PROGRAMA PROFESIONAL DE INGENIERÍA DE TELECOMUNICACIONES

Implementación de Mecanismos de Enrutamiento sin Estado en un nodo para el Tráfico Multicast MPLS en la Plataforma NetFPGA

Tesis presentada por el Bachiller en Ingeniería Electrónica y de Telecomunicaciones:

Roberto Boris Martínez Aguilar

Para optar el título profesional de Ingeniero Electrónico y de Telecomunicaciones.

Asesor: **Dr. Fernández del Carpio, Gonzalo Mauricio**

Diciembre, 2014

“¿Por qué esta magnífica tecnología científica, que ahorra trabajo y nos hace la vida mas fácil, nos aporta tan poca felicidad? La repuesta es ésta, simplemente: porque aún no hemos aprendido a usarla con tino.”

Albert Einstein, 1879-1955

Índice general

Abstract **5**

Resumen **6**

1. Introducción 7

1.1. Motivación y Contexto 10

1.2. Planteamiento del problema 12

1.3. Objetivos 13

1.3.1. Objetivo general 13

1.3.2. Objetivos específicos 13

1.4. Recursos y materiales 13

1.5. Cronograma del Proyecto 14

1.6. Metodología 14

1.7. Organización de este trabajo 15

2. Marco Teórico 16

2.1. El problema del Multicast 16
2.1.1. IP Multicast 16
2.1.2. MPLS multicast 19
2.2. Soluciones de enrutamiento multicast con Filtros de Bloom 22
2.2.1. Filtros de Bloom 23
2.2.2. Enrutamiento multicast con filtros de Bloom 25
2.3. Implementación de mecanismos multicast basados en Filtros de Bloom 25
2.3.1. Plataforma NetFPGA 25
2.3.2. Componentes de la NetFPGA 25
2.3.3. Interfaz para el software 27
2.3.4. NetFPGA packages . . . . . 27 2.3.5. Registros 27

3. Estado del Arte 29

3.1. LIPSIN y MPSS 29
3.2. D-MPSS 32
3.3. Estrategias de implementación de encaminamiento con filtros de Bloom en NetFPGA 35
3.3.1. Implementación de zFilter para LIPSIN y MPSS 35
3.3.2. Implementación de mecanismos de seguridad 36

4. Diseño 41

4.1. Configuración de las conexiones 41

4.2. Escenario MPSS 44

4.3. Escenario D-MPSS 45

4.4. Router Data Path 45

4.5. Output Port selector 46

5. Implementación, resultados y comparaciones 49

5.1. Administrador de software - PSM (PubSubManager) 49

5.1.1. Interfaz de usuario 49

5.1.2. Generación de paquetes 49

5.2. Implementación en la NetFPGA 50

5.2.1. Instalación de la NetFPGA 50

5.2.2. Programación en Verilog 50

5.2.3. Generación de un nuevo bitfile 53

5.2.4. Cargar un nuevo bitfile 53

5.3. Generación de tráfico 54

5.4. Pruebas y resultados 56

5.4.1. Tests para la implementación 56

5.4.2. Análisis de complejidad ciclomática 58

5.4.3. Mediciones del rendimiento 60

6. Conclusiones, contribuciones y trabajos futuros 66

A. Anexo 69

B. Anexo 70

C. Anexo 72

Bibliografía 75

Nomenclatura 78

Índice de figuras

2.1. Unicast vs. Multicast [Sol01]. 17

2.2. Árboles compartidos [Sol03]. 18

2.3. Árboles de distribución por emisor [Sol03]. 19

2.4. Escenario MPLS VPN [FL12]. 20

2.5. MPLS unicast [Elaboración propia]. 22

2.6. MPLS multicast [Elaboración propia]. 22

2.7. Formación del filtro de Bloom [Elaboración propia]. 24

2.8. Aparición de un falso positivo [Elaboración propia]. 24

2.9. Tarjeta NetFPGA [GNH+08]. 26

2.10. Diagrama de bloques de la NetFPGA.[LMW+07] 26

3.1. Reenvío con Zfilter[ZJS+10]. 30

3.2. Ejemplo de la construcción de la pila de filtros de Bloom [FL12]. . .33

3.3. Ejemplo de convertir BF_i, con $M = 5$ en rBF_i, con $m_i = 3$ de acuerdo a Ecuación 3.2. Los valores que estén fuera de los límites del arreglo son puestos a 0[FL12]. 34

3.4. Cabecera del paquete D-MPSS. Los subíndices indican la profundidad del árbol [FL12]. 34

3.5. En la izquierda, ruta de datos de referencia. En la derecha, ruta de datos modificada [GH11]. 36

3.6. Formato general del recorrido de los paquetes en el bus de paquetes [GH11]. 37

3.7. Estructura del modulo output_port_selector [GH11]. 39

4.1. Representación del diagrama de bloques de la plataforma NetFPGA y un Host [Elaboración propia]. 42

4.2. Esquema del proceso de envío de datos con filtros de Bloom en un host [Elaboración propia]. 43

4.3. Esquemático de la propuesta de conexiones entre dos NetFPGA para realizar pruebas entre dos hosts [Elaboración propia]. 43

4.4. Ejemplo de la creación del filtros de Bloom de acuerdo a las interfaces

de la NetFPGA. En a) con tres interfaces de salida permitidas, en b), c) y d) con dos interfaces de salida permitidas [Elaboración propia] . 44

4.5. Diagrama de la ruta de datos del usuario en la NetFPGA [NGBM08]. 46

4.6. Estructura del módulo Output Port selector para el bus de datos

[Elaboración propia]. 47

4.7. Proceso de compresión del filtro de Bloom utilizando módulo 12. Ena) filtro de Bloom de tamaño completo, en b) busqueda mediante el

módulo y en c) filtro de Bloom comprimido [Elaboración propia]. . 48 4.8. Proceso de descompresión del filtro de Bloom utilizando módulo 12.

En a) filtro de Bloom comprimido seleccionado en bloques, en b)

operaciones AND entre los bloques seleccionados según el módulo

para determinar la salida [Elaboración propia]. 48

5.1. Captura wireshark, eliminación de paquetes. 51

5.2. Captura wireshark, función de f para el cálculo de cabecera. 51

5.3. Captura wireshark, cabecera a descomprimir y comparar. 53

5.4. Mapa de la red Abilene, http://abilene.internet2.edu/. 54

5.5. Mapa de la red NSFNET, http://www.nsf.gov/. 55

5.6. Mapa de la red COST-266, http://www.cse.buffalo.edu/. 55

5.7. Grafo correspondiente al análisis de complejidad ciclomática del al-

goritmo de decisión [Elaboración propia]. 59

5.8. Diagrama de flujo de D-MPSS [Elaboración propia]. 61

5.9. Latencia promedio en la red Abilene [Elaboración propia]. 62

5.10. Latencia promedio en la red COST-266 [Elaboración propia]. 63

5.11. Latencia promedio en la red NSFNET [Elaboración propia]. 63

5.12. Cantidad de bits utilizados por nivel según el método utilizado [Ela-

boración propia]. 64

B.1. Fedora 13. 70

B.2. CentOS 5.5. 71

B.3. Resultado del Test self con solo un cable Ethernet conectando dos interfaces de la tarjeta 71

Índice de cuadros

1.1. Cronograma de trabajo. 14

4.1. Cantidad de bits formadores del filtro de Bloom con k bits en 1 para MPSS [Elaboración propia]. 44

4.2. Cantidad de bits formadores del filtro de Bloom con un mult de 8 y 32 y k bits en 1 para D-MPSS [Elaboración propia]. 45

5.1. Redes usadas en las simulaciones e implementación.[FL12] 54

5.2. Evaluación de riesgo asociado a cada valor de la complejidad ciclomática para un programa dado [McC76]. 58

5.3. Latencia promedio en la red Abilene. 62

5.4. Latencia promedio en la red COST-266. 62

5.5. Latencia promedio en la red NSFNET 62

5.6. Cálculo de latencia de varias implementaciones. 64

A.1. Cotización de materiales para complementar la tesis 69

Algoritmos

5.1. Pseudocódigo del algoritmo de decisión. 60

Abstract

The Internet Protocol Multicast (IP Multicast) is a standard that was developed to be able to support, efficiently, a large number of services and applications. However, many transport networks do not have a real multicast capability. Multiple unicast connections are used to emulate a multicast network, wasting bandwidth and growing largely routing tables. This discourages to the Internet Service Providers (ISP) to include multicast services in their networks.

Recent research works have emerged with the aim of including routing information into the packet headers, reducing drastically the resource consumption in managing routing tables with thousands of entries. These solutions use the operating principle of Bloom filters, allowing summarize routes and trees and encode them into simple arrays of bits that are inserted into the packet headers. Some of these proposals have achieved to be physically implemented in node prototypes. However, the D-MPSS (the most recent proposals) mechanism, which codifies paths/trees as hierarchical filter stacks of Bloom filters, has not yet been implemented and tested onto a prototype hardware.

This paper aims to make the implementation of a prototype node with the D-MPSS solution in the NetFPGA platform, which is designed especially for transport networks. In this design, the MPLS headers are replaced with batteries headers hierarchical Bloom filters and, using a minimum number of entries in the routing tables, it is possible to route unicast or multicast packets. The design steps towards the realization of a prototype is not a trivial problem, but involves devising assessment strategies filters, compression and decompression of them, in addition to various other aspects that represent important challenges.

The NetFPGA platform is a reconfigurable hardware device free use which can work assuming the operation of any device on the network, whether routers, switches, network interfaces, etc. It also allows present models to be modified or even create new ones, giving the user complete freedom to be programmed in it any digital design. Thus, making the implementation of the presented routing mechanisms, is highly feasible. On the other hand, obtaining results (when working at physical layer) becomes more accurate and closer to a real environment.

Resumen

El Internet Protocol Multicast (IP Multicast) es un estándar que se desarrolló para poder soportar, de manera eficiente, un gran número de servicios y aplicaciones. Sin embargo, muchas redes de transporte no cuentan con una capacidad multicast real. Para emular una red multicast, se utilizan varias conexiones unicast, malgastando ancho de banda y haciendo crecer en gran medida las tablas de enrutamiento. Esto desalienta a los Proveedores de Servicios de Internet (ISP) de incorporar servicios multicast en sus redes.

Recientes investigaciones han surgido con el objetivo de incluir la información de las rutas en las cabeceras de los paquetes, reduciendo así drásticamente el consumo de recursos en la gestión de tablas de enrutamiento con miles de entradas. Estas soluciones utilizan el principio de funcionamiento de los filtros de Bloom, los cuales permiten resumir rutas y árboles y codificarlos en simples arreglos de bits que se insertan en las cabeceras de los paquetes. Algunas de éstas propuestas se han llegado a implementar físicamente en prototipos de nodos. Sin embargo, el mecanismo DMPSS (la más reciente de las propuestas), que consiste en codificar en una pila de filtros de Bloom jerárquicos las rutas/árboles de los paquetes, aún no ha sido implementado ni probado en un prototipo de hardware.

El presente trabajo tiene como objetivo realizar la implementación de un prototipo de nodo con la solución D-MPSS en la plataforma NetFPGA, la cual ha sido diseñada especialmente para redes de transporte. En este diseño, se reemplazan las cabeceras MPLS por cabeceras con pilas de filtros de Bloom jerárquicos y, usando un mínimo número de entradas en las tablas de enrutamiento, es posible encaminar los paquetes unicast o multicast. El paso del diseño hacia la concreción de un prototipo no es un problema trivial, sino que implica el idear estrategias de evaluación de los filtros, la compresión y descompresión de los mismos, además de diversos otros aspectos que significan importantes retos.

La plataforma NetFPGA es un dispositivo de hardware reconfigurable de uso libre, que puede trabajar asumiendo el funcionamiento de cualquier tipo de dispositivo de la red, ya sean enrutadores, conmutadores, interfaces de red, etc. También permite que se modifiquen los modelos preestablecidos o incluso crear nuevos, dándole completa libertad al usuario que desee programar en ella cualquier diseño digital. De esta manera, realizar la implementación de los mecanismos de enrutamiento presentados, resulta ser altamente viable. Por otro lado, la obtención de resultados (al trabajar a nivel de capa física) llega a ser mucho más precisa y cercana a un entorno real.

1. Introducción

Internet Protocol Multicast (IP Multicast) es un estándar que se desarrolló para dar soporte, de manera eficiente, un gran número de servicios y aplicaciones que hoy en día -con la importancia del Internet- son indispensables en la comunicación entre uno o más emisores y múltiples receptores. Dichas aplicaciones requieren entregar determinada información a un subconjunto de subscriptores, como es el caso de las videoconferencias, el aprendizaje a distancia (e-learning), la difusión de televisión por Internet (IPTV), la compartición de archivos, las actualizaciones de software, la sindicación de noticias (RSS), el video bajo demanda (VoD) y la computación grid. Otro caso conocido es el de los videojuegos en línea de múltiples roles (MMORPG: Massively Multiplayer Online Role-Playing Game). El caso más importante es el juego World of Warcraft que cuenta con alrededor de 10 millones de subscriptores [Bli14] y tiene la capacidad de admitir grandes cantidades de jugadores en simultáneo mediante el uso de varios servidores asignados a cada región virtual dentro del juego. En los MMORPG todos los movimientos de los jugadores deben ser informados (diseminados) a sus vecinos, lo cual requiere altas capacidades de envío multicast.

A pesar de que las aplicaciones mencionadas se pueden dar bajo el soporte de un modelo IP punto a punto (unicast), replicando los paquetes a enviar desde el origen hacia todos los destinos, esto tiene la gran desventaja de generar copias de los paquetes y desperdiciar así ancho de banda en todos los enlaces. El elevado consumo del ancho de banda, a medida que el número de destinatarios aumente, convierte a esta posibilidad en algo inviable.

A diferencia del modelo unicast, la principal ventaja del IP multicast (en que sólo se hacen copias de los paquetes en los puntos de ramificación en el árbol multicast) es la disminución del tráfico en la red, ya que permite que la fuente envíe una sola copia del mensaje, usando una sola dirección de grupo. Los encaminadores (routers intermedios) entre las fuentes y los destinos utilizarán la dirección del grupo para trasladar los paquetes hacia su destino. Así, serán los routers los encargados de hacer copias de los paquetes, bastando con asegurar que todos los miembros reciban una copia de los mismos.

En la actualidad, gran cantidad de equipos soportan el manejo de paquetes IP multicast, sin embargo no existe un amplio desarrollo de este servicio. Una de las razones de esto es que los esquemas multicast convencionales son escalables para atender el tráfico de grupos multicast grandes, pero al trabajar con mayores números de grupos multicast diferentes, se presentan problemas de escalabilidad. Así mismo, otras razones que impiden el desarrollo de IP multicast, y que desaniman a los proveedores

de servicios de Internet (ISP) de incorporarlo, son la cantidad de información de reenvío requerida en los nodos intermedios y los potenciales problemas de seguridad que pueden surgir.

Los problemas de escalabilidad aparecen debido a que IP multicast utiliza una estructura de entrega tipo árbol, la cual necesita que todos los nodos mantengan información de reenvío específica para cada grupo. Dicha información de reenvío crecerá linealmente con el número de grupos. Además la información contenida en la cabecera de los paquetes aumentará al incluir campos de control, que son necesarios para permitir el establecimiento y mantenimiento de los árboles multicast. Esto hace que se necesite una mayor cantidad de memoria de estado en los nodos encaminadores y que la búsqueda de una dirección determinada tenga mayor latencia. Dada la cantidad de ventajas que podría proveer un soporte verdaderamente multicast, ha surgido el interés de vencer estas limitaciones ideando mecanismos para reducir la cantidad de información enviada en las cabeceras. Para ello es necesario considerar distintas tecnologías y mecanismos que permitan lograr este objetivo sin perder calidad de servicio y así poder aumentar el desempeño de una red. Así, los autores en [JZR09] propusieron LIPSIN, una arquitectura para el reenvío de paquetes multicast en sistemas de publicación/subscripción. Esta propuesta es ingeniosa, utiliza como base el funcionamiento de los filtros de Bloom [Blo70] para codificar en la cabecera del paquete la ruta/árbol multicast que se debe seguir. Propuestas anteriores con una idea similar consistían también en maneras de hacer esta codificación en la cabecera del paquete, pero sin la utilización de filtros de Bloom. Todas estas propuestas se resumen en que buscan eliminar o reducir la información de estado de reenvío en los nodos encaminadores.

Los inconvenientes del multicast no atañen únicamente a Internet sino también a otros tipos de redes. Por ejemplo, queda resolver el problema de Virtual Private LAN Services (VPLS), que emula la capacidad broadcast/multicast de una LAN privada virtual sobre una red de transporte de gran extensión. En este contexto, reducir la información de estado en los nodos intermedios (tablas de enrutamiento) resulta fundamental para reemplazar el sistema de replicación de los paquetes al ingreso de la red (ingress replication), que consume mucho ancho de banda, por uno verdaderamente multicast. Adicionalmente, hoy existe una importante penetración de aplicaciones multicast basadas en VPNs (Virtual Private Networks), las cuales deben garantizar a sus clientes conectividad sobre una infraestructura IP compartida.

Es importante considerar que en el protocolo de Internet las direcciones de los puntos de conexión y los protocolos de encaminamiento aseguran que los equipos de comunicación conozcan la ruta hacia estos puntos de conexión, sin embargo IP no nombra las rutas, sólo los nodos. MPLS (Multi-Protocol Label Switching) cambia esto introduciendo una etiqueta de tamaño fijo. De esta manera el protocolo de encaminamiento asegura que los nodos de borde conozcan que etiqueta usar para determinado destino y que los nodos

intermedios simplemente sigan las instrucciones de ruta, indicadas en la etiqueta. Esto mejora la velocidad de reenvío, la realización Introducción

de servicios de Traffic Engineering (TE) de forma natural y el soporte para varias aplicaciones tales como VPNs, VPLS, etc. Así, MPLS ha obtenido una enorme aceptación en las capas 2 y 3 de las redes de transporte.

Sin embargo, al hacer multicast con MPLS [ER], el número de etiquetas (y, por consiguiente, el número de entradas de reenvío en los nodos) se multiplica por cada árbol de encaminamiento calculado para cada grupo, y por cada una de las ramas que se desprenden en cada nodo. El problema se agrava cuando el número de árboles es grande en comparación al número de nodos. Así es que se propuso el mecanismo de encaminamiento unicast/multicast MPSS [ZJS+10] (Multiprotocol Stateless Switching) que, de la misma forma que en LIPSIN [JZR09], consiste en codificar el camino/árbol, que cada paquete sigue, mediante un filtro de Bloom (BF)[Blo70] que se utiliza en lugar de la etiqueta MPLS del paquete.

Para reducir en estos sistemas las anomalías de reenvío que se presentan debido al uso de los filtros de Bloom [SRA+11], los autores en [FL12] desarrollaron una propuesta de mejora denominada D-MPSS (Depth-Wise Multi-Protocol Stateless Switching), convirtiendo la idea original en un mecanismo más escalable, reduciendo las anomalías de reenvío y la cantidad de bits necesarios en la cabecera de los paquetes.

A pesar que los mecanismos propuestos han sido demostrados de forma teórica y mediante simulaciones, hay una carencia de aportes desde el punto de vista de la implementación real de estos esquemas. Es decir, es visible la necesidad de implementar bancos de pruebas en equipos de hardware reales, en los que se puedan observar el comportamiento, la eficiencia y los parámetros que caracterizan a estos mecanismos. La implementación en hardware de estos protocolos de comunicación propuestos no es un proceso trivial, se requiere solucionar problemas y tomar decisiones en el sistema de reenvío, tales como el procesamiento de las cabeceras (que contienen los filtros de Bloom), que pueden llegar a superar el tamaño del bus de datos del nodo, y las cuales además pueden estar estructuradas en pilas (como es el caso de D-MPSS). Sería posible además, contar con información precisa y real del cálculo de la latencia, número de búsquedas que se realizan en cada salto, uso de recursos, velocidad de procesamiento, etc. Siendo necesario controlar la cantidad de procesos que se dan en cada salto de ciclo de reloj, descompresión de cabeceras, eliminación de datos innecesarios y optimización del rendimiento en el nodo.

En la literatura se han desarrollado muy pocas implementaciones relacionadas [KJS09]y [JZR09], y en ese sentido la presente tesis pretende constituirse en una contribución, sobre todo conociendo la importancia que tiene el problema del multicast. En este trabajo desarrollaremos la implementación de los modelos basados en filtros de Bloom sobre la plataforma NetFPGA [LMW+07, GNH+08], la cual es una plataforma de hardware

reconfigurable de compuertas y circuitos lógicos, de bajo costo y que contiene las herramientas necesarias para construirse un nodo encaminador (router). Mediante la programación a bajo nivel de compuertas lógicas, este dispositivo de hardware puede reproducir prácticamente todas las características de un equipo

de comunicaciones de alta velocidad de conmutación, llegando a velocidades de procesamiento del orden de 1 Gbps, es decir, al mismo nivel de velocidad de los equipos que trabajan con fibra óptica. De esta manera se pretende obtener resultados que corroboren la efectividad de los métodos mencionados, así como dar oportunidad a realizar mejoras que sólo en la implementación real se pueden plantear.

A continuación ahondaremos más en el problema de las comunicaciones multicast y las diferentes tecnologías existentes involucradas. Luego formularemos la justificación y la descripción del problema, para finalmente exponer los objetivos de la presente tesis.

1.1. Motivación y Contexto

De acuerdo a lo revisado anteriormente, diferentes servicios y aplicaciones (difusión de contenidos audiovisuales, emulación de redes LAN, etc.) requieren cada vez más del soporte multicast. Este tipo de tráfico se encuentra en creciente demanda, y por eso la solución al problema del multicast se ha convertido en una gran necesidad para las tecnologías de telecomunicaciones. Actualmente, los operadores de telecomunicaciones prefieren replicar conexiones punto a punto en lugar de dar un soporte verdaderamente multicast, para evitar, entre otras cosas, que las tablas de enrutamiento crezcan, o que se produzcan ataques de denegación de servicio.

