Routers Reconfigurables de Altas Prestaciones

Proyecto de Grado

Autores:
Emiliano Viotti
Rodrigo Amaro

Supervisores:
Dr. Eduardo Grampín

MSc. Martín Giachino

Instituto de Computación

Facultad de Ingeniería, Universidad de la República

Agosto de 2015
Montevideo, Uruguay

Emiliano quiere dedicar
este trabajo a
su hermano Andrés
y a sus abuelos.

Rodrigo quiere dedicar
este trabajo a
su amor Sofía,
a su padre Jorge
y su madre Silvia.

Agradecimientos

A todos aquellos que estuvieron presentes en los buenos momentos y en especial en los malos, sean familiares, amigos o compañeros de carrera. Agradecemos a todos los integrantes del MuDi por el apoyo incondicional en el día a día. A nuestros trabajos y novias que nos comprendieron en momentos de ausencia con infinita paciencia. Además agradecemos a nuestros tutores, Eduardo y Martín por el permanente apoyo a lo largo de este proyecto, por la ayuda brindada en los momentos más difíciles, a todo el grupo de investigación MINA por prestarnos una oficina en la cual trabajar, a la Agencia de Investigación e Innovación (ANII) y al programa Frida por el apoyo económico brindado.

Resumen

Actualmente, desarrollar aplicaciones de control de red, manipular el contenido de un paquete e incluso algo tan simple como analizar un cabezal de paquete es una tarea difícil que requiere trabajar a un nivel de abstracción muy bajo y con un nivel de conocimiento bastante específico, trabajando con herramientas no propietarias y de código abierto basadas en Linux (por ejemplo OpenWRT). Por otro lado si se quiere hacer esto en un contexto comercial con tecnologías propietarias, solo es posible si el equipo o API de funcionalidades disponible así lo permite. Este escenario no hace otra cosa que entorpecer y ralentizar el desarrollo de nuevos protocolos y servicios, así como la innovación en el área. El modelo de las Redes Definidas por Software (SDN) desacopla los planos de control y datos a la vez que estandariza la forma en que cualquier equipo de red puede ser manipulado y así como también el tráfico que estos procesan. De esta forma se generan las condiciones para que cualquier equipo pueda ser utilizado de forma transparente en la definición de nuevos servicios y protocolos independientemente del fabricante, facilitando así la innovación en el área.

En este proyecto se construye un prototipo de red de backbone MPLS utilizando el enfoque SDN y placas de red aceleradas en hardware reconfigurable (NetFPGA), orientado a la provisión de redes privadas virtuales como servicio, de cara a lo que podría ser una posible implementación de la nueva red académica RAU2. Se diseña e implementa un equipo enrutador de red denominado RAU-Switch, compatible con las tecnologías IP, MPLS y OpenFlow, orientado a convivir con equipos legados y a un bajo costo económico. Además se desarrolla un conjunto de aplicaciones de control y gestión de red denominado RAUFlow y se contribuye a la comunidad en el desarrollo de documentación exhaustiva en relación a tecnologías muy recientes de las que algunas poco se conoce y otras aún están en fases experimentales.

Palabras clave: Redes de Computadoras, SDN, OpenFlow, NetFPGA, MPLS, Open vSwitch.

Tabla de Contenidos

Tabla de Figuras **xiii**

Tabla de Cuadros **xvii**

Glosario **1**

1 Introducción **5**
1.1 Motivación 5
1.2 Definición del Problema 6
1.3 Objetivos 7
1.4 Resultados Esperados 7
1.5 Estructura del documento 7

2 Estado del Arte **9**
2.1 Antecedentes de SDN 9
2.1.1 Open Signaling 10
2.1.2 Active Networking 10
2.1.3 Routing Control Platform 10
2.1.4 ForCES Protocol 11
2.1.5 Proyecto 4D 11
2.2 Software Defined Networking 12
2.3 Arquitecturas basadas en SDN 14
2.3.1 OpenFlow 14
2.3.2 Reglas OpenFlow 16
2.3.3 Acciones OpenFlow 16
2.3.4 Controlador OpenFlow 17
2.4 Herramientas SDN/OpenFlow disponibles 20
2.4.1 Controlador 20
2.4.2 Mininet y miniNExT 22
2.5 Aplicaciones de SDN 22
2.6 Casos de éxito de SDN/OpenFlow 23

2.7 Red Privada Virtual . . . 24
2.7.1 Redes Privadas en Uruguay y proyecciones para la RAU . . . 26
2.8 Multiprotocol Label Switching . . . 27
2.9 NetFPGA . . . 28

3 Análisis del problema **31**
3.1 Definición de requerimientos . . . 31
3.2 ¿Por qué utilizar SDN? . . . 32
3.3 ¿Por qué utilizar NetFPGA? . . . 33
3.4 ¿Qué arquitectura basada en SDN utilizar? . . . 34
3.5 Alternativas de diseño para el router . . . 34
3.6 ¿Qué implementación de Controlador OpenFlow utilizar? . . . 36

4 Diseño e Implementación del prototipo **39**
4.1 Lineamientos generales . . . 39
4.1.1 Algoritmo de Ruteo . . . 39
4.1.2 Algoritmo de distribución de etiquetas . . . 40
4.2 Construcción del prototipo y arquitectura . . . 41
4.3 RAU-Switch . . . 42
4.3.1 Plataforma de la PC . . . 43
4.3.2 Sistema Operativo . . . 43
4.3.3 Hardware NetFPGA . . . 43
4.3.4 Open vSwitch . . . 45
4.3.5 Quagga . . . 45
4.3.6 Agente SNMP . . . 46
4.4 Entidad Controlador . . . 46
4.4.1 Controlador Ryu . . . 47
4.4.2 Quagga . . . 47
4.4.3 LSDB Sync . . . 47
4.4.4 Administrador SNMP . . . 48
4.4.5 Visión general de la arquitectura . . . 48

5 RAUFlow **49**
5.1 Análisis de requerimientos . . . 49
5.2 Modelado de la realidad . . . 50
5.3 Casos de uso . . . 52
5.4 Arquitectura de RauFlow . . . 54
5.4.1 Capa de Presentación . . . 55
5.4.2 Capa de Control . . . 56
5.4.3 Capa de Dispositivos . . . 59

5.5 Implementación . . . 59
5.5.1 Clasificación de tráfico . . . 60
5.5.2 Implementación de Servicio . . . 60
5.5.3 Algoritmo de ruteo . . . 62
5.5.4 Algoritmo de distribución de etiquetas . . . 65
5.5.5 Actualización de la topología . . . 69
5.5.6 Ciclo de vida de un Nodo . . . 72
5.5.7 QoS en RAUFlow . . . 75

6 Laboratorio de pruebas 77
6.1 Definición del Laboratorio . . . 77
6.2 VPN de capa 3 . . . 79
6.2.1 Escenario 1 . . . 79
6.2.2 Escenario 2 . . . 84
6.3 VPN de capa 2 . . . 85
6.3.1 Escenario . . . 86

7 Gestión del proyecto 89
7.1 Ejecución del proyecto . . . 89

8 Conclusiones 93
8.1 Conclusiones . . . 93
8.2 Trabajo a futuro . . . 94

Bibliografía 97

Apendice A Solución a errores encontrados en la plataforma NetFPGA 103
A.1 Problema con la herramienta pcieprog y ReferenceNIC . . . 103
A.2 Error en el driver para el proyecto ReferenceNIC . . . 104

Apendice B Desafíos técnicos encontrados 107
B.1 Problemas con SFPs y Patchcoords . . . 107
B.2 Desprogramación del hardware NetFPGA . . . 107
B.3 Falta de licencias para suite de Xilinx ISE SDK . . . 108
B.4 Falta de driver para cable JTAG Xilinx . . . 109
B.5 Problemas técnicos con Open vSwitch . . . 109
B.6 MPLS Linux y Quagga LDP . . . 112

Apendice C Clasificación de tráfico utilizando cabezales OpenFlow **113**
C.1 Cabezal OpenFlow versión 1.3.1 . . . 113
C.2 Sintaxis de un flujo en Open vSwitch . . . 114
C.3 OpenFlow v1.3.1 atributos soportados para la definición de reglas . . . 114
C.4 OpenFlow v1.3.1 atributos soportados para definir acciones . . . 118

Apendice D Archivos de configuración **121**
D.1 Archivos de configuración Quagga . . . 121
D.2 Archivos de configuración de Open vSwitch . . . 124
D.3 Otros scripts de configuración . . . 126

Apendice E Ejecución de RAUFlow **127**

Apendice F Detalles adicionales sobre pruebas realizadas **129**
F.1 Verificación del algoritmo de ruteo en VPN de capa 3 escenario 1 . . . 129
F.2 Clasificación de tráfico en VPN de capa 3 escenario 1 . . . 133
F.3 Actualización de Rutas en VPN de capa 3 escenario 1 . . . 135
F.4 Numeraciones IP solapadas en VPN de capa 3 escenario 2 . . . 138
F.5 Procesamiento del tráfico en VPN de Capa 2 . . . 141

I Manual de construcción de RAU-Switch **145**
I.1 Resumen . . . 145
I.2 Plataforma utilizada . . . 145
I.3 Instalación y configuración . . . 146
I.3.1 Instalación de sistema operativo . . . 147
I.3.2 Instalación de librerías y dependencias . . . 147
I.3.3 Instalación de suite de desarrollo Xilinx ISE SDK . . . 148
I.3.4 Configuración del entorno de desarrollo NetFPGA . . . 150
I.3.5 Pruebas de aceptación del hardware NetFPGA . . . 150
I.3.6 Programación de la tarjeta . . . 158
I.3.7 Instalación de Open vSwitch . . . 161
I.3.8 Instalación de software de ruteo Quagga . . . 162
I.3.9 Instalación de agente SNMP . . . 163

Anexos **144**

Tabla de Figuras

2.1 Capas de la arquitectura 11
2.2 Arquitectura de SDN - Capas lógicas 13
2.3 Estructura de un switch OpenFlow 14
2.4 Visión esquemática del funcionamiento de un Switch OpenFlow 15
2.5 Ciclo de vida de un paquete en pipe OpenFlow 15
2.6 OF 1.0 Matching Fields 16
2.7 OF 1.3.3 Matching Fields 16
2.8 Arquitectura de OpenDaylight Hydrogen 18
2.9 Esquema de una VPN Punto a Punto 25
2.10 Esquema de una VPN Multipunto L3 26

4.1 Esquema general del prototipo 41
4.2 RAU-Switch - diagrama de componentes 42
4.3 Diagrama de componentes del Controlador 47
4.4 Vista lógica ampliada del prototipo 48

5.1 Modelo de datos 51
5.2 Casos de Uso de RAUFlow 54
5.3 Arquitectura de RAUFlow 55
5.4 Algoritmo de ruteo centralizado sin restricciones (SPF) 63
5.6 Algoritmo de distribución de etiquetas 66
5.8 Algoritmo de actualización topológica 70
5.11 Ciclo de vida de un Nodo 73

6.1 Laboratorio de pruebas - Topología 78
6.2 Laboratorio de pruebas - Costos de enlaces en la topología 79
6.3 VPN de capa 3 - Escenario 1 80
6.4 Escenario 1 - LSPs 82
6.5 Escenario 1 - Caminos para servicios recalculados 83
6.6 VPN de capa 3 - Escenario 2 84

6.7 VPN de capa 2 Escenario 86
6.8 Paquetes generados con PacketH 88

7.1 Diagrama de Gantt, Actividades del Proyecto 89

B.1 Funcionamiento de puertos Open vSwitch 110
B.2 Puertos Open vSwitch - Solución propuesta 111

E.1 Ejecución RAUFlow 127
E.2 Capturas de interfaz gráfica de RAUFlow 128

F.1 Tabla de flujos ovs - Galois 130
F.2 Tabla de flujos ovs - Oz 130
F.3 Tabla de flujos ovs - Poisson 130
F.4 Tabla de flujos ovs - Alice 131
F.5 Flujo 1 131
F.6 Flujo 2 131
F.7 Capturas de tráfico con tcpdump - servicio S3 133
F.8 Comando Ping H1 Subread A - H3 Subred B 134
F.9 Capturas de tráfico con tcpdump - servicio S4 134
F.10 Tabla de flujos ovs - Galois 135
F.11 Tabla de flujos ovs - Oz 135
F.12 Tabla de flujos ovs - Alice 135
F.13 Tabla de flujos ovs - Poisson 136
F.14 Capturas de tráfico con tcpdump - servicio S3 actualización topológica 137
F.15 Comando Ping H1 Subread A - H3 Subred B 138
F.16 Tabla de flujos ovs - Galois 139
F.17 Tabla de flujos ovs - Oz 139
F.18 Tabla de flujos ovs - Poisson 139
F.19 Tabla de flujos ovs - Alice 140
F.20 SSH desde Subred A hacia Subred B 140
F.21 SSH desde Subred A' hacia Subred B' 141
F.22 Caso de Uso 2 capturas 142
F.23 Paquetes IPv4 - Wireshark 142
F.24 CPaquetes IPv6 - Wireshark 143
F.25 Paquetes ARP - Wireshark 143

I.1 Jumper J16 151
I.2 SW9 en posicion PCIe 152
I.3 Posiciones DIP SW1 SW2 SW6 SW10 152
I.4 Conector ATX 153

I.5 Cable samtec twinax . . . 153
I.6 Cables samtec twinax conectados . . . 153
I.7 Cable serial puerto COM . . . 154
I.8 Cable JTAG . . . 154
I.9 Tarjeta NetFPGA esquema de instalación . . . 154
I.10 Tarjeta NetFPGA instalada en una PC . . . 155
I.11 Reserva de bloque de memoria . . . 156
I.12 Programación Impact . . . 156
I.13 Reserva bloque de memoria . . . 157
I.14 Test de Producción salida esperada . . . 158
I.15 Configuración SW3 JTag/PCIe programing . . . 159

Tabla de Cuadros

2.1 Principales controladores disponibles 21

3.1 OpenFlow NetFPGA vs ReferenceNIC - Ventajas 35
3.2 OpenFlow NetFPGA vs ReferenceNIC - Desventajas 36

6.1 VPN capa 3 - Escenario 1, servicios creados 81
6.2 VPN capa 3 - Escenario 2, servicios creados 85
6.3 VPN de Capa 2, servicios creados 87

I.1 RAU-Switch, especificaciones de hardware 146
I.2 RAU-Switch, especificaciones de software 146

Glosario

A

AS Autonomous System, conjunto de redes bajo una administración común que comparten una misma estrategia de ruteo., p. 12.

ASIC Application-Specific Integrated Circuit, chip diseñado para una aplicación específica., p. 10.

B

BGP Protocolo de ruteo intradominio, permite intercambiar información de alcanzabilidad con otros sistemas BGP., p. 25.

BIRD Suite de ruteo IP Open Source., p. 40.

bitfile Archivo con toda la información necesaria para configurar un chip FPGA con la implementación del diseño creado, a partir de las herramientas utilizadas (por ejemplo suite Xilinx ISE SDK)., p. 104.

C

calidad de servicio Calidad de servicio, rendimiento promedio de una red., p. 5.

CPE Customer provisioned Equipment, equipo (hardware) de un proveedor de servicios desplegado en las instalaciones de un cliente y conectado a la red del proveedor., p. 24.

CPLD Complex Programmable Logic Device, dispositivo de hardware reprogramable., p. 44.

D

DoS Ataque a un sistema o red que causa que un servicio o recurso sea inaccesible a usuarios legítimos., p. 17.

E

Ethertype Campo de 16 bits en el cabezal Ethernet que indica el protocolo que se encuentra en la capa superior., p. 68.

F

FPGA Field Programmable Gate Array, dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada "in situ" mediante un lenguaje de descripción de hardware (por ejemplo VHDL)., p. 10.

frescura Se define frescura de un dato al nivel de actualización del mismo; es decir, que tan reciente es la información., p. 11.

H

HDL Hardware Description Language, es una familia de lenguajes de programación especializados en la descripción de circuitos electrónicos., p. 28.

I

IETF Comunidad formada por grupos de trabajo de área específicas responsables de desarrollar estándares de Internet., p. 10.

IRTF Internet Research Task Force, comunidad de expertos en redes donde se tratan temas relacionados con Internet., p. 12.

IS-IS Intermediate System to Intermediate System, protocolo de ruteo dinámico IP., p. 39.

L

L2 L2 es el diminutivo de Layer 2 utilizado para referirse a la capa 2 en el modelo de capas OSI., p. 24.

L3 L3 es el diminutivo de Layer 3 utilizado para referirse a la capa 3 en el modelo de capas OSI., p. xiii.

LDP Label Distribution Protocol, protocolo utilizado por MPLS para distrubir etiquetas entre los nodos de una red., p. 28.

LSDB Link State Database, creada y utilizada por el protocolo OSPF, es la base de datos donde se almacena la topología de la red., p. 45.

M

MPLS Multi Protocol Label Switching, protocolo de red basado en etiquetas que opera entre la capa de enlace y la capa de red., p. 9.

N

NPU Network Processing Unit, unidad de procesamiento orientada a aplicaciones de red., p. 10.

O

Open vSwitch Switch de red implementado en software., p. 42.

OpenFlow Protocolo de red perteneciente a la capa de aplicación que permite la comunicación entre un controlador con dispositivos habilitados., p. 12.

OSPF Open Shortest Path First, protocolo de ruteo IP dinámico basado en algoritmos de vector de distancias., p. 39.

P

Proxy Intermediario entre las peticiones de red que realiza un cliente a un servidor., p. 19.

Q

QoS Ver calidad de servicio., p. 18.

Quagga Suite de ruteo IP Open Source., p. 40.

R

Rau-Switch Denominación de los nodos fabricados y utilizados en el marco de este proyecto., p. 42.

RauFlow Aplicacion de control y gestión desarrollada en el marco de este proyecto., p. 41.

RFC Request For Comment, documento que especifica un estándar concebido por la IETF., p. 25.

RIP Routing Information Protocol, protocolo de ruteo IP basado en algoritmo vector de distancias., p. 13.

Router Dispositivo que reenvía paquetes en base a su dirección de capa de red (L3)., p. 10.

RYU Controlador SDN compatible con el protocolo OpenFlow., p. 21.

S

SDN Software Defined Networking o Redes Definidas por Software es un enfoque alternativo que desacopla los planos de control y datos (ver capítulo 2, estado del arte)., p. 6.

Switch Dispositivo que reenvía paquetes en base a la dirección de capa de enlace (L2) de dichos paquetes., p. 10.

T

TAP Dispositivo para interceptar el tráfico en una red, por ejemplo para su posterior análisis., p. 22.

TCAM Ternary Content-Addressable Memory, memoria de alta velocidad que busca y accede a sus contenidos en un ciclo de reloj., p. 10.

V

Verilog Lenguaje de descripción de hardware., p. 28.

Capítulo 1

Introducción

1.1 Motivación

En la actualidad, sobre la red Internet conviven aplicaciones académicas, comerciales y particulares con iguales prioridades. Este escenario no es apropiado para actividades de experimentación, investigación y estudio de nuevas herramientas a gran escala. A su vez los proveedores de servicios sobre Internet, contando con que no todos sus clientes hacen uso de su infraestructura a la vez, venden más servicios de los que realmente puede soportar, lo que hace que no sea viable garantizar un ancho de banda para cada servicio. Esta situación es comúnmente conocida como "sobreventa" del ancho de banda y se agrava en horas de mayor uso de Internet (horas pico), lo cual es crítico cuando se piensa en aplicaciones que requieren de niveles de calidad de servicio garantizados. Por estas razones Internet no parece apropiada para su utilización en el contexto académico, ya que se necesita una red que permita proveer de conectividad y a su vez soportar actividades experimentales como desplegar nuevos servicios y protocolos, así como asegurar determinados parámetros de calidad de servicio.

En este contexto y en todo el mundo se vienen desarrollando desde mediados de la década de los 90's las redes académicas avanzadas de alta velocidad, con el objetivo de proveer una red que posibilite a docentes e investigadores colaborar en aplicaciones altamente demandantes de ancho de banda (educación a distancia, transferencia de grandes cantidades de información, acceso a equipos remotos, telemedicina, etc.), sin competir por este recurso con aplicaciones de naturaleza comercial. En Estados Unidos por ejemplo, el proyecto que lidera este desarrollo es Internet2 [24], en Canadá es el proyecto CA*net4 [19], en Europa el proyectos GÉANT [22] y en Asia el proyecto APAN [17]. Todas estas redes se encuentran a su vez conectadas entre sí, formando una gran red avanzada de alta velocidad de alcance mundial. En Latinoamérica, las redes académicas de Argentina, Brasil y Chile se encuentran integradas a Internet2.

En las últimas dos décadas no solo se han hecho esfuerzos para desarrollar las redes académicas nacionales, sino que también se ha trabajado arduamente en la construcción de acuerdos que permitan interconectar cada una de estas redes en una gran red académica de alcance global. Una de las iniciativas que encausó estos esfuerzos fue la denominada CAESAR (Connecting All European and

South (Latin) American Researchers), reconociendo la necesidad de crear una red troncal regional en América Latina que eventualmente se pueda conectar a GÉANT y tras la cual se llega a la conocida "Declaración de Toledo". Poco tiempo después las redes latinoamericanas se unen bajo su propia agrupación denominada CLARA (Cooperación Latino Americana de Redes Avanzadas) y adscriben a dicha declaración[1].

En Uruguay, la Universidad de la República a través del Servicio Central de Informática (SeCIU), es la fundadora de la Red Académica Uruguaya (RAU). Este proyecto persigue en el contexto de las redes académicas mundiales los siguientes objetivos: unir las Instituciones Nacionales Académicas, Universidades (pública y privadas) y Centros de Investigación del Uruguay, promover el desarrollo de Redes Académicas y Científicas donde ellas hagan falta, planificar y desarrollar una red nacional, incentivar la colaboración con iniciativas similares y conectar la RAU con Latinoamérica.

La RAU brinda servicios a instituciones educativas y de investigación, así como implementa servicios de la propia Universidad de la República a través de una red de alcance nacional. Actualmente entre facultades, escuelas, institutos y servicios de la Universidad de la República, así como diversas instituciones educativas y de investigación, son 31 las instituciones que forman parte de la RAU, las cuales hacen uso de los 64 nodos que la conforman para brindar servicios a un total aproximado de 8.000 docentes, 4.000 técnicos y 100.000 estudiantes en diferentes partes del país[2].

Por otro lado, se viene trabajando en un proyecto para reemplazar la infraestructura actual de la RAU por una red avanzada de altas prestaciones, que pueda brindar mayores y mejores servicios a las instituciones que forman parte de la misma, así como conectarse a las demás redes académicas de Latinoamérica. Este proyecto se denominó RAU2

1.2 Definición del Problema

El problema elegido como eje para el desarrollo de este trabajo es la construcción de lo que se dio en llamar RAU2, la cual estaría dotada de funciones de virtualización de redes que permitan una gran flexibilidad en su definición y uso. Como red universitaria pretende no solo mejorar el ancho de banda disponible para desarrollar las tareas comunes con cualquier otra red de esta magnitud, sino que además busca virtualizar la red como mecanismo de asignar recursos independientes a cada una de las instituciones que hacen uso de la misma (por ejemplo universidades y centros de investigación) y poder utilizarla al mismo tiempo como laboratorio de pruebas sin que esto interfiera con el funcionamiento normal de la red.

Interesa entonces trabajar en el desarrollo de un prototipo para la RAU2 utilizando como plataforma PCs con placas de red aceleradas en hardware reconfigurable, fruto de un desarrollo de la Universidad de Stanford denominado NetFPGA (Field Programmable Gate Arrays) [83] y el enfoque de las Redes Definidas por Software (SDN) [66] y [82].

[1] Datos históricos extraídos de sitio oficial de Red Clara [51].
[2] Datos de escala de la RAU aproximados brindados por técnicos del SeCIU.

1.3 Objetivos

El objetivo principal de este trabajo es la implementación de un prototipo de red de altas prestaciones para la RAU2, utilizando como punto de partida el enfoque de SDN y el hardware NetFPGA, para luego ejecutar un conjunto de pruebas que permitan validar estas tecnologías para su utilización en la posible construcción de la RAU2.

A su vez dado la ausencia de experiencias similares trabajando con estas tecnologías a nivel local y en la órbita de la academia, se busca conocer a fondo estas tecnologías, en particular la arquitectura de SDN y los lenguajes de programación relacionados, así como generar conocimiento y experiencias que puedan contribuir en trabajos a futuro. Por ello se hizo especial hincapié en la generación de material sobre el estado del arte de las redes definidas por software y una detallada documentación de la experiencia obtenida utilizando el hardware NetFPGA.

1.4 Resultados Esperados

Los resultados esperados de este proyecto son:

1. El estado del arte en el enfoque de las Redes Definidas por Software (SDN) y el hardware reconfigurable NetFPGA. En particular interesa conocer en profundidad ambas tecnologías, relevando el nivel de madurez y desarrollo de ellas y reunir suficiente información para evaluar su utilización en la construcción de un prototipo más complejo en funcionalidades y superior en rendimiento.

2. Una implementación de un prototipo de gestión y control de red utilizando estrategias de SDN, en particular enfocado en las funcionalidades y capacidades que se pretenden de la RAU2.

3. Diseño e implementación de pruebas funcionales que permitan validar el prototipo desarrollado y evaluar el potencial de las tecnologías utilizadas.

1.5 Estructura del documento

A continuación se describe la forma en que está estructurado este documento y las temáticas asociadas a cada capítulo.

El capítulo 2 presenta los resultados obtenidos en la investigación del estado del arte en SDN, NetFPGA y conceptos esenciales para el entendimiento de este trabajo; no es estrictamente necesaria la lectura de las secciones 2.1,2.5 y 2.6. Luego en el capítulo 3 se presenta el análisis del problema, en donde se incluyen las principales ventajas en la utilización del enfoque SDN y hardware NetFPGA. En el capítulo 4 se muestra el diseño general del prototipo planteado, mientras que en el capítulo 5 se presenta el diseño de RAUFlow, la aplicación que implementa el plano de control SDN. Se incluyen en este capítulo los casos de uso implementados en RAUFlow, el modelo de datos diseñado

y detalles de la arquitectura e implementación. En cuanto al capítulo 6, en él se presenta el laboratorio de pruebas diseñado para validar funcionalmente el prototipo junto con los resultados obtenidos. Le sigue el capítulo 7 en donde se ofrece una evaluación de los resultados más importantes, así como un breve estudio de la ejecución del proyecto. Finalmente en el capítulo 8 se presentan las conclusiones obtenidas a partir del desarrollo de este trabajo y las principales líneas de trabajo e investigación identificadas como trabajo a futuro.

Adicionalmente se incorpora el capítulo de referencias con la información bibliográfica del material utilizado en el desarrollo de este trabajo y un capítulo de apéndices. También se anexa un manual para la construcción del dispositivo que se dio a llamar RAU-Switch, orientado a la reproducción y posible mejora a futuro del trabajo realizado.

Capítulo 2

Estado del Arte

En este capítulo se presentan los resultados obtenidos en el trabajo de investigación sobre el estado del arte de las Redes Definidas por Software (Software Defined Networking), lenguajes de programación y herramientas para el desarrollo sobre dicha arquitectura, hardware NetFPGA y otros conceptos esenciales para el desarrollo de este trabajo. En caso que el lector desee profundizar en alguno de los conceptos presentados en este capítulo o consultar enfoques alternativos y/o complementarios se recomiendan los siguientes trabajos [84] y [64]. Para el caso particular en que el lector esté familiarizándose con el enfoque de las Redes Definidas por Software, se sugiere también la lectura de [82] y [53].

En relación a cómo se estructura este capítulo, primero se introduce el concepto de Software Defined Networking (SDN) analizando sus principales antecedentes para luego presentar una definición del concepto. Luego se presenta una de las arquitecturas existentes basadas en este enfoque y en particular utilizada en trabajo para la construcción de un prototipo. Se continúa con la introducción de los conceptos Redes Privadas Virtuales y Multilabel Protocol Switch (MPLS) esenciales para el entendimiento de este trabajo. Finalmente se presenta el relevamiento en el estado del arte realizado sobre la plataforma NetFPGA.

2.1 Antecedentes de SDN

Software Defined Networking, Redes Definidas por Software en español o simplemente SDN por su sigla en inglés, es un concepto relativamente nuevo ya que bajo este nombre puede contextualizarse en la última década. No obstante muchas de las ideas que yacen detrás de este enfoque han sido introducidas por diferentes propuestas en el pasado. Por ello vale la pena repasar los principales antecedentes que desde nuestra perspectiva contribuyeron al enfoque de SDN.

2.1.1 Open Signaling

En el año 1995 el grupo de investigación OPENSIG [56] dentro del IETF, comenzó a trabajar en alternativas para hacer ATM, Internet y las redes móviles "más" abiertas, extensibles y programables. El principal objetivo de este grupo fue proveer de un acceso al hardware de red (Routers, Switches, etc) mediante un conjunto de interfaces abiertas. Con esto se buscaba permitir el despliegue de nuevos servicios de red a través de un ambiente de programación distribuido.

El principal desafío con el que tuvo que lidiar esta iniciativa fue la estricta arquitectura en forma vertical que presenta el hardware de red (modelo de capas OSI), cuya naturaleza a su vez solía ser cerrada. Esto dificultaba una estricta separación entre el software de control y el hardware.

Los esfuerzos de esta iniciativa desembocaron en la especificación del General Switch Managment Protocol (GSMP); un protocolo de propósito general para control de etiquetas en switches. GSMP permitía a un controlador establecer y liberar conexiones en un dispositivo conmutador de tramas (mejor conocido como switch), agregar, eliminar y dejar una conexión multicast, manejar los puertos de un switch, obtener información acerca de la configuración, obtener y eliminar recursos reservados en un switch y obtener información estadística.

Tras años de trabajo, el grupo consideró concluida su labor, siendo su último trabajo publicado la especificación de GSMPv3 en Junio del 2002[59].

2.1.2 Active Networking

Active Networking [93],[94],[79] o Redes Activas en español es otra iniciativa que surge en la década de los 90's con la propuesta de una red programable que posibilitara la configuración de servicios sobre la misma. Fue propulsada por la USA Defense Advanced Research Projects Agency (DARPA), con el principal objetivo de acelerar la innovación en una área de redes marcada por la lentitud en incorporar nuevas tecnologías y servicios.

Se propusieron dos enfoques: (1) enfoque de Switches Programables en donde el tráfico es encaminado a través de nodos que ejecutan programas y cada paquete o su segmento correspondiente es derivado al programa apropiado en el nodo y (2) enfoque de Cápsulas en donde cada paquete enviado en la red transporta un programa, evaluándose en cada nodo en su ambiente de ejecución correspondiente.

Como tal, este proyecto no alcanzó la suficiente masa crítica como para imponerse en la industria, entre otras razones por la necesidad de un hardware diferente más caro (tradicionalmente se utilizaba hardware de tipo ASIC y se requería de tipo TCAM, FPGA ó NPU) además de interrogantes de seguridad y desarrollo de lenguajes de programación apropiados.

2.1.3 Routing Control Platform

Routing Control Platform (RCP) [63],[55] o Plataforma de Control de Ruteo, propone una entidad de control (Routing Control Platform) encargada de computar las rutas en una red, que luego utilizando

protocolos estándares como iBGP traslada dicha configuración a los diferentes dispositivos. La principal ventaja de esta estrategia es que utiliza protocolos existentes estándares facilitando así la transición entre el paradigma actual y el propuesto. No obstante su potencial se encontraba limitado justamente a las capacidades de los protocolos de comunicación existentes.

2.1.4 ForCES Protocol

Otro grupo de trabajo dentro del IETF denominado ForCES (Forwarding and Control Element Separation) comenzó a trabajar a mediados del año 2007 en un protocolo que permitiera la separación entre los dispositivos de red y la inteligencia que los gobierna. Dicho protocolo fue especificado como una propuesta de estándar y publicado en el año 2010 bajo el nombre ForCES Protocol [60].

Este enfoque plantea dos entidades lógicas denominadas Elementos de Control (CE) y Elementos de Reenvío (FE), así como un protocolo de comunicación entre ambas entidades.

2.1.5 Proyecto 4D

En el año 2004 el proyecto 4D [86],[65] propone una re-estructuración arquitectónica de una red basada en tres principios:

- Políticas de red (network-level objectives): conjunto de objetivos o requerimientos sobre una red tales como performance, confiabilidad entre otras políticas.
- Vistas de la red (network-wide views): vistas globales sobre la topología de red, tráfico y eventos, con ciertos niveles de frescura y exactitud en los datos.
- Control directo: se debe proveer una interfaz directa para el control de los dispositivos de red

Basándose en estos principios se construye el esquema denominado “4D” que merece su nombre por los cuatro planos presentes en su arquitectura: decisión (decision), diseminación (dissemination), descubrimiento (discovery) y datos (data).

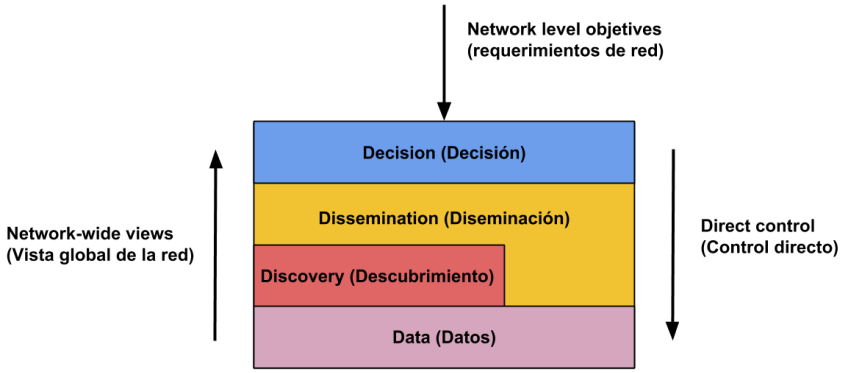

Fig. 2.1 Capas de la arquitectura. Imagen extraída de [65]

Dicha arquitectura hace un fuerte énfasis en una completa separación de lo que son las decisiones lógicas que toma un Sistema Autónomo (AS), del conjunto de protocolos que gobiernan las redes e interacciones entre sus diferentes componentes.

Los objetivos al nivel de un AS son especificados en el plano de decisiones y mediante los servicios de los planos de diseminación y descubrimiento se ejerce un control directo sobre cómo se encamina y redirigen paquetes en el plano de datos.

En resumen, utilizando el plano de diseminación se recolecta diferente información acerca del estado de la red, sobre la cual luego el plano de decisión construye las *Políticas de red*. También a través del plano de diseminación se distribuyen dichas políticas a los distintos dispositivos de red. Mientras tanto el plano de descubrimiento permite a dispositivos descubrir vecinos directamente conectados en la red. Finalmente el plano de datos se encarga de redirigir el tráfico en la red.

Una vez introducidos los principales antecedentes de las redes definidas por software, en la siguiente sección se propone profundizar en dicho concepto.

2.2 Software Defined Networking

Definir el concepto de Software Defined Networking no es una tarea simple, puesto que no existe una definición única entre las distintas organizaciones que se encuentran trabajando en la temática. En particular se pueden destacar el grupo de trabajo Software-Defined Networking Research Group (SDNRG) del IRTF, quien se encuentra trabajando en la redacción de las definiciones y estándares para este concepto y la organización Open Networking Foundation (ONF) [42] que concentra gran cantidad de material y documentación sobre SDN y además se encuentra trabajando en el desarrollo del protocolo OpenFlow el cual se explica más adelante en este trabajo.

Acorde a la Open Networking Foundation, SDN puede resumirse como:

> *"La separación física del plano de control de la red del plano de datos, y donde el plano de control controla varios dispositivos."*

Más en detalle, SDN es una arquitectura de red que desacopla los planos de control y de datos, moviendo el plano de control (inteligencia, construcción y cálculo de políticas de red) hacia una entidad denominada Controlador. Esto habilita al control de la red a ser directamente programable y a la infraestructura subyacente ser abstraída por las aplicaciones y servicios de red.

La ONF presenta a SDN como una arquitectura emergente, dinámica, escalable, rentable y adaptable, haciéndola ideal para aplicaciones de naturaleza dinámica y exigentes de los recursos de red disponibles. En particular destaca las cualidades de presentar una gestión centralizada y de estar basada en estándares abiertos neutrales a los vendedores.

El enfoque de SDN propone tres capas lógicas (ver imagen 2.2): (1) capa de aplicaciones (Application Layer), (2) capa de control (Control Layer) y (3) capa de infraestructura (Infrastructure Layer).

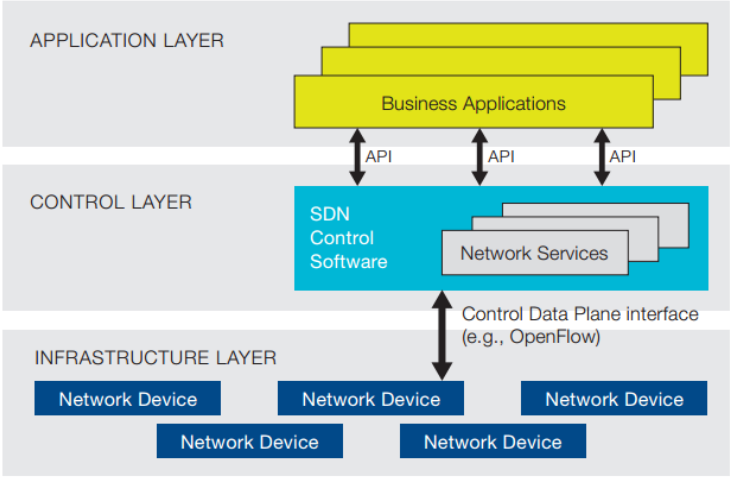

Fig. 2.2 Arquitectura de SDN - Capas lógicas. Imagen extraída de [42]

La capa de infraestructura se compone por los dispositivos de red tradicionales como switches y routers, en particular compatibles con la arquitectura SDN. La inteligencia de estos dispositivos a diferencia de los convencionales es retirada de ellos para ser trasladada a la capa de control. A su vez, cada dispositivo implementa un conjunto de operaciones bien definidas mediante la cual es manipulado por la capa de control. Esta API de operaciones forma parte de lo que se denomina Interfaz Sur.

En la capa de control se encuentra el software encargado de implementar el plano de control en dicho modelo, usualmente denominado Controlador. Esta entidad expone un conjunto de operaciones a las aplicaciones SDN de la capa de Aplicación para la manipulación de los dispositivos de la capa inferior (capa de Infraestructura). Esta API de operaciones es conocida en el modelo bajo el nombre de Interfaz Norte.

La capa de aplicaciones, contiene implementado en software (aplicaciones SDN) toda la inteligencia que originalmente se agregaba en un dispositivo de la capa de infraestructura como por ejemplo la implementación de protocolos de red. En dicha capa se pueden implementar desde protocolos de ruteo como OSPF y RIP, backbones sobre MPLS, políticas de seguridad y hasta aplicaciones de Ingeniería de tráfico. Nótese aquí la diferencia entre la capa de aplicación del modelo de capas OSI y la capa de aplicación del modelo SDN. En el modelo SDN, una aplicación puede implementar protocolos y servicios de capa de enlace, capa de red y capas superiores en el modelo OSI.

De esta forma Software Defined Networking centraliza el control de la red en una aplicación de software (Controlador) y transforma algoritmos, procesos de control, políticas de seguridad y toda la inteligencia antiguamente acoplados a los dispositivos de red, en aplicaciones de controlador.

2.3 Arquitecturas basadas en SDN

Existen varias arquitecturas o implementaciones en las que de una forma u otra puede encontrarse argumentos para afirmar que siguen el enfoque de SDN. Si bien este trabajo se basa en la utilización de la arquitectura OpenFlow, existen otras arquitecturas basadas en este enfoque como el protocolo ForCES y alternativas más comerciales como OpFlex [45].

A continuación se explican los principales conceptos relacionados a la arquitectura OpenFlow.

2.3.1 OpenFlow

Siguiendo el enfoque de tres capas de SDN y bajo la misma premisa de desacoplar completamente los planos de datos y control, OpenFlow [77] brinda una implementación estándar para el mecanismo de comunicación entre las capas de control e infraestructura (Interfaz Sur). En otras palabras, provee un protocolo de comunicación estándar para manipular los distintos dispositivos de la red y así el plano de datos.

OpenFlow se basa en tres componentes (ver figura 2.3): (1) Controlador OpenFlow compatible, (2) Protocolo OpenFlow y (3) Switch OpenFlow compatible.

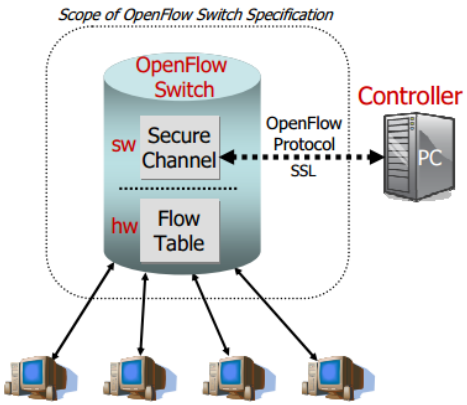

Fig. 2.3 Estructura de un switch OpenFlow. Imagen extraída de [77]

Desde la perspectiva de un Controlador OpenFlow, todos los dispositivos de red son equivalentes en sus funcionalidades y se denominan simplemente switches OpenFlow. Un switch OpenFlow se compone de tres componentes principales (ver figura 2.3): (1) Tabla de flujos, (2) Canal Seguro (Secure Channel) que conecta el switch con el Controlador habilitando el intercambio de paquetes y comandos entre estos últimos dos utilizando (3) el protocolo OpenFlow.

Cada switch OpenFlow presenta una o más tablas de flujos, compuestas por entradas denominadas flujo, las cuales determinan cómo una clase de paquetes deben ser procesados y reenviados. Cada entrada o flujo (ver figura 2.4) se compone de: (1) campos de selección (matching fields), utilizados en la definición de reglas de selección con las cuales identificar a un conjunto de paquetes a ser tratados de una forma en particular, basándose en información relacionada a campos en los cabezales

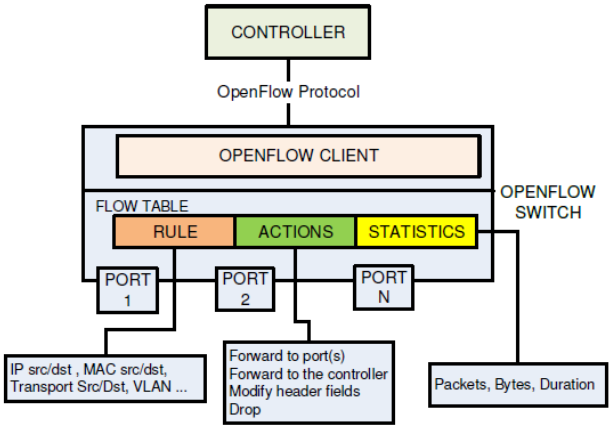

Fig. 2.4 Visión esquemática del funcionamiento de un Switch OpenFlow. Imagen extraída de [77]

del paquete para esto; (2) contadores para la recolección de información estadística en relación al flujo (número de paquetes recibidos, cantidad de bytes, duración de un flujo, etc.) y (3) un conjunto de instrucciones y acciones que son aplicadas a cada paquete que coincide con la regla del flujo. Estos campos establecen la forma en que un paquete es procesado y reenviado.

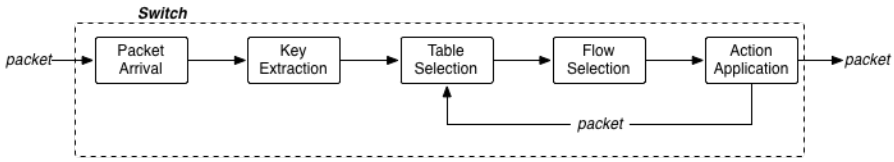

Fig. 2.5 Ciclo de vida de un paquete en el pipe OpenFlow

Cuando un paquete arriba a un switch (ver figura 2.5), se extraen y comparan cabezales del paquete acorde a las reglas definidas en el flujo, comparando los campos utilizados. En caso que el paquete coincida con las reglas definidas, se aplican el conjunto de instrucciones y acciones asociadas a dicho flujo. En caso que el paquete no coincida con ningún flujo, el accionar a desempeñar dependerá del conjunto de instrucciones y acciones definidas por una entrada especial en la tabla de flujos, denominada Table-miss Flow Entry.

La Table-miss Flow Entry está pensada para contemplar el tipo de tráfico que no es definido por ningún otro flujo, por ello en esencia es el último flujo a ser considerado (menor prioridad) al momento de procesar un paquete. Generalmente tiene asociada la acción de eliminar el paquete (Drop), procesar el paquete en la siguiente tabla de flujos (GoTo Table) ó reenviar el paquete al Controlador.

Otra característica importante de un switch OpenFlow, es la capacidad para procesar paquetes utilizando protocolos tradicionales de red además del procesamiento normal de OpenFlow. Esta característica a su vez da lugar a la clasificación en switches OpenFlow puros y switches OpenFlow

híbridos. Mientras que los switches OpenFlow puros son aquellos que solamente soportan el protocolo de igual nombre, los switches OpenFlow híbridos son aquellos que adicionalmente permiten procesar paquetes de la misma forma en que lo haría un hardware de red legado.

Aprovechando las capacidades de un switch OpenFlow híbrido, se puede definir por ejemplo la Table-miss-Flow-Entry de forma que se procese todo paquete contemplado por esta entrada como un hardware de red tradicional, aplicando por ejemplo esquemas de reenvío IP.

2.3.2 Reglas OpenFlow

Cada entrada en la tabla de flujos, distingue un tipo de tráfico en particular y la forma en que se procesa (acciones e instrucciones). Es uno de los pilares del protocolo la expresividad disponible para escribir las reglas de cada flujo; es decir, la capacidad de agrupar tipos de tráfico acorde a diferentes propiedades en el mismo.

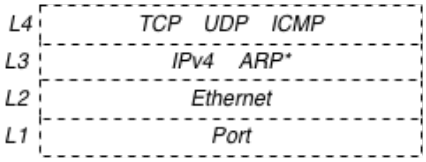

Fig. 2.6 OpenFlow v1.0 Matching Fields

Las reglas OpenFlow se basan en los valores de cabezales asociados a diferentes capas del modelo OSI. Estos cabezales a su vez varían con la versión del protocolo, incorporándose más opciones en cada versión del mismo. Por ejemplo en la versión 1.0 del protocolo se tiene un soporte mínimo para cabezales de capas 1 a 4 (ver imagen 2.6).

Por otro lado en la versión 1.3.3 del protocolo se cuenta con soporte para utilizar cabezales de IPv6, ICMPv6 y MPLS en la definición de los flujos (ver imagen 2.7).

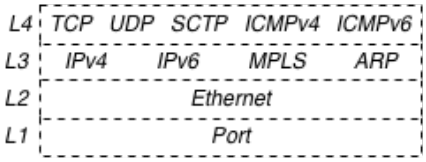

Fig. 2.7 OF 1.3.3 Matching Fields

2.3.3 Acciones OpenFlow

Como se menciona anteriormente la acción de un flujo OpenFlow determina la forma en que un paquete es procesado en un switch. Las tres acciones principales que todo switch OpenFlow debe implementar son:

1. **[Output:Puerto]:** Reenviar un paquete hacia un puerto determinado (o conjunto de puertos). Esto permite encaminar paquetes a través de la red.

2. **[Output:Controller]:** Encapsular y reenviar un paquete hacia el Controlador. Utilizando el canal de comunicación con el Controlador y el protocolo OpenFlow, se reenvía el paquete al plano de Control. Esto permite procesar un paquete no identificado en ningún flujo en cualquier momento como por ejemplo en una etapa de configuración de la red. Luego el Controlador decide si descartar el paquete o instalar un nuevo flujo que lo contemple.

3. **[Drop]:** Descartar un paquete. Puede ser utilizado por razones de seguridad, para prevenir ataques de denegación de servicios (DoS), eliminar paquetes sospechosos, disminuir el impacto de paquetes de descubrimiento en redes de difusión (broadcast), o simplemente descartar paquetes no contemplados.

2.3.4 Controlador OpenFlow

Un controlador SDN brinda implementaciones a las interfaces Sur y Norte. Un controlador OpenFlow/SDN requiere además de esto la compatibilidad con el protocolo OpenFlow en la interfaz Sur.

Basándose solamente en estos principios, las implementaciones de controladores OpenFlow pueden ser muy variadas. Desde controladores muy simples como NOX[8] y POX[11] que siguen estrictamente el enfoque de SDN/OpenFlow, hasta controladores más elaborados como OpenDaylight [9] (ver imagen 2.8) que implementan OpenFlow como una característica más de su implementación de interfaz Sur.

