time));
delta_out = timeval_diff(&(traceband_table[i].rcv_time),
&(traceband_table[i-1].rcv_time));
jitter = max((delta_out-delta_in),0); // only positive values for jitter
strain = jitter/delta_in;
strain = strain > 1.7 ? 1.7 : strain; // according to spruce, to ignore pre
emptions
(*pair_cnt)++;
traceband_array[(*pair_cnt)-1].timestamp =
timeval_diff(&(traceband_table[i].rcv_time),&(traceband_table[0].rcv_time));
traceband_array[(*pair_cnt)-1].av_bw = capacity * (1-strain); // Available bandwidth
traceband_array[(*pair_cnt)-1].delta_in = delta_in;
traceband_array[(*pair_cnt)-1].delta_out = delta_out;
symbol[*pair_cnt] = (int) ceil(M * fabs(1-strain)); // A number from 1 to M
sign[*pair_cnt] = strain > 1 ? -1 : 1; // multiplier
return traceband_array;
}
```

```
void ReadHMM(FILE *fp, HMM *phmm)
{
int i, j, k;
fscanf(fp, "M= %d\n", &(phmm->M));

fscanf(fp, "N= %d\n", &(phmm->N));

fscanf(fp, "B:\n");
phmm->B = (double **) dmatrix(1, phmm->N, 1, phmm->M);
for (j = 1; j <= phmm->N; j++) {
for (k = 1; k <= phmm->M; k++) {
fscanf(fp, "%lf", &(phmm->B[j][k]));
}
fscanf(fp,"\n");
}
}
```

REFERENCIAS

WIKIPEDIA b. (28 de 07 de 2013). *WIKIPEDIA B*. Recuperado el 06 de 08 de 2013, de http://en.wikipedia.org/wiki/Wireshark

Aceto, G., Botta, A., Pescapé, A., & D'Arienzo, M. (2011). Unified architecture for network measurement: The case of available bandwidth. *Journal of Network and Computer Applications*, 1402-1414.

Alessio Botta, A. P. (2008). An approach to the identification of network elements composing. *Elsevier B.V. Computer Network*, 2975-2988.

Ali, A. A., & Lepage, F. (2007). IGMPS, a new tool for estimating end to end available bandwidth in IP network paths. *Networking and service IEEE*, 115-120.

Antichi, G., Giordano, S., Miller, D. J., & Moore, A. W. (2012). Enabling open-source high speed network monitoring on NetFPGA. *Network Operations and Management Symposium IEEE*, 1029-1035.

Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximazation tecnique occurring in the statistical analysis of probabilistic funtion of mrkov chains. *The annals of mathematical statistics*, 164-171.

CONASA Comunicaciones Navarra S.A. (s.f.). *Redes y Comunicaciones*. Recuperado el 7 de 10 de 2013, de http://www.conasa.es/soluciones/soluciones-de-sistemas/seguridad-y-networking/redes-y-comunicaciones

Covington, G. A., Gibb, G., McKeown, N., & Lockwood, J. W. (2009). A packet generator on the NetFPGA plataform. *Field programmable custom computing maquines IEEE*, 235-239.

Digital Design Engenier's Source. (2000). *DIGILENT Inc*. Recuperado el 3 de JULIO de 2013, de http://www.digilentinc.com/NavTop/AboutUs.cfm#info

Goldoni, E., & Schivi, M. (2010). End to end available bandwidth estimation tools, an experimental comparison. *Proceedings of TMA, IEEE*, 171-182.

Goldoni, T., & Rossi, E. (2009). Assolo, a new method for available bandwidth estimation. *Internet and protection IEEE*, 130-136.

Guerrero, C. (2013). Estimación de ancho de banda disponible por variación en la trasmisión de paquetes a través de NETFPGA.

Guerrero, C. D., & Labrador, M. A. (2006). Experimental and Analytical Evaluation of Available Bandwidth Estimation Tools. *IEEE*, 710-717.

Guerrero, C. D., & Labrador, M. A. (2009). Traceband: A fast, low overhead and accurate tool for available bandwidth estimation and monitoring. *Computer Network Elsevier*, 977-990.

Hu, N., & Steenkiste, P. (2002). Estimating Available Bandwidth Using Packet Pair Probing. *IEEE*, 27.

Hu, N., & Steenkiste, P. (2003). Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 21,*, 879-894.

Hu, N., & Steenkiste, P. (2005). Exploting internet route sharing for larga scale available bandwidth estimation. *Internet Measurement Conference ACM*, 16-20.