Tal como hemos visto, desde hace ya algunos años se han venido planteando diversas soluciones a este problema, en particular, para el escenario de una red de transporte, como MPLS. En general, todas estas propuestas buscan un equilibrio razonable entre evitar generar mucha información de estado de reenvío (lo que es típico en multicast MPLS) y no desperdiciar ancho de banda, es decir, un equilibrio entre eficiencia y capacidad.

El ámbito de este trabajo se centra en un escenario de red VPN BGP/MPLS [RR06] ,el cual permite a los proveedores de servicios proporcionar una red VPN a sus clientes. En este escenario MPLS se usa para el reenvío de los paquetes sobre la red troncal, y BGP se utiliza para la distribución de rutas. Múltiples VPNs se interconectan a la red troncal y cada VPN cuenta con un dispositivo CE (Customer Edge) que se encarga del tráfico interno de la misma y establece adyacencia con el PE (Provider Edge) luego de establecer adyacencia, anuncia las rutas locales de la VPN y aprende rutas remotas desde el PE.. El enrutador PE intercambiará información de enrutamiento con el enrutador CE, cada enrutador PE

mantiene una VRF(Virtual Routing and Forwarding) para cada uno de los dispositivos con los que este directamente conectado. En el núcleo de la red, se encontraran los enrutadores P, los cuales manejarán el tráfico de la red e interconectarán a los dispositivos PE. Los enrutadores P funcionan como LSR (label switch router, un tipo de encaminador utilizado en el medio de una red MPLS responsable de la conmutación de etiquetas para el envío de paquetes) de MPLS. Este escenario permite que usuarios ubicados

en distintos lugares puedan accesar a una red como si ésta estuviera en el mismo edificio, campus, establecimiento, etc.

Recientemente, ha ocurrido un giro muy interesante en estas propuestas hacia la eliminación prácticamente total de toda información de estado de reenvío en los nodos intermedios. Esto se viene haciendo mediante el uso de los filtros de Bloom, los cuales, ingeniosamente adecuados, hacen las veces de etiquetas de encabezado en los paquetes multicast, guardando en sí mismos la información del árbol que el paquete tiene que recorrer. En cada salto, la cabecera es evaluada con una operación simple (como AND), y así el nodo reconoce por cuales interfaces de salida el paquete debe continuar. De este modo se eliminan (como ocurriría en MPLS multicast) las consultas a las tablas de intercambio de etiquetas, pero el problema es que, como consecuencia de trabajar con filtros de Bloom, para evitar el incremento de ocurrencias de falsos positivos (evaluaciones positivas de paquetes por interfaces de salida equivocadas), las cabeceras deben ser más grandes (512, 800 bits, etc.) La mayor parte de estos sistemas han sido propuestos en forma teórica y comprobados mediante simulaciones. Sin embargo, existen muchos retos particulares a la implementación de estas propuestas en hardware real que merecen ser tratados. Uno de estos retos es el procesamiento de las largas cabeceras, que podrían exceder por mucho la capacidad del bus de datos en cada ciclo de reloj.

Podemos concluir entonces que, en general, es muy importante concentrarse en el problema de la implementación en hardware de este tipo particular de mecanismos multicast. El despliegue de una plataforma de pruebas (testbed) sería de mucha utilidad para observar las verdaderas prestaciones de las nuevas e interesantes propuestas que recientemente han cambiado en mucho el paradigma que usualmente se tenía. En el camino de esta implementación será posible mejorar y perfeccionar el procesamiento de encaminamiento y reenvío de los paquetes, reduciendo la latencia y la eficiencia de aspectos tales como el uso de la memoria volátil (para el encolamiento propio del análisis de la cabecera), y el aprovechamiento de la velocidad del procesador.

La compresión y descompresión de los filtros, es también un reto importante que se presenta al profundizar las distintas soluciones .Ya que mientras menos información tengan que buscar los nodos intermedios en las cabeceras de los paquetes, su procesamiento será más sencillo y rápido. La compresión puede reducir significativamente el tamaño de los filtros de Bloom enviados, sin embargo en los nodos

intermedios también será necesario realizar la descompresión de los mismos para luego hacer la comparación respectiva. En este punto es importante considerar la cantidad de recursos que se consumirá en este proceso y de qué manera se realizará. En cuanto a la lectura de las cabeceras de los paquetes, una forma de evitar la latencia producida por el procesamiento, es aprovechando el paralelismo que lenguajes de descripción de hardware poseen, para realizar operaciones simultáneas. Tal es el caso que en vez de hacer comparaciones una tras otra, se pueden realizar al mismo tiempo siempre y cuando no excedan los recursos del dispositivo. Al aprovechar el paralelismo se

ahorran muchos ciclos de reloj reduciendo la latencia total.

La implementación de las distintas propuestas presentadas permite crear un escenario de pruebas más cercano a la realidad. Así mismo con una implementación real se pueden obtener mediciones precisas de la latencia generada por el procesamiento de los paquetes, permitiendo de esta manera verificar el rendimiento de estas propuestas comparadas con las utilizadas actualmente por los ISPs. Por último la utilización de una plataforma, que a su vez trabaje en bajo nivel, crea la necesidad de superar inconvenientes que solo en la implementación se pueden observar, como pueden ser: la división de los paquetes para su ingreso en el bus de datos, el tiempo de la compresión, el procesado entre otros que en la teoría solo se pueden suponer.

La NetFPGA es una plataforma abierta y disponible para los desarrolladores de todo el mundo, permite crear diseños personalizados en hardware, logrando así aplicaciones más cercanas a la realidad en materia del procesamiento y envío de paquetes dentro de una red. La NetFPGA permite a los investigadores construir prototipos de trabajo de alta velocidad y sistemas de redes de aceleración de hardware. Así mismo se puede utilizar como herramienta para enseñar a estudiantes como manejar conmutadores Ethernet y protocolos de Internet. También permite el desarrollo de aplicaciones de bajo nivel en el procesamiento de paquetes. Es así que la NetFPGA se vuelve en un instrumento indispensable para investigadores y estudiantes ya que al ser abierta facilita su modificación para aplicaciones específicas de determinados proyectos sin necesidad de recurrir a tecnologías propietarias.

1.2. Planteamiento del problema

En la actualidad, los servicios multicast están empezando a ser más utilizados debido a sus múltiples aplicaciones en video y audio streaming, broadcasting, videojuegos multijugador entre otros. Sin embargo muchas redes de transporte no tienen una capacidad multicast real, ya que esto implicaría un mayor consumo de recursos en los nodos intermedios de una red a consecuencia de saturar las cabeceras con todas las rutas existentes en la red. Para ello han surgido recientes investigaciones con el objetivo de

reducir significativamente la información de las rutas en las cabeceras de los paquetes IP multicast. En los trabajos realizados se puede apreciar el uso de Filtros de Bloom para realizar el reenvío multicast. De esta manera la cabecera del paquete contiene un Filtro de Bloom que es evaluado en cada salto para emparejarlo con el identificador del enlace de salida correspondiente. Mediante los Filtros de Bloom se logran buenos resultados en cuanto a la reducción de cabeceras pero también se dan falsos positivos en los resultados, este problema se soluciona en otros trabajos que mejoran dicho método, así como también problemas de escalabilidad y otras anomalías han sido superados. Gran cantidad de las investigaciones mencionadas presentan resultados eficientes, sin embargo hasta el momento no han sido implementadas y no se puede percibir su impacto real.

1.3 Objetivos

1.3. Objetivos

1.3.1. Objetivo general

El objetivo de la presente tesis es la implementación real de un enrutador a nivel de hardware, utilizando la plataforma NetFPGA, con los mecanismos de enrutamiento multicast MPSS y D-MPSS (los cuales no utilizan información de estado, evitando así la sobrecarga en la búsqueda de las entradas en las tablas de enrutamiento típica de los conmutadores MPLS), aplicando estrategias para alcanzar la mejor eficiencia en el proceso de conmutación del nodo aprovechando las características de los filtros de Bloom.

1.3.2. Objetivos específicos

1. Estudiar los mecanismos de enrutamiento multicast sin estado que hacen usode los filtros de Bloom, que se encuentran en el estado del arte.
2. Profundizar en el entendimiento y dominio de la plataforma NetFPGA parala implementación de mecanismos de enrutamiento de paquetes.
3. Desarrollar el escenario de pruebas para verificar las estrategias de conmutación mediante filtros de Bloom tratados en este trabajo.

1.4. Recursos y materiales

Para la elaboración de este trabajo de investigación, es necesario contar con los siguientes recursos y materiales, la mayoría de ellos existentes y disponibles en el Programa Profesional:

- 01 Tarjetas NetFPGA (Se cuenta con 2 tarjetas adicionales para trabajos futuros y un mejor estudio de la plataforma).
- 01 PC con slots PCI estándar o PCI-X, para albergar la tarjeta NetFPGA, con sistema operativo Fedora 13 para NetFPGA (de distribución gratuita) o CentOS 5.5 (para diseños de referencia más antigüos) y con un puerto GbE (Gigabit Ethernet).
- 01 NIC (Network Interface Card) Dual GbE para poder realizar los distintos tests de regresión de cada diseño de referencia.
- Software Xilinx (para FPGA) y ModelSim.

1.5 Cronograma del Proyecto

1.5. Cronograma del Proyecto

En el siguiente cuadro se presenta un cuadro con el resumen de las actividades planeadas para la elaboración de la presente tesis:

Actividad	Inicio	Fin
Investigación preliminar del problema	25/08/13	25/09/13
Estado del arte	25/09/13	10/10/13
Definición del problema	10/10/13	20/10/13
Estudio de las especificaciones de NetFPGA	20/10/13	12/01/14
Diseño de estrategias de solución	12/01/14	20/04/14
Implementación	20/04/14	01/08/14
Pruebas	01/08/14	20/10/14
Análisis	20/10/14	7/11/14
Revisiones, elaboracion del documento final y presentación	7/11/14	10/12/14

Cuadro 1.1.: Cronograma de trabajo.

1.6. Metodología

La presente tesis se desarrolla de manera secuencial y presenta en primer lugar una introducción para exponer las ideas principales que se ampliarán a medida que se avance en la investigación. A continuación ya con un marco teórico establecido y con ayuda del estado del arte se pretende realizar el diseño de un enrutador en la plataforma NetFPGA con las características necesarias que faciliten la comprobación de la eficiencia del método presentado. Una vez realizado el diseño se puede llevar a cabo la implementación en una tarjeta NetFPGA luego de la correcta instalación y puesta a punto del software respectivo. Luego, con la implementación realizada se puede proceder a la experimentación, es decir

pasar tráfico de datos desde un ordenador hacia la tarjeta por uno de sus puertos Ethernet. Por último es posible comprobar mediante varias pruebas si los resultados son los esperados generando diferentes escenarios donde se pueden presentar problemas reales, registrando cada uno y dando las soluciones adecuadas.

Para realizar las pruebas de enrutamiento se utilizará una tarjeta NetFPGA, como se mencionó anteriormente, la cual representará a un nodo intermediario. Los cuatro puertos GbE (Gigabit Ethernet) de la tarjeta actuarán como interfaces de entrada y salida de tal manera que el nodo pueda recibir y enviar tráfico. Para corroborar el envío y la recepción de paquetes, se conectarán dos interfaces de la NetFPGA a los dos puertos correspondientes a una Dual NIC alojada en la misma PC. Un puerto de la Dual NIC enviará tráfico hacia la primera interfaz de la NetFPGA para ser procesado y luego enviado desde una segunda interfaz de la tarjeta hacia el segundo

puerto de la Dual NIC. Al realizar este tipo de conexiones, se puede verificar el envío de paquetes y también estimar la latencia producida por el procesamiento del tráfico, restando el momento en el que se recibió menos el momento en el que se envió.

Para procesar los datos que ingresan a la tarjeta, se plantea dividir las cabeceras que contienen los filtros de Bloom en bloques de 64 bits. De esta manera los filtros de Bloom pueden ingresar por el bus estándar de 64 bits de la NetFPGA por cada ciclo de reloj; es decir que para procesar un filtro de Bloom de 512 bits, se necesitará de 8 ciclos de reloj. Los filtros de Bloom están formados únicamente de ceros y unos distribuidos de manera tal que la cantidad de 0's (ceros) sea mucho mayor que la de 1's (unos) en la cabecera. Una estrategia que se propone para reducir ciclos de reloj es aprovechar el paralelismo que ofrece la programación en Verilog al momento de realizar las operaciones necesarias. Hacer la descompresión y comparación de los filtros de Bloom, con las interfaces de salida mediante las propiedades del módulo, facilita el ahorro en el consumo de recursos, evitando la realización de cálculos innecesarios.

1.7. Organización de este trabajo

En este capítulo se han presentado los aspectos generales de esta tesis, introduciendo en primer lugar, la problemática, la motivación y el contexto para el desarrollo del trabajo. En la Sección 2.1, Sección 2.2 y Sección 2.3 se ha hecho un repaso sobre las bases teóricas necesarias para entender la importancia del problema a resolver.

Seguidamente, este documento está estructurado de la siguiente forma. El estado del arte, en el Capítulo 3, presenta en forma concreta las últimas contribuciones que se extraen de los trabajos relacionados con el presentado en esta tesis. El Capítulo 4 desarrollará la propuesta de solución (diseño) y todos los aspectos acerca de las

estrategias que se toman para dar solución al problema. Seguidamente, en el Capítulo 5 se detallarán los aspectos de implementación del protocolo en la plataforma NetFPGA, indicando las pruebas y los resultados obtenidos. Finalmente este trabajo será culminado en el Capítulo 6, mediante la exposición de las conclusiones y los posibles trabajos futuros.

2. Marco Teórico

2.1. El problema del Multicast

La diferencia entre un servicio unicast y uno multicast es que en el primero existe un emisor y un receptor de cada datagrama, mientras que en el segundo hay un emisor pero normalmente son varios los destinatarios del mismo. Son numerosas las aplicaciones que se basan en un servicio multicast, por ejemplo: teleconferencia (audio, video, pizarra compartida, editor de texto), juegos compartidos y simulaciones distribuidas, Internet TV, aplicaciones distribuidas, transferencias fiables de ficheros y localización de un servidor/servicio entre otras.

Aunque este tipo de transmisión podría conseguirse con el envío de datagramas unicast (punto a punto) a cada uno de los posibles nodos de destino, hay numerosas razones que hacen aconsejable la capacidad multicast.

En la Figura 2.1 se puede observar, por ejemplo, que en el caso unicast se utiliza un mayor ancho de banda para la comunicación del origen a los destino.

2.1.1. IP Multicast

Multicast es un servicio muy popular usado principalmente por las empresas en sus redes IP. Permite la distribución eficiente de información entre una fuente multicast y múltiples receptores. Un ejemplo de fuente multicast en una red corporativa sería un servidor de información financiera provisto por una tercera compañía como Bloomberg's ó Reuters. En este caso, los receptores serían un conjunto de PCs repartidos por la red, todos ellos recibiendo la misma información financiera del servidor. El servicio multicast permite a la fuente transmitir una sola trama de información independientemente del número de receptores deseosos de recibirla. Los enrutadores implicados en el proceso de transmisión, automáticamente replican una sola copia de la trama por cada uno de los interfaces desde donde se puede alcanzar a receptores del servicio. Por lo tanto, el servicio multicast reduce significativamente la cantidad de tráfico requerido para distribuir una misma información a un conjunto de clientes interesados en ella [AGS12].

La dirección destino de un paquete multicast es siempre una dirección de grupo multicast. Esta dirección se encuadra dentro del bloque IANA 224.0.0.0–239.255.255.255 (antes de que existiera el concepto de "classless interdomain routing" ó CIDR, este rango de direcciones era conocido como clase D). Una fuente transmite un paquete

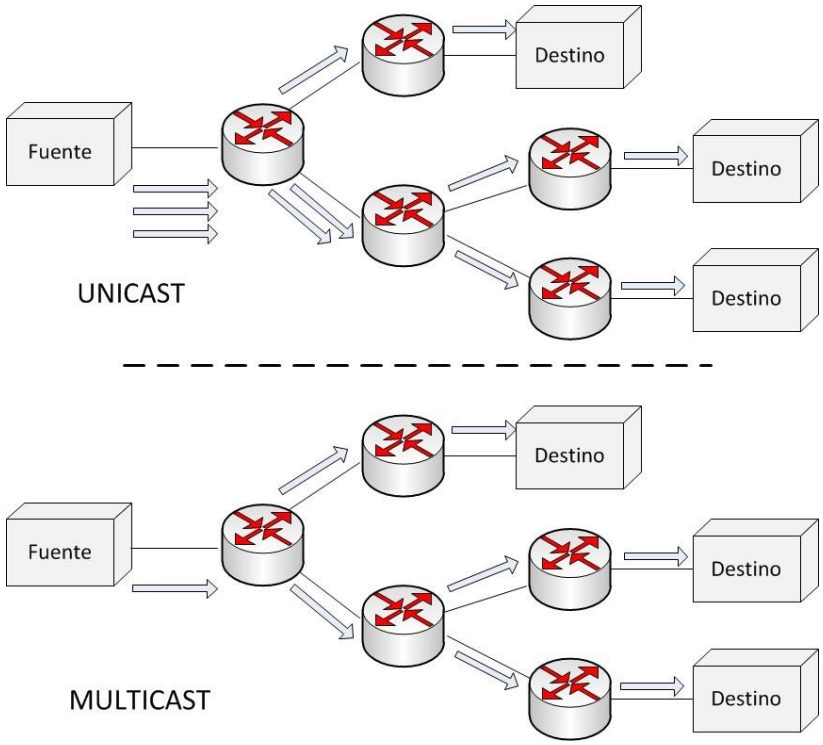

Figura 2.1.: Unicast vs. Multicast [Sol01].

multicast usando una dirección de grupo multicast, mientras que un conjunto de receptores "escuchan" el tráfico procedente de esa misma dirección de grupo.

Los paquetes multicast son reenviados a través de la red utilizando un árbol de distribución multicast. La red es responsable de replicar el mismo paquete en cada punto de bifurcación (el punto en el cual las ramas se dividen). Esto significa que una sola copia del paquete viaja sobre un determinado enlace en la red, haciendo de los árboles multicast una forma extremadamente eficiente de distribuir la misma información a varios receptores.

Árboles de Distribución Multicast

Los enrutadores multicast crean árboles de distribución para controlar la ruta que toma el tráfico multicast a través de una infraestructura de red para poder entregarlo. Los árboles de distribución consisten en dos tipos básicos: árboles compartidos y árboles de distribución por emisor [Sol01] y [Sol03].

Árboles compartidos: En este planteamiento la raíz del árbol es un nodo común situado en algún lugar de la red, ver Figura 2.2. Este nodo común se conoce como *Rendezvous Point* (RP). El RP es el punto al que los receptores se registran para recibir la información de las fuentes activas. Las fuentes multicast deben transmitir su tráfico al RP. Cuando los receptores se unen a un grupo multicast en un árbol compartido, la raíz de ese árbol es

siempre el RP, y el tráfico multicast es transmitido desde el RP hacia los receptores. Por lo tanto, el RP se comporta como un punto de enlace entre las fuentes y los receptores. Un RP puede ser la raíz de todos los grupos multicast activos en la red, o también existe la posibilidad de que diferentes rangos de grupos multicast se asocien con distintos RPs.

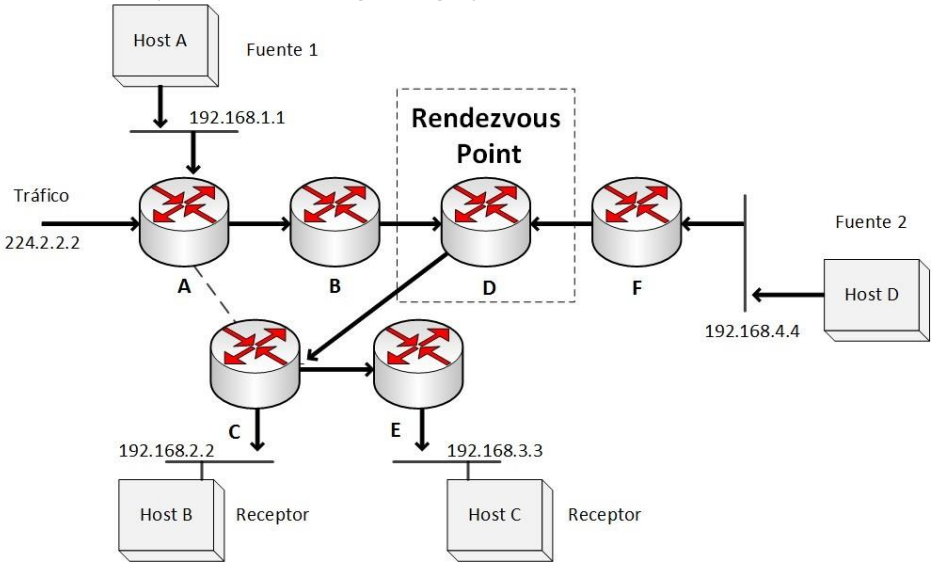

Figura 2.2.: Árboles compartidos [Sol03].

Este modelo de árbol de distribución no es tan óptimo en cuanto al proceso de enrutamiento como el anterior. Esto se debe a que todo el tráfico originado por las fuentes debe viajar al RP y entonces seguir el mismo camino hacia los receptores. Sin embargo, la cantidad de información de estado de enrutamiento multicast requerida es menor que la que se tendría en el caso de estar usando un árbol de distribución por emisor.

Árboles de distribución por emisor: Se trata de la forma más simple de árbol de distribución. La fuente de la información se encuentra situada en la raíz del árbol, y los receptores se localizan al final de las ramas. El tráfico multicast viaja desde la fuente, atravesando el árbol, hasta llegar a los receptores. La decisión sobre la interfaz por dónde el paquete debe ser transmitido se basa en una tabla de reenvío multicast. Esta tabla consiste en una serie de entradas de estado multicast que se almacenan en el enrutador. En la Figura 2.3 se representa un árbol de distribución por emisor.