Sin embargo existen aspectos relacionados a la entidad Controlador dentro del enfoque OpenFlow/SDN que son afines a cualquier implementación y arquitectura. A continuación se mencionan los principales:

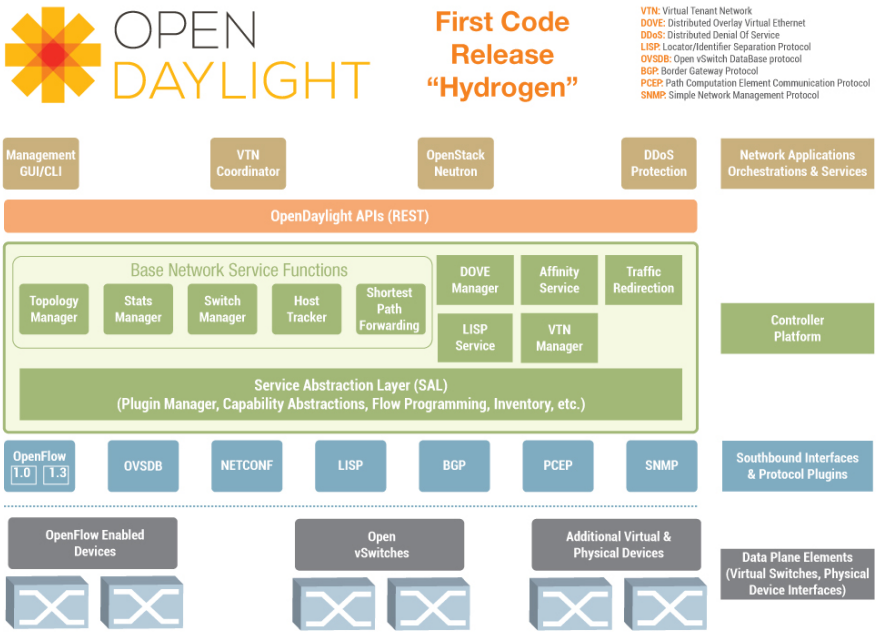

Fig. 2.8 Arquitectura de OpenDaylight Hydrogen. Imagen extraída de [44]

Granularidad en el control

Tradicionalmente, la unidad básica de tráfico en una red es el paquete. Cada paquete contiene información de direccionamiento necesaria para la toma de decisiones dentro del switch a nivel de reenvío. Sin embargo muchas aplicaciones de red envían datos como un flujo de muchos paquetes individuales, mientras que las redes que desean proveer de calidad de servicios (QoS) se benefician utilizando flujos. Por ello resulta beneficioso adoptar un nivel de abstracción superior en el control del plano de datos, distinguiendo tráfico por flujos y no a través de paquetes individuales.

Por otro lado uno de los puntos vulnerables del enfoque SDN, es la comunicación entre switch y Controlador. Sobrecargar este canal de comunicación, pensando en redes a gran escala en la cantidad de dispositivos, supone una debilidad importante. Basar la granularidad de control en paquetes individuales, profundiza esta debilidad; mientras que el enfoque basado en flujos atenúa esta debilidad.

Para fijar ideas, en el enfoque de flujos, cuando un paquete recibido no es contemplado por ningún flujo particular puede ser reenviado al Controlador para decidir su procesamiento. Este, tras analizar dicho paquete puede resolver además del procesamiento instalar un nuevo flujo para contemplar más adelante paquetes similares (incluyendo al mismo paquete). De esta forma cualquier otro paquete contemplado por el nuevo flujo no es más reenviado al Controlador, mientras que en el enfoque de paquetes individuales la regla instalada contemplaría solamente al paquete inicial.

A su vez se pueden utilizar las mismas prácticas que con el control a nivel de paquetes individuales, como agrupar flujos relacionados al tráfico entre dos hosts, tomando decisiones sobre los flujos agregados.

Centralizado vs Distribuido

SDN no establece ninguna restricción en cuanto a si el plano de control debe ser tanto lógica como físicamente centralizado o distribuido. De hecho resulta conveniente pensar en enfoques distribuidos cuando se piensa en arquitecturas robustas y escalables. No es el objetivo de este trabajo profundizar en esta área de investigación por lo cual se recomienda para profundizar sobre esta temática las referencias [68] y [74].

En relación a OpenFlow, si bien el protocolo no establece mecanismos para la comunicación entre controladores, habilita a un switch a conectarse a múltiples controladores. Algunos proyectos como Onix [70] e HyperFlow [95] toman la idea de mantener un plano de control lógicamente centralizado pero físicamente distribuido. Este enfoque tiene como principal beneficio disminuir la sobrecarga en el proceso de búsqueda sobre una tabla de flujos, permitiendo la comunicación con controladores locales. Por otro lado, Kandoo [67] proponen utilizar controladores locales para el manejo de aplicaciones y redirigir hacia un Controlador global flujos que requieran de una visión global del estado de la red, bastante similar a lo anterior. Esto tiene como principal beneficio la reducción de la carga sobre el Controlador global, debido al filtrado de solicitudes de flujo que realizan los controladores locales a la vez proveen al plano de datos de una rápida respuesta para las solicitudes que pueden ser manejadas por los controladores locales. Finalmente otro de los enfoques que buscan generar una distribución del plano de control en SDN es el asumido por FlowVisor [92], el cual propone la virtualización de una red de switches OpenFlow con múltiples controladores, construyendo lo que denominan Slices de la red. De esta forma brindan a cada controlador una visión local a su Slice, mientras que un controlador especial actúa de Proxy manteniendo una visión global de toda la red.

Políticas reactivas vs proactivas

Existen dos enfoques para la intervención del controlador en el plano de datos: (1) reactivo y (2) proactivo.

1. **Instalación reactiva de políticas:** En este enfoque, un switch se configura con la tabla de flujos vacía o al menos una configuración mínima al encenderse. Luego, cada vez que un paquete arriba al switch y no es contemplado por ningún flujo, es reenviado al Controlador. Este decide que hacer con el paquete e instala un flujo en el dispositivo para procesar de igual forma paquetes similares.

   Este modelo adoptado en particular por Ethane [57], presenta como principal desventaja la sobrecarga en el canal de comunicación Switch-Controlador. Por ello juega un rol sumamente

importante en este esquema la definición de los flujos (flujos más generales disminuyen la sobrecarga con el controlador y flujos muy específicos aumentan esta sobrecarga), tiempo de vida de los flujos en la tabla (flujos con tiempo de vida de un solo paquete aumentan la sobrecarga), la ubicación física del controlador y las características del canal de comunicación. Utilizando una jerarquía de controladores, canales de comunicación de alta velocidad o redundancia de enlaces, se puede mejorar la penalización en los tiempos de respuesta por la sobrecarga del canal de comunicación.

2. **Instalación proactiva de políticas:** En este enfoque, un switch se configura con la mayor cantidad de flujos posibles, contemplando la mayor cantidad de tráfico en la red al momento de encender el switch. Luego, cuando un paquete ingresa al switch, en caso de no ser contemplado por ninguno de los flujos instalados, se reenvía al Controlador para decidir cómo actuar. Luego el Controlador puede instalar un flujo para contemplar el caso asociado a este paquete.

   En [97] se analiza el impacto de utilizar un enfoque de políticas proactivas en la performance de una red.

2.4 Herramientas SDN/OpenFlow disponibles

La principal herramienta para el desarrollo de aplicaciones SDN/OpenFlow es el software de control o Controlador. No obstante también existen entornos de emulación compatibles con estas tecnologías y que merecen la pena ser mencionados en este trabajo por su utilidad.

2.4.1 Controlador

Existe un cantidad interesante de propuestas para la implementación del plano de control. En este trabajo se ha mencionado algunos como OpenDaylight, NOX y POX, pero existen varias alternativas. Pueden encontrarse desde controladores comerciales y académicos, en código abierto y propietario, compatibles solamente con OpenFlow y compatibles con múltiples protocolos incluyendo OpenFlow, etc.

Estas características son utilizadas en este proyecto para la elección de la mejor alternativa de software de control, basándose en las necesidades y restricciones del prototipo. Por ello se cree conveniente incluir el siguiente cuadro comparativo (ver tabla 2.1), en el que se presentan las principales características utilizadas en la elección del controlador más apropiado. Cabe destacar que esta tabla está basada en una tabla más reducida presentada en [84].

Controlador	Implementación	Opensource	Desarrollador	OpenFlow	Detalle
POX[11]	Python	Si	Nicira	1.0	Controlador de propósito general implementado en Python, compatible con Linux, Mac OS, y Windows.
NOX[8]	C++	Si	Nicira	1.0	Uno de los primeros controladores ampliamente utilizado. Implementa funcionalidades para descubrimiento de topologías, learning switch y Network-wide switch.
OpenMUL[10]	C	Si	Kulcloud	1.4	Controlador basado en OpenFlow con compatibilidad hacia atrás (OpenFow v1.0, v1.2, etc) con soporte multi-thread basado en lenguaje C, soporte SSL, soporte a múltiples interfaces norte, API REST
Maestro[6]	Java	Si	Rice University	-	Maestro es un controlador orientado a la orquestación de aplicaciones de control de red. Provee interfaces para el acceso y modificación de los dispositivos de la red a estas aplicaciones, así como para coordinar la interacción entre ellas.
Trema[15]	Ruby/C	Si	NEC	1.0	Framework full-stack para el desarrollo de Controladores OpenFlow en Ruby y C, compatible con Ubuntu 13.04 y Fedora 16-19 entre otras distribuciones Linux.
Beacon[1]	Java	Si	Stanford	1.0	Controlador multi-plataforma, rápido y modular basado en OpenFlow con soporte a programación orientada a eventos y concurrente (threads). Incorpora la plataforma web Jetty y un módulo GUI extensible.
Jaxon[5]	Java	Si	Desarrolladores independientes	1.0	Controlador OpenFlow basado en NOX para desarrollo en Java.
Helios[4]	C	No	NEC	-	Controlador basado en OpenFlow orientado a lenguaje C, extensible. Interfaz de comunicación Norte en formato Shell.
Floodlight[2]	Java	Si	BigSwitch	1.3	Controlador SDN basado en OpenFlow, soporte para switches virtuales y físicos, maneja redes OpenFlow y redes no OpenFlow así como islas OpenFlow, soporte a OpenStack.
SNAC[14]	C++	No	Nicira	-	Controlador basado en OpenFlow para redes LAN con GUI y un lenguaje definido para declarar reglas. Está basado en el controlador NOX más un módulo para un lenguaje de modelado formal (FML).
RYU[13]	Python	Si	NTT, OSRG group	1.4	Framework de programación para SDN, provee controladores basados en OpenFlow, Netconf, Ofconfig entre otros. Brinda soporte para OpenFlow v1.0 v1.2 v1.3 y v1.4 y extensiones propuestas por le empresa Nicira.
Nodeflow[7]	Javascript	Si	NTT, OSRG group		Controlador basado en OpenFlow orientado a programación en Javascript, basado en el framework Node.js.
OVS-Controller[46]	C	Si	Desarrolladores independientes	-	Controlador basado en OpenFlow de referencia con soporte a Open vSwitch y gran parte de otros tipos de switches.
FlowVisor[3]	C	Si	Stanford/Nicira	-	Controlador de propósitos particulares, que actúa como proxy de forma transparente entre una red OpenFlow y múltiples Controladores OpenFlow.
RouteFlow[12]	C++	Si	CPqD	1.3	Proyecto Open Source que provee servicios de routing virtualizados sobre hardware OpenFlow. Un escenario de uso común puede ser su utilización en conjunto con otro Controlador como Ryu.
OpenDaylight[9]	Java	Si	Linux Foundation	1.3	Open Daylight es un framework abierto a la comunidad y apoyado por fabricantes, orientado a mejorar y agilizar la adopción de nuevos protocolos y aplicaciones así como la innovación. Este controlador esta desarrollado en java y provee diferentes módulos de conexión para la interfaz Sur, entre los cuales se encuentra el protocolo OpenFlow. Brinda soporte al protocolo OpenFlow hasta versión 1.3 actualmente.

Table 2.1 Principales controladores disponibles (Basada en una tabla similar y más reducida en [84])

2.4.2 Mininet y miniNExT

Mininet [29] es un emulador de red, que habilita a crear hosts, switches, links y controladores, en un entorno virtual. Se encuentra disponible tanto para instalarse nativamente en un entorno Linux, como a través de una máquina virtual configurada con todo el software de desarrollo necesario para iniciarse en el desarrollo de aplicaciones OpenFlow/SDN (Mininet Kit Starter).

Soporta diferentes versiones del protocolo OpenFlow y es una de las herramientas más utilizadas para la prototipación y experimentación de aplicaciones desarrolladas en dicha arquitectura, puesto que habilita a investigadores y desarrolladores aprender, prototipar, probar y depurar aplicaciones rápidamente utilizando una computadora convencional.

Por otro lado, la herramienta miniNExT [30] es una extensión de Mininet que permite entre otras cosas asignar a cada host su propio sistema de ficheros, permitiendo así modificar la configuración por defecto de cada nodo virtual e instalar diferentes herramientas en ellos. Está orientado al despliegue virtual de arquitecturas más complejas que las soportadas por Mininet.

En este trabajo Mininet es utilizado para realizar pruebas de experimentación y comparación entre las alternativas de controladores disponibles, con el objetivo de determinar el controlador OpenFlow a ser utilizado en la arquitectura del prototipo.

2.5 Aplicaciones de SDN

SDN presenta un campo de aplicación bastante extenso y es posible que a medida que el concepto siga evolucionando y asentándose se irán descubriendo nuevos casos de uso y campos de aplicación. A continuación se muestran algunos casos de uso:

- **Monitorización de Red:** En una arquitectura OpenFlow por ejemplo, cada switch provee información estadística (similar a SNMP) de cada flujo en tiempo real. De esta forma un Controlador puede recolectar información estadística sobre el tráfico en el plano de datos, lo cual es de gran interés para operadores de red y aplicaciones, con la granularidad exacta que estos requieran. Esta información puede ser agregada de diferentes formas: por dirección IP, por dirección MAC, etiquetas VLAN, por aplicación, etc. De esta forma se logra una gran versatilidad en la entrega de información estadística sobre el tráfico en una red.

- **Redes TAP programables:** Utilizando OpenFlow se pueden implementar taps de redes programables en forma centralizada de una forma simple, transparente y eficiente. También pueden filtrarse los datos que son enviados por ejemplo a un sistema de detección de intrusos (IDS), reduciendo la carga de trabajo a los dispositivos de procesamiento de datos utilizados, filtrando previamente los datos que son realmente necesarios y evitando replicar datos innecesarios. Además desde que el control de los dispositivos intermediarios puede realizarse desde un controlador, se facilita enormemente la gestión de los mismos, permitiendo una configuración dinámica en caso de fallas o averías técnicas.

- **Balanceo de carga y QoS:** Contar con una visión global de los dispositivos del plano de datos, habilita al Controlador a implementar diferentes políticas de balanceo de carga en función del estado de la red y permite un uso óptimo de todos los recursos disponibles. A su vez se pueden priorizar diferentes tipos de tráfico, encaminar tráfico dependiendo de un tipo de servicio o contenido y otras técnicas de QoS.

- **Herramientas para mitigación de ataques DoS:** Herramientas para la mitigación de ataques de denegación de servicios (DoS) pueden beneficiarse de las estadísticas provistas por un switch OpenFlow por ejemplo para detectar anomalías. Además utilizando la acción de redirección de un flujo al Controlador se puede analizar y detectar tráfico sospechoso de ataques DoS, permitiendo a posteriori insertar un flujo en el switch de ingreso específico para bloquear el tipo de ataque o tráfico malicioso detectado.

  De todos modos vale la pena recordar la inconveniencia de sobrecargar el canal de comunicación entre un dispositivo y el controlador.

- **Implementación de Servicios de red:** Con una arquitectura basada en SDN como OpenFlow, resulta sumamente sencillo la incorporación de servicios al plano de datos de una infraestructura de red dada. Independientemente de la marca del hardware utilizado, sus características y funcionalidades, siempre que sean compatibles con OpenFlow puede incorporarse nuevos servicios como autenticación, firewall, almacenamiento secundario, etc., simplemente programando una nueva aplicación en el Controlador que implemente dichos servicios.

- **Experimentación:** El desarrollo, verificación y puesta en producción de nuevos servicios y protocolos resulta sumamente sencillo puesto que consta simplemente de desplegar una nueva aplicación en un entorno de software (Controlador). Por ello resulta sumamente útil el enfoque de SDN para la academia y la industria en lo que refiere al desarrollo de nuevos protocolos de red, nuevos servicios, mejoras a servicios existentes, entre otros.

Por estas razones soluciones basadas en el enfoque SDN con OpenFlow por ejemplo u otras arquitecturas basadas en SDN resultan de gran utilidad en escenarios como:

1. Redes empresariales
2. Centros de cómputo (Data Centers)
3. Academia

2.6 Casos de éxito de SDN/OpenFlow

OpenFlow en particular se ha hecho con un cierto protagonismo recientemente. Tal es el hecho que se pueden nombrar casos de éxito en la migración de servicios sobre esquemas tradicionales de redes a

SDN/OpenFlow. Por ejemplo, la Open Networking Foundation elaboró un artículo [78] en donde se recomiendan métodos y se proveen guías para la migración de servicios desde un esquema tradicional a SDN nombrando en particular tres casos de éxito en este tipo de migraciones, en escenarios reales: (1) Google InterDatacenter WAN, (2) NTT Provider Edge y (3) Stanford Campus Network.

Por mayores detalles en relación a estos casos de éxito se recomienda continuar con la lectura en dicho artículo.

2.7 Red Privada Virtual

En pocas palabras una Red Privada Virtual o VPN por su sigla en inglés, es la extensión de una red privada sobre una infraestructura de red pública como lo es por ejemplo Internet.

Este concepto habilita a un equipo en una determinada subred privada a enviar y recibir información a otro equipo en otra subred separadas físicamente, utilizando una infraestructura de red pública o compartida para la comunicación entre ambas subredes, y de la misma forma que si estuviese directamente conectada a la red privada. Además permite mantener las políticas de seguridad y funcionalidades de la red privada.

Las implementaciones de redes privadas se caracterizan entre otras cosas por las funcionalidades que son capaces de proveer. En relación a aspectos de seguridad se buscan principalmente funcionalidades de autenticación de usuarios, confidencialidad e integridad de los datos. Por otro lado en relación al procesamiento del tráfico dentro de la red pública, algunas implementaciones de VPN ofrecen funcionalidades de clasificación de tráfico y calidad de servicios (QoS), permitiendo por ejemplo garantizar un ancho de banda mínimo para el tráfico de una organización cuando es transportado a través de la red pública de una red privada a otra.

Algunas implementaciones proveen además la capacidad de priorizar tipos de tráfico como por ejemplo el asociado a una aplicación en particular, o distribuír la carga entre diferentes caminos, lo que se conoce como balanceo de carga.

En relación a la implementación de redes privadas, a lo largo de los años han surgido diferentes alternativas diferenciándose por las funcionalidades que implementan y también por las capas en el stack OSI en la cual operan. Algunos ejemplos son:

- VPN tradicionales
    - Frame Relay (Capa 2)
    - ATM (Capa 2)
- VPNs basadas en CPE
    - L2TP (Capa 2)
    - IPSec (Capa 2)
- VPNs implementadas por proveedores de servicios

- VPN de capa 2 con MPLS
- BGP/MPLS VPNs de capa 3

Otra de las características por la cual es interesante diferenciar a dichas implementaciones, es la capacidad de conectar dos o más puntos de una red privada, lo cual da lugar a redes privadas punto a punto o multipunto.

Redes Privadas Punto a Punto

Las redes privadas punto a punto (ver figura 2.9) son un tipo de implementación orientada a conectar solamente dos extremos de la red de una organización, utilizando lo que se denominan túneles sobre la infraestructura de un proveedor de servicios o Internet. Típicamente se utilizan para conectar un cliente con un servidor en una organización cuando se encuentran físicamente separados, o dos edificios de una misma organización de una forma simple.

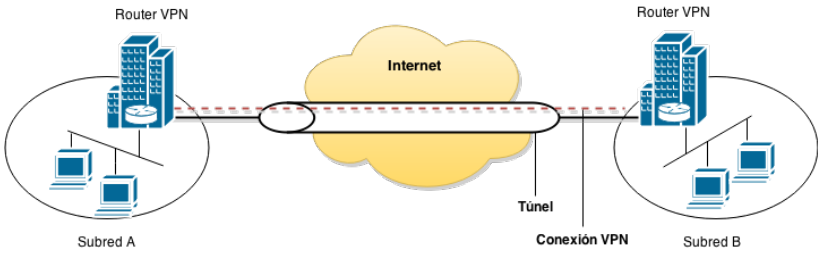

Fig. 2.9 Esquema de una VPN Punto a Punto

En este enfoque el tráfico de la subred A al llegar al router VPN, es encapsulado y enviado a través de un túnel sobre la red pública. Al llegar al otro extremo del túnel el tráfico es desencapsulado en el router VPN y entregado a la subred B, tomando luego el camino correspondiente dentro de la misma en forma normal. Esto genera la percepción desde ambas subredes que se tiene un "cable" entre ambos routers VPN, conectando ambas subredes como si fuesen una sola red privada.

Redes Privadas Multipunto

Las redes privadas multipunto, extienden el enfoque punto a punto con el objetivo de brindar conectividad a una organización dispersa geográficamente en múltiples lugares. Permiten establecer dominios de difusión compartidos, creando de forma transparente la ilusión de que se tiene una gran LAN compuesta por varias subredes. Sin embargo al ser más complejas en funcionalidades y casos de uso soportados, también son más complejas de implementar.

Virtual Private LAN Service (VPLS), es una implementación de red privada multipunto, bastante difundida y provista de los dos enfoques definidos en los RFC4761 [69] y RFC4762 [71]. Esta implementación crea dominios de difusión Ethernet utilizando diferentes tecnologías como IP/MPLS, L2TPv3 y túneles GRE.

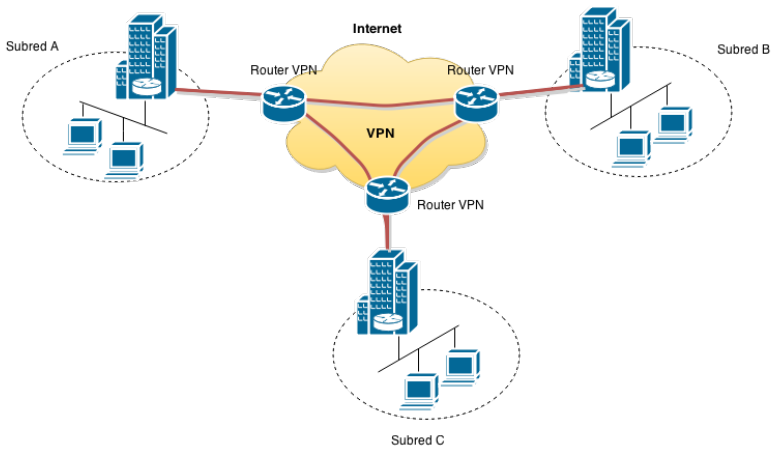

Fig. 2.10 Esquema de una VPN Multipunto L3

En una red privada multipunto de capa 3 como la de la figura 2.10, compuesta por las subredes A, B y C, cuando se origina un paquete en un equipo de la subred A con destino a otro equipo en la subred B sucede lo siguiente: (1) primero el paquete es encaminado al nodo de borde de la subred A (Router VPN), (2) luego se resuelve dinámicamente a qué punto de la VPN debe ser enviado el paquete (subred B o subred C), para lo cual se utiliza un protocolo de ruteo de borde como BGP, después (3) el paquete es encapsulado dentro de MPLS y encaminado mediante conmutación de etiquetas hasta el nodo de borde en la subred B, finalmente (4) el paquete es desencapsulado y entregado a la subred B para ser encaminado en forma normal hacia el equipo destino.

2.7.1 Redes Privadas en Uruguay y proyecciones para la RAU

En Uruguay ANTEL (el principal proveedor de servicios de telecomunicaciones del país) ofrece servicios de redes privadas basados en tres tecnologías principalmente[1]:

- Lan to Lan con conexiones Ethernet punto a punto
- VLAN Hub con conexiones Ethernet
- VPN IP/MPLS multipunto con QoS opcionalmente

Dentro de las proyecciones existentes para la RAU2, se encuentran funcionalidades como calidad de servicios, clasificación de tráfico y priorización del mismo de acuerdo a políticas definidas. A su vez se busca conectar múltiples oficinas y organizaciones dispersas geográficamente en todo el país, manteniendo las políticas de seguridad de la red privada de cada organización.

De acuerdo a las implementaciones mencionadas anteriormente, la solución que más se adapta a los requerimientos de la RAU2 es una VPN IP/MPLS multipunto con funcionalidades de QoS, en donde el rol de proveedor de servicios lo asume la propia red académica o la organización encargada

[1] Información extraída en base a los servicios comerciales ofrecidos por ANTEL en su sitio oficial [16].

de su mantenimiento (SeCIU). Para complementar la lectura en relación a este tipo de implementación recomendamos continuar con RFC2547 [90] y RFC4364 [61].

2.8 Multiprotocol Label Switching

Multiprotocol Label Switching (MPLS) o conmutación de etiquetas con soporte para múltiples protocolos, es un mecanismo de transporte de datos desarrollado y estandarizado por el IETF, en particular por los RFC 3031 [89], 3032 [88] y 3814 [81]. De acuerdo al modelo de capas OSI, opera entre las capas de Red y Enlace y fue diseñado para unificar el servicio de transporte de datos basado en circuitos virtuales y en conmutación de paquetes.

MPLS sustituyó a los servicios de Frame Relay y ATM en la construcción de servicios de redes privadas, siendo hoy en día el estándard de facto para esto y contando con una amplia adopción de fabricantes. Entre sus cualidades más importantes se distinguen soporte para VPNs, Ingeniería de Tráfico, calidad de servicios (QoS), clases de servicios (CoS), a la vez que es independiente del protocolo de capa de enlace pudiendo operar por ejemplo sobre ATM ó Ethernet.

En una red MPLS, cuando un paquete ingresa se le coloca un cabezal compuesto por cuatro campos: Label, Exp, S, TTL. De ellos se destacan el valor de la etiqueta MPLS, utilizada para encaminar paquetes a través de la red hacia su nodo destino y eventualmente realizar clasificación de tráfico. Este cabezal es extraído y analizado en cada nodo de la red para determinar el próximo nodo al que se debe enviar dicho paquete, colocando un nuevo cabezal con un nuevo valor de etiqueta antes de reenviarse. De esta forma el paquete es encaminado dentro de la red MPLS mediante la conmutación de etiquetas, hasta que el paquete arriba a un nodo de salida de la red, en donde ya no contiene etiquetas.

Por otro lado, un paquete MPLS puede presentar más de un cabezal MPLS solapado. Esta superposición de etiquetas se conoce bajo el nombre de pila o stack MPLS. De esta forma se pueden agregar más variables al servicio que simplemente encaminar paquetes en la red, como puede ser distinguir entre diferentes tipos de tráfico.

Otro punto a destacar de este protocolo es la configuración del plano de reenvío. El plano de control de las redes MPLS se compone de un algoritmo de ruteo y un algoritmo de distribución de etiquetas. Mientras que con el primero se calculan los mejores caminos que un paquete debe tomar para atravesar la red, el segundo se encarga de la asignación y distribución de etiquetas para que cada switch conmute correctamente el paquete hasta su destino.

Algunos de los principales términos en la arquitectura MPLS son:

- LER (Label Edge Router): Elemento de la red MPLS por la cual ingresa o egresa tráfico hacia o desde la misma.

- LSR (Label Switching Router): Elemento de la red MPLS que conmuta etiquetas.

- FEC (Forwarding Equivalence Class): Nombre asignado al tráfico que es encaminado bajo una misma etiqueta. En otras palabras, es una clase de equivalencia de tráfico a la cual se le asigna una etiqueta.

- LSP (Label Switched Path): Nombre genérico que se le asigna a un camino MPLS para tráfico de una determinada FEC, es decir un túnel MPLS establecido entre dos extremos de la red.

- LDP (Label Distribution Protocol): Protocolo de distribución de etiquetas MPLS entre los equipos de la red.

En la siguiente sección se presenta un resumen del estudio en el estado del arte realizado sobre la plataforma NetFPGA.

2.9 NetFPGA

NetFPGA [83] es una plataforma de hardware reconfigurable y software de código abierto (Open Source), flexible y potente, diseñada para un uso académico en tareas de investigación y enseñanza.

El hardware está basado en un chip FPGA (Field Programmable Gate Array), el cual se programa con lenguajes de descripción de hardware (HDL) como por ejemplo Verilog. Cuenta con una amplia variedad de proyectos desarrollados tanto por el equipo de NetFPGA (proyectos de referencia), como por instituciones académicas y comerciales involucradas (proyectos comunitarios) que permiten programar el comportamiento del hardware de diferentes formas. Se pueden encontrar desde proyectos que programan el hardware con una implementación de referencia de una tarjeta de red (Reference NIC), como un Learning Switch, Simple Router, Router OpenFlow, hasta un generador de tráfico entre otros (por mayor información ver [34]).

Surge en el año 2007 como un proyecto de investigación en la Universidad de Stanford bajo el nombre de NetFPGA-1G, con la consigna de construir una plataforma de desarrollo e investigación basada en un chip FPGA. La versión comercial alcanzada consistió en una placa PCI con un chip Xilinx Virtex-II proFPGA y cuatro interfaces Ethernet de 1Gigabit, más un repositorio de código fuente descargable y abierto a la comunidad, conteniendo librerías IP y unos pocos ejemplos de diseño.

El proyecto prosperó vendiendo más de 2.600 tarjetas en 150 instituciones educativas en 15 países diferentes del mundo hasta el momento en que se discontinuó dicho producto.

Tras el éxito de la NetFPGA-1G, en el año 2009 comenzó a trabajarse en la NetFPGA-10G una versión más potente que la anterior basada en un chip Xilinx Virtex5 y cuatro interfaces 10-Gigabit SFP+, que remplazaría a la NetFPGA-1G. Tras iniciarse su comercialización en el año 2011 la misma cosechó un éxito similar a su antecesora llegando a vender más de 470 unidades al año 2014, y contando con una plataforma de al menos 15 proyectos desarrollados específicamente para esta plataforma.

A la fecha, NetFPGA cuenta con cuatro versiones de plataformas comerciales, la NetFPGA-1G (discontinuada), NetFPGA-10G, NetFPGA CML y NetFPGA-SUME. A su vez la plataforma se ha popularizado entre investigadores y desarrolladores como plataforma ideal para la experimentación e innovación, contando a la fecha con más de 226 artículos académicos [33] en diversas áreas, utilizando como plataforma de trabajo NetFPGA.

En resumen NetFPGA es una plataforma robusta que propicia a investigadores, docentes y estudiantes en el área de redes de computadoras a la construcción de prototipos sobre sistemas de redes de alta velocidad y acelerados por hardware, en poco tiempo y con un costo inferior a otras alternativas de prototipación.

En este trabajo, se utiliza la plataforma NetFPGA-10G para la prototipación de un dispositivo compatible con OpenFlow a partir del cual a su vez implementar una red prototipo basada en el enfoque OpenFlow/SDN.

Capítulo 3

Análisis del problema

El primer paso en el proceso de construcción del prototipo para la RAU2, es el análisis del problema planteado. Definir en función de requerimientos el alcance del prototipo, investigar qué alternativas se tienen para construir un nodo del prototipo a partir del hardware NetFPGA, resolver qué estrategia será la utilizada para implementar servicios de VPNs, entre otros aspectos.

El presente capítulo está destinado al planteo de estos aspectos y los fundamentos sobre los que se basan las decisiones de diseño.

3.1 Definición de requerimientos

Para identificar posibles requerimientos a implementar en el prototipo para la RAU2, se trabaja inicialmente en identificar requerimientos generales sobre esta red. Luego los resultados obtenidos serán contextualizados en el alcance, los objetivos y resultados esperados de este trabajo, alcanzando así los requerimientos para el prototipo.

La RAU definió las funcionalidades esperadas para su evolución hacia RAU2, considerando a ANTEL como el proveedor de la infraestructura de red necesaria para el despliegue. Las principales características que se desean implementar son las siguientes:

1. **Clasificación y separación del tráfico:** Una de las necesidades y objetivos de la futura actualización de la RAU, es la facilidad para clasificar y aislar tráfico. Alineado con las actuales necesidades, en particular se precisa al menos diferenciar las siguientes 3 categorías, así como aislar el tráfico de cada categoría y para cada organización: (a) público, (b) académico y (c) servicios de contenido.

2. **Manejo de grandes volúmenes de datos:** En la RAU intervienen instituciones como el Instituto Pasteur, Centro Uruguayo de Imagenología Molecular (CUDIM) entre otros, en donde la generación e intercambio de grandes volúmenes de datos como lo son los exámenes PET del Pasteur o una secuenciación de ADN del CUDIM entre otros es una de los servicios que la RAU2 deberá soportar.

3. **Escalabilidad:** Se espera alcanzar en un mediano plazo un total de 11.000 docentes, 7.000 funcionarios y 140.000 estudiantes, por lo que el prototipo para la RAU2 debe ser escalable en la cantidad de usuarios soportados por la infraestructura.

4. **Red de entrega de contenidos (Content Delivery Network):** Resulta sumamente útil tomar un enfoque de red de entrega de contenidos para el diseño de la red académica. Las organizaciones partícipes de la misma tienen y generan grandes volúmenes de información de gran interés por parte de otras organizaciones de la RAU. Una red de distribución de contenidos garantiza un mejor acceso en tiempo real a dicha información por parte de múltiples organizaciones en simultáneo.

Este conjunto de funcionalidades o requerimientos representa un problema de dimensiones mayores al que se pretende resolver en este trabajo, por ello pensando en resolver un problema acorde a las dimensiones de un proyecto de fin de carrera, se decide hacer foco en el primer requerimiento, la Clasificación de Tráfico.

Partiendo de la clasificación de tráfico como funcionalidad principal, se llega a los siguientes requerimientos:

Requerimientos Funcionales
• Dada una organización o aplicación, el Sistema debe tener la capacidad de clasificar y distinguir el tráfico asociado a la misma de cualquier otro tipo de tráfico. Esto se conoce como clasificación de tráfico. • Dadas dos subredes asociadas a una misma organización e interconectadas mediante el Sistema, se debe garantizar que la numeración IP del tráfico generado por una de las subredes sea mantenida al ser entregado a la segunda subred, manteniendo de esta forma la identidad de los usuarios en ambas subredes.

Requerimientos no Funcionales
• Open Source: En la medida que sea posible interesa utilizar herramientas y componentes libres y abiertas como software libre y de código abierto, hardware libre.

3.2 ¿Por qué utilizar SDN?

Se tiene ya una noción básica del sistema que se pretende construir, interesa entonces analizar por qué se quiere utilizar el enfoque de SDN en el mismo.

El mecanismo estándar y transparente que propone SDN para manipular los diferentes dispositivos de red (se han hecho grandes esfuerzos para estandarizar algunas de las implementaciones de la Interfaz Sur existentes, OpenFlow es un buen ejemplo de ellos) brinda flexibilidad y agilidad para el desarrollo de nuevos protocolos y servicios de redes en una infraestructura de dispositivos heterogénea, facilitando así la innovación en el área. A su vez, se gana independencia de la implementación propuesta por cada fabricante. Actualmente utilizar productos de un determinado fabricante para la construcción de servicios, muchas veces implica configurar equipos y desarrollar aplicaciones sobre la plataforma del mismo, la cual usualmente tiene su propia API de operaciones generalmente propietaria y su propio lenguaje de desarrollo. Esto implica dificultades para la incorporación de nuevos dispositivos a una red existente, eventualmente de diferente fabricante, dificultades para la incorporación de nuevos servicios y más dificultades para migrar servicios existentes, eventualmente a una plataforma diferente.

En pocas palabras, SDN facilita notablemente la investigación e innovación en el desarrollo de nuevos servicios y protocolos, eliminando barreras en la manipulación de los diferentes dispositivos de red, y ganando libertad de los fabricantes de hardware.

3.3 ¿Por qué utilizar NetFPGA?

Se necesita para el desarrollo de este proyecto una plataforma tecnológica compatible con SDN y en particular con el protocolo OpenFlow con la cual desarrollar un prototipo. El hardware NetFPGA puede configurarse para construir dicha plataforma a un precio significativamente menor que el de un equipo comercial, con funcionalidades y capacidades suficientes para el desarrollo del prototipo.

Por otro lado, como se quiere estudiar la aplicabilidad del enfoque OpenFlow/SDN en la construcción de la RAU2, es necesario además de la construcción de un prototipo implementar algunos casos de uso representativos sobre el mismo en una red experimental. Por tanto, es imperativo la construcción de un laboratorio de pruebas compuesto por un conjunto de nodos que de ser hardware comercial tendría un costo significativamente mayor que el hardware NetFPGA.

Otro de los beneficios de utilizar hardware reconfigurable, es la capacidad de prototipar diferentes dispositivos de red en un mismo equipo. Se puede modificar dinámicamente el laboratorio de pruebas en función de los casos de uso que se quieran probar sin la necesidad de adquirir nuevo hardware, reutilizando el disponible. Además la capacidad para programar el hardware, permite sacar provecho de las características del mismo para la implementación de funcionalidades específicas para la RAU, que en hardware comercial solo se podrían implementar si el mismo ya tuviera incorporadas dichas funcionalidades.

Finalmente la utilización de hardware abierto, en conjunción con software libre y de código abierto posibilitan la construcción de lo que se podría llamar “Router Open Source”, un nodo para la nueva red académica basado en tecnologías abiertas. Esto otorgaría una mayor flexibilidad y capacidades de reutilización del hardware disponible en la red académica avanzada.

3.4 ¿Qué arquitectura basada en SDN utilizar?

Naturalmente, otra de las decisiones de diseño importantes a tomar es la elección de una de las arquitecturas existentes basadas en el enfoque SDN. Dentro de las alternativas existentes, OpenFlow se presenta como una opción madura y probada [41]. Como se mencionó en el estado del arte OpenFlow se ha caracterizado por un desarrollo sostenido y una amplia adopción tanto por la academia como por la industria, además de ser compatible con una amplia variedad de tecnologías. Debido a esto cuenta con una amplia comunidad de usuarios, extensa documentación y es soportado en varios productos comerciales (algunos ejemplos son [48], [23], [20] y [27]. Finalmente se puede agregar que OpenFlow está íntegramente desarrollado bajo la filosofía Open Source (código abierto).

Por estas razones, se decide utilizar OpenFlow como la implementación de SDN en el desarrollo del prototipo.

Por otro lado, OpenFlow ha ido incorporando nuevas funcionalidades a medida que evolucionaron las versiones del protocolo. Actualmente con cinco versiones estables entre las cuales se pueden destacar la versión 1.4 (versión disponible más reciente al momento de iniciar este trabajo) y versión 1.5 (versión más reciente al momento de culminar este trabajo). Se debe decidir entonces cuál es la versión del protocolo a utilizar, teniendo en cuenta a su vez que la versión elegida debe garantizar:

- Soporte para MPLS, tanto en la capacidad de reconocer los cabezales como para la manipulación de los mismos mediante las primitivas POP, PUSH y SWAP
- Soporte para calidad de servicios (QoS)

A partir de la versión 1.3.1 OpenFlow brinda tanto soporte completo para MPLS, así como también incorpora el concepto de métricas por flujo, orientado a implementar funcionalidades de QoS. Por ello se decide utilizar la versión 1.3.1 pese a no ser la más nueva para la construcción del prototipo.

En la siguiente sección veremos por qué es de interés trabajar con la versión de OpenFlow más sencilla y minimalista posible que dé soporte a todas las funcionalidades y restricciones impuestas sobre el prototipo.

3.5 Alternativas de diseño para el router

Como se menciona en el capítulo 1, una de las premisas para la construcción del prototipo es la utilización del hardware NetFPGA. El mismo es utilizado en la construcción de cada nodo del prototipo; por lo que asumiendo el uso de OpenFlow en la arquitectura, es necesario partiendo del mismo obtener nodos compatibles con OpenFlow.

Existen dos estrategias bien definidas para la construcción un switch OpenFlow partiendo del hardware NetFPGA y utilizando los diferentes proyectos de la plataforma. Una de ellas es programar el hardware para que se comporte como un switch compatible con el protocolo OpenFlow y la otra es

programar el hardware para que se comporte como una tarjeta de red estándar, e implementar todo el comportamiento de un switch OpenFlow en software.

Para la primer estrategia se cuenta con un proyecto de la plataforma y disponible libremente en el repositorio de código fuente. No obstante este proyecto presenta una dificultad y es que a pesar de haber sido diseñado para soportar en un futuro cualquier versión disponible del protocolo OpenFlow, en su versión actual solo soporta un conjunto reducido de funcionalidades de la versión 1.0 de dicho protocolo. Como se mencionó anteriormente la mínima versión de este protocolo que permite soportar el conjunto de requerimientos es la versión 1.3.1. Esto conlleva a la necesidad de extender el proyecto existente, para soportar las nuevas características incorporadas en las sucesivas versiones posteriores a la 1.0, ó al menos aquellas que son esenciales para soportar las características pretendidas sobre el prototipo.

Para la segunda estrategia se cuenta con un proyecto de la plataforma NetFPGA denominado ReferenceNIC. Este proyecto habilita a programar el hardware para que se comporte como una placa de red estándar. Adicionalmente se debe incluir o desarrollar herramientas que permitan implementar por software el comportamiento de un switch OpenFlow. En particular sobre este último punto vale la pena destacar la existencia de Open vSwitch, herramienta que entre otras características realiza esto mismo utilizando hardware convencional como una placa de red estándar.

En las tablas 3.1 y 3.2 se exponen comparativamente las principales ventajas y desventajas de cada alternativa.

Ventajas	
Extender proyecto OpenFlow NetFPGA	ReferenceNIC + Open vSwitch
• Óptimo aprovechamiento de la capacidad de cómputo y procesamiento del hardware disponible. • Mayor posibilidad de lograr velocidades de procesamiento competitivas con productos comerciales similares. • Mayor posibilidad de obtener resultados aceptables en performance para puesta en producción.	• No se tiene la necesidad de modificar o desarrollar software en el lenguaje y en el entorno de programación de la tarjeta NetFPGA. • Evitar desarrollar software para la NetFPGA ahorra tiempo del proyecto que se puede invertir en otras líneas de trabajo igualmente importantes. • Programar el hardware NetFPGA con proyectos precompilados como el ReferenceNIC requiere de licencias de software que son accesibles sin costo ya sea mediante licencias gratuitas o de prueba.

Table 3.1 OpenFlow NetFPGA vs ReferenceNIC + Open vSwitch - Ventajas

Desventajas	
Extender proyecto OpenFlow NetFPGA	ReferenceNIC + Open vSwitch
• Extender el proyecto existente en sí mismo constituye un empresa del porte de un proyecto de fin de carrera • El conocimiento técnico necesario se perfila más al de un Ingeniero Eléctrico que al de un Ingeniero en Computación lo cual constituye un riesgo del proyecto. • Desarrollar software para el hardware NetFPGA y compilarlo requiere de licencias de software costosas.	• No se aprovecha de forma óptima las capacidades de procesamiento del hardware disponible. En otras palabras se tiene hardware "caro" y potente en forma ociosa. • Los resultados obtenidos en relación al rendimiento del prototipo probablemente no sean los esperados para un equipo de producción.

Table 3.2 OpenFlow NetFPGA vs ReferenceNIC + Open vSwitch - Desventajas

Teniendo presente el alcance del proyecto y el tiempo disponible para su ejecución, se optó por la estrategia de programar el hardware con el proyecto ReferenceNIC e implementar en software funcionalidades de OpenFlow. Con esta estrategia se logra obtener en forma rápida un prototipo de switch OpenFlow con el cual trabajar en la implementación del plano de control, desarrollando estrategias para construir servicios en una red híbrida IP/MPLS, así como diseñar casos de uso y un laboratorio de pruebas que permitan validar tecnológicamente la solución propuesta.

3.6 ¿Qué implementación de Controlador OpenFlow utilizar?

En el desarrollo de este proyecto, basándose en el estudio del estado del arte de las redes definidas por software realizado y en particular en el estudio sobre los controladores OpenFlow disponibles, se plantean dos alternativas que se ajustan bien a los requerimientos de este proyecto. Estas alternativas surgen de analizar todas las propuestas estudiadas y comparadas en la tabla 2.1.

Se elige dentro de este conjunto, las propuestas OpenDaylight y Ryu puesto que están desarrolladas bajo la filosofía de software libre y de código abierto, cuentan con soporte, tienen una buena comunidad de usuarios, soportan el protocolo OpenFlow en su versión 1.3.1 y son las dos propuestas más utilizadas en la comunidad SDN/OpenFlow en la actualidad.

Para elegir el controlador a utilizar en el prototipo, se instalan ambas implementaciones y utilizando el entorno de emulación Mininet se desarrollan pequeñas aplicaciones para cada alternativa. Comparando los resultados obtenidos y evaluando principalmente la expresividad y facilidad para el desarrollo de aplicaciones en cada entorno se resuelve utilizar el controlador Ryu para la implementación del prototipo de acuerdo a las siguientes razones:

- El ambiente de desarrollo de OpenDaylight hace un uso intensivo de los recursos CPU y Memoria de un equipo, impidiendo ejecutarlo en un equipo de capacidades limitadas. A su vez la instalación del mismo es compleja y requiere de módulos adicionales en gran medida.
- Empíricamente el tiempo de adaptación al entorno de desarrollo de OpenDaylight es considerablemente mayor al de Ryu (curva de aprendizaje).
- El controlador Ryu está completamente implementado en el lenguaje Python y así lo debe ser también las aplicaciones que se ejecutan sobre el. Al ser Python un lenguaje interpretado, se gana agilidad en el proceso de desarrollo, verificación y despliegue de las aplicaciones en comparación a un lenguaje compilado como el utilizado por OpenDaylight (Java).
- El enfoque de Ryu es más minimalista y académico que el enfoque de OpenDaylight. Mientras que este último está enfocado a la industria y cuenta con un extenso conjunto de módulos para la compatibilidad de diferentes protocolos, Ryu ofrece una implementación más liviana enfocada en el protocolo OpenFlow en exclusividad.

Capítulo 4

Diseño e Implementación del prototipo

Este capítulo está destinado a la comprensión de los aspectos principales que hacen al diseño e implementación del prototipo para la RAU2, tomando como punto de partida las decisiones reseñadas en el capítulo anterior.

Por otro lado aspectos muy técnicos o problemas encontrados en cada componente de la arquitectura son tratados en profundidad en los apéndices, citándose aquí las referencias en caso que corresponda para complementar su lectura.

4.1 Lineamientos generales

Como se menciona en el capítulo destinado al estado del arte, la arquitectura del prototipo a construir está orientada a la implementación de servicios de VPN IP/MPLS Multipunto.

Sin embargo varios aspectos de la arquitectura aún deben ser definidos. Es necesario definir la estrategia para calcular las rutas o caminos en la red del prototipo (algoritmo de ruteo) y la forma en que se asignan y distribuyen etiquetas MPLS para implementar el plano de reenvío (algoritmo de distribución de etiquetas).

A continuación se explica de que forma el prototipo implementa cada uno de los aspectos anteriores.

4.1.1 Algoritmo de Ruteo

Una alternativa para el cálculo de rutas en el prototipo es utilizar protocolos de ruteo en base a IP, sacando provecho así de algoritmos existentes, robustos y ampliamente probados tanto en la academia como en la industria.

Históricamente se han utilizado algoritmos distribuidos para el cómputo de rutas, siendo algunos de los más renombrados en la literatura OSPF [80], RIP [75] y IS-IS [91]. Por otro lado también puede sacarse provecho de la visión global del plano de control de SDN para implementar un algoritmo de ruteo centralizado, sencillo y transparente.

En este trabajo se combinan ambos enfoques, utilizando el protocolo de ruteo OSPF en conjunto con un algoritmo centralizado en el controlador.

Se decide utilizar OSPF dado que es una opción robusta y confiable, además de que existen buenas implementaciones en software libre (por ejemplo Quagga [50] y BIRD[18]). OSPF basa su algoritmo para el cálculo de rutas en el algoritmo Dijkstra (ampliamente conocido para el cálculo de mejores caminos en un grafo) y utiliza como métrica el costo asociado a cada enlace. Este costo es ingresado manualmente por el administrador de la red y es un número único, así que es necesario definir una relación matemática entre las diferentes características de un enlace a considerar (ancho de banda disponible, tecnología, latencia, etc.) y éste número.

Sin embargo, para la definición de políticas en ingeniería de tráfico que permitan hacer balanceo de carga y construcción de caminos alternativos, es necesario incorporar más información en el algoritmo de ruteo, como la saturación de enlaces entre otras restricciones. Esto es equivalente a extender OSPF para implementar un algoritmo de ruteo con restricciones, lo que en algunas implementaciones se denomina CSPF (Constrained Shortest Path First). Además interesa que estas restricciones puedan definirse dinámicamente en función de la alta y baja de servicios en el prototipo.

En el prototipo se ejecuta el protocolo OSPF en cada nodo y el controlador para construir y mantener una base de datos topológica de la red. Luego esta información es utilizada por el algoritmo de ruteo centralizado en el controlador (eventualmente CSPF) para el cómputo de las mejores rutas asociadas al tráfico de cada servicio de VPN.

La clasificación de tráfico, creación de múltiples caminos y balanceo de carga son implementados en el controlador a través de una aplicación SDN, por lo que serán explicados en el capítulo 5.

4.1.2 Algoritmo de distribución de etiquetas

Para la implementación de un algoritmo de distribución de etiquetas, se pueden tomar también dos enfoques posibles: enfoque centralizado y enfoque distribuido.

Siguiendo un enfoque distribuido una de las propuestas existentes es el protocolo LDP (Label distribution Protocol), del cual además se cuenta con la extensión de la herramienta Quagga para soportar este protocolo (Quagga-LDP). Al igual que la implementación normal de Quagga, la cual funciona interactuando con el kernel de Linux para la construcción de la tabla de ruteo, Quagga-LDP interactúa con el kernel de Linux para la construcción de dos tablas. Estas dos tablas implementan los conceptos de FTN (Fec To NHLFE), ILM (Incoming Label Mapping) y NHLFE (Next Hop Label Forwarding Entry) introducidos en la definición del protocolo MPLS. Estas tablas determinan la forma en que un paquete es procesado y reenviado en un nodo.

