UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

EMULACIÓN DE UNA RED MPLS EN UNA ARQUITECTURA DE REDES DEFINIDAS POR SOFTWARE. ANÁLISIS DE FACTIBILIDAD TÉCNICA

TESIS
PARA OBTENER EL GRADO DE
INGENIERA EN REDES

PRESENTA
ELENA EUNISE BAACK VALLE

DIRECTOR DE TESIS
ING. JOSÉ ALBERTO DOMINGO INCERA DIÉGUEZ

ASESORES
MTI VLADIMIR VENIAMIN CABAÑAS VICTORIA
DR. JAIME SILVERIO ORTEGÓN AGUILAR
MSI. LAURA YÉSICA DÁVALOS CASTILLA
MTI. MELISSA BLANQUETO ESTRADA

CHETUMAL QUINTANA ROO, MÉXICO, MAYO DE 2015

UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

TRABAJO DE TESIS ELABORADO BAJO SUPERVISIÓN DEL COMITÉ DE ASESORÍA Y APROBADO COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE:

INGENIERA EN REDES

COMITÉ DE TRABAJO DE TESIS

DIRECTOR: _____

ING. JOSÉ ALBERTO DOMINGO INCERA DIÉGUEZ

ASESOR: _____

MTI. VLADIMIR VENIAMIN CABAÑAS VICTORIA

ASESOR: _____

DR. JAIME SILVERIO ORTEGÓN AGUILAR

UNIVERSIDAD DE QUINTANA ROO
DCI DIVISIÓN DE CIENCIAS E INGENIERÍA

CHETUMAL, QUINTANA ROO, MÉXICO

Agradecimientos

A Dios, por brindarme la oportunidad de vivir esta maravillosa experiencia llena de retos y aprendizajes.

A mi papá, a mis hermanas Sara, Zoila, Raquel y Betsa, a mis sobrinos y a mí cuñado Gustavo por brindarme su apoyo y cariño incondicional.

A mi asesor MTI. Vladimir Cabañas Victoria, por su gran apoyo incondicional durante la carrera y durante la realización de este trabajo, por sus consejos, motivación y paciencia.

Al Dr. José Alberto Incera Diéguez por darme la oportunidad de trabajar en sus proyectos de investigación durante los dos veranos de investigación que realicé, por su apoyo y motivación brindado para la realización de esta tesis. Por su tiempo, paciencia y por los conocimientos que me transmitió.

A la Ingeniera Mayumi Park Campos por darme la oportunidad y la experiencia de trabajar en el proyecto de Redes Definidas por Software y por los conocimientos transmitidos durante las estancias de verano.

Al Dr. Jaime Ortegón Aguilar y al MSI. Rubén González Elixavide por la ayuda y el tiempo que me brindaron durante el desarrollo de la tesis

A mi mejor amiga Lidia Ley y a todas las personas que me brindaron su apoyo y motivación para terminar la tesis.

Este trabajo fue financiado en la convocatoria 2014 "Apoyo a la Titulación" de la División de Ciencias e Ingenierías con recursos PIFI 2013

Dedicatoria

Con mucho cariño primeramente a Dios y a mis padres, a mi madre que aunque fue poco el tiempo que estuvo conmigo, siempre me demostró su gran amor y muy en especial a mi padre por apoyarme en cada etapa de mi vida, por sus consejos y por dame la oportunidad de tener una carrera universitaria. Gracias papá, te amo y te admiro mucho.

Con especial cariño para mis hermanas Sara, Zoila, Raquel y Betsa por su gran amor y apoyo incondicional, por sus consejos y por estar siempre conmigo en cada etapa de mi vida. Las amo mucho.

A mis amados sobrinos Mariela, Vanessa, Andrés, Isaí, Efraín, Cinthia, Ian, Iker y Adriel que son parte de mí.

A mí cuñado Gustavo por su gran apoyo durante el año que desarrolle la tesis.

A mis Asesores MTI. Vladimir Cabañas, Dr. José Incera y Dr. Jaime Ortegón por la confianza que tuvieron en mí para la realización de esta tesis.

A mis amigos Lilo, Heriberto, Aketzhaly, y compañero de la carrera Enrique Gudiño que siempre me animaron y estuvieron pendientes para que termine la tesis.

A la familia Cabañas Balam por su gran apoyo durante la carrera y en mis proyectos personales, por sus consejos y su cariño.

A todas las personas que influyeron con sus palabras de aliento y ánimo para que pudiera terminar la tesis.

Contenido

CAPÍTULO 1. INTRODUCCIÓN 2
1.1 Justificación 2
1.2 Objetivo General 2
1.3 Objetivo Específico 3
1.4 Alcance 3
CAPÍTULO 2. MPLS 5
2.1 Arquitectura MPLS 5
2.1.1 Dominio MPLS 5
2.1.2 Clase de Equivalencia de Reenvío (FEC, *Forwarding Equivalence Class)* 6
2.1.3 Plano de Control y Datos 6
2.1.4 Formato de Etiquetas 8
2.1.5 Camino de Conmutación de Etiquetas (*LSP, Label Switched Path*). 8
2.1.6 Base de Información de Etiquetas (LIB, *Label Information Base*) 10
2.1.7 Base de Información de Reenvío de Etiquetas (LFIB, *Label Forwarding Information Base*). 10
2.2 Funcionamiento de MPLS 12
CAPÍTULO 3. REDES DEFINIDAS POR SOFTWARE 16
3.1 Arquitectura de las Redes Definidas por Software 16
3.1.1 Protocolo OpenFlow 17
3.2.1 Conmutador OpenFlow 19
3.2.2 Controlador 20
3.3 Funcionamiento de OpenFlow 20
CAPÍTULO 4. DESARROLLO 23
4.1 Escenario 1. Basado en encaminamiento de paquetes con enrutamiento IP Linux 26
4.1.1 Configuración de red 26
4.1.2 Configuración de enrutamiento 28
4.2 Escenario 2. Encaminamiento de paquetes a través de la conmutación por etiquetas. 30
4.2.1 Construcción del escenario MPLS 30
4.2.2 Configuración de los Enrutadores de Conmutación por Etiquetas 31
4.3 Escenario 3. Encaminamiento de Paquetes a través del protocolo OpenFlow. 35
4.4 COMPROBACIÓN DE ENCAMINAMIENTO DE PAQUETES. 38
4.4.1 Comprobación de encaminamiento con enrutamiento IP en Linux. 38

4.4.2 Comprobación de encaminamiento de paquetes a través de la Conmutación por Etiquetas..... 40
4.4.3 Comprobación de encaminamiento de paquetes con OpenFlow..... 51
CAPÍTULO 5. CONCLUSIONES..... 54
BIBLOGRAFÍA..... 55
GLOSARIO..... 57
ANEXOS..... 58
Instalación del Kernel Linux-MPLS..... 58
Instalación de iproute2..... 59
Instalación de Quagga- LDP..... 60
Configuración del parche LDP..... 60
Configuración de Quagga-LDP..... 62

FIGURAS

Figura 1 Plano de control y plano de datos. 7
Figura 2 Cabecera MPLS (Rosen, Multiprotocol Label Switching Architecture, 2001). 8
Figura 3 reenvío IP. 12
Figura 4 Reenvío MPLS 13
Figura 5 Reenvío IP y MPLS (Pim Van Heuven) 14
Figura 6 Arquitectura de las Redes Definidas por Software (Fundation, 2012) 17
Figura 7 Elementos de OpenFlow 18
Figura 8 Estructura de flujo 20
Figura 9 Diagrama de flujo del funcionamiento de un conmutador y controlador OpenFlow (Vargas, 2014). 21
Figura 10 Escenario 1. Encaminamiento de paquetes con enrutamiento IP 26
Figura 11 Configuración de red del enrutador Cisco 2900 27
Figura 12 Configuración IP de RLinux1 27
Figura 13 Configuración IP de RLinux2 27
Figura 14 Acceso a la VTY de zebra 28
Figura 15 Acceso a la VTY de ospfd 28
Figura 16 Configuración IP de RLinux1 en zebra 28
Figura 17 Configuración IP de RLinux2 en Zebra 29
Figura 18 Configuración de enrutamiento IP con OSPF en los enrutadores Linux 29
Figura 19 Configuración de enrutamiento IP con OSPF en los enrutadores Cisco 29
Figura 20 Activación del forwarding en Linux 29
Figura 21 Escenario 2. Conmutación por Etiquetas 30
Figura 22 Configuración IP del LSRLinux en Linux 31
Figura 23 Acceso a la VTY de zebra 31
Figura 24 Configuración IP en zebra 32
Figura 25 configuración básica de los enrutadores Cisco 32
Figura 26 Configuración IP en enrutadores Cisco 33
Figura 27 Configuración de enrutamiento dinámico con OSPF en Linux 33
Figura 28 Configuración de enrutamiento dinámico con OSPF en Cisco. 33
Figura 29 Acceso a la VTY de ldpd 34
Figura 30 Activación de MPLS en Linux 34
Figura 31 Activación de CEF 34
Figura 32 Activación de MPLS a través del protocolo LDP en Cisco 34
Figura 33 Escenario 3. Encaminamiento de Paquetes a través del Protocolo OpenFlow 35
Figura 34 Descarga del controlador POX 36
Figura 35 Descarga e Instalación del conmutador OpenFlow 36
Figura 36 Configuración del camino de datos y el data-ID. 37
Figura 37 Comando para realizar la conexión del conmutador con el controlador 37
Figura 38 Comando para realizar la conexión del controlador con el conmutador OpenFlow 37
Figura 39 Inyección de tráfico UDP desde 10.4.0.2 a 10.1.0.2 38
Figura 40 Tabla de enrutamientoR1Linux 39
Figura 41 Tabla de enrutamiento R4 39

Figura 42Tabla de enrutamiento R1 40
Figura 43 Tabla de enrutamiento de LSRLinux 41
Figura 44 Captura de Wireshark: mensajes LDP Hello 41
Figura 45 Captura de Wireshark: establecimiento de la conexión entre pares LDP 42
Figura 46 show mpls ldp discovery en LSR1 42
Figura 47 show mpls ldp discovery en LSR2 42
Figura 48 show mpls ldp discovery en LSR3 43
Figura 49 Shoe mpls ldp discovery en LSRLinux 43
Figura 50 show mpls neighbor LSR1 43
Figura 51 show mpls ldp neighbor LSR2 44
Figura 52 show mpls ldp neighbor LSR3 44
Figura 53 show mpls ldp neighbor LSRLinux 44
Figura 54 Tabla de Información de Etiquetas LSR1 45
Figura 55 Tabla de Información de Etiquetas LSR2 45
Figura 56 Tabla de Información de Etiquetas LSR3 46
Figura 57 Tabla de Información de Etiquetas LSRLinux 46
Figura 58 Captura de Wireshark de paquetes MPLS 47
Figura 59 Tabla de información de reenvío de etiquetas de R1 48
Figura 60 Tabla de información de reenvío de etiquetas de R2 48
Figura 61 Tabla de información de reenvío de etiquetas de R3 48
Figura 62 Tabla de Información de Reenvío-FIB R1 49
Figura 63 Tabla de Información de Reenvío-FIB R2 50
Figura 64 Tabla de Información de Reenvío-FIB R3 50
Figura 65 Captura de Wireshark: Flujo de paquetes TCP entre controlador y conmutador OpenFlow 51
Figura 66 copia de archivos zebra y ldpd 62
Figura 67 Agregación del usuario quagga al grupo root 62
Figura 68 Cambio de permisos a los directorios /run y /usr/local/etc 62
Figura 69 Agregación del archivo libzebra al directorio ld.conf 63
Figura 70 reconfiguración del ldconfig 63

RESUMEN

Existen redes como MPLS y SDN que realizan el encaminamiento de paquetes basados en un plano de control y plano de datos y permiten la integración de nuevos servicios como la calidad de servicio, la ingeniera de tráfico, balancear cargas y crear políticas en la red que no son posibles con la redes tradicionales como IP.

Esta tesis describe la arquitectura, funcionamiento e implementación de las redes MPLS ya establecidas y las recientes Redes Definidas por Software (SDN) utilizando el protocolo OpenFlow a través de diversos escenarios de integración de estas tecnologías, con las cuales, se establecerán conexiones de red que permitan el encaminamiento de paquetes basado en un plano de control y un plano de datos.

Capítulo 1

Introducción

CAPÍTULO 1. INTRODUCCIÓN

La disponibilidad de ofrecer diferentes niveles de calidad de servicio, la gran demanda de aplicaciones en tiempo real, la priorización de paquetes, los cambios tecnológicos actuales en avalancha y la fusión de los servicios prestados, han propiciado la investigación, el estudio y el desarrollo de nuevas tecnologías con capacidades superiores a las redes IP como MPLS y SDN (Vieira & Guardieiro, 2005).