Este tipo de árboles lleva implícito el hecho de que el camino entre la fuente y los

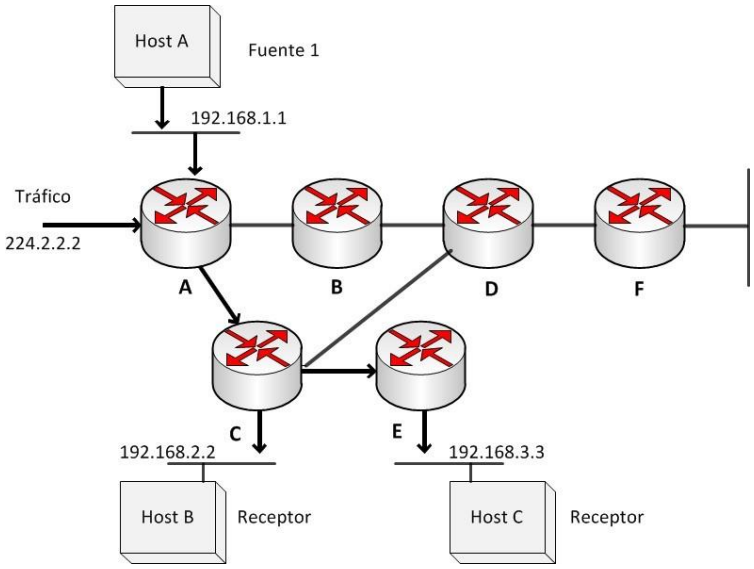

Figura 2.3.: Árboles de distribución por emisor [Sol03].

receptores es el camino más corto disponible y por esa razón, también son conocidos como árboles de camino más corto (shortest path trees SPTs). A cada una de las fuentes que transmiten paquetes multicast se les asigna un árbol en exclusiva, incluso en el caso de que estén transmitiendo información para el mismo grupo. Esto significa que existirá una entrada de estado para cada uno de las fuentes activas en la red. Por lo tanto, los árboles SPTs proporcionan un enrutamiento óptimo a costa de una mayor información de estado a mantener en la red.

La elección entre un modelo de árbol u otro depende de los recursos disponibles en la red. Como ya hemos indicado en los párrafos anteriores, el modelo de distribución por emisor ofrece una mayor fiabilidad y un menor retardo, pero tiene una mayor complejidad y necesita una mayor información de estado.

2.1.2. MPLS multicast

Una red BGP MPLS basada en VPNs [RR06] de un proveedor de servicios (SP) está formada por enrutadores proveedores (P) en el núcleo y de enrutadores de borde (PE) interconectados a dispositivos de borde del cliente (CE), que son enrutadores o conmutadores ubicados en el lado de los usuarios. Para demandas unicast o multicast, una vez que sea calculado el árbol o camino, los paquetes (en el caso de L3VPN[1]) o las tramas Ethernet (en el caso de L2VPN[2]) son etiquetados por el PE de ingreso y enviados

[1] Enrutadores/conmutadores de la red troncal del SP reenvían los paquetes basándose en información de nivel 3 (p.e., la dirección IP)

[2] Enrutadores/conmutadores de la red troncal del SP reenvían los paquetes basándose en información de nivel 2 (DLCI y MAC)

por caminos conmutados por etiquetas (LSP) punto a punto o punto a multipunto [ER], y el reenvió de paquetes es realizado intercambiando etiquetas en cada salto, indiferente a su contenido, un escenario de este tipo se puede observar en la Figura 2.4.

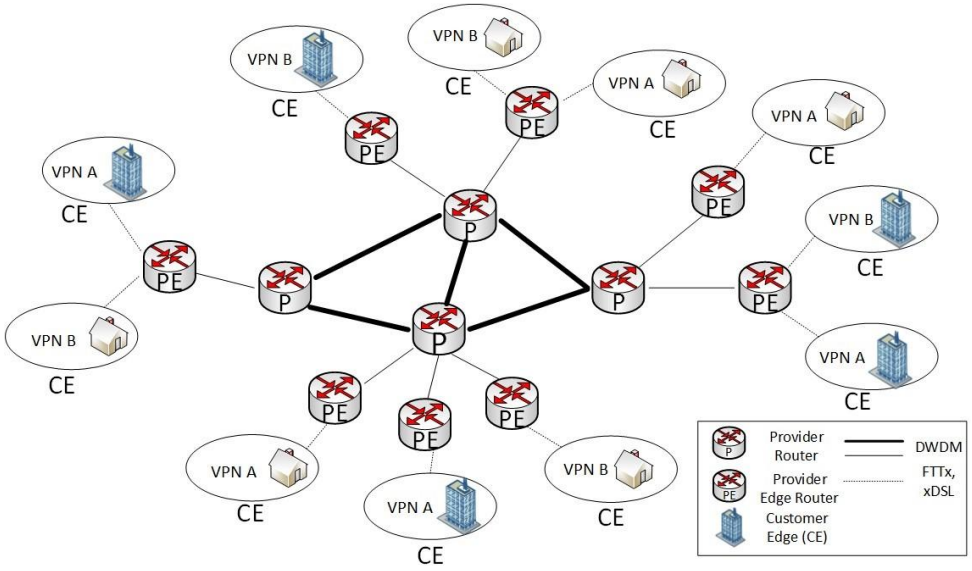

Figura 2.4.: Escenario MPLS VPN [FL12].

En redes de transporte no es posible usar IP multicast porque el número de grupos, cada uno de ellos formado por árboles con gran cantidad de nodos, es mucho mayor. Además las redes de transporte, para brindar una calidad de servicio determinada y constante, realizan generalmente conmutación de circuitos, por lo que suelen estar basadas en MPLS ó ATM (Asynchronous Transfer Mode), en la parte de núcleo de la red.

En el caso de una red basada en MPLS, se introduce una etiqueta de tamaño fijo para nombrar a cada ruta ubicada entre los nodos intermedios. Para realizar un correcto encaminamiento de los datos, los nodos intermedios poseen tablas de etiquetas, que se encargarán de hacer una correspondencia entre la información que entra al enrutador y la que sale. Es decir, una vez que los nodos intermedios reciben un paquete con una etiqueta específica, identifican la interfaz por la que están entrando y revisan en la tabla de etiquetas por donde deben salir. Adicionalmente a este proceso, los nodos deberán retirar la etiqueta con la que entró el paquete y reemplazarla por una nueva que le permita continuar hacia su destino. En MPLS unicast si se desea enviar la misma información a más de un usuario, se debe enviar un paquete por cada usuario, cada uno irá por una ruta distinta por lo tanto tendrá una etiqueta distinta.

En la Figura 2.5 se presenta un caso unicast MPLS en el que el enrutador intermedio (B) recibe dos paquetes iguales con diferentes etiquetas (cada paquete posee dos campos, uno para la etiqueta y otro para los datos). El nodo B, verifica la etiqueta y la interfaz por la que llegó, luego revisa en su tabla por donde debe salir. En este caso ambos paquetes llegan por la interfaz 0 con etiquetas 17 y 20 respectivamente. Según la tabla, el paquete que ingresó con la etiqueta 17, deberá salir por la interfaz 1, con la etiqueta 50 y el que

ingresó con la etiqueta 20, deberá salir por la interfaz 2, con la etiqueta 10. Una red MPLS multicast, a diferencia de una unicast solo necesitará enviar un único paquete, ya que el nodo intermedio se encargará de replicarlo y enviarlo por las interfaces de salida correspondientes según su tabla. La Figura 2.6 muestra el caso anterior para una red MPLS multicast, el nodo B sólo recibe un paquete por la interfaz 0, consulta su tabla e inmediatamente replica el paquete para luego enviar uno por la interfaz 1 y el otro por la 2. Así mismo agrega una etiqueta diferente, 50 y 10 respectivamente, a cada paquete para que continúe su camino. Ambos métodos cumplen la misma función, sin embargo con MPLS multicast, no es necesario saturar la red con paquetes repetidos, que a una escala mayor pueden representar serios problemas en el rendimiento de los nodos intermedios. En una VPN BGP/MPLS, el enrutamiento unicast para los paquetes de una VPN se realiza sin la necesidad de mantener ninguna información de estado sobre las VPNs que hacen uso del servicio en los enrutadores de la red troncal (P) del proveedor de servicio. La información de enrutamiento de una VPN concreta se guarda sólo en los enrutadores de borde del proveedor (PE), que se conectan directamente con los sitios de esa VPN. Los datos del cliente viajan a través de los nodos P en túneles desde un nodo PE a otro. De esta manera, para ofrecer el servicio de VPN, los nodos P sólo necesitan conocer rutas a los nodos PE.

El enrutamiento PE-PE es óptimo, además, la información de estado asociado en los enrutadores P, depende sólo del número de PEs y no del número de VPNs activas. No obstante, si se ofrece un enrutamiento multicast óptimo (en párrafos siguientes se definirá este concepto) para un flujo multicast en particular, los enrutadores P a través de los que ese flujo viaja, necesitan mantener el estado específico de ese flujo. La escalabilidad sería pobre si la información de estado en los enrutadores P fuese proporcional al número de flujos multicast en las VPNs. Por lo tanto, cuando se ofrece servicio multicast para VPNs BGP/MPLS, existe un compromiso entre la escalabilidad de la red de enrutadores P y la optimización del enrutamiento multicast.

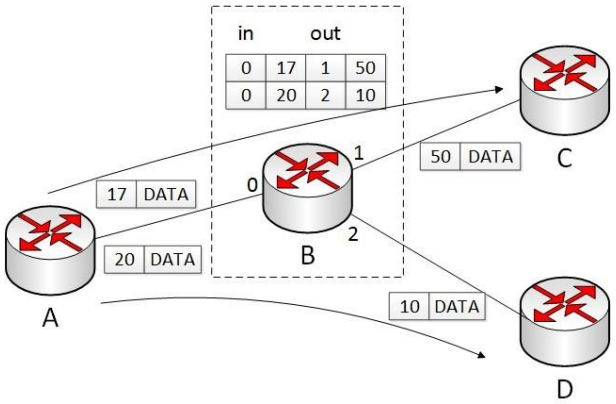

Figura 2.5.: MPLS unicast [Elaboración propia].

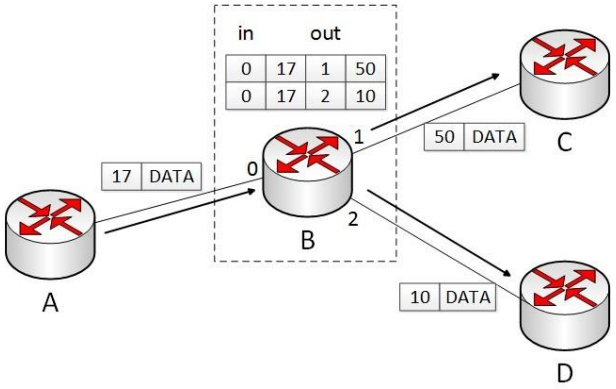

Figura 2.6.: MPLS multicast [Elaboración propia].

2.2. Soluciones de enrutamiento multicast con Filtros de Bloom

Hay diversos métodos que permiten reducir la información de estado en los enrutadores, los trabajos de investigación presentados en [CKM+05] y [MYLS07] buscan dar solución a este problema en redes IP-MPLS, usando principalmente el concepto de agregación de demandas multicast a un grupo reducido de árboles; eliminando así la información de estado para el reenvío de paquetes en el núcleo de la red. Sin embargo en este enfoque hay una pérdida inherente de ancho de banda, debido a que todas las demandas individuales comparten un solo súper-árbol.

2.2.1. Filtros de Bloom

Los filtros de Bloom [Blo70], introducidos por Burton Bloom en 1970, son una estructura de datos probabilística que puede almacenar grandes sistemas, simple y eficientemente

con un mínimo consumo de memoria y con una pequeña probabilidad de falsos positivos. Los filtros de Bloom se pueden utilizar en diversas aplicaciones de red, tales como el almacenamiento de caché distribuida, redes P2P/overlay, enrutamiento de recursos, enrutamiento de paquetes, y en infraestructuras de medición [BM03]. Recientemente, los investigadores han estudiado las aplicaciones de los filtros de Bloom para redes ad hoc, para la aceleración de búsquedas de caché [PPL05], la gestión de grupos [LSSI05], y el punto de acceso basado en el rastreo en reversa [HL05].

Muchos problemas ocasionados en IP multicast se deben a la saturación de los nodos intermedios con tablas de estado muy extensas, por lo cual se han abierto distintas investigaciones para reducir e incluso eliminar dichas tablas. Un método ampliamente eficiente es el uso de filtros de Bloom (BF), utilizado en [FL12], [ZJS+10] y [JZR09], portado en la cabecera de los paquetes.

Un filtro de Bloom, para representar un conjunto $S = s_1, s_2, ..., s_n$ de n elementos, es descrito por una matriz de m bits inicializados en 0 como valor predeterminado.

Un filtro de Bloom utiliza k funciones hash independientes $h_1, ..., h_k$ en un rango de $0, ..., m - 1$. Por conveniencia matemática se realiza la suposición natural de que estas funciones hash asignan a cada elemento del universo a una serie aleatoria uniforme en el intervalo $0, ..., m - 1$. Para cada elemento $s \in S$, los bits $h_i(s)$ se establece en 1 para $1 \leq i \leq k$. Una ubicación puede ser establecida en 1 varias veces, pero sólo el primer cambio tiene un efecto. Para comprobar si un elemento x está en S, se verifica si todos los $h_i(x)$ se establecen en 1. Si no, entonces x no es un miembro de S. Si todos los $h_i(x)$ se establecen en 1, se asume que x está en S, aunque con alguna probabilidad de error. Por lo tanto un filtro Bloom puede producir un falso positivo, donde se sugiere que un elemento x está en S a pesar de que no lo esté. Para muchas aplicaciones, esto es aceptable, siempre y cuando la probabilidad de un falso positivo sea lo suficientemente pequeña [Mit01].

Para introducir elementos al filtro de Bloom y hacer las consultas en caso sea necesario deben existir funciones hash. Las funciones hash son algoritmos que se utilizan para representar, en una salida alfanumérica, un conjunto finito de información a partir de determinada entrada a dicha función. En la Figura 2.7 a modo de ejemplo se muestra la creación de un filtro de Bloom. Dos funciones hash representadas por E1 y E2 son asignadas a dos elementos cualesquiera de determinado conjunto. Se realiza entonces una operación lógica OR entre ambas funciones hash, el resultado de esta operación será un filtro de Bloom de las mismas dimensiones que las funciones hash individuales. En un principio todas las posiciones del filtro estarán establecidas en cero. Si se deseara agregar un elemento más al filtro, bastará con realizar otra operación OR entre la función hash del elemento y el propio filtro. Para comprobar si un elemento se encuentra en el

conjunto, se ejecuta la misma operación OR entre la función hash del elemento a saber y el filtro, si el resultado es la misma función hash, entonces el elemento pertenece al conjunto. Sin embargo otros elementos ajenos al conjunto pueden ser reconocidos como propios, como se muestra en la Figura 2.8, al filtro del ejemplo anterior se le hace pasar un elemento representado por la función hash E3. E3 no pertenece al conjunto con el que se creó el filtro, pero al realizar un OR entre E3 y el filtro, nos da la misma función hash, es decir, el filtro lo considera como miembro del conjunto. Este error en el reconocismiento de algunas funciones hash se denomina probabilidad de falso positivo.

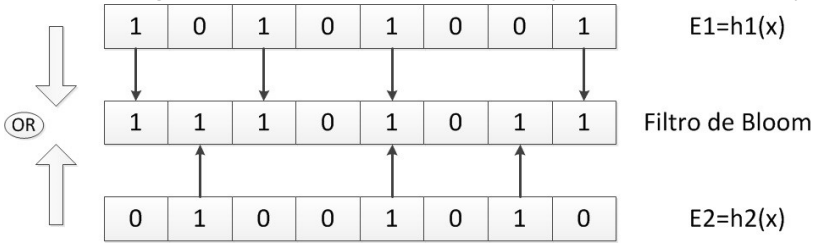

Figura 2.7.: Formación del filtro de Bloom [Elaboración propia].

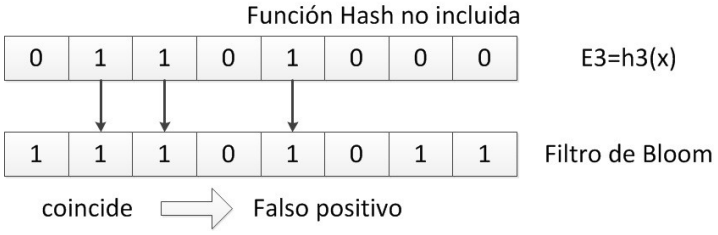

Figura 2.8.: Aparición de un falso positivo [Elaboración propia].

La probabilidad de un falso positivo o tasa de falsos positivos (*fpr*) para un elemento fuera del conjunto, se puede calcular de una manera sencilla, dada la suposición de que las funciones de hash son perfectamente aleatorias. Después de que a todos los elementos de *S* se les aplique el código hash en el filtro de Bloom, la probabilidad de que un bit específico sea aún 0 es:

$$1 - \frac{1}{m}^{kn} \approx e^{-\frac{kn}{m}} \quad (2.1)$$

Haciendo $p = e^{-\frac{kn}{m}}$. La tasa de falsos positivos(*fpr*) es entonces:

$$1 - 1 - \frac{1}{m}^{k} \approx 1 - e^{-\frac{kn}{m}k} = (1 - p)^{k} \quad (2.2)$$

Nótese que para valores más grandes de m la fpr disminuye, pero con un conjunto mayor de elementos a ser insertados n, se incrementa.

2.2.2. Enrutamiento multicast con filtros de Bloom

Se asignan unos identificadores pseudoaleatorios a cada enlace unidireccional y la cabecera del paquete resultara de realizar una operación OR entre todos los identificadores de enlace del árbol. La decisión de reenvió en cada nodo es solo una operación de comparación entre los BF de los identificadores de enlace de sus interfaces de salida y los paquetes.

Los BF son capaces de eliminar información de estado, ya que los nodos intermedios solo necesitan almacenar la información de cada enlace. No obstante bajo esta solución existe la aparición de falsos positivos, por ejemplo en el caso que un identificador de enlace se empareja con el BF aun si no había sido añadido intencionalmente a este. Es así como se generan anomalías de reenvió a manera de tormentas de paquetes, bucles de reenvió y flujos duplicados [SRA+11]. Para reducirlos de una manera probabilística, se proponen mejoras recientes en [FL12] y [JZR09]. Además otro problema aún por solucionar es que con el fin de reducir la tasa de falsos positivos, los BF aumentan su tamaño para grandes árboles multicast, creando así una sobrecarga en la cabecera. La implementación de muchos trabajos como el visto en [JZR09] también es un reto importante para conseguir resultados reales que permitan mostrar problemas que no aparecen en las simulaciones.

2.3. Implementación de mecanismos multicast basados en Filtros de Bloom

2.3.1. Plataforma NetFPGA

En la Figura 2.9 se muestra la plataforma NetFPGA .El Centro de la NetFPGA está formado por un gran dispositivo Xilinx FPGA. Alrededor de la FPGA existen 4 dispositivos de memoria- dos RAM estáticas (SRAMs) y dos dispositivos DDR2. En el lado izquierdo, se proporciona un transmisor-receptor de capa física de cuatro puertos que permite enviar y recibir paquetes por medio de cuatro cables estándar de Ethernet. Al lado derecho de la tarjeta, dos conectores SATA (Serial Advanced Technology Attachment) sobre la plataforma permiten a múltiples NetFPGAs dentro de un sistema, intercambiar datos a alta velocidad usando un bus PCI incluido.

2.3.2. Componentes de la NetFPGA

Un diagrama de bloques detallado de los componentes de la plataforma de la NetFPGA es mostrado en la Figura 2.10:

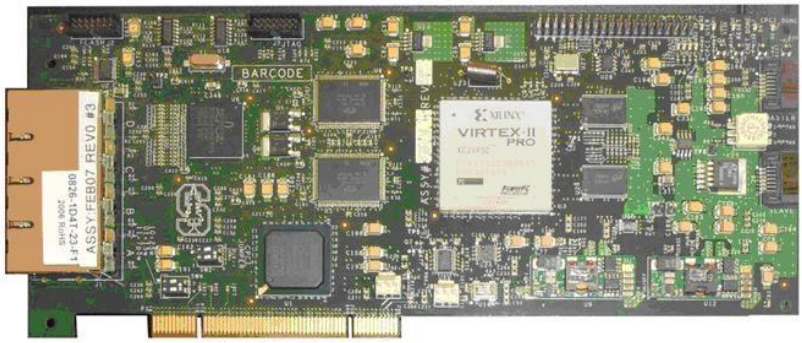

Figura 2.9.: Tarjeta NetFPGA [GNH+08].

- Una tarjeta PCI estándar que se puede conectar a una ranura PCI de un ordenador de sobremesa u otro dispositivo. Permite reconfigurar la Virtex-II Pro, y por lo tanto la NetFPGA, directamente desde el PC, sin la necesidad de un cable JTAG. Además proporciona una forma de comunicación entre el PC y la FPGA mediante el acceso a registros y a memoria.

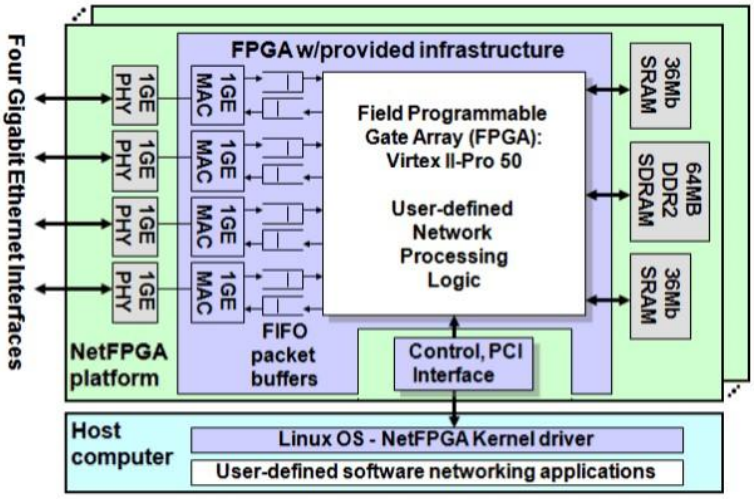

Figura 2.10.: Diagrama de bloques de la NetFPGA.[LMW+07]

- Una Spartan 2, denominada CPCI (Control del bus PCI). Como su propio nombre indica, es la encargada de controlar la comunicación entre la Virtex II y el bus PCI.