Quagga-LDP toma como entrada el resultado de la ejecución del protocolo OSPF, para luego ejecutar el protocolo LDP y así poblar las tablas mencionadas.

Como Linux no ofrece nativamente soporte al protocolo MPLS, Quagga-LDP funciona en conjunto con MPLS-Linux; un kernel de Linux modificado para ofrecer una implementación de MPLS a nivel de kernel.

Utilizar Quagga-LDP y MPLS-Linux tiene dos ventajas: en primer lugar no se debe implementar un algoritmo propio con lo cual se ahorra tiempo de implementación y se tiene un algoritmo probado y confiable; en segundo lugar se disminuye la carga de cómputo en el controlador puesto que se ejecuta el algoritmo en cada nodo.

Pese a dedicar un tiempo considerable a la instalación de las herramientas Quagga-LDP y MPLS-Linux, proceso el cual implica entre otras cosas recompilar diferentes versiones de kernels de Linux, no fué posible contar con ambas herramientas funcionando correctamente. A su vez la documentación disponible sobre estas herramientas es escasa e inexacta. En el apéndice B.6 se explica en detalle la experiencia generada trabajando con esta herramienta.

Por consiguiente, en este trabajo decidimos implementar un algoritmo de distribución de etiquetas centralizado en el controlador en vez de utilizar Quagga LDP. Los detalles de implementación del mismo serán explicados más adelante en la sección 5.5.4.

4.2 Construcción del prototipo y arquitectura

A continuación se explica la arquitectura general del prototipo.

La arquitectura del prototipo (ver figura 4.1), sigue la arquitectura del enfoque OpenFlow/SDN como se mencionó anteriormente. En concordancia con esta arquitectura en el prototipo se tienen dos componentes importantes: el controlador SDN y el switch compatible con OpenFlow.

En el controlador SDN se ejecuta RauFlow, la aplicación encargada de proveer una implementación para servicios de redes privadas virtuales, desarrolladas para este proyecto. A su vez esta aplicación implementa el algoritmo de ruteo centralizado, el algoritmo de distribución de etiquetas e implementa a su vez una interfaz gráfica de gestión para la red prototipo.

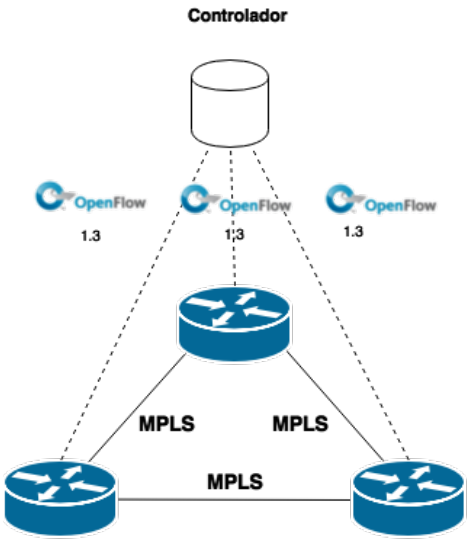

Fig. 4.1 Esquema general del prototipo

Por otro lado se tiene el switch compatible con el protocolo OpenFlow. Este dispositivo, denominado de aquí en más Rau-Switch, es implementado mediante una PC de escritorio con el hardware NetFPGA instalado y el software Open vSwitch entre otras componentes.

Para la implementación del plano de reenvío (encaminar un paquete de un nodo origen a otro destino), en cada nodo del prototipo se utilizan las tablas de flujos de OpenFlow. Mediante el esquema de flujos de OpenFlow se definen reglas de reenvío en base a la conmutación de etiquetas MPLS, así como políticas de clasificación de tráfico.

De esta forma el prototipo para la RAU2 se compone entonces de unos pocos nodos construídos en base a RAU-Switch y un controlador SDN compatible con OpenFlow, en donde se ejecuta la aplicación RAUFlow.

A continuación se explica en detalle cada una de estas componentes y su interacción.

4.3 RAU-Switch

Cada nodo en la red prototipo es lo que en este trabajo se denomina RAU-Switch: un switch IP/MPLS híbrido compatible con OpenFlow. Por un lado es un switch OpenFlow diseñado en base a dicha arquitectura y por otro lado es un router IP/MPLS puesto que conceptualmente el plano de reenvío es implementado a través de la conmutación de etiquetas MPLS, mientras que para el descubrimiento de la topología y la construcción de rutas se utilizan algoritmos en base a IP.

Debido a que en la literatura OpenFlow es común la denominación de switch OpenFlow a cualquier dispositivo, no importando si en la práctica implementa las funcionalidades de un switch de capa 2 ó un router de capa 3, se decidió denominar a esta componente RAU-Switch.

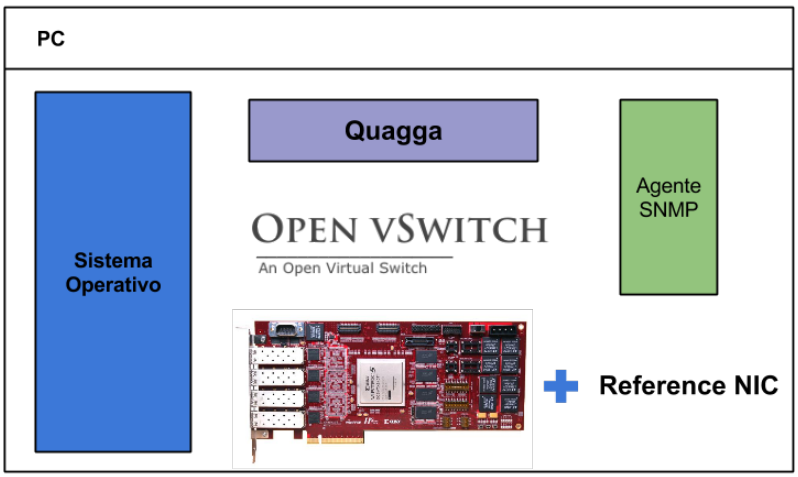

Fig. 4.2 RAU-Switch - diagrama de componentes

La estructura de este dispositivo está caracterizada por las componentes que se muestran en la figura 4.2. Se utiliza una PC de escritorio como plataforma inicial a la cual se le incorpora una tarjeta

NetFPGA-10G programada para que se comporte como una placa de red (proyecto ReferenceNIC) más componentes adicionales de software como Open vSwitch, Quagga y un agente SNMP. A continuación se explica en profundidad el rol de cada componente.

4.3.1 Plataforma de la PC

El switch está construído sobre la plataforma de una PC de escritorio convencional. En particular se trabaja con un procesador Intel Core i7 de 64 bits, una Mother ASUS ROG Maximus Formula VI, 16GB de memoria DDR3, y un disco HDD de 1TB de capacidad. En el anexo I.2 pueden encontrarse estos detalles con mayor profundidad.

4.3.2 Sistema Operativo

En relación al sistema operativo, se decide trabajar con Ubuntu 12.04 sobre una arquitectura de 64 bits pese a que la opción más natural sea utilizar la plataforma Fedora 14 o alguna versión superior de la misma (la tarjeta NetFPGA 10G fué desarrollada y verificada trabajando sobre esta plataforma además de que el fabricante recomienda utilizar esta plataforma).

En este proyecto se prueban diferentes versiones de la plataforma recomendada como Fedora 17 y Fedora 19 además de la versión 14, detectando problemas de compatibilidad entre la placa madre y dichas versiones además de falta de drivers para el cable JTAG. También se prueban diferentes versiones de la distribución Ubuntu, logrando en particular instalar y configurar exitosamente el hardware sobre la plataforma Ubuntu 12.04.

De esta forma y siguiendo siempre la premisa de trabajar con software libre y código abierto se elige Ubuntu 12.04 como plataforma de trabajo.

4.3.3 Hardware NetFPGA

El hardware NetFPGA puede funcionar tanto conectado a una PC por un slot PCIe (modo servidor), como conectado únicamente a una fuente de energía eléctrica (modo standalone). En el diseño planteado el hardware se encuentra conectado a la PC mediante un slot PCIe.

Herramientas de Programación

El hardware puede programarse mediante un cable programador JTAG y la herramienta Impact de la suite de desarrollo de Xilinx ISE SDK. Para ello es indispensable contar con una estación de trabajo con dichas herramientas instaladas, habilitando la programación de la tarjeta NetFPGA que luego será colocada en la PC de cada nodo. A su vez esta suite se compone por varias herramientas las cuales están licenciadas. Estas licencias además contemplan la arquitectura del chip Xilinx que en este caso tiene la tarjeta NetFPGA, por lo que es indispensable contar con el paquete de licencias apropiado al modelo de chip con que se trabaja (en nuestro caso Virtex5) y a las herramientas utilizadas.

Esta estación de trabajo puede o bien ser la propia PC utilizada para el switch (alternativa utilizada en este proyecto) ó bien puede ser una PC independiente.

Programación simple

El hardware puede programarse al menos de dos formas diferentes; una de ellas es lo que aquí se denomina programación simple. Esta estrategia consiste en la utilización de la herramienta Impact y el cable JTAG para grabar en dos chips de la tarjeta (chip FPGA y chip CPLD) dos archivos binarios, uno con la arquitectura de un proyecto y otro con la implementación del mismo.

Como principales ventajas de esta estrategia se destacan su simplicidad, no requiere de licencias pagas (puede descargarse una licencia gratuita para la herramienta Impact) y además es el procedimiento que se describe en la documentación de la plataforma NetFPGA.

No obstante presenta una desventaja importante y es que al producirse un corte de corriente el hardware pierde su programación . Concretamente el contenido del chip FPGA es borrado y solo perdura el contenido del chip CPLD.

Programación persistente

El hardware NetFPGA cuenta en su arquitectura con dos unidades de memoria flash (Flash A y Flash B). En la programación persistente estas unidades se utilizan para almacenar la programación del hardware, permitiendo que en cada encendido el chip FPGA sea programado a partir del contenido de una de estas unidades. Por defecto el chip siempre se programa con el contenido de la memoria flash A, habilitando su reprogramación desde la memoria flash B vía la interfaz PCIe (por mayores detalles de este procedimiento ver [31]).

Reprogramar el contenido del chip FPGA en tiempo de encendido, así como también mediante la interfaz PCIe requiere de módulos adicionales tanto en el contenido del chip FPGA, como en el del CPLD. En particular los proyectos ReferenceNIC [39], ReferenceSwitch [37] y ReferenceRouter [40] incorporan estas características.

En el procedimiento empleado, inicialmente se programa el hardware con el proyecto ReferenceNIC utilizando la Programación Simple. Luego es necesario transformar la implementación del proyecto (archivo bitfile) en un archivo con el formato apropiado para ser almacenado en una de las memorias flash (archivo binario). Esto último se realiza utilizando herramientas incluídas en la plataforma de NetFPGA. Finalmente utilizando la herramienta **pcieprog** también de la plataforma, se transfiere el archivo generado a una de las memorias.

Cabe destacar que durante la ejecución de este procedimiento se detectaron dos BUGs en el proyecto ReferenceNIC los cuales fueron reportados al equipo de desarrollo de NetFPGA a través de la lista oficial de correos de soporte. Por más detalles referirse al apéndice A.

Otro detalle a destacar de esta estrategia, es el requisito de contar con licencias pagas para la suite de desarrollo Xilinx ISE SDK. En relación a este inconveniente se solicitó una donación de licencias mediante el programa de apoyo a universidades de Xilinx, obteniéndose a través de este programa las

licencias necesarias para programar el hardware de forma persistente. Por mayores detalles referirse al apéndice B.3.

4.3.4 Open vSwitch

En la arquitectura del dispositivo, Open vSwitch es el encargado de implementar el plano de datos de OpenFlow. En otras palabras es la componente que convierte al nodo construido en un switch OpenFlow, mediante una implementación en software.

Siguiendo la especificación de OpenFlow, Open vSwitch implementa el concepto de tabla de flujos. Estas tablas (Open vSwitch implementa 256 tablas) son utilizadas por la aplicación RAUFlow para construir caminos en la topología, instalando en forma de flujos OpenFlow las reglas de reenvío que sean necesarias en cada nodo involucrado en un camino.

Para implementar el plano de reenvío en base a la conmutación de etiquetas MPLS, se utilizan primitivas del protocolo OpenFlow, que permiten colocar y extraer etiquetas MPLS de un paquete (PUSH y POP), así como clasificar un paquete acorde al valor de la etiqueta MPLS contenida en él.

Vale la pena destacar en relación a este último punto, que la documentación de la herramienta presenta inconsistencias entre algunas de las funcionalidades declaradas y las realmente implementadas. En particular la manipulación de etiquetas MPLS no está completamente soportada. Por mayores detalles en relación a estos aspectos y la resolución de otros problemas detectados al trabajar con esta herramienta referirse al apéndice B.5.

Por otro lado, resulta interesante tener en cuenta que las operaciones de MATCH, PUSH y POP de MPLS son implementadas en modo usuario, con una considerable penalización en rendimiento en comparación a una implementación en modo Kernel.

4.3.5 Quagga

En cada nodo se ejecuta una instancia del software de enrutamiento Quagga, configurada para ejecutar el demonio Unix OSPF. Este demonio Unix ejecuta el protocolo de igual nombre, utilizado en la arquitectura propuesta para obtener información de cada nodo y sus adyacencias en la topología, guardando dicha información en una base de datos local (Link-State-Database o LSDB). Entre los datos que se incluyen en esta base de datos topológica se destaca el costo asociado a cada adyacencia.

Como se menciona anteriormente, OSPF se utiliza solamente para la construcción de la base de datos topológica. Si bien en un inicio se pensó en trabajar con la salida de dicho algoritmo, finalmente se decidió trabajar con el algoritmo de ruteo centralizado implementado en el controlador. Por ello en la versión final del prototipo no se utiliza la salida de este algoritmo a pesar de que la arquitectura permite su utilización para alimentar la entrada de procesos que se ejecuten en el controlador.

De esta forma cada switch del prototipo así como el propio controlador ejecuta una instancia de Quagga con el demonio Unix OSPF configurado como se muestra en el apéndice D. Cabe destacar de dicha configuración que la adyacencia entre un switch y el Controlador (enlace utilizado para el canal de comunicación OpenFlow) tiene asociado costo infinito para evitar que sea utilizado tanto por el

algoritmo OSPF como por el algoritmo SPF en la construcción de algún camino. En la implementación de Quagga el costo infinito es representado por el número 65535 (16 bits).

4.3.6 Agente SNMP

El protocolo de comunicación OpenFlow dentro de las estructuras de datos utilizadas para el intercambio de información entre un switch y el Controlador, no prevé una forma de comunicar el direccionamiento IP propio del equipo (direcciones IP de cada interfaz física). En particular la estructura *ofp_port* utilizada por el mensaje *OFPMP_PORT_DESCRIPTION* no provee dicho campo (ver especificación del protocolo OpenFlow 1.3 [85]). Por esta razón es necesaria una forma de comunicar al controlador información adicional sobre características de cada nodo como la dirección IP de una interfaz, que no puede ser enviada a través del protocolo OpenFlow. Para evitar modificar el protocolo extendiéndolo para comunicar dicha información, se utiliza un agente SNMP.

SNMP definido a través de los RFC1065 [87] y RFC1157 [58] entre otros, es un protocolo que permite el intercambio de información entre dispositivos de red y una entidad administradora. Para su despliegue en una red son necesarias tres componentes:

1. Dispositivo administrado: Es el dispositivo de red que se quiere monitorizar.
2. Agente: Es el software que se ejecuta en el dispositivo administrado y se encarga de comunicarse mediante el protocolo con el sistema administrador de red.
3. Sistema administrador de red: El dispositivo que se encargará de realizar las consultas sobre la información deseada a los dispositivos administrados.

En el prototipo se instala un agente SNMP en cada switch (dispositivo administrado), para enviar la correspondencia entre números de puerto OpenFlow y direcciones IP al Controlador (sistema administrador).

No es el objetivo de este trabajo entrar en detalle sobre el protocolo SNMP por lo que si el lector desea profundizar en estos conceptos se recomienda seguir las referencia mencionadas anteriormente.

4.4 Entidad Controlador

Como se muestra en 4.1 cada nodo del prototipo responde a una entidad denominada Controlador. Esta entidad en la práctica es una PC convencional con determinadas componentes de software (ver figura 4.3) entre las cuales se destaca el controlador SDN Ryu, pilar fundamental en la implementación de la aplicación RAUFlow. A su vez se encuentran los módulos Administrador SNMP utilizado en la comunicación con los agentes SNMP instalados en cada nodo RAU-Switch y LSDBSync utilizado en el proceso de extracción de información de la base de datos topológica de Quagga.

Estas tres componentes se ejecutan en una PC de escritorio con Ubuntu 12.04 como sistema operativo.

A continuación se explica en detalle cada una de las tres componentes mencionadas.

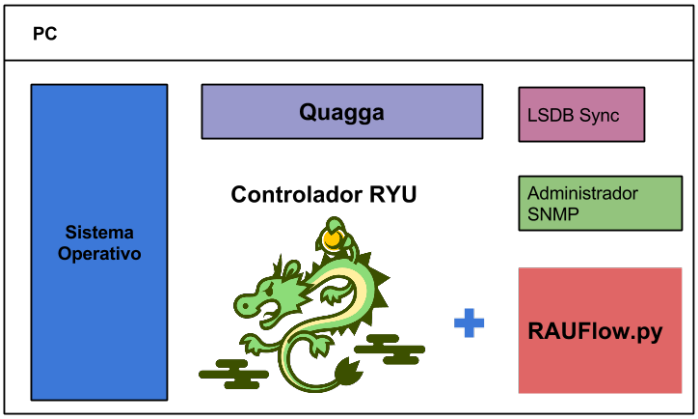

Fig. 4.3 Diagrama de componentes del Controlador

4.4.1 Controlador Ryu

Ryu es la alternativa de software elegido para implementar el controlador SDN compatible con el protocolo OpenFlow. Utilizando esta herramienta para la construcción y ejecución de una o más aplicaciones SDN, se implementa prácticamente la totalidad del plano de control del prototipo. Esto es en otras palabras el modelado de la realidad, la implementación de funcionalidades para la creación de servicios de redes privadas, implementación de clasificación de tráfico y eventualmente la implementación de políticas de ingeniería de tráfico y calidad de servicios (QoS).

Dada la complejidad en el diseño y arquitectura de esta componente, se destina el próximo capítulo a la explicación de las diferentes características e implementación de las aplicaciones Ryu que constituyen el plano de control del prototipo, así como la interacción de las otras componentes mencionadas dentro del plano de control como los módulos LSDBSync y Administrador SNMP.

4.4.2 Quagga

El controlador ejecuta una instancia de Quagga, obteniendo de esta forma acceso local a la información de la base de datos topológica construida por OSPF (Link-State-Database), una vez que el algoritmo converge.

Para obtener esta información y detectar cambios la topología se utiliza el módulo LSDB Sync.

4.4.3 LSDB Sync

LSDB Sync se encarga de tomar la información de la base de datos topológica una vez que el algoritmo OSPF converge, procesarla y enviarla a las aplicaciones que se ejecutan en el controlador.

Esta componente consta de dos módulos. El primero se encarga de escuchar los mensajes del protocolo enviados por Quagga y generar un evento cuando el mismo converge luego de producirse un cambio en la topología. Este evento luego es tomado por el segundo módulo el cual toma la información topológica en la base local del controlador, la procesa y luego la envía a las aplicaciones

en el controlador. Luego se produce un proceso de actualización topológica, en el que eventualmente el controlador utilizando el módulo Administrador SNMP obtiene información adicional sobre los dispositivos del prototipo.

4.4.4 Administrador SNMP

El administrador SNMP es utilizado para consultar al agente SNMP instalado en un nodo, por la correspondencia entre números de puerto OpenFlow y direcciones IP. Esta componente es utilizada por las aplicaciones en el Controlador cada vez que es preciso obtener esta información de mapeo para un nodo (por ejemplo en el proceso de actualización de la topología).

4.4.5 Visión general de la arquitectura

En resumen, los nodos del prototipo son construídos en base al hardware NetFPGA y una PC convencional a la cual se le instalan diferentes componentes de software de forma de implementar un switch MPLS/OpenFlow. Luego el plano de control de la red se implementa en una PC con el software controlador Ryu instalado más un conjunto de módulos adicionales.

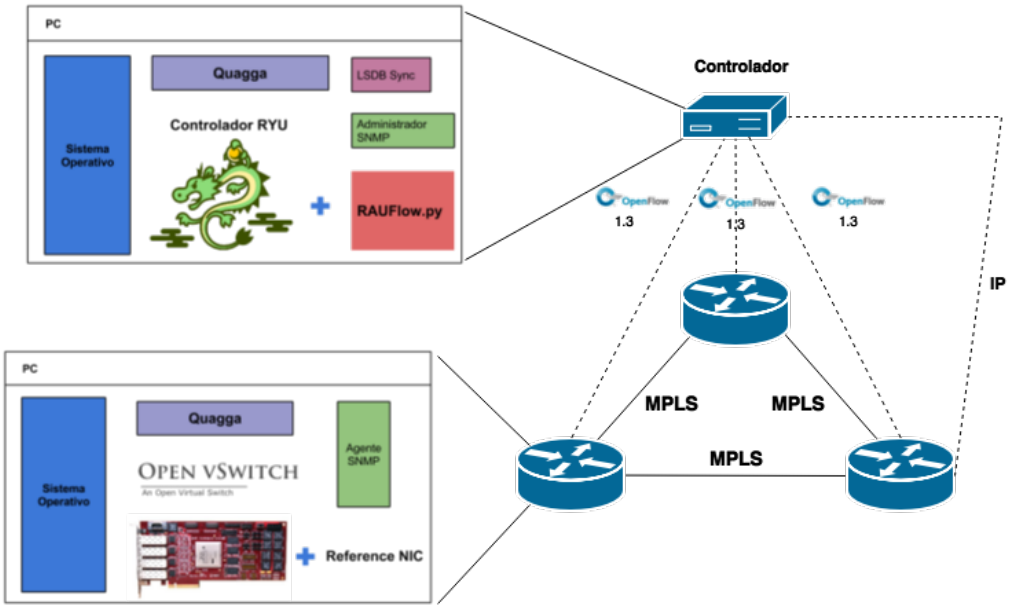

Fig. 4.4 Vista lógica ampliada del prototipo

En el siguiente capítulo se explica en detalle el proceso de diseño e implementación de la aplicación RAUFlow, así como la interacción con las diferentes componentes de RAU-Switch.

Capítulo 5

RAUFlow

En capítulos anteriores se han detallado las características más importantes del prototipo así como sus componentes. Resta detallar entonces, el diseño e implementación de RAUFlow la aplicación encargada de implementar el plano de control en el prototipo.

En el presente capítulo se propone un análisis de RAUFlow siguiendo un proceso de diseño tradicional de Ingeniería de Software dividido en cuatro etapas. Una primera etapa de análisis de requerimientos, una segunda etapa de identificación de casos de uso, una tercera etapa de diseño del modelo de datos y finalmente una cuarta etapa destinada al diseño general de la arquitectura. Además se presentan los aspectos más importantes relacionados a la implementación de RAUFlow como por ejemplo las implementaciones del algoritmo de ruteo y del algoritmo de distribución de etiquetas, como se implementa clasificación de tráfico y como se puede implementar QoS entre otros detalles.

5.1 Análisis de requerimientos

En la sección 3.1 se definieron los requerimientos recabados para el prototipo de la RAU2. De ellos y de un trabajo de análisis sobre la realidad modelada se desprende la siguiente tabla de requerimientos para RauFlow:

Requerimientos Funcionales
• El Sistema debe proveer la facilidad para obtener la información asociada a cada nodo de la red, permitiendo a su vez agregar información que facilite la identificación del mismo para un usuario. • El Sistema debe proveer la facilidad para agregar, modificar y eliminar servicios de redes virtuales. • El Sistema debe proveer la facilidad para obtener toda la información relevante a un servicio de red privada. • El Sistema debe permitir visualizar los caminos construídos para encaminar el tráfico de una red virtual en particular, a través de la red prototipo. • El Sistema debe proveer la facilidad para visualizar el estado de las tablas de flujos asociadas a cualquier nodo de la red del prototipo.

Requerimientos no Funcionales
• Se debe utilizar siempre que sea posible herramientas de software libre y código abierto.

Teniendo en cuenta la descripción del problema y los requerimientos anteriores, se procede con el modelado de la realidad.

5.2 Modelado de la realidad

Para la representación de la realidad se utiliza el paradigma de orientación a objetos. De esta forma el modelo de datos queda representado a través del siguiente diagrama de clases de diseño (ver ver figura 5.1).

En el mismo se destacan en color amarillo las clases utilizadas para representar la topología de red y sus elementos (Nodos, Interfaces y Enlaces). Cabe destacar sobre estas tres clases que en la representación asumida se busca modelar la topología como un multigrafo dirigido. Esto se debe a que:

1. Tradicionalmente se utilizan grafos para el modelado de una topología de red, siendo una representación sencilla y clara.

2. En MPLS un camino o mejor llamado LSP tiene “sentido”, permitiendo por ejemplo asegurar un valor de ancho de banda en un enlace para un sentido y un valor diferente para el otro. Además permite establecer caminos diferentes para el tráfico en un sentido y en otro, priorizando por

ejemplo el tráfico en uno de los sentidos al utilizar el "mejor" camino. Utilizando grafos dirigidos puede modelarse este comportamiento.

3. En la práctica nada impide que dos equipos o nodos estén conectados por más de un enlace. De hecho este escenario es bastante beneficioso para asegurar conectividad ante la falla de enlaces. Utilizando multigrafos este escenario es modelado de forma clara.

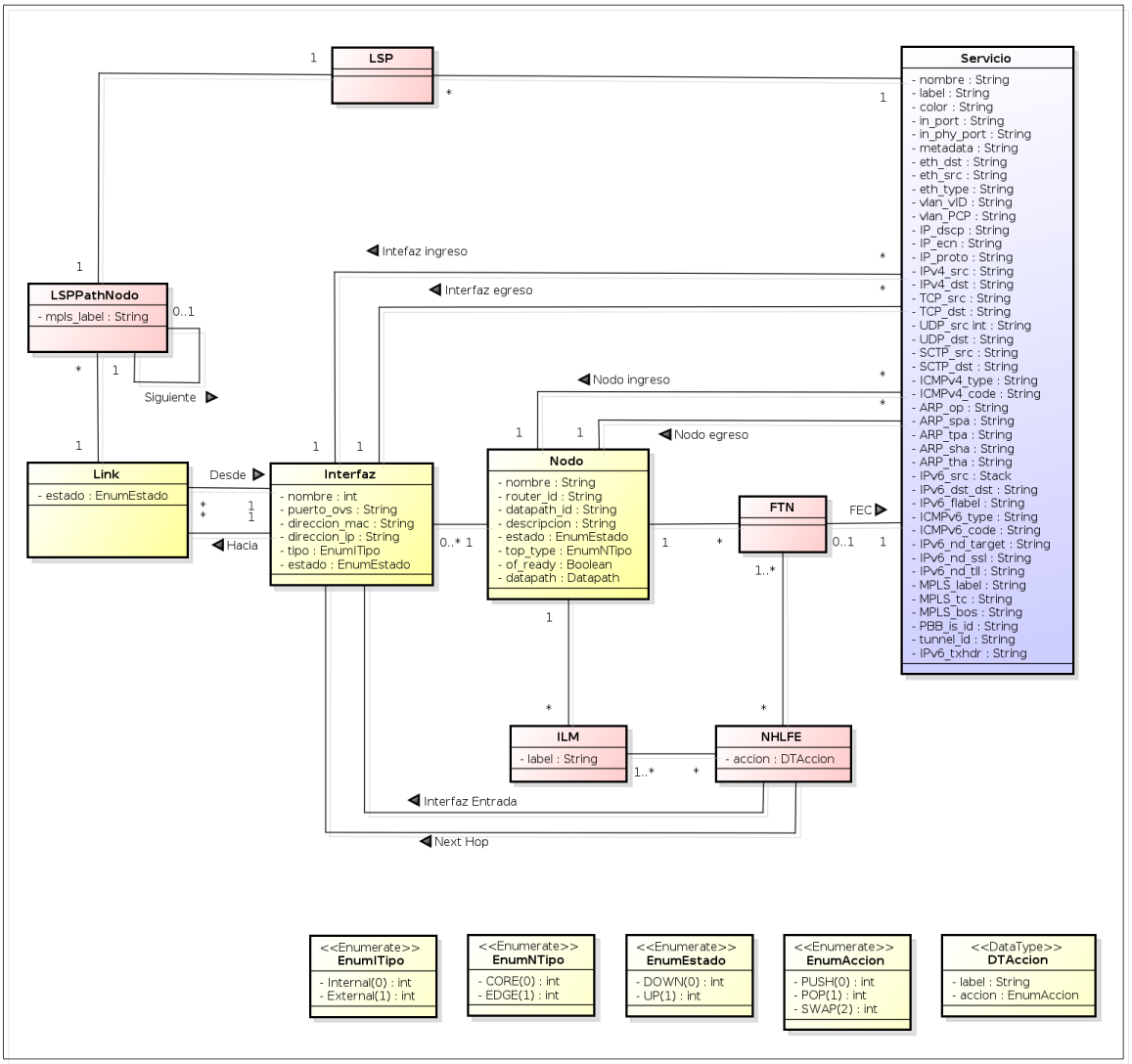

Fig. 5.1 Modelo de datos

En un mulrigrafo dirigido se tienen nodos y aristas con sentido. Cada nodo de la red puede ser representado por un nodo del grafo y cada enlace entre un par de nodos con dos aristas (una arista para cada sentido del enlace).

En el modelo se tiene además el concepto de Interfaz de dispositivo. Este concepto re-define la noción de adyacencia entre dos nodos en el grafo de la siguiente forma: existe una arista para un par

de nodos si cada uno de ellos está asociado a una instancia de la clase Interfaz y existe una instancia de la clase Link asociada a ambas interfaces mediante las relaciones “Desde” y “Hacia”; indicando además éstas el sentido del dicho Link.

Cabe destacar que en el modelo la clase Nodo representa solamente dispositivos de capa 3. Esto quiere decir que dispositivos de capa 2 como switches no son contemplados. Sin embargo no se asume que el nodo esté físicamente implementado por RAU-Switch, con lo que se permite modelar nodos implementados en base a otro tipo de dispositivos.

Por otro lado en rosado se destacan las clases utilizadas para representar los principales conceptos de **MPLS** como las tablas FTN, ILM, NHLFE y el concepto de LSP. Si bien esta última clase no tiene atributos asociados en esta versión del modelo de datos, eventualmente tiene sentido en un futuro que se puedan asociar atributos del camino como el máximo ancho de banda disponible y algunas otras propiedades orientadas a funcionalidades de QoS.

Finalmente en azul se destaca el concepto Servicio con el cual se busca representar un servicio de red privada virtual. En particular cada instancia de Servicio representa una red privada punto a punto entre dos nodos en el prototipo; por ello esta clase tiene asociados los atributos nodo de ingreso y nodo de egreso. Además se asume que por cada nodo de borde se tiene a lo sumo una única red privada conectada directamente a cada interfaz. Por ello un servicio queda determinado por los pares <nodo, interfaz> origen y <nodo, interfaz> destino. Luego una red privada multipunto puede construirse definiendo servicios punto a punto para cada par de nodos involucrados.

Esta clase también tiene asociados atributos con los cuales se implementa clasificación de tráfico; más adelante en este capítulo se detallan este y otros aspectos asociados a la implementación de esta clase.

Teniendo en cuenta los requerimientos mencionados, y el modelado de la realidad presentado, se procede a identificar los casos de uso.

5.3 Casos de uso

La lista de casos de uso presentada a continuación se corresponde con un conjunto de funcionalidades básicas, que permiten explorar el potencial del enfoque SDN aplicado a la implementación del prototipo para la RAU2.

- **CU1. Listar Servicios**: Devuelve un listado con los servicios existentes en el sistema, mostrando para cada uno la información básica asociada.
- **CU2. Seleccionar Servicio**: Selecciona un Servicio de un listado(CU2. Listar servicios).
- **CU3. Ver Servicio**: Devuelve la información asociada al Servicio.
- **CU4. Modificar Servicio**: Modifica una instancia de Servicio, actualizando los atributos seleccionados por el usuario con los valores ingresados.

- **CU5. Eliminar Servicio**: Elimina un Servicio existente en el Sistema.
- **CU6. Agregar Servicio**: Crea un nuevo Servicio de red privada en el Sistema, con la información de nodos y sus respectivas interfaces a las que las subredes están directamente conectadas con la red del prototipo. Además solicita para cada servicio los datos utilizados para la clasificación del tráfico asociado.
- **CU7. Ver Topología**: Muestra la topología de red mediante una representación gráfica.
- **CU.8 Filtrar LSPs**: Superpone en la representación gráfica de la red (CU7.Ver Topología) un conjunto de LSPs seleccionados.
- **CU9. Seleccionar Nodo**: Selecciona un nodo de una lista de Nodos (CU7. Ver Topología).
- **CU10. Ver información básica Nodo**: Despliega la información asociada al nodo seleccionado de la topología (CU9. Seleccionar Nodo).
- **CU11. Ver tabla de Flujos Nodo**: Muestra la información de la tabla de flujos OpenFlow asociada al nodo seleccionado (CU9. Seleccionar Nodo).
- **CU12. Ver tablas MPLS**: Muestra la información asociada a las tablas FTN, ILM y NHLFE asociadas al nodo seleccionado (CU9. Seleccionar Nodo).
- **CU13. Editar Información extra Nodo**: Permite ingresar información adicional sobre un Nodo como el Nombre y el tipo (nodo interno o de borde).
- **CU14. Editar Información extra Interfaz**: Ingresa información adicional sobre la interfaz de un nodo seleccionado (CU9. Seleccionar Nodo), por ejemplo si la interfaz es interna o externa.

En la figura 5.2 se representan las dependencias existentes entre los casos de uso mencionados anteriormente.

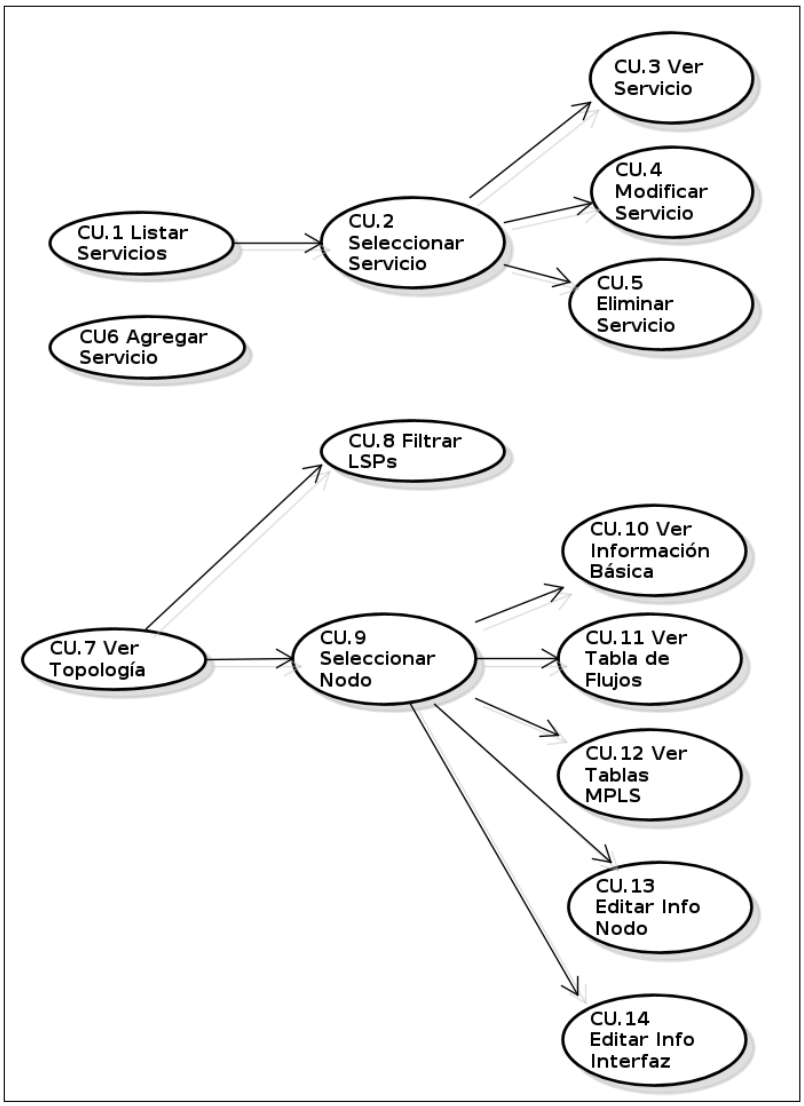

Fig. 5.2 Casos de Uso de RAUFlow

En la siguiente sección se presenta la arquitectura de la aplicación RAUFlow, detallándose las componentes más importantes.

5.4 Arquitectura de RauFlow

Como se menciona en el capítulo anterior, RAUFlow es el nombre dado a la aplicación encargada de implementar el plano de control en el prototipo. Sin embargo como se menciona también en dicho capítulo, la o las aplicaciones que se ejecutan en el controlador Ryu interactúan con diferentes módulos y componentes como el Administrador SNMP o la base de datos topológica de Quagga. Por esta razón tiene sentido pensar en RAUFlow como el conjunto de todas estas componentes las cuales interactuando entre ellas implementan efectivamente el plano de control.

Conceptualmente RAUFlow es un conjunto de aplicaciones de control y gestión de red, basada en el enfoque de SDN e implementado utilizando el software de Control Ryu y el protocolo OpenFlow entre otras tecnologías. No es puramente una aplicación OpenFlow puesto que como se ha mencionado reiteradas veces utiliza un conjunto de componentes y módulos independientes del ambiente del controlador Ryu.

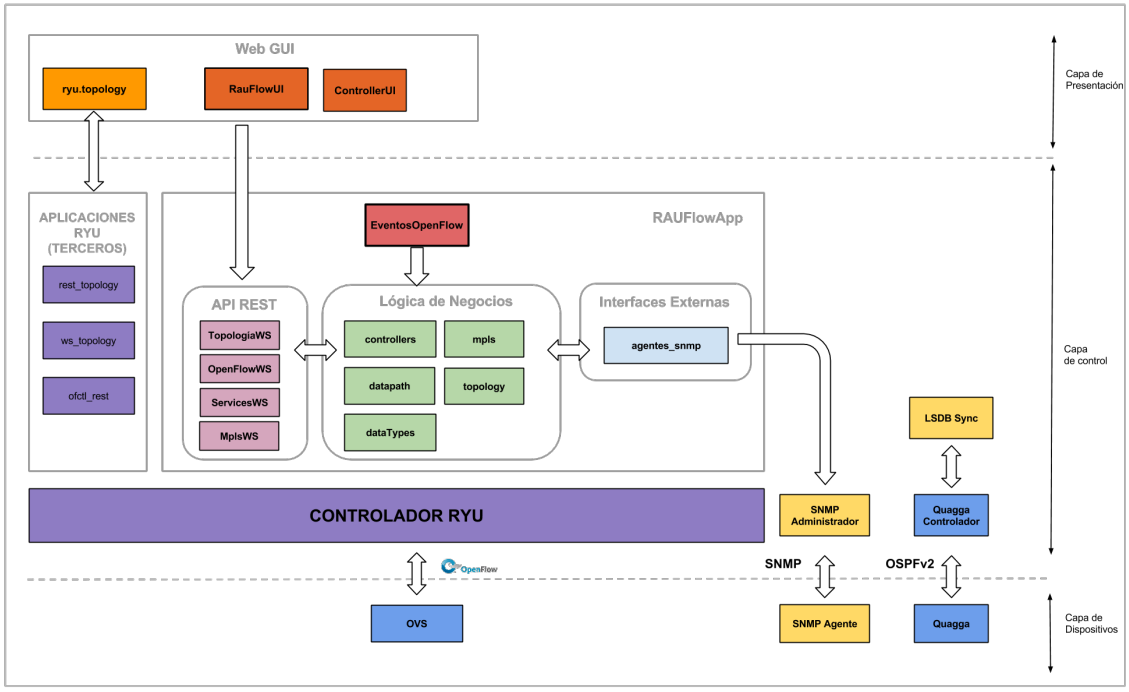

Fig. 5.3 Arquitectura de RAUFlow

Teniendo en cuenta lo anterior, la arquitectura de la aplicación (ver figura 5.3), se caracteriza por un diseño en tres capas lógicas: (1) capa de presentación, (2) capa de control y (3) capa de dispositivos. A continuacíon se explica en detalle cada una de estas capas, así como la interacción entre ellas y los principales detalles de los módulos que las componen.

5.4.1 Capa de Presentación

Esta capa está destinada a proveer funcionalidades para el acceso y manipulación de datos de la aplicación (estado de la topología e información asociada a servicios de redes privadas), así como todas las funcionalidades especificadas como requerimientos en la sección 5.1.

Dentro de la capa de presentación se destacan: (1) una interfaz gráfica identificada en el esquema como RauFlowUI, la cual implementa cada uno de los casos de uso mencionados en la sección 5.3, (2) ControladorUI, componente que implementa los métodos de acceso entre las funcionalidades de RauFlowUI y la capa de control a través de una API Rest de Servicios en la capa de control y (3) un módulo javascript denominado **ryu.topology**.

Ryu incluye a modo de documentación un conjunto de aplicaciones muy sencillas en las que se ejemplifica el uso de las principales funcionalidades del protocolo OpenFlow. Entre estas aplicaciones se encuentra **gui_topology**, la cual brinda una representación gráfica básica de las componentes topológicas en una red OpenFlow (switches, puertos y links). Esta aplicación a su vez se basa en tres aplicaciones Ryu de la capa de control (más adelante se detalla el funcionamiento de ellas cuando se explique la capa de control), una pagina web desarrollada puramente sobre HTML y un módulo de javascript encargado de la manipulación de información entre las aplicaciones Ryu y esta página mediante websockets (este módulo recibe el nombre de **ryu.topology**).

En RAUFlow se combinan las componentes de la aplicación **gui_topology** como el módulo javascript **ryu.topology** y las tres aplicaciones Ryu con el resto de las componentes de la aplicación con el objetivo de mejorar la interfaz gráfica RAUFlowUI, brindando una representación gráfica del estado topológico de la red reutilizando las herramientas existentes.

5.4.2 Capa de Control

La capa de control está conformada por cuatro componentes: el software de control Ryu con su respectivo conjunto de aplicaciones SDN, el módulo LSDBSync, el módulo Administrador SNMP y la instancia de Quagga ejecutada en el Controlador.

Aplicaciones RYU (Terceros)

En RAUFlow se ejecuta un conjunto de tres aplicaciones con el objetivo de facilitar la representación gráfica del estado de la topología, a partir de las funcionalidades de la aplicación **gui_topology**, anteriormente mencionada.

Las tres aplicaciones utilizadas son: (1) rest_topology, (2) ws_topology y (3) ofctl_rest. Las funcionalidades de cada una de estas aplicaciones son las siguientes:

1. rest_topology: Implementa una API REST de servicios para el acceso a información topológica de switches OpenFlow y links existentes entre ellos. Esta API es utilizada por **ryu.topology** para construir la representación gráfica de la red.

2. ws_topology: Mantiene en memoria información de cada switch de la capa de dispositivos registrado con el controlador. A su vez gestiona eventos OpenFlow generados cuando un switch se reporta o deja de reportarse, cuando cambia el estado de un puerto, cuando llega un paquete, entre otros eventos del protocolo.

3. ofctl_rest: Mediante una API REST de servicios expone un conjunto de métodos para acceder a información del plano de datos OpenFlow (datapath OpenFlow) y así obtener todo tipo de información relacionada a un switch en particular. Implementa desde métodos para obtener la tabla de flujos de un dispositivo, estadísticas agregadas por flujo, puerto, grupo de puertos y colas entre otros. Esta aplicación es utilizada por la página web original de **gui_topology** y

por RAUFlowUI en la arquitectura de RAUFlow para el acceso a información estadística de la tabla de flujos de un switch.

Por otro lado se ejecuta sobre el controlador una cuarta aplicación Ryu. A continuación se detalla su estructura y funcionamiento.

RAUFlowApp

Dentro de las aplicaciones Ryu, RauFlowApp es realmente la encargada de implementar el plano de control del prototipo. Esto quiere decir, guardar información de la realidad, servicios de redes privadas creados, guardar el estado de la topología, implementar funcionalidades de QoS, ruteo dinámico etc. Vale la pena aclarar, que a diferencia de las tres aplicaciones anteriores, las cuales fueron desarrolladas por el equipo detrás del software Ryu, esta aplicación es completamente diseñada e implementada en el marco de este proyecto.

En relación al diseño de esta aplicación, como puede apreciarse en la figura 5.3 responde a una organización modular, en la cual se destacan cuatro componentes principalmente: (1) EventosOpenFlow (2) Lógica de Negocios, (3) API REST de Servicios y (4) Interfaces Externas. El funcionamiento de cada uno de ellos es el siguiente:

1. EventosOpenFlow
   EventosOpenFlow es una aplicación Ryu con el esquema básico de este tipo de programas (las aplicaciones Ryu tienen un esquema particular). Se encarga de proveer métodos para los principales eventos del protocolo OpenFlow; por ejemplo cuando un switch se reporta con el controlador por primera vez o deja de reportarse, cuando el controlador recibe un paquete (Packet IN), cuando se produce un cambio en la topología como por ejemplo un puerto en un dispositivo que deja de funcionar, entre otros.

   El modelado de la realidad, la implementación de funcionalidades de acceso a la capa de dispositivos, estructuras de datos almacenadas en memoria, así como los algoritmos de ruteo y distribución de etiquetas son implementados en la componente Lógica de Negocios. De esta forma se desacoplan las responsabilidades, mejorando la claridad y transparencia en el diseño de la aplicación. No obstante esta aplicación es la responsable de gestionar el ciclo de vida de los diferentes objetos asociados a la componente Lógica de Negocios.

2. Lógica de Negocios
   Como se ha mencionado esta componente es responsable del modelado de la realidad y la implementación de las principales funcionalidades del plano de control de RAUFlow. Se subdivide a su vez en diferentes módulos que agrupan funcionalidades acorde a su naturaleza y responsabilidad. Vale la pena destacar a su vez que cada uno de estos módulos se corresponde en la implementación realizada con un módulo de Python. A continuación se explican las responsabilidades de cada uno de ellos:

- **controller:** Este módulo agrupa diferentes controladores de objetos. Actualmente contiene un único controlador fachada responsable de mantener el único punto de acceso a la las componentes de la lógica de negocios. Contiene desde la implementación de funciones para dar de alta Servicios, crear LSPs, obtener el mejor camino entre dos nodos de la red, entre otras.
- **topology:** Agrupa las definiciones de objetos utilizados para representar la topología como las clases Nodo, Interfaz y Link, así como otros conceptos de la realidad.
- **mpls:** Contiene las definiciones de los conceptos FTN, ILM, NHLFE, así como los conceptos de servicio de red y LSP (Label Switched Path).
- **dataTypes:** Mantiene representaciones reducidas de los principales objetos definidos en todos los módulos de la componente de negocios para el intercambio de datos por ejemplo con la capa de presentación.
- **datapath:** Agrupa funcionalidades para el acceso al datapath de OpenFlow, como funciones para agregar y eliminar flujos en un switch, u obtener estadísticas de una tabla de flujos.

Para acceder a la componente de Lógica de Negocios se tiene una API REST de Servicios.

3. API REST de servicios
   Esta componente define métodos para el acceso a los datos y funcionalidades de la componente Lógica de Negocios. Es utilizada por la capa de Presentación para acceder a información de la realidad o crear nuevos servicios. A su vez es utilizada por otras componentes de la capa de control como lo es el caso del módulo LSDBSync para enviar información actualizada de la topología de red al momento de ejecutar RAUFlow por primera vez y luego cuando la topología cambia.

   Se subdivide en varios módulos respondiendo al criterio utilizado para el diseño modular de la anterior componente.

4. Interfaces Externas
   Cuando se enciende RAUFlow y se recibe por primera vez la información topológica, o cuando la topología cambia y se produce una actualización, es necesario obtener información adicional sobre cada dispositivo (hecho ya mencionado en reiteradas ocasiones). Para esto la aplicación utiliza la componente denominada Interfaces Externas. Esta componente funciona como punto de conexión entre la componente Lógica de Negocios y otras componentes externas a la aplicación RAUFlowApp. Un ejemplo de una componente externa es el módulo Administrador SNMP, encargado de obtener la información extra en el proceso de actualización topológica mencionado, utilizando el Agente SNMP instalado en cada dispositivo RAU-Switch del prototipo.

Dentro de esta componente, actualmente existe un solo módulo denominado **agentes_snmp**. Este módulo es el encargado de la ya mencionada comunicación con el módulo Administrador SNMP.

Administrador SNMP

El funcionamiento de este módulo se ha explicado anteriormente en el capítulo 4.

LSDB Sync

Como se muestra en el diagrama de arquitectura, este módulo interactúa con la instancia de Quagga instalada en el controlador en dos formas distintas. Por un lado advierte cuando la topología cambia observando los mensajes que se envían a través del protocolo OSPF en la red, y por otro lado accede, procesa y envía la información de la base de datos topológica LSDB local a la aplicación RAUFlowApp a través de un web service de la API Rest implementada en esta aplicación.