MPLS proporciona servicios de redes virtualizadas basadas en tráfico de aplicaciones y grupo de usuarios, que garantizan la calidad de servicio y la seguridad, permitiendo satisfacer los requerimientos de los usuarios, principalmente en los entornos corporativos que requieren soporte de aplicaciones críticas. Lamentablemente MPLS cuenta con una administración compleja, una infraestructura poco flexible y costosa de mantener.

Esta situación se complementa con la nueva arquitectura de red SDN (Redes Definidas por Software) que ofrece los mismos beneficios que las redes MPLS pero con una importante ventaja que es la administración simplificada e infraestructura de hardware más flexible y de bajo costo de implementación.

1.1 Justificación

La validez de la propuesta se demuestra a partir de la oportunidad de ofrecer nuevos servicios que no son posibles con las técnicas actuales de redes IP y la falta de capital humano con conocimientos de frontera que puedan ser inmediatamente absorbidos por las empresas.

1.2 Objetivo General

Este trabajo tiene por objeto explorar los fundamentos de tecnologías súper trascendentes en redes de comunicaciones como MPLS que es usada por la mayoría de los operadores y SDN que muy probablemente será una tecnología dominante en los próximos años.

1.3 Objetivo Específico

I. Conocer la arquitectura y el funcionamiento de MPLS y SDN.

II. Implementar tres escenarios de red:

1. Basado en enrutamiento IP sobre Linux.
2. Basado en MPLS/IP sobre Linux y sobre enrutadores Cisco a través del Protocolo de Distribución de Etiquetas (LDP).
3. Basado en el Protocolo OpenFlow para habilitar las Redes Definidas por Software.

1.4 Alcance

El alcance del proyecto será demostrar que se puede encaminar paquetes con enrutamiento en Linux, implementando MPLS así como el encaminamiento con SDN a través de la captura de paquetes con analizador de protocolos Wireshark.

Capítulo 2

Multiprotocolo de Conmutación por Etiquetas

CAPÍTULO 2. MPLS

MPLS es una tecnología de mapeo de etiquetas y reenvío de paquetes creado por la IETF (*Internet Engineering Task Force*) definido en el RFC 3031 (Rosen, Multiprotocol Label Switching Architecture, 2001), que integra el intercambio de etiquetas con el enrutamiento a nivel de red. El intercambio de etiquetas o mapeo implica el cambio del valor de la etiqueta en la cabecera del paquete a medida que este se desplaza de nodo en nodo dentro de la red. (Black, 2002).

2.1 Arquitectura MPLS

En esta sección se definen los elementos básicos de la arquitectura MPLS.

2.1.1 Dominio MPLS

El dominio MPLS está conformado por un conjunto de nodos contiguos que operan con enrutamiento y reenvío MPLS que se encuentran bajo un misma autoridad administrativa (Rosen, Multiprotocol Label Switching Architecture, 2001).

Los nodos MPLS son dispositivo de interconexión que soportan MPLS. Tienen conocimiento de los protocolos de control MPLS, operan en uno o más protocolos de enrutamiento de capa 3 y son capaces de reenviar paquetes con base a etiquetas. Opcionalmente, pueden reenviar paquetes nativos capa tres (Rosen, Multiprotocol Label Switching Architecture, 2001).

Los nodos MPLS se clasifican por su funcionamiento en el dominio MPLS de la siguiente manera:

- Nodo de tránsito (*Transit Node*): Recibe el PDU y usa la cabecera MPLS para tomar las decisiones de reenvío, asimismo realiza intercambio de etiquetas. También es llamado LSR (*Label Switching Router*) interior, o LSR de núcleo (Black, 2002).
- Nodo de borde (*Edge Node*): Conecta un dominio MPLS con un nodo fuera del dominio, ya sea porque no soporta MPLS, y/o porque está en un dominio diferente. Puede haber dos tipos, de acuerdo al rol que adopten en un momento dado:

- Nodo de egreso (*Egress Node*): maneja el tráfico que sale del dominio MPLS. También es llamado eLER (egress Label Edge Router)
- Nodo de ingreso (*Ingress Node*): maneja el tráfico que entra en el dominio MPLS. También es llamado iLER (ingress Label Edge Router)

El tráfico que se maneja entre los nodos de un el dominio MPLS es conocido como FEC.

2.1.2 Clase de Equivalencia de Reenvío (FEC, *Forwarding Equivalence Class)*

La FEC (por sus siglas en inglés) es un grupo de paquetes IP que son reenviados del mismo modo, es decir, por el mismo camino, con el mismo tratamiento de reenvío (Rosen, Multiprotocol Label Switching Architecture, 2001). Una FEC es asociada con una clase de datagramas; la clase depende de varios factores, como la dirección de destino o el tipo de tráfico en el datagrama. Basándose en la FEC, una etiqueta es negociada entre LSRs vecinos, desde el ingreso hasta el egreso del dominio MPLS (Black, 2002).

La etiqueta negociada entre los LSRs es un identificador corto, de longitud fija y de alcance local que identifica una FEC. La etiqueta que se coloca en un paquete en particular representa a la FEC a la que el paquete está asignado, y comúnmente, esta asignación se hace basándose en la dirección de red del destino, sin llegar a ser una codificación de la misma (Rosen, Multiprotocol Label Switching Architecture, 2001).

2.1.3 Plano de Control y Datos

Los nodos MPLS hacen la separación entre las funciones de control y envío de datos a través de un plano de control y plano de datos *(ver Figura 1).*

El plano de control determina la disponibilidad del acceso hacia una red destino, por lo tanto, contiene toda la información de direccionamiento de la capa 3 y facilita el intercambio de etiquetas entre LSRs vecinos utilizando un protocolo de señalización, este plano incluye la tabla o Base de Información de Etiquetas, (LIB, por sus siglas en inglés). La tabla LIB gestiona la base de información de las

etiquetas que administran los LSR. En ella solo se observa información de etiquetas y es utilizada por el protocolo de señalización para la gestión y el envío de etiquetas.

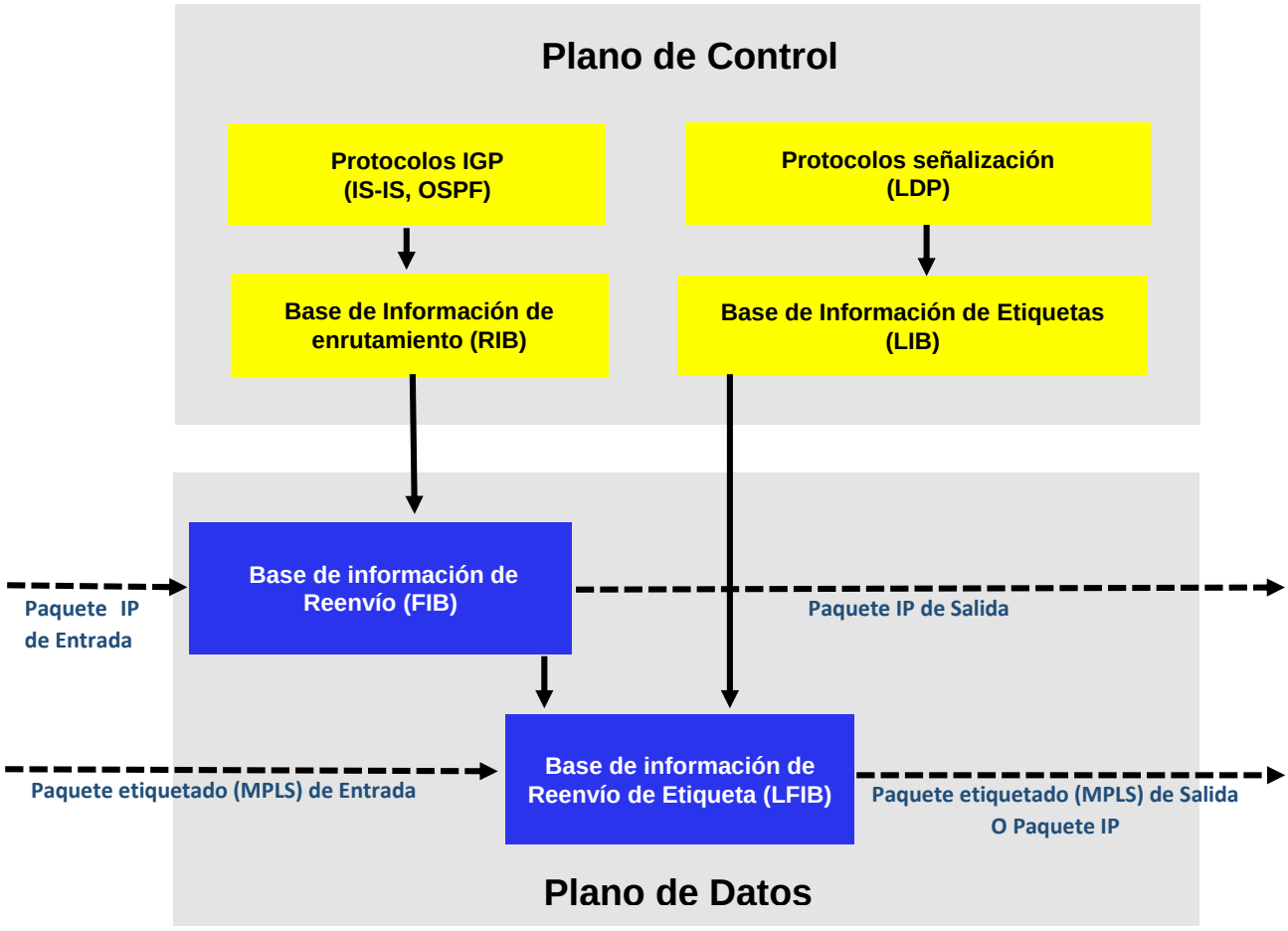


Figura 1 Plano de control y plano de datos.

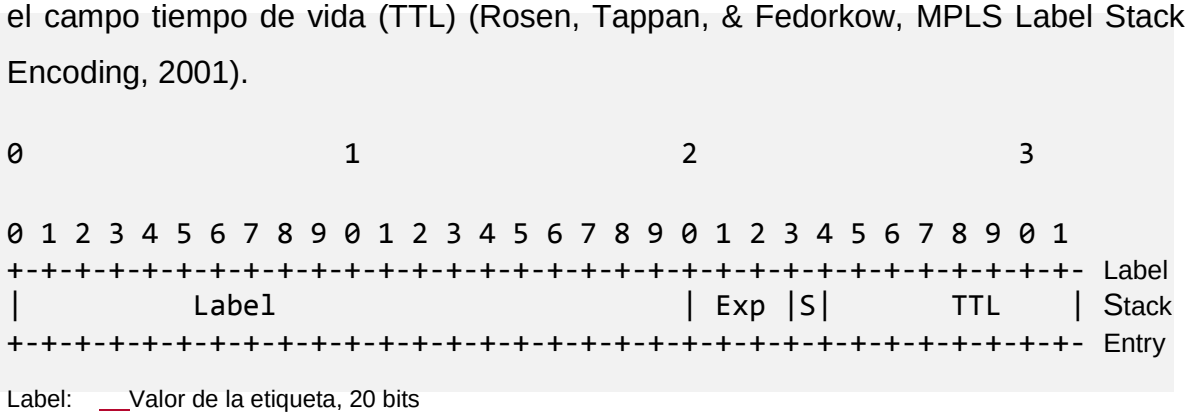
El plano de datos es el encargado de conmutar los paquetes, si la conmutación se basa etiquetas, utiliza la tabla de Información de Reenvío de Etiquetas, LFIB o si la conmutación se basa en paquetes IP se utiliza la tabla de información de reenvío, FIB. *(Ver figura 1*) (Ivana Cruza, 2013).

La conmutación basada en etiquetas del plano de datos hace referencia a un paquete al cual se le ha codificado una etiqueta. En algunos casos, la etiqueta se coloca en una cabecera de encapsulación que existe específicamente para este propósito, que se ubica entre las cabeceras de capa 2 y capa 3. En otros casos, la etiqueta puede colocarse en la cabecera de capa 2 o capa 3, siempre y cuando

exista un campo disponible como se define en el formato de etiquetas MPLS. (Rosen, Multiprotocol Label Switching Architecture, 2001).

2.1.4 Formato de Etiquetas

El formato de etiquetas de MPLS está definido por la RFC3032 (Rosen, Multiprotocol Label Switching Architecture, 2001) donde el protocolo MPLS añade una cabecera de 32 bits (*Figura 2*) entre las cabeceras de la capa de enlace de datos y la capa de red. Dentro de estos 32 bits, 20 bits están reservados para la etiqueta con valor local, 3 bits para uso experimental (TC), 1 bit en el campo (S) para el apilado jerárquico de etiquetas y los 8 bits restantes están reservados para el campo tiempo de vida (TTL) (Rosen, Tappan, & Fedorkow, MPLS Label Stack Encoding, 2001).