Es una FPGA de menor capacidad transparente para el desarrollador ya que, una vez configurada la memoria flash, no es necesario preocuparse por ella.

- Una FPGA modeloVirtex-II Pro 50a la que se denominada CNET (chip de Control de la NetFPGA). Mediante la programación de esta FPGA se consigue que la plataforma NetFPGA sea reconfigurable. Cargando en ella distintos bit files la NetFPGA se comporta como una tarjeta de red, un router, etc.
- Cuatro puertos Gigabit Ethernet.
- Memoria SRAMde 4,5 MBytes. Memoria
- DDR2de 64 MBytes.

2.3.3. Interfaz para el software

El NetFPGA kernel driver permite a un usuario en un PC, enviar y recibir paquetes por medio de los puertos Ethernet de la tarjeta. El driver también permite que programas aplicativos escriban o lean los registros asignados en memoria que corresponden a los registros en el chip, SRAM y DRAM. En total, 128 MB de espacio de direcciones en host, están reservados para cada tarjeta NetFPGA instalada en un ordenador.

2.3.4. NetFPGA packages

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

Un ejemplo de paquete NFP es el Reference Router. Éste contiene el conjunto de fuentes que permiten configurar la Net FPGA para que funcione como un router. Existen tres formas en que los desarrolladores pueden usar los paquetes NFPs que se presentan a continuación;

- Modificar el software sin necesidad de tocar el hardware.
- Comenzar con el diseño inicial e implementar alguna mejora o nueva funcionalidad.
- Crear un paquete nuevo con todo el diseño hardware y software.

2.3.5. Registros

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

1. Software: registros en los que se escribe desde el PC y de los que se lee desdela FPGA.
2. Hardware: registros en los que se escribe desde la FPGA y de los que se leedesde el PC.

Usando estos registros es posible leer y escribir datos en la NetFPGA desde el PC utilizando un driver apropiado que se comunica con la NetFPGA a través del PCI. Además la plataforma proporciona una serie de librerías en C con las funciones para acceder a dichos registros.

3. Estado del Arte

3.1. LIPSIN y MPSS

La arquitectura Multiprotocol Label Switching (MPLS) ha llegado a ser una verdadera historia de éxito en el mundo de las telecomunicaciones, por la eficiencia del reenvío y la baja latencia producto de las búsquedas en las tablas de encaminamiento. Sin embargo, como ya comentamos anteriormente, MPLS no es eficiente cuando se habla de multicast. Si se opta por atender las demandas de grupos multicast utilizando LSPs (Label Switch Paths) punto a punto, se producirán grandes pérdidas de ancho de banda. Por otro lado, si se opta por configurar árboles verdaderamente multicast (con el uso de multicast MPLS [RFC 6513[ER]]), la cantidad de entradas de estado (información de las rutas de los paquetes) en las tablas de reenvío afectan grandemente al rendimiento de los nodos de la red.

Debido a que al proporcionar VPNs Multicast, los operadores necesitan incrementar el uso de ancho de banda con la cantidad de estados multicast, sacrificando eficiencia. El envío con filtros de Bloom en la cabecera de los paquetes ofrece la oportunidad de tener elementos de red casi sin estado; la cantidad de estados de reenvío no depende del número de rutas/árboles que participan en el nodo.

En el trabajo [ZJS+10] se introduce Multiprotocol Stateless Switching (MPSS), se afirma que es el matrimonio entre MPLS y el mecanismo de reenvío basado en filtros de Bloom. MPSS es una arquitectura de reenvío de paquetes, flexible y sin estado. Esta arquitectura hereda la flexibilidad de MPLS y brinda a los operadores la oportunidad de ofrecer servicios VPN multicast, evitando el difícil proceso de puesta a punto en el intercambio entre el uso de ancho de banda y el estado.

Como se muestra en LIPSIN [JZR09], rutas y árboles de origen se pueden codificar de manera eficiente en la cabecera del paquete usando filtros Bloom (BF) [Blo70] y nombrando enlaces localmente en lugar de nodos. Una vez que se determina la ruta o árbol, el identificador de reenvío, llamado zfilter (ZF), se forma mediante la compresión del conjunto de identificadores de enlace en el filtro de Bloom.

Un identificador de enlace es una cadena de longitud fija de m-bits de largo, con k bits establecidos en uno (1), donde $k << m$, y m es relativamente grande. Con m=256 y k=5, se tiene: $m!/(m - k)! \approx 1012$ diferentes identificadores de enlace, haciendo a los identificadores de enlace estadísticamente únicos (asumiendo que los k bits se distribuyen aleatoriamente). Cada nodo tiene una función $Z(L,I)$, que se

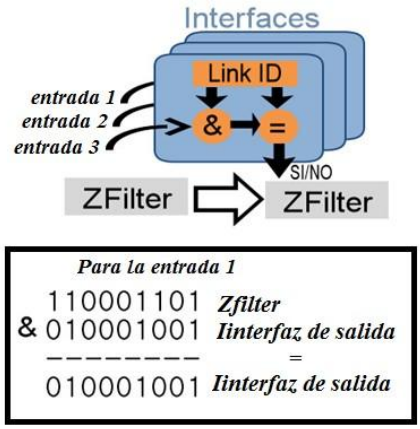

Figura 3.1.: Reenvío con Zfilter[ZJS+10].

utiliza para calcular el identificador de enlace basado en la información local L y alguna información I a partir de la cabecera del paquete.

Para construir un zfilter, se calcula un OR binario sobre todos los identificadores de enlace que forman el árbol T. la fuente utiliza el resultado del ZT ZFilter para enviar paquetes a lo largo del árbol de la entrega T. Un ejemplo de la toma de decisión con filtros de Bloom se observa en la Figura 3.1 donde un nodo recibe un paquete con un filtro de Bloom en su cabecera; para determinar si el paquete puede continuar su recorrido o si es descartado se realizan operaciones de emparejamiento entre la cabecera con filtro de Bloom y cada interfaz de salida, representada por un identificador de enlace, mediante un AND lógico (representado por &). Si la respuesta de la operación es igual a la interfaz de salida, el paquete puede continuar caso contrario se descarta, el mismo procedimiento se realiza con todas las interfaces del nodo. Al costado derecho el primer sumando representa la cabecera con filtro de Bloom y el segundo representa la interfaz de salida, al operarlos con un AND lógico el resultado es igual a la interfaz de salida lo que indica que el paquete puede salir por esa interfaz.

LIPSIN: Line Speed Publish/Subscribe Inter-Networking

En este a trabajo [JZR09] basándose en la idea de colocar un filtro de Bloom en los paquetes de datos, se propone una nueva estructura de reenvió para trafico Multicast. Con cabeceras razonablemente pequeñas comparadas con ipv6, se puede manejar gran parte de grupos Multicast con distribución Zipf, de hasta 20 subscriptores en una topología de escala metropolitana sin añadir ningún estado en la red y con una sobrecarga de reenvío insignificante. Para el resto del tráfico, el enfoque proporciona la capacidad de balancear envíos múltiples sin estado con enfoques con estado. Con

3.1 LIPSIN y MPSS

el enfoque Multicast con estado se puede manejar grupos Multicast densos (con muchos nodos) con una muy buena eficiencia de reenvió. Las decisiones de reenvió son simples, eficientes, paralelizadas en hardware y tienen atractivas propiedades de seguridad. Todos estos atributos hacen este trabajo una potencial elección para aplicaciones de centros de datos.

Según se indica en el trabajo, si bien, aún queda mucho por hacer, los resultados indican que puede ser factible para soportar Multicast masivo de forma escalable a lo largo del Internet. Técnicamente los principales obstáculos que quedan están relacionados con la determinación del árbol de entrega correcto para el tráfico que llega desde fuera del dominio.

Para obtener los resultados se implementó dos prototipos uno en FreeBSD (nodo final) y otro basado en la NetFPGA de Stanford(nodo de transmisión). En la implementación FreeBSD se realizó una API (Application Programming Interface) publish/subscribe usando la memoria virtual. Con la implementación basada en NetFPGA, el reenvío de paquetes individuales sólo tarda unos 25 microsegundos, en comparación con un cable de bucle de retorno normal, lo que sugiere un plano de reenvío de paquetes muy rápido.

VPNs Multicast Basadas en MPSS

Dentro de MPSS multicast se tiene un plano de control el cual es responsable de dos tareas, La primera es la construcción de caminos y árboles dentro de la red del proveedor. La otra tarea es de anunciar a los clientes la información de enrutamiento a través de la red del proveedor. Como se utiliza etiquetas MPLS para demultiplexar PEs en la salida, el plano de control de las implementaciones VPN actuales se puede utilizar fácilmente. Para anunciar caminos unicast, BGP no requiere extensiones adicionales. Como resultado, los PEs se distribuyen las rutas de las redes de los clientes, y los VRFs (Virtual Routing and Forwarding) serán establecidos correctamente.

En el caso de multicast, entre las opciones que las especificaciones actuales ofrecen, podría elegirse un protocolo de distribución por subscripción que permita el seguimiento explícito de los receptores. Las fuentes de los PEs podrían estar siempre al tanto del receptor PE para un grupo multicast dado, y por lo tanto, siempre se puede utilizar un ZFilter apropiado para la transmisión dentro de la red.

Plano de datos:

Cuando un PE recibe un paquete IP de unidifusión destinado a un sitio a distancia, comprueba su correspondiente VRF, y determina la salida PE y la etiqueta interior. Se adhiere la etiqueta interior para el paquete, y un zfilter para alcanzar el PE específico. Los routers P reenviarán el paquete basado en el ZFilter. Cuando se alcanza la salida PE, se elimina el zfilter y comprueba la etiqueta MPLS. En base a la decisión, se envía el paquete IP original a su correcta red de destino.

Multidifusión funciona de una manera similar. Cuando se recibe un paquete de multidifusión IP destinado a redes remotas, el PE etiqueta con una etiqueta interior comunicada por el plano de control y un zfilter para alcanzar el PE con los receptores, o alternativamente, el PE que sirve de punto de encuentro para el grupo particular.

3.2. D-MPSS

Con el fin de reducir el estado ocasionado por el reenvío multicast en redes conmutadas de paquetes, diversas propuestas han sido estudiadas en la literatura. Uno de los recientes enfoques estudiados es el de MPSS [ZJS+10], en el cuál la ruta (unicast) o el árbol (multicast) es codificado mediante un filtro de Bloom portado en la cabecera de cada paquete. Este tipo de método de reenvío hace que sea posible eliminar el estado de reenvío en los nodos de red. En [FL12] se propone una mejora a MPSS a la vez que se resuelve anomalías de reenvío observadas en enfoques multicast basados en filtros de Bloom. En ésta propuesta se codifica el árbol multicast en una pila de filtros de Bloom de longitud variable agrupados en una cabecera, representando a un conjunto de interfaces de salida (de cada nodo que conforme el árbol) a una profundidad (nivel) de árbol determinada. Así mismo se prueba que este enfoque es más eficiente que MPSS sobre multicast, especialmente en grandes redes, y reduce el alcance de las anomalías de reenvío derivadas de falsos positivos: tormentas de paquetes, bucles de reenvío y duplicación de flujo. Además, el procesamiento de paquetes es más simple y el tamaño de la cabecera promedio se reduce.

Este método se da en un escenario de red BGP MPLS basado en VPN como se ve en secciones anteriores y en el desarrollo de MPSS. Las limitaciones encontradas para los enfoques basados en filtros de Bloom son las siguientes:

- El conjunto de elementos a ser codificados podría ser demasiado grande (n= número total de enlaces del árbol).
- Esto implica la necesidad de grandes BFs, para mantener la fpr baja.
- Incluso los BFs relativamente grandes no pueden conseguir que $fpr = 0$.
- Los falsos positivos cercanos a la parte superior del árbol tienden a causar más severamente anomalías de reenvió (las secuencias de bucle y los duplicados solo son limitados por el TTL).
- Tratando de enfrentar estos problemas, a continuación se presenta la arquitectura D-MPSS (Depth Wise MPSS).

Filtros de Bloom apilables

Esta idea plantea la reducción del espacio donde se insertan elementos en el BF a medida que el paquete se distribuye hacia las partes bajas del árbol. Para cumplir este objetivo, se crean BFs individuales para cada nivel. Así un filtro de Bloom BF_i representa los enlaces pertenecientes al nivel de profundidad i-ésimo del árbol como se muestra en Figura 3.2. Los niveles individuales de los BFs en el árbol pueden ser muy pequeños (ya que no tienen un gran número de enlaces a insertar).

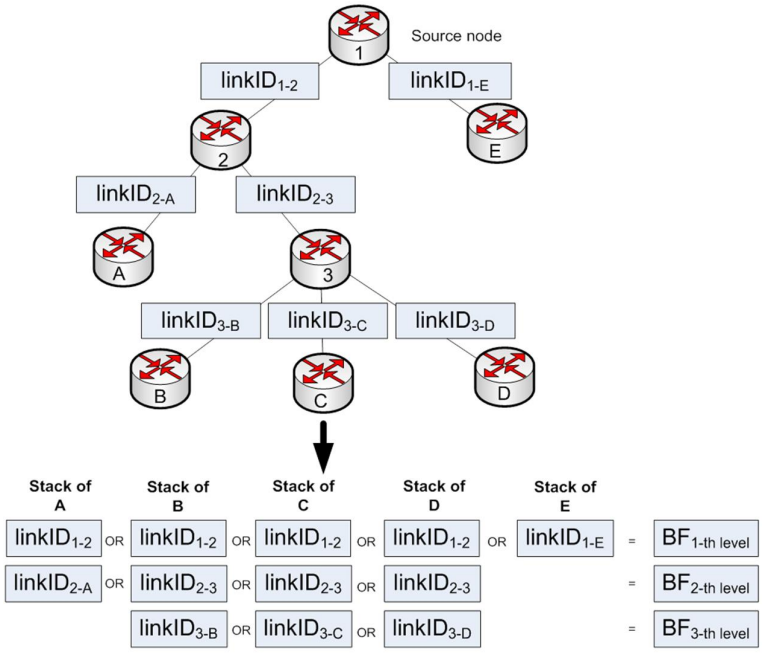

Figura 3.2.: Ejemplo de la construcción de la pila de filtros de Bloom [FL12].

Variando el tamaño de los filtros de Bloom individuales

La cantidad de enlaces en cuanto más profundo es el árbol aumenta, a diferencia de lo que sucede en los niveles superiores. Por lo tanto se podría utilizar menos bits para la codificación de los niveles superiores. En este trabajo se propone utilizar BF de longitud variable, mediante el uso de un BF por nivel en lugar de un BF para todo el árbol. En primer lugar, el BF_i (BF del i-ésimo nivel de profundidad del árbol) se crea como un gran BF de tamaño M (para ello los identificadores de enlace deben haber sido creados con longitud M, que puede tomar valores iguales a 512, 800, o

1024 bits, etc.) Después se obtiene p_i (factor de llenado de BF_i) al dividir el número de bits establecidos en 1 entre el número de bits en 0 del filtro de Bloom, entonces el tamaño óptimo para el BF para la profundidad i es:

$$m_i = d - M \lg_2(1 - \rho_i) e \tag{3.1}$$

Un filtro de esta longitud resultaría en un factor de llenado de 50%. En la Figura 3.3 se tiene un ejemplo de conversión de BF_i en el filtro reducido rBF_i. La posición aésima del arreglo del BF reducido de longitud m_i tiene que ser

$$rBF_i[a] = \bigvee_{j=1}^{dM/m_i e} BF_i[(a-1) \cdot m_i + j] \tag{3.2}$$

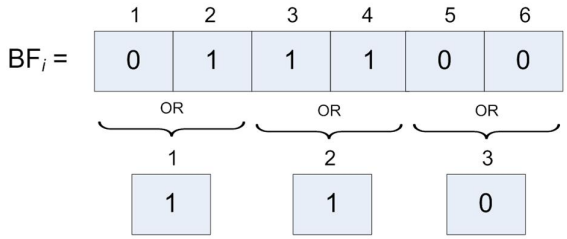

Figura 3.3.: Ejemplo de convertir BF_i, con $M = 5$ en rBF_i, con $m_i = 3$ de acuerdo a Ecuación 3.2. Los valores que estén fuera de los límites del arreglo son puestos a 0[FL12].

De esta manera, se construye una pila de h rBFs, cada uno con un tamaño óptimo reducido. Es necesario añadir algunos bits de señalización antes de cada rBF_i con el fin de indicar su correspondiente tamaño m_i. Para ahorrar bits en la cabecera del paquete, sólo los múltiplos de mult (por ejemplo $mult = 32$) son aceptados como valores de cualquier m_i. Entonces $m_i = f_i \cdot mult$ (donde f_i corresponde al i-ésimo nivel de profundidad del árbol), y f_h (corresponde al rBF_h más grande) necesitará sólo $\log_2 fh+1$ bits para ser representado. La cabecera del paquete tiene la estructura mostrada en la Figura 3.4.

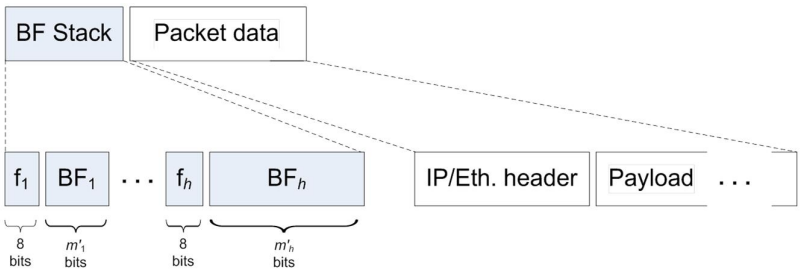

Figura 3.4.: Cabecera del paquete D-MPSS. Los subíndices indican la profundidad del árbol [FL12].

Reenvío de Paquetes

La pila compuesta de $rBF_i,...,rBF_h$ es computada y puesta en la cabecera del paquete por el nodo origen. En cada salto en el i-ésimo nivel de profundidad del árbol, sólo rBF_i es evaluado para operaciones de chequeo; luego, el rBF_i y su correspondiente campo f_i son removidos. Sin embargo, podría darse el caso que el PE destino tenga que eliminar todos los rBFs que aún permanecen en la pila. Esto ocurrirá para todos los nodos PEs que están a una distancia más corta del nodo origen que la profundidad h del árbol. En el contexto de MPSS y D-MPSS, el nodo PE es finalmente capaz de leer la cabecera interna con la información de reenvío de VPN para enviar el paquete a los nodos de borde del cliente (CE).

Para el chequeo del BF, el BF_i tiene que extenderse hasta el tamaño de M duplicando los bits en las posiciones adecuadas. Luego si hay una coincidencia, el paquete es copiado, el rBF_i (el filtro de la parte superior) es descartado, y el paquete resultante es reenviado al siguiente salto.

3.3. Estrategias de implementación de encaminamiento con filtros de Bloom en NetFPGA

3.3.1. Implementación de zFilter para LIPSIN y MPSS

En este paper [KJS09], que es una mejora del visto en [JZR09], se describe la implementación de un nodo de reenvió sin IP basado en NetFPGA. La implementación requiere de la eliminación de la IP tradicional y reemplazarla mediante técnicas de emparejamiento con filtros de Bloom para realizar las decisiones de reenvió.

Para validar las afirmaciones se implementó el prototipo de un nodo de reenvió y se probó su funcionamiento. Con algunas mediciones en el nodo de reenvió se concluyó que el procesamiento de la NetFPGA para un Zfilter crea un retraso de 3 µs. Este retraso podría ser reducido ya que la operación de reenvío debe tomar sólo 64ns (8 ciclos de reloj). En comparación con la implementación con un Router IP se realizó la implementación mediante peticiones de eco ICMP(Internet Control Message Protocol), mostrando que la realizada con zFilter es ligeramente más rápida que las basadas en el reenvió de IP, ejecutándose en la misma NetFPGA.

Esta implementación aún carece de algunas características avanzadas descritas en [JZR09] por ejemplo la creación de rutas inversas y la señalización. Sin embargo, no será muy complicado añadir tales características al diseño existente. Además, los estudios iniciales

indican que debería ser posible comenzar a añadir incluso características más avanzadas, como el almacenamiento en caché, corrección de errores o el control de congestión para la implementación.

3.3.2. Implementación de mecanismos de seguridad

En el capítulo 5 de la tesis presentada en [GH11]y [GN10], se describe su implementación en la plataforma NetFPGA. Para este trabajo se ha tomado la implementación de Zfilter basada en nodo de reenvio en una NetFPGA [GNH+08] y se optimizó según las necesidades del mismo.

Se desarrollan dos implementaciones para el cálculo, una utilizando el cifrado de flujo Moustique y la otra mediante el uso de cifrado de bloques AES. En cada caso, los ID de Enlace se calculan utilizando i) La información en los paquetes (I), ii) una clave cambiada periódicamente (K_3) y iii) el índice de interfaz de salida (O_2). Nodo de reenvío

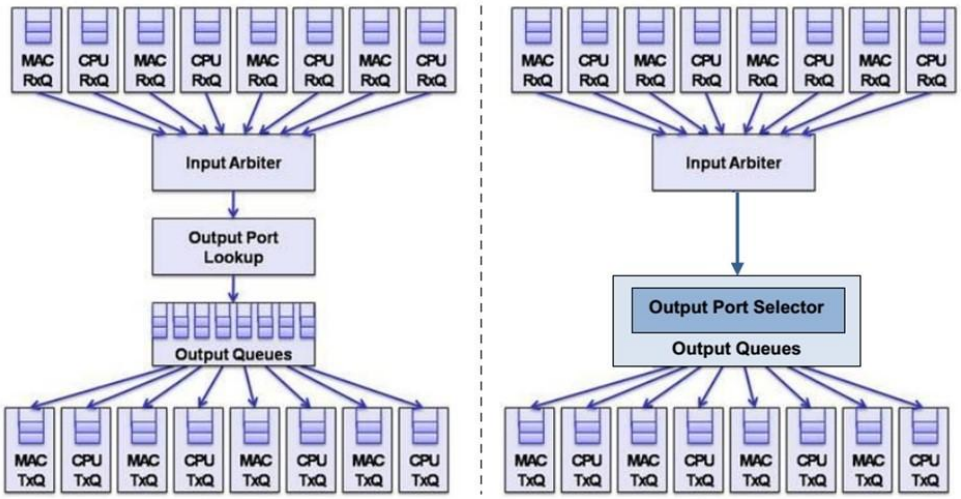

Figura 3.5.: En la izquierda, ruta de datos de referencia. En la derecha, ruta de datos modificada [GH11].