5.4.3 Capa de Dispositivos

En la capa de dispositivos se muestran las diferentes componentes de RAU-Switch que interactúan con la capa de control, indicando para cada una de ellas el protocolo de comunicación utilizado.

Por un lado se tiene la herramienta Open vSwitch de la cual tanto su rol en la implementación del prototipo, como su funcionamiento ya se ha explicado anteriormente. Vale la pena mencionar simplemente, que las únicas componentes de la capa de control de RAUFlow que interactúan con la misma, además de las tres aplicaciones que vienen con Ryu, son la aplicación EventosOpenFlow y el módulo datapath.

También forma parte de la capa de dispositivos, el agente SNMP consultado por el Administrador SNMP en la capa de control mediante el protocolo SNMP.

Finalmente se tiene la instancia de Quagga, la cual a través del intercambio de mensajes del protocolo OSPF con las restantes instancias de Quagga en cada dispositivo RAU-Switch, construyen la base de datos topológica en el controlador.

Una vez presentada y explicada en detalle la arquitectura de RAUFlow, el lector está en condiciones de abordar la siguiente sección, en la que se abordan detalles relacionados a la implementación.

5.5 Implementación

En esta sección se presentan los aspectos más importantes relacionados a la implementación de RauFlow. Entre ellos se destacan la estrategia utilizada para implementar clasificación de tráfico, la implementación de la clase servicio, la implementación de los algoritmos de ruteo y distribución de etiquetas, entre otros.

5.5.1 Clasificación de tráfico

En la arquitectura MPLS, tradicionalmente se utiliza el concepto de FEC (fordwarding equivalence class) para distinguir a un conjunto de paquetes a ser tratados en forma similar (por ejemplo para aplicar técnicas de QoS). Este concepto se define localmente a un dispositivo, determinando la forma en que un conjunto de paquetes son tratados solamente en el mismo.

En RAUFlow se aprovecha la visión global del plano de control SDN para definir una noción de FEC global en una topología de red. En esta redefinición se clasifica tráfico solamente en el nodo de ingreso a la red del prototipo, determinándose allí el camino a seguir por un paquete hasta el nodo de egreso mediante el resultado obtenido en la ejecución del algoritmo de ruteo.

Por otro lado se define un mapeo entre servicios de VPN y etiquetas MPLS, permitiendo identificar el tráfico asociado a un servicio particular en cualquier nodo de la red. Luego a partir del valor de esta etiqueta se puede realizar un procesamiento diferencial en nodos intermedios de un LSP.

En relación a como se implementa clasificación de tráfico, en el prototipo se utilizan las tablas de flujos de Open vSwitch en conjunto con los campos del cabezal OpenFlow (matching fields) para la definición de flujos. Con los matching fields de OpenFlow se define la regla de un flujo (recordar figura 2.4) distinguiendo de este modo entre diferentes clases de tráfico para un procesamiento diferencial.

Dentro del cabezal OpenFlow, existen campos para la definición de las reglas de un flujo, campos para la definición de la acción de un flujo y campos que pueden ser utilizados para la definición de ambas componentes. Esta lista de campos varía con la versión del protocolo OpenFlow y en el contexto del prototipo está acotada a su vez por la implementación de Open vSwitch. En el apéndice C se muestra la lista entera de atributos que contiene el cabezal OpenFlow para la versión 1.3.1 y de éstos la lista de atributos que son soportados por Open vSwitch y que pueden ser utilizados para la definición de reglas y acciones. Ambas listas fueron construidas experimentalmente trabajando con las herramientas Open vSwitch y Ryu.

En el prototipo, el concepto de clasificación de tráfico está estrictamente ligado al concepto de servicio de red privada. En la siguiente sección se explica en detalle la implementación de este concepto.

5.5.2 Implementación de Servicio

La clase Servicio implementa el concepto de servicios de red privada virtual. Como se menciona anteriormente un servicio define una clase de tráfico; por tanto además de la información básica que define a un servicio se tienen asociados también todos los campos utilizados para implementar clasificación de tráfico (matching fields de OpenFlow).

En el prototipo se asume que por cada nodo de borde se tiene una única red privada directamente conectada a cada interfaz externa. Esto permite definir un servicio a partir de la clase de tráfico definida y los pares nodo-interfaz origen y nodo-interfaz destino.

Por otro lado se podría refinar la definición de servicio agregando más dimensiones; una dimensión bastante útil podría ser por ejemplo el tiempo, donde se podría por ejemplo definir un servicio de red privada para un rango horario determinado.

De igual forma se pueden incorporar más dimensiones orientado a brindar una mayor flexibilidad en la definición de servicios, de cara a lo que podrían ser diferentes requerimientos de la RAU2.

De todos modos teniendo en cuenta el alcance del proyecto se decide dejar estas dimensiones extras como una posible línea de trabajo a futuro.

De esta forma la clase servicio queda determinada de la siguiente forma:

```
class Service (object):

        # Atributos generales
        ID                  # str (uuid.uuid4()) ID unico
        name                # Nombre del servicio para

        lsps                # Lista de LSPs para el servicio

        ingress_node        # Nodo de ingreso del servicio
        egress_node         # Nodo de egreso del servicio
        ingress_interface   # Interfaz de ingreso en el nodo
                            # de ingreso
        egress_interface    # Interfaz de egreso en el nodo
                            # de ingreso

        # Campos del cabezal OFv1.3
        in_port          # Switch input port.
        metadata         # Metadata passed between tables.
        eth_dst          # Ethernet destination address.
        eth_src          # Ethernet source address.
        eth_type         # Ethernet frame type.
        vlan_vID         # VLAN id.
        vlan_PCP         # VLAN priority.
        IP_dscp          # IP DSCP (6 bits in ToS field).
        IP_ecn           # IP ECN (2 bits in ToS field).
        IP_proto         # IP protocol.
        IPv4_src         # IPv4 source address.
        IPv4_dst         # IPv4 destination address.
        TCP_src          # TCP source port.
        TCP_dst          # TCP destination port.
        UDP_src          # UDP source port.
        UDP_dst          # UDP destination port.
        SCTP_src         # SCTP source port.
        SCTP_dst         # SCTP destination port.
        ICMPv4_type      # ICMP type.
        ICMPv4_code      # ICMP code.
        IPv6_src         # IPv6 source address.
        IPv6_dst         # IPv6 destination address.
        ICMPv6_type      # ICMPv6 type.
        ICMPv6_code      # ICMPv6 code.
        MPLS_label       # MPLS label.
        MPLS_tc          # MPLS TC.
```

En RAUFlow, cuando se crea un nuevo servicio se ejecutan los algoritmos de ruteo y distribución de etiquetas para construir al menos un LSP. Luego el mismo es traducido a flujos OpenFlow que luego son instalados en cada uno de los nodos en el mismo.

En la siguiente sección se detalla el algoritmo de ruteo utilizado en RAUFlow.

5.5.3 Algoritmo de ruteo

El algoritmo de ruteo implementado parte de un algoritmo Shortest Path First (SPF) centralizado para el cálculo del mejor camino entre un par de nodos en la topología. Luego incorporando restricciones al mismo es posible llegar a un Constrained Shortest Path First (CSPF) con el cual se podría implementar funcionalidades de QoS.

En el desarrollo de este proyecto, por razones de tiempo solo fué posible la implementación del algoríitmo SPF centralizado. De esta forma la implementación de un algoritmo de ruteo CSPF se identifica como una posible línea de trabajo a futuro. A continuación se explica la implementación del algoritmo SPF.

Shortest Path First

El algoritmo SPF centralizado está basado en el algoritmo Dijkstra. Este algoritmo permite calcular en forma eficiente el mejor camino entre un par de nodos en un grafo ponderado sin costos negativos.

Puesto que en este trabajo se representa a una topología de red mediante un multigrafo dirigido, es necesario o bien extender el algoritmo Dijkstra a esta representación o bien cambiar de representación. Como los multigrafos dirigidos ofrecen un modelado de la realidad intuitivo y directo, se opta por la alternativa de extender el algoritmo Dijkstra. Cabe destacar la existencia de trabajos previos en el desarrollo de una posible extensión a este algoritmo para multigrafos; en este trabajo se sigue la propuesta de [54].

En 5.4 se muestra el pseudo-código del algoritmo SPF centralizado para multigrafos dirigidos obtenido a partir de la extensión del algoritmo Dijkstra. Este recibe como parámetros la topología de red (G) y el par de nodos inicio y fin para el cual se quiere calcular el mejor camino posible. Las operaciones *ObtenerNodosMenorCostoLink* y *ObtenerNodoAdyacenteMenorCosto* son funciones auxiliares que se explican a continuación:

- ObtenerNodosMenorCostoLink (w, v): Permite obtener el link <w, v> de menor costo asociado en la topología entre los nodos w y v. Notar que para un par de nodos w, v pueden existir múltiples enlaces que los conecten, eventualmente con costos diferentes.
- ObtenerNodoAdyacenteMenorCosto ($G \setminus S$, D): Devuelve el nodo en $G \setminus S$ para el cual el costo de ir desde el nodo inicio a este es mínimo en la lista de costos D.

Function `CaminoMasCortoMultigrafo` *(G, inicio, fin)*
▷ Obtiene el mejor camino en G entre los nodos inicio y fin, utilizando como métrica el costos asociado a cada adyacencia
Pre-condiciones: inicio ∈ G y fin ∈ G

camino[] ← {}
D,P ← MultiDijkstra(G, inicio, fin) ▷ Utiliza algoritmo Dijkstra para obtener el mejor camino
i ← fin ▷ Arma el camino en función de la lista de predecesores
while *i* <> *inicio* **do**
  l ← P[i]
  camino ← camino ∪ l
  i ← l.origen
end
camino.Reverse() ▷ Hay que invertir el orden del camino
return *camino*

Fig. 5.4 Algoritmo de ruteo centralizado sin restricciones (SPF)

```
Function MultiDijkstra (G, inicio, fin)
    ▷ Extensión de algoritmo Dijkstra para cálculo de mejores caminos en un multigrafo dirigido ponderado

    D ← {}                                                    ▷ Distancias finales
    P ← {}                          ▷ Lista de links predecesores <nodo_origen, nodo_destino>
    S ← {}                                                    ▷ Lista de nodos procesados
    GmenosS ← G                                               ▷ Inicializa G \ S como G

    S ← S ∪ inicio                                            ▷ El algoritmo empieza con nodo inicio
    GmenosS ← GmenosS −{inicio}
    foreach i in G do
        if i in G[inicio] then
            ▷ Si los nodos son adyacentes actualiza el costo en D y el link predecesor l ← ObtenerNodosMenorCostoLink(inicio, i)▷ Obtiene el link de menor costo entre los nodos
            D[i] ← l.costo
            P[i] ← l
        else
            D[i] ← -1
            P[i] ← Null
        end
    end

    for i in (1 . . . |G|) do
        w = ObtenerNodoAdyacenteMenorCosto(GmenosS, D)▷ Obtiene el nodo adyacente de menor costo
        S ← S ∪ w
        GmenosS ← GmenosS −{w}
            ▷ Actualiza la lista de costos y predecesores para todos los nodos en G \ S
        foreach v in GmenosS do
            l ← ObtenerNodosMenorCostoLink(w, v)
            if l <> Null AND (D[v] = -1 OR D[w] + l.costo < D[v]) then
                D[v] ← D[w] + l.costo
                P[v] ← l
            end
        end
    end
    return (D, P)
```

Fig. 5.5 SPF Centralizado continuación

5.5.4 Algoritmo de distribución de etiquetas

MPLS utiliza etiquetas para: (a) distinguir el tráfico de una VPN particular y (b) encaminar tráfico en la red (reenvío en base a etiquetas). De esta forma se tienen dos niveles de etiquetas usualmente denominadas inner label (para marcar el tráfico) y outer label (para encaminar tráfico - LSP).

Para la distribución de etiquetas (inner y outer label) existen diferentes protocolos entre los cuales puede destacarse el protocolo LDP (Label Distribution Protocol) [52]. Este protocolo representa un buen modelo a seguir para el algoritmo de distribución de etiquetas, pero está definido en base a la visión topológica local de un nodo, mientras que en RAUFlow es necesario definir un algorítmo basado en una visión topológica global. Por ello es necesario implementar un algoritmo de distribución de etiquetas diferente.

A continuación se explican los algoritmos utilizados para la distribución de etiquetas de ambos niveles.

Etiquetas internas (Inner Labels)

Mapear etiquetas MPLS a servicios de VPN para posteriormente poder identificar el tráfico asociado dentro del prototipo es una tarea bastante simple. Una solución posible es asignar secuencialmente etiquetas dentro de un espacio de etiquetas disponible. Esta estrategia es simple y permite crear en el sistema tantos servicios como etiquetas disponibles. Puesto que una etiqueta MPLS se representa mediante 20bits $= 2^{20}$ posibles valores, descontando los valores reservados por el protocolo se pueden crear más que suficientes servicios utilizando esta estrategia.

No obstante, si bien esta estrategia es suficiente para la asignación de la etiqueta interna más adelante se verá que es necesario modificarla ligeramente para soportar ciertos casos de uso en el prototipo.

Etiquetas externas (Outer Label)

Asignar etiquetas a un camino MPLS es un problema más complejo. Aprovechando la visión global de la red puede redefinirse el algoritmo de distribución de etiquetas LDP de varias formas. Algunas de ellas pueden ser:

1. Asignar secuencialmente etiquetas a cada salto en un camino utilizando un espacio de etiquetas global para el LSP. De esta forma no existen dos LSPs en el sistema con etiquetas en común. Esta solución tiene como ventajas su simplicidad y bajo costo en el cómputo del algoritmo. Por otro lado tiene como desventaja que consume el espacio de etiquetas disponibles más rápido (hay que considerar la escala del prototipo).

2. Asignar secuencialmente etiquetas a cada salto en un camino utilizando un espacio de etiquetas local a cada nodo y descartando para cada interfaz etiquetas ya utilizadas por otros LSPs. Esta estrategia tiene como ventaja que es más austera en el consumo de etiquetas y no existen dos LSPs con iguales etiquetas para una misma interfaz de entrada en un mismo nodo.

Nótese en relación a la segunda alternativa, que debido a la forma en que se distribuyen etiquetas puede determinarse en un nodo a que servicio corresponde un paquete basándose solamente en la etiqueta externa y la interfaz por la que entra. Esto permite prescindir de la etiqueta interna para determinar lo que en un esquema tradicional de MPLS se denomina FEC y aplicar diferentes políticas de ingeniería de tráfico. Esto a su vez repercute en un aumento del MTU del paquete.

En RauFlow se implementa la segunda alternativa mencionada. En 5.6 se muestra el pseudocódigo de una posible implementación de dicho algorítmo:

```
Function obtenerEtiquetasLSP (path)
        ▷ Recibe una camino en la topología representado como una lista ordenada de links y
    devuelve una lista ordenada de etiquetas donde cada etiqueta se corresponde a un link

    MPLS_LABEL_SPACE_MIN ← 10
    mplsPath[] ← {}
    labelBase ← Null
    if len(path) = 1 then
        mplsPath ← mplsPath ∪ {Null} ▷ Si el camino es de largo 1, no tiene etiquetas porque
        primer nodo hace PHP
    else
        labelBase ← MPLS_LABEL_SPACE_MIN ▷ Se asignan etiquetas secuencialmente en
        el espacio de etiquetas
        foreach l in path-1 do
            label ← obtenerEtiquetaParaInterfaz (l, labelBase)   ▷ Obtiene una etiqueta
            libre del espacio de etiquetas local a la interfaz
            mplsPath ← mplsPath ∪ {label}
            labelBase ← labelBase + 1
        end
        mplsPath ← mplsPath ∪ {Null}  ▷ Como último nodo implementa PHP el último link
        no tiene etiqueta asociada
    end
    return mplsPath[]
```

Fig. 5.6 Algoritmo de distribución de etiquetas

```
Function obtenerEtiquetaParaInterfaz (link, label)
    ▷ Devuelve una etiqueta dentro del espacio de etiquetas global que no este en uso por la
  interfaz destino en el link. Si no existe etiqueta disponible devuelve Null

  etiquetas_usadas ← link.destino.etiquetas_usadas
  next_label ← label_base
  label ← Null
  while label == Null AND next_label < MPLS_LABEL_SPACE_MAX do
      if (next_label in etiquetas_usadas) then
          next_label ← next_label + 1
      else
          label ← next_label
      end
  end
  if (label <> Null) then
      link.destino.etiquetas_usadas ← link.destino.etiquetas_usadas ∪ label
  end
  return label
```

Fig. 5.7 Algoritmo de distribución de etiquetas - continuación

Stack de etiquetas MPLS

Se le denomina stack de etiquetas MPLS a la pila de etiquetas que resulta de superponer una etiqueta sobre otra en un paquete. Al inicio de esta sección se menciona la utilización de dos niveles de etiquetas: una etiqueta interna para identificar el servicio y una etiqueta externa para marcar el camino dentro de la red del prototipo. Luego cuando se describe la implementación del algoritmo de distribución de etiquetas de segundo nivel (etiquetas externas), se muestra que es posible prescindir del primer nivel de etiquetas (etiquetas internas).

Por un lado puede implementarse el prototipo trabajando solamente con un nivel de etiquetas, garantizando reenvío en base a etiquetas y clasificación de tráfico. Sin embargo para poder identificar el servicio asociado a un paquete en el nodo de egreso en un LSP, es necesario que el paquete presente la etiqueta correspondiente, permitiendo de esta forma reenviarlo por la correcta interfaz de salida del prototipo. Esto no es posible si se implementa Penultimate Hop Popping (PHP) dado que la etiqueta es removida en el penúltimo hop, llegando al nodo de egreso sin etiquetas.

Pensando en permitir en un futuro la conexión de nodos al prototipo implementados en base a diferentes tecnologías; es decir, nodos no implementados en base a RAU-Switch, es necesario mantener la implementación estándar del protocolo MPLS. Esto quiere decir que se debe implementar PHP. Por ello en el prototipo se trabaja con dos niveles de etiquetas.

Anteriormente en esta sección se mencionó la necesidad de modificar el algoritmo de distribución de etiquetas de primer nivel con el objetivo de poder implementar ciertos casos de uso. A continuación se explica el escenario que origina este problema y su solución.

En una red MPLS cada vez que un paquete es manipulado para colocar un cabezal de dicho protocolo, el Ethertype del paquete original es sustituido por el del protocolo. A su vez en el nodo de egreso el ethertype original del paquete debe ser colocado nuevamente. Este problema puede resolverse en una VPN de capa 3 obligando a indicar el ethertype de cada servicio definido. De esta forma se utiliza la información del servicio para definir un flujo que coloque el ethertype apropiado en el nodo de egreso, para cada paquete. De esta forma no se deben realizar modificaciones sustanciales a la implementación del prototipo. No obstante para una VPN de capa 2 no se puede implementar una solución similar, siendo necesario guardar el valor original de ethertype para cada paquete que ingresa y volver a colocarlo en el último nodo del camino, si se quiere soportar la implementación de VPNs de capa 2. Esto puede realizarse al menos de las siguientes dos formas:

1. Cada paquete que ingresa a la red del prototipo es enviado al Controlador en donde se extrae y guarda el valor original de ethertype. Luego se continúa con el procesamiento del paquete acorde a las reglas ya definidas. Luego cuando el paquete arriba al último nodo, tras procesarse acorde a las reglas definidas (extraer etiqueta interna por ejemplo) el paquete es enviado al Controlador donde se le coloca el valor original de ethertype. Finalmente se reenvía el paquete por la interfaz de salida de la red.

   Esta solución se corresponde con un enfoque reactivo en una red SDN (recordar estado del arte en las redes definidas por software).

2. Se utilizan etiquetas para indicar el ethertype de un paquete. Se puede definir un tercer nivel de etiquetas para esto, definiendo un mapeo entre ethertype y etiqueta. Luego el nodo de egreso utilizando esta información puede localmente mapear etiquetas a ethertypes colocando en el paquete el valor original antes de reenviarlo por la interfaz de salida. Sin embargo utilizar un tercer nivel de etiqueta agregaría complejidad al procesamiento del paquete, pudiendo perjudicar la performance del prototipo, además de que reduce el MTU de un paquete.

   Una alternativa es aprovechar el primer nivel de etiquetas, el cual se utiliza para identificar el servicio asociado a un paquete, combinando dicha información con el valor original de ethertype del paquete. Para esto se define para cada servicio un rango de etiquetas MPLS en lugar de una única etiqueta, mapeando cada valor dentro del rango de etiquetas de un servicio a un valor diferente de ethertype. De esta forma se definen flujos en el nodo de egreso para un servicio y cada valor de ethertype posible, con las acciones OpenFlow necesarias para colocar nuevamente el valor original de ethertype dependiendo del valor de etiqueta de primer nivel que traiga un paquete.

   En RAUFlow se implementa esta estrategia en la implementación de servicios de redes privadas de capa 2.

5.5.5 Actualización de la topología

Una vez que el módulo LSDB Sync envía a la aplicación la información topológica actualizada, se desencadena una secuencia acciones para la actualización de la información almacenada localmente (información topológica y de servicios) en el controlador. Este proceso puede resumirse de la siguiente forma:

1. Se ejecuta el algoritmo **ActualizarTopologia** el cual mezcla la información de la topología mantenida en memoria (información desactualizada) con la información recibida (información actual). Se actualiza cada instancia de Nodo, Interfaz y Link considerando los casos en que se deben crear nuevos objetos, actualizar existentes y eventualmente eliminarlos. Cabe destacar que la operación eliminar tanto para Nodos, Interfaces como Links no efectúa un borrado físico de estos objetos, simplemente los marca como no operativos (atributo "state" en la clase).

2. Se ejecuta el algoritmo **ActualizarServiciosLSPs** el cual recalcula para cada servicio en el sistema los LSPs asociados. El proceso de actualización de un LSP implica:

   (a) Recalcular el mejor camino entre los nodos origen y destino del servicio

   (b) Mapear etiquetas al nuevo "mejor camino" conservando las mismas etiquetas que tenía el camino viejo para los links que se mantienen en el camino nuevo

   (c) Actualizar las tablas MPLS en cada nodo de la topología

   (d) Actualizar las tablas de flujos en cada nodo de la la topología

De ambos algoritmos se elige **ActualizarServiciosLSPs** para mostrar a continuación su funcionamiento a través de su especificación en pseudo-código.

```
Declare topología                                   ▷ Representación de la topología
Declare servicios                                   ▷ Servicios del sistema
Function ActualizarServiciosLSPs ()
    ▷ Actualiza para cada servicio en el sistema la lista de LSPs asociados. Luego se actualizan
    todas las tablas de flujos en la topología, eliminando entradas viejas y agregando nuevas
    mpls_tables_ftn ← {}
    mpls_tables_ilm ← {}
    ▷ Copia el contenido de las tablas MPLS de cada nodo y luego elimina el contenido de cada
    una de estas tablas
    foreach n in topologia do
        mpls_tables_ilm[n.router_id] ← n.ilm
        n.ilm ← []
        mpls_tables_ftn[n.router_id] ← n.ftn
        n.ftn ← []
        n.nhlfe ← []
    end
                                        ▷ Para cada servicio en el sistema recalcula los LSPs
    foreach s in servicios do
        lsps_nuevos ← []
        foreach lsp in s.lsps do
            lsp_nuevo ← ActualizarLSP(s, lsp)
            lsps_nuevos ← lsps_nuevos ∪ lsp_nuevo
        end
        s.lsps ← lsps_nuevos
    end
    ▷ Actualiza las tablas de flujos de cada Nodo en la topología, a partir de las entradas FTN e
    ILM que deben ser eliminadas y agregadas
    foreach n in topologia do
        ftn_adds ← { n.ftn } - { mpls_tables_ftn[n.router_id]}
        ftn_removes ← { mpls_tables_ftn[n.router_id]} - { n.ftn }
        ilm_adds ← { n.ilm } - { mpls_tables_ilm[n.router_id] }
        ilm_removes ← { mpls_tables_ilm[n.router_id]} - { n.ilm }
        ActualizarDatapath(n, ftn_adds, ftn_removes, ilm_adds, ilm_removes)
    end
```

Fig. 5.8 Algoritmo de actualización topológica

```
Function ActualizarLSP (servicio, lsp)
    ▷ Precondición largo camino mayor a 0
    etiqueta_base ← MPLS_LABEL_SPACE_MIN
    etiquetas[] ← {}
    nuevo_lsp_camino[] ← {}
    viejo_lsp_camino[] ← lsp.path

    camino ← CaminoMasCortoMultigrafo(topologia, servicio.NodoIng, servicio.NodoEgr)
    ultimo ← path.SacarUltimo()          ▷ El último elemento de la lista se procesa aparte
    foreach l in camino do
        pe ← ObtenerPeDeLink(viejo_lsp_camino, l)▷ Obtiene entrada del LSP asociada a un
        Link particular
        if pe == Null then
            etiqueta ← ObtenerEtiquetaParaInterfaz(l, etiqueta_base)
            pe ← <etiqueta, l>
            nuevo_lsp_camino ← nuevo_lsp_camino ∪ pe
            etiquetas ← etiquetas ∪ etiqueta
        else
            nuevo_lsp_camino ← nuevo_lsp_camino ∪ pe
            viejo_lsp_camino ← viejo_lsp_camino - {pe}
            etiquetas ← etiquetas ∪ pe.etiqueta
        end
    end
    pe ← ObtenerPeDeLink(viejo_lsp_camino, ultimo)
    if pe == Null then
        pe ← <Null, ultimo>
        nuevo_lsp_camino ← nuevo_lsp_camino ∪ pe
        etiquetas ← etiquetas ∪ Null
    else
        nuevo_lsp_camino ← nuevo_lsp_camino ∪ pe
        viejo_lsp_camino ← viejo_lsp_camino - {pe}
        etiquetas ← etiquetas ∪ pe.etiqueta
    end

    ActualizarTablasMPLS(servicio, camino, etiquetas)    ▷ Actualiza tablas MPLS para cada
    nodo en el camino

                                                ▷ Libera etiquetas que ya no se usen
    foreach pe in viejo_lsp_camino do
        if pe.etiqueta <> Null then
            pe.link.destino.SacarEtiquetasEnUso(pe.etiqueta)
        end
    end
```

Fig. 5.9 Algoritmo de actualización de la topología - continuación

```
Function ActualizarDatapath (n, ftn_adds, ftn_removes, ilm_adds, ilm_remmoves)
        ▷ Agrega flujos para cada entrada nueva de la tabla FTN e ILM y elimina los flujos
    asociados a las entradas viejas de estas tablas
    foreach ftn in ftn_removes do
        service ← ftn.service
        remove_egress_node_service_flows(service, n, ftn)
    end
    foreach ilm in ilm_removes do
        foreach nhlfe in ilm.nhlfes do
            remove_node_service_flow(n, ilm, nhlfe)
        end
    end
    foreach ftn in ftn_adds do
        service ← ftn.service install_ingress_node_service_flows(service, n, ftn)
    end
    foreach ilm in ilm_adds do
        foreach nhlfe in ilm.nhlfes do
            if next_hop.i_type == 1 then
                ▷ Si el tipo de interfaz es 1 entonces es un nodo de borde y el flujo OpenFlow es
                diferente
                install_egress_node_flow_for_service(service, n, ilm, nhlfe)
            else
                                                ▷ Es un nodo interno entonces el flujo es normal
                install_node_flow_for_service(n, ilm, nhlfe)
            end
        end
    end
```

Fig. 5.10 Algoritmo de actualización de la topología - continuación

5.5.6 Ciclo de vida de un Nodo

En el prototipo se obtiene información de varias fuentes de datos para la construcción de la topología como la base de datos topológica de Quagga (LSDB), el datapath OpenFlow y a través del ingreso manual de información mediante la interfaz gráfica de RAUFlow.

Estas fuentes aportan datos en forma independiente, lo cual incide directamente en la construcción de la topología y en decisiones que se toman dentro del prototipo como el algoritmo de ruteo entre otros. Por ello a continuación se muestra el proceso de construcción de la topología a partir de las fuentes de datos, tomando como ejemplo el ciclo de vida de un nodo en particular. Luego el ciclo de vida de interfaces y enlaces puede deducirse de forma análoga.

Como se muestra en la figura5.11, descartando el pseudo-estado inicial se identifican cinco estados diferentes para un Nodo: *Descubrimiento OpenFlow*, *Sin Conexión OpenFlow*, *Operativo*, *LSDB*

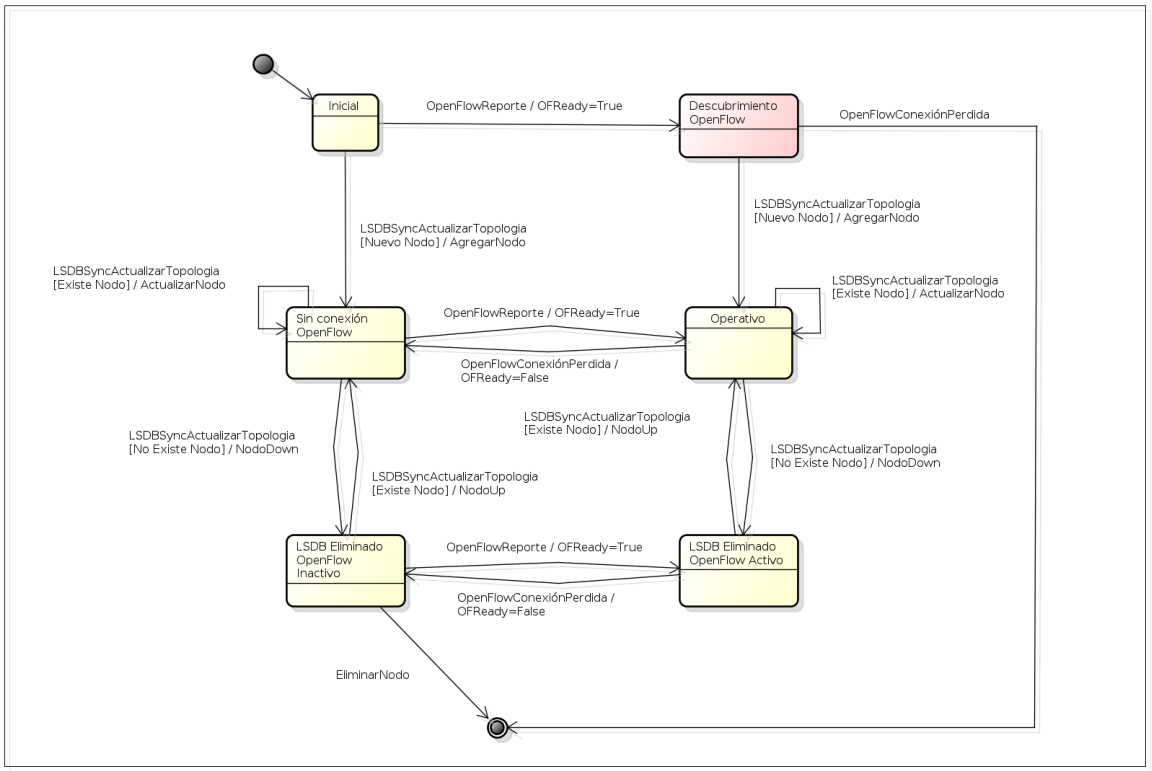

Fig. 5.11 Ciclo de vida de un Nodo

Eliminado OpenFlow Inactivo y *LSDB Eliminado OpenFlow Activo*. El significado de cada uno de estos así como las transiciones posibles se explican a continuación:

- *Descubrimiento OpenFlow:* Cuando un switch se reporta mediante el canal OpenFlow por primera vez con RAUFlow, se guarda información del mismo entre la cual se encuentra la instancia de la clase Datapath en la implementación de Ryu (esta instancia es utilizada posteriormente para interactuar con el dispositivo). Si el switch OpenFlow no se encuentra instanciado en el Sistema como Nodo, la información recibida se almacena en una lista auxiliar de nodos. Como decisión de diseño no se crea una instancia de Nodo como tal aún, instanciándose solamente cuando se advierte la presencia de éste en la Link State Database (LSDB).

  De este estado existen dos transiciones posibles. En primer lugar el switch puede dejar de reportarse con el controlador en cuyo caso es eliminado de la lista temporal de nodos y en segundo lugar la componente LSDBSync puede enviar una actualización de la topología, en donde se incluye el nodo en cuestión. En este caso se instancia un objeto Nodo con toda la información provista por la LSDB, datapath de OpenFlow y la información adicional obtenida con el agente SNMP. Luego el Nodo cambia al estado *Operativo*.

Vale la pena destacar que en caso de no poder obtener la información adicional con el agente SNMP, el Nodo no es instanciado y se mantiene en el estado actual.

- *Operativo:*

  Este estado es utilizado para representar a aquellos nodos que por un lado se encuentran en la LSDB y fueron rectificados o instanciados en la última actualización de la topología, y por otro se encuentran reportándose mediante el protocolo OpenFlow.

  En este estado el nodo es mostrado por la interfaz gráfica de RAUFlow y se tiene acceso a todas las características del mismo, tanto del modelo de datos como las que son accesibles a través del datapath OpenFlow. Además es considerado para el cálculo de las mejores rutas y puede ser utilizado como nodo de ingreso o egreso en la definición de servicios.

  En pocas palabras es un nodo completamente funcional a los efectos de la implementación de RAUFlow.

  Desde este estado se tienen tres transiciones posibles. Una primera posibilidad es que se produzca una actualización de la LSDB (evento LSDBSyncActualizarTopologia) y que en la topología recibida se encuentre el Nodo en cuestión. En este caso solamente se actualiza la información asociada al nodo (acción ActualizarNodo), entre la cual se incluye la actualización de las Interfaces y Links, quedando en el estado actual. Una segunda posibilidad es que el switch asociado deje de reportarse por el canal OpenFlow (evento OpenFlowConexiónPerdida) en cuyo caso se modifica el atributo **of_ready** del Nodo asignando el valor False y se cambia al estado *Sin Conexión OpenFlow*. Por último una tercera posibilidad es que se produzca una actualización de la LSDB (evento LSDBSyncActualizarTopologia) y que el Nodo no este en la topología recibida. En este caso se deshabilita el Nodo (NodoDOWN) asignando el valor DOWN al atributo **state** y se cambia al estado *LSDB Eliminado OpenFlow Activo*.

- *LSDB Eliminado OpenFlow Activo:* Este estado contempla a un Nodo que no se encontraba en la LSDB recibida en la última actualización topológica pero que aún se reportan por el canal de comunicación OpenFlow.

  Desde este estado se tienen solamente dos transiciones posibles. Una primera posibilidad es que se produzca nuevamente una actualización de la topología (LSDBSyncActualizarTopologia) y que el nodo se encuentre dentro de la información recibida. En este caso se actualiza el atributo **state** del nodo con el valor UP y se cambia al estado *Operativo*. Una segunda posibilidad es que el nodo deje de reportarse por el canal OpenFlow (OpenFlowConexiónPerdida) en cuyo caso se actualiza el valor del atributo **of_ready** al valor False y se cambia al estado *LSDB Eliminado OpenFlow Inactivo*.

- *Sin Conexión OpenFlow:* Este estado contempla a un Nodo que fue creado o rectificado en la última actualización de la LSDB pero que no se ha reportado por el canal OpenFlow, ya sea porque nunca lo hizo o porque dejó de hacerlo. En este estado el Nodo es mostrado a través

de la interfaz gráfica de RAUFlow pero no es considerado ni para el el cálculo de mejores caminos ni para la creación de nuevos servicios como nodo de ingreso o egreso. Además la información asociada al datapath de OpenFlow como las tabla de flujos y estadísticas tampoco son accesibles.

De este estado existen tres transiciones posibles. Por un lado el nodo puede estar presente en la LSDB al producirse una actualización de la topología (LSDBSyncActualizarTopologia), actualizándose la información del nodo (ActualizarNodo) y manteniéndose en el mismo estado. Por otro lado también al producirse una actualización de la topología, el nodo puede no estar en la LSDB, en cuyo caso se actualiza el atributo **state** al valor DOWN y se cambia al estado *LSDB Eliminado OpenFlow Inactivo*. Finalmente el switch puede empezar a reportarse por el canal OpenFlow (OpenFlowReport) en cuyo caso se actualiza el valor del atributo **of_ready** a True y se cambia al estado *Operativo*.

- *LSDB Eliminado OpenFlow Inactivo:* Este estado modela a un Nodo que nunca se reportó o dejó de reportarse por el canal OpenFlow, y que a su vez no se encuentra dentro de la LSDB en la última actualización topológica realizada. De este estado existen tres transiciones posibles.

  Una primera alternativa es que el switch en cuestión se reporte con la aplicación (OpenFlowReporte), en cuyo caso se actualiza el valor del atributo **of_ready** a True y se cambia el estado a *LSDB Eliminado OpenFlow Activo*. Una segunda alternativa es que se produzca una actualización de la topología y el Nodo se encuentre dentro de la LSDB, en cuyo caso se cambia el valor del atributo **state** a UP y se cambia al estado *Sin conexión OpenFlow*. Finalmente una tercera alternativa es que el nodo sea eliminado del sistema. Si bien esta funcionalidad no se encuentra actualmente implementada, la idea es que un Nodo para el cual no se ha recibido reportes por el canal OpenFlow luego de un tiempo determinado, y que tampoco aparece en la LSDB tras una actualización, pueda ser eliminado del sistema, eventualmente tras la intervención de un usuario administrador.

El ciclo de vida de una Interfaz o un Link puede ser explicado de forma análoga con una máquina de estados similar, teniendo presente las relaciones existentes entre las tres clases, Nodos, Interfaces y Links.

5.5.7 QoS en RAUFlow

Por razones de tiempo no se implementaron en RAUFlow funcionalidades de QoS como balanceo de carga y asignación de prioridades por tipo de tráfico. Sin embargo, a continuación se describen las alternativas a explorar en caso de seguir con el desarrollo de esta aplicación.

Dentro de las funcionalidades de calidad de servicio (QoS) que se podrían implementar en RAUFlow interesan dos principalmente: (1) la capacidad para construir caminos alternativos para un servicio y de esta forma hacer balanceo de carga entre todos los LSPs construídos y (2) asignar prioridades por servicio para el procesamiento de tráfico en cada uno de los nodos de la red.

Para la construcción de caminos alternativos para un mismo servicio existen dos soluciones: (1) una solución simple la cual permite a un administrador definir manualmente caminos y (2) una solución más compleja en la que se mejora el algoritmo de ruteo centralizado para implementar un CSPF que permita calcular diferentes caminos para un mismo par nodo-interfaz de ingreso y egreso, en función del ajuste de diferentes parámetros de QoS (ancho de banda por ejemplo).

Para implementar (1) en el prototipo es necesario modificar levemente la implementación de la clase Servicio para distinguir entre los servicios en que el LSP se calcula con el algoritmo de ruteo y los que se definen de forma manual, a la vez que es necesario extender la interfaz gráfica para implementar este nuevo caso de uso.

Para implementar (2) se requiere de un tiempo de desarrollo considerablemente mayor que en la alternativa (1) puesto que es necesario modificar el algoritmo de ruteo e implementar un CSPF.

Asumiendo que se soportan múltiples caminos para un mismo servicio, puede implementarse balanceo de carga de dos formas: (1) "partir" la definición de un servicio de VPN en varios subservicios diferenciando por el tipo de tráfico mediante el cabezal OpenFlow y así hacer balanceo de carga por ejemplo por tipo de aplicación, (2) implementar balanceo de carga basándose en las estadísticas de los flujos OpenFlow y así balancear dinámicamente el tráfico de un servicio de VPN entre diferentes caminos.

Para implementar (1) solo es necesario que RAUFlow implemente alguno de los mecanismos mencionados para la definición de múltiples caminos para un servicio.

Por otro lado para implementar (2) se puede utilizar el concepto de "Meters" o métricas de OpenFlow que permiten analizar la cantidad de paquetes procesados por un mismo flujo y en función de esto aplicar una acción o conjunto de acciones. Sin embargo Open vSwitch no implementa esta característica del protocolo, por lo que no es posible implementarlo en RAUFlow; de hecho Open vSwitch ofrece otras funcionalidades de QoS que delega al kernel de Linux en vez de implementar los mecanismos contemplados por el protocolo OpenFlow.

Dentro de las funcionalidades de QoS que implementa Open vSwitch se encuentran la definición de "colas" a las que se les puede definir tasas de transferencia mínima y máxima. Mediante esta funcionalidad es posible implementar prioridades por tipo de tráfico y asi por servicio, aunque como es una implementación propia de la herramienta no puede ser manipulada dinámicamente desde el controlador mediante el protocolo OpenFlow.

De esta forma queda explicada la implementación de la aplicación de gestión RAUFlow, el diseño de los nodos que formarán parte de la red prototipo (RAU-Switch) y el funcionamiento general del prototipo en su conjunto. El siguiente capítulo está destinado al diseño y construcción de un laboratorio de pruebas, para la validación de las componentes desarrolladas y la implementación de algunos casos de uso representativos.

Capítulo 6

Laboratorio de pruebas

Para validar funcionalmente el prototipo es preciso la construcción de un laboratorio de experimentación sobre el cual ejecutar un conjunto de pruebas e implementar algunos casos de uso representativos. Dentro de las pruebas funcionales se busca verificar aspectos importantes de la implementación como la clasificación de tráfico, el algoritmo de ruteo dinámico y el algoritmo de distribución de etiquetas. Por otro lado con la implementación de algunos casos de uso representativos se busca validar la utilización del enfoque OpenFlow/SDN en la construcción de la RAU2.

El siguiente capítulo está destinado a la presentación del laboratorio de pruebas construido y a la descripción de las diferentes pruebas realizadas.

6.1 Definición del Laboratorio

Como se menciona anteriormente uno de los objetivos de este laboratorio de experimentación es verificar el correcto funcionamiento de la implementación realizada, en particular en los aspectos críticos de la misma como la capacidad para clasificar tráfico y los diferentes algoritmos de ruteo, distribución de etiquetas, actualización topológica.

Para esto es necesario construir un prototipo con suficientes nodos y enlaces como para poder definir varios servicios de redes privadas. También interesa la existencia de caminos alternativos entre un par de nodos origen y destino para poder comprobar el funcionamiento del algoritmo de ruteo y evaluar el comportamiento del prototipo ante cambios en la topología (desconexión de nodos o fallas en enlaces por mencionar algunos ejemplos).

Teniendo en cuenta los requerimientos mencionados se construye un laboratorio de pruebas con la siguiente topología (ver figura 6.1).

El laboratorio se compone de cuatro nodos implementados en base al dispositivo RAU-Switch conectados entre si con enlaces de fibra óptica multimodo. Cada nodo está conectado a los otros tres nodos de la topología implementando de esta forma una topología full mesh de cuatro nodos. A su vez cada nodo está conectado al controlador SDN del prototipo mediante una interfaz de red ethernet (interfaces **eth0**).

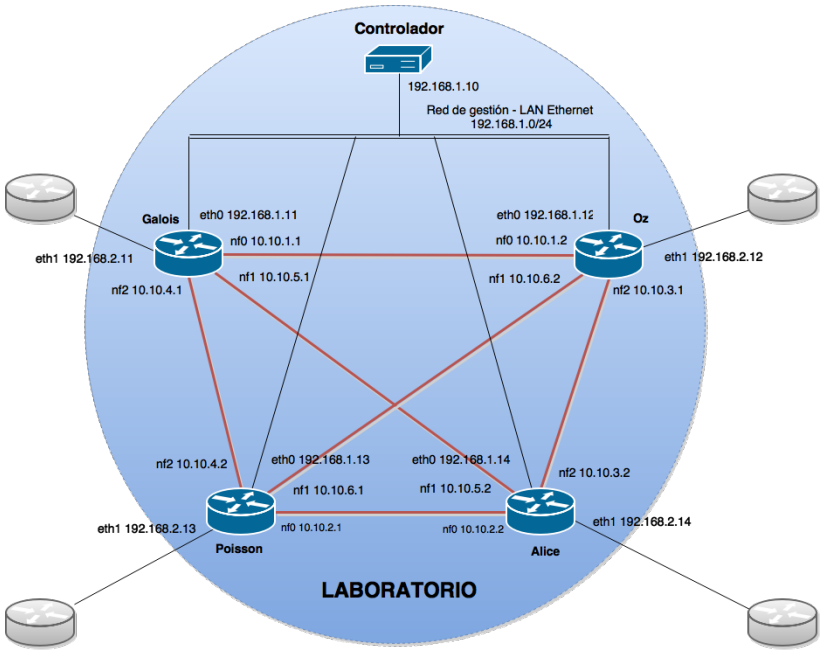

Fig. 6.1 Laboratorio de pruebas - Topología

Por la forma que tiene la topología, los cuatro nodos nombrados *Galois*, *Poisson*, *Oz* y *Alice* son nodos de borde. Esto quiere decir que RAUFlow los considera como nodos habilitados para ser nodo de ingreso o egreso en la definición de servicios de redes privadas.

Por otro lado cada nodo cuenta con una interfaz de red de ethernet adicional (interfaz **eth1**) utilizada como interfaz externa para la conexión con otras subredes. Estas interfaces son utilizadas por RAUFlow para la definición de servicios de VPN como los puntos de entrada y salida de tráfico. Por tanto cada una de estas interfaces se encuentra directamente conectada a la subred de una VPN en particular.

Como se menciona anteriormente en el capítulo 5, la representación utilizada para modelar una topología de red es la de un multigrafo dirigido ponderado. En la figura 6.2 se muestra esta representación para la topología del prototipo. Como puede apreciarse en la imagen cada enlace tiene un costo asociado. Este costo se define mediante el archivo de configuración del demonio Unix OSPF en cada nodo de la red y es utilizado por el algoritmo SPF centralizado para el cálculo de las mejores rutas.

Por simplicidad en el diagrama se han obviado los enlaces existentes entre el Controlador y cada uno de los nodos. Como se menciona en el capítulo 4 la instancia de Quagga ejecutada en el controlador tiene como único objetivo contar con un acceso local a la LSDB. Por ello conceptualmente el costo asociado a cada uno de estos enlaces es infinito, lo cual en la práctica se realiza asignando el valor 65535.

Sobre este laboratorio se implementan dos casos de uso representativos: (a) VPN de capa 3 y (b) VPN de capa 2. Sobre cada uno de estos casos de uso a su vez se ejecutan una serie de

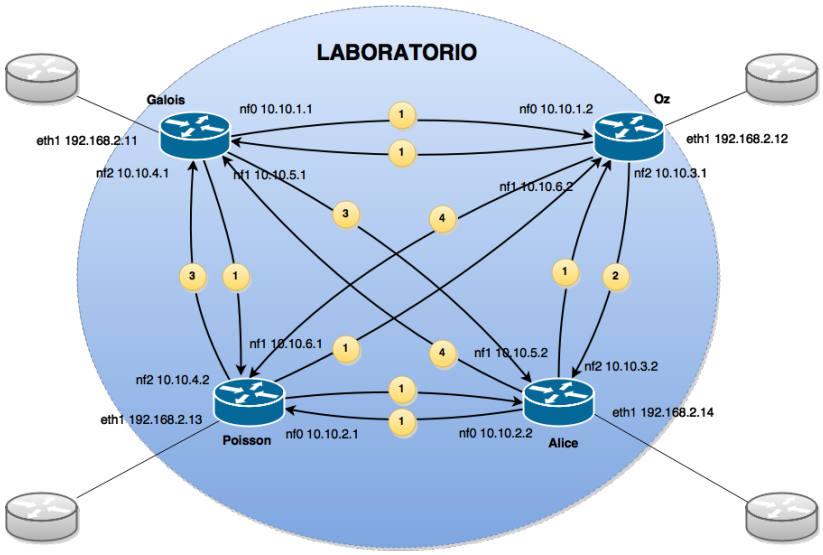

Fig. 6.2 Laboratorio de pruebas - Costos de enlaces en la topología

pruebas orientadas a verificar el correcto funcionamiento de las componentes más importantes en la implementación de RAUFlow y RAU-Switch.

A continuación se describen los resultados obtenidos en la implementación de estos casos de uso y la ejecución de estas pruebas, comenzando por el caso de uso VPN de capa 3.

6.2 VPN de capa 3

Las redes privadas virtuales de capa 3 son un tipo de servicio comúnmente brindado por un operador de red y es uno de los servicios que se quiere implementar en la RAU2. Sobre este tipo de redes privadas puede implementarse clasificación de tráfico por tipo de aplicación o numeración de capa 3 por citar algunos ejemplos.

En este trabajo se implementan dos escenarios diferentes para este tipo de red privada: (a) red privada multipunto con una única organización y tres sucursales físicamente separadas, (b) red privada punto a punto con dos organizaciones, cada una de ellas con dos sucursales físicamente separadas.

A continuación se detalla la construcción de ambos escenarios y los resultados obtenidos.

6.2.1 Escenario 1

Este escenario representa una red privada multipunto de capa 3. En el mismo se tiene una sola organización o red privada dividida en 3 sucursales o subredes físicamente separadas.

En el laboratorio cada una de estas subredes y de aquí en más cada subred cliente en los siguientes escenarios de pruebas son implementadas mediante el emulador Core[21]. Esta herramienta permite emular dispositivos de red como routers, switches y las conexiones entre ellos mediante maquinas virtuales Linux.