Label: Valor de la etiqueta, 20 bits
Exp: Uso Experimental, 3 bits
S: Parte inferior de la pila, 1 bit
TTL: Tiempo de vida, 8 bits

Figura 2 Cabecera MPLS (Rosen, Multiprotocol Label Switching Architecture, 2001).

Este formato de etiquetas permite que los paquetes se distribuyan en el dominio MPLS a través de un camino de conmutación de Etiquetas, LSP (por sus siglas en inglés).

2.1.5 Camino de Conmutación de Etiquetas (*LSP, Label Switched Path*).

LSP es el camino que siguen por la red todos los paquetes asignados a la misma FEC y se establecen de forma estática o dinámica (Juniper Networks, Inc., 2010).

En este trabajo se implementarán LSPs dinámicos, los cuales, utilizarán el protocolo de señalización LDP para establecerse y propagar información LSP a otros LSRs en el dominio MPLS. La elección del Protocolo de Distribución de Etiquetas, LDP se

basó en su señalización simple y de adyacencia rápida que establece automáticamente LSPs dentro de una red MPLS.

LDP se ejecuta en la parte superior de un protocolo de Puerta de Enlace Interna, IGP como IS-IS u OSPF, para esta tesis se manejó el protocolo OSPF, por lo tanto, se debe configurar LDP y OSPF en el mismo conjunto de interfaces. Después de que ambos protocolos están configurados, LDP comienza a transmitir y recibir mensajes LDP de descubrimiento a través de todas las interfaces habilitadas con LDP (Juniper Networks, Inc., 2010).

Cuando se utiliza IP como protocolo de red, los LSRs anuncian su estado mediante UDP/IP; si utiliza IP lo hace mediante multicast[1] a todos los enrutadores suscritos al domino MPLS. Cuando los enrutadores suscritos reciben el mensaje, que se comporta como un ping tradicional, estos los reenvían también. En los mensajes *ping* o *hello* vía UDP se transporta también la IP del servidor que se anuncia con el fin de que si alguno de los LSR desea establecer una sesión LDP, lo hará con el protocolo TCP y a la dirección IP anunciada.

Cuando el LSR adyacente recibe mensajes de descubrimiento LDP, se establece una sesión TCP subyacente. Entonces se crea una sesión LDP en la parte superior de la sesión TCP. La señalización TCP de tres vías asegura que la sesión LDP tenga conectividad bidireccional. Después de establecer la sesión LDP, los vecinos LDP deben mantener y terminar la sesión mediante el intercambio de mensajes.

Cualquier cambio de topología, como la falla de un enrutador, genera notificaciones LDP que puede terminar la sesión LDP o generar notificaciones LDP adicionales para propagar un cambio al LSP.

Los mensajes de notificación LDP generan la Base de Información de Etiquetas, LIB y la Base de Información de Reenvío de Etiquetas, LFIB que permiten a los LSRs

[1] Multicast en LDP. Se envía mensajes “hello” periódicamente mediante el protocolo UDP para ver si algún vecino quiere establecer una conexión LDP.

determinar la información del cambio de etiquetas de los siguientes saltos dentro del LSP

2.1.6 Base de Información de Etiquetas (LIB, *Label Information Base*)

LIB es la base de datos que almacena la información sobre las direcciones en tabla de rutas y se asigna una etiqueta a cada una (Black, 2002). La tabla contiene sólo información de etiquetas MPLS con valor local que manejan los LSR.

2.1.7 Base de Información de Reenvío de Etiquetas (LFIB, *Label Forwarding Information Base*).

Es usada para determinar cómo procesar los paquetes entrantes etiquetados, tal como determinar el próximo nodo que debe recibir el paquete. La LFIB es para MPLS lo que la tabla de rutas es para IP (Black, 2002). Los términos LIB y LFIB son usados por Cisco, sin embargo, existen otros términos empleados para definir estos elementos de información, que se explican a continuación (Black, 2002).

La especificación de MPLS usa tres elementos en lugar de la LFIB (Rosen, Multiprotocol Label Switching Architecture, 2001). Se define la LFIB como la combinación de estos tres elementos: NHLFE (*Next Hop Label Forwarding Entry*), ILM (*Incoming Label Map*) y FTN (*FEC-to-NHLFE map*).

2.1.7.1 NHLFE (*Next Hop Label Forwarding* Entry)

La NHLFE es usada cuando un paquete etiquetado es reenviado. Contiene la siguiente información:

• El próximo salto del paquete.

• La operación a realizar en la pila de etiquetas del paquete; que corresponde a una de las siguientes:

- Swap. Reemplaza la etiqueta en el tope de la pila con una nueva etiqueta específica.
- Pop. Elimina la etiqueta del tope de la pila.
- Push. Reemplaza la etiqueta del tope de la pila con una nueva etiqueta específica, y luego apila una o más nuevas etiquetas específicas.

La NHLFE también puede contener información acerca del método de encapsulación de enlace de datos, y de la forma de codificar la pila de etiquetas. (Rosen, Multiprotocol Label Switching Architecture, 2001).

2.1.7.2 FTN (*FEC-to-NHLFE Map*)

FTN correlaciona cada FEC a un conjunto de NHLFEs. Este es usado cuando los paquetes a reenviar llegan sin etiquetar pero deben ser etiquetados antes de ser reenviados. (Rosen, Multiprotocol Label Switching Architecture, 2001).

2.1.7.3 ILM (*Incoming Label Map*)

ILM mapea cada etiqueta entrante a un conjunto de NHLFEs. Es usado cuando los paquetes a reenviar llegan etiquetados. (Rosen, Multiprotocol Label Switching Architecture, 2001).

2.2 Funcionamiento de MPLS

En las redes IP cuando un paquete entra a la red cada enrutador examina la cabecera del paquete y determina de forma independiente el siguiente salto basada en su tabla de enrutamiento interna que contiene información acerca del próximo salto y la interfaz de salida para la dirección de destino de cada dirección. (*Ver Figura 3*)

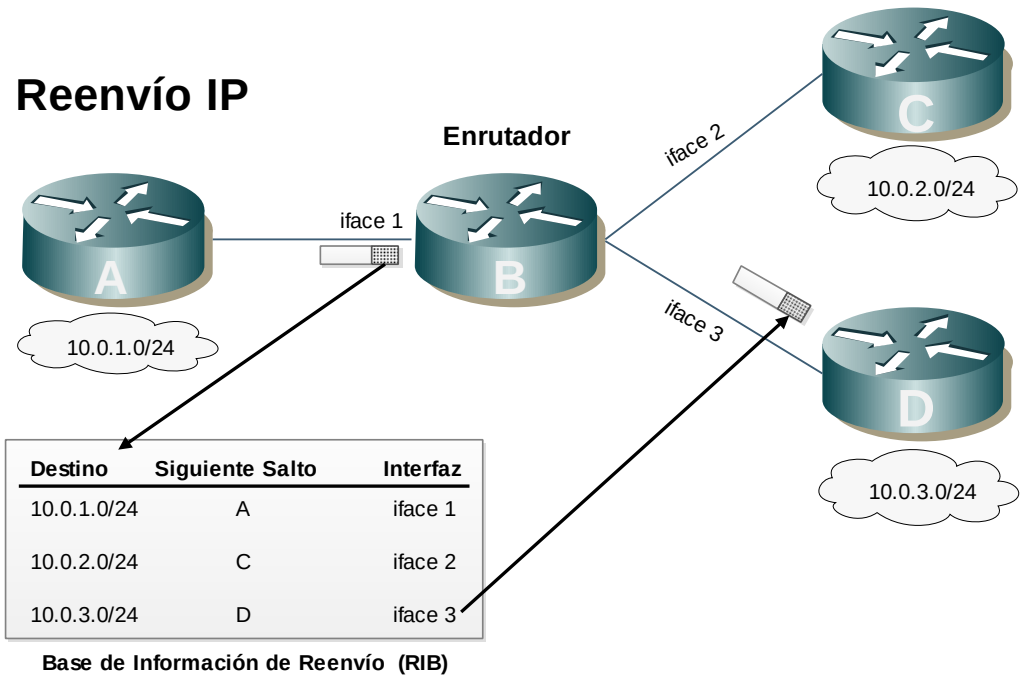


Figura 3 reenvío IP.

A comparación de IP, el reenvío MPLS consta de un mapeo de etiquetas que implica el cambio del valor de la etiqueta a medida que este se desplaza de nodo en nodo dentro en la red MPLS. Estas etiquetas son asignas a través de la tabla LIB desde plano de control por medio de un protocolo de distribución de etiquetas. Para realizar el reenvío, MPLS utiliza la tabla LFIB que contiene la información de las etiquetas de entrada (ILM), la información del próximo salto, la nueva etiqueta que se asignará en ese trayecto y la interfaz de salida (NHLF) (*Ver Figura 4*)

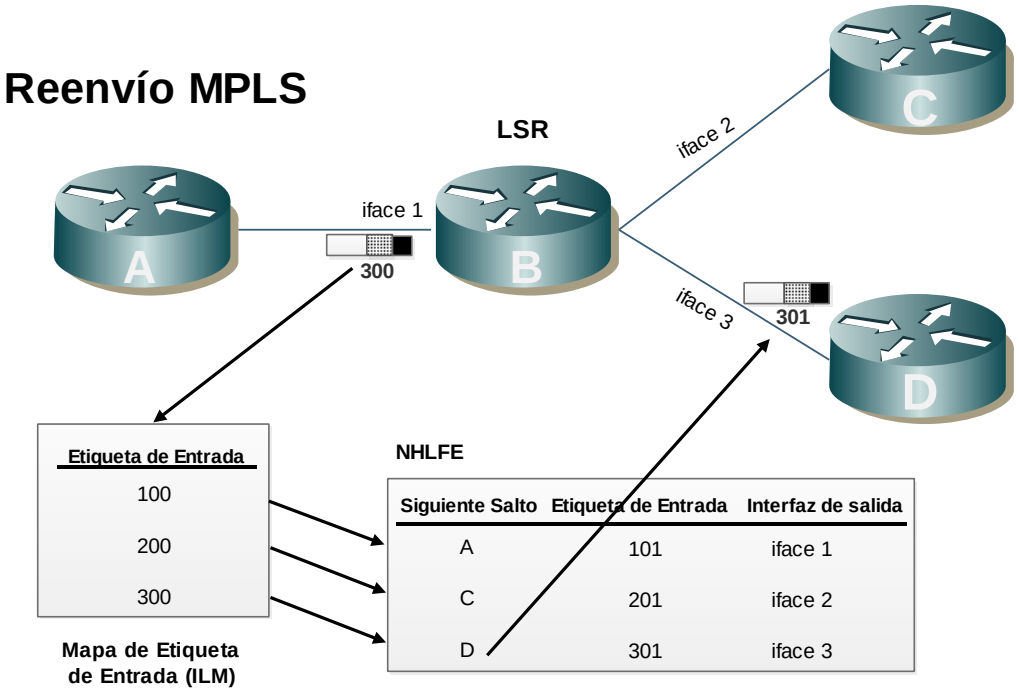


Figura 4 Reenvío MPLS

Cuando un paquete IP entra al dominio MPLS es procesado por un LER (Enrutador de Conmutación de Etiquetas) de ingreso que le asigna una etiqueta para que pueda ser encaminado a través de un LSP (Camino de Conmutación de Etiquetas) que está conformado por LSRs (Enrutador de Conmutación de Etiquetas) que hacen el cambio del valor de la etiqueta dentro del LSP. El LSP es generado un por un protocolo de distribución de etiquetas LDP. Cuando el paquete etiquetado llega al final del LSP generado, el LER de egreso retira la etiqueta para enviarlo de nuevo como paquete IP a su destino. (*Ver Figura 5).*

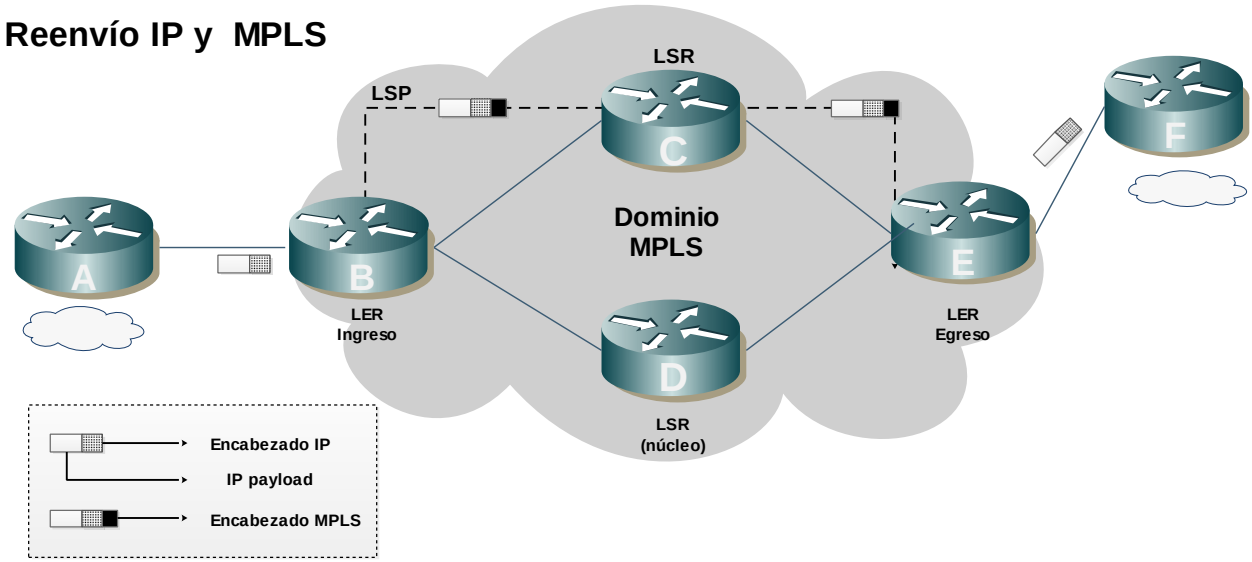


Figura 5 Reenvío IP y MPLS (Pim Van Heuven)

Se puede apreciar que el reenvío MPLS no examina los paquetes IP durante su encaminamiento a través del LSP si no que solamente lo hacen al momento en que el paquete IP ingresa o egresa del dominio MPLS, lo que hace que el reenvío de paquetes sea más eficiente que en IP.