La aplicación adoptada utiliza sólo un conjunto limitado de módulos del switch de referencia de Stanford, sin modificar el resto como se muestra en la Figura 3.5.

Ruta de datos modificada

En la derecha de la Figura 3.5 se tiende la ruta de datos modificada donde varias pilas son nombradas como Rx_queues (RxQ) seguidas por un m{odulo denominado input_arbiter. Al módulo input_arbiter le sigue un módulo output_queues que a su vez tiene un sub-

módulo denominado output_port_selector. Este sub-módulo contiene la implementación principal de la zFormation (cálculo dinámico de los identificadores de enlace en función de cada paquete). Por último se tiene las Tx_queues, que son las pilas de salida.

Bus de paquetes

Los detalles de la señalización entre los buses y los registros se presentan de la siguiente manera.

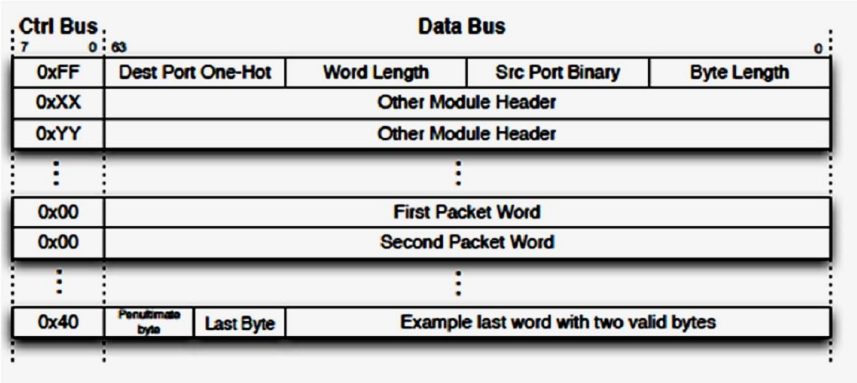

Figura 3.6.: Formato general del recorrido de los paquetes en el bus de paquetes

[GH11].

La velocidad a la que el user_data_path de 64 bits de ancho se está ejecutando es 125 MHz. Esto significa que el ancho de banda máximo que se puede lograr es de 8 Gbps. Los paquetes llegan a diferentes módulos y son desplazados a los otros módulos utilizando FIFO (First in, first out) síncrono como protocolo. Las señales que se utilizan son, de escritura (WR), preparado (RDY), de datos (DATA) y de control (CTRL). Los tamaños de estas señales son un bit para WR, un bit de RDY, 64 bits para datos y 8 bits de CTRL.

El bus CTRL tiene dos propósitos. El primero es hacer una distinción entre las cabeceras de los módulos y el segundo es para indicar el último byte del paquete. Tiene un valor distinto de 0 (cero) para las diferentes cabeceras hasta que reciba la carga útil. Para la parte de datos en el paquete, se establece en cero. Finalmente, cuando llega el final del paquete, el bus de CTRL obtiene el valor del último byte. Como se muestra en la Figura 3.6, el bus CTRL tiene un valor de 0xXX y 0xYY (algunos valores diferentes de cero) para diferentes cabeceras, el paquete empieza una vez que se tiene el valor de 0x00. La última palabra tiene el valor de 0x40, que en binario es 0b01000000. Esto muestra que el byte 7 es el último byte en la última palabra.

Bus de registro

El bus de registro está diseñado en forma de una tubería encadenada que da acceso al host para cambiar los valores en sus propios módulos. Los 32 bits de ancho de bus del registro también funcionan a 125 MHz. Hay dos conjuntos de señales. Una para las solicitudes de entrada y la segunda para las respuestas de salida.

Por petición/respuesta se utilizan los siguientes registros:

REG_REQ_IN y REG_REQ_OUT se establecen en alto por ciclo de reloj. Las señales REG_RD_WR_L_IN y REG_RD_WR_L_OUT muestran si la petición/respuesta se lee o escribe. Para leer se establece en alto y para escribir en bajo. Las señales REG_ACK_IN y REG_ACK_OUT deben estar en nivel bajo para la petición y en alto para la respuesta. Las señales REG_SRC_IN / OUT son usadas por los inicializadores de solicitud de registro para identificar las respuestas que son destinadas al solicitante. Cada registrante debe utilizar un valor único como su dirección de origen.

REG_ADDR_IN transporta la dirección correcta del módulo para que se realice la solicitud, el módulo busca la dirección en este registro, si esta coincide se realiza la solicitud. Una vez finalizada la solicitud, coloca los datos procesados en REG_DATA_OUT y establece el REG_ACK_OUT; el resto de las señales de salida se obtienen de todos los valores de entrada. Todo esto se hace en un solo ciclo de reloj; en los módulos si REG_ADDR_IN no coincide, este reenviará todas las entradas a los registros de salida. Operaciones de reenvío de paquetes

La principal funcionalidad se implementa en el módulo llamado output_port_selector, donde se lleva a cabo la decisión de envío y el paquete se coloca en la pila de salida correcta. Para el cálculo de los IDs de enlace dinámico, se implementa un módulo separado moustique, el cual es un cifrador de flujo, autosincronizable orientado a hardware; y aes_cipher_top para el cifrado de bloques. Para cada enlace, estos módulos son instanciados, en paralelo con el módulo output_port_selector. La estructura básica de output_port_selector se presenta en la Figura 3.7.

Los paquetes llegan en piezas de 64 bits en cada ciclo de reloj. Para el cálculo de los IDs de enlace dinámico, se envían los paquetes desde el módulo input_arbiter a la SRAM y al módulo output_port_selector y luego se toma la decisión de reenvío. Junto con la decisión de reenvío que se lleva a cabo por el bloque lógico do_zfiltering, tienen lugar, para la verificación del paquete, tres operaciones paralelizadas, bit_counter, ethertype, y controles TTL. **do-zfiltering**

En el módulo do-zfiltering, se realizan las operaciones de emparejamiento entre el zfilter entrante y cada enlace de salida. Para cada interfaz se tiene un solo bit formando un vector de bits. Estos bits se establecen en 1 (uno) inicialmente. Rl emparejamiento es hecho por cada máscara de Bloom y filtro de Bloom empaquetado (iBF) por medio de un “AND” y de operaciones de comparación. Si hay una falta de coincidencia para un enlace

en particular, el bit correspondiente se vuelve cero. Al final cuando el emparejamiento para cada mascara de Bloom se termina, el vector de bits muestra las interfaces para enviar el paquete. Siempre que se tiene uno en el vector de bit, el paquete es reenviado en esa interfaz.

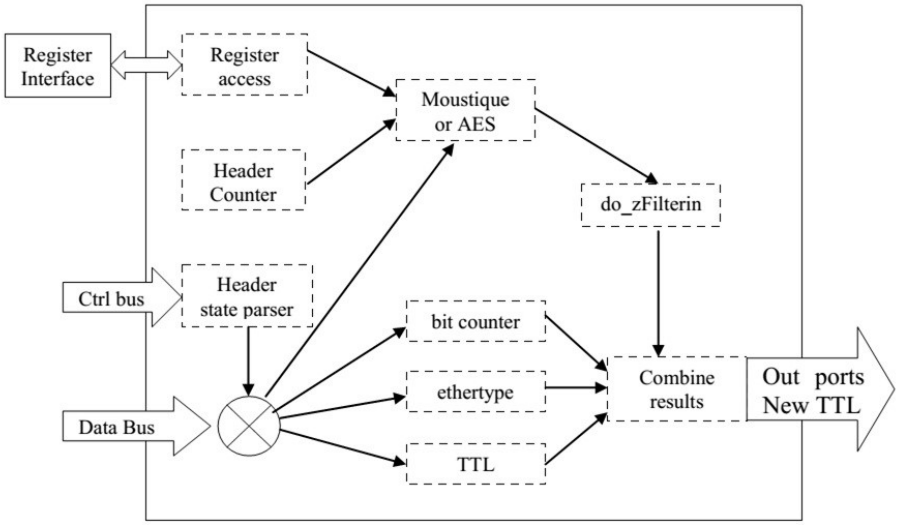

Figura 3.7.: Estructura del modulo output_port_selector [GH11].

bit_counter

El módulo bit_counter cuenta el número de unos en el iBF. Todos los bits en el iBF se establecen en uno para evitar ataques. El número máximo de unos permitidos es un valor constante. Si el iBF contiene un mayor número de unos de los que el valor constante, se descarta el paquete. El módulo es implementado usando solamente cables y elementos lógicos. Se toma 64 bits de entrada y se devuelve el número de unos. Esto significa que por 256 bits de iBF se tarda 4 ciclos de reloj para contar el número de unos.

Gestión de Software

Para la generación de la salida O tenemos:

$$O = Z(K,M,I) \tag{3.3}$$

Donde K es una clave secreta semiestática que cambia periódicamente, M es un término dinámico medio que incluye los indices de las interfaces de entrada y salida e I denota información adicional de los paquetes.
Para fines de desempeño, la clave k es dividida en tres partes criptográficamente separadas, K_1, K_2 y K_3, las cuales son creadas usando una función de derivación de claves (KDF, Key Derivation Function):

$$K_i = KDF(K, L_i) \tag{3.4}$$

Donde el término Li es un literal que identifica una clave en particular

Desde el espacio de usuario y basandose en K, el software de gestión calcula las claves K_1, K_2 y K_3 utilizando HMAC-SHA-256. Del mismo modo, define las interfaces de salida y entonces calcula los valores O_1 y O_2 , valores necesarios para la construcción de zFormation, establecidos usando HMAC-SHA-256. Se escribe la clave K_3 y el valor O_2 establecidos en los registros de la tarjeta NetFPGA. En el lado del software también se calcula la zFormation por ejemplo F_3 utilizando K_3, O_2 y genera un I por cada paquete. Esto se hace para cada interfaz, entonces se calcula el iBF y es almacenado dentro de un paquete.

El software puede enviar paquetes personalizados a la tarjeta NetFPGA. Esto controla el retraso entre la transmisión de los paquetes, tamaño de los paquetes, tiempo de vida (TTL) en la cabecera de los paquetes y el campo de protocolo ethernet.

4. Diseño

Tanto los mecanismos de enrutamiento MPSS como D-MPSS mejoran características anteriormente explicadas reemplazando las cabeceras de los paquetes MPLS con filtros de Bloom. En consecuencia el tratamiento que se les dará a los paquetes que ingresan a un enrutador intermediario no será el mismo que para paquetes con encabezados MPLS. Sin embargo muchas de sus funciones se basan en la arquitectura de un enrutador IPv4, lo que hace posible que se le tome como referencia. Utilizar un enrutador convencional o propietario, no permitirá que podamos desarrollar distintos mecanismos orientados a una función en específico. En cambio el uso de una plataforma libre a nivel de hardware, como es el caso de la NetFPGA, proporcionará la flexibilidad necesaria para crear y modificar distintos diseños, de tal manera que resulte ideal su uso para nuestras necesidades en el caso de la implementación de los mecanismos de enrutamiento mencionados.

En la presente tesis se logró la implementación de MPSS y D-MPSS en un nodo enrutador a nivel hardware, para ello fué de suma importancia realizar un diseño adecuado de dicha implementación. Entre los aspectos relevantes en el diseño, se encuentra la generación del tráfico hacia el nodo, el ingreso de los paquetes al nodo, la revisión y descompresión de las cabeceras, la comprobación de la interfaz de salida y el envío del paquete por la interfaz adecuada.

4.1. Configuración de las conexiones

La plataforma NetFPGA brinda la posibilidad de descargar paquetes preestablecidos para la tarjeta. Uno de los paquetes es un enrutador de referencia IPv4 y otro de ellos es zfilter. Zfilter es el nombre que se da al filtro de Bloom el cual tomará las decisiones de reenvío en una red, en el trabajo LIPSIN explicado en el estado del arte se muestra su estructura con mayor detenimiento. De esta manera, como los mecanismos de enrutamiento que deseamos implementar son basados en LIPSIN, se trabajará con este modelo de referencia.

Para tener una idea general de cómo se trabajará el diseño, en laFigura 4.1 se muestra los bloques necesarios para la puesta a punto de la NetFPGA. En la parte superior se encuentra el host CPU donde se instala el software necesario para el funcionamiento de la tarjeta, cuenta además con acceso a una memoria principal que permite que se ejecuten los programas alojados en el host. El host se conecta a la tarjeta mediante un puerto PCI de 64 bits que será el responsable de la comunicación entre ambos.

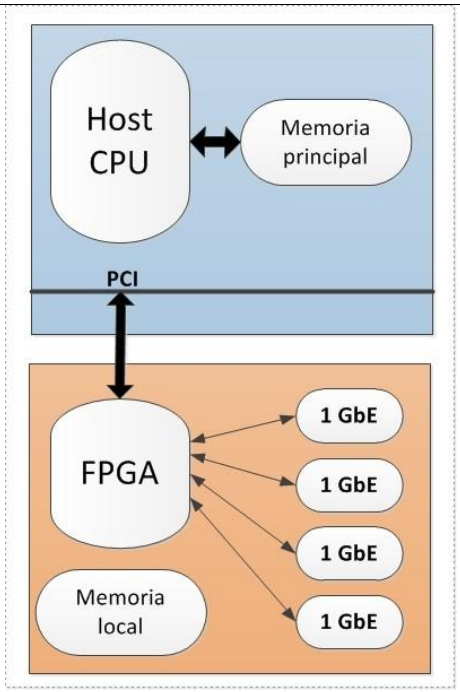

Figura 4.1.: Representación del diagrama de bloques de la plataforma NetFPGA

y un Host [Elaboración propia].

La tarjeta posee memorias locales conectadas a una FPGA interna que a su vez se conecta a cuatro puertos GbE, que permiten la conexión hacia otras tarjetas, módems o PCs capaces de soportar GbE.

Uno de los pasos iniciales es realizar las pruebas que verifiquen el buen funcionamiento de la tarjeta utilizando el esquema mostrado en la Figura 4.2. El usuario, mediante el host, luego de configurar correctamente el puerto PCI de la tarjeta puede ser capaz de generar tráfico a través de dos formas. La primera es utilizando la conexión PCI para lo cual deberá seccionar los paquetes que se enviarán a la tarjeta en bloques de 64 bits a través las interfaces GbE virtuales de la tarjeta. La otra forma es mediante la tarjeta de red, que para evitar pérdida de paquetes debe soportar hasta 1 GbE y será a través de los puertos físicos de la tarjeta. En ambos casos se genera una serie de datos aleatorios que representaran la carga de los paquetes y los filtros de Bloom, que irán en la cabecera del paquete, tendrán características como se menciona en la siguiente subsección.

Para lograr resultados más precisos donde se minimicen retardos propios del sistema operativo, se propone utilizar el diagrama de conexiones visto en la Figura 4.3. Se plantea hallar los resultados mediante el uso de dos tarjetas conectadas entre sí; una de ellas contará con un modelo de referencia denominado Packet Generator, el cual

4.1 Configuración de las conexiones

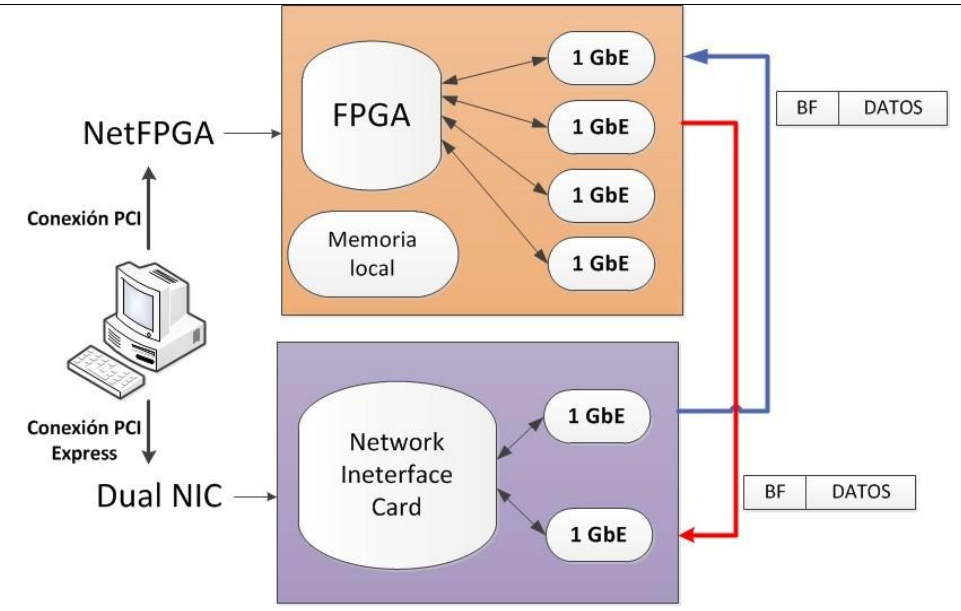

Figura 4.2.: Esquema del proceso de envío de datos con filtros de Bloom en un host [Elaboración propia].

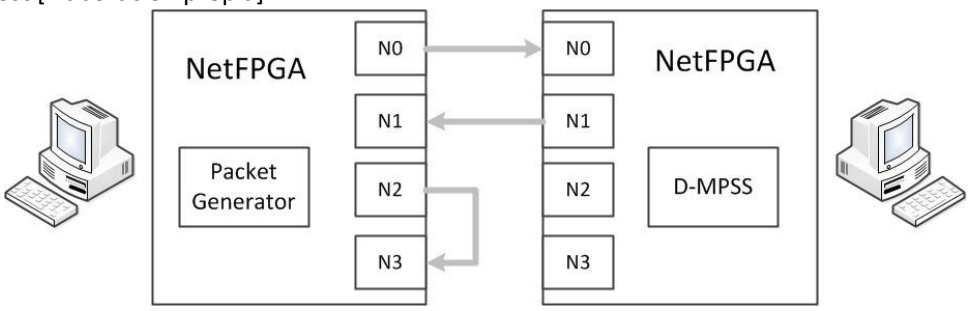

Figura 4.3.: Esquemático de la propuesta de conexiones entre dos NetFPGA para realizar pruebas entre dos hosts [Elaboración propia].

es capaz de generar y capturar tráfico por cualquier puerto de la tarjeta indicando estadísticas halladas en el proceso. La otra tarjeta tendrá cargado el modelo D-MPSS implementado, que recibirá el tráfico por un puerto o interfaz proveniente del Packet Generator y lo procesará para luego enviarlo por otro puerto nuevamente hacia la otra tarjeta.

4.2 Escenario MPSS

4.2. Escenario MPSS

Para realizar las pruebas de la implementación en MPSS se tienen cuatro casos como se observa en la Figura 4.4. La interfaz IF 1-1 no se considera para la formación del filtro de

Bloom en ningún caso ya que representa la interfaz de entrada por la que ingresará el bloque a ser comparado. Los casos representados indican que dos o las tres interfaces restantes son salidas permitidas. En el Cuadro 4.1 se tiene la cantidad de bits que se considerará como longitud del filtro de Bloom y por ende de las interfaces. Así mismo se presenta el valor de k que indica la cantidad de unos presentes en el filtro.

bits	k
256	3
384	4
512	5
800	5

Cuadro 4.1.: Cantidad de bits formadores del filtro de Bloom con k bits en 1 para MPSS [Elaboración propia].

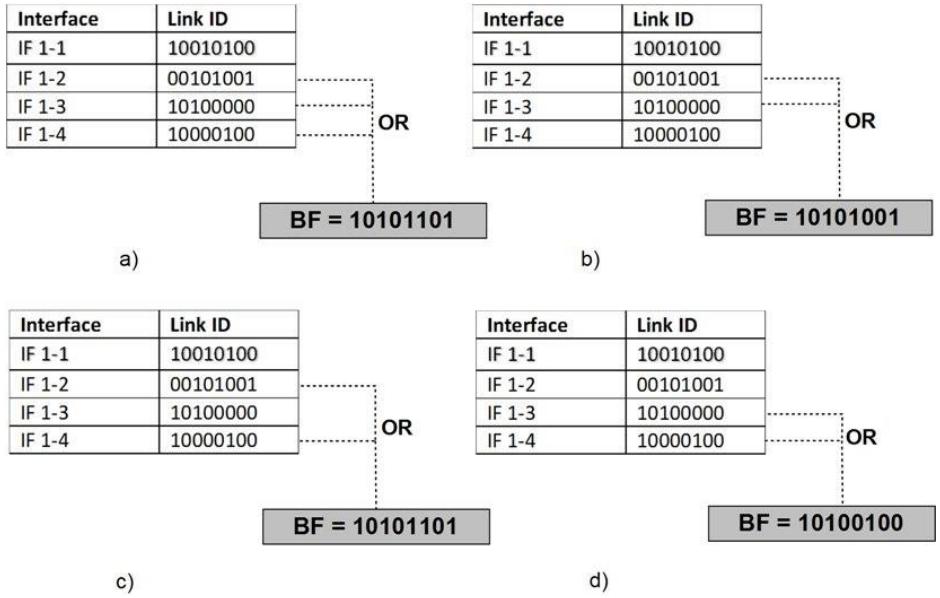

Figura 4.4.: Ejemplo de la creación del filtros de Bloom de acuerdo a las interfaces de la NetFPGA. En a) con tres interfaces de salida permitidas, en b), c) y d) con dos interfaces de salida permitidas [Elaboración propia]

4.3 Escenario D-MPSS

4.3. Escenario D-MPSS

Para el escenario D-MPSS se consideran longitudes de 512, 800 y 1024 bits, y un *k* de 5 y 6. Además se agrega una variable "*mult*" con valores de 8 y 32, como se muestra en el Cuadro 4.2. El campo mult representa un factor de multiplicación que indicará de cuantos bits estáformada la cabecera y cúantos se eliminan en cada nodo. Este último valor se considera representativo ya que al realizar la implementación en un solo nodo no se aprecia su impacto real; sin embargo, se puede ver su eficiencia variando el valor mult y considerando el mismo nodo como si fueran varios.

bits	mult	k

512	8	5
512	32	5
800	8	5
800	32	5
1024	8	6
1024	32	6

Cuadro 4.2.: Cantidad de bits formadores del filtro de Bloom con un mult de 8 y

32 y k bits en 1 para D-MPSS [Elaboración propia].