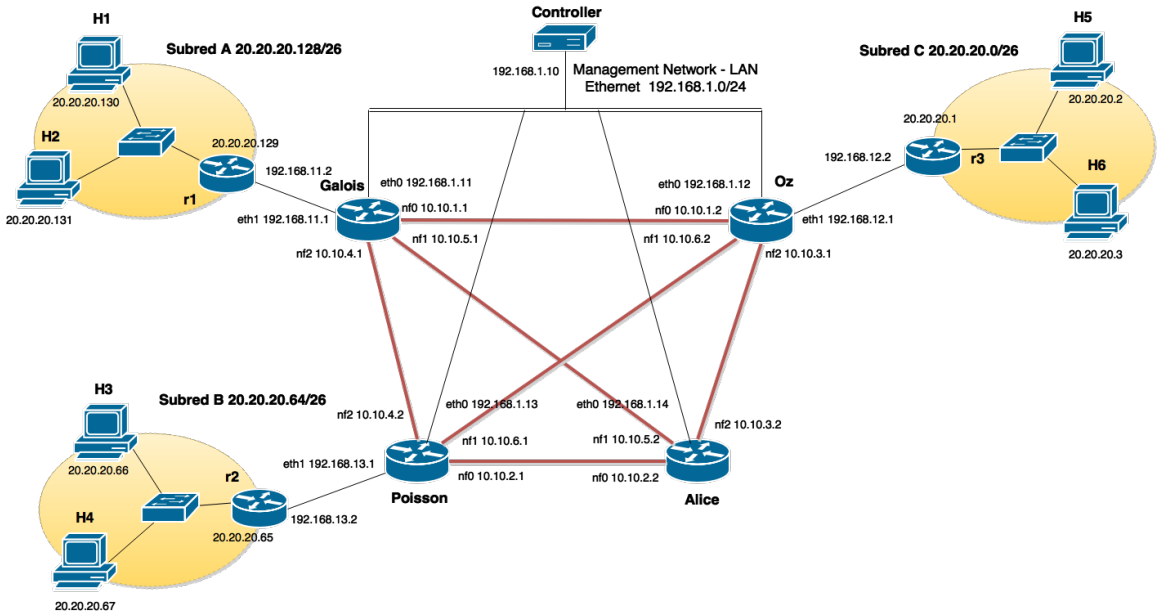

Fig. 6.3 VPN de capa 3 - Escenario 1

Sobre este escenario se ejecutan una serie de pruebas orientadas a verificar los siguientes aspectos relacionados al prototipo:

1. Algoritmo de ruteo
2. Algoritmo de distribución de etiquetas
3. Clasificación de tráfico
4. Actualización de rutas cuando cambia la topología
5. Capacidad para crear VPN multipunto

Para construir la red privada multipunto uniendo las tres subredes mencionadas y además verificar los puntos anteriores, se instancia los siguientes servicios (ver tabla 6.1). Por cada par de subredes se crean dos servicios (un servicio para cada sentido del tráfico).

Para cada uno de estos servicios además se indica el etherype 0x0800 correspondiente al tipo de tráfico IPv4.

Este conjunto de servicios establece el procesamiento de paquetes de capa 3 para las diferentes subredes de forma de conectarlas en una red privada de capa 3. A continuación se detallan las pruebas realizadas para cada uno de los aspectos mencionados anteriormente.

Nombre	Ingreso	Egreso	Clasificación	Descripción
S1	Galois - eth1	Oz - eth1	ip_src=20.20.20.128/26 ip_dst=20.20.20.0/26	Tráfico IP de Subred A a Subred C
S2	Oz - eth1	Galois - eth1	ip_src=20.20.20.0/26 ip_dst=20.20.20.128/26	Tráfico IP de Subred C a Subred A
S3	Galois - eth1	Poisson - eth1	ip_src=20.20.20.128/26 ip_dst=20.20.20.64/26	Tráfico IP de Subred A a Subred B
S4	Poisson - eth1	Galois - eth1	ip_src=20.20.20.64/26 ip_dst=20.20.20.128/26	Tráfico IP de Subred B a Subred A
S5	Poisson - eth1	Oz - eth1	ip_src=20.20.20.64/26 ip_dst=20.20.20.0/26	Tráfico IP de Subred B a Subred C
S6	Oz - eth1	Poisson - eth1	ip_src=20.20.20.0/26 ip_dst=20.20.20.64/26	Tráfico IP de Subred C a Subred B

Table 6.1 VPN capa 3 - Escenario 1, servicios creados

Verificación del Algoritmo de ruteo

En esta sección se evalúa el correcto funcionamiento del algoritmo de ruteo. Para ello se comparan los caminos computados por el algoritmo para cada servicio, con los respectivos mejores caminos teóricos (previamente calculados de forma manual).

Para cada camino se verifican dos cosas: (1) que el camino calculado se corresponde con el camino teórico y (2) que el camino es correctamente instalado en forma de reglas de reenvío (en base a conmutación de etiquetas) en las respectivas tablas de flujos OpenFlow de cada nodo del camino.

Como se ha mencionado, los caminos teóricos son calculados manualmente observando los costos de la topología (figura 6.2). Los resultados de este procedimiento se muestran en la figura 6.4, identificado cada camino mediante el código de colores utilizado en la tabla de servicios.

Para obtener los caminos calculados por el algoritmo de ruteo se analizan las tablas de flujos de Open vSwitch en cada uno de los nodos de la red prototipo utilizando el comando **dump-flows** de esta herramienta; de esta forma a partir de la información en las mismas se construye cada uno de los caminos. Cabe destacar que también puede obtenerse la información de las tablas de flujos mediante la interfaz gráfica de RAUFlow.

Aplicando este procedimiento para cada uno de los servicios en la tabla 6.1 se comprueba la correspondencia entre los caminos computados por el algoritmo de ruteo y los caminos teóricos, a la vez que se comprueba que son correctamente instalados como flujos en las tablas de cada nodo en el prototipo. En el apéndice F.1 se incluye a modo de ejemplo el análisis realizado para los LSPs asociados a los servicios S3 y S4.

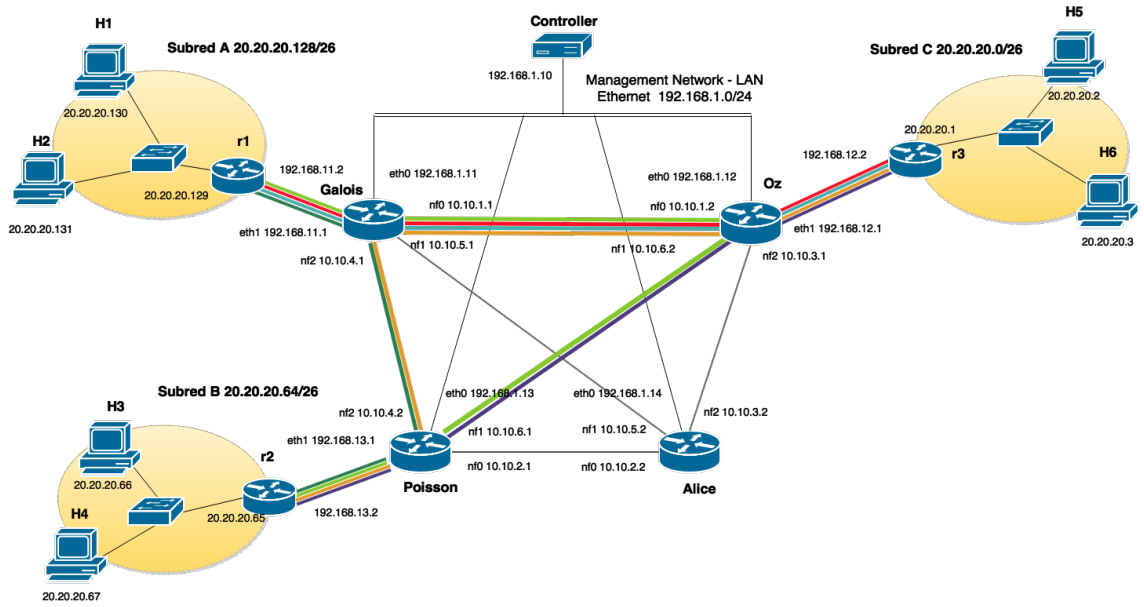

Fig. 6.4 Escenario 1 - LSPs

Clasificación de tráfico

Para cada servicio se implementa clasificación de tráfico en el prototipo solamente en el nodo ingreso, implementándose para el escenario definido en base a la numeración IP origen y destino de un paquete. Luego en cada nodo intermedio los paquetes son procesados acorde a las reglas de reenvío basadas en la conmutación de etiquetas MPLS.

Para verificar que realmente se implementa clasificación de tráfico se observa el comportamiento de cada flujo OpenFlow involucrado en el procesamiento del tráfico asociado a un servicio (flujos que clasifican tráfico en el nodo de ingreso y flujos que construyen un LSP). Para ello se genera tráfico para los servicios definidos (ver tabla 6.1) y mediante la herramienta de Linux **tcpdump** se captura tráfico por las diferentes interfaces en cada nodo involucrado en los LSPs correspondientes.

Mediante este procedimiento se genera tráfico para todos los servicios definidos, constatando que el mismo es correctamente enrutado a través de la red laboratorio mediante flujos OpenFlow y siendo entregado a la respectiva subred destino. Por simplicidad los detalles técnicos de este procedimiento se han omitido de este capítulo, aunque si el lector desea profundizar en ellos se incluye en el apéndice F.2 una descripción de este procedimiento y los resultados obtenidos para los servicios S1 y S3.

En la siguiente sección se discute el comportamiento del prototipo, cuando la topología cambia por ejemplo por una falla técnica en un enlace, utilizando el escenario y los servicios definidos anteriormente.

Actualización de rutas

Para probar el correcto funcionamiento del algoritmo de ruteo en la actualización de rutas ante un cambio en la topología, se trabaja con los servicios previamente definidos en 6.1. Partiendo de este escenario, se simula un desperfecto técnico en los enlaces
$< (Galois, nf_2), (Poisson, nf_2) >$ y $< (Poisson, nf_2), (Poisson, nf_2) >$ (por ejemplo desconectando el cable que conecta ambos nodos), provocando de esta forma una actualización en la topología y la posterior ejecución de los algoritmos de ruteo y distribución de etiquetas para los nuevos LSPs.

Acorde a los costos de la topología (ver figura 6.2), en la nueva topología los caminos teóricos asociados a cada servicio quedan de la siguiente forma:

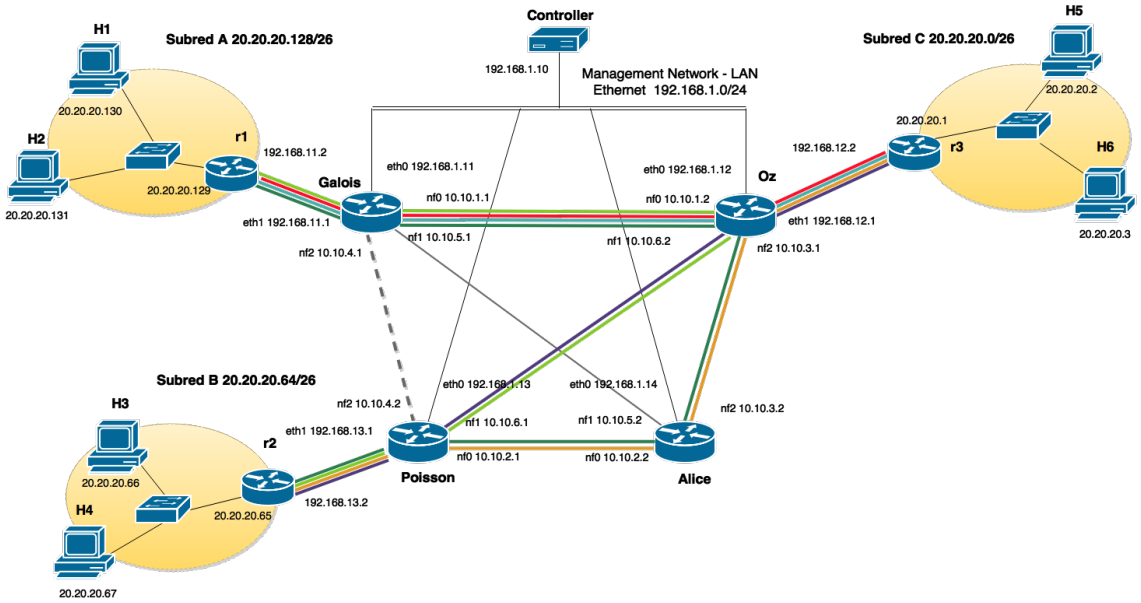

Fig. 6.5 Escenario 1 - Caminos para servicios recalculados

Nótese que los únicos caminos que cambian son los asociados a los servicios S3 y S6. Cabe destacar además que en la nueva topología existe más de un camino de mínimo costo para el servicio S3; los caminos $< (Galois, nf0), (Oz, nf1) >$ y
$< (Galois, nf0), (Oz, nf2), (Alice, nf0) >$ presentan ambos el costo 4. Por tanto se tienen dos resultados válidos posibles para la salida del algoritmo de ruteo para este servicio.

Analizando nuevamente las tablas de flujos de cada nodo en el laboratorio, se comparan los caminos computados por el algoritmo de ruteo con los caminos teóricos para cada servicio. Mediante este procedimiento se comprueba la correcta actualización de caminos LSPs tras un cambio en la topología. En el apéndice F.3 se muestran los detalles técnicos de este procedimiento para los LSPs asociados a los servicios S3 y S6.

De esta forma se verifica para el laboratorio de pruebas construido, la correctitud del algoritmo de ruteo, el algoritmo de distribución de etiquetas, se prueba el correcto funcionamiento de las tablas de flujos Open vSwitch y los flujos particulares utilizados para implementar clasificación de tráfico,

se comprueba el correcto accionar del programa ante cambios en la topología (recalculo de LSPs) y finalmente se valida la construcción de un servicio de red privada multipunto de capa 3.

En la siguiente seccción se discute la verificación de otros aspectos importantes en el funcionamiento del prototipo, utilizando como escenario de pruebas la construcción de dos servicios de red privada punto a punto de capa 3, utilizando dos organizaciones diferentes.

6.2.2 Escenario 2

Este escenario (ver figura 6.6) representa una red privada punto a punto de capa 3. Está compuesto por dos organizaciones diferentes, cada una de ellas con dos sucursales físicamente separadas. Además, ambas organizaciones utilizan el mismo espacio de direcciones IP para sus respectivas subredes.

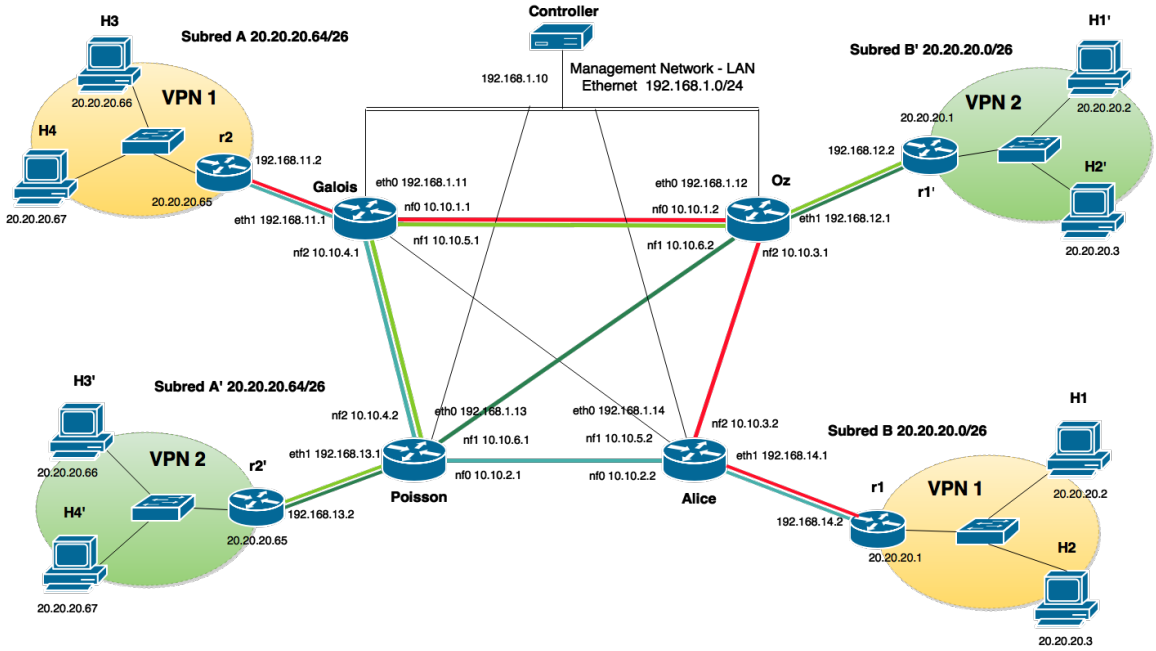

Fig. 6.6 VPN de capa 3 - Escenario 2

Este escenario se traduce en dos servicios de VPN punto a punto, aunque los aspectos de implementación que en esta sección se pretenden verificar son independientes de si el servicio de VPN es punto a punto o multipunto. Sobre este escenario se busca validar los siguientes aspectos en relación a la implementación:

1. El prototipo soporta la construcción de servicios de VPN de capa 3 para múltiples organizaciones (recordar que en el escenario anterior se trabaja con una única organización).

2. La implementación utilizada en el prototipo para transportar tráfico de una organización desde un nodo de ingreso a un nodo de egreso, es independiente de la numeración de capa 3 de dicho tráfico. En otras palabras soporta organizaciones con numeración IP solapadas.

Para la construcción de este escenario se instancia los siguientes servicios en el sistema (ver tabla 6.2). Por cada red privada se instancia dos servicios (uno para cada sentido del tráfico).

Nombre	Ingreso	Egreso	Clasificación	Descripción
S1	Galois - eth1	Alice - eth1	ip_src=20.20.20.64/26 ip_dst=20.20.20.0/26	Tráfico de Subred A a Subred B
S2	Alice - eth1	Galois - eth1	ip_src=20.20.20.0/26 ip_dst=20.20.20.64/26	Tráfico de Subred B a Subred A
S3	Poisson - eth1	Oz - eth1	ip_src=20.20.20.64/26 ip_dst=20.20.20.0/26	Tráfico de Subred A' a Subred B'
S4	Oz - eth1	Poisson - eth1	ip_src=20.20.20.0/26 ip_dst=20.20.20.64/26	Tráfico de Subred B' a Subred A'

Table 6.2 VPN capa 3 - Escenario 2, servicios creados

Para todos estos servicios además se indica como valor de ethertype 0x0800, correspondiente al tipo de tráfico IPv4.

Sobre este escenario se prueban diferentes combinaciones para la generación de tráfico, variando tanto la subred origen y destino como el tipo de tráfico. Se prueba generar paquetes ICMP mediante el comando **Ping** para ambas subredes, en especial el caso en que la numeración IP se repite (por ejemplo ping de un host en subred A a otro host en subred B en simultáneo con un ping de un host en subred A' a otro host en subred B'); observando que el prototipo enruta correctamente el tráfico asociado a cada servicio, garantizando así el aislamiento del espacio de direcciones.

También se logra de forma exitosa establecer conexiones **SSH** entre dos hosts en diferentes subredes para ambas organizaciones y generar requests **HTTP**, cargando desde un host en una subred una página web alojada en la otra subred. Esta prueba se realiza para ambas organizaciones utilizando páginas web con diferente contenido; de esta forma se determina fácilmente para una organización si las solicitudes HTTP son correctamente enrutadas y aisladas del tráfico de la otra organización. En el apéndice F.4 se muestran los detalles técnicos asociados a estas pruebas como capturas de las tablas de flujos de cada nodo en la red, capturas del procesamiento de tráfico asociado a los servicios en los diferentes casos mencionados, entre otra información.

En la siguiente sección se discute la configuración de un servicio de VPN de capa 2 utilizando la red prototipo y la aplicación RAUFlow.

6.3 VPN de capa 2

A diferencia de las VPN de capa 3, las cuales se definen para un protocolo o aplicación particular (por ejemplo IPv4), las redes privadas de capa 2 permiten definir servicios de forma agnóstica a los mismos, transportando tráfico entre dos subredes de cualquier protocolo de capa 3 o aplicación

superior. Puede pensarse en un servicio de VPN de capa 2 entre dos subredes como un “cable” virtual que los conecta directamente.

En este trabajo, a modo de ejemplo se implementan dos servicios de VPN de capa 2 sobre un escenario minimalista en el que se cuenta con una única organización compuesta por dos sucursales físicamente separadas, sobre la red del laboratorio previamente definida. De esta forma lo que se busca con este escenario es construir dos “cables” y verificar que se transporta correctamente tráfico desde una subred a otra, independientemente del protocolo de capa 3 o superior.

6.3.1 Escenario

El escenario configurado para este caso de uso, así como los caminos asociados a cada servicio (LSP) se muestra en la figura 6.7. Como puede apreciarse el mismo se compone de una organización con dos sucursales físicamente separadas, para las cuales se instancia dos servicios (ver tabla 6.3).

Notar cómo no se define una clasificación de tráfico para ambos servicios creados. Esto se debe justamente a la naturaleza del servicio de VPN creado. Al ser un servicio de VPN capa 2 las únicos parámetros utilizados para su construcción son los nodos de ingreso y egreso del servicio; en otras palabras, los extremos del “cable” virtual que se está creando.

Teniendo este escenario configurado, se genera tráfico desde la subred A a la subred B (servicio S1), probando con diferentes protocolos de capa 3, buscando verificar si el mismo atraviesa la red del laboratorio independientemente del protocolo.

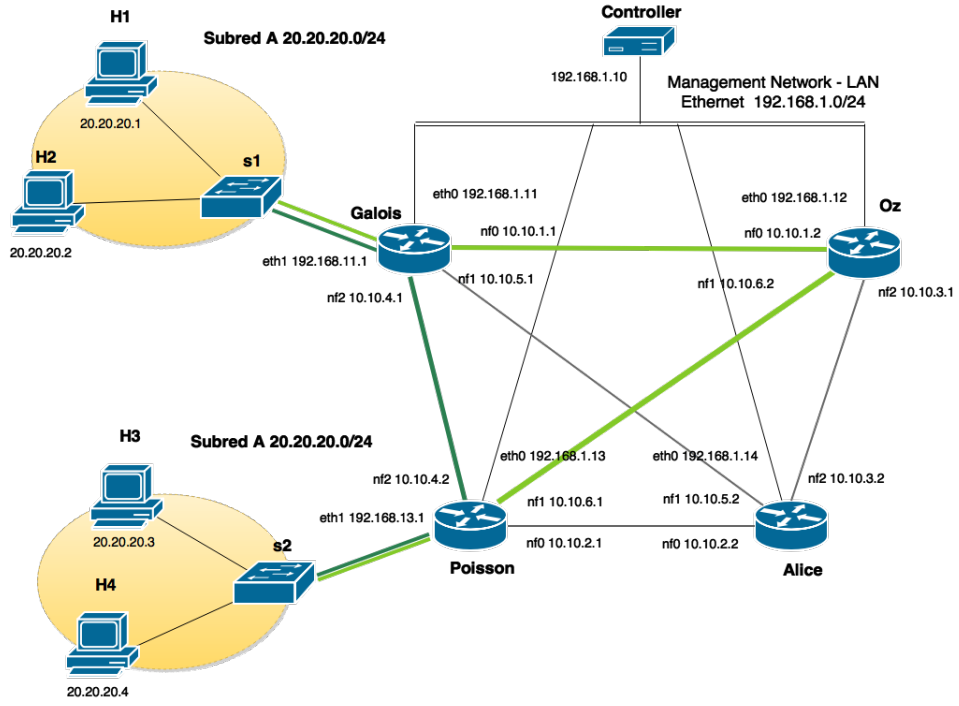

Fig. 6.7 VPN de capa 2 Escenario

Para generar el tráfico se utiliza la herramienta PacketH[47], la cual permite generar paquetes asociados a diferentes protocolos desde una interfaz gráfica. Con esta herramienta se generan paquetes asociados a tres protocolos diferentes: ARP, IPv4 e IPv6 con tags de VLAN. En la figura 6.8 se muestran los paquetes generados. Por otro lado detalles más técnicos como capturas tcpdump del procesamiento del tráfico entre otros se incluyen en el apéndice F.4 en caso de que el lector desee profundizar en los mismos.

Nombre	Ingreso	Egreso	Clasificación	Descripción
S1	Galois - eth1	Poisson - eth1	-	Tráfico de capa 2 de Subred A a Subred B
S2	Poisson - eth1	Galois - eth1	-	Tráfico de capa 2 de Subred B a Subred A

Table 6.3 VPN de Capa 2, servicios creados

De esta forma se verifica la capacidad de esta implementación de servicio de VPN de capa 2, para procesar tráfico asociado a diferentes protocolos de capas superiores definiendo un único servicio. De igual forma puede verificarse la capacidad para procesar paquetes de cualquier otro protocolo de capa 3 o superiores.

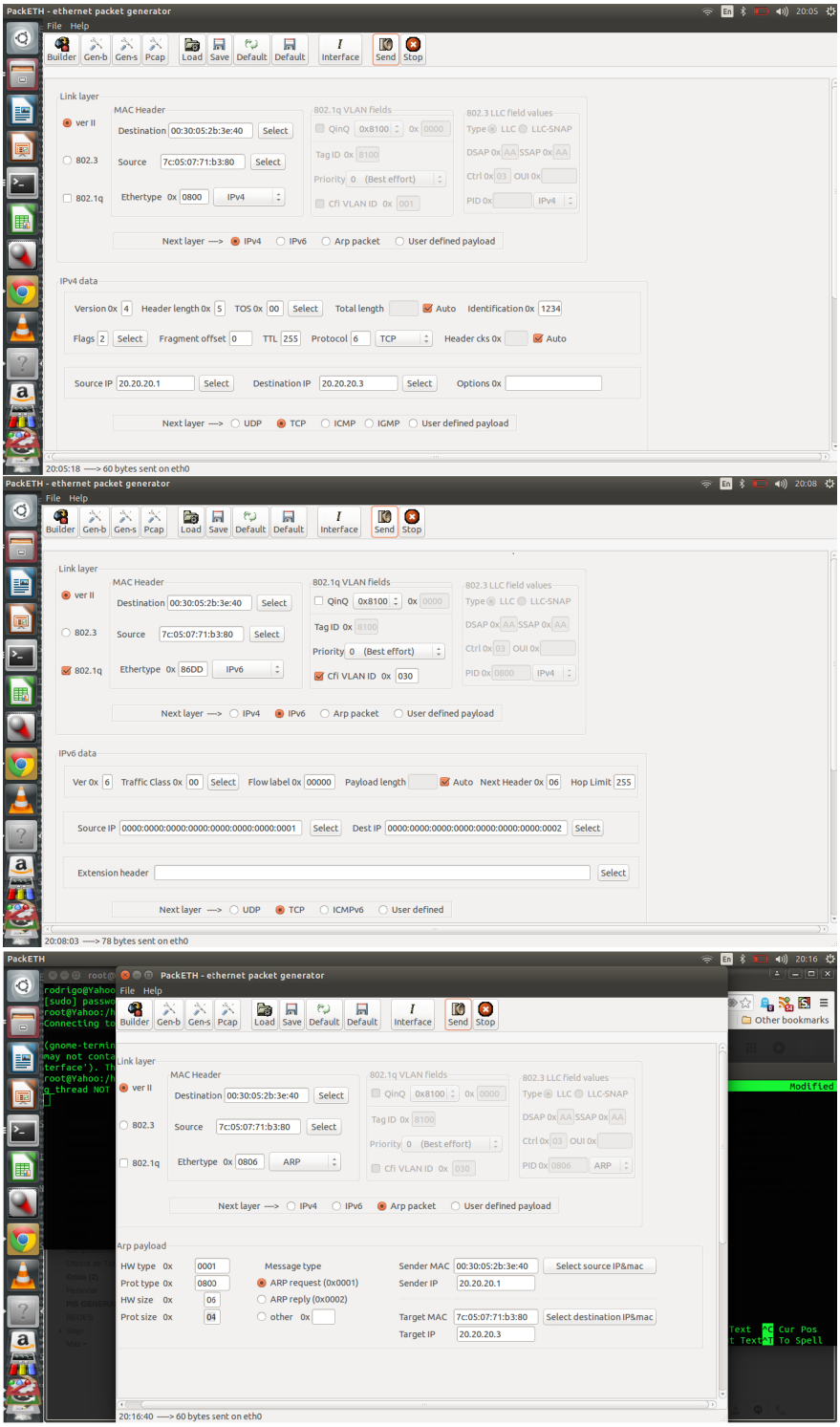

Fig. 6.8 Paquetes generados con PacketH

Capítulo 7

Gestión del proyecto

En el presente capítulo se realiza un breve análisis acerca de la ejecución del proyecto, mostrando de las principales actividades y líneas de trabajo seguidas, las respectivas fechas de inicio y fin de las actividades, destacando en los casos en que corresponde las posibles dependencias y atrasos.

7.1 Ejecución del proyecto

En la figura 7.1 se muestra un diagrama de Gantt con las actividades más importantes durante la ejecución de este proyecto, con sus respectivas fechas de inicio y fin.

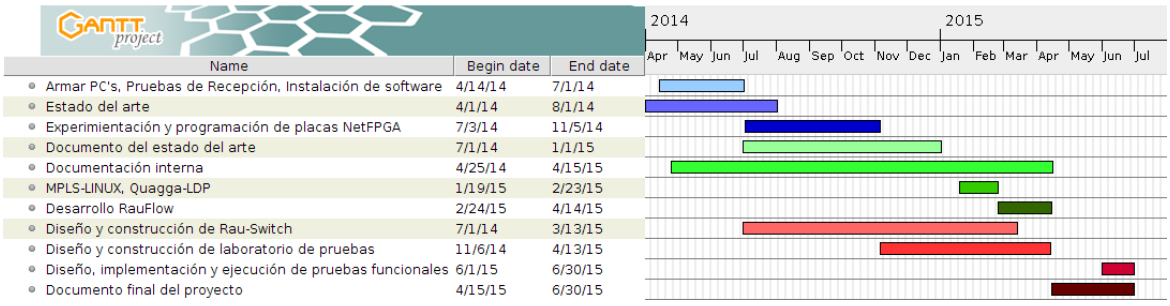

Fig. 7.1 Diagrama de Gantt, Actividades del Proyecto

A continuación se describen los principales contratiempos experimentados en la ejecución de cada una de estas actividades.

Armar PC's, Pruebas de Recepción, Instalación de software básico: Esta línea de trabajo engloba las actividades realizadas al inicio del proyecto en las que se monta la plataforma base de cuatro dispositivos RAU-Switch utilizados además como estación de trabajo durante toda la ejecución del proyecto. En esta etapa se recibe el hardware utilizado (PC's, tarjetas NetFPGA, transceiver SFP y patch cords de fibra entre otros), se realiza un inventario, pruebas de aceptación y se procede con la

instalación del hardware y software básico (Sistema Operativo, suite de desarrollo de Xilinx entre otros).

Dentro de esta etapa son críticas la recepción de todos los dispositivos de hardware y las pruebas de recepción. Cabe destacar sobre esto, el 14/04/2014 se reciben las primeras dos PC's con las cuales empezar a trabajar, mientras que en la primera semana del mes de Junio llegan cinco placas NetFPGA.

Estado del Arte: Esta línea de trabajo se desarrolla sin contratiempos aunque se extiende un mes más de lo previsto cuando comienza la etapa de experimentación con el hardware.

Experimentación y programación de las placas NetFPGA: Esta linea de trabajo engloba actividades como la programación del hardware con los diferentes proyectos disponibles (de referencia y comunitarios), aprendizaje del entorno de desarrollo, evaluar la posibilidad de programar directamente el hardware con desarrollos propios, diseñar una estrategia de implementación para RAU-Switch.

Dentro de esta etapa es crítica la obtención de un paquete de licencias pagas especiales para la suite de Xilinx ISE SDK. Oficialmente se obtiene este paquete el 29/10/2014 aunque el 16/10/2014 se obtiene una solución provisoria a este problema. A su vez es crítico de esta etapa completar la programación persistente del hardware sin errores en el funcionamiento, hecho que sucede el día 05/11/2014.

Documento del estado del Arte: Esta actividad se realiza en paralelo con el resto de las actividades del proyecto, teniendo dos iteraciones para la generación de un documento del estado del arte previo al expuesto en el informe final del proyecto. Estas son el 15/09/2014 y el 01/12/2015.

Documentación interna: Esta actividad se desarrolla en el transcurso de todo el proyecto orientada a la redacción del documento final del proyecto.

MPLS-Linux, Quagga-LDP: Esta línea de trabajo engloba las actividades de experimentación realizada con ambas herramientas durante la fase de construcción de RAU-Switch. A pesar del tiempo invertido en esta actividad no se obtienen los resultados esperados y se decide abandonar esta línea de trabajo sopesando el tiempo invertido.

Esta actividad es crítica en el desarrollo de RAUFlow no permitiendo comenzar con esta última hasta que se finalizara.

Desarrollo de RAUFlow: Esta línea de trabajo engloba las actividades de diseño e implementación de RAUFlow. Se invierte poco más de un mes en esta actividad debido al atraso general en el cronograma del proyecto, sacrificando un conjunto bastante interesante de funcionalidades de la aplicación.

Diseño y construcción de RAU-Switch: Esta línea de trabajo engloba actividades de programación, experimentación e investigación con diferentes herramientas de software y el hardware utilizado, hasta alcanzar la construcción del dispositivo en cuestión. Como etapa en el proyecto comienza una vez que se finalizan las pruebas de recepción de hardware y finaliza una vez que se alcanza un producto estable funcionalmente.

Como etapa de proyecto son críticas las pruebas de recepción de hardware, las cuales habilitan una vez concluidas a comenzar a trabajar en la misma.

Diseño y construcción del laboratorio de pruebas: Comienza como etapa cuando culmina la etapa de experimentación con el hardware y comienza a pensarse en el diseño de RAU-Switch y de un laboratorio de pruebas utilizando los cuatro nodos disponibles y culmina el 13/04/2015 cuando se reciben las últimas componentes de hardware necesarias para la construcción del mismo.

Diseño, implementación y ejecución de pruebas funcionales: Engloba todas las actividades de verificación del prototipo construido utilizando el laboratorio de pruebas.

Son críticas para realizar las actividades de ejecución de pruebas el contar con el laboratorio de pruebas construido y una versión estable de RAUFlow.

Documentación del proyecto: Si bien a lo largo de todo el proyecto se genera documentación, es de carácter interno. La elaboración del documento final de proyecto se posterga hasta el momento de contar con una versión estable de RAUFlow, lo cual sucede el 15/04/15.

Capítulo 8

Conclusiones

En este capítulo se resumen los principales resultados, logros y conclusiones de este trabajo. Luego se enumeran las principales líneas de trabajo a futuro identificadas, entre las cuales se incluyen mejoras y extensiones al prototipo, como posibles líneas de investigación.

8.1 Conclusiones

Se realizó una investigación en profundidad del estado del arte de las redes definidas por software (SDN) y la plataforma de hardware NetPGA, presentando un resúmen de los resultados obtenidos en el capítulo 2 de este trabajo.

Por otro lado se logró desarrollar un prototipo funcional utilizando el hardware NetFPGA y el enfoque de SDN, dotado de funcionalidades para la creación y gestión de servicios de redes privadas virtuales. Este prototipo es la prueba de que es posible implementar una red IP/MPLS en la que la inteligencia de los dispositivos de red convencionales es extraída y colocada en una entidad centralizada.

El prototipo alcanzado se compone de un dispositivo de red denominado RAU-Switch y una aplicación de gestión denominada RAUFlow. Orientado a una futura reproducción de este trabajo, se generó un manual de construcción de RAU-Switch donde se detallan todas las componentes de hardware y software utilizadas, junto con el procedimiento de instalación y configuración de las mismas, para obtener un dispositivo idéntico al desarrollado en este trabajo. Además se creó un sitio del proyecto en la plataforma Github (Proyecto RRAP[49]) en donde se compartió con la comunidad el código fuente de todas las componentes de software desarrolladas en el marco de este proyecto, el conocimiento y experiencia generados en relación a las herramientas utilizadas.

Por otro lado se diseñó e implementó un laboratorio de experimentación sobre el cual se ejecutaron una serie de pruebas orientadas a la validación funcional del prototipo referidas a dos casos de uso representativos, ellos fueron la configuración de una VPN de capa 3 multipunto y una VPN de capa 2 punto a punto, sobre los cuales a su vez se ejecutaron una serie de pruebas para verificar la correcta

implementación. De esta forma se logró validar en lo que concierne a los requerimientos planteados, la aplicabilidad del enfoque SDN en la construcción de la RAU2.

Además se generó una publicación científica en la cual se presentan los resultados obtenidos por este trabajo en el contexto de la construcción de un prototipo para la RAU2 bajo el nombre "RAU2 testbed: a network prototype for evolved service experimentation" [62]. Este artículo fue presentado en la conferencia Latin American Network Operations and Management Symposium (LANOMS) y aceptado en formato de poster.

No menos importante, se logró integrar el equipo de desarrollo con la comunidad de NetFPGA a través de la participación en la lista oficial de correos; contribuyendo desde la experiencia obtenida en la contestación de dudas y reportando dos errores importantes en el proyecto ReferenceNIC. A su vez se generó un grupo de trabajo local integrando profesionales del SeCIU, Centro de Capacitación y Desarrollo de ANTEL y del Centro Universitario de la Región Este (CURE), en el cual se organizaron reuniones quincenales durante un período de casi ocho meses para la puesta en común y generación de experiencia y conocimiento en el área de SDN.

Finalmente, pese q que el dispositivo RAU-Switch diseñado está fuertemente limitado en su rendimiento y performance por su implementación mayoritariamente en software y a pesar de que no se pudieron realizar pruebas comparativas con productos comerciales similares en funcionalidades, se logró validar la utilización de esta plataforma en la construcción de un dispositivo compatible con OpenFlow. Además se identificó el camino a seguir para la construcción de un prototipo con mejores prestaciones explotando al máximo las capacidades del hardware disponible.

8.2 Trabajo a futuro

En este trabajo no se compara el rendimiento del prototipo con productos comerciales similares en funcionalidades dado que estos implementan el plano de datos de OpenFlow en hardware, mientras que el prototipo lo implementa en software (Open vSwitch). De esta forma los tiempos de rendimiento serían incomparables.

Por ello interesa fuertemente extender el proyecto OpenFlow de la plataforma NetFPGA para que implemente el protocolo como mínimo en la versión 1.3.1 y así poder realizar una evaluación experimental que permita validar desde el punto de vista del rendimiento la aplicación de las tecnologías utilizadas en la construcción de la RAU2.

El algoritmo de ruteo implementado basa su métrica solamente en el costo asociado a un enlace de red. Extender la definición de esta métrica, contemplando otros atributos de un enlace como el ancho de banda disponible o tecnología (fibra óptica, enlace de cobre, etc) y a la vez contemplar restricciones como garantizar un ancho de banda mínimo, cantidad de enlaces atravesados, incluir ó excluir nodos por los que pasará el tráfico y demoras de extremo a extremo (en otras palabras implementar un CSPF) redituaría en la capacidad para incorporar funcionalidades de calidad de servicio.

De la mano de la extensión del algoritmo de ruteo a un algoritmo CSPF, se puede trabajar en el desarrollo de funcionalidades avanzadas que permitan implementar técnicas de Ingeniería de Tráfico,

desarrollando así un prototipo más flexible en la asignación de los recursos disponibles y con mejor calidad en los servicios brindados.

Un servicio queda definido en el sistema por los nodos e interfaces de entrada y salida a la red y las características del tráfico asociado. La incorporación de nuevas dimensiones a la definición de un servicio, como dimensiones de calidad de servicios (QoS) o simplemente la dimensión tiempo representaría una gran mejora funcional y un salto de calidad en el aprovechamiento de las capacidades de la infraestructura del prototipo. Por ejemplo incorporando la dimensión tiempo se podrían definir servicios para rangos horarios, mejorando la precisión con la que se distribuye el ancho de banda disponible.

En el prototipo se almacena en memoria información que es ingresada por un usuario de la aplicación RAUFlow. En particular se almacenan datos extra de cada nodo e interfaz y se almacena la definición de los servicios. Esta información se pierde en caso de que la aplicación sea interrumpida puesto que no se persiste de forma no volátil. Resulta interesante entonces incorporar a la arquitectura de RAUFlow una capa de persistencia que permita guardar de forma no volátil esta información, además de la definición de estrategias para la reconstrucción de servicios cuando se carga esta información eventualmente en una topología de red diferente a la inicial.

Almacenar en memoria y de forma centralizada la información topológica de red, información asociada a servicios y eventualmente funcionalidades de QoS e Ingeniería de tráfico, así como ejecutar algoritmos que utilicen intensivamente estos datos como un CSPF centralizado puede acarrear serias barreras de escalabilidad en el prototipo. Una línea de investigación interesante sería el desarrollo de una jerarquía de Controladores, cada uno responsable del plano de control de una porción de la topología global. De esta forma se generarían islas SDN en donde el tráfico interno es resuelto por el controlador local y se consulta al Controlador global para resolver la forma en que se enruta tráfico entre diferentes islas. A su vez se puede crecer en la cantidad de niveles dentro de la jerarquía tanto como se quiera.

Finalmente y no menos importante se puede trabajar en el desarrollo de nuevas funcionalidades en RAUFlow, así como en la mejora de las ya existentes. Algunas de las funcionalidades que se pueden incorporar son la capacidad para definir manualmente caminos, indicando los nodos por los que se quiere pasar, soportar múltiples caminos para un mismo par de nodos origen y destino y así poder implementar balanceo de carga, creación de una VPN multipunto automáticamente a partir de una lista de nodos e interfaces involucradas.

Bibliografía

[1] (2014a). Beacon Controller. url: https://openflow.stanford.edu/display/Beacon/Home, último acceso Junio 2014.

[2] (2014b). Floodlight Controller. url: http://www.projectfloodlight.org/floodlight/, último acceso Junio 2014.

[3] (2014c). Flowvisor Controller. url: https://github.com/OPENNETWORKINGLAB/flowvisor/wiki, último acceso Junio 2014.

[4] (2014d). Helios Controller. url: http://www.nec.com/, último acceso Junio 2014.

[5] (2014e). Jaxon Controller. url: http://jaxon.onuos.org/wp/?page_id=12, último acceso Junio 2014.

[6] (2014f). Maestro Controller. url: https://code.google.com/p/maestro-platform/, último acceso Junio 2014.

[7] (2014g). NodeFlow Controller. url: https://github.com/dreamerslab/node.flow, último acceso Junio 2014.

[8] (2014h). NOX Controller. url: http://www.noxrepo.org/nox/about-nox/, último acceso Junio 2014.

[9] (2014i). OpenDaylight Controller. url: http://www.opendaylight.org/, último acceso Junio 2014.

[10] (2014j). OpenMUL Controller. url: http://kulcloud.wordpress.com/, último acceso Junio 2014.

[11] (2014k). POX Controller. url: http://www.noxrepo.org/pox/about-pox/, último acceso Junio 2014.

[12] (2014l). Routeflow Controller. url: http://cpqd.github.io/RouteFlow/, último acceso Junio 2014.

[13] (2014m). Ryu Controller. url: http://osrg.github.io/ryu/, último acceso Junio 2014.

[14] (2014n). SNAC Controller. url: https://github.com/bigswitch/snac, último acceso Junio 2014.

[15] (2014o). Trema Controller. url: http://trema.github.io/trema/, último acceso Junio 2014.

[16] (2015). ANTEL Servicios de redes privadas. url: https://www.antel.com.uy/antel/empresas/datos-e-internet/redes-privadas/redes-privadas-fijas, último acceso Julio 2015.

[17] (2015). Asia Pacific Advanced Network(APAN). url: https://www.apan.net/, último acceso Mayo 2015.

[18] (2015). BIRD internet routing daemon. url: http://www.nongnu.org/quagga/, último acceso Junio 2015.

[19] (2015). CA*net4 sitio oficial. url: http://rcsg-gsir.imsb-dsgi.nrc-cnrc.gc.ca/documents/internet/node12.html, último acceso Mayo 2015.

[20] (2015). Centec Networks whitebox solutions for sdn. url: https://www.opennetworking.org/listings/centec-networks, último acceso Mayo 2015.

[21] (2015). Common Open Research Emulator (CORE). url: http://www.nrl.navy.mil/itd/ncs/products/core, último acceso Julio 2015.

[22] (2015). GÉANT Project sitio oficial. url: http://www.geant.net/Pages/default.aspx, último acceso Mayo 2015.

[23] (2015). HP productos compatibles con openflow. url: http://h17007.www1.hp.com/uy/es/solutions/technology/openflow/index.aspx, último acceso Mayo 2015.

[24] (2015). Internet 2 sitio oficial. url: http://www.internet2.edu/, último acceso Mayo 2015.

[25] (2015). JTAG Driver para Ubuntu 12.04. url: http://git.zerfleddert.de/cgi-bin/gitweb.cgi/usb-driver, último acceso Julio 2015.

[26] (2015a). Lista de correos netfpga-nf10g-beta. url: cl-netfpga-nf10g-beta@lists.cam.ac.uk, último acceso Mayo 2015.

[27] (2015). Lista de productos compatibles con SDN. url: https://www.opennetworking.org/products-listing, último acceso Mayo 2015.

[28] (2015b). Manual de ejecucion test rldram netfpga-10g. url: https://github.com/NetFPGA/NetFPGA-public/wiki/RLDRAM%20Test%20Manual, último acceso Mayo 2015.

[29] (2015a). Mininet - Entorno de emulación virtual. url: http://mininet.org/, último acceso Mayo 2015.

[30] (2015b). miniNExT, para prototipar redes complejas con MiniNet. url: https://github.com/USC-NSL/miniNExT, último acceso Julio 2015.

[31] (2015). NetFPGA - PCIE Programming. url: https://github.com/NetFPGA/NetFPGA-public/wiki/PCIE-Programming, último acceso Mayo 2015.

[32] (2015). NetFPGA - Production Test Manual. url: https://github.com/NetFPGA/NetFPGA-public/wiki/Production%20Test%20Manual, último acceso Mayo 2015.

[33] (2015c). NetFPGA - Publicaciones academicas. url: http://netfpga.org/site/#/publications/, último acceso Mayo 2015.

[34] (2015d). NetFPGA-1G - Lista de proyectos para la plataforma. url: http://netfpga.org/site/#/systems/4netfpga-1g/applications/, último acceso Mayo 2015.

[35] (2015e). NetFPGA Descripción de RLDRAM Test NetFPGA-10G. url: https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G%20RLDRAM%20Test, último acceso Mayo 2015.

[36] (2015f). NetFPGA Descripción de Test de Producción NetFPGA-10. url: https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G-Production-Test, último acceso Mayo 2015.

[37] (2015a). NetFPGA Learning CAM Switch. url: https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G-Learning-CAM-Switch, último acceso Mayo 2015.

[38] (2015g). NetFPGA Manual de Test de Produción NetFPGA-10. url: https://github.com/NetFPGA/NetFPGA-public/wiki/Production%20Test%20Manual, último acceso Mayo 2015.

[39] (2015b). NetFPGA Reference NIC. url: https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G-Reference-NIC, último acceso Mayo 2015.

[40] (2015c). NetFPGA Reference Router. url: https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G-Reference-Router, último acceso Mayo 2015.

[41] (2015). Ofelia openflow in europe: Linking infrastructure and applications. url: http://www.fp7-ofelia.eu/, último acceso Mayo 2015.

[42] (2015). Open Networking Foundation (ONF) sitio oficial. url: https://www.opennetworking.org/sdn-resources/sdn-definition, último acceso Julio 2015.

[43] (2015a). Open vSwitch Repositorio de código fuente. url: https://github.com/openvswitch/ovs, último acceso Mayo 2015.

[44] (2015). OpenDaylight arquitectura. url: http://www.opendaylight.org/announcements/2013/09/opendaylight-project-releases-new-architecture-details-its-software-defined, último acceso Julio 2015.

[45] (2015). OpFlex An Open Source Approach. url: http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731304.html, último acceso Julio 2015.

[46] (2015b). OVS Controlador de referencia. url: http://manpages.ubuntu.com/manpages/trusty/man8/ovs-controller.8.html, último acceso Julio 2015.

[47] (2015). PacketH software para generar paquetes de red. url: http://packeth.sourceforge.net/packeth/Home.html, último acceso Julio 2015.

[48] (2015). Pica8 whitebox sdn. url: http://www.pica8.com/, último acceso Mayo 2015.

[49] (2015). ProyectoRRAP repositorio git de código fuente. url: https://github.com/ProyectoRRAP/LiveCode, último acceso Agosto 2015.

[50] (2015). Quagga routing suite. url: http://bird.network.cz/, último acceso Junio 2015.

[51] (2015). Red CLARA Historia. url: http://www.redclara.net/index.php/somos/sobre-redclara/historia, último acceso Julio 2015.

[52] Andersson, L., AB, A., and Minei, I. (1996). LDP Specification rfc5036. *Network Working Group*.

[53] Azodolmolky, S. (October 2013). *Software Defined Networking with OpenFlow, Get hands on with the platforms and development tools used to build OpenFlow network applications.* Packt Publishing, 35 Livery Street, published by packt publishing ltd. edition.

[54] Biswas, S. S., Alam, B., and Doja, M. (2013). Generalization of dijkstra's algorithm for extraction of shortest paths in directed multigraphs. *Journal of Computer Science*, 9(3):377–382.