Capítulo 3

Redes Definidas por Software

CAPÍTULO 3. REDES DEFINIDAS POR SOFTWARE

Las redes definidas por software son un relativamente nuevo enfoque en la programación de las redes en el cual el control se desvincula del hardware y se pasa a una plataforma de software. El plano de control es separado de la red física y puede controlar flujos por separado, dependiendo de las necesidades de las aplicaciones localizadas en capas superiores.

En las redes convencionales, cuando un paquete llega al conmutador las reglas incorporadas en el firmware propietario del conmutador le dicen hacia donde renviar el paquete. En una red SDN el administrador de la red utiliza software creado por terceros para darle forma al tráfico desde una consola o un servidor de control centralizado; el administrador puede cambiar las reglas de acuerdo a las necesidades como: priorizas, despriorizar o negar el paso a tipos específicos de paquetes (Park & Baack, 2012), por lo tanto, a continuación se explicará a detalle fundamentos de las SDN.

3.1 Arquitectura de las Redes Definidas por Software

SDN es una arquitectura emergente dinámica, manejable, rentable y adaptable, lo que es ideal para las aplicaciones actuales. La arquitectura SDN se basa en tres capas (ver Figura 7) (Hidalgo, 2014):

La capa de aplicación que permite crear aplicaciones para automatizar tareas de configuración, provisión y despliegue de nuevos servicios en la red

La capa de control que crea una vista centralizada de la topología de la red de datos, la cual permite gestionar el flujo de datos en la capa de infraestructura que está conformada por dispositivos de red.

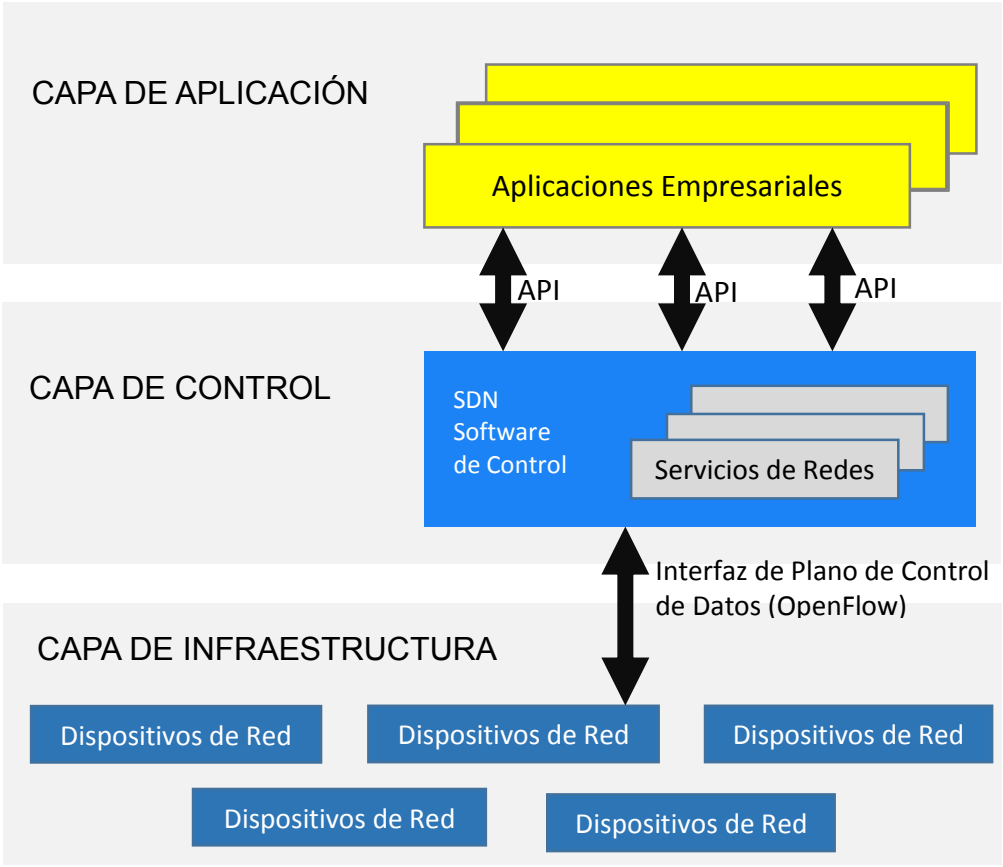


Figura 6 Arquitectura de las Redes Definidas por Software (Fundation, 2012)

En esta arquitectura el único lugar donde los estándares y protocolos entran en juego es en el lenguaje que utilizan para comunicar la capa de control con la capa de infraestructura. Es aquí donde OpenFlow toma importancia.

3.1.1 Protocolo OpenFlow

OpenFlow es un emergente estándar abierto definido por la Fundación de Redes Abiertas (ONF-*Open Networking Foundation*) desde 2011. Se trata de un protocolo de comunicación de estándar abierto que sienta las bases para las redes definidas por software (SDN). Está diseñado para dirigir el manejo y enrutamiento del tráfico en una red conmutada. La interfaz de OpenFlow ofrece acceso y comunicación entre las capas de control e infraestructura de la arquitectura de SDN. Al centralizar el control de los dispositivos de capa de infraestructura. (Hewlett-Packard Development Company, L.P., 2014)

OpenFlow simplifica la administración de las redes y la programación de dispositivos de redes como conmutadores y enrutadores, ya sean físicos o virtuales, permitiendo cambios dinámicos en el flujo de tráfico es decir, manipula la forma en que son procesados los flujos de datos (paquetes, tramas) y como resultado permite que la red tenga mayor capacidad de respuesta para las necesidades cambiantes.

OpenFlow consiste de tres partes (*ver Figura 7*):

1. Tablas de flujos en los nodos de conmutación

2. Un controlador

3. Un protocolo OpenFlow para que el controlador se comunique de forma segura con los nodos vía Secure Socket Layer (SSL).

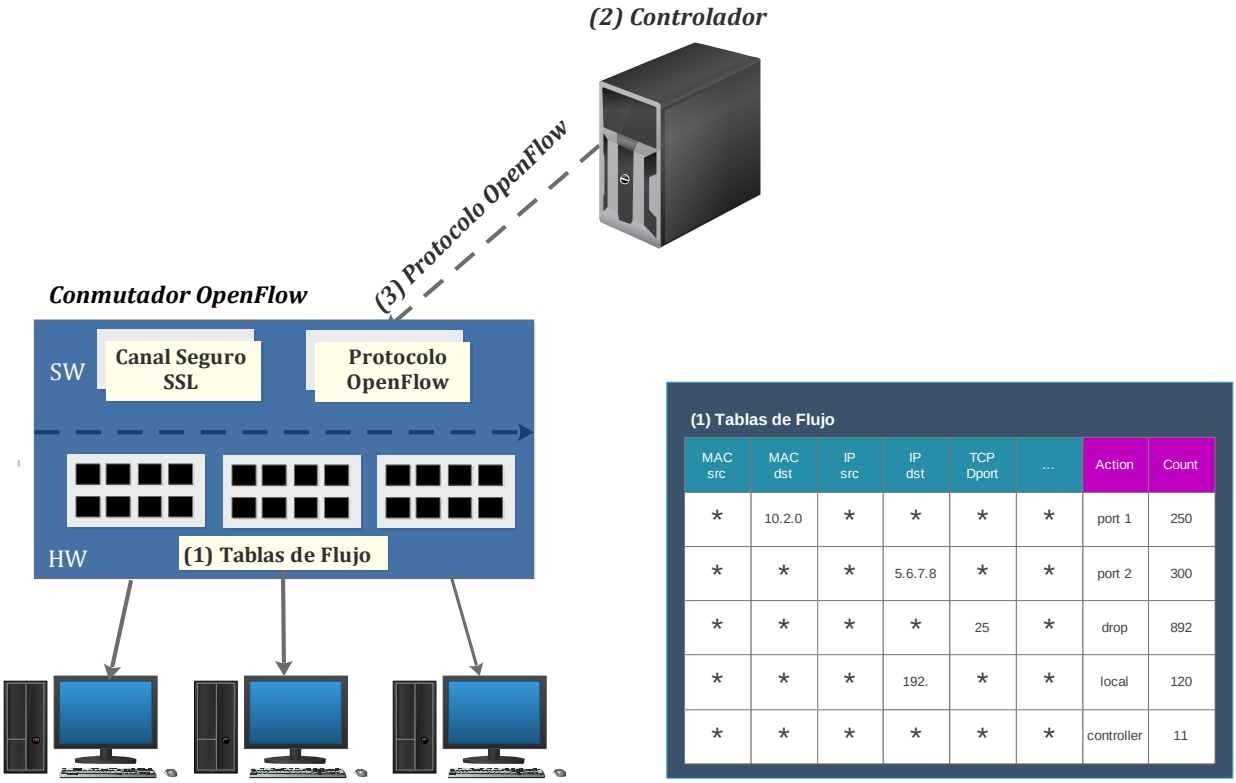


Figura 7 Elementos de OpenFlow

3.2.1 Conmutador OpenFlow

El conmutador OpenFlow es un dispositivo de conmutación comercial cuyo firmware soporta el protocolo OpenFlow y su forma de trabajo consiste en renviar el tráfico, basado en flujos, de acuerdo a la lógica de control que le imponga un controlador externo. Puede agregar, modificar o eliminar reglas en las tablas de flujo ya existentes y puede agregar o eliminar nuevas tablas de flujo con sus reglas (Park & Baack, 2012).

3.2.1.1 Tablas de Flujo.

Las tablas de flujo son una o varias tablas que contienen reglas, acciones y contadores que indican a los conmutadores como deben procesar los flujos a través de la red.

Las tablas de flujo se encuentran en los conmutadores y el controlador se comunica con los conmutadores por el protocolo de OpenFlow imponiendo ciertas políticas en los flujos. Estas políticas pueden permitir a los conmutadores establecer rutas óptimas sobre la red para características específicas, como pueden ser: ingeniería de tráfico, enrutamiento, autentificación, control de acceso, monitoreo, diagnóstico, menor número de saltos o reducción de latencia.