4.4. Router Data Path

Una característica principal de la plataforma NetFPGA es un diseño modular fácil de entender y modificar. El material de referencia fomenta a estudiantes e investigadores a transportar esta modularidad a través de sus diseños.

El código de inicio se estructura en el siguiente conjunto de componentes: la ruta de datos del usuario, los bloques de E/S Ethernet (encapsulando el MAC), y la lógica de la interfaz de host (que permite la comunicación y la transferencia de paquetes entre el host y NetFPGA). La ruta de datos de usuario se implementa como una tubería, utilizando múltiples módulos. Un árbitro lee los paquetes de la red Ethernet o desde la interfaz del host y luego los alimenta a través de la tubería. Debido a que los paquetes se multiplexan juntos, no se necesita ningún travesaño dentro de la ruta de datos. Los datos demultiplexados en forma de paquetes se escriben en las colas de salida. El ancho de banda pico a través de la ruta de datos de usuario es de 8 Gbps en la estructura de la FPGA. Un diagrama de bloques de la ruta de datos se muestra en la Figura 4.5

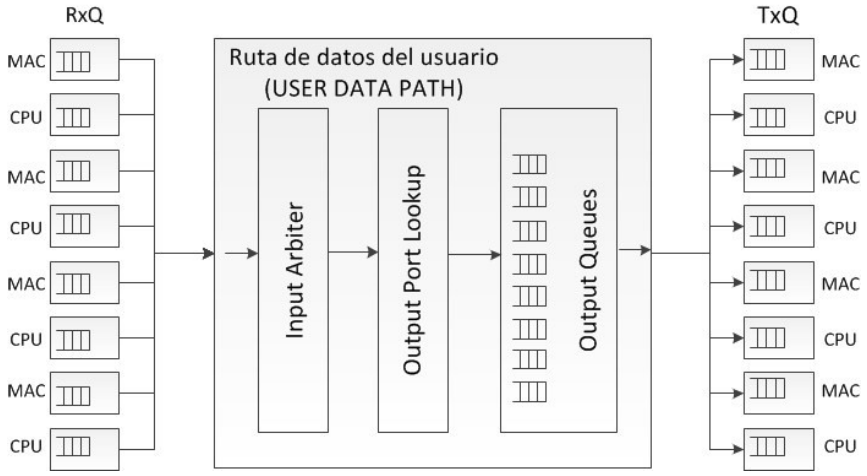

Figura 4.5.: Diagrama de la ruta de datos del usuario en la NetFPGA [NGBM08].

4.5. Output Port selector

Basados en la estructura del modelo de referencia mostrado en la Figura 4.6 se modifica el módulo del output Port Lookup uniéndolo al Output Queues para que se acomode mejor a las necesidades del presente trabajo, tal como se hace en [KJS09]. El data bus recibe la secuencia, de 32 bits, que ingresa a la tarjeta NetFPGA para su procesamiento, seguidamente se hacen copias de la misma para realizar distintas verificaciones. Una copia se envía al módulo filtrado donde se realizará el emparejamiento y la comprobación para conocer la o las interfaces por las que puede salir. El módulo de id almacenadas tendrá las cuatro interfaces con los identificadores de enlace de cada una para luego enviarlas al filtrado para la respectiva comprobación. Otra copia se envía al contador de bits al que se le asignara una variable que almacene el número de unos que posea la secuencia. Si al finalizar, el contador de bits indica que el número de unos es muy alto, se descartará el paquete, este paso sirve para proveer seguridad y evitar ataques que pretendan inundar la red con paquetes. Los siguientes campos que reciben una copia son el de TTL y mult para MPSS y D-MPSS respectivamente, el de TTL ayudará a que en MPSS las secuencias resultantes de falsos positivos no inunden y saturen la red y en el caso de mult indicará que parte de la secuencia se eliminará para su siguiente salto. Para finalizar una vez que se haya realizado las comprobaciones indicadas, la secuencia almacenada puede salir por el out_ports según sea el caso.

Se plantea otra manera de realizar el emparejamiento del filtro de Bloom y los identificadores de enlace de las interfaces de salida. Un ejemplo de ello se ve en la Figura 4.7 y en la Figura 4.8 que representan el proceso de compresión y descompresión de una cabecera de filtro de Bloom respectivamente. El proceso de compresión, que se da en la generación del paquete, consiste en reducir el tamaño de la cabe-

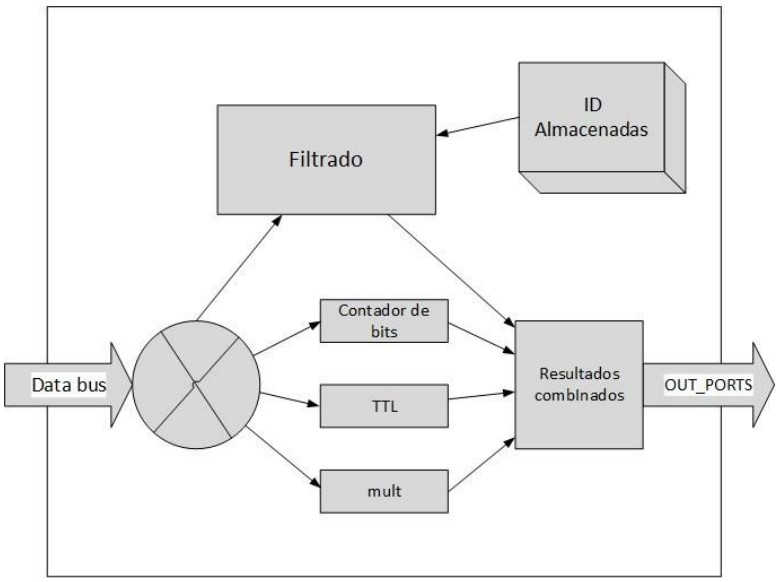

Figura 4.6.: Estructura del módulo Output Port selector para el bus de datos [Elaboración propia].

cera dependiendo del nivel del árbol en el que se encuentre la ruta. Es así que se tendrán cabeceras de distintos tamaños, así mismo estas cabeceras se irán apilando generando una cabecera final. Para la compresión por cada nivel se utiliza un algoritmo determinado por el módulo, de esta manera, se elige en primer lugar la tasa de compresión, luego con este valor se evalúa el módulo de las posiciones de cada bit de esta manera: posición mod(tasa de compresión); el resultado será la nueva posición que los valores anteriores ocuparán en el filtro comprimido. En el caso que se repita la posición, solo será necesario realizar una operación OR lógica entre los valores con las mismas posiciones.

Para la descompresión, ya en el nodo intermedio, se utilizan las mismas propiedades del módulo, dividiendo las cabeceras comprimidas en bloques del tamaño del bus de datos o menores; luego se realiza el mismo proceso con los valores de las interfaces de salida del nodo, advirtiendo en no exceder el número de bloques en que se separó el filtro comprimido, si el tamaño de la interfaz es mayor (como lo es en la mayoría de los casos), se enumerará los bloques de la interfaz reiniciado el conteo cuando se sobrepase el número de bloques del filtro comprimido. A continuación se realiza la comparación mediante operaciones AND lógicas entre los bloques con igual número, si el resultado es el mismo a los bloques pertenecientes a las interfaces, se reúnen todos los resultados y el paquete puede salir.

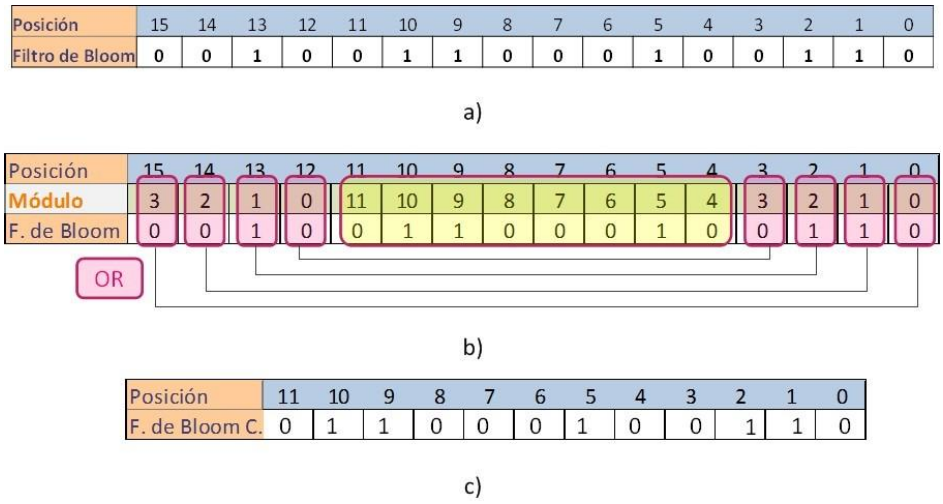

Figura 4.7.: Proceso de compresión del filtro de Bloom utilizando módulo 12. En a) filtro de Bloom de tamaño completo, en b) busqueda mediante el módulo y en c) filtro de Bloom comprimido [Elaboración propia].

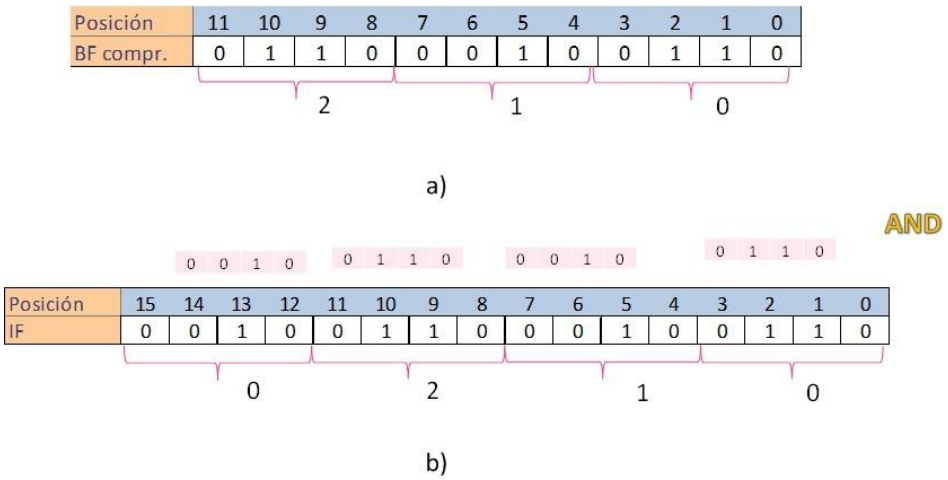

Figura 4.8.: Proceso de descompresión del filtro de Bloom utilizando módulo 12. En a) filtro de Bloom comprimido seleccionado en bloques, en b) operaciones AND entre los bloques seleccionados según el módulo para determinar la salida

[Elaboración propia].

5. Implementación, resultados ycomparaciones

5.1. Administrador de software - PSM (PubSubManager)

La generación de paquetes se basa en una herramienta de software denominada PSM y formulada en lenguaje de programación C. Esta herramienta se encarga de organizar una serie de parámetros ingresados por el usuario de tal manera que el paquete generado pueda ser ingresado e identificado por la tarjeta NetFPGA.

La función principal del código es generar y enviar el paquete por un puerto o interfaz de la tarjeta, portando determinada información como la cabecera D-MPSS. EL paquete estará conformado en primer lugar por una cabecera Ethernet utilizándola básicamente para el reconocimiento del paquete dentro de un entorno de red mediante el programa "wireshark". A continuación se agrega una cabecera D-MPSS la cual cuenta con un campo de 8 bits para el factor de multiplicación, seguido de los filtros de Bloom apilables que tendrán una tamaño variable según el factor de multiplicación. Por último se añade la carga o los datos formados por valores aleatorios.

EL código en C, formulado originalmente para Zfilter y actualmente disponible en [KJS09], se modificó según los requerimientos de esta tesis previo análisis de ingeniería inversa para determinar su funcionamiento interno. Así mismo se adaptó para que el usuario pueda ingresar valores con las características propias de D-MPSS.

5.1.1. Interfaz de usuario

El administrador de software permite al usuario ingresar los siguientes parámetros: el puerto o interfaz de la tarjeta por donde ingresará el paquete, el factor de multiplicación representado por "f", la cantidad de paquetes, el retraso entre paquetes, el tamaño de la carga y por último el filtro de Bloom apilable. Además de enviar el paquete el administrador permite modificar los identificadores de enlace de cada interfaz de la NetFPGA, así como mostrarlos en pantalla una vez modificados.

5.1.2. Generación de paquetes

Una vez ingresados los parámetros requeridos, el programa se encarga de formar un paquete estructurado que es enviado a la tarjeta. Para verificar el paquete ya

formado se puede realizar una captura por la interfaz de salida mediante el software wireshark.

5.2. Implementación en la NetFPGA

Para realizar el procesamiento de los paquetes y las decisiones de reenvió es necesario programar la tarjeta en un lenguaje de descripción de hardware de bajo nivel como verilog. El código en verilog debe ser capaz de ser sintetizable para así generar un archivo que puede ser cargado en la tarjeta NetPFGA y ejecutar las operaciones de selección de las interfaces de salida

5.2.1. Instalación de la NetFPGA

La instalación de la tarjeta NetFPGA se detalla en el anexo B, es necesario contar con un sistema operativo Linux CentOS 5.5 para poder ejecutar el modelo de referencia zfilter, es necesario instalar determinados paquetes a diferencia de la versión en Fedora la cual ya cuenta con la configuración completa.

5.2.2. Programación en Verilog

El proyecto zfilter como se mencionó anteriormente cuenta con un diseño preestablecido para trabajos realizados sobre la plataforma NetFPGA, dentro de este modelo podemos encontrar una carpeta con el código en verilog. El código está basado en el diseño de un enrutador de referencia y está formado por módulos diferenciados, cada uno de ellos almacenado en un archivo diferente.

Para adaptar el código en verilog a D-MPSS fue necesario modificar los módulos de “output_queues” y “output_port_selector” que se encargan de realizar las decisiones de reenvió y preparar los datos para ser enviados.

Dentro de las funciones que realiza el código destacan las siguientes:

Remover paquetes

D-MPSS cuenta con la característica de tener filtros de Bloom apilables de múltiples tamaños; es necesario que en cada salto, en cada nodo, se retire parte de la cabecera ya utilizada y así no se repita a fin de reducir las anomalías de reenvió. En cada salto se debe reconocer el factor de multiplicación “f” y a continuación calcular la cantidad de bits que se debe eliminar. Por ejemplo, en caso de utilizar un $mult$ (valor a multiplicar o múltiplo) de 64 bits y un f de 4 será necesario eliminar $f \cdot mult$ = 256 bits.

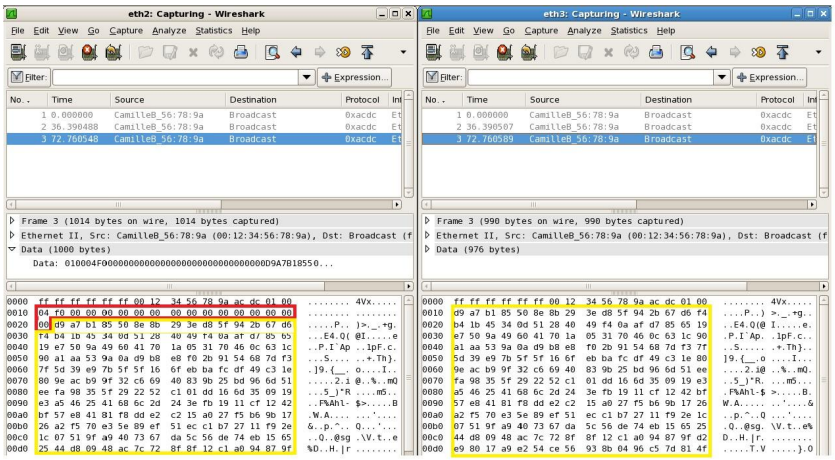

Figura 5.1.: Captura wireshark, eliminación de paquetes.

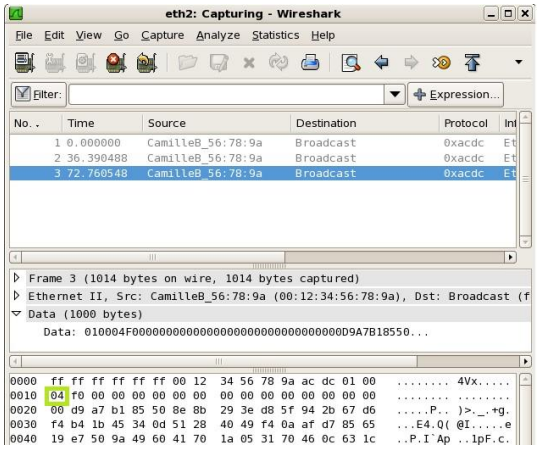

Figura 5.2.: Captura wireshark, función de f para el cálculo de cabecera.

Reacomodar la cabecera y el nuevo paquete

Una vez eliminada parte de la cabecera es necesario desplazar el resto de información para que en el siguiente salto se pueda leer adecuadamente. Para ello es necesario indicar al programa que en la cantidad de bits indicados, se deje de colocar los datos en las colas de salida y así evitar que se escriban ceros en vez de eliminar la información innecesaria.

Eliminar tablas ID, LIT y características innecesarias de MPSS

MPSS a diferencia de D-MPSS utiliza tablas de identificadores de enlace agregando más de un identificador de enlace por interfaz. D-MPSS únicamente necesita un identificador de enlace ID por interfaz de tal manera que se deben eliminar el resto de IDs. Además se configura para que se pueda variar el tamaño de los IDs.

Descompresión y comparación

Como se muestra en el estado del arte, D-MPSS utiliza filtros de Bloom apilables, de tal manera que por cada nivel se tendrá una cabecera de diferente tamaño, las cuales se irán añadiendo en forma de cola, según el camino que tomará el paquete, hasta formar una cabecera más grande. El tamaño de cada cabecera resulta de comprimir el filtro de Bloom en un valor de f, determinado por el nivel del árbol. La tasa de compresión será mayor en los niveles superiores y menor o inexistente en los niveles posteriores, esto debido a que la cantidad de nodos en los niveles superiores tiende a ser menor y una mayor compresión no tendrá gran impacto al momento de realizar las operaciones de descompresión y decisión posteriores. En los niveles posteriores suele existir una mayor congestión de nodos por lo que es recomendable reducir la tasa de compresión con el fin de que no se pierda información al momento de descomprimir la cabecera.

Para realizar las operaciones de comparación y verificación de las interfaces de salida, es necesario que el nodo intermediario descomprima la cabecera del nivel correspondiente a dicho nodo y la compare mediante una operación AND lógica con todas las interfaces de salida. Realizar la descompresión y luego la comparación resulta un proceso complejo por lo que genera un consumo de recursos excesivo, es así que se sugirió el método de compresión y descompresión mediante el módulo. Utilizando el módulo simplemente se indica la tasa de compresión, con ello se busca posiciones especificas en las interfaces y se realiza una comparación en tiempo real respectiva, en cada ciclo de reloj.

En cada ciclo de reloj se procesan 64 bits, para un mejor rendimiento se utiliza un mult del mismo tamaño del bus de datos, 64 bits en nuestro caso. De esta manera si la cabecera descomprimida tiene 800 bits, se podrá comprimir hasta un máximo de 64 bits. Para comparar si el identificador de enlace corresponde con el filtro de bloom sin hacer una descompresión, se realiza una búsqueda mediante el módulo de f, si el módulo corresponde a la posición del filtro se realiza una operación AND entre el filtro y la interfaz, si el resultado es igual a la interfaz por esa interfaz podrá salir el paquete.

Elección de interfaz de salida

Una vez realizadas las decisiones de reenvió es necesario indicar qué interfaces pueden enviar el paquete, para ello se almacenan en un vector las interfaces de salida

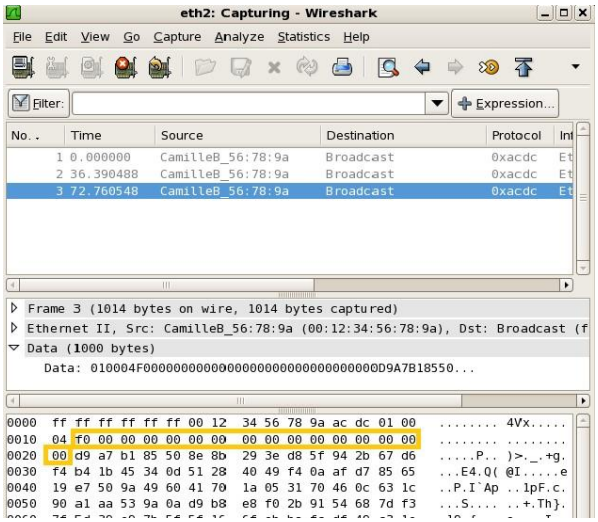

Figura 5.3.: Captura wireshark, cabecera a descomprimir y comparar.

indicando con 1 si es que puede salir por la interfaz y con cero si no es posible. Estos datos son interpretados por la tarjeta permitiendo o no la salida del paquete.

5.2.3. Generación de un nuevo bitfile

Un bitfile es un archivo con extensión .bit generado con Xilinx luego de realizar la síntesis de un código en un lenguaje de descripción de hardware. Este archivo convierte el lenguaje ingresado por el usuario en uno expresado por compuertas lógicas y multiplexores y puede ser leído directamente por una FPGA. Para generar el bitfile del presente proyecto hay que considerar ciertas restricciones en la programación del código para que sea sintetizable. El proceso de generación del bitfile puede tomar entre 40 minutos a más dependiendo de la complejidad de las operaciones a realizar. Cada vez que se realice una modificación al código en verilog será necesario volver a generar un nuevo bitfile.