[55] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. (2005). Design and implementation of a routing control platform. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 15–28. USENIX Association.

[56] Campbell, A. T., Katzela, I., Miki, K., and Vicente, J. (1999). Open signaling for atm, internet and mobile networks (opensig'98). *ACM SIGCOMM Computer Communication Review*, 29(1):97–108.

[57] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM.

[58] Case, J., Fedor, M., Schoffstall, M., and Davin, C. (1989). A simple network management protocol (snmp).

[59] Doria, A., Hellstrand, F., Sundell, K., and Worster, T. (2002). General switch management protocol (gsmp) v3.

[60] Doria, A., Salim, J. H., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and Halpern, J. (2010). Forwarding and control element separation (forces) protocol specification. *Internet Requests for Comments, RFC Editor, RFC*, 5810.

[61] E. Rosen, Y. R. (2006). Bgp/mpls ip virtual private networks (vpns).

[62] Eduardo, G., Giachino, M., Amaro, R., and Viotti, E. (2015). RAU2 testbed: a network prototype for evolved service experimentations. Accepted as a poster at the 8th Latin American Network Operations and Management Symposium, October 1–3, Joao Pessoa, Brazil.

[63] Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., and Van Der Merwe, J. (2004). The case for separating routing from routers. In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 5–12. ACM.

[64] Feamster, N., Rexford, J., and Zegura, E. (2013). The road to sdn. *Queue*, 11(12):20.

[65] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. (2005). A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54.

[66] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110.

[67] Hassas Yeganeh, S. and Ganjali, Y. (2012). Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 19–24. ACM.

[68] Heller, B., Sherwood, R., and McKeown, N. (2012). The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 7–12. ACM.

[69] Kompella, K. and Rekhter, Y. (2007). Virtual private lan service (vpls) using bgp for auto-discovery and signaling.

[70] Koponen, T., Casado, M., Gude, N., and Stribling, J. (2014). Distributed control platform for large-scale production networks. US Patent 8,830,823.

[71] Lasserre, M. and Kompella, V. (2007). Virtual private lan service (vpls) using label distribution protocol (ldp) signaling. Technical report, RFC 4762, January.

[72] Leu, J. R. (2015a). MPLS Linux proyecto original 2005. url: http://repo.or.cz/w/mpls-linux.git, último acceso Julio 2015.

[73] Leu, J. R. (2015b). Quagga LDP, extension de Quagga para trabajar con MPLS-Linux. url: http://repo.or.cz/w/jleu-quagga.git, último acceso Julio 2015.

[74] Levin, D., Wundsam, A., Heller, B., Handigol, N., and Feldmann, A. (2012). Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 1–6. ACM.

[75] Malkin, G. (1994). Rip version 2-carrying additional information.

[76] Maravick, I. (2015). MPLS Linux, proyecto basado en MPLS Linux original. url: https://github.com/i-maravic/MPLS-Linux, último acceso Julio 2015.

[77] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

[78] Migration Working Group (2014). *ONF Migration Use Cases and Methods*.

[79] Moore, J. T. and Nettles, S. M. (2001). Towards practical programmable packets. In *Proceedings of the 20th Conference on Computer Communications (INFOCOM). Citeseer*. Citeseer.

[80] Moy, J. (1998). rfc 2328: Ospf version 2.

[81] Nadeau, T., Srinivasan, C., Bloomberg, L., and Viswanathan, A. (2004). Multiprotocol label switching mpls forwarding equivalence class to next hop label forwarding entry fec-to-nhlfe management information base mib. Technical report, RFC 3814.

[82] NetFPGA (2015a). Compendio de referencia de SDN. url: https://sites.google.com/site/sdnreadinglist/, último acceso Mayo 2015.

[83] NetFPGA (2015b). NetFPGA sitio del proyecto. url: http://www.netfpga.org/, último acceso Mayo 2015.

[84] Nunes, B., Mendonca, M., Nguyen, X., Obraczka, K., and Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks.

[85] ONF (Agosto 2012). Specification openflow switch, pages 70–71.

[86] Rexford, J., Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Xie, G., Zhan, J., and Zhang, H. (2004). Network-wide decision making: Toward a wafer-thin control plane. In *Proc. HotNets*, pages 59–64.

[87] Rose, M. T. and McCloghrie, K. (1990). Structure and identification of management information for tcp/ip-based internets. *Structure*.

[88] Rosen, E., Rekhter, Y., Tappan, D., Farinacci, D., Fedorkow, G., Li, T., and Conta, A. (2001a). Mpls label stack encoding.

[89] Rosen, E., Viswanathan, A., Callon, R., et al. (2001b). Multiprotocol label switching architecture.

[90] Rosen, E. C. and Rekhter, Y. (1999). Bgp/mpls vpns.

[91] Routing, O. I.-I. I.-d. (1990). Protocol, 1990. internet engineering task force. Technical report, RFC 1142.

[92] Sherwood, R., Chan, M., Covington, A., Gibb, G., Flajslik, M., Handigol, N., Huang, T.-Y., Kazemian, P., Kobayashi, M., Naous, J., et al. (2010). Carving research slices out of your production networks with openflow. *ACM SIGCOMM Computer Communication Review*, 40(1):129–130.

[93] Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., and Minden, G. J. (1997). A survey of active network research. *Communications Magazine, IEEE*, 35(1):80–86.

[94] Tennenhouse, D. L. and Wetherall, D. J. (2002). Towards an active network architecture. In *DARPA Active NEtworks Conference and Exposition, 2002. Proceedings*, pages 2–15. IEEE.

[95] Tootoonchian, A. and Ganjali, Y. (2010). Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3. USENIX Association.

[96] Westphal, R. (2015). Quagga LDP, proyecto basado en Quagga LDP original para funcionar con proyecto MPLS-Linux de I. Maravick. url: https://github.com/rwestphal/quagga-public/tree/mpls, último acceso Julio 2015.

[97] Yu, M., Rexford, J., Freedman, M. J., and Wang, J. (2011). Scalable flow-based networking with difane. *ACM SIGCOMM Computer Communication Review*, 41(4):351–362.

Apendice A

Solución a errores encontrados en la plataforma NetFPGA

Durante el tiempo en que se trabajó con el hardware NetFPGA, utilizando la versión 5.0.5 del código fuente de la plataforma, se detectaron errores en el mismo de severidad alta en relación a su impacto en el diseño del prototipo. En total se detectaron dos errores, los cuales fueron reportados a la comunidad y equipo de desarrollo de NetFPGA a través de una lista de correos[26]. El equipo de NetFPGA confirmó ambos errores y sugirió soluciones pasajeras a ambos problemas.

A su vez estos errores fueron solucionados en la siguiente versión del código fuente de la plataforma.

A.1 Problema con la herramienta pcieprog y ReferenceNIC

El primer error detectado se encuentra en la herramienta **pcieprog** dentro de la plataforma de NetFPGA. La misma se utiliza para transferir el archivo binario generado a partir del bitfile del proyecto (en este caso ReferenceNIC) a una de las memorias flash del hardware.

Al ejecutarse esta herramienta con el binario generado para el proyecto ReferenceNIC, la misma se ejecutaba durante horas con un 100% de utilización de la CPU, sin indicios de terminar su ejecución. Tras probar con varias ejecuciones, se realizó la misma prueba con dos proyectos adicionales (reference_ switch y reference_ router), compatibles también con la programación persistente.

Para estos proyectos la ejecución del programa terminaba tras algunos minutos, por lo que se presumió la presencia de algún error en el código del proyecto ReferenceNIC. Por otro lado se realizó una inspección al código de la herramienta **pcieprog**, para determinar en qué momento el programa se estancaba. Se detectó que el programa quedaba trancado en un bucle infinito dado que no se cumplía nunca la condición de salida.

Tras reportarse este comportamiento a la lista en correos[26], se confirmó el error y se proporciono la siguiente solución transitoria:

1. Dentro del directorio donde se encuentra el código fuente de la plataforma, posicionarse en projects/reference_nic/hw/. Abrir el archivo system.mhs.
   Cambiar:

   "PORT axi_ emc_ 0_ Mem_ DQ_ pin = axi_ emc_ 0_ Mem_ DQ, DIR = IO, VEC = [*7*:0]"

   por

   "PORT axi_ emc_0 _Mem_ DQ_ pin = axi_ emc_ 0_ Mem_ DQ, DIR = IO, VEC = [*31*:0]"

2. Posicionarse ahora en projects/reference_nic/hw/. Abrir el archivo xflow.opt.
   Cambiar:

   "-t *1*"

   por

   "-t *5*"

3. Recompilar el proyecto ReferenceNIC ejecutando el comando **make** en el directorio projects/reference_ nic/

4. Utilizar el nuevo bitfile generado para programar el hardware, generar un nuevo archivo binario y transferir a la memoria flash.

Una vez realizados estos cambios la herramienta **pcieprog** funcionó correctamente, transfiriendo el contenido del archivo binario a la memoria flash.

A.2 Error en el driver para el proyecto ReferenceNIC

El segundo error detectado trabajando con este proyecto en la modalidad de programación persistente se encuentra relacionado al driver del proyecto, utilizado en la interacción entre el hardware y el sistema operativo anfitrión.

Este error fue detectado experimentalmente al realizar pruebas funcionales con varios nodos y el hardware programado con el proyecto ReferenceNIC. A continuación se detalla dicho error:

- **Descripción:**
  Al programarse el hardware con el proyecto ReferenceNIC desde la memoria flash utilizando la programación pcie (programación persistente), si el equipo es encendido con los enlaces

de fibra óptica conectados en cada uno de los puertos físicos, el hardware deja de funcionar correctamente. Por otro lado si el equipo es encendido sin enlaces conectados a algún puerto y luego son conectados manualmente una vez que el encendido ha terminado, el hardware funciona correctamente.

- **Escenario:**
  - Dos PCs cada una con una tarjeta NetFPGA instalada
  - Las tarjetas están conectadas por cuatro enlaces de fibra óptica (nf0-nf0, nf1-nf1, nf2-nf2 y nf3-nf3)
  - Ambas tarjetas están programadas con el proyecto ReferenceNIC vía programación pcie
  - Se utiliza la versión 5.0.5 de código fuente con el parche mencionado en la sección anterior
- **Efecto:**
  Incapacidad parar ejecutar exitosamente el comando ping entre cualquier par de interfaces conectadas.
- **Reproducción:**
  Partiendo de ambas PCs apagadas
  1. Encender una de las PCs, teniendo conectados los enlaces ópticos entre ambos equipos
  2. Cargar el driver y configurar manualmente direcciones IP para cada interfaz (nf0 . . . nf3)
  3. Realizar los pasos 1 y 2 para la otra PC
  4. LEDs asociados a cada puerto en una de las tarjetas NetFPGA son encendidos por completo mientras que en la otra tarjeta solamente la mitad de los LEDs se encuentran encendidos
  5. Si se apaga y enciende la PC con la tarjeta NetFPGA cuyos LEDs se encuentran todos encendidos luego la otra tarjeta enciende todos sus LEDs mientras que la primera enciende solo la mitad
  6. El último paso puede repetirse y el comportamiento observado se mantiene, en particular la tarjeta que es iniciada al final funciona correctamente mientras que la otra no
- **Solución:**
  1. Apagar ambas PCs
  2. Desconectar todos los enlaces ópticos
  3. Encender ambas PCs (cargar el driver y configurar cada interfaz con una dirección IP apropiada)
  4. Luego que cada PC está encendida y configurada conectar los enlaces ópticos uno a uno.

Este reporte fue enviado a la comunidad NetFPGA en donde se constata nuevamente la presencia de un error, esta vez en la configuración de los chips PHY de la tarjeta. La solución propuesta por el equipo de NetFPGA es la siguiente:

1. Dentro del directorio donde se encuentra el código fuente de la plataforma, posicionarse en projects/reference_ nic/sw/host/driver/. Abrir el archivo nf10_ phy_ conf.c
2. Comentar las lineas 217,219 y 240
3. Recompilar el driver nf10
4. Reiniciar la PC y cargar el nuevo driver

Tras esta corrección el proyecto ReferenceNIC funciona correctamente, comportándose el hardware como una tarjeta de red convencional. A su vez cuando la PC es encendida el hardware se reprograma desde la memoria flash apropiadamente.

Apendice B

Desafíos técnicos encontrados

Durante la ejecución del proyecto, nos enfrentamos a desafíos y en particular de carácter técnico. Dentro de estos desafíos podemos encontrar algunos que se destacan por su complejidad y dificultad para resolverlos y otros que se destacan por su severidad como riesgo tecnológico dentro del proyecto. Por ello consideramos apropiado incluir un apéndice en donde se hace mención a los mismos y a sus respectivas soluciones.

B.1 Problemas con SFPs y Patchcoords

Los transceivers SFP+ son una componente de hardware utilizada en cada puerto físico de la tarjeta NetFPGA en la construcción del dispositivo RAU-Switch. Básicamente se encargan de implementar el mecanismo de acceso a la capa física, en este caso fibra óptica.

Pueden encontrarse transceivers compatibles con patch cords de fibra óptica monomodo, multimodo y compatibles con ambos tipos.

En el desarrollo de este proyecto se trabajó con transceivers SFP+ compatibles con ambos tipos de patch cords, aunque para la construcción de la red prototipo por razones económicas se utilizaron transceivers SFP+ compatibles solamente con patch cords de fibra multimodo.

B.2 Desprogramación del hardware NetFPGA

En el desarrollo de este proyecto se utilizan dos estrategias diferentes para programar el hardware NetFPGA. La primera estrategia, detallada en la guía de ejecución del Test de Producción [32], permite de forma sencilla programar el hardware utilizando la herramienta Impact de la suite Xilinx ISE SDK. Sin embargo este procedimiento no persiste la programación del hardware, por lo que al producirse un corte de corriente en el circuito (apagado de la PC por ejemplo) el mismo pierde la programación. Este efecto puede comprobarse fácilmente utilizando la herramienta Impact para leer el valor de diferentes registros del hardware, los cuales indican el estado del mismo.

La segunda estrategia es la programación PCIe denominada en este trabajo como programación persistente, la cual utiliza la herramienta **pcieprog** de la plataforma NetFPGA para almacenar la programación del hardware en una de las unidades de memoria Flash del equipo. Concretamente este procedimiento implica programar el hardware con un proyecto compatible con la programación PCIe (es necesario que el proyecto implemente un módulo para la configuración del chip FPGA desde una de las memorias Flash) utilizando la herramienta Impact. Luego se almacena en una de las unidades de memoria Flash un archivo binario con la implementación del proyecto con el cual se quiere programar el hardware. Luego al momento del encendido del equipo, el chip FPGA es programado desde el contenido almacenado en una de las memorias Flash.

De esta forma la programación del equipo se almacena de forma persistente, perdurando incluso cuando el equipo es apagado.

B.3 Falta de licencias para suite de Xilinx ISE SDK

La suite de desarrollo Xilinx ISE SDK se compone por un conjunto extenso de herramientas para la progrmación y desarrollo de software utilizando los productos de este fabricante; entre los cuales se encuentra el chip Virtex 5 utilizado en el hardware NetFPGA. Estas herramientas de software son licenciadas lo que significa que para su uso es necesario contar con una licencia de software apropiada.

Cada herramienta particular de la suite Xilinx tiene una licencia particular y se agrupan en diferentes paquetes de licencias. De esta forma dependiendo del uso que se vaya a hacer de esta plataforma, el o los paquetes de licencias que se deben adquirir. Además cada licencia se corresponde con una herramienta para una familia de chips en particular. Por ejemplo para trabajar con las tarjetas NetFPGA-10G se necesitan licencias para el chip Virtex5 TX240T.

Xilinx ofrece un paquete de licencias gratuitas para trabajar con un conjunto reducido de herramientas (entre ellas se incluye el programa Impact) y un paquete de licencias completo de prueba por un tiempo de 30 días compatible con los últimos modelos de chips (Virtex6 y Virtex7). Como dentro de estos paquetes no se incluyen las licencias necesarias para la programación PCIe (programación persistente) una alternativa es adquirir un paquete de licencias con un valor aproximado de USD500 por PC (costo en dolares americanos al mes de Setiembre del año 2014).

En este proyecto, por razones económicas primero se solicitó mediante el programa de soporte a Universidades de Xilinx el paquete de licencias necesario, a la vez que se inició un contacto con el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería (IIE) solicitando orientación en relación a este problema. Finalmente se recibió a través de dicho programa de apoyo a universidades una interesante donación de licencias, posibilitando una real explotación del hardware y la plataforma en su conjunto, aunque esta espera representó un contratiempo en el cronograma inicial del proyecto.

B.4 Falta de driver para cable JTAG Xilinx

Para la programación del hardware NetFPGA se utiliza un cable programador especial con una interfaz de conexión JTAG. Este cable presenta el inconveniente de que sus fabricantes no desarrollaron drivers para la distribución de Linux con la que se trabaja en este proyecto, existiendo drivers oficiales para entornos Windows y una versión no oficial para la distribución Fedora.

Si bien inicialmente esto obligó a programar el hardware en un ambiente Windows para realizar las pruebas de verificación del mismo (Producction Test y RLDRAM Test) finalmente se consigue un driver no oficial[25] también compatible con Ubuntu 12.04, habilitando de esta forma a programar el hardware en la misma PC.

B.5 Problemas técnicos con Open vSwitch

Open vSwitch es uno de los pilares fundamentales de RAU-Switch puesto que es el encargado de implementar el plano de datos de OpenFlow. Durante el desarrollo de este proyecto, en el tiempo empleado a la experimentación con esta herramienta, se genera conocimiento en relación a aspectos importantes sobre la herramienta; los cuales no son triviales y vale la pena mencionar en este apartado. A continuación se enumeran los siguientes:

1. Parte de las funcionalidades de Open vSwitch, están implementadas en modo kernel y otra parte en modo usuario. Estas últimas presentan un nivel de performance muy pobre en comparación a las primeras; y en particular las funcionalidades de MPLS se encuentran solamente disponibles en modo usuario.

2. De acuerdo a la página de preguntas frecuentes, la última versión de Open vSwitch al momento de realizarse este proyecto, garantiza soporte a las operaciones de MATCH, PUSH y POP de hasta tres etiquetas MPLS apiladas, así como su posterior procesamiento acorde al pipe de OpenFlow. Todas estas funcionalidades además se implementan en modo usuario.

   Por otro lado, de acuerdo a las notas de liberación de la versión 2.3.1, solamente se garantiza el soporte a las funciones de MATCH, PUSH y POP de MPLS para un único nivel de etiquetas; esto es una sola etiqueta MPLS. Luego se garantiza además el posterior procesamiento acorde al pipe OpenFlow. Esta información no sólo es contradictoria, sino que además no se condice con la realidad. Experimentalmente se prueba que si bien la versión 2.3.1 implementa correctamente las operaciones de MATCH y PUSH de una única etiqueta MPLS, la operación POP no se encuentra implementada para esta versión, rompiendo además el pipe de procesamiento OpenFlow.

   Afortunadamente éste comportamiento había sido detectado y reportado por otros usuarios de Open vSwitch como un BUG, resolviéndose posteriormente en la versión de desarrollo [43].

3. Trabajando con la versión de desarrollo en el repositorio de código fuente de Open vSwitch [43] (rama master en el repositorio Git), se comprueba experimentalmente el soporte para las operaciones de Match, Push y Pop de hasta 3 etiquetas, así como el posterior procesamiento del paquete según el pipe de OpenFlow.

4. Puertos OpenFlow con direcciones IP:

   Para lograr que cada interfaz física de la tarjeta NetFPGA se comporte tanto como un puerto OpenFlow, como una interfaz IP, es necesario:

   (a) Crear un bridge con Open vSwitch y agregar cada interaz física como un puerto OpenFlow al mismo.

   (b) Asignar una dirección IP a la propia intefaz física (por ejemplo utilizando el comando ifconfig).

   El comportamiento deseado por una interfaz híbrida IP/OpenFlow en el prototipo sería el que se muestra en la figura B.1 (mitad izquierda). El paquete ingresa al router através de la interfaz física **nf0** y de ahí en más el procesamiento del mismo es delegado al módulo de Open vSwitch en el kernel de Linux. Aquí el paquete es procesado y tratado en consecuencia a lo que indica la tabla de flujos en Open vSwitch. En el prototipo, para este paquete existen dos alternativas: (1) se procesa el contenido del mismo y se reenvía por otra interfaz acorde a la regla correspondiente, (2) es contemplado por una regla con la acción *NORMAL* y por tanto es procesado como lo haría un switch legado (en este caso como lo procesaría el kernel de linux).

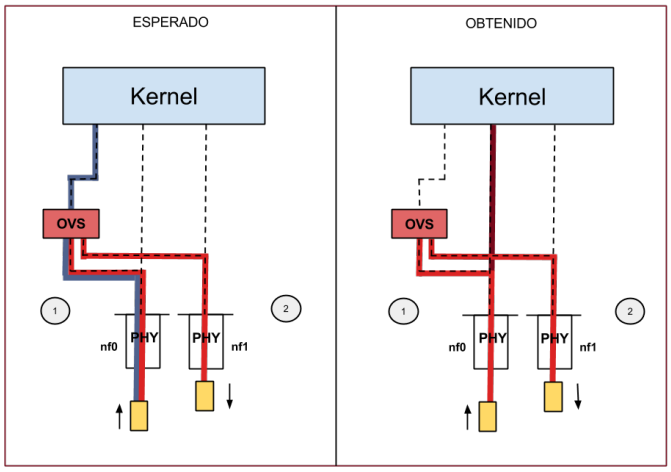

Fig. B.1 Funcionamiento de puertos Open vSwitch

No obstante, el procesamiento del paquete al ingresar por la interfaz física **nf0** difiere del comportamiento deseado (ver mitad derecha de la imagen). El paquete es procesado por Open

vSwitch como se describió anteriormente, pero también es enviado para su procesamiento en el kernel de Linux.

Esto implica que el switch se comporte como una mezcla de un switch OpenFlow con una PC Linux normal, ocasionando un tratamiento de paquetes en el prototipo diferente al especificado por las reglas OpenFlow calculadas.

Para solucionar este problema se utilizan interfaces virtuales de la siguiente forma:

(a) Se crea una interfaz virtual por cada interfaz física y se conecta a través de dichas interfaces, openvswitch con el kernel.

(b) La dirección IP que antes se asignaba a cada interfaz física ahora se le asigna a su correspondiente interfaz virtual.

(c) Se deja la interfaz física sin dirección IP y de esta forma se asegura que ningún paquete que arribe a dicha interfaz será procesado por el kernel, ya que nunca coincidirá la dirección IP que contenga un paquete con la dirección de la interfaz.

(d) Se crean entradas de prioridad mínima en la tabla de flujos de openvswitch que indiquen que todo lo que entre por la interfaz física se envíe a través su correspondiente interfaz virtual y viceversa.

(e) De esta forma si no hay un flujo de mayor prioridad para procesar los paquetes que entran por determinada interfaz física, estos serán enviados al kernel mediante la interfaz virtual, procesados por este para luego ser enviado por la interfaz virtual a la física y seguir su camino.

(f) Estos pasos aseguran que todo el tráfico sea procesado por las tablas de Open vSwitch, y serán procesados por el kernel solo en caso de ser necesario.

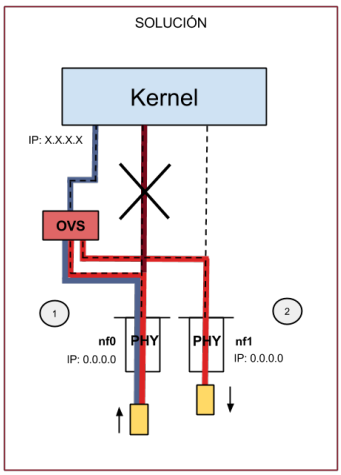

Fig. B.2 Puertos Open vSwitch - Solución propuesta

B.6 MPLS Linux y Quagga LDP

MPLS Linux es un proyecto orientado a dotar de funcionalidades de MPLS al Kernel de Linux, mediante un parche con el cual se extiende al mismo y el cual tras recompilarse queda pronto para brindar soporte nativo al protocolo MPLS.

Existen principalmente dos versiones: el proyecto original[72] encabezado por James R. Leu el cual data de hace más de 15 años y no cuenta con contribuciones importantes desde hace aproximadamente 9 y otro proyecto[76] encabezado por Igor Maravick (quien trabajó en el proyecto anterior) que data del año 2010 y surge como una actualización del proyecto original.

El primer proyecto fue pensado como un parche para el kernel de Linux versión 2.6.3.16 (tener en cuenta que la versión actual es la 4.1.1). Adicionalmente se desarrolló una extensión de la suite Quagga[73] que implementa el algoritmo LDP para la construcción de redes IP/MPLS.

Por otro lado, el proyecto de Igor Maravick es un kernel de Linux versión 3.9.0-rc3 con las modificaciones necesarias para soportar el protocolo MPLS ya incorporadas. Adicionalmente existe una nueva extension de la suite Quagga que implementa el protocolo LDP [96] y es compatible con esta versión de MPLS-Linux.

En el contexto de este proyecto se trabajó con ambas versiones de MPLS-Linux y las respectivas versiones de Quagga-LDP. Para la versión original de MPLS-Linux se probó compilar el parche con diferentes versiones de kernel, entre ellas las versiones 2.6.32.16 y 2.6.32.65 no pudiendo así configurar correctamente el ambiente de trabajo. En particular estas versiones son tan antiguas que gran parte de las herramientas utilizadas para la construcción de RAU-Switch no son compatibles. Adicionalmente se prueba compilar el parche para la versión de kernel 3.11.0-15 generic (versión que trae Ubuntu 12.01) pero tampoco se obtienen buenos resultados.

En relación a la versión de Igor Maravik se logró configurarla correctamente, así como la extensión Quagga-LDP. No obstante al momento de ejecutar dichas herramientas se detectaron problemas en la interacción entre las mismas; mediante los logs de la herramienta Quagga se detectaron errores al momento de insertar entradas en las tablas MPLS del kernel.

Finalmente por la falta de información en relación a estas componentes, ya que sobre el proyecto original existe muy poca documentación mientras que para el proyecto de Igor Maravik no existe la misma, se decidió descartar esta alternativa para implementar el algoritmo LDP.

Apendice C

Clasificación de tráfico utilizando cabezales OpenFlow

En este apéndice se detallan de los atributos presentes en el cabezal OpenFlow versión 1.3.1 cuáles pueden ser utilizados para la definición de reglas y acciones de un flujo. Esta información es de carácter experimental y está acotada a la compatibilidad de Open vSwitch; es decir, aquellos atributos no soportados por Open vSwitch o que no se consiguió documentación para entender su utilización, no son considerados.

C.1 Cabezal OpenFlow versión 1.3.1

```
of_v13_match_fields:

    in_port          # Switch input port.
    in_phy_port      # Switch physical input port.
    metadata         # Metadata passed between tables.
    eth_dst          # Ethernet destination address.
    eth_src          # Ethernet source address.
    eth_type         # Ethernet frame type.
    vlan_vID         # VLAN id.
    vlan_PCP         # VLAN priority.
    IP_dscp          # IP DSCP (6 bits in ToS field).
    IP_ecn           # IP ECN (2 bits in ToS field).
    IP_proto         # IP protocol.
    IPv4_src         # IPv4 source address.
    IPv4_dst         # IPv4 destination address.
    TCP_src          # TCP source port.
    TCP_dst          # TCP destination port.
    UDP_src          # UDP source port.
    UDP_dst          # UDP destination port.
    SCTP_src         # SCTP source port.
    SCTP_dst         # SCTP destination port.
    ICMPv4_type      # ICMP type.
```

```
23      ICMPv4_code      # ICMP code.
24      ARP_op           # ARP opcode.
25      ARP_spa          # ARP source IPv4 address.
26      ARP_tpa          # ARP target IPv4 address.
27      ARP_sha          # ARP source hardware address.
28      ARP_tha          # ARP target hardware address.
29      IPv6_src         # IPv6 source address.
30      IPv6_dst         # IPv6 destination address.
31      IPv6_flabel      # IPv6 Flow Label
32      ICMPv6_type      # ICMPv6 type.
33      ICMPv6_code      # ICMPv6 code.
34      IPv6_nd_target   # Target address for ND.
35      IPv6_nd_ssl      # Source link-layer for ND.
36      IPv6_nd_tll      # Target link-layer for ND.
37      MPLS_label       # MPLS label.
38      MPLS_tc          # MPLS TC.
39      MPLS_bos         # MPLS BoS bit.
40      PBB_is_id        # PBB I-SID. */
41      tunnel_id        # Logical Port Metadata.
42      IPv6_txhdr       # IPv6 Extension Header pseudo-field
```

Estos atributos fueron estudiados uno por uno y se llega a las dos listas presentadas en las siguientes secciones.

C.2 Sintaxis de un flujo en Open vSwitch

La sintaxis del comando ovs-ofctl utilizado para agregar un flujo en la tabla de Open vSwitch es la siguiente:

```
ovs-ofctl add-flow <bridge> <flow>
```

Los parámetros de este comando son el nombre del brdige (<bridge>) y el flujo definido. La sintaxis de un flujo es la siguiente:

```
attr1_name=<value>,attr2_name=<value>,..  actions=<action1>,<action2>,...
```

Primero tiene una lista de pares atributo valor separados por una "," las cuales definen la regla de un flujo; es decir, los campos del cabezal OpenFlow utilizados para aparear tráfico. Un ejemplo común es es utilizar el numero de puerto OpenFlow de entrada, in_port=1 por ejemplo.

La segunda parte de un flujo es la lista de acciones precedida por la palabra **actions**. Un ejemplo común es reenviar paquetes por un puerto particular, por ejemplo out_port:2.

C.3 OpenFlow v1.3.1 atributos soportados para la definición de reglas

A continuación se detallan los atributos soportados para la definición de reglas en un flujo, lo cual permite la construcción de una FEC para clasificar tráfico.

- **in_port**: Es utilizado para discriminar por el puerto de entrada en un switch
- **metadata**: Es utilizado entre otras cosas para compartir información entre varios flujos en diferentes tablas OpenFlow
- **eth_dst**: Permite a un flujo seleccionar tráfico distinguiendo por la dirección destino de capa 2 (dirección MAC). El valor utilizado se compone de 6 pares de hexadecimales.

```
ovs-ofctl add-flow <bridge> dl_dst=<mac>,actions=<action>
```

- **eth_src**: Permite a un flujo selecionar tráfico distinguiendo por dirección origen de capa 2 (dirección MAC). El valor utilizado se compone de 6 pares de hexadecimales.

```
ovs-ofctl add-flow <bridge> dl_src=<mac>,actions=<action>
```

- **dl_type**: Permite a un flujo selecionar tráfico distinguiendo por el tipo de ethertype de un paquete. Se utiliza como valor el código hexadecimal de cada protocolo.

```
ovs-ofctl add-flow <bridge> dl_type=<ethernet type>, actions=<action>
```

- **vlan_vID**: Permite a un flujo selecionar tráfico distinguiendo por el ID de VLAN. El valor de vlan_id es un entero en el rango 0-4095.

```
ovs-ofctl add-flow <bridge> dl_vlan=<vlan_id>,actions=<action>
```

- **vlan_PCP**: Permite a un flujo selecionar tráfico distinguiendo por el valor de prioridad de VLAN. El valor dl_vlan_pcp es un entero en el rango 0-7.

```
ovs-ofctl add-flow <bridge> dl_vlan_pcp=<value>,actions=<action>
```

- **IP_dscp**: Permite a un flujo selecionar tráfico distinguiendo por los campos IP TOS/DHCP o IPv6 traffic class field en un paquete IP, dependiendo del dl_type utilizado. Se utiliza junto con los valores de dl_type 0x0800 para IPv4 y 0x86DD para IPv6. Para otros valores de dl_type el valor del campo nw_tos es ignorado. El atributo nw_tos toma valores en el rango 0-255.

```
ovs-ofctl add-flow <bridge> dl_type=<ethernet type>, nw_tos=<tos>,
                            actions=<action>
```

- **IP_ecn**: Permite a un flujo selecionar tráfico distinguiendo por el valor de los bits ECN (Explicit Congestion) en un paquete IP. Cuando el valor del campo dl_type es 0x0800 aparea los bits ecn en el campo IP TOS IPv4 mientras que cuando se utiliza el valor 0x86DD aparea los campos de clase de tráfico (trafic class fields) de IPv6. Si se utiliza otro valor de dl_type estos bits son ignorados. El campo nw_ecn toma valores en el rango 0-3.

```
ovs-ofctl add-flow <bridge> dl_type=<ethernet type>,
          nw_ecn=<ecn>,actions=<action>
```

- **IP_proto**: Se utiliza para indicar el protocolo IP utilizado en la definición de un flujo. Usualmente se utiliza en conjunto con otros atributos como tp_src y tp_dst. Cuando se utiliza el valor del campo dl_type debe ser o 0x0800 en el caso de IPv4 o 0x86DD para el caso IPv6. El campo nw_proto es el número de protocolo IP (valores en el rango 0-255).

```
ovs-ofctl add-flow <bridge> dl_type=<ethernet
   type>,nw_proto=<proto>,actions=<action>
```

- **IPv4_src**: Permite a un flujo seleccionar tráfico distinguiendo por la dirección origen IPv4. Se debe indicar para el campo dl_type el valor 0x0800.

```
ovs-ofctl add-flow <bridge> dl_type=<ethernet type>,
       nw_src=ip[/netmask],actions=<action>
```

- **IPv4_dst**: Permite a un flujo seleccionar tráfico distinguiendo por la dirección destino IPv4. Se debe indicar para el campo dl_type el valor 0x0800.

```
ovs-ofctl add-flow <bridge> dl_type=<ethernet type>,
       nw_dst=ip[/netmask],actions=<action>
```

- **TCP_src**: Permite a un flujo seleccionar tráfico distinguiendo por el número de puerto TCP origen. Se debe indicar para el campo dl_type el valor 0x0800 (IPv4) y para el campo nw_proto el valor 6 (TCP). El campo tp_src toma valores en el rango 0-65535.

```
ovs-ofctl add-flow <bridge> dl_type=0x0800,
 nw_proto=6,tp_src=<port>,actions=<action>
```

  De forma análoga se utilizan los campos UDP_src y SCP_src para clasificar tráfico por los puertos origen UDP y SCTP.

```
ovs-ofctl add-flow <bridge> dl_type=0x0800,
 nw_proto=17,tp_src=<port>,actions=<action>

ovs-ofctl add-flow <bridge> dl_type=0x0800,
nw_proto=132,tp_src=<port>,actions=<action>
```

- **TCP_dst**: Permite a un flujo seleccionar tráfico distinguiendo por el número de puerto TCP destino. Se debe indicar para el campo dl_type el valor 0x0800 (IPv4) y para el campo nw_proto el valor 6 (TCP). El campo tp_dst toma valores en el rango 0-65535.

```
ovs-ofctl add-flow <bridge> dl_type=0x0800,
 nw_proto=6,tp_dst=<port>,actions=<action>
```

  De forma análoga se utilizan los campos UDP_dst y SCP_dst para clasificar tráfico por los puertos destino UDP y SCTP.

```
ovs-ofctl add-flow <bridge> dl_type=0x0800,
 nw_proto=17,tp_dst=<port>,actions=<action>

ovs-ofctl add-flow <bridge> dl_type=0x0800,
nw_proto=132,tp_dst=<port>,actions=<action>
```

- **ICMPv4_type**: Permite a un flujo seleccionar tráfico distinguiendo por tipo de paquete ICMPv4. Open vSwitch utiliza el mismo campo para la definición del tipo ICMPv4 o ICMPv6 en un flujo (icmp_type). Se debe indicar en el campo dl_type el valor 0x0800(IPv4) o 0x86DD (IPv6) y en el campo nw_proto el valor 1 (ICMPv4) o 58 (ICMPv6). El campo icmp_type toma valores en el rango 0-255.

```
ovs-ofctl add-flow<bridge> dl_type=<ethernettype>,nw_proto=1,
            icmp_type=<type>,actions=<action>
```

- **ICMPv4_code**: Permite a un flujo seleccionar tráfico distinguiendo por código de paquete ICMPv4. Open vSwitch utiliza el mismo campo para la definición del código ICMPv4 o ICMPv6 en un flujo (icmp_code). Se debe indicar en el campo dl_type el valor 0x0800 para IPv4 o 0x86DD para IPv6 y en el campo nw_proto el valor 1 para ICMPv4 o 58 para ICMPv6 respectivamente. El campo icmp_code toma valores en el rango 0-255.

```
ovs-ofctl add-flow<bridge> dl_type=<ethernet type>,
 nw_proto=[1|58],icmp_code=<type>,actions=<action>
```

- **IPv6_src**: Permite a un flujo seleccionar tráfico distinguiendo por la dirección origen IPv6. Se debe indicar para el campo dl_type el valor 0x86DD.

```
ovs-ofctl add-flow<bridge> dl_type=0x86DD,
ipv6_src=<ipv6[/netmask]>,actions=<action>
```

- **IPv6_dst**: Permite a un flujo seleccionar tráfico distinguiendo por la dirección destino IPv6. Se debe indicar para el campo dl_type el valor 0x86DD.

```
ovs-ofctl add-flow<bridge> dl_type=0x86DD,
ipv6_dst=<ipv6[/netmask]>,actions=<action>
```

- **ICMPv6_type**: Ver campo ICMP_type por ejemplo acerca de la utilización de este campo.
- **ICMPv6_code**: Ver campo ICMP_type por ejemplo acerca de la utilización de este campo.
- **MPLS_label**: Permite a un flujo seleccionar tráfico distinguiendo por el valor de etiqueta MPLS. Se debe indicar para el campo dl_type el valor 0x8847 o 0x8848. El campo label toma valores en el rango 0-1048575.

```
ovs-ofctl add-flow<bridge> dl_type=[0x8847|0x8848]ovs-ofctl,
mpls_label=<label>,actions=<action>
```

- **MPLS_tc**: Permite a un flujo seleccionar tráfico distinguiendo por el valor del campo TC en el cabezal MPLS (para uso experimental). Se debe indicar para el campo dl_type el valor 0x8847 o 0x8848. El campo mpls_tc toma valores en el rango 0-7.

```
ovs-ofctl add-flow<bridge>
dl_type=[0x8847|0x8848],mpls_tc=<tc>,actions=<action>
```

- **MPLS_bos**: Permite a un flujo seleccionar tráfico distinguiendo por el valor del campo BOS (Bottom of the stack) en el cabezal MPLS. Se debe indicar para el campo dl_type el valor 0x8847 o 0x8848. El campo mpls_bos toma el valor 1 cuando el paquete contiene una única etiqueta MPLS y 0 cuando contiene más de una etiqueta en el stack.

```
ovs-ofctl add-flow<bridge> dl_type=[0x8847|0x8848],
mpls_bos=<mpls_bos>,actions=<action>
```

C.4 OpenFlow v1.3.1 atributos soportados para definir acciones

A continuación se detallan los atributos soportados para la definición de acciones en un flujo lo cual permite la manipulación de tráfico.

- **eth_src**: Permite modificar el valor de dirección origen de capa 2 (dirección MAC).

```
ovs-ofctl add-flow<bridge><match-field>actions=mod_dl_src:<mac>
```

- **eth_dst**: Permite modificar el valor de dirección destino de capa 2 (dirección MAC).

```
ovs-ofctl add-flow<bridge><match-field>actions=mod_dl_dst:<mac>
```

- **vlan_vID**: Permite modificar el valor del campo VLAN ID (identificador de VLAN).

```
ovs-ofctl add-flow <bridge> <match-field>
actions=push_vlan:<ethertype>,set_field:<value>-\>vlan_vid,output:<port>
```

- **vlan_PCP**: Permite modificar el valor del campo VLAN PCP (prioridad de VLAN).

```
ovs-ofctl add-flow<bridge><match-field>
    actions=mod_vlan_pcp:<vlan_pcp>
```

- **IP_dscp**: Permite modificar el valor de los bits TOS en el un cabezal IP. Esta acción no modifica los 2 bits más bajos del campo, los cuales representan los bits ECN.

```
ovs-ofctl add-flow<bridge><match-field>actions=mod_nw_tos:<tos>
```

- **IPv4_src**: Permite modificar el valor de dirección origen de capa 3 (IP).

```
ovs-ofctl add-flow<bridge><match-field>actions=mod_nw_src:<ip>
```

- **IPv4_dst**: Permite modificar el valor de dirección destino de capa 3 (IP).

```
ovs-ofctl add-flow<bridge><match-field>actions=mod_nw_dst:<ip>
```

- **TCP_src**: Permite modificar el valor de puerto TCP origen de un paquete. Debe seleccionarse mediante la regla del flujo tráfico TCP para distinguirlo explicitamente de otro tipo de tráfico.

```
ovs-ofctl add-flow<bridge><match-field>actions=mod_tp_src:<port>
```

  De forma similar utilizando la misma primitiva y los campos **UDP_src** y **SCTP_src** se modifican puerto origen UDP y SCTP.

- **TCP_dst**: Permite modificar el valor de puerto TCP destino de un paquete. Debe seleccionarse mediante la regla del flujo tráfico TCP para distinguirlo explicitamente de otro tipo de tráfico.

```
ovs-ofctl add-flow<bridge><match-field>actions=mod_tp_dst:<port>
```

De forma similar utilizando la misma primitiva y los campos **UDP_dst** y **SCTP_dst** se modifica el puerto destino UDP y SCTP.

- **MPLS_label**: Permite manipular el valor de una etiqueta en un cabezal MPLS. Las primitivas existentes son PUSH y POP.

```
ovs-ofctl add-flow <bridge> <match-field> dl_type=[0x8847 |
0x8848],actions=push_mpls:<ethertype>,set_field:<value>-\>mpls_label

ovs-ofctl add-flow<bridge> <match-field>
 actions=set_field:<label>-\>mpls_label

ovs-ofctl add-flow<bridge> <match-field> actions=pop_mpls:<ethertype>
```

- **MPLS_tc**: Permite modificar el valor de los bits experimentales en el cabezal MPLS.

```
ovs-ofctl add-flow<bridge> <match-field>
  actions=set_field:<label>-\>mpls_tc
```

Apendice D

Archivos de configuración

En el presente anexo se incluyen los scripts y archivos de configuración de las principales herramientas utilizadas en la construcción del prototipo.

Los siguientes archivos son tomados del PC identificado en el laboratorio de pruebas como **Galois**; los archivos de conifiguración de los restantes tres nodos son similares.

D.1 Archivos de configuración Quagga

Para la configuración de Quagga, se utilizan tres archivos: el archivo **daemons** en donde se indica la configuración de demonios (en el caso del proyecto se activa el demonio zebra y ospfd), el archivo **zebra.conf** y el archivo **ospfd.conf**.

Archivo **daemons**:

```
#
# Entries are in the format: <daemon>=(yes|no|priority)
#   0, "no"  = disabled
#   1, "yes" = highest priority
#   2 .. 10  = lower priorities
# Read /usr/share/doc/quagga/README.Debian for details.
#
# Sample configurations for these daemons can be found in
# /usr/share/doc/quagga/examples/.
#

vtysh=yes
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
isisd=no
```

Archivo **zebra.conf**:

```
!
! Zebra configuration saved from vty
!   2015/02/05 21:26:43
!
hostname Router
password zebra
enable password zebra
log file /var/log/zebra.log
!
debug zebra kernel
!
!
interface eth0
 ipv6 nd suppress-ra
!
interface lo
!
interface mpls0
 !ipv6 nd suppress-ra
!
interface vnf0
 ipv6 nd suppress-ra
!
interface vnf1
 ipv6 nd suppress-ra
!
interface vnf2
 ipv6 nd suppress-ra
!
interface vnf3
 ipv6 nd suppress-ra
!
ip forwarding
!
line vty
!
```

Archivo **ospfd.conf**:

```
! -*- ospf -*-
!
! OSPFd sample configuration file
!
!
hostname ospfd
password zebra
!enable password please-set-at-here
!
interface lo
!
interface eth0
  ip ospf cost 65535
!
interface vnf0
  ip ospf cost 1
!
interface vnf1
  ip ospf cost 3
!
interface vnf2
  ip ospf cost 1
!
interface vnf3
  ip ospf cost 1
!

router ospf
 ospf router-id 192.168.1.11
 network 10.10.1.0/24 area 0.0.0.0
 network 10.10.4.0/24 area 0.0.0.0
 network 10.10.5.0/24 area 0.0.0.0
 network 192.168.1.0/24 area 0.0.0.0
!
log stdout
!
```

D.2 Archivos de configuración de Open vSwitch

Script de configuración inicial de ovs:

```
1  ##################################################
2  # Configuracion de Open vSwitch para nodo Galois  #
3  ##################################################
4  #
5  # Declara bridge, interfaces
6  ovs-vsctl add-br bral
7  ovs-vsctl set bridge bral datapath_type=netdev
8  ovs-vsctl set bridge bral protocols=OpenFlow13
9  ovs-vsctl set bridge bral other-config:datapath-id=000000000000000B
10 ovs-vsctl add-port bral eth1
11 ovs-vsctl add-port bral nf0
12 ovs-vsctl add-port bral nf1
13 ovs-vsctl add-port bral nf2
14 #
15 # Setea por defecto el modo de falla como seguro
16 ovs-vsctl set-fail-mode bral secure
17 #
18 # Interfaces virtuales
19 ovs-vsctl add-port bral veth1
20 -- set interface veth1 type=internal
21 ovs-vsctl add-port bral vnf0
22 -- set interface vnf0 type=internal
23 ovs-vsctl add-port bral vnf1
24 -- set interface vnf1 type=internal
25 ovs-vsctl add-port bral vnf2
26 -- set interface vnf2 type=internal
27 #
28 # Conexion con el controlador
29 ovs-vsctl set-controller bral tcp:192.168.1.10:6633
30 #
```

Script para iniciar OVS:

```
ovsdb-server /usr/local/etc/openvswitch/conf.db \
--remote=punix:/usr/local/var/run/openvswitch/db.sock \
--remote=db:Open_vSwitch,Open_vSwitch,manager_options \
--private-key=db:Open_vSwitch,SSL,private_key \
--certificate=db:Open_vSwitch,SSL,certificate \
--bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
--pidfile --detach
--log-file=/var/log/openvswitch/openvswitch.log

ovs-vsctl --no-wait init
ovs-vswitchd --pidfile --detach
--log-file=/var/log/openvswitch/vswitch.log
ovs-vsctl show
```

Script de instalación de flujos para bridging entre interfaces físicas y virtuales:

```
ovs-ofctl -O openflow13 add-flow bral table=0,priority=0,
hard_timeout=0,idle_timeout=0, arp,in_port=1,actions=output:5
ovs-ofctl -O openflow13 add-flow bral table=0,priority=0,
hard_timeout=0,idle_timeout=0, arp,in_port=5,actions=output:1

ovs-ofctl -O openflow13 add-flow bral table=0,priority=0,
hard_timeout=0,idle_timeout=0,in_port=2,actions=output:6
ovs-ofctl -O openflow13 add-flow bral table=0,priority=0,
hard_timeout=0,idle_timeout=0,in_port=6,actions=output:2

ovs-ofctl -O openflow13 add-flow bral table=0,priority=0,
hard_timeout=0,idle_timeout=0,in_port=3,actions=output:7
ovs-ofctl -O openflow13 add-flow bral table=0,priority=0,
hard_timeout=0,idle_timeout=0,in_port=7,actions=output:3

ovs-ofctl -O openflow13 add-flow bral table=0,priority=0,
hard_timeout=0,idle_timeout=0,in_port=4,actions=output:8
ovs-ofctl -O openflow13 add-flow bral table=0,priority=0,
hard_timeout=0,idle_timeout=0,in_port=8,actions=output:4
```

D.3 Otros scripts de configuración

Configuración de interfaces físicas:

```
1
2  ########################################################
3  # Configuracion de interfaces de Red para Host Galois #
4  ########################################################
5  #
6  # Interfaz para la LAN de gestion
7  ifconfig eth0 192.168.1.11 netmask 255.255.255.0 up
8  #
9  # las interfaces fisicas no tienen direccion IP
10 ifconfig eth1 0.0.0.0 up
11 ifconfig nf0 0.0.0.0 up
12 ifconfig nf1 0.0.0.0 up
13 ifconfig nf2 0.0.0.0 up
```

Configuración de interfaces virtuales:

```
1 # Interfaces virtuales Galois
2 ifconfig veth1 192.168.11.1 netmask 255.255.255.0 up
3 ifconfig vnf0 10.10.1.1 netmask 255.255.255.0 up
4 ifconfig vnf1 10.10.5.1 netmask 255.255.255.0 up
5 ifconfig vnf2 10.10.4.1 netmask 255.255.255.0 up
```

Apendice E

Ejecución de RAUFlow

En el presente anexo se incluyen una guía para la ejecución de la aplicación RAUFlow y una descripción de la interfaz web.