Los flujos se definen mediante la combinación de una tupla de 10 elementos que son parte de los encabezados de las capas 2, 3 y 4 del modelo OSI y son (Park & Baack, 2012) (*ver Figura 8*):

1. Puerto de entrada del conmutador
2. Dirección fuente MAC
3. Dirección destino MAC
4. Tipo Ethernet
5. VLAN ID
6. Dirección IP origen
7. Dirección IP destino
8. Protocolo IP
9. Puerto TCP/UDP origen

10. Puerto TCP/UDP destino

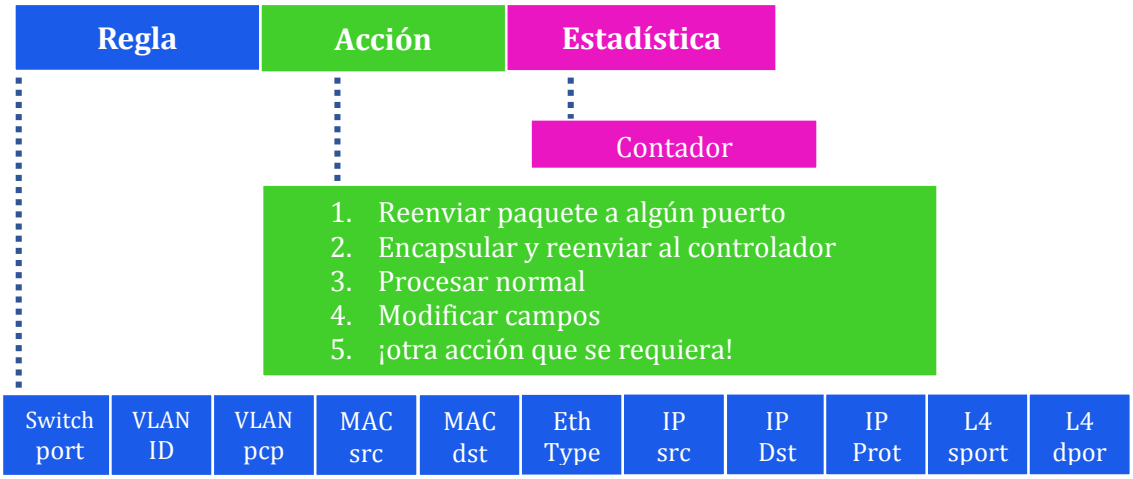


Figura 8 Estructura de flujo

3.2.2 Controlador

El controlador es un software lógicamente centralizado basado en SDN que mantiene una visión global de la red e indica al conmutador OpenFlow la serie de parámetros que definen a cada flujo, cómo los paquetes, que coinciden con el flujo deben ser procesados y por qué caminos serán enviados (Rouse, 2012). El resultado es un tráfico que fluye sobre la red en caminos óptimos predeterminados por el controlador y desarrollados por los conmutadores.

3.3 Funcionamiento de OpenFlow

Una vez que el conmutador recibe un paquete, se realizan las funciones mostradas en la *figura 9*, inicialmente el conmutador recibe el paquete, luego revisa las tablas de flujo y si la dirección se encuentra en las tablas envía el paquete, caso contrario se lo envía al controlador y este se hace responsable de indicarle al conmutador que debe hacer con el mismo (Vargas, 2014).

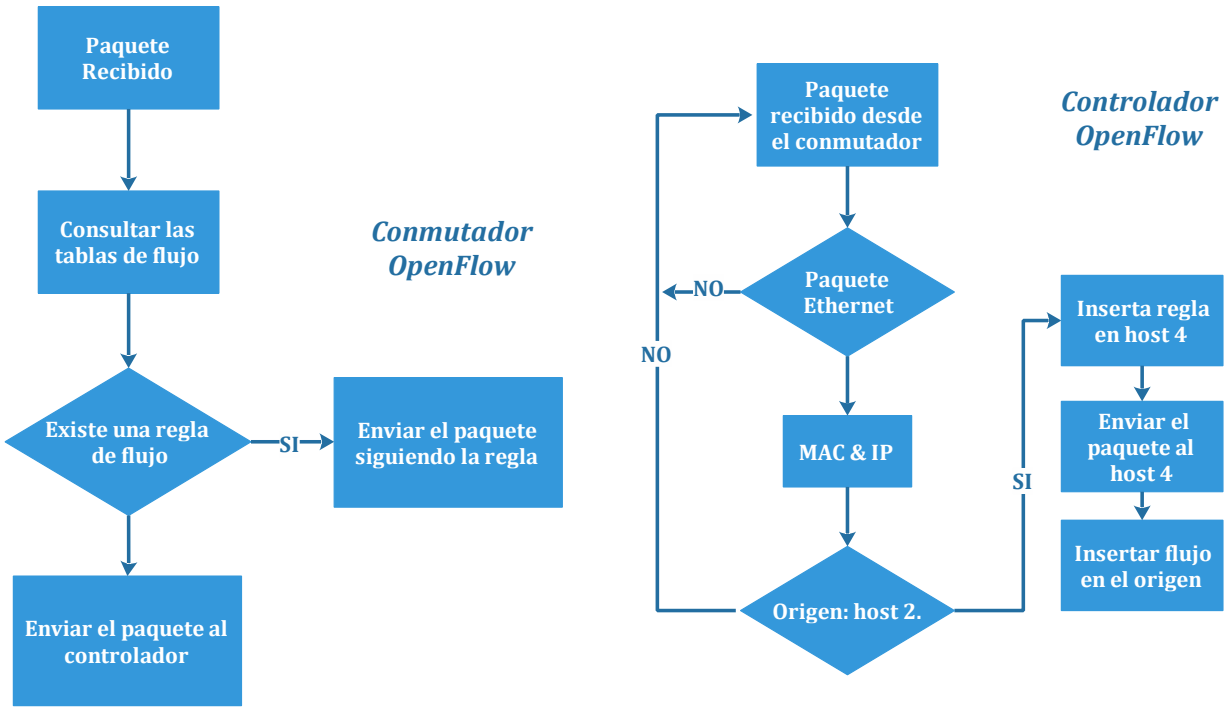


Figura 9 Diagrama de flujo del funcionamiento de un conmutador y controlador OpenFlow (Vargas, 2014).

El controlador recibirá el paquete y revisara si este es un paquete Ethernet, hay dos casos posibles:

- Si el paquete no es de Ethernet, el controlador lo descarta.
- Si el paquete es Ethernet, el controlador analizará las cabeceras del paquete para obtener la dirección de destino (dirección MAC y dirección IP). Después, el controlador inserta las reglas necesarias en los conmutadores para que pueda ser encaminado al destino.

Capítulo 4

Desarrollo

CAPÍTULO 4. DESARROLLO

En este capítulo se describe la implementación de tres escenarios de red en los cuales se van a implementar:

1.- Encaminamiento de paquetes a través de enrutamiento IP en Linux.

2.- Encaminamiento de paquetes a través de la conmutación por etiquetas.

3.- Encaminamiento de paquetes a través del protocolo OpenFlow.

Esta tesis solo maneja configuraciones básicas para el establecimiento de conexiones IP, MPLS y SDN para el encaminamiento de paquetes.

Los escenarios fueron implementados en ambientes controlados en GNU/Linux con licencia GNU GPL y con enrutadores Cisco. Para el desarrollo de los escenarios se utilizaron cuatro computadoras, tres computadoras de escritorio y una portátil con las siguientes características:

	Computadora 1	Computadora 2	Computadora 3	Computadora 4
Marca	DELL Dimension	HP Compaq Pro 6305 Small Form Facto	HP Compaq Pro 6305 Small Form Facto	Portátil HP EliteBook 8440P
Hardware	Procesador Intel x64	Procesador AMD 64 bits	Procesador AMD 64 bits	Procesador Intel core i5
RAM	3 GiB	4 GiB	4 GiB	4GiB
Disco Duro	80 GiB	1 TB	1 TB	250 GiB
Red	2 tarjetas de red de 1000 mbps	2 tarjetas de red de 1000 mbps	2 tarjetas de red de 1000 mbps	1 tarjeta de red 1000 mbps
Sistema Operativo	Ubuntu Server 12.04 LTS 64 bits núcleo 3.5.0-49	Ubuntu Server 12.04 LTS 64 bits núcleo 3.5.0-49	Ubuntu Server 12.04 LTS 64 bits núcleo 3.5.0-49	Windows 8
Función	Escenario 1:Enrutador Linux	Escenario 1:Enrutador Linux	Host C: realizar pruebas conexión a	Host D: realizar pruebas conexión

	Escenario 3: conmutador OpenFlow	Escenario 2. Enrutador de Conmutación de Etiquetas Escenario 3: controlador OpenFlow	todos los escenarios a través de ICMP e inyección de tráfico UDP	a todos los escenarios a través de ICMP e inyección de tráfico UDP

Tabla 1 Características técnicas de las computadores usadas.

Los enrutadores Cisco utilizados en los escenarios fueron cuatro con las siguientes características:

	Enrutador 1	Enrutador 2	Enrutador 3	Enrutador 4
Modelo	2800	2800	2800	2900
Memoria	DRAM 64 bits paridad habilitada 239K bytes de Memoria no volátil 62720K bytes de ATA CompacFlash (Red/write)	DRAM 64 bits paridad habilitada 239K bytes de Memoria no volátil 62720K bytes de ATA CompacFlash (Red/write)	DRAM 64 bits paridad habilitada 239K bytes de Memoria no volátil 62720K bytes de ATA CompacFlash (Red/write)	DRAM 64 bits con paridad habilitada ROM system Bootstrap, versión 15.0(1r)m15 255K de memoria no volátil. 250880k bytes ATA system CompactFlash 0 (Read/Write)
Interfaces	2 Interfaces FastEthernet 2 Interfaces seriales Low-speed (sync/async) 1 Módulo Virtual Private Network (VPN)	2 Interfaces FastEthernet 2 Interfaces seriales Low-speed (sync/async) 1 Módulo Virtual Private Network (VPN)	2 Interfaces FastEthernet 2 Interfaces seriales Low-speed (sync/async) 1 Módulo Virtual Private Network (VPN)	2 interfaces Gigabit Ethernet 2 interfaces Seriales (sync/async), 1 línea terminal 1 modulo para VPN
IOS	Cisco IOS software 2800, Software	Cisco IOS software 2800, Software	Cisco IOS software 2800, Software	Cisco IOS software, Software (c2900 –

	(C2800NM-ADVIPSERVICESK9-M), Version 12.4(24)T 6 RELEASE SOFTWARE (FC2)	(C2800NM-ADVIPSERVICESK9-M), Version 12.4(24)T 6 RELEASE SOFTWARE (FC2)	(C2800NM-ADVIPSERVICESK9-M), Version 12.4(24)T 6 RELEASE SOFTWARE (FC2)	Universal9-M),versión 15.1(4)M3, RELEASE SOFTWARE)
Función	Escenario 2: LSR	Escenario 2: LSR	Escenario 2: LSR	Enrutador IP en los tres escenarios.

Tabla 2 Características técnicas de los enrutadores usados

4.1 Escenario 1. Basado en encaminamiento de paquetes con enrutamiento IP Linux

Para la implementación del el primer escenario de red, se necesitó conocimientos previos en el manejo del sistema operativo de Linux y conocimientos de configuración básica de Cisco. Este escenario está compuesto por un enrutador Cisco 2900, dos enrutadores Linux y dos host (ver figura 10). El encaminamiento de paquetes se realizó a través del Protocolo de Gateway Interior OSPF.

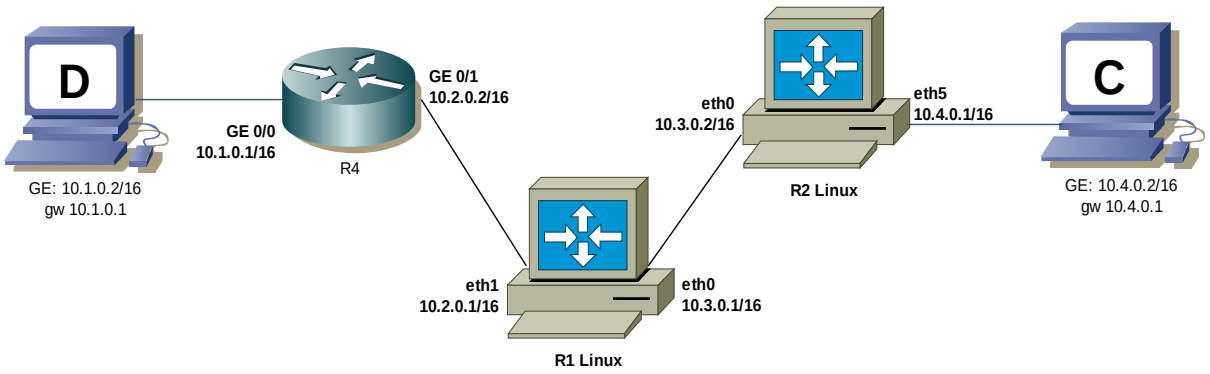


Figura 10 Escenario 1. Encaminamiento de paquetes con enrutamiento IP

En este escenario de enrutamiento tradicional, los paquetes se transmiten con una cabecera IP que incluye una dirección de origen y de destino. Cuando los enrutadores reciben un paquete, examina sus tablas de enrutamiento de la dirección del siguiente salto asociado con la dirección de destino del paquete y envía el paquete a la ubicación del siguiente salto. Esto sucede porque el reenvío de paquetes (plano de datos) y las decisiones de enrutamiento de alto nivel (plano de control) ocurren en el mismo dispositivo.