Jain, M. &. (2002). A measurement toolfor end-to-end available bandwidth. *In proceedings of Passive and Active Measurement* .

Jain, M., & Dovrolis, C. (2004). Ten fallacies and pitfalls on end to end available bandwidth estimation. *Internet measurement conference ACM*.

Jin, G., & Tierney, B. (2003). Nestet: A tool to measure the maximum burts size, available bandwidth and achievable throughput. *Proceedings In information Technology: Research and Education* , 578-582.

Kanungo, T. (1999). *Hmm Toolkit.* Obtenido de www.kanungo.com

Lao, L., Dovrolis., C., & Sanadidi, M. (2006). The probe gap model can underestimate the available bandwidth of multihop paths. *ACM SIGCOMM Computer Communication Review*, 29–34.

Loschmidt, P., Exel, R., & Gadere, G. (Diciembre de 2011). Highly Accurate Timestamping for Ethernet-Based. *Journal of Computer Networks and Communications*, 11.

Melander, B., Bjorkman, M., & Gunniengberg, P. (2000). A new ent to end probing and analysis method for estimating bandwidth bottlenecks. *Global Telecommunications Conference IEEE*, 415- 420.

Navratil, J., & Cottrell, R. (2003). ABING: A practical approach to available bandwidth estimation. *Passive and active measurement workshop IEEE*.

NetFPGA. (2013). *NetFPGA*. Recuperado el 15 de 07 de 2013, de www.netfpga.org

Obara, H., Koseki, S., & Selin, P. (2012). Packet Train Pair: A fast and efficient technique for measuring available bandwidth in the Internet. *SICE Annual Conference*, 1833-1836.

Orosz, P., Tamas, S., & Imrek, J. (2012). A NetFPGA based network monitoring system with multi-layer Timestamping: Rnetprobe. *IEEE*.

Quintans, C., Lago, J. M., Menéndez, L. M., & Mandado, E. (2006). Plataforma hardware para el autoaprendizaje de las FPGA y sus aplicaciones. *VII Congreso de tecnologías aplicadas a la enseñanza de la electrónica*(84.689-9590-8).

Ribeiro, V. J., Riedi, R. H., Baranjuk, R. G., Navratil, J., & Cottrell, L. (2003). Pathchirp: Efficient available bandwidth estimation for network paths. 1-11.

Salmon, G., Ghobadi, M., Labrecque, M., Ganjali, Y., & Steffan, J. G. (2009). NetFPGA based precise traffic generation. *Proc of NetFPGA Developers Workshop*.

Sharma, P. (2003). Study and Analysis of Bandwidth Flow Estimation Techniques for Wired/Wireless Networks. *International Journal,* .

Shriram, A., Murray, M., Hyun, Y., Brownlee, N., Broido, A., Fomenkov, M., & Claffy, K. (2005). Comparison of public end to end bandwidh estimation Tools on high speed links. *Passive and active measurement workshop*.

Sommers, J., Barford, P., & Willinger, W. (2006). A Proposed Framework for Calibration of Available Bandwidth Estimation Tools. *Computer and Caomunications IEEE*, 709-718.

Strauss, J., Katabi, D., & Kaashoek, F. (2003). A Measurement Study of Available Bandwidth Estimation Tools. *ACM*, 6.

Strauss, J., Katabi, D., Kaashoek, F., & Prabhakar, B. (2003). Spruce: A lightweight end to end toll for measuring available bandwidth. *Internet measurement Conference IEEE*.

Tanenbaum, A. S. (2003). *Redes de computadores* (Cuarta ed.). Mexico: Prentice Hall Inc.

Vinay Ribeiro, R. R. (2003). Pathchirp Efficient available bandwith estimation for network path. *In Passive and Active Measurement Workshop*.

Wang, Q., & Cheng, L. (2006). FEAT: Improving accuracy in end to end available bandwidth measurement. *Global Telecommunications Conference IEEE*, 1-4.

WIKIPEDIA a. (29 de 07 de 2013). *WILIPEDIA* . Recuperado el 06 de 08 de 2013, de http://en.wikipedia.org/wiki/Internet_protocol_suite

Zadnik , M. (2008). NetFlow probe on NetFPGA. *CESNET*, 7.

Zhou, Z., Cong, L., Lu, G., Deng, B., & Li, X. (Septiembre de 2010). HATS: High Accuracy Timestamping System Based on NetFPGA. *International journal of future generation communication and Networking*, 12.