Para ejecutar la aplicación es necesario iniciar el controlador Ryu y ejecutar las cuatro aplicaciones Ryu incluidas en la arquitectura. Para ello se ejecuta el siguiente comando en el directorio Proyecto/ryu-master dentro de la estructura de directorios del proyecto.

```
./bin/ryu run -observe-links ryu/app/proyecto/businessLogic/RAUFlowApp.py
```

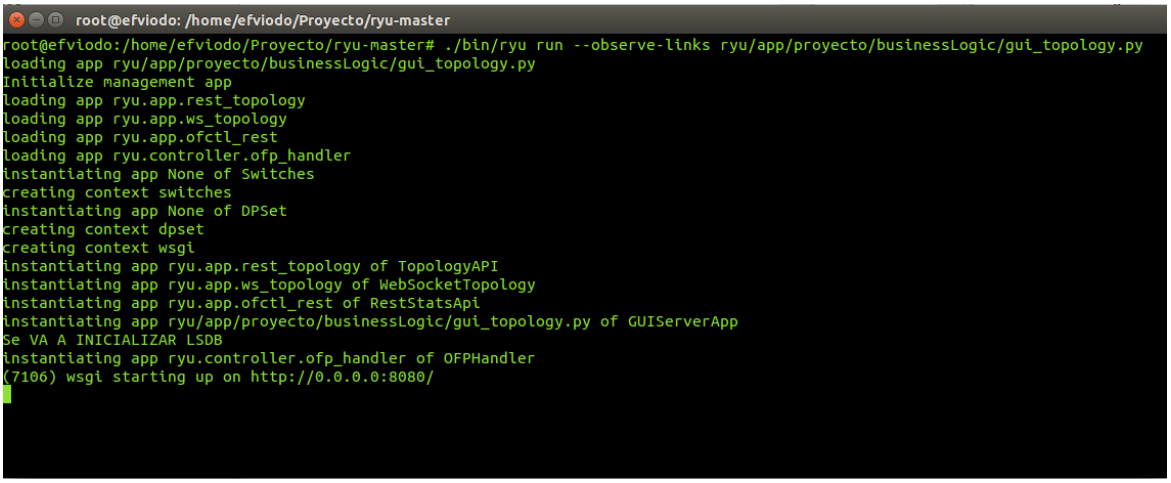

Fig. E.1 Ejecución RAUFlow

Este comando levanta el controlador Ryu y le pasa como parámetro la aplicación RAUFlowApp la cual internamente instancia las otras tres aplicaciones Ryu.

Por otro lado la aplicación instancia la API REST de servicios, por defecto en el puerto 8080 en la dirección localhost. Ambos valores pueden configurarse desde el archivo wsgi.py.

Para acceder a la aplicación abrir desde un navegador web la siguiente dirección:

http://localhost:8080/

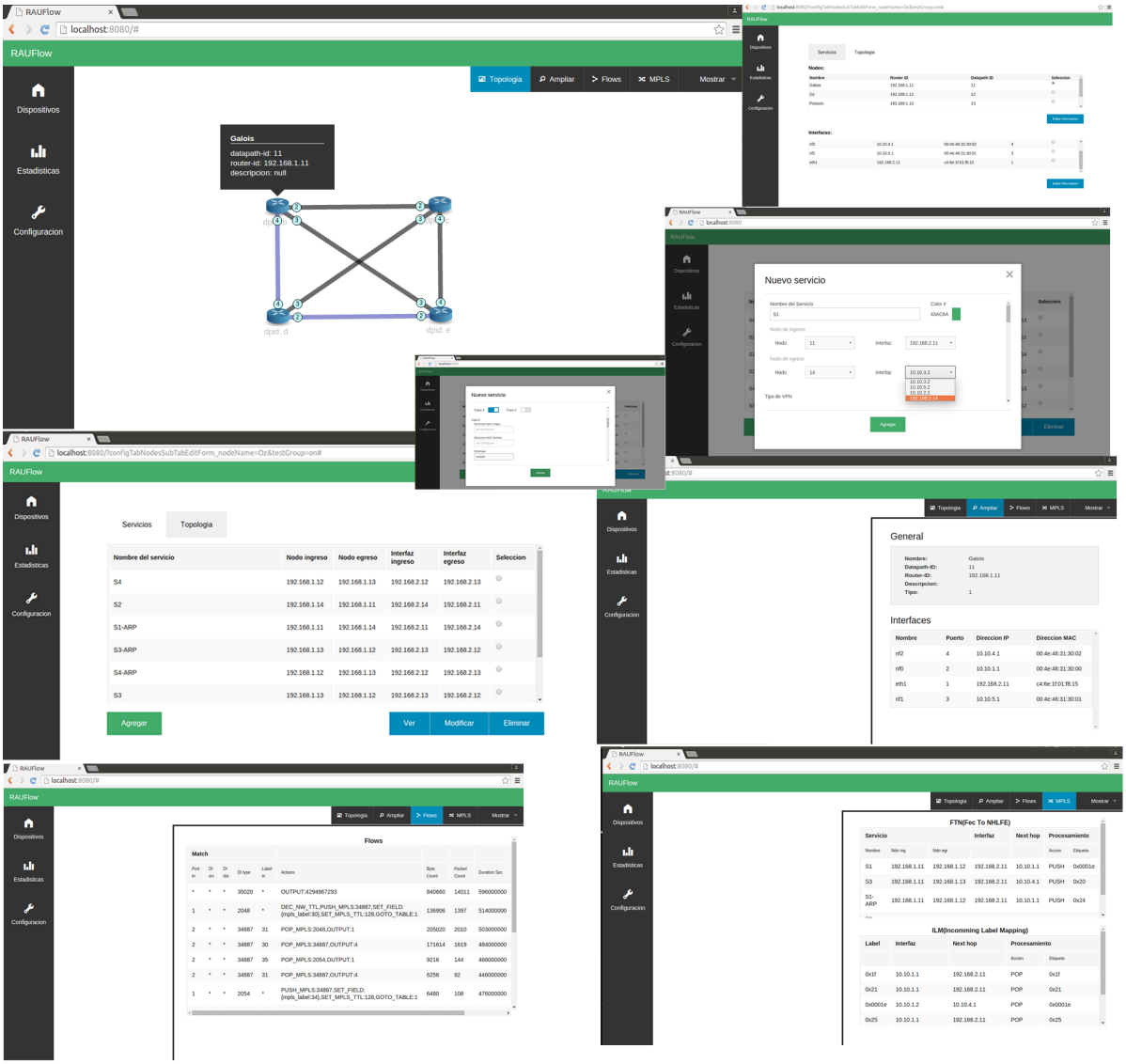

Fig. E.2 Capturas de interfaz gráfica de RAUFlow

Apendice F

Detalles adicionales sobre pruebas realizadas

En el presente anexo se incluyen detalles técnicos acerca de las pruebas realizadas sobre el Laboratorio de experimentación con el fin de validar el prototipo. El mismo está dividido en secciones complementarias a diferentes secciones del capítulo 6; por ello se recomienda leer primero la correspondiente sección en el capítulo 6 y luego complementar la lectura con este capítulo.

F.1 Verificación del algoritmo de ruteo en VPN de capa 3 escenario 1

En las figuras F.1 - F.4 se muestran las tablas de flujos asociadas a cada nodo del laboratorio, capturadas con el comando **dump-flows** de la herramienta Open vSwitch.

Asumiéndose la notación (n,i) para referirse a un enlace, donde n indica nodo origen e i interfaz de reenvío en n para el próximo salto, entonces $< (n_1,i_1),\ldots,(n_k,i_k) >$ puede usarse para denotar un camino en la red laboratorio. De esta forma se pueden comparar fácilmente los caminos teóricos con los calculados. Tomando como ejemplo el caso del servicio S3, el camino teórico puede denotarse de la siguiente forma:

$$< (Galois, nf_2) >$$

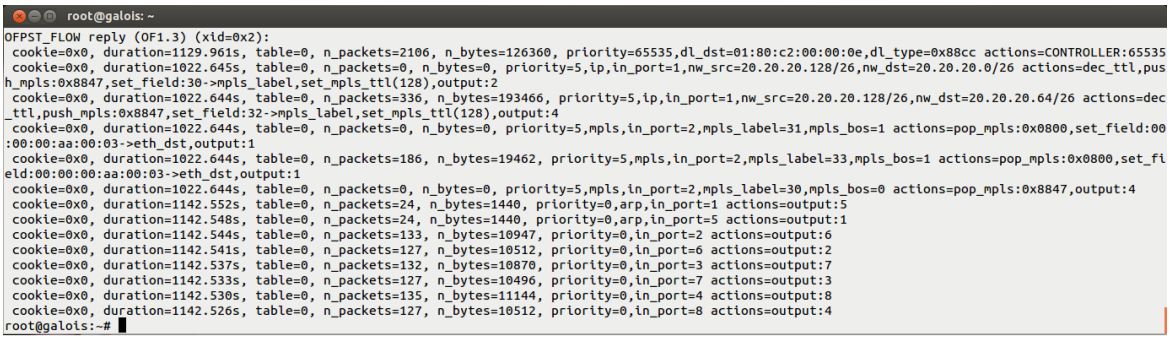

Fig. F.1 Tabla de flujos ovs - Galois

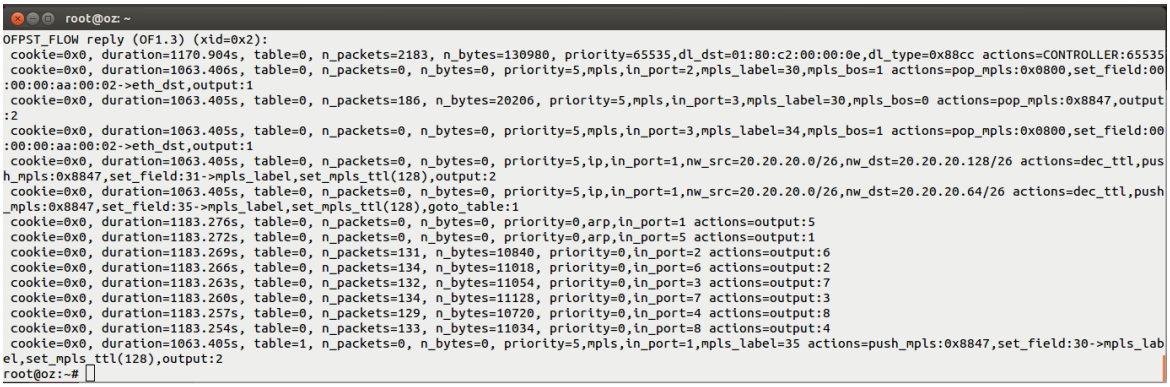

Fig. F.2 Tabla de flujos ovs - Oz

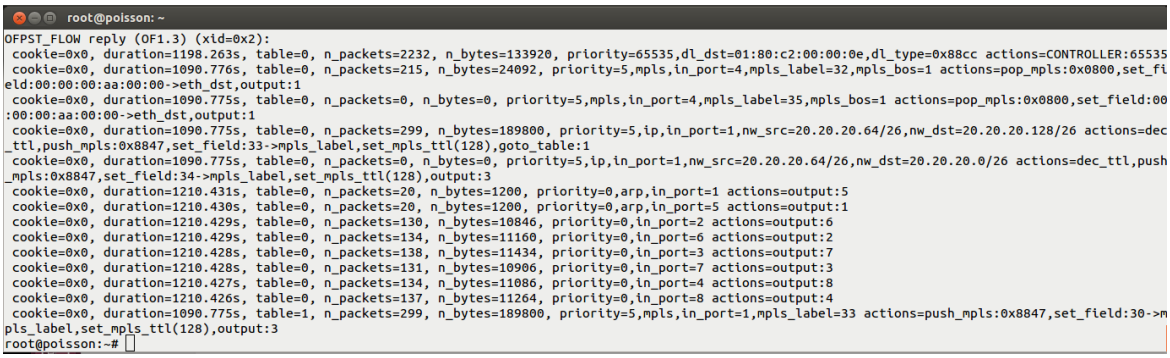

Fig. F.3 Tabla de flujos ovs - Poisson

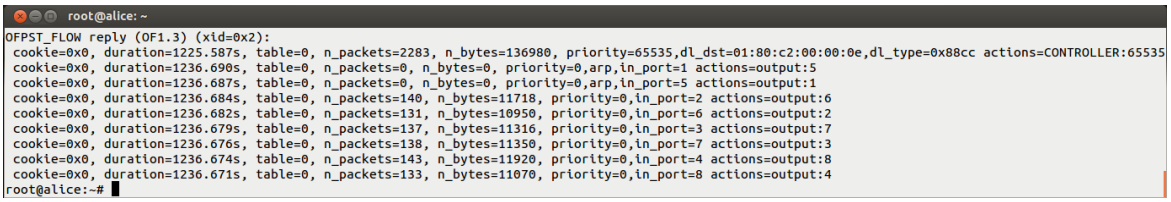

Fig. F.4 Tabla de flujos ovs - Alice

En otras palabras todo tráfico IP con origen en la subred A y destino a la subred B, es encaminado a través de la interfaz nf_2 en el nodo de ingreso *Galois*. Luego en el nodo de egreso *Poisson* es reenviado por la interfaz externa asociada al servicio.

Analizando las tablas de flujos en los nodos *Galois* y *Poisson*, puede comprobarse fácilmente la correspondencia entre el camino calculado y el camino teórico. Por un lado en la tabla de flujos del nodo *Galois* se tiene el siguiente flujo:

```
cookie=0.0, duration=1022.644s, table=0, n_packets=336,
     n_bytes=193466, priority=5,ip,in_port=1,
    nw_src=20.20.20.128/26,nw_dst=20.20.20.64/26
actions=dec_ttl,push_mpls:0x8847,set_field:32->mpls_label,
                    set_mpls_ttl(128),output:4
```

Fig. F.5 Flujo 1

Este flujo toma todo paquete recibido por el puerto OpenFlow identificado con el número 1 (interfaz eth1), numeración IP origen 20.20.20.128/26 (subred A) y destino 20.20.20.64/26 (subred B), decrementa el TTL del paquete, coloca un cabezal MPLS con la etiqueta 32 y finalmente lo reenvía por el puerto OpenFlow número 4 (interfaz nf2). Observar que el valor de la etiqueta MPLS colocado es el utilizado para identificar el servicio (etiqueta interna).

El procesamiento de los paquetes asociados al servicio S3 en el nodo *Poisson* está dado por el siguiente flujo:

```
    cookie=0.0, duration=1090.776s, table=0, n_packets=215,
n_bytes=24092, priority=5, mpls,in_port=4,mpls_label=32,mpls_bos=1
actions=pop_mpls:0x0800,set_field:00:00:00:aa:00:00->eth_dst,output:1
```

Fig. F.6 Flujo 2

Este flujo toma todo paquete recibido por el puerto OpenFlow número 4 (interfaz nf2), retira el cabezal MPLS, modifica el valor del campo dirección MAC destino de la trama ethernet y finalmente lo reenvía por el puerto número 1 (interfaz eth1).

De esta forma todos los paquetes asociados al servicio S3 son transportados desde el nodo de ingreso *Galois* al nodo de egreso *Poisson* mediante un solo enlace, utilizando un solo nivel de etiquetas MPLS. Analizando el camino que deben seguir los paquetes al atravesar la red laboratorio en el

sentido inverso; es decir, desde la subred B hacia la subred A (servicio S4), el camino teórico es el siguiente:

$$< (Poisson, nf_1), (Oz, nf_0) >$$

Analizando primero la tabla de flujos del nodo *Poisson*, el primer salto del camino es implementado por los siguientes flujos:

```
cookie=0.0, duration=1090.775s, table=0, n_packets=299,
n_bytes=189800, priority=5, ip,in_port=1,
nw_src=20.20.20.64/26,nw_dst=20.20.20.128/26
actions=dec_ttl,push_mpls:0x8847,set_field:33->mpls_label,set_mpls_ttl (128),
goto_table:1

cookie=0.0, duration=1090.775s, table=0, n_packets=299,
n_bytes=189800, priority=5, mpls,in_port=1,mpls_label=33
actions=push_mpls:0x8847,set_fied:30->mpls_label,set_mpls_ttl(128),output:3
```

Notar como primera diferencia en comparación al camino anterior que se tienen dos flujos: el primero coloca la etiqueta asociada al servicio (etiqueta interna) y el segundo coloca la etiqueta de reenvío (etiqueta externa). El camino anterior carece de etiqueta externa puesto que al ser un camino de un solo salto, el primer nodo coincide con el penúltimo y al implementar PHP no es necesario colocar etiqueta externa.

Tras colocar el par de etiquetas MPLS en el paquete el mismo es reenviado a través del puerto OpenFlow número 3 (interfaz nf1). Luego, el siguiente salto en el camino es implementado por el siguiente flujo en *Oz*:

```
cookie=0.0, duration=1063.405s, table=0, n_packets=186,
n_bytes=20206, priority=5,mpls,in_port=3,mpls_label=30,mpls_bos=0
actions=pop_mpls:0x8847,output:2
```

Como Oz representa el penúltimo salto en el LSP las acciones asociadas son retirar la etiqueta externa del paquete y reenviarlo por el puerto número 4 (interfaz nf2) al último nodo en el camino. Tras esto, el tramo final del camino es implementado en el nodo *Galois* mediante el siguiente flujo:

```
cookie=0.0, duration=1022.644s, table=0, n_packets=186,
n_bytes=19462, priority=5,mpls,in_port=2,mpls_label=33,mpls_bos=1
actions=pop_mpls:0x0800,set_field:00:00:00:aa:00:03->eth_dst,output:1
```

Análogamente el lector puede completar el análisis de la correspondencia entre los caminos teóricos y calculados por el algoritmo de ruteo.

F.2 Clasificación de tráfico en VPN de capa 3 escenario 1

En la figura F.7 se muestra el procesamiento de los paquetes asociados al servicio S3 en cada uno de los nodos que componen el LSP, cuando se genera tráfico desde un host en la sub red A con destino a otro host en la subred B. Como puede observarse en el primer cuarto de la imagen (cuarto superior izquierdo), el cual se corresponde con una captura hecha con el comando **tcpdump** en la interfaz **eth1** en *Galois*, se reciben paquetes ICMP request con origen 20.20.20.130 y destino 20.20.20.66. También puede observarse en esta imagen paquetes ICMP reply a los paquetes request enviados. Luego, como se muestra en el segundo cuarto de la imagen (cuarto superior derecho), el cual se corresponde con una captura realizada en la interfaz nf2 de dicho nodo, a cada paquete ICMP request recibido por la interfaz eth1 se le coloca un cabezal MPLS con la etiqueta 31 y se reenvía por la interfaz en cuestión. Finalmente al arribar al nodo *Poisson* por la interfaz **nf2** (cuarto inferior izquierdo de la imagen), el cabezal MPLS es retirado de cada paquete para luego ser reenviado por la interfaz **eth1** hacia la subred B (cuarto inferior derecho de la imagen).

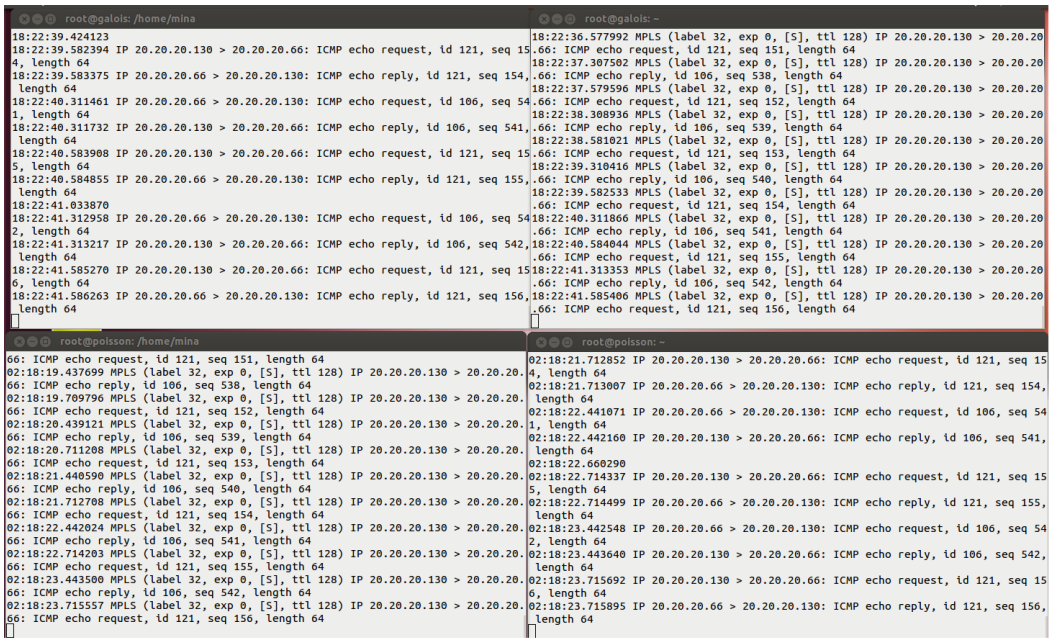

Fig. F.7 Capturas de tráfico con tcpdump - servicio S3

En la figura F.8 se muestra una captura de pantalla del comando **Ping** utilizado para generar tráfico ICMP desde el host 20.20.20.130 en la subred A, al host 20.20.20.66 en la subred B.

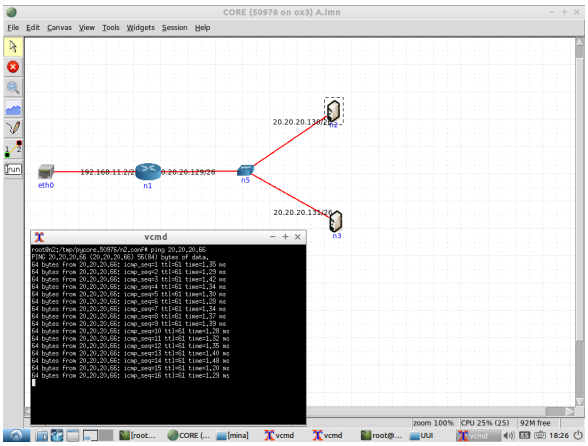

Fig. F.8 Comando Ping H1 Subread A - H3 Subred B

Análogamente se procesan los paquetes asociados a los restantes servicios en el sistema. Para complementar el ejemplo anterior, en la figura F.9 se muestra el procesamiento de los paquetes asociados al servicio S4, utilizando el comando ping para generar tráfico desde el host 20.20.20.66 en la subred B, al host 20.20.20.130 en la subred A. En la figura F.9 se muestran las capturas de tráfico utilizando el comando tcpdump para las interfaces eth1 (cuarto superior izquierdo de la imagen) y nf1 (cuarto superior derecho de la imagen) en el nodo Poisson, la interfaz nf0 en el nodo Oz (cuarto inferior izquierdo) y la interfaz eth1 en el nodo Galois (cuarto inferior derecho).

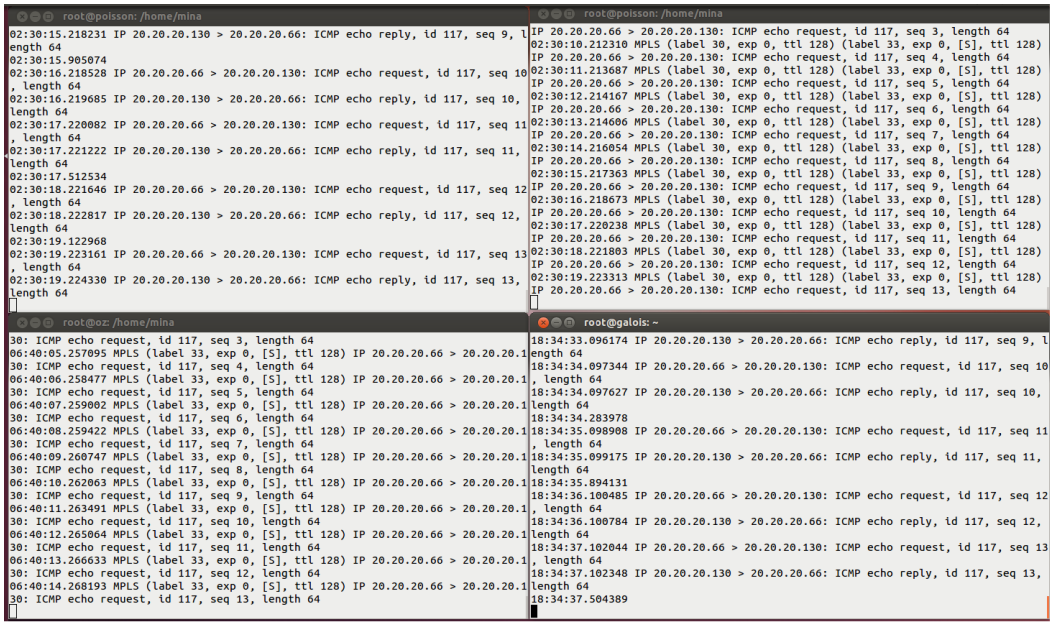

Fig. F.9 Capturas de tráfico con tcpdump - servicio S4

F.3 Actualización de Rutas en VPN de capa 3 escenario 1

Para comprobar que el algoritmo de ruteo recaclula correctamente las rutas, se inspeccionan nuevamente las tablas de flujos de cada nodo en la red laboratorio y se comparan los caminos calculados con los caminos teóricos. En las figuras F.10-F.13 se muestran las tablas de flujos de cada uno de los nodos tras la actualización topológica.

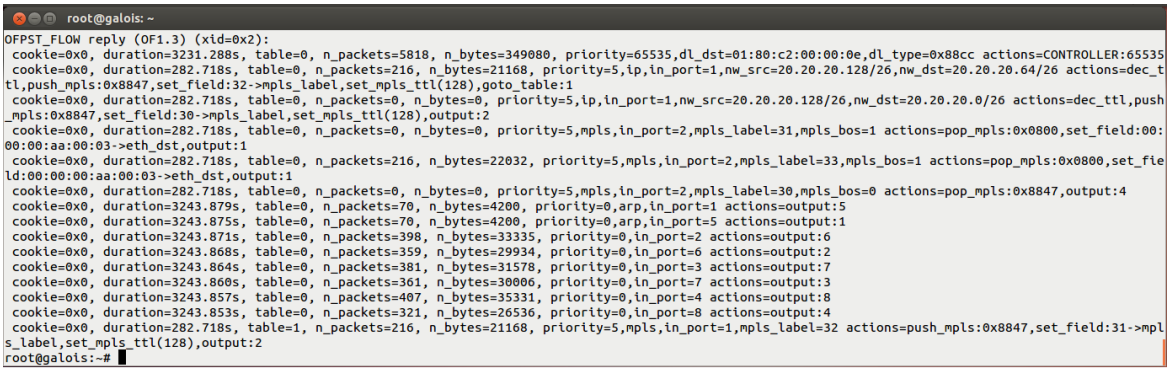

Fig. F.10 Tabla de flujos ovs - Galois

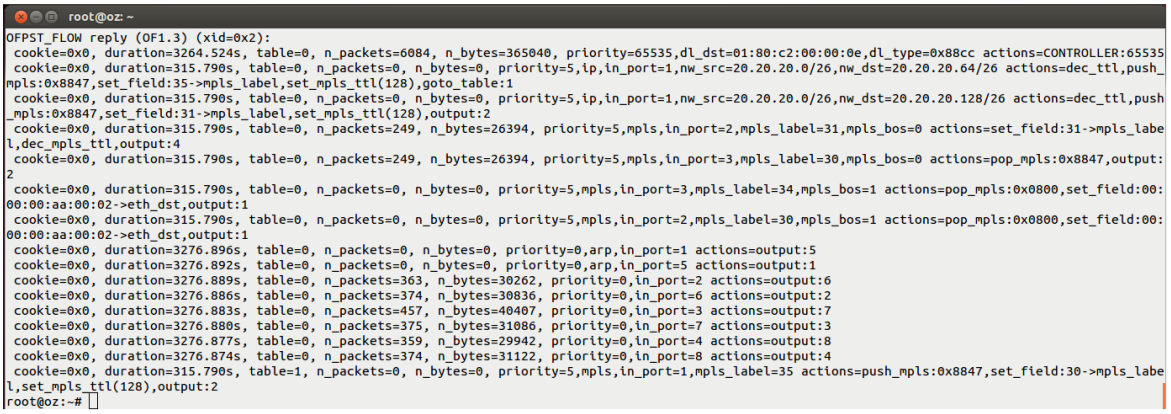

Fig. F.11 Tabla de flujos ovs - Oz

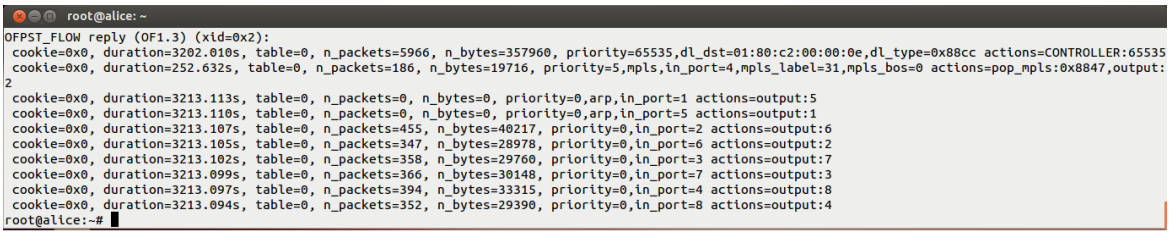

Fig. F.12 Tabla de flujos ovs - Alice

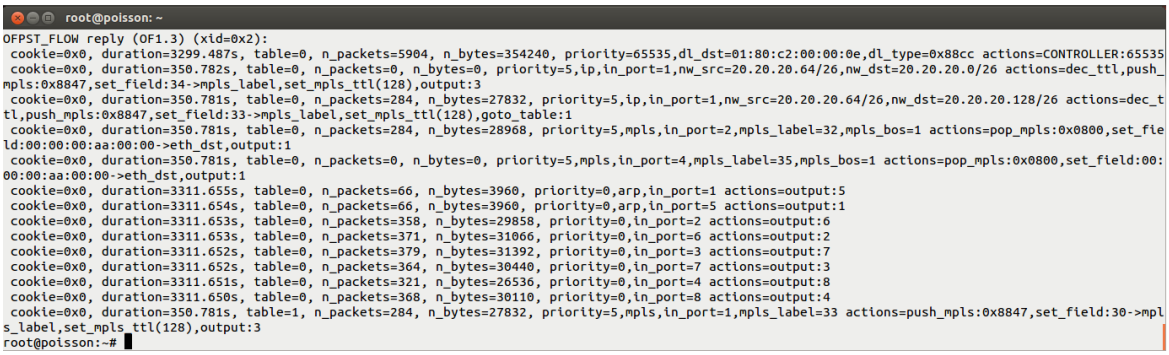

Fig. F.13 Tabla de flujos ovs - Poisson

Tomando como ejemplo la actualización del LSP asociado al servicio S3, mientras que en la topología original el camino asociado es $< (Galois, nf2) >$, tras la actualización el camino correcto puede ser o bien $< (Galois, nf0), (Oz, nf1) >$ o bien
$< (Galois, nf0), (Oz, nf2), (Alice, nf0) >$. Al cambiar el camino, los flujos asociados en cada nodo también cambian. En particular como el camino nuevo no comparte ningún salto con el camino original, los flujos asociados al camino viejo deben ser eliminados de cada nodo.

Recordando la tabla de flujos original del nodo *Galois* (ver imagen F.1), el flujo F.5 es utilizado para procesar y reenviar paquetes al nodo Poisson. En la tabla de flujos actualizada, este flujo es remplazado por el siguiente flujo (ver imagen F.10):

```
cookie=0.0, duration=282.718s, table=0, n_packets=216, n_bytes=21168,
priority=5,ip,in_port=1, nw_src=20.20.20.128/26,nw_dst=20.20.20.64/26
actions=dec_ttl,push_mpls:0x8847,set_field:32->mpls_label,set_mpls_ttl(128),
goto_table:1
```

```
cookie=0.0, duration=282.718s, table=0, n_packets=216, n_bytes=21168,
priority=5,mpls,in_port=1,mpls_label=32
actions=push_mpls:0x8847,set_fied:31->mpls_label,set_mpls_ttl(128),output:2
```

Mediante este par de flujos, se les colocan dos cabezales MPLS a los paquetes asociados al servicio, para luego ser reenviados por el puerto número 2 (interfaz nf0) al nodo *Oz*.

Por otra parte en la tabla de flujos del nodo *Oz*, se incorpora el siguiente flujo:

```
cookie=0.0, duration=315.790s, table=0, n_packets=249, n_bytes=26394,
priority=5,mpls,in_port=2,mpls_label=31,mpls_bos=0
actions=set_field:31->mpls_label,dec_mpls_ttl,output:4
```

El mismo implementa el cambio de etiqueta MPL en el paquete y su posterior reenvío por el puerto número 4 (interfaz nf2). Luego en la tabla de flujos del nodo *Alice* se incorpora el siguiente flujo:

```
cookie=0.0, duration=252.632s, table=0, n_packets=186,
n_bytes=19716, priority=5,mpls,in_port=4,mpls_label=31,mpls_bos=0
actions=pop_mpls:0x8847,output:2
```

Este flujo implementa el pop de la etiqueta MPLS externa en el penúltimo nodo del LSP (penultimate-pop-hoping). Tras realizar esta acción reenvía el paquete por el puerto número 2 (interfaz nf0).

Finalmente cuando el paquete arriba al nodo *Poisson*, el procesamiento final del paquete, realizado con anterioridad por el flujo F.6 ahora se realiza mediante el siguiente flujo:

```
cookie=0.0, duration=350.781s, table=0, n_packets=284,
n_bytes=28968, priority=5,mpls,in_port=2,mpls_label=32,mpls_bos=1
actions=pop_mpls:0x0800,set_field:00:00:00:aa:00:00->eth_dst,output:1
```

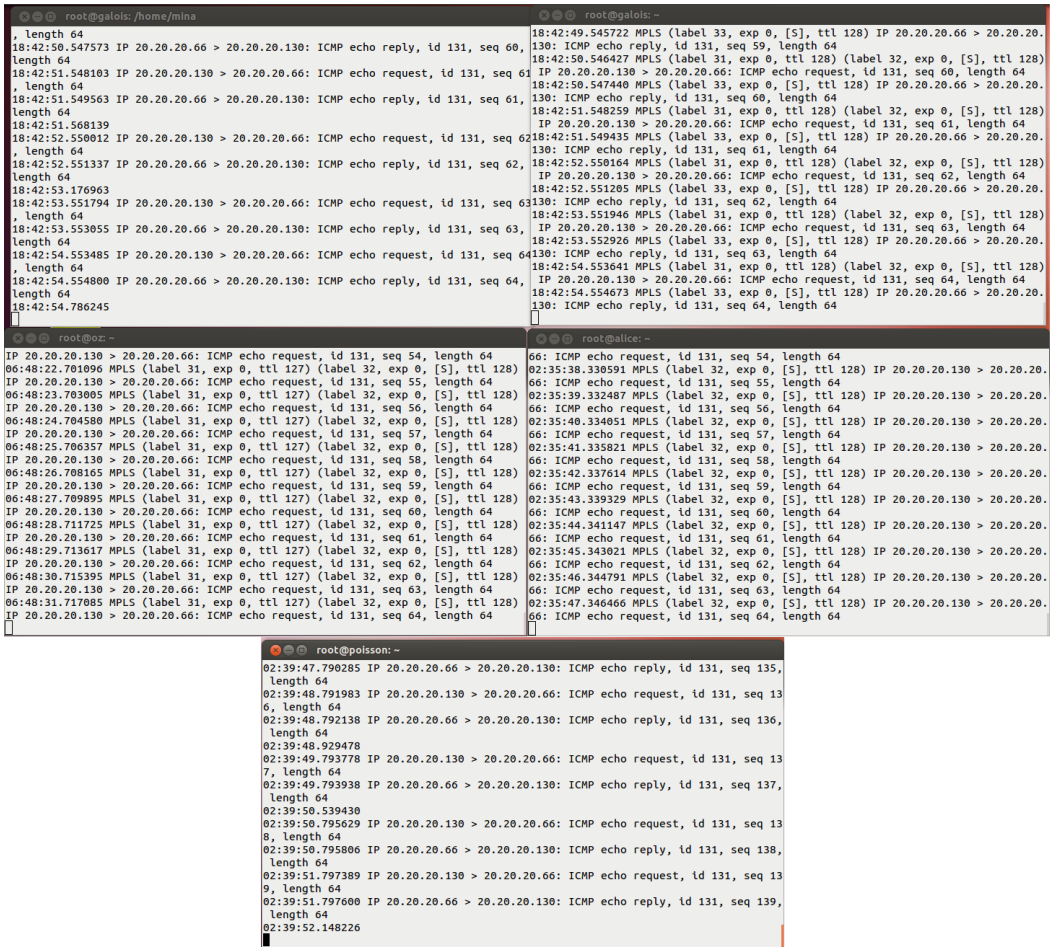

Fig. F.14 Capturas de tráfico con tcpdump - servicio S3 actualización topológica

En la figura F.14 se muestran capturas de tráfico en las interfaces por las que atraviesa el nuevo LSP asociado al servicio S3. Enumerando las figuras de izquierda a derecha y de arriba hacia abajo,

la primer imagen se corresponde con una captura sobre la interfaz eth1 del nodo Galois, la segunda con la interfaz nf0 del mismo nodo, la tercera con la interfaz nf2 del nodo Oz, la cuarta con la interfaz nf0 del nodo Alice y finalmente la quinta con la interfaz eth1 en el nodo Poisson.

Adicionalmente en la figura F.15 se muestra le ejeución del comando Ping en el host H1 de la subred A con destino al host H3 de la subred B durante la ejecución de la prueba. En la misma puede observarse cómo se envían y reciben paquetes antes del desperfecto en el enlace (paquetes ICMP con número de secuencia 1 al 5) y como luego se continuá enviando y recibiendo paquetes tras el desperfecto en el enlace (paquetes ICMP con número de secuencia 11 al 16). Por otro lado los paquetes perdidos (secuencia entre el 6 y el 10) se deben al tiempo transcurrido mientras la base de datos topológica (LSDB) se actualiza, la aplicación RAUFlow es notificada, ejecuta el algoritmo de ruteo para los servicios en el sistema y actualiza las tablas de flujo correspondientes.

Análogamente se puede estudiar la correspondencia entre el camino teórico y el camino calculado para el servicio S6.

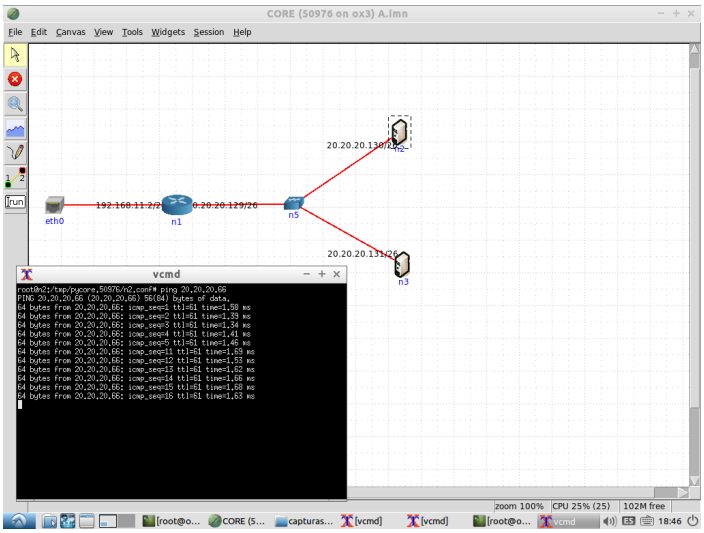

Fig. F.15 Comando Ping H1 Subread A - H3 Subred B

F.4 Numeraciones IP solapadas en VPN de capa 3 escenario 2

Para verificar que el tráfico asociado a un servicio es enrutado correctamente a la subred destino sin importar que la numeración IP esté solapada (aislamiento del espacio de direcciones), se prueba generar tráfico IP desde la subred A hacia la subred B al mismo tiempo que se genera tráfico desde la subred A' hacia la subred B'; el caso inverso (tráfico desde la subred B hacia la subred A) es análogo. Al mismo tiempo se observan las tablas de flujos Open vSwitch en cada uno de los nodos de la red para verificar los caminos calculados para cada servicio y se observa el procesamiento de cada paquete mediante capturas tcpdump en las interfaces asociadas a cada LSP.

En las figuras F.16-F.19 se muestran capturas de la tabla de flujos asociada a cada nodo en la red prototipo.

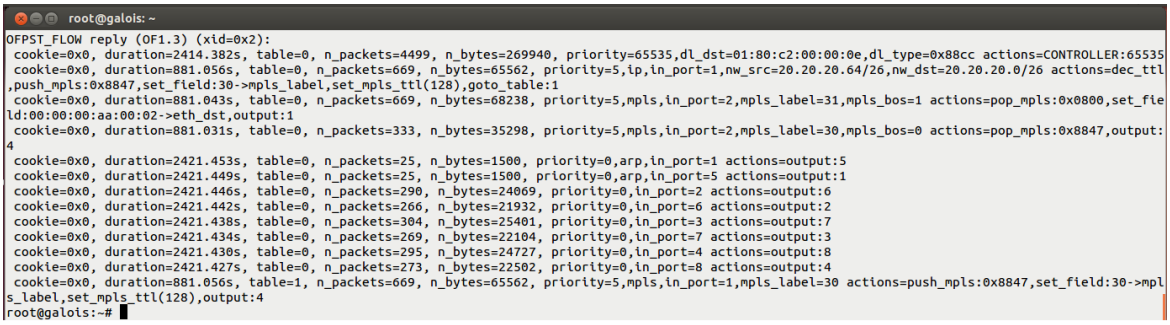

Fig. F.16 Tabla de flujos ovs - Galois

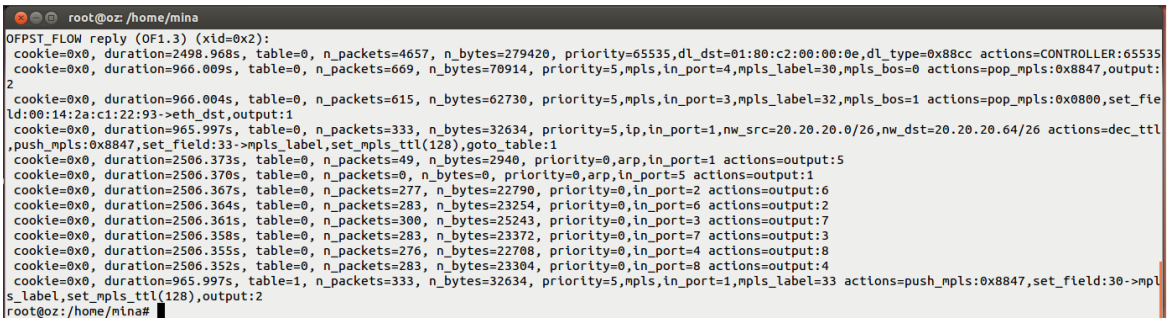

Fig. F.17 Tabla de flujos ovs - Oz

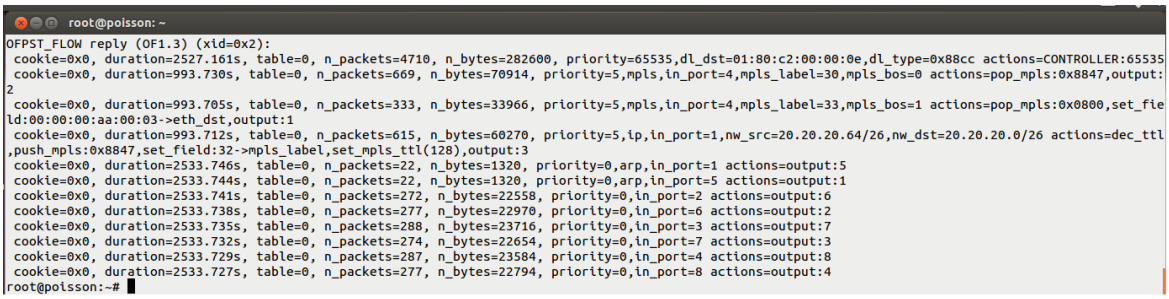

Fig. F.18 Tabla de flujos ovs - Poisson

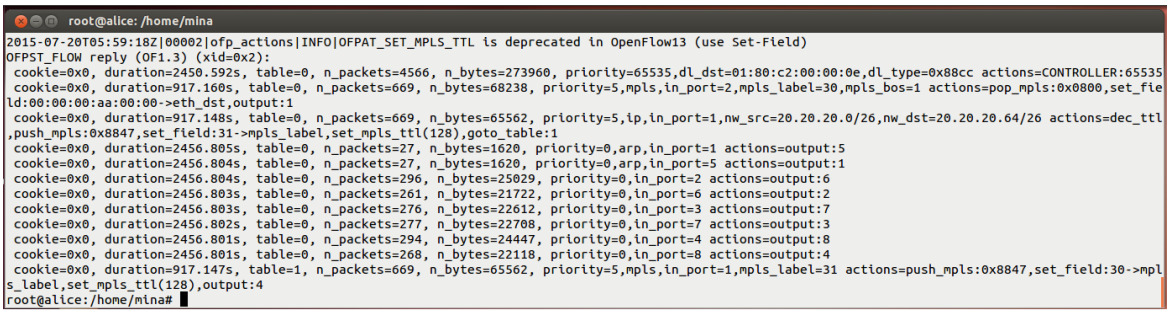

Fig. F.19 Tabla de flujos ovs - Alice

Se prueba generar paquetes ICMP mediante el comando ping simultáneamente entre los host H3 y H1 (subred A a subred B) y los hosts H3' y H1' (subred A' a subred B'). Para verificar que el tráfico es correctamente enrutado se realizan capturas tcpdump en las interfaces atravesadas por los LSPs de los servicios S1 y S3.

Por otro lado se prueba establecer conexiones SSH entre ambos pares de nodos, observando también el procesamiento del tráfico asociado a cada servicio. En la figura F.20 se muestra la conexión SSH establecida entre los hosts H3 (20.20.20.66) y H1 (20.20.20.2) en las subredes A y B respectivamente; nótese el hostanme y el usuario en la consol Linux.

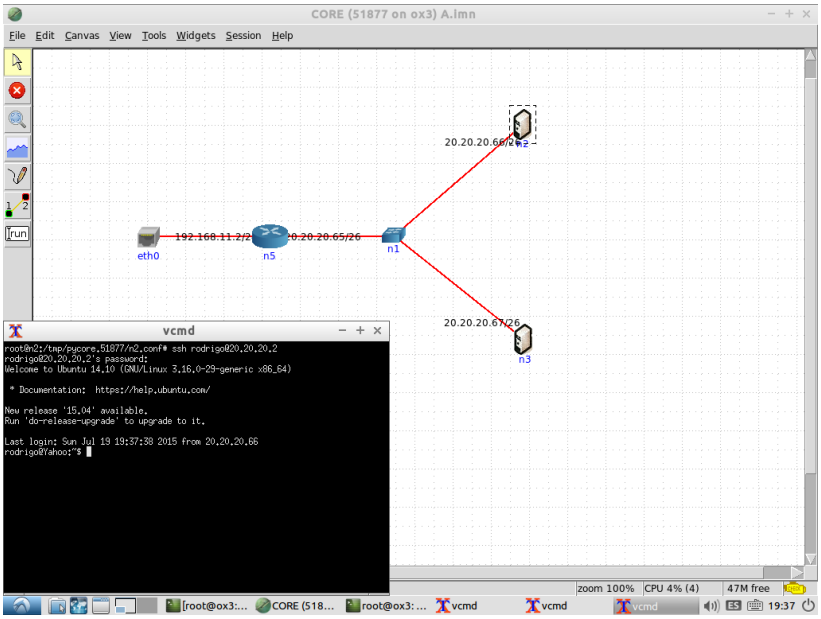

Fig. F.20 SSH desde Subred A hacia Subred B

Por otro lado en la figura F.21 se muestra la conexion SSH establecida entre los hosts H3 (20.20.20.66) y H1 (20.20.20.2) en las subredes A' y B' respectivamente. Nótese el hostname y el nombre de usuario de la consola Linux diferente al de la conexión SSH anterior.

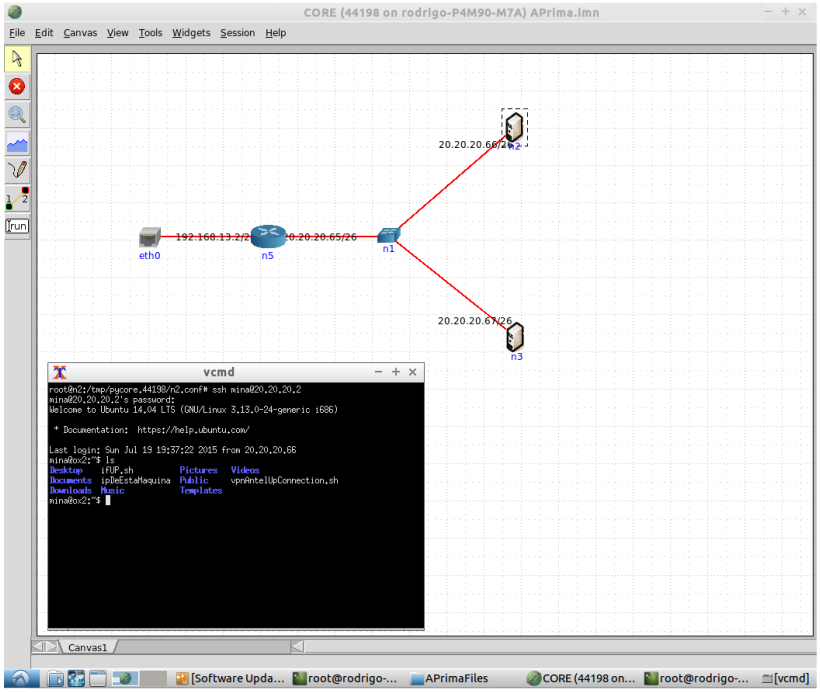

Fig. F.21 SSH desde Subred A' hacia Subred B'

F.5 Procesamiento del tráfico en VPN de Capa 2

De acuerdo al camino calculado para el tráfico asociado al servicio S1, los paquetes ingresan a la red del laboratorio a través el nodo Galois y dejan la misma mediante nodo Poisson. Capturando tráfico en las interfaces asociadas a esta camino, se observa el siguiente procesamiento (ver figura F.25):

En dicha figura se observan (de izquierda a derecha) capturas realizadas en las interfaces **eth1**, **nf2** de *Galois* y la interfaz **eth1** de *Poisson*. Por otro lado la primer fila de capturas muestra el procesamiento de paquetes IPv4, la segunda fila paquetes IPv6 y la tercera fila paquetes ARP.