4.1.1 Configuración de red

Dispositivo	Interfaz	Red	Dirección IP	Puerta de Enlace
R4	GE 0/0	10.1.0.0/16	10.1.0.1/16	----------
	GE0/1	10.2.0.0/16	10.2.0.2/16	----------
RLinux1	eth1		10.2.0.1/16	----------
	eth0	10.3.0.0/16	10.3.0.1/16	----------
RLinux2	eth1		10.3.0.2/16	----------
	eth5	10.4.0.0/16	10.4.0.1/16	----------
PC C	eth0		10.4.0.2/16	10.4.0.1
PC D	eth0	10.1.0.0/16	10.1.0.2/16	10.1.0.1

Tabla 3 Direccionamiento del escenario 1

- Router Cisco 2900 series:

```
Router(config)#hostname R1
R1(config)#no ip domain-lookup
R1(config)#exit

R1(config)# interface GE 0/0
R1(config-if)# ip address 10.1.0.1 255.255.0.0
R1(config-if)# no shutdown
R1(config-if)# interface GE 0/1
R1(config-if)# ip address 10.2.0.2 255.255.0.0
R1(config-if)# no shutdown
```

Figura 11 Configuración de red del enrutador Cisco 2900

- Enrutadores R1 y R2 Linux:

```
RLinux1#ifconfig eth0 10.3.0.1 netmask 255.255.0.0

         broadcast 10.3.255.255 up eth0

RLinux1#ifconfig eth1 10.2.0.1 netmask 255.255.0.0

         broadcast 10.2.255.255 up eth1
```

Figura 12 Configuración IP de RLinux1

```
RLinux2#ifconfig eth2 10.3.0.2 netmask 255.255.0.0

         broadcast 10.3.255.255 up eth0

RLinux2#ifconfig eth5 10.4.0.1 netmask 255.255.0.0

         broadcast 10.4.255.255 up eth1
```

Figura 13 Configuración IP de RLinux2

4.1.2 Configuración de enrutamiento

Para el enrutamiento IP en Linux se utilizó el software de enrutamiento Quagga el cual consiste en un servicio de núcleo, llamado Zebra, que actúa como una capa de abstracción para el núcleo Unix subyacente y presenta el API ZServ a los clientes Quagga sobre Unix o TCP. Estos clientes son ZServ que implementan un protocolo de enrutamiento y comunican las actualizaciones de enrutamiento al demonio Zebra. Las Implementaciones existentes de ZServ son: ripd, ripngd, ospfd, ospf6d, bgpd, isisid, babeld, olsrd, ldpd, bfdd (Quagga Routing Suite, 2013).

La configuración consistió en la modificación de los archivos zebra.conf y ospfd.conf dentro del directorio usr/local/etc a través de sus VTY.

Para zebra:

```
lsr@LinuxPatchMPLS# cd /usr/local/etc
lsr@LinuxPatchMPLS:/cd /usr/local/etc#  zebra &
lsr@LinuxPatchMPLS:/cd /usr/local/etc # telnet localhost zebra
```

Figura 14 Acceso a la VTY de zebra

Para ldpd:

```
lsr@LinuxPatchMPLS:/cd /usr/local/etc#  ospfd &
lsr@LinuxPatchMPLS:/cd /usr/local/etc # telnet localhost ospfd
```

Figura 15 Acceso a la VTY de ospfd

```
zebra(config)#hostname RLinux1
RLinux1(config)#no ip domain-lookup
RLinux1(config)#exit

RLinux1(config)# interface eth1
RLinux1(config-if)# ip address 10.2.0.1/16
RLinux1(config)# interface eth0
RLinux1(config-if)# ip address 10.3.0.1/16
```

Figura 16 Configuración IP de RLinux1 en zebra

```
zebra(config)#hostname RLinux2
RLinux2(config)#no ip domain-lookup
RLinux2(config)#exit

RLinux2(config)# interface eth0
RLinux2(config-if)# ip address 10.3.0.2/16
RLinux2(config)# interface eth5
RLinux2(config-if)# ip address 10.4.0.1/16
```

Figura 17 Configuración IP de RLinux2 en Zebra

Para simplificar se anunció toda la red 10.0.0.0/8, para que cada enrutador anuncie solamente las subredes que tiene directamente conectadas.

```
ospfd(config)# router ospf
ospfd(config-router)# network 10.0.0.0/8 area 1
```

Figura 18 Configuración de enrutamiento IP con OSPF en los enrutadores Linux

```
R4(config)# router ospf 1
R4(config-router)# network 10.0.0.0 0.255.255.255 area 1
```

Figura 19 Configuración de enrutamiento IP con OSPF en los enrutadores Cisco

Para que puedan funcionar los protocolos de enrutamiento en Quagga se necesita activar el *forwarding* en Linux, para llevar a cabo esta acción se utilizó la instrucción de la sección pasada.

```
El fowarding temporal desde la consola:

            echo 1> /proc/sys/net/ipv4/ip_forward

De manera Permanente:

            nano /etc/sysctl.conf
            net.IPv4.IP_foward "1"
```

Figura 20 Activación del forwarding en Linux

4.2 Escenario 2. Encaminamiento de paquetes a través de la conmutación por etiquetas.

Para el despliegue de este escenario de se necesitó de previo conocimiento del funcionamiento y manejo de MPLS en el núcleo de Linux y MPLS en los dispositivos Cisco.

El segundo escenario consistió en la implementación de MPLS en modo trama con base al Protocolo de Distribución de Etiquetas, LDP sobre el núcleo MPLS-Linux y sobre enrutadores Cisco 2800.

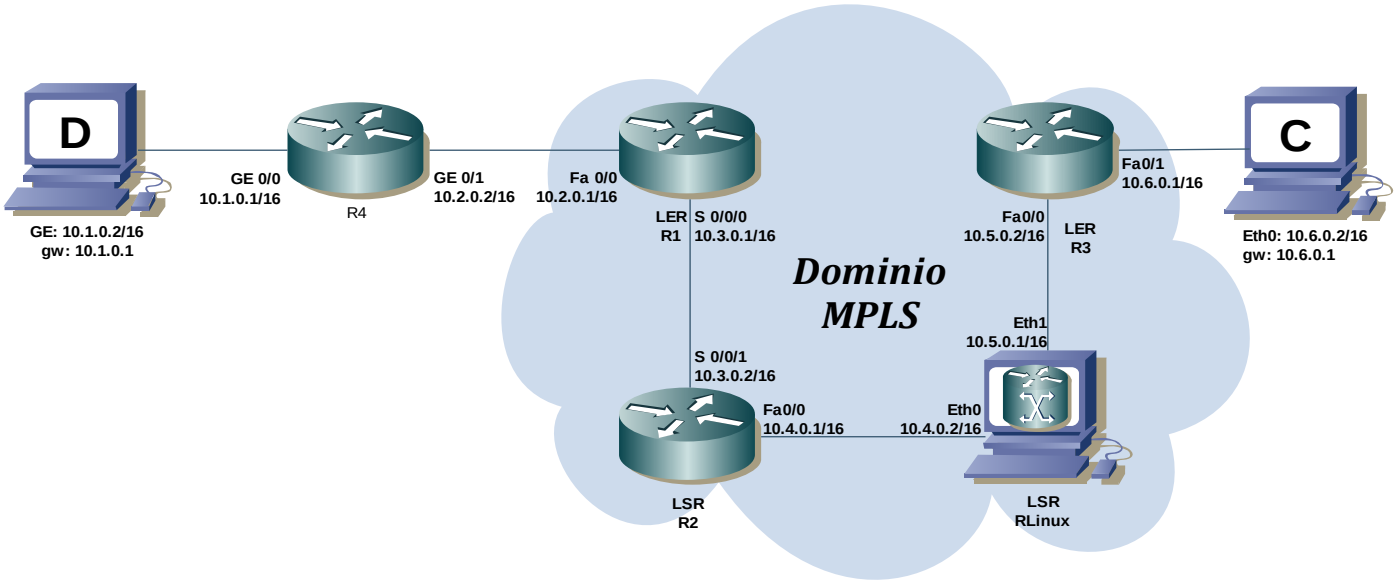


Figura 21 Escenario 2. Conmutación por Etiquetas

MPLS en modo trama consiste en anidar o apilar etiquetas, es decir, introducir una trama MPLS dentro de otra. En este encapsulado se introduce una etiqueta de 32 bits que permite a los enrutadores utilizar técnicas de conmutación como se expuso en el capítulo 2.

4.2.1 Construcción del escenario MPLS

Para la creación del dominio MPLS se utilizó como base el enrutador R1Linux del primer escenario para el LSRLinux, dejando la configuración de enrutamiento del enrutador Cisco 2900. Se agregó la nube MPLS integrada por 4 LSR de los cuales 3 son enrutadores Cisco y uno basado en el núcleo de Linux. Se implementó un LSP dinámico a través del protocolo LDP. Debido a que LDP se ejecuta en la parte superior de un protocolo de capa 3, se utilizó el protocolo OSPF el cual permite contener toda la información de direccionamiento de capa 3 a

través de las tablas de enrutamiento para facilitar el intercambio de etiquetas entre los LSRs vecinos utilizando LDP.

La construcción del LSRLinux consistió en la instalación del núcleo MPLS-Linux 3.9.0-rc3+ basado en la implementación original de James Leu's implementation (Leu, 2013), donde se activó el módulo Multiprotocol Label Switching. Posteriormente se instaló la aplicación Quagga versión 0.99.22 y se aplicó el parche LDP el cual necesito de configuraciones extras después de su aplicación para su funcionamiento, ya que Quagga originalmente no cuenta en su base con el protocolo LDP. Para mayor detalle de instalación y configuración consultar el anexo.

4.2.2 Configuración de los Enrutadores de Conmutación por Etiquetas

4.2.2.1 Configuración de Red

La implementación del LSRLinux se basó en los servicios de enrutamiento IP de Linux (IP forwarding) y zebra.

```
RLinux1#ifconfig eth0 10.4.0.1 netmask 255.255.0.0
         broadcast 10.4.255.255 up eth0
RLinux1#ifconfig eth1 10.5.0.1 netmask 255.255.0.0
         broadcast 10.5.255.255 up eth1
```

Figura 22 Configuración IP del LSRLinux en Linux

Se activó el forwarding en Linux de manera permanente para el funcionamiento de los protocolos OSPF y LDP en Quagga.

La configuración IP en el enrutador Linux se realizó a través de la vty de zebra.

Para zebra:

```
lsr@LinuxPatchMPLS# cd /usr/local/etc
lsr@LinuxPatchMPLS:/cd /usr/local/etc#  zebra &
lsr@LinuxPatchMPLS:/cd /usr/local/etc # telnet localhost zebra
```

Figura 23 Acceso a la VTY de zebra

```
zebra>enable
zebra#configure terminal
zebra(config)#hostname LSRLinux
LSRLinux(config)#no ip domain-lookup
LSRLinux(config)#exit

LSRLinux(config)# interface eth0
LSRLinux(config-if)# ip address 10.4.0.1/16
LSRLinux(config)# interface eth1
LSRLinux(config-if)# ip address 10.5.0.1/16
```

Figura 24 Configuración IP en zebra

Para los enrutadores Cisco 2960:

```
router>enable
router#configure terminal
Router(config)#hostname R1
R1(config)#no ip domain-lookup
R1config)#exit

Router(config)#hostname R2
R2(config)#no ip domain-lookup
R2config)#exit

Router(config)#hostname R3
R3(config)#no ip domain-lookup
R3config)#exit
```

Figura 25 configuración básica de los enrutadores Cisco

```
R1(config)# interface Fa0/0
R1(config-if)# ip address 10.2.0.1 255.255.0.0
R1(config-if)#no shutdown
R1(config)# interface S0/0/0
R1(config-if)# ip address 10.3.0.1 255.255.0.0
R1(config-if)#clock rate 2000000
R1(config-if)#no shutdown
R2(config)# interface S0/0/0
R2(config-if)# ip address 10.3.0.2 255.255.0.0
R2(config-if)#clock rate 2000000
R2(config-if)#no shutdown
R2(config)# interface Fa0/0
R2(config-if)# ip address 10.4.0.2 255.255.0.0
R2(config-if)#no shutdown
R2(config)# interface Fa0/0
R2(config-if)# ip address 10.5.0.2 255.255.0.0
R2(config-if)#no shutdown
R2(config)# interface Fa0/1
R2(config-if)# ip address 10.6.0.1 255.255.0.0
R2(config-if)#no shutdown
```

Figura 26 Configuración IP en enrutadores Cisco

4.2.2.2 Configuración de enrutamiento

Como en la configuración de enrutamiento del escenario 1 para simplificar se anunció toda la red 10.0.0.0/8, para que cada enrutador anuncie solamente las subredes que tiene directamente conectadas.

```
ospfd(config)# router ospf
ospfd(config-router)# network 10.0.0.0/8 area 1
```

Figura 27 Configuración de enrutamiento dinámico con OSPF en Linux

```
R1(config)# router ospf 1
R1(config-router)# network 10.0.0.0 0.255.255.255 area 1

R2(config)# router ospf 1
R2(config-router)# network 10.0.0.0 0.255.255.255 area 1

R3(config)# router ospf 1
R3(config-router)# network 10.0.0.0 0.255.255.255 area 1
```

Figura 28 Configuración de enrutamiento dinámico con OSPF en Cisco.