5.2.4. Cargar un nuevo bitfile

Para cargar el bitfile generado hay que ubicar la carpeta synth y ejecutar el comando: nf2_download nf2_top_par.bit, y automáticamente se cargara el archivo en la tarjeta y estará listo para ser usado.

Figura 5.4.: Mapa de la red Abilene, http://abilene.internet2.edu/.

Nombre	Nodos del Núcleo	Nodos PE	Número total de Enlaces
NSFNET	14	107	282
Abilene	10	209	504
COST-266	39	329	796

Cuadro 5.1.: Redes usadas en las simulaciones e implementación.[FL12]

5.3. Generación de tráfico

Para generar el tráfico se modificó un código en lenguaje Java utilizado en simulaciones anteriores para D-MPSS para que sea compatible con los datos ingresados a la tarjeta.

En [FL12] se evalúa el rendimiento analíticamente y con simulaciones para calcular los falsos positivos, la sobrecarga de la cabecera y su comportamiento en una topología de red real a nivel de software. La topología de red planteada considera enrutadores proveedores de agregación (PA), proveedores de borde (PE) y proveedor troncalizado (P). En el presente trabajo se desarrolla su implementación en hardware, para ello se considera el mismo tráfico en tres tipos de redes usadas para las simulaciones: Abilene en la Figura 5.4, NSFNET en la Figura 5.5 y COST-266 en la Figura 5.6. En el Cuadro 5.1 se indica la cantidad de nodos y enlaces presentes en cada red.

Una vez modificado el código en Java se preparó un archivo de texto para que pueda ser interpretado por el código en lenguaje C el cuál se encarga de generar el paquete

5.3 Generación de tráfico

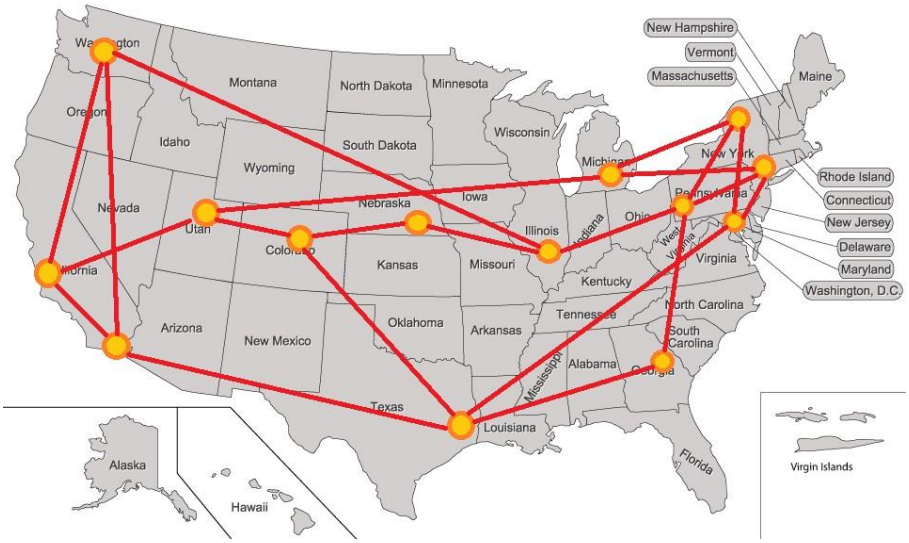

Figura 5.5.: Mapa de la red NSFNET, http://www.nsf.gov/.

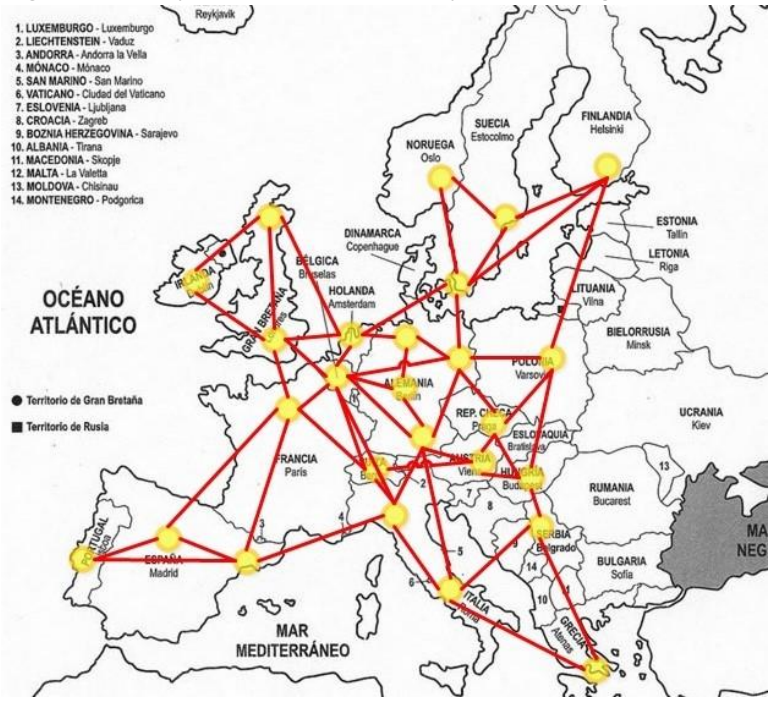

Figura 5.6.: Mapa de la red COST-266, http://www.cse.buffalo.edu/.

como se explicó anteriormente.

5.4. Pruebas y resultados

Las pruebas realizadas consisten en la verificación del funcionamiento de la tarjeta una vez descargado el bitfile mediante tests de regresión y a continuación el control del rendimiento por medio de la latencia de procesamiento una vez se ingrese el tráfico en la tarjeta. Las pruebas de rendimiento consideran distintas redes reales y casos para determinadas variables.

5.4.1. Tests para la implementación

Test 1: Enviando paquetes por el puerto 1 y esperando su salida en el resto de puertos

Se enviaron 10 paquetes por el puerto 1 de la tarjeta que corresponde al nf2c0. Se esperó a que lleguen los paquetes en todos los puertos excepto en el 1(donde se está enviando) a fin de evitar bucles, y se siga la ruta indicada.

Se verificó que en la salida llegaban 10 paquetes en todos los puertos de salida excepto en el 1. De esta forma se verifica la correcta funcionalidad de la implementación para broadcast

Test 2: Enviando paquetes por el puerto 2 y esperando su salida en el resto de puertos

Se enviaron 10 paquetes por el puerto 2 de la tarjeta que corresponde al nf2c1. Se esperó a que lleguen los paquetes en todos los puertos excepto en el 2(donde se está enviando) a fin de evitar bucles, y se siga la ruta indicada.

Se verificó que en la salida llegaban 10 paquetes en todos los puertos de salida excepto en el 2. De esta forma se verifica la correcta funcionalidad de la implementación para broadcast

Test 3: Enviando paquetes por el puerto 3 y esperando su salida en el resto de puertos

Se enviaron 10 paquetes por el puerto 3 de la tarjeta que corresponde al nf2c2. Se esperó a que lleguen los paquetes en todos los puertos excepto en el 1(donde se está enviando) a fin de evitar bucles, y se siga la ruta indicada.

Se verificó que en la salida llegaban 10 paquetes en todos los puertos de salida excepto en el 3. De esta forma se verifica la correcta funcionalidad de la implementación para broadcast

Test 4: Enviando paquetes por el puerto 4 y esperando su salida en el resto de puertos

Se enviaron 10 paquetes por el puerto 4 de la tarjeta que corresponde al nf2c3. Se esperó a que lleguen los paquetes en todos los puertos excepto en el 2(donde se está enviando) a fin de evitar bucles, y se siga la ruta indicada.

Se verificó que en la salida llegaban 10 paquetes en todos los puertos de salida excepto en el 4. De esta forma se verifica la correcta funcionalidad de la implementación para broadcast

Test 5: Enviando paquetes por el puerto 1 y considerando al puerto 2 fuera de la ruta

Se enviaron 10 paquetes por el puerto 1 de la tarjeta que corresponde al nf2c0. Se esperó a que lleguen los paquetes en todos los puertos excepto en el 1(donde se está enviando) y en el 2(no está incluido en la ruta), no existiendo un emparejamiento correcto con el puerto 2.

Se verificó que en la salida llegaban 10 paquetes en los puertos 3 y 4; y ningún paquete en los puertos 1 y 2.

Test 6: Enviando paquetes por el puerto 1 y considerando al puerto 3 fuera de la ruta

Se enviaron 10 paquetes por el puerto 1 de la tarjeta que corresponde al nf2c0. Se esperó a que lleguen los paquetes en todos los puertos excepto en el 1(donde se está enviando) y en el 3(no está incluido en la ruta), no existiendo un emparejamiento correcto con el puerto 3.

Se verificó que en la salida llegaban 10 paquetes en los puertos 2 y 4; y ningún paquete en los puertos 1 y 3.

Test 7: Enviando paquetes por el puerto 1 y considerando al puerto 4 fuera de la ruta

Se enviaron 10 paquetes por el puerto 1 de la tarjeta que corresponde al nf2c0. Se esperó a que lleguen los paquetes en todos los puertos excepto en el 1(donde se está enviando) y en el 4(no está incluido en la ruta), no existiendo un emparejamiento correcto con el puerto 4.

Se verificó que en la salida llegaban 10 paquetes en los puertos 2 y 3; y ningún paquete en los puertos 1 y 4.

Test 8: Test para el factor de multiplicación

Se envió 10 paquetes usando el puerto 1 que corresponde al nf2c0. El campo f se establece en 0, los paquetes no deberían salir por ningún puerto. La salida de la simulación verifica

la funcionalidad al no observarse salida alguna por ningún puerto. Se envió 10 paquetes usando el puerto 1 que corresponde al nf2c0. El campo f se establece en 30, los paquetes no deberían salir por ningún puerto, ya que f excede al valor permitido. La salida de la simulación verifica la funcionalidad al no observarse salida alguna por ningún puerto.

5.4.2. Análisis de complejidad ciclomática

El análisis de complejidad ciclomática en el presente trabajo, puede contribuir a tener una mejor visión de la calidad del diseño del software, de una manera rápida y con independencia del tamaño de la aplicación. Así mismo ayuda a planificar las mejoras al código para priorizar las partes del diseño a mejorar. De manera general, es un indicador que ayuda a determinar muchas de las posibles deficiencias del código como rigidez del diseño, carencias modulares del diseño etc. Existe una relación entre el esfuerzo (costos) necesario para mantener el programa y la complejidad de su diseño. La complejidad ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. Una vez realizado el cálculo de la complejidad ciclomática, se puede determinar el riesgo que supone utilizando los valores representados en el Cuadro 5.2[McC76]. Para calcular el valor de la complejidad ciclomática en determinado fragmento de código se puede calcular de la siguiente manera:

$$v(G) = a - n + 2 \square p \tag{5.1}$$

Complejidad ciclomática	Evaluación de riesgo
1-10	Programa sencillo, no hay riesgo.
11-20	Programa más complicado, riesgo moderado.
21-50	Programa complejo, riesgo alto.
50	Programa inestable, riesgo muy alto.

Cuadro 5.2.: Evaluación de riesgo asociado a cada valor de la complejidad ciclo-

mática para un programa dado [McC76].

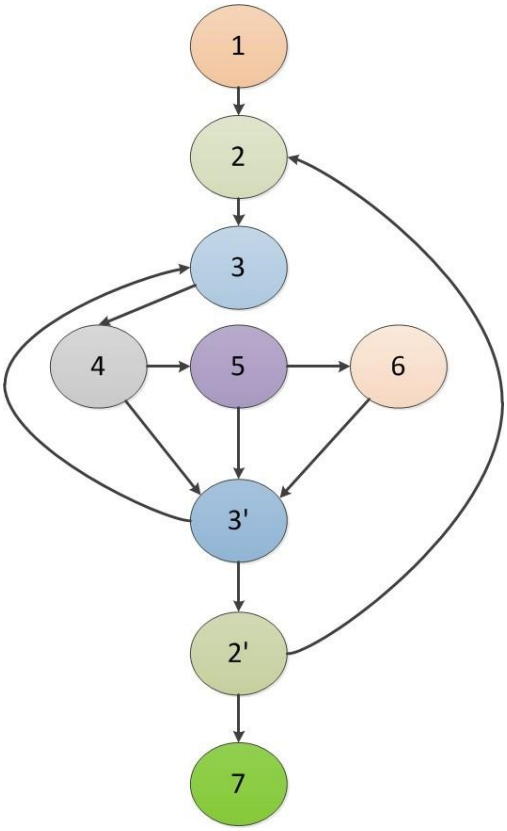

Figura 5.7.: Grafo correspondiente al análisis de complejidad ciclomática del algoritmo de decisión [Elaboración propia].

A partir de un grafo de control $G = (N,V)$, donde N es un conjunto de nodos y V un conjunto de aristas entre los nodos, se calcula la complejidad ciclomática, siendo a: número de aristas, n: número de nodos y p: número de componentes conexos, es decir, la cantidad de programas enlazados.

Para el cálculo actual se consideró uno de los módulos del User Data Path, el Output Port Selector, ya que dentro del mismo se encuentra una de las partes más críticas de la evaluación que es la que corresponde a la decisión de reenvío. Así, se tomó como base el fragmento de código en Verilog donde se realizan las operaciones de descompresión y comparación visto en Apéndice C, en el apartado correspondiente a Output Port Selector.

En la Figura 5.7 se observa el grafo correspondiente al análisis de complejidad ciclomática del algoritmo de decisión, representando los estados mediante números. Para tener una

visión más clara, cada número del grafo está relacionado a una línea del pseudocódigo del algoritmo de decisión descrito en el Algoritmo 5.1.

Algoritmo 5.1 Pseudocódigo del algoritmo de decisión.

```
1   selections_a=1
2   Para i de 0 hasta 3
3   Para j de 0 hasta 3
4   Si la posición es igual a i mod 64 entonces
5   Si el emparejamiento es incorrecto entonces
6   Actualizar el valor de selections_a=0;
7   Se almacena selections_a en una variable de salida hacia el siguiente
    módulo
```

Al realizar los cálculos mediante la Ecuación 5.1 se obtuvo:

$$a = 12, n = 9, p = 1\ 99K\ v(G) = 12 - 9 + 2 \square 1 = 5$$

Según la Tabla 5.2, el resultado obtenido de complejidad ciclomática para el segmento de decisión no implica riesgo al tener un valor de 5. Queda destacar que el mismo proceso se elabora cuatro veces de manera paralela para selections_a, selections_b, selections_c y selections_d; siendo procesos independientes que una vez terminados reuniran sus resultados.

5.4.3. Mediciones del rendimiento

Los tiempos de procesamiento del paquete fueron medidos en un ambiente de pruebas. Los paquetes fueron enviados en un rango de 25 paquetes por segundo. Las operaciones de envío y recepción fueron implementadas en consola de Linux CentOS 5.5. Las medidas fueron tomadas para un valor de M= 768 bits (Tamaño de Identificador de enlace ID) y valores de k (cantidad de 1's) de 4,5 y 6. Para realizar el cálculo teórico de la latencia requerida en la toma de decisiones, se considera la cantidad de ciclos de reloj ocupados como se ve en la Figura 5.8, en cada ciclo de reloj se realizan determinadas actividades. A partir del tercer ciclo de reloj se observa como se puede realizar más de una actividad (filtrado mientras se esperan los siguientes paquetes) por ciclo de reloj debido a la capacidad de paralelismo de la FPGA. En el caso de D-MPSS la cantidad de ciclos de reloj es variable y depende de la variable f, a cual puede ser diferente en cada salto. En el penúltimo ciclo de reloj se realiza el filtrado con los últimos bits y en el último se comprueban los resultados finales.

Se realizó la medición de las latencias obteniendo las siguientes resultados en microsegundos para la red Abilene como se observa en el Cuadro 5.3 y Figura 5.9. Para la red COST-266 como se observa en el Cuadro 5.4 y en la Figura 5.10 Para la red NSFNET como se observa en el Cuadro 5.5 y en la Figura 5.11.

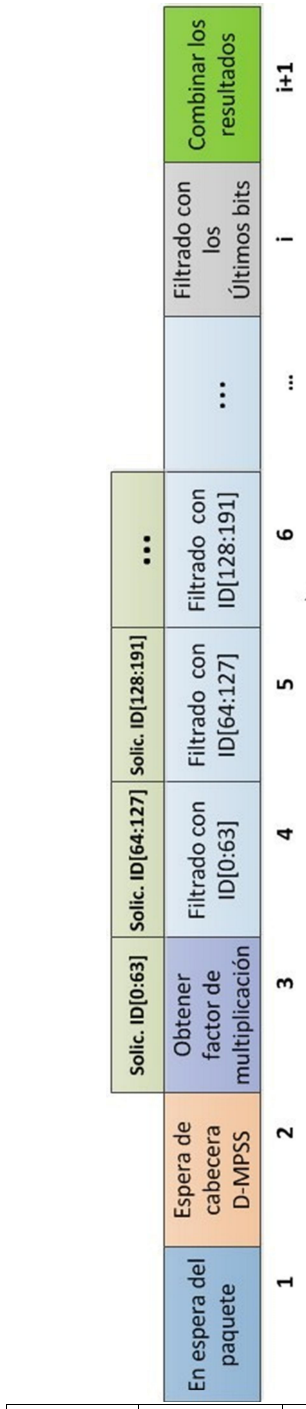

Figura5.8.: DiagramadeflujodeD-MPSS[Elaboraciónpropia].

k	4	5	6
Latencia	8.0887 μs	6.7856 μs	8.3313 μs

Cuadro 5.3.: Latencia promedio en la red Abilene.

k	4	5	6
Latencia	6.5606 μs	8.4285 μs	8.3732 μs

Cuadro 5.4.: Latencia promedio en la red COST-266.

k	4	5	6
Latencia	8.5324 μs	8.8245 μs	8.5604 μs

Cuadro 5.5.: Latencia promedio en la red NSFNET

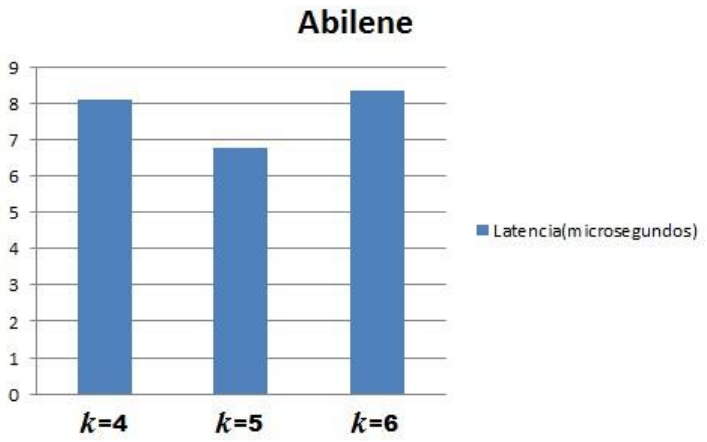

Figura 5.9.: Latencia promedio en la red Abilene [Elaboración propia].

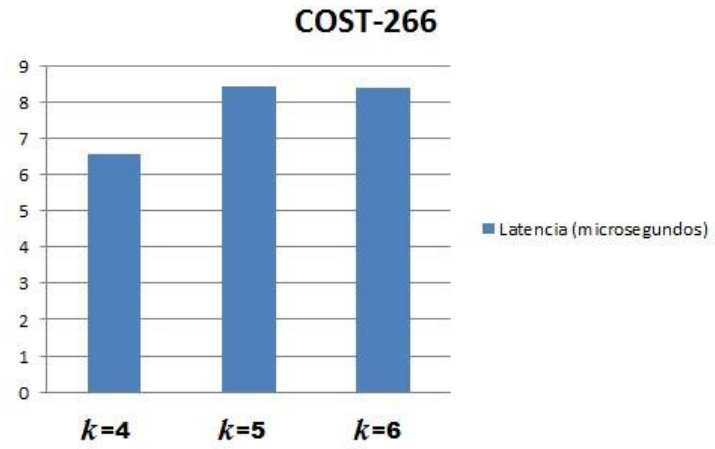

Figura 5.10.: Latencia promedio en la red COST-266 [Elaboración propia].

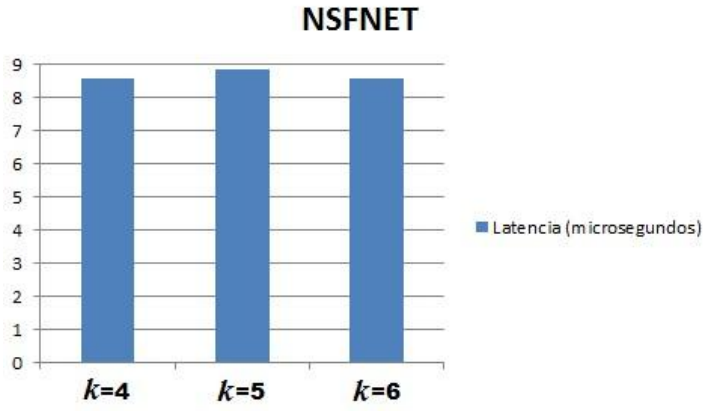

Figura 5.11.: Latencia promedio en la red NSFNET [Elaboración propia].

A pesar de variar el valor de k (cantidad de 1's en los identificadores de enlace), la latencia presentada es muy similar a causa del procesamiento no solo de la NetFPGA sino de los procesos internos del sistema y de algunos programas involucrados, quedando así que superar esas limitaciones para obtener resultados más precisos. También cada red posee distintos valores de f y $mult$ lo cual hace que no se pueda apreciar concretamente la diferencia entre cada red, es por ello que se deben tomar medidas aisladas para observar los distintos procesamientos para cada caso.