Por simplicidad se omitieron capturas de las tablas de flujos Open vSwitch para los servicios creados dada la cantidad de entradas en la misma. Recordar que la implementación de RAUFlow crea una entrada en dicha tabla para cada ethertype soportado con el fin de rescribir dicho campo en el cabezal de un paquete en el nodo de egreso, puesto que este valor es sobrescrito en el primer salto de un LSP por el del protocolo MPLS cuando se coloca la primera etiqueta.

A continuación se muestran los paquetes recibidos en la subred B utilizando la herramienta Wireshark, en donde puede observarse que los paquetes entregados por la red del prototipo son exactamente iguales a los recibidos.

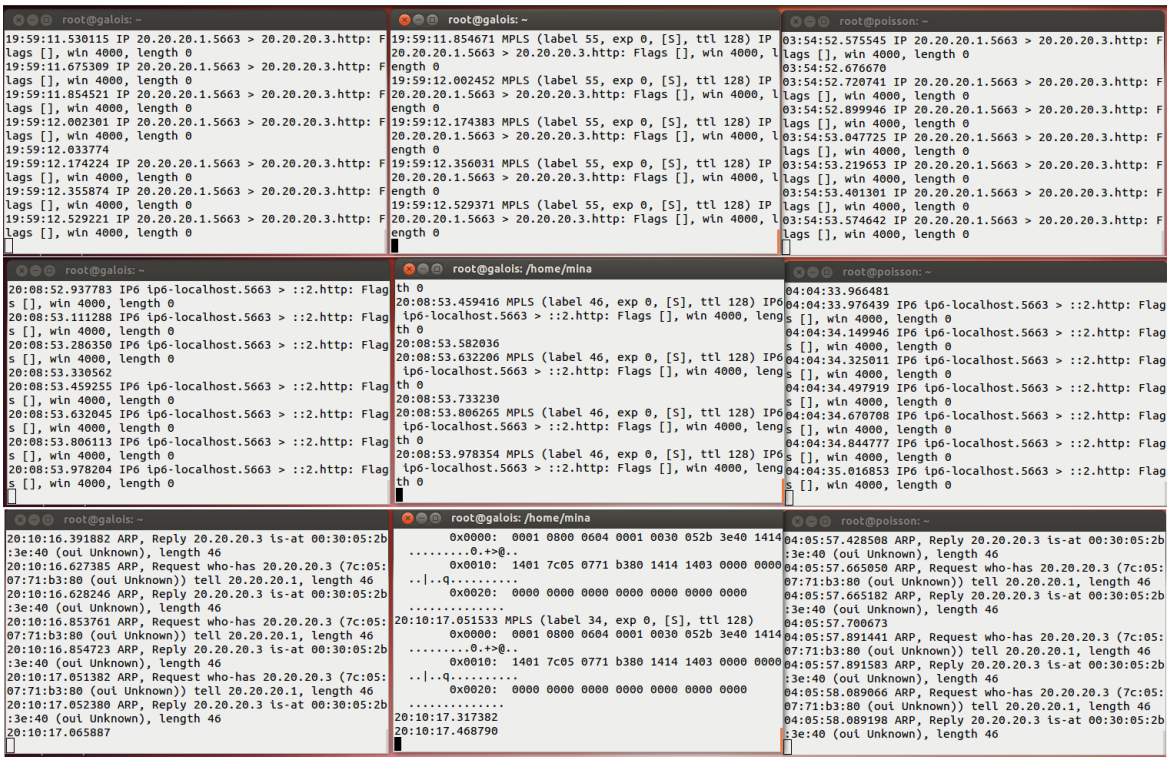

Fig. F.22 Caso de Uso 2 capturas

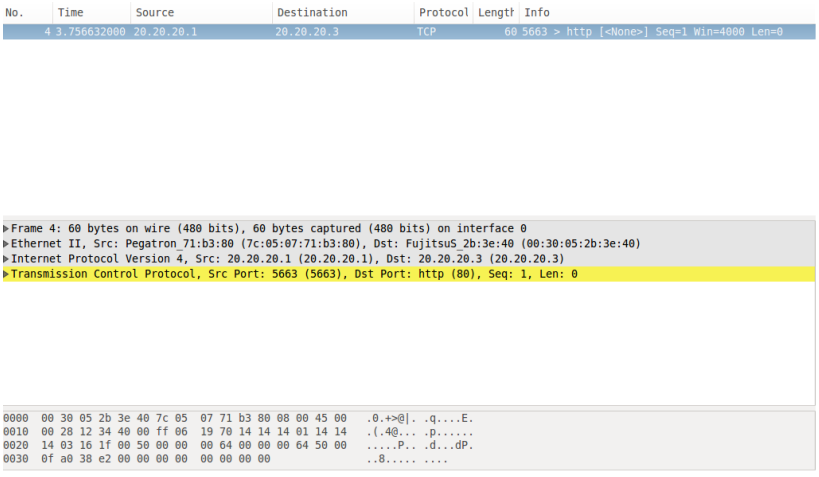

Fig. F.23 Paquetes IPv4 - Wireshark

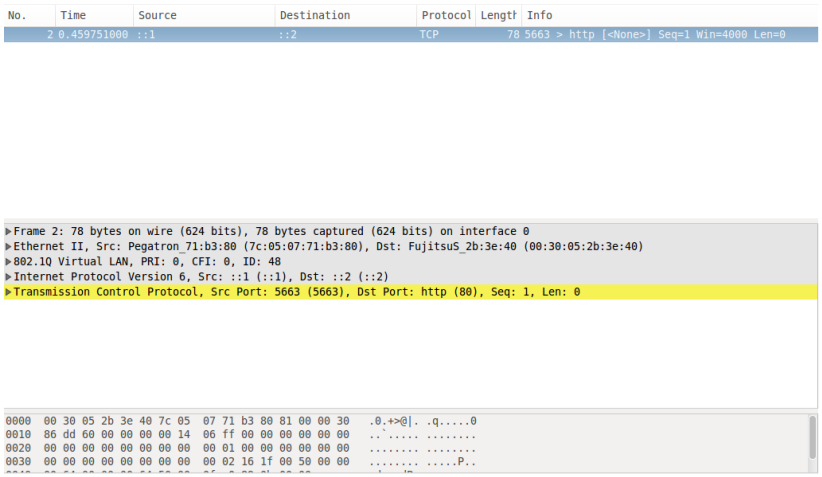

Fig. F.24 Paquetes IPv6 - Wireshark

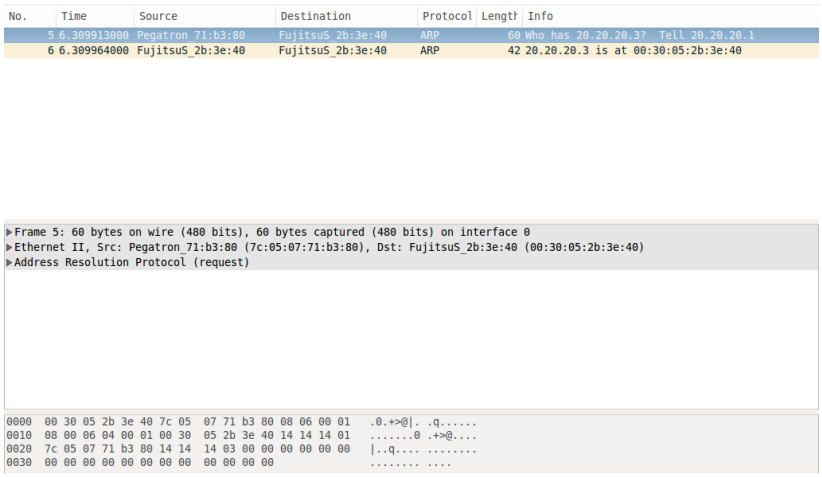

Fig. F.25 Paquetes ARP - Wireshark

Anexo I

Manual de construcción de RAU-Switch

I.1 Resumen

En este documento se ofrece una guía paso a paso para construir partiendo de una PC de escritorio, una tarjeta NetFPGA-10G y diferentes herramientas de software, un switch OpenFlow 1.3 IP/MPLS híbrido; lo que denominamos RAUSwitch.

En la misma no se asume ningún conocimiento previo acerca de la plataforma NetFPGA, Xilinx ISE ni sobre el protocolo OpenFlow. Cabe destacar además que existen tutoriales y guías para algunas etapas de la construcción del switch, pero se espera que este documento sea un resumen práctico de todos ellos, para agilizar el proceso de construcción para una persona que recién se encuentra familiarizándose con los conceptos aquí tratados.

En caso de querer profundizar en alguna de las etapas, se puede recurrir al material (en caso de existir) utilizado para la construcción de esta guía, siguiendo las referencias indicadas a lo largo de este documento.

I.2 Plataforma utilizada

A continuación se especifican las características del hardware utilizado en la construcción de RAU-Switch, así como la especificación de las herramientas de software con sus correspondientes versiones de producto. Se sugiere respetar los mismos en la medida que sea posible para evitar comportamientos inesperados o no contenidos en el alcance de esta guía.

[1]En el desarrollo del proyecto, para la construcción del laboratorio de pruebas se utiliza una tarjeta de red Ethernet adicional en el diseño RAU-Switch, para ser utilizada como interfaz para el ingreso de tráfico a la red prototipo. Dicho de otro modo esta interfaz se utiliza para conectarse a un Customer Edge router (CE) de una red privada.

Hardware
• Procesador: Intel Core i7 4770K 3.50GHz • Motherboard Asus ROG Maximus VI Formula • Fuente Thermaltake TR2 600W ATX 12V 2.3 • Memoria RAM Kingston 8GB DIMM DDR3 1333 MHz (x2) • Disco duro Seagate 1TB • NetFPGA-10G: 10-Gigabit SFP+ (x4), x8 gen1 PCIe, Xilinx's Virtex-5 TX240TFPGA • Transceiver Finisar FTLX8571D3BCL 850nm 13-50 Class 1 21CFR1040.10 LN#50 701 • Patchcord multimodo OM3 50/125 duplex lc lc 10 metros pvc 2mm il $\leq$ 0.25dB RL $\geq$ 28dB • TP-Link Gigabit PCI Express TG-3468 [1]

Table I.1 RAU-Switch, especificaciones de hardware

Software
• Sistema Operativo: Ubuntu 12.04 LTS release 12.04 precise 3.11.0-15 generic x86_64 • Open vSwitch v2.3.9 • Repositorio NetFPGA v5.0.5 • Xilinx ISE_DS_LIn 13.04_087xd.3.0 • SNMP 5.4.3 • Quagga 0.99.22

Table I.2 RAU-Switch, especificaciones de software

I.3 Instalación y configuración

En esta sección se explica el proceso de instalación y configuración del software necesario para la construcción de RAUSwitch. En cuanto al orden de precedencia esta guía está organizada de la siguiente forma:

1. Instalación del sistema operativo

2. Instalación de librerías y dependencias
3. Instalación de suite de desarrollo Xilinx ISE SDK
4. Configuración del entorno de desarrollo NetFPGA
5. Pruebas de aceptación del hardware NetFPGA
6. Programación del hardware
7. Instalación de Open vSwitch
8. Instalación de Quagga
9. Instalación de agente SNMP

I.3.1 Instalación de sistema operativo

Para la instalación del sistema operativo, utilizamos un dvd de instalación de Ubuntu 12.04 LTS descargado desde la página oficial. Para descargar una imagen con el instalador de este sistema se puede recurrir a:

http://releases.ubuntu.com/12.04/

En caso de no estar familiarizado con el proceso de instalación de un sistema operativo recomendamos el siguiente tutorial:

http://www.ubuntu.com/download/desktop/install-ubuntu-desktop

I.3.2 Instalación de librerías y dependencias

Las siguientes librerías son necesarias por las diferentes herramientas de software utilizadas en el switch: gitk, git-gui, libusb, build-essential, libc6-dev, fxload, autotools-dev, autoconf, uml-utilities, libtool. Para instalarlas abrir una consola y ejecutar los siguientes comandos:

```
1 # sudo apt-get update
2 # sudo apt-get install gitk git-gui libusb-dev build-essential
3 # sudo apt-get install libc6-dev-i386 fxload autotools-dev
4 # sudo apt-get install autoconf uml-utilities libtool
```

En caso de tener en algún momento errores con el comando gmake (por ejemplo “gmake comand not found”), crear un link simbólico entre *gmake* y *make*. Para ello ejecutar en una consola:

```
1 # sudo ln -s /usr/bin/make /usr/bin/gmake
```

I.3.3 Instalación de suite de desarrollo Xilinx ISE SDK

La instalación de la suite Xilinx ISE SDK la realizaremos desde el código fuente por lo que primero que nada debemos descargar dichos archivos desde la página oficial de Xilinx. En este trabajo se descargó el siguiente archivo:

Xilinx_ISE_DS_Lin_13.4_087xd.3.0.tar

Luego debemos ir al directorio en donde se descargó el archivo y ejecutamos los siguientes comandos:

```
# tar -xvf Xilinx_ISE_DS_Lin_13.4_087xd.3.0.tar
# sudo chmod +x xsetup
# sudo ./xsetup
```

De esta forma se ejecuta el asistente de instalación de la suite de Xillinx. Dejar todas las opciones que marca por defecto con la excepción de la instalación de los drivers para el cable de programación. En este caso destildar el checkbox correspondiente para evitar que la instalación finalice con errores.

Terminada la instalación de la suite de Xilinx debemos instalar los drivers para el cable de programación JTag. Trabajaremos con un driver proporcionado por terceras partes disponible en un repositorio github.

Por defecto la instalación de Xilinx se realizó en el directorio /opt/Xilinx, en caso contrario trabajar en lo que prosigue en el directorio correspondiente.

Abrimos una consola y ejecutamos lo siguiente:[2]

```
# cd /opt/Xilinx/
# git clone git://git.zerfleddert.de/usb-driver
# cd usb-driver/
# make
# chmod +x setup_pcusb
# ./setup_pcusb /opt/Xilinx/13.4/ISE_DS/ISE
```

Finalmente reiniciamos la PC.

Configuración de variables de entorno

Para utilizar la suite de Xilinx es necesario agregar ciertas variables de entorno al sistema. Para ello una alternativa es editar el archivo **.bashrc** en la consola del usuario root, agregando las siguientes líneas:

```
# sudo su
# nano /root/.bashrc

PATH=\$PATH:/opt/Xilinx/13.4/ISE_DS/EDK/bin/lin64
:/opt/Xilinx/13.4/ISE_DS/common/lib/lin64
:/opt/Xilinx/13.4/ISE_DS/ISE/bin/lin64/
```

[2]Para la arquitectura de la máquina se debe compilar la versión de 64 bits del driver; lo cual se hace por defecto. En caso de trabajar con una arquitectura de 32bits se puede ejecutar el make para 32bits.

```
:/opt/Xilinx/13.4/ISE_DS/common/bin/lin64
:/opt/Xilinx/13.4/ISE_DS/EDK/gnu/microblaze/lin64/bin

XILINX=/opt/Xilinx/13.4/ISE_DS/ISE/
XILINX_EDK=/opt/Xilinx/13.4/ISE_DS/EDK
XILINXD_LICENSE_FILE=/opt/Xilinx
LD_LIBRARY_PATH=/opt/Xilinx/13.4/ISE_DS/ISE/lib/lin64/
:/opt/Xilinx/13.4/ISE_DS/EDK/lib/lin64

export PATH
export XILINX
export XILINX_EDK
export XILINXD_LICENSE_FILE
export LD_LIBRARY_PATH
```

De esta forma queda instalada la suite de Xilinx y los drivers del cable JTag para la programación de las tarjetas.

Ejecución de herramientas de la suite de Xilinx

Para la ejecución de cualquier herramienta de la suite Xilinx es necesario ejecutar un script de configuración, presente en el directorio de instalación de la herramienta. Para ello abrir una consola y ejecutar lo siguiente:

```
# /opt/Xilinx/13.4/ISE_DS/settings64.sh
```

En caso de que el archivo **setting64.sh** no cuente con permisos de ejecución asignárselos.

Luego se puede ejecutar cualquier herramienta de la suite Xilinx, como por ejemplo la herramienta Impact para programar normalmente las tarjetas de la siguiente forma:

```
# impact
```

Instalación de licencias de usuario

Para trabajar con las herramientas de Xilinx es necesario contar con licencias correspondientes para cada una de ellas y en particular según el tipo de hardware con el que se trabaje dependerá el tipo de licencia necesaria. En nuestro caso se necesita una licencia completa (full license) y vale la pena destacar que con una licencia de prueba no se cuentan con las licencias apropiadas para compilar los proyectos de NetFPGA con los que se trabajará. Para obtener una licencia full se debe ir al sitio de Xilinx de gestión de licencias, registrarse y comprar una licencia full.

Asumiendo que ya se tiene descargada una licencia full, para cargar la misma abrir una consola y ejecutar lo siguiente:

```
# /opt/Xilinx/13.4/ISE_DS/common/bin/lin64/xlcm
```

Una vez que se ejecuta el comando anterior, se abrirá el programa Xilinx Lincense Configuration Manager que es el asistente de licencias de la suite de Xilinx. Una vez abierto, ir a la pestaña de Manejo de Licencias (Manage Xilinx Licenses) e importar los archivos de licencias *.lic* obtenidos

(botón copy license). Una vez importada la licencia se podrán ver abajo las diferentes características que proveen las licencias cargadas. En caso de que no se actualice la pestañea apretar el botón **Refresh**.

I.3.4 Configuración del entorno de desarrollo NetFPGA

A continuación se explica como acceder al código fuente de NetFPGA y como empezar a programar el hardware con el mismo comenzando por obtener el código fuente con las implementaciones de proyectos y herramientas. Para ello es necesario crearse un usuario github y luego registrarse como desarrollador NetFPGA en el sitio oficial; este último proceso puede llevar algunas horas dado que el proceso de aceptación se realiza manualmente. Para registrarse como desarrollador NetFPGA completar el siguiente formulario:

http://netfpga.org/2014/#/10G_going_beta/

Una vez aceptada la solicitud de unirse al equipo de NetFPGA, se tendrá acceso a los repositorios de código fuente en github. Descargamos la versión base de código fuente en el sitio de NetFPGA en github. Para descargar el código fuente basta con acceder al siguiente link estando logueado en Github:

https://github.com/NetFPGA/NetFPGA-10G-live/tags

En particular trabajaremos con la release 5.0.5 la cual se encuentra accesible mediante el siguiente enlace:

https://github.com/NetFPGA/NetFPGA-10G-live/releases/tag/release_5.0.5

Una vez descargado el código fuente, descomprimimos el archivo .zip en un directorio a elección (en nuestro caso utilizaremos el directorio /home/mina). Luego abrimos una consola en el directorio y ejecutamos lo siguiente:

```
1 # cd /home/mina/NetFPGA-10G-live-release_5.0.5/
2 # make cores
```

Este proceso puede llevar de 5 a 10 minutos dependiendo de las prestaciones del hardware. Una vez finalizado el proceso anterior estamos en condiciones de programar las tarjetas NetFPGA con el proyecto que creamos conveniente.

I.3.5 Pruebas de aceptación del hardware NetFPGA

Antes de empezar a trabajar con el hardware NetFPGA y en especial a programarlo con cualquier proyecto, es sumamente importante ejecutar un conjunto de pruebas desarrolladas específicamente para la tarjeta NetFPGA-10G, con el objetivo de chequear el correcto funcionamiento de sus componentes. En particular existen dos pruebas:

- RLDRAM Test[36]
- Production Test[35]

Estos tests son proyectos de referencia; es decir fueron desarrollados por NetFPGA y la universidad de Stanford. Con los mismos se programan las tarjetas de igual forma que se programaría cualquier otro proyecto y luego se ejecutan una serie de scripts que realizan pruebas de verificación del hardware. En pocas palabras, además de verificar el correcto funcionamiento de las tarjetas, al ejecutar los test se tiene una primera experiencia programando el hardware.

Es importante tener en cuenta que las tarjetas NetFPGA pueden funcionar tanto conectadas a una PC (escenario elegido) denominado server mode como conectadas solamente a una fuente de poder denominado standalone mode.

En particular, ambos proyectos de pruebas pueden programarse para ejecutarse en ambas modalidades de funcionamiento. En esta guía se explica como programar y ejecutar dichos proyectos bajo el modo de funcionamiento servidor. En caso de requerirse de una ejecución de pruebas en modo standalone puede encontrarse en[38][28] una guía detallada.

Programación y ejecución del Production Test

A continuación se detalla el procedimiento para programar el hardware con el test de producción y luego ejecutar el conjunto de pruebas asociadas. Esta guía esta basada a su vez en una guía mas completa accesible en[38]

EL primer paso para programar el test de producción es configurar la tarjeta NetFPGA-10G correctamente, así como instalarla en una PC. Es sumamente IMPORTANTE seguir cuidadosamente cada una de las indicaciones que aquí se mencionan para lograr los resultados correctos y no dañar el hardware en el proceso de manipulación.

La tarjeta necesita un suministro de corriente mediante el conector ATX para poder funcionar. A continuación se describe como instalar el hardware:

- Asegurarse que el jumper J16 no esta activado. Este Jumper se encuentra en la esquina superior izquierda de la tarjeta, en la cercanía del conector RS232 DB-9 (figura I.1).

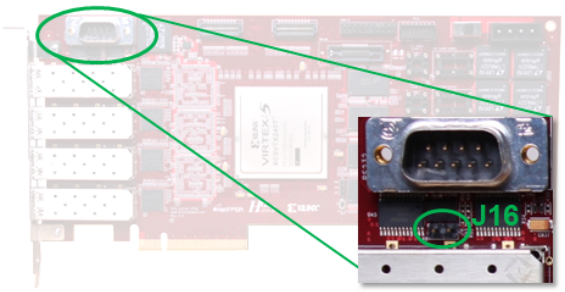

Fig. I.1 Jumper J16, Imagen extraída de [38]

- Setear el switch SW9 en la posición PCIe. Asegurarse bien de que el switch está completamente puesto en dicha posición (figura I.2).

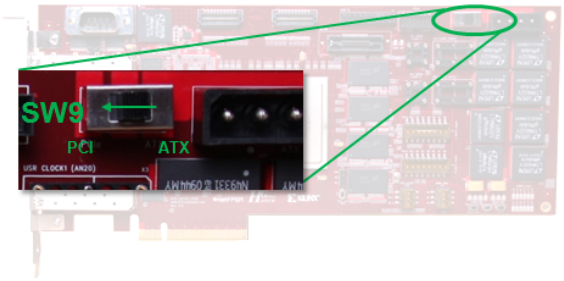

Fig. I.2 SW9 en posicion PCIe, Imagen extraída de [38]

- Asegurarse que los switches DIP SW1, SW2, SW6 y SW10 están correctamente ubicados acorde al siguiente esquema (figura I.3):

SW1: SEL0, SEL1, M2, M1, M0, N2, N1, N0 son seteados en off, off, off, on, on, on, off, off
SW6: SEL0, SEL1, M2, M1, M0, N2, N1, N0 son seteados en off, off, off, on, off on, off, on
SW3: M0, M1, M2 son seteados en on, on, on
SW10: SEL0, SEL1, SEL2 son seteados en on, off, off

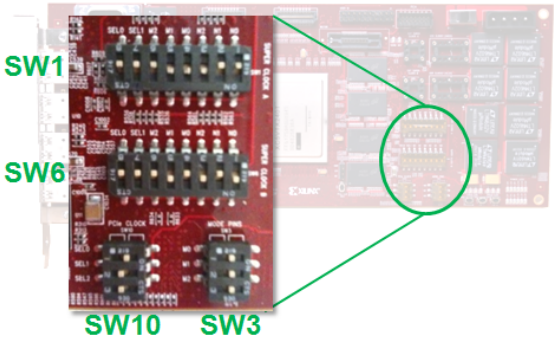

Fig. I.3 Posiciones DIP SW1 SW2 SW6 SW10, Imagen extraída de [38]

- Con la PC apagada y desconectada de la red eléctrica instalar la tarjeta NetFPGA-10G en un slot PCIe disponible.
- Conectar alguno de los conectores ATX disponibles de la PC al conector ATX de la tarjeta NetFPGA-10G (figura I.4).

Fig. I.4 Conector ATX, , Imagen extraída de [38]

- Conectar el cable Samtec Twinax a la interfaz de expansión de la tarjeta formando un “loop” (figura I.5)

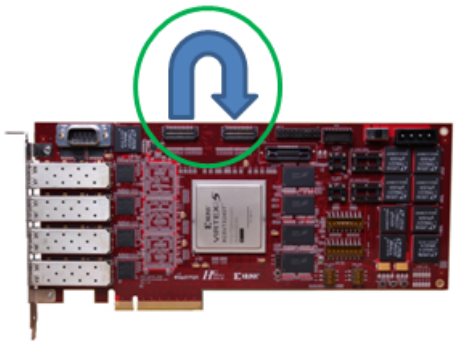

Fig. I.5 Cable samtec twinax, , Imagen extraída de [38]

- Conectar un par de cables 10GE SFP+ a las interfaces de la tarjeta. Cada cable conecta dos puertos adyacentes formando dos “bucles” (ver figura I.6):

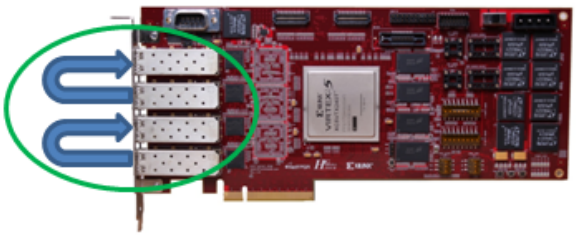

Fig. I.6 Cables samtec twinax conectados

- Conectar un cable serial RS232 al puerto serial RS232 en la tarjeta. Conectar el otro extremo en el puerto serial COM de la PC o a un adaptador USB por ejemplo (ver figura I.7).

Fig. I.7 Cable serial puerto COM, Imagen extraída de [38]

- Conectar el cable de programación Xilinx JTAG al conector JTAG en la tarjeta. Luego conectar el otro extremo a la PC a través de un puerto USB por ejemplo (figura I.8).

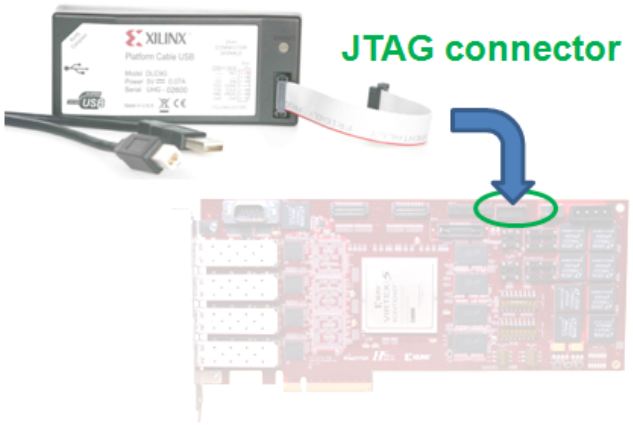

Fig. I.8 Cable JTAG, Imagen extraída de [38]

Al finalizar, la tarjeta NetFPGA debería estar instalada como se muestra en la siguientes figuras I.9 I.10:

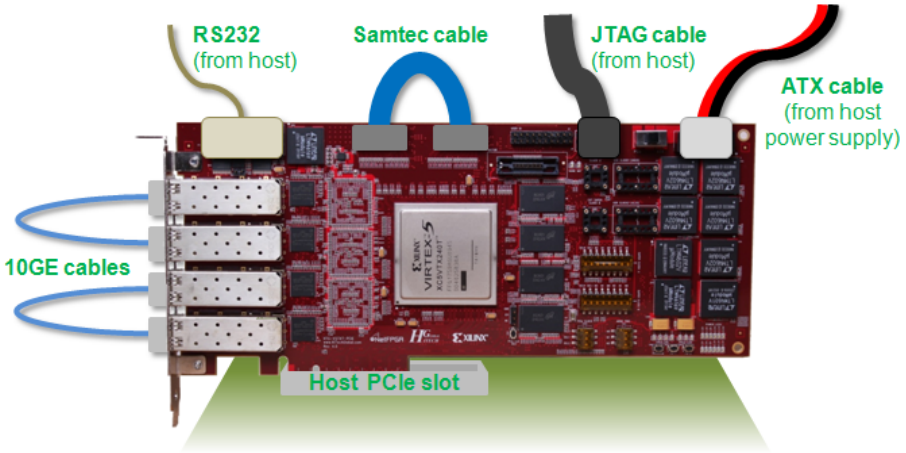

Fig. I.9 Tarjeta NetFPGA esquema de instalación, Imagen extraída de [38]

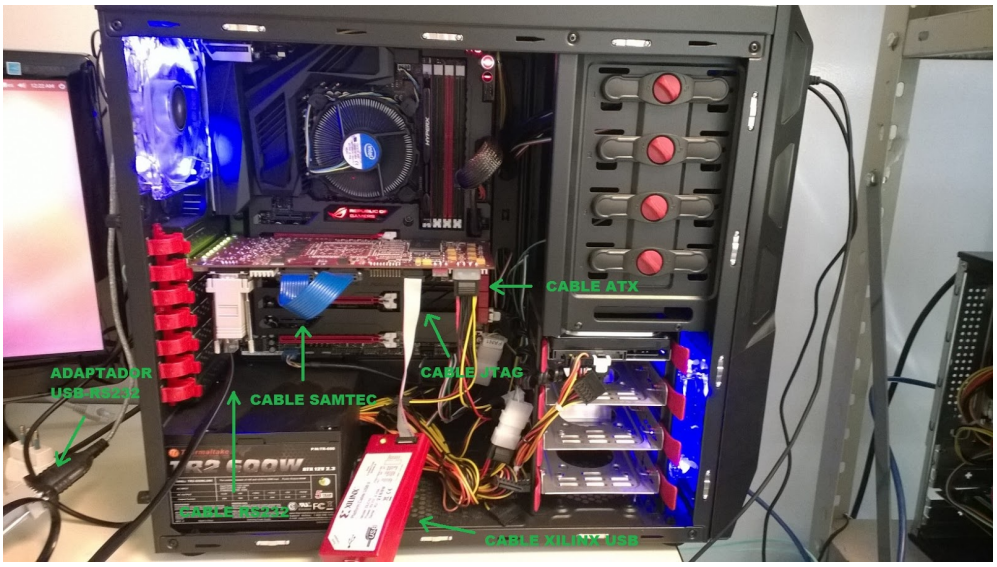

Fig. I.10 Tarjeta NetFPGA instalada en una PC

Para ejecutar el Test de Producción seguir los siguientes pasos:

- Encender el PC y como usuario root abrir el archivo /boot/grub/grub.cfg con un editor de texto. Realizar las siguientes modificaciones:
  - Localizar la entrada particular para la distribución del kernel que se tenga
  - Encontrar la línea que comienza con Kernel
  - Concatenar **memmap=256M\$0x5f700000** al final de la línea (ver figura I.11).
    El parámetro memmap=256M\$0x5f700000 es pasado al kernel al momento de iniciar el sistema. El mismo reserva una región de 256MB de memoria disponible comenzando en la dirección 0x5f700000. Este bloque es utilizado posteriormente en el Test PCIe DMA.
  - Guardar los cambios realizados al archivo

```
  GNU nano 2.2.6                                File: grub.cfg

    set linux_gfx_mode=keep
  fi
else
  set linux_gfx_mode=text
fi
export linux_gfx_mode
if [ "${linux_gfx_mode}" != "text" ]; then load_video; fi
menuentry 'Debian, with Linux 3.11.0-26-generic' --class debian --class gnu-linux --class gnu --class os {
        recordfail
        gfxmode $linux_gfx_mode
        insmod gzio
        insmod part_msdos
        insmod ext2
        set root='(hd0,msdos1)'
        search --no-floppy --fs-uuid --set=root 1b794e5c-015e-4158-aa4b-b71cea308f1b
        linux   /boot/vmlinuz-3.11.0-26-generic root=UUID=1b794e5c-015e-4158-aa4b-b71cea308f1b ro memmap=256M\$0x5f700000 quiet splash $vt_handoff
        initrd  /boot/initrd.img-3.11.0-26-generic
}
menuentry 'Debian, with Linux 3.11.0-26-generic (recovery mode)' --class debian --class gnu-linux --class gnu --class os {
        recordfail
        insmod gzio
        insmod part_msdos
        insmod ext2
        set root='(hd0,msdos1)'
        search --no-floppy --fs-uuid --set=root 1b794e5c-015e-4158-aa4b-b71cea308f1b
        echo    'Loading Linux 3.11.0-26-generic ...'
        linux   /boot/vmlinuz-3.11.0-26-generic root=UUID=1b794e5c-015e-4158-aa4b-b71cea308f1b ro recovery nomodeset memmap=256M\$0x5f700000
        echo    'Loading initial ramdisk ...'
        initrd  /boot/initrd.img-3.11.0-26-generic
```

Fig. I.11 Reserva de bloque de memoria

- Iniciar la aplicación IMPACT de Xilinx indicando la opción “Yes” cuando se pregunte si queremos que la herramienta cree y guarde automáticamente un proyecto por nosotros.
- Programar la CPLD con el diseño localizado en el directorio (ver figura I.12) /NF_ROOT/projects/production_test/bitfiles/cpld.jed

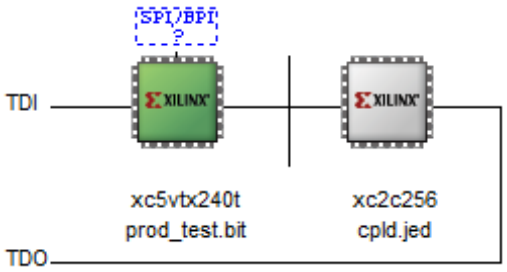

Fig. I.12 Programación Impact

- Programar la FPGA con el bitstream localizado en el directorio /NF_ROOT/projects/production_test/bitfiles/prod_test_no_rldram.bit

  IMPACT posiblemente nos pregunte si queremos asociar un dispositivo PROM, seleccionar que no. Programar la tarjeta puede llevar algún tiempo; de 10 a 20 minutos dependiendo del proyecto. Este proyecto en particular no debería tomar más de 10 minutos.

- Reiniciar la PC
- Verificar que el bloque de memoria DMA ha sido reservado correctamente a través de los siguientes pasos:
  - Ejecutar en una consola:

```
# cat /proc/cmdline
```

- La salida debería ser similar a la figura I.13

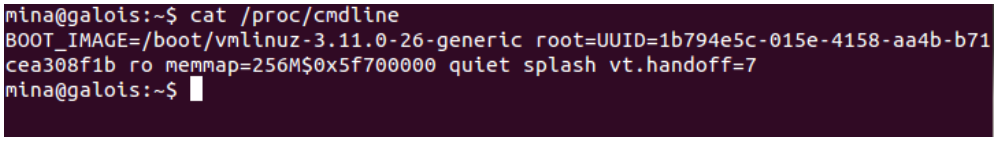

Fig. I.13 Reserva bloque de memoria

- Verificar que la tarjeta ha sido correctamente instalada siguiendo los siguientes pasos:
  - Desde una consola verificar el dispositivo NetFPGA-10G ejecutando:

```
#lspci | grep Xilinx
```

  - La salida debería ser similar a:

```
# 01:00.0 Ethernet controller: Xilinx Corporation Device 4244
```

- Desde la PC ejecutar el test de producción a través de los siguientes pasos:
  - Situarse en la carpeta del Test de Producción

```
# cd projects/production_test/sw/
```

  - Compilar el test

```
# make
```

  - Posicionarse en la carpeta de scripts

```
# cd scripts
```

  - Ejecutar el Test de Producción

```
# sudo ./production_test.py
```

    La salida debería ser como la que se muestra a continuación (ver figura I.14):

```
NetFPGA-10G Production Test
Version: 1.0
Time: 2010-10-27 02:13:18.852668

Running UART test...
UART test finished!
---------- UART Test Report ----------
UART Test Result: PASS

Running PCIe DMA test... (will take 10 seconds)...
PCIe DMA test finished!
---------- PCIe DMA Test Report ----------
DMA Test Result:              PASS
DMA Transfers:                69451 totaling 142.24 MB

Checking FPGA system status registers...
---------- FPGA System Report ----------
Address  Name                      Value
0x00000: HW Version                0x00000104    PASS: Detected Supported Board Revision
0x00004: CLOCK ok                  0x00000007    PASS
0x00008: QDRII ok                  0x00000015    PASS
0x0000c: PWR ok                    0x00000001    PASS
0x00010: CPLD/Flash ok             0x00000007    PASS
0x00018: 10G if 0 link             0x00000001    PASS
0x00024: 10G if 0 er count         0x00000000    PASS
0x00028: 10G if 1 link             0x00000001    PASS
0x00034: 10G if 1 er count         0x00000000    PASS
0x00038: 10G if 2 link             0x00000001    PASS
0x00044: 10G if 2 er count         0x00000000    PASS
0x00048: 10G if 3 link             0x00000001    PASS
0x00054: 10G if 3 er count         0x00000000    PASS
0x0005c: Samtec 0 link             0x00000001    PASS
0x00064: Samtec 1 link             0x00000001    PASS

Production test finished.
```

RS232/UART Interface test

PCIe Interface test

10GE Interface tests

Expansion Interface tests

Fig. I.14 Test de Producción salida esperada

Para ejecutar el RLDRAM Test recomendamos utilizar la guía [28].

I.3.6 Programación de la tarjeta

A continuación se describen los pasos necesarios para programar las tarjetas NetFPGA en particular con el proyecto ReferenceNIC. En lo que prosigue se utiliza NF_ROOT para referirse al directorio donde se encuentra descargado el código fuente de NetFPGA.

Configuración de la tarjeta

Antes de comenzar, es necesario realizar algunas modificaciones en el hardware NetFPGA para habilitar la programación PCIe del mismo, utilizada en la programación persistente. Para ello es necesario cambiar la posición del switch Disp SW3 desde la posición por defecto (M0:On, M1:On, M2:On) a la posición de programación PCIe (M0:Off, M1:On, M2:On). Esta configuración habilita tanto la programación de la tarjeta por el cable de programación JTAG como por la interfaz PCIe. La siguiente figura muestra la posición del Dip SW3 en la configuración PCIe (ver figura I.15).

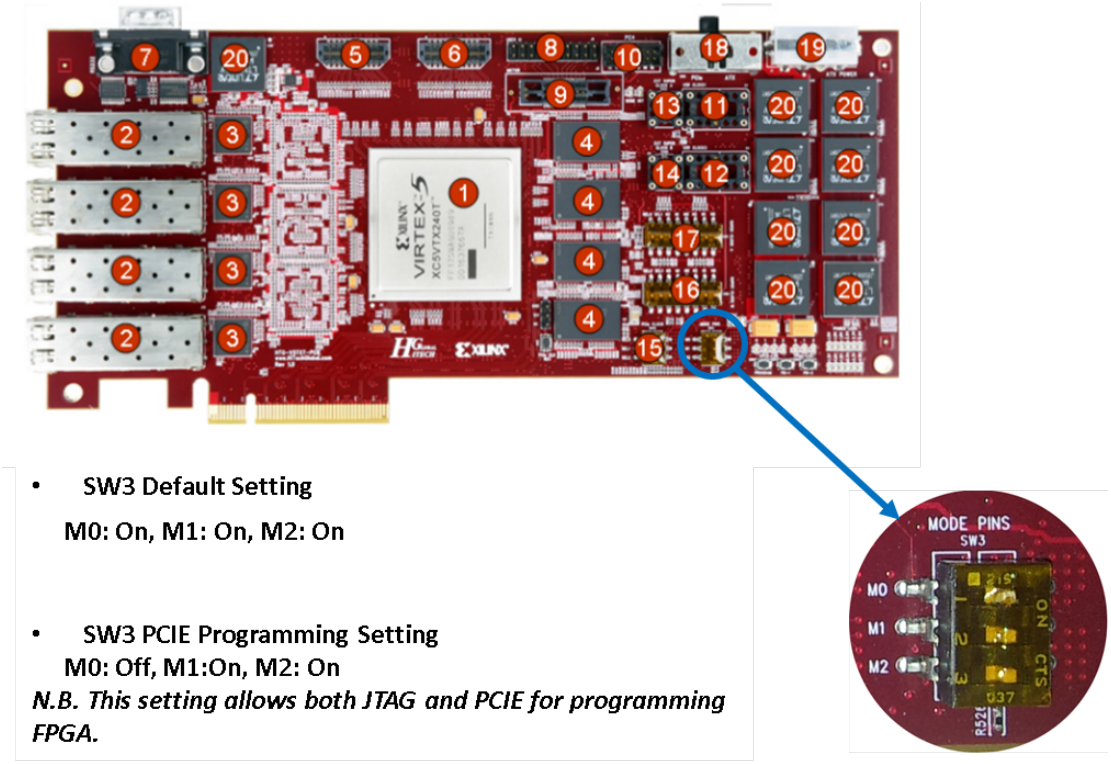

Fig. I.15 Configuración SW3 JTag/PCIe programing

Recordar que los cambios a la configuración de la tarjeta se deben realizar con la PC apagada y sin conexión a la corriente eléctrica para evitar daños en el hardware.

Una vez configurada la tarjeta conectar la PC a la corriente eléctrica y encender la misma.

Programación de la tarjeta

Antes que nada es necesario realizar algunos cambios en archivos del código fuente asociados a los proyectos NetFPGA utilizados, arreglando algunos errores reportados para la version de código fuente utilizada en este manual. Entonces en caso de trabajar con la versión 5.0.5 (recomendada) se deben realizar las siguientes modificaciones:

1. En el directorio NF_ROOT/projects/reference_nic/hw/ cambiar en el archivo *system.mhs* lo siguiente:

```
1 PORT axi_emc_0_Mem_DQ_pin = axi_emc_0_Mem_DQ, DIR = IO,
2 VEC = [*7*:0]
```

   cambiar por

```
1 PORT axi_emc_0_Mem_DQ_pin = axi_emc_0_Mem_DQ, DIR = IO,
2 VEC = [*31*:0]
```

2. En el directorio NF_ROOT/projects/reference_nic/hw/nf10 cambiar en el archivo *xflow.opt* lo siguiente:

```
-t *1*
```

cambiar por

```
-t *5*
```

(*) Los "*" no van

3. Modificar el driver utilizado por el proyecto ReferenceNIC:

   (a) Posicionarse en el directorio NF_ROOT/projects/reference_nic/sw/host/driver

   (b) Abrir el archivo nf10_phy_conf.c y comentar las líneas 217,219 y 240

Finalmente se debe compilar el proyecto. Para ello en una consola ejecutamos

```
# cd NetFPGA_ROOT/projects/reference_nic
# make
# cd cpld
# make
```

Este proceso puede llevar de 10 a 20 minutos.

Ahora programamos la tarjeta NetFPGA con el proyecto ReferenceNIC, para lo cual es necesario conectar el cable de programación JTAG a la tarjeta y a la PC. Luego en una consola se ejecuta lo siguiente:

```
# cd NetFPGA_ROOT
# cd tools/scripts
# ./impact_run
  ../../projects/reference_nic/bit/reference_nic.bit
  ../../projects/reference_nic/cpld/cpld.jed
```

(*) En caso de que aparezca el error "NF_ROOT or NF_DESIGN_DIR variable is not defined" abrir el script impact_run.sh y definir una variable de nombre NF_ROOT con el valor del directorio donde esta instalado el código fuente de NetFPGA.

Luego de finalizada la ejecución del comando reiniciar la máquina. Es sumamente importante que se reinicie y no que se apague, puesto que hasta el momento solo se ha programado el hardware en forma volátil. Esto quiere decir que de producirse un corte de corriente el misma se desprogramaría.

Una vez reiniciada la PC es necesario cargar el driver del ReferenceNIC en el kernel del sistema. Para ello posiblemente primero se deba compilar el driver. Para compilar el driver ejecutar en una consola:

```
# cd NetFPGA_ROOT/projects/reference_nic/sw/host/driver
# make
# insmod nf10.ko
```

Ahora vamos a programar una de las memorias flash de la tarjeta (en particular utilizaremos la unidad A con el proyecto ReferenceNIC, de forma de hacer persistente la programación de la tarjeta. Para ello en una consola ejecutamos:

```
# cd NF_ROOT/projects/reference_nic/bitfiles
# ./bit2bin.sh reference_nic.bit
```

Con lo anterior se genera un archivo binario a partir del bitfile del proyecto, con el que programaremos la memoria. Luego de este comando se deberían haber creado 3 archivos

- reference_nic.prm
- reference_nic.bin
- reference_nic.cfi

Luego ejecutamos lo siguiente:

```
# cd ../sw/host/pcieprog
# ./nf10_configure -b ../../../bitfiles/reference_nic.bin -f a
```

Apagamos la PC y la volvemos a prender. Ahora el chip FPGA de la tarjeta se encuentra programado a partir del contenido de la memoria flash A, con el proyecto que allí grabamos.

Para no cargar el driver del ReferenceNIC cada vez que se reinicie la máquina podemos modificar el archivo /etc/rc.local agregando la siguiente línea:

```
insmod NF_ROOT/preojects/reference_nic/sw/host/driver/nf10.ko
```

I.3.7 Instalación de Open vSwitch

Open vSwitch (ovs) lo instalaremos también desde su código fuente ejecutando lo siguiente en una consola:

```
# wget http://openvswitch.org/releases/openvswitch-2.3.0.tar.gz
# tar zxvf openvswitch-2.3.0.tar.gz
# cd openvswitch-2.3.0
#./boot.sh
#./configure --with-linux=/lib/modules/\$(uname -r)/build
```

Luego se compila el código fuente:

```
# make && make install
```

Se inserta en el kernel de linux el módulo de ovs:

```
# cd datapath/linux
# modprobe openvswitch
```

Podemos verificar que el módulo se insertó correctamente de la siguiente forma:

```
# lsmod | grep openvswitch

openvswitch    57291      0
gre          14236          1     openvswitch
```

Luego creamos el siguiente directorio necesario para la configuración del ovs:

```
# mkdir -p /usr/local/etc/openvswitch
```

Finalmente creamos la base de datos de configuración de ovs:

```
# ovsdb-tool create /usr/local/etc/openvswitch/conf.db
vswitchd/vswitch.ovsschema
```

Ya se está en condiciones de levantar cualquiera de los demonios de ovs. En particular para iniciar ovs debemos indicar una serie de parámetros como la base de datos donde se encuentra la configuración, en caso de utilizar una conexión segura con el controlador los certificados SSL correspondiente entre otras cosas. Para ello es conveniente crear un script como el siguiente:

```
ovsdb-server /usr/local/etc/openvswitch/conf.db \
--remote=punix:/usr/local/var/run/openvswitch/db.sock \
--remote=db:Open_vSwitch,Open_vSwitch,manager_options \
--private-key=db:Open_vSwitch,SSL,private_key \
--certificate=db:Open_vSwitch,SSL,certificate \
--bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert
--pidfile --detach --log-file

ovs-vsctl --no-wait init
ovs-vswitchd --pidfile --detach
ovs-vsctl show
```

Por simplicidad hemos comentado las líneas relacionadas a la parte de seguridad en la comunicación entre el ovs y el controlador.

Finalmente para automatizar la carga de los drivers de Open vSwitch al encender la PC, crear una carpeta de nombre ovs dentro del directorio /lib/modules/$(uname -r)/kernel/drivers como se muestra a continuación.

```
# mkdir /lib/modules/$(uname -r)/kernel/drivers/ovs
# cd /lib/modules/$(uname -r)/kernel/drivers/ovs
```

Posteriormente copiar los archivos de extensión .o y .ko dentro de la carpeta de instalación de ovs (llamemos le OVS_ROOT) en el siguiente directorio
OVS_ROOT/datapath/linux al directorio anteriormente creado.

Finalmente ejecutamos el siguiente comando para actualizar las listas de dependencias para cada módulo agregado.

```
# depmod
```

I.3.8 Instalación de software de ruteo Quagga

El software de ruteo Quagga puede instalarse con un gestor de aplicaciones (apt-get) o desde su código fuente; en este tutorial utilizamos esta ultima opción. El código fuente podemos obtenerlo desde el sitio oficial de quagga:

http://www.nongnu.org/quagga/

o desde el siguiente link:

http://www.nongnu.org/quagga/docs/docs-info.html#OSPFv2

Una vez descargado el código fuente descomprimimos el archivo *quagga.public-master.zip* en el directorio deseado. En este caso utilizaremos el directorio /home/mina como venimos haciendo para el resto del software instalado. Luego abrimos una consola y ejecutamos los siguientes comandos para compilar el código fuente:

```
1 # ./configure
2 # make
```

I.3.9 Instalación de agente SNMP

Para la instalación del agente SNMP se utiliza el gestor de paquetes de Ubuntu **apt-get** (en otras plataformas se podría utilizar por ejemplo opkg o yum). Por lo tanto para instalar el agente SNMP ejecutamos en una consola lo siguiente:

```
1 # apt-get install snmp snmpd
```