4.2.2.3 Configuración del Protocolo de Conmutación de Etiquetas.

En el núcleo de Linux la activación de MPLS fue a través del demonio LDPD:

```
lsr@LinuxPatchMPLS:/cd /usr/local/etc#  ldpd &
lsr@LinuxPatchMPLS:/cd /usr/local/etc # telnet localhost ldpd
```

Figura 29 Acceso a la VTY de ldpd

```
ldpd>enable
ldpd#configure terminal
ldpd# mpls ip

ldpd(config)# interface eth0
ldpd(config-if)# mpls ip
ldpd(config)# interface eth1
ldpd(config-if)# mpls ip
```

Figura 30 Activación de MPLS en Linux

En los enrutadores Cisco se activó el mecanismo CEF (*Cisco Express Forwarding*) para acelerar el proceso de conmutación de paquetes, que permite construir la tabla FIB. MPLS necesita que CEF esté activado, ya que utiliza la tabla FIB para construir la tabla LFIB que se utiliza para la conmutación de sus paquetes.

```
R1(config)# ip cef
R2(config)# ip cef
R3(config)# ip cef
```

Figura 31 Activación de CEF

```
R1(config)# mpls label protocol ldp
R1(config-if)# interface Fa0/0
R1(config-if)#mpls ip
R1(config-if)# interface S0/0/0
R1(config-if)# mpls ip

R2(config)# mpls lable protocol ldp
R2(config-if)# interface S0/0/0
R2(config-if)#mpls ip
R2(config-if# interface Fa0/0
R2(config-if)#mpls ip

R3(config)# mpls label protocol ldp
R3(config-if)# interface Fa0/0
R3(config-if)#mpls ip
R3(config-if)# interface Fa0/1
R3(config-if)# mpls ip
```

Figura 32 Activación de MPLS a través del protocolo LDP en Cisco

4.3 Escenario 3. Encaminamiento de Paquetes a través del protocolo OpenFlow.

El tercer escenario está compuesto por un conmutador OpenFlow, un controlador, el enrutador Cisco 2900 y dos host (*ver figura 33*). El encaminamiento de paquetes se realizó a través del protocolo OpenFlow.

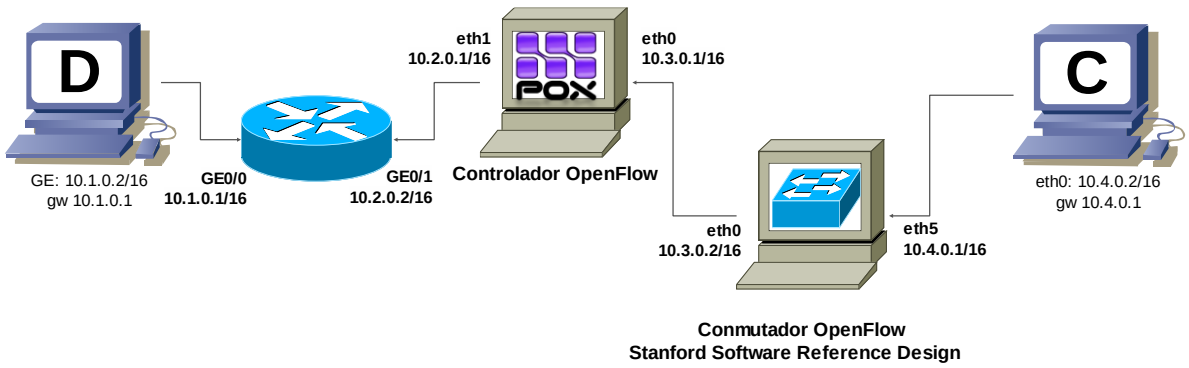


Figura 33 Escenario 3. Encaminamiento de Paquetes a través del Protocolo OpenFlow

En este escenario de red se realizará el encaminamiento de paquetes basado en el protocolo OpenFlow, el cual, consiste en que las instrucciones de reenvío se basan en flujos, es decir, que todos los paquetes trasmitidos compartirán una serie de características comunes. OpenFlow traslada la decisión de reenvío del conmutador al controlador.

Por lo tanto no se realizó ninguna configuración de enrutamiento en el conmutador más que la configuración de su dirección IP que utilizará para conectarse al controlador. Al conectarse con el controlador el conmutador quedará como una simple caja de reenvío ya que el control de la red se trasladó al controlador.

4.3.1 Controlador OpenFlow

El protocolo OpenFlow permite la comunicación entre el conmutador y el controlador. Los conmutadores deben ser capaces de establecer comunicación con los controladores, usando una dirección IP y puerto especificado por el usuario. Cuando el conmutador conozca la dirección del controlador y la conexión se establezca por primera vez, el conmutador iniciará una conexión TCP estándar con el controlador.

El controlador OpenFlow que se implementó en este escenario de red fue el controlador POX, el cual es una plataforma de desarrollo de código abierto para la creación de redes definidas por software (SDN) que consiste una aplicación de control basada en Python.

La instalación consistió:

```
$ git clone http://github.com/noxrepo/pox
$ cd pox
```

Figura 34 Descarga del controlador POX

4.3.2 Conmutador OpenFlow

El conmutador OpenFlow que se utilizó en este escenario fue el software de conmutación *OpenFlow Reference System* el cual añade la capacidad de conmutación OpenFlow a una PC Linux con varias tarjetas de red.

El conmutador *OpenFlow Reference Linux* es una aplicación de conmutación para pruebas de procesamiento lento y por lo tanto no puede aprovechar de los múltiples CPUs y requiere transiciones del núcleo a espacio de usuario.

La instalación consistió:

```
Descarga:
$ git clone git://gitosis.stanford.edu/openflow.git
$ cd openflow
$ git checkout -b openflow.v1.0 origin/release/1.0.0

Compilación:
$ ./boot.sh
$ ./configure
$ make
```

Figura 35 Descarga e Instalación del conmutador OpenFlow

4.3.3 Configuración para conexión del conmutador con el controlador OpenFlow

La configuración para la conexión entre el conmutador y el controlador OpenFlow consistió primeramente en la configuración del conmutador.

Se asumió que eth0 y eth5 se incluirán en el camino de datos OpenFlow. También tenemos que asignar camino de datos-id a este interruptor OpenFlow. El camino de datos-id debe ser único entre el conmutadores controlados por un solo controlador OpenFlow. Para la elección del identificador único de camino de datos se utilizó la dirección MAC de la interfaz eth0 ya que es la interfaz que lo conecta directamente con el controlador. El camino de datos-id es 2c44fd0bfdb para este interruptor OpenFlow.

```
# ./udatapath/ofdatapath --detach punix:/var/run/dp0 -d
2c44fd0bfdb -i eth1,eth5
```

Figura 36 Configuración del camino de datos y el data-ID.

La opción '-detach' hace que se ejecute en segundo plano como demonio. 'punix: / var / run / dp0' está especificando un archivo de socket de dominio Unix a través del cual se puede hablar con el conmutador.

El módulo del protocolo OpenFlow se ejecuta al realizarse la conexión con el controlador. Se asume que el controlador se está ejecutando en el puerto 10.3.0.1:6633, que es el predeterminado para OpenFlow.

```
#./secchan/ofprotocol unix:/var/run/dp0 tcp:10.3.0.1:6633
```

Figura 37 Comando para realizar la conexión del conmutador con el controlador

La conexión del controlador POX se realizó a través de la ejecución del código l2_learning

```
#./pox.py samples.pretty_log forwarding.l2_learning
```

Figura 38 Comando para realizar la conexión del controlador con el conmutador OpenFlow

Este componente hace que el conmutador OpenFlow actúe como un tipo de conmutador de aprendizaje de capa 2. Este componente aprende direcciones de capa 2, los flujos que instala son unidos-exactamente en tantos campos como sea posible. Por ejemplo, las diferentes conexiones TCP darán lugar a diferentes flujos que se estarán instalando.

4.4 COMPROBACIÓN DE ENCAMINAMIENTO DE PAQUETES.

Esta sección muestra los resultados obtenidos en los escenarios implementados a través del analizador de protocolos Wireshark. Lo que se buscó fue comprobar el establecimiento de las conexiones de las tres tecnologías implementadas a través de la captura de los paquetes que se trasmitieron durante su funcionamiento.

4.4.1 Comprobación de encaminamiento con enrutamiento IP en Linux.

En este escenario se pueden apreciar el establecimiento de la conexión IP a través del protocolo de enrutamiento OSPF que permitió la trasmisión de paquetes con una cabecera IP que incluye una dirección de origen y de destino. Como se puede observar en la *Figura 39,* los paquetes IP trasmitidos por el protocolo de enrutamiento no tienen campos que permitan manejar la calidad de servicio o ingeniería de tráfico que son servicios muy demandados actualmente, ya que las redes IP se limitan al enrutamiento de paquetes que solo ofrecen un servicio *de "mejor esfuerzo" a* diferencia de MPLS y SDN que si integran éstas características en su protocolo

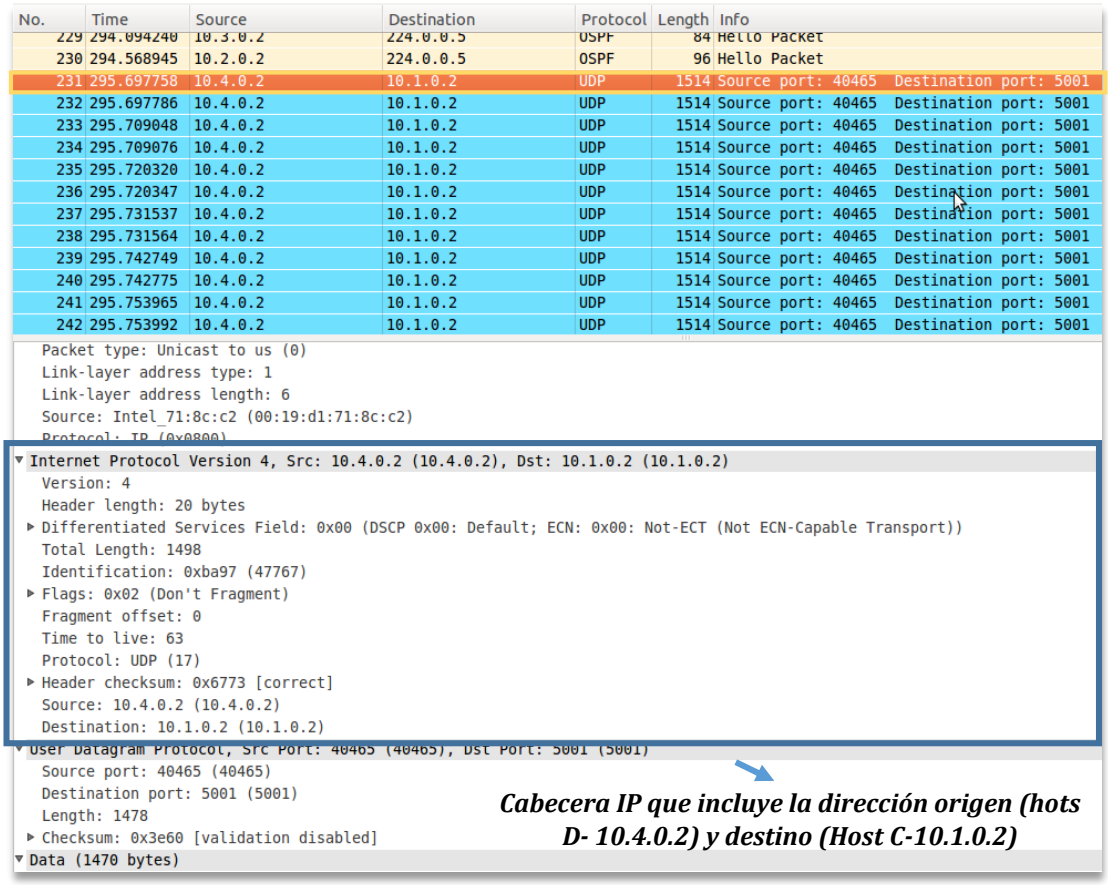


Figura 39 Inyección de tráfico UDP desde 10.4.0.2 a 10.1.0.2

A continuación se muestran las tablas de enrutamiento generadas por el protocolo de enrutamiento OSPF, que comprueban que la conexión de red fue establecida y configurada correctamente para el encaminamiento de paquetes. (Ver *Figuras 40 y 41).*

```
R1Linux# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O>* 10.1.0.0/16 [110/11] via 10.2.0.2, eth1, 00:27:46
O   10.2.0.0/16 [110/10] is directly connected, eth1, 00:33:22
C>* 10.2.0.0/16 is directly connected, eth1
O   10.3.0.0/16 [110/10] is directly connected, eth0, 00:34:03
C>* 10.3.0.0/16 is directly connected, eth0
O>* 10.4.0.0/16 [110/20] via 10.3.0.2, eth0, 00:30:10
C>* 127.0.0.0/8 is directly connected, lo
K>* 169.254.0.0/16 is directly connected, eth0
R1Linux#
```