El Tabla 5.6 representa una comparación en latencias basada en la representación realizada en LIPSIN [JZR09] , utilizando en primer lugar una conexión directa (Plain wire) en un lazo entre dos puertos GbE de un PC, en segundo lugar se tienes los

Ruta	Latencia prom.	Desv. Estand.
Plain wire	94 μs	28 μs
IP router	102 μs	44 μs
LIPSIN	96 μs	28 μs
D-MPSS	96 μs	28 μs

Cuadro 5.6.: Cálculo de latencia de varias implementaciones.

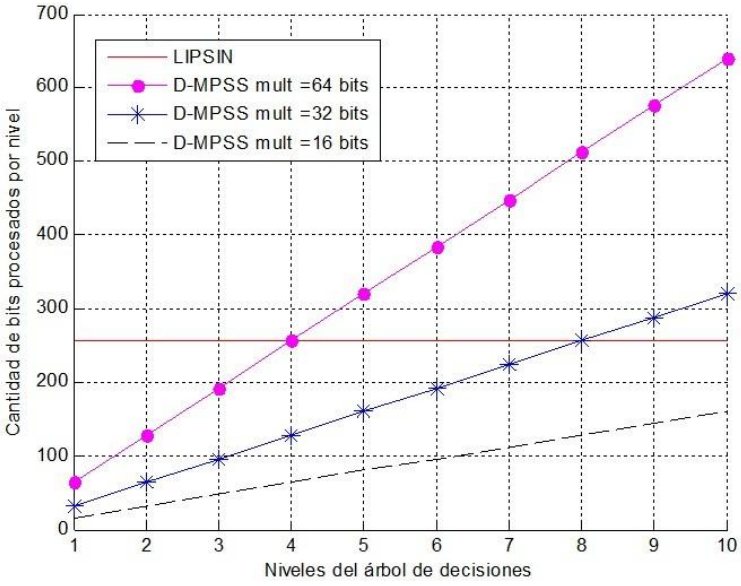

Figura 5.12.: Cantidad de bits utilizados por nivel según el método utilizado [Elaboración propia].

valores obtenido por LIPSIN con una carga de datos baja, seguidamente se realiza el enrutamiento convencional con la revisión de solo cinco tablas de enrutamiento internas para tener una perspectiva más equilibrada; por último se presentan los métodos de enrutamiento basados en filtros de Bloom LIPSIN y D-MPSS, los cuales poseen las mismas latencias ya que la comparación se realiza en un plano de microsegundos. Las diferencias más importantes entre LIPSIN y D-MPSS radican en el mayor o menor uso de ciclos de reloj, considerando que cada ciclo con un bus de 64 bits es del orden de 8 nanosegundos.

La Figura 5.12 representa una comparación entre LIPSIN y D-MPSS según la cantidad de bits utilizados en las cabeceras por cada nivel del árbol donde se encuentren. Una característica de D-MPSS es que tiene cabeceras de tamaños variables, si consideramos un árbol de decisiones jerárquico, se observa que la cantidad de bits a procesar en D-MPSS es mucho menor que con LIPSIN en los niveles superiores, que resultan ser más críticos ya que de ellos dependen las siguientes ramas. A medida que los niveles del árbol aumentan las cabeceras de D-MPSS se incrementan, sin embargo al reducir el valor de *mult* se puede apreciar que la cantidad de bits a procesar, crece con una pendiente reducida, es así que a medida que el valor de *mult* sea menor, es más probable que el crecimiento de las cabeceras conforme aumentan los niveles no sea tan drástico. A pesar

de que reducir el valor de *mult* sea favorable en la reducción de cabeceras, es necesario considerar que requiere también utilizar más recursos para realizar las decisiones, caso que no ocurriría con buses de datos más pequeños.

6. Conclusiones, contribuciones y trabajos futuros

En esta tesis se ha descrito el funcionamiento de mecanismos de enrutamiento multicast MPLS, poniendo especial atención a D-MPSS desarrollado en [FL12], entendiendo y estudiando su funcionamiento para así ser capaces de realizar su correcta implementación en hardware. El propósito inicial fue la implementación en un nodo a nivel de hardware que, gracias a la plataforma de red NetFPGA, se pudo realizar y así medir su rendimiento con tráfico emulado muy cercano a un escenario real, determinando la latencia causada por el nodo de reenvío.

Para lograr la correcta implementación se realizó un estudio exhaustivo a la plataforma NetFPGA y se determinó que era adecuada para este trabajo por su reciente uso en trabajos de investigación presentados en congresos internacionales en el ámbito de redes.

En el análisis se muestra que es posible utilizar cabeceras con filtros de Bloom de tamaños variables en los paquetes, para ser analizadas dentro de cada nodo de reenvío. Así mismo se comprueba la fiabilidad de retirar en cada salto parte de la cabecera, permitiendo que el siguiente nodo no tenga problemas en procesar el paquete. También se determinó un método con el cual es posible realizar las decisiones de conmutación sin necesidad de restar rendimiento ni acumular ciclos de reloj innecesarios. En base al código tratado en [KJS09] se logró optimizar la generación de paquetes, de tal manera que tengan la estructura que D-MPSS requiere.

Según el análisis realizado la implementación de D-MPSS minimiza las anomalías de reenvío presentadas en MPSS, sin hacer uso de una cabecera TTL. También reduce la latencia presentada en especial, en los niveles superiores del árbol de decisiones. Las tablas de enrutamiento solo consistirán en las interfaces de salida sin necesidad de utilizar tablas LIT adicionales, como se usa en MPSS, las cuales llegaban a utilizar hasta cuatro veces el número de interfaces de salida.

Si bien es cierto, los resultados también indican un mayor consumo de recursos por parte de D-MPSS; sin embargo, la diferencia no es significativa considerando los beneficios que se pueden tener al momento de realizar las decisiones en niveles superiores del árbol, que son las más críticos ya que de ellos depende el resto de niveles. De esta manera los niveles superiores procesan cabeceras comprimidas a tal punto que puedan ser menores o iguales al ancho del bus y por lo tanto realizar las comparaciones en una cantidad reducida de ciclos de reloj que se traducen en menores latencias.

Contribuciones

Se desarrolló un paper basado en la presente tesis denominado: Implementation of stateless routing mechanisms for multicast traffic on NetFPGA card, presentado en el 2015 IEEE Colombian Conference on Communications and Computing, COLCOM 2015.

Martínez-Aguilar R.B., Fernández G.M. Implementation of stateless routing mechanisms for multicast traffic on NetFPGA card. 2015 IEEE Colombian Conference on Communications and Computing, COLCOM 2015 - Conference Proceedings, art. no. 7152107 [RG15]

Abstract: Recent researches have studied how to support the online distribution of multimedia contents efficiently, reducing almost completely the state information within switching nodes. However, many of these studies have been carried out only in a theoretical way and have been verified by simulations, without actual hardware implementations, which will reveal their true performance. One of the main proposals consists on changing the current paradigm in transport networks, by encoding the path information into packet headers using Bloom filters to virtually eliminate the routing tables. In this paper we work on the implementation of a node in a NetFPGA card to evaluate the performance of MPSS and D-MPSS mechanisms (proposed in previous works), which use Bloom filters into the packet headers. To build this, it is required to develop some techniques to work with header Bloom filters of variable size, which need compression and decompression in the fastest possible way, optimizing the resources used during the switching decisions. Performance measurements are provided to verify the forwarding efficiency, considering latencies and the comparison between both methods, in which an advantage for D-MPSS is checked.

Con la tesis presentada se ganó una de las becas PEIT 2013, organizada La Dirección de Investigación de la Universidad Católica San Pablo.

En base al trabajo realizado y con el conocimiento de la herramienta utilizada para la implementación se presentó un tutorial en el Latincom 2015 con el nombre: Introduction to the Reconfigurable Networking Hardware Platform, NetFPGA. Detallando su puesta en marcha y el funcionamiento.

Con el fin de profundizar el conocimiento en la tarjeta NetFPGA y realizar trabajos futuros que complementen la presente tesis, el autor fue uno de los seleccionados para la V Movilización Nacional e Internacional en CTI organizada por el CONCYTEC, Consejo Nacional de Ciencia, Tecnología e Innovación Tecnológica, con el fin de realizar una pasantia entre Enero y Marzo del 2016 en la Universidad Autonoma de Bucaramanga, Colombia.

Trabajos futuros

Como trabajo futuro, se plantea mejorar el método de decisión, de tal manera que permita considerar en las cabeceras múltiplos menores a 64 bits, ya que al momento

de implementar cabeceras con múltiplos menores al ancho del bus de la tarjeta, la cantidad de operaciones de comparación se incrementa considerablemente. También se plantea agregar sistemas de seguridad mediante cifrado, como se hizo en [GH11], y métodos para evitar ataques de denegación de servicio en la red. Por último sería muy interesante desarrollar pruebas con distintas configuraciones de tres nodos para evaluar el rendimiento de la NetFPGA en determinadas condiciones.

A. Anexo

Presupuesto

Para la ejecución de la presente tesis se contó con herramientas existentes en los laboratorios:

1. Una tarjeta NetFPGA.
2. Un PC con capacidad para soportar la tarjeta NetFPGA con sistema operativoFedora.
3. Cables ethernet.
4. Herramientas CAD de software como Xilinx y Multisim

Siendo propuestos para la complementación del trabajo los siquientes materiales con sus respectivas cotizaciones:

Material	Cotización
3 tarjetas NetFPGA	$ 600 c/u
CPU con procesador potente para soportar las 3 tarjetas NetFPGA	$ 700
Tarjeta NIC dual Giga Ethernet (para hacer test de regresión)	$ 40
Total	$ 2540

Cuadro A.1.: Cotización de materiales para complementar la tesis

B. Anexo

Puesta en marcha de la tarjeta NetFPGA

Para realizar la instalación de la tarjeta fue necesario la instalación del S.O. Fedora 13 Figura B.1, con NetFPGA disponible en la página de la tarjeta: netfpga.org y CentOS 5.5 Figura B.2, para versiones antigüas de modelos de referencia.

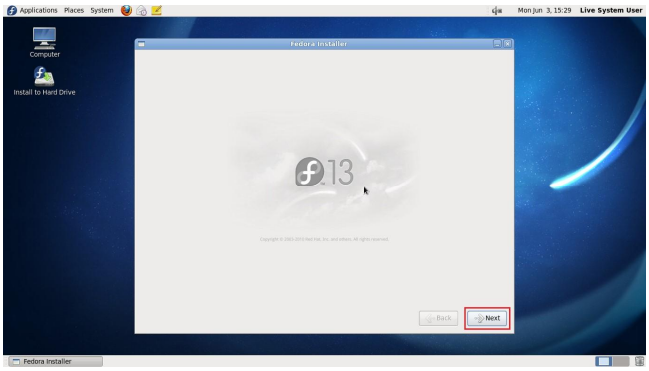

Figura B.1.: Fedora 13.

A continuación la instalación de las herramientas CAD Xilinx y Modelsim para Linux. Despues:

1. Descargar la última versión del NetFPGA packaged
2. Descomprimir y copiar la carpeta netfpga a home
3. Agregar las variables de entorno de la NetFPGA
4. Instalación de módulos de memoria para simulación
    a) Micron DDR2 SDRAM
    b) Cypress SRAM

Para la verificación:

1. Verificación de las interfaces
2. Re-programar el CPCI
3. Ejecutar el Selftest
4. Ejecutar los test de regresión

Figura B.2.: CentOS 5.5.

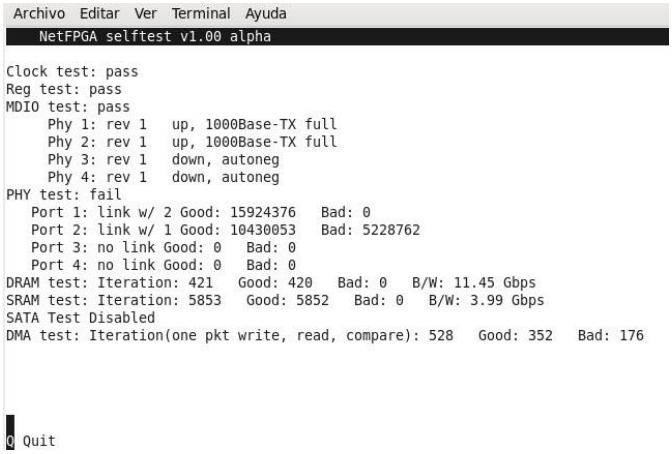

Figura B.3.: Resultado del Test self con solo un cable Ethernet conectando dos interfaces de la tarjeta

C. Anexo

Códigos

Output Queues

```
    // Realizar las operaciones requeridas a los paquetes de salida
    // a partir de la posición 3 se retira la cabecera actual y se reordena los datos de salida

        assign temporal_B_data_in = (position == ind_b) ?
input_fifo_data_out : 0;

        //assign modified_bram_oq_data_in = (position == ind_b) ?
{temporal_A_data_in[55:0], temporal_B_data_in[63:56]} : 0;

        assign modified_bram_oq_data_in = (conmutador == 0) ? ((position == ind_b) ?
{temporal_A_data_in[55:0], temporal_B_data_in[63:56]} : 0) : ((position == ind_b) ?
{temporal_A_data_in[23:0], temporal_B_data_in[63:24]} : 0);

        //assign bram_oq_data_in = (collected_index == 0) ? input_fifo_data_out : {ttl};

assign bram_oq_data_in = (collected_index == 0) ? input_fifo_data_out :
modified_bram_oq_data_in;
```

Output Port Selector

```
                // Operaciones de comparacion, a partir de zfiltering para D-MPSS for(i=0;i<4;i=i+1)

      begin for(j=0;j<4;j=j+1) begin
```

```
  selections_a[i][j] <= selections_a[i][j] & (!((header_counter - 3)
== mod1(j*4,(mult/2)) ) | (({temp_in_data[55:0],in_selection_data[63:56]}
& filt_A[j+(i*4)][63:0]) == filt_A[j+(i*4)][63:0]));

end end for(i=0;i<4;i=i+1) begin

      for(j=0;j<4;j=j+1) begin

  selections_b[i][j] <= selections_b[i][j] & (!((header_counter - 3)
== mod1((j*4)+1,(mult/2)) ) | (({temp_in_data[55:0],in_selection_data[63:56]}
& filt_B[j+(i*4)][63:0]) == filt_B[j+(i*4)][63:0]));
```

```
end end for(i=0;i<4;i=i+1) begin
        for(j=0;j<4;j=j+1) begin

selections_c[i][j] <= selections_c[i][j] & (!((header_counter - 3)
== mod1((j*4)+2,(mult/2)) ) | (({temp_in_data[55:0],in_selection_data[63:56]}
& filt_C[j+(i*4)][63:0]) == filt_C[j+(i*4)][63:0])); end end

for(i=0;i<4;i=i+1) begin for(j=0;j<4;j=j+1) begin

selections_d[i][j] <= selections_d[i][j] & (!((header_counter - 3)
== mod1((j*4)+3,(mult/2)) ) | (({temp_in_data[55:0],in_selection_data[63:56]}
& filt_D[j+(i*4)][63:0]) == filt_D[j+(i*4)][63:0])); end end

  last_count <= count+last_count; end
```

```
header_counter <= header_counter+1;
temp_in_data[55:0] <= in_selection_data[55:0];
```

Modificación del codigo en Java para convertir los datos a información legible por la NetFPGA en un entorno Linux

```
public void record(ArrayList<Short[]> stack, int deep, int maxDeep) { try { fichero = new
FileWriter("d:/nodos/nodo_" + name + ".txt", true); pw = new PrintWriter(fichero);
this.pw.println("# N filters = " + stack.size()); this.pw.println("# deep: " + deep);
this.pw.println("# Max. deep: " + maxDeep); this.pw.println("inicio_ns=‘date +%s%N‘");
this.pw.println("inicio=‘date +%s‘");

for (int i=(maxDeep - deep - 1); i<(maxDeep - deep); i++) { this.pw.println("# filter " + i + " = " +
stack.get(i).length); this.pw.print("time /ruta/psm send packet ttl "
+ ((stack.get(i).length)/64)*2
+ " if $IFACE2 cnt 1 delay 0 len 1000 output "); for (int j=0;
j<stack.get(i).length; j++) { this.pw.print(stack.get(i)[j]); }
this.pw.println();
this.pw.println("fin_ns=‘date +%s%N‘"); this.pw.println("fin=‘date +%s‘");
this.pw.println("let total_ns=$fin_ns-$inicio_ns"); this.pw.println("let total=$fin-$inicio");
```

this.pw.println("echo -$total_ns- nanosegudos, -$total- >>/root/file.txt"); this.pw.println(); } this.pw.println("echo #######"); this.pw.close();

} catch (Exception e) { e.printStackTrace(); } }

Bibliografía

[AGS12] Álvaro García Salinas. *Creación de registros NetFlow usando la plataforma NetFPGA*. PhD thesis, Universidad Autónoma de Madrid, 2012.

[Bli14] Activision Blizzard. Annual report 2014. Technical report, Activision Blizzard, 2014.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13(7): 422-426*, pages 422–426, 1970.

[BM03] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet Math 1,* pages 485–509, 2003.

[CKM+05] Jun-Hong Cui, Jinkyu Kim, Dario Maggiorini, Khaled Boussetta, and Mario Gerla. Aggregated multicast: A comparative study. *Cluster Computing*, 8: 15–26, 2005.

[ER] Multicast in mpls/bgp ip vpns (rfc 6513), Internet Engineering Task Force (IETF).

[FL12] Gonzalo M. Fernandez and David Larrabeiti. Depth-wise multi-protocol stateless switching of multicast traffic. 2012.

[GH11] Ghani and Adnan Hassan. *Secure In-packet Bloom Filter Based Forwarding on a Reusable Network Hardware Design*. PhD thesis, Aalto University, 2011.

[GN10] Adnan Hassan Ghani and Pekka Nikander. Secure in-packet bloom filter forwarding on the netfpga. *1st European NetFPGA Developers Workshop, University of Cambridge, Computer Laboratory, Cambridge, UK*, 2010.

[GNH+08] Glen Gibb, Jad Naous, Paul Hartke, John W. Lockwood, and Nick McKeown. Netfpga: Open platform for teaching how to build gigabit-rate network switches and routers. *IEEE Transactions on Education*, 2008.

[HL05] Yian Huang and Wenke Lee. Hotspot-based traceback for mobile ad hoc networks. *in Proceedings of the ACM Workshop on Wireless Security (WiSe'05)*, 2005.

[JZR09] P Jokela, A. Zahemszky, and C. Esteve Rothenberg. Lipsin: line speed publish/subscribe inter-networking. *SIGCOMM '09 Proceedings of the*

ACM SIGCOMM 2009 conference on Data communication, pages 195– 206, 2009.

[KJS09] J. Keinänen, P. Jokela, and K. Slavov. Implementing zfilter based forwarding node on a netfpga. *NetFPGA Developers Workshop*, https://github.com/NetFPGA/netfpga/wiki/ZFilter, 2009.

[LMW+07] John W. Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad

an open platform for gigabit-rate network switching and routing. *MSE*, 2007.

[LSSI05] J. Liu, F. Sailhan, D. Sacchetti, and V. Issarny. Group management for mobile ad hoc networks: Design, implementation and experiment. *Proceedings of the 6th IEEE International Conference on Mobile Data Management (MDM'2005)*, 2005.

[McC76] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, pages 308–320, 1976.

[Mit01] Michael Mitzenmacher. Compressed bloom filters. *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 144–150, 2001.

[MYLS07] Isaias Martinez-Yelmo, David Larrabeiti, and Ignacio Soto. Multicast

Naous, Ramanan Raghurama, and Jianying Luo. Netfpga traffic aggregation

in mpls-based vpn networks. *IEEE Communications Magazine*, 45(10): 78–85,

2007.

[NGBM08] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. Netfpga: Reusable router architecture for experimental research. *SIGCOMM PRESTO Workshop*, 2008.

[PPL05] E. Papapetrou, E. Pitoura, and K. Lillis. Speeding up cache lookups in wireless ad hoc routing using bloom filters. *the 16th Annual International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC 2005)*, 2005.

[RG15] Martinez Aguilar R.B. and Fernandez G.M. Implementation of stateless routing mechanisms for multicast traffic on netfpga card. 2015. cited By 0.

[RR06] E. Rosen and Y. Rekhter. Bgp/mpls ip virtual private networks (vpns). Rfc 4364 (standard): http://www.ietf.org/rfc/rfc4364.txt, IETF, Feb. 2006.

[Sol01] Karl Solie. *CCIE Practical Studies Volume I*. 2001.

[Sol03] Karl Solie. *CCIE Practical Studies Volume II*. 2003.

[SRA+11] M. Sarela, C.E. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, and J. Ott. Forwarding anomalies in bloom filter-based multicast. *INFOCOM, 2011 Proceedings IEEE*, 2011.

[ZJS+10] A. Zahemszky, P. Jokela, M. Sarela, S. Ruponen, J. Kempf, and P. Nikander. Mpss: Multiprotocol stateless switching. *INFOCOM IEEE Conference on Computer Communications Workshops , 2010*, 1 - 6, 2010.

Nomenclatura

ATM	Asynchronous Transfer Mode
BF	Bloom Filter
CE	Customer Edge
CIDR	classless interdomain routing
D-MPSS	Depth-Wise Multi-Protocol Stateless Switching
FIFO	First in, first out
FPGA	Field Programmable Gate Array
fpr	False positive rate
GbE	Gigabit Ethernet
ICMP	Internet Control Message Protocol
IP Multicast	Internet Protocol Multicast
IPTV	Internet Protocol Television
ISP	Internet service provider
KDF	Key Derivation Function
LIPSIN	Line Speed Publish/Subscribe Inter-Networking
LSP	Label Switched Path
LSR	label switch router
MMORPG	Massively Multiplayer Online Role-Playing Game
MPLS	Multi-Protocol Label Switching
MPSS	Multiprotocol Stateless Switching
NIC	Network Interface Card

PE	Provider Edge
PSM	PubSubManager
RP	Rendezvous Point

RSS	Really Simple Syndication
SP	Service Provider
SPTs	shortest path trees
TE	Traffic engineering
VoD	Video on Demand
VPLS	Virtual Private LAN Service
VPN	Virtual Private Network
VRF	Virtual Routing and Forwarding