Figura 40 Tabla de enrutamientoR1Linux

```
R4#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override

Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 7 subnets, 2 masks
C        10.1.0.0/16 is directly connected, GigabitEthernet0/0
L        10.1.0.1/32 is directly connected, GigabitEthernet0/0
C        10.2.0.0/16 is directly connected, GigabitEthernet0/1
L        10.2.0.2/32 is directly connected, GigabitEthernet0/1
O        10.3.0.0/16 [110/782] via 10.2.0.1, 01:17:07, GigabitEthernet0/1
O        10.4.0.0/16 [110/1563] via 10.2.0.1, 01:17:07, GigabitEthernet0/1
O        10.5.0.0/16 [110/1564] via 10.2.0.1, 01:00:22, GigabitEthernet0/1
R4#
```

Figura 41 Tabla de enrutamiento R4

4.4.2 Comprobación de encaminamiento de paquetes a través de la Conmutación por Etiquetas.

Este escenario de red demuestra el encaminamiento de paquetes a través de la arquitectura MPLS basada en dos planos, el plano de control (utilizado por los protocolos de enrutamiento IP y los protocolos de gestión de MPLS) y el plano de datos donde se realiza la conmutación de los paquetes.

4.4.2.1 Plano de Control.

El plano de control utiliza información de los protocolos de enrutamiento IP a través de las tablas de enrutamiento (*show ip route*) y la información del protocolo de gestión de MPLS se genera a través de la adyacencia de los pares LDP (*show mpls ldp discovery* y *show mpls ldp neighbor*) y de la tabla LIB (*show mpls ldp bindings*) que almacena información de las direcciones IP en una tabla de rutas y le asigna una etiqueta a cada una.

4.4.2.1.1 Tablas de enrutamiento (show ip route)

El establecimiento del LSP se realizó a través del Protocolo de Distribución de Etiquetas (LDP). En este ambiente el LSP se configuró por default consultando la información de las tablas de enrutamiento de los LSR construidas por el protocolo de enrutamiento OSPF como se muestra en las *figuras 42y 43*

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     10.0.0.0/16 is subnetted, 6 subnets
C       10.2.0.0 is directly connected, FastEthernet0/0
C       10.3.0.0 is directly connected, Serial0/0/0
O       10.1.0.0 [110/2] via 10.2.0.2, 01:17:14, FastEthernet0/0
O       10.6.0.0 [110/793] via 10.3.0.2, 00:08:38, Serial0/0/0
O       10.4.0.0 [110/782] via 10.3.0.2, 00:20:02, Serial0/0/0
O       10.5.0.0 [110/792] via 10.3.0.2, 00:10:12, Serial0/0/0
R1#
```

Figura 42Tabla de enrutamiento R1

```
Router# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O>* 10.1.0.0/16 [110/793] via 10.4.0.2, eth0, 00:05:12
O>* 10.2.0.0/16 [110/792] via 10.4.0.2, eth0, 00:05:12
O>* 10.3.0.0/16 [110/791] via 10.4.0.2, eth0, 00:05:12
O   10.4.0.0/16 [110/10] is directly connected, eth0, 00:05:23
C>* 10.4.0.0/16 is directly connected, eth0
O   10.5.0.0/16 [110/10] is directly connected, eth1, 00:05:21
C>* 10.5.0.0/16 is directly connected, eth1
O>* 10.6.0.0/16 [110/11] via 10.5.0.2, eth1, 00:03:46
C>* 127.0.0.0/8 is directly connected, lo
Router#
```

Figura 43 Tabla de enrutamiento de LSRLinux

4.4.2.1.2 Información de adyacencia de LDP (Show mpls ldp discovery y show mpls ldp neighbor)

La adyacencia de LDP consistió en la configuración de MPLS en cada interfaz y en el descubrimiento de los LSRs a través del envío periódico de mensajes LDP *Hello* para establecer conexiones LDP. Los LSRs enviaron mensajes LDP *Hello,* como paquetes UDP a la dirección *multicast* MPLS 224.0.0.2 y sobre el puerto de destino 646. (*Figura 44*).

2965	1439.254508	10.5.0.2	224.0.0.2	LDP	78	Hello Message
2966	1440.785164	10.4.0.1	224.0.0.2	LDP	78	Hello Message
2967	1440.785212	10.5.0.1	224.0.0.2	LDP	78	Hello Message
2968	1440.913440	10.4.0.2	224.0.0.2	LDP	78	Hello Message

Figura 44 Captura de Wireshark: mensajes LDP Hello

Para el establecimiento de la conexión entre los pares LDP se abrió una sesión TCP en el puerto 646 para que los enrutadores vecinos sean adyacentes. También se puede observar los mensajes *Initialization Message, Initialization Message Keep Alive Message, Label Mapping Message* y *Address Message Label Mapping,* donde los LSRs distribuyen información de sus etiquetas y direcciones IP a través del protocolo LDP durante el establecimiento de la sesión TCP (ver *Figura 45*).

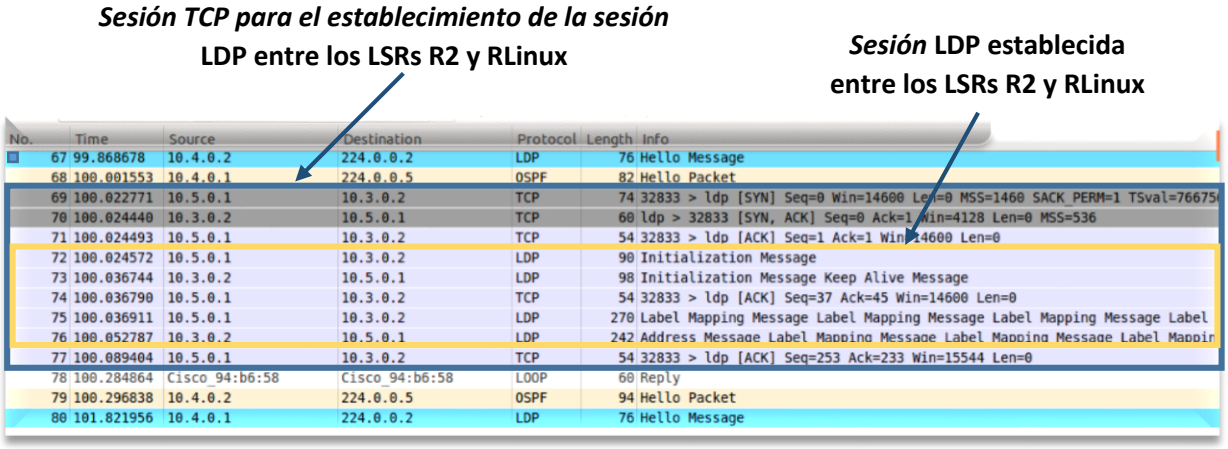


Figura 45 Captura de Wireshark: establecimiento de la conexión entre pares LDP

El comando *show mpls ldp discovery* muestra información del identificador LDP local, la interfaz y dirección IP de los LSRs directamente conectados, descubiertos a través de la adyacencia LDP. Ver *figura 46, 47, 48 y 49.*

```
R1#show mpls ldp discovery
 Local LDP Identifier:
    10.3.0.1:0
    Discovery Sources:
    Interfaces:
        Serial0/0/0 (ldp): xmit/recv
            LDP Id: 10.3.0.2:0; no host route
R1#
```

Figura 46 show mpls ldp discovery en LSR1

```
R2#show mpls ldp discovery
 Local LDP Identifier:
    10.3.0.2:0
    Discovery Sources:
    Interfaces:
        FastEthernet0/0 (ldp): xmit/recv
            LDP Id: 10.5.0.1:0; no host route
        Serial0/0/0 (ldp): xmit/recv
            LDP Id: 10.3.0.1:0; no host route
R2#
```

Figura 47 show mpls ldp discovery en LSR2

```
R3#show mpls ldp discovery
 Local LDP Identifier:
    10.5.0.2:0
    Discovery Sources:
    Interfaces:
        FastEthernet0/0 (ldp): xmit/recv
            LDP Id: 10.5.0.1:0; no host route
R3#
```

Figura 48 show mpls ldp discovery en LSR3

```
ldpd# show mpls ldp discovery
 Local LDP Identifier:
    10.5.0.1:0
    Discovery Sources:
    Interfaces:
        eth1 (ldp): xmit/recv
            LDP Id: 10.5.0.2:0
        eth0 (ldp): xmit/recv
            LDP Id: 10.3.0.2:0
ldpd#
```

Figura 49 Shoe mpls ldp discovery en LSRLinux

El comando *show mpls ldp neighbor* muestra información detallada de los vecinos LDP como el identificador LDP local y remoto, la conexión TCP establecida, el estado operacional y tiempo de conexión de las sesiones establecidas. Por defecto los LSRs utilizaron la técnica de Downstream para el anuncio de sus etiquetas. Ver *figura 50, 51, 52 y 53.*

```
R1#show mpls ldp neighbor
    Peer LDP Ident: 10.3.0.2:0; Local LDP Ident 10.3.0.1:0
        TCP connection: 10.3.0.2.50661 - 10.3.0.1.646
        State: Oper; Msgs sent/rcvd: 204/197; Downstream
        Up time: 02:35:43
        LDP discovery sources:
          Serial0/0/0, Src IP addr: 10.3.0.2
        Addresses bound to peer LDP Ident:
          10.3.0.2        10.4.0.2
R1#
```

Figura 50 show mpls neighbor LSR1

```
R2#show mpls ldp neighbor
    Peer LDP Ident: 10.3.0.1:0; Local LDP Ident 10.3.0.2:0
        TCP connection: 10.3.0.1.646 - 10.3.0.2.50661
        State: Oper; Msgs sent/rcvd: 206/213; Downstream
        Up time: 02:43:34
        LDP discovery sources:
          Serial0/0/0, Src IP addr: 10.3.0.1
        Addresses bound to peer LDP Ident:
          10.3.0.1        10.2.0.1
R2#
```

Figura 51 show mpls ldp neighbor LSR2

```
R3#show mpls ldp neighbor
    Peer LDP Ident: 10.5.0.1:0; Local LDP Ident 10.5.0.2:0
        TCP connection: 10.5.0.1.646 - 10.5.0.2.29981
        State: Oper; Msgs sent/rcvd: 50/46; Downstream
        Up time: 00:36:21
        LDP discovery sources:
          FastEthernet0/0, Src IP addr: 10.5.0.1
        Addresses bound to peer LDP Ident:
          10.5.0.1        10.4.0.1
R3#
```

Figura 52 show mpls ldp neighbor LSR3

```
ldpd# show mpls ldp neighbor
    Peer LDP Ident: 10.3.0.2:0; Local LDP Ident 10.5.0.1:0
        TCP connection: 10.3.0.2.646 - 10.5.0.1.34309
        state: OPERATIONAL; Msgs sent/rcvd: 13/9; Downstream
        Up time: 00:47:11
        LDP discovery sources:
          eth0, Src IP addr: 10.4.0.2
        Addresses bound to peer LDP Ident:
          10.3.0.2          10.4.0.2
    Peer LDP Ident: 10.5.0.2:0; Local LDP Ident 10.5.0.1:0
        TCP connection: 10.5.0.2.29981 - 10.5.0.1.646
        state: OPERATIONAL; Msgs sent/rcvd: 57/62; Downstream
        Up time: 00:47:22
        LDP discovery sources:
          eth1, Src IP addr: 10.5.0.2
        Addresses bound to peer LDP Ident:
          10.5.0.2          10.6.0.1
ldpd#
```

Figura 53 show mpls ldp neighbor LSRLinux

4.4.2.1.3 Tabla de información de Etiquetas-LIB (show mpls ldp bindings)

Como último elemento del plano de control, el comando *show mpls ldp bindings* permite visualizar la información de la tabla LIB, la cual contiene las asociaciones de etiquetas con las redes de destino que solo tienen significado local en el LSR, es decir, las etiquetas asignadas por un LSR no tienen en principio relación con las asignadas por el siguiente LSR al mismo destino, ver *Figuras 54, 55, 56 y 57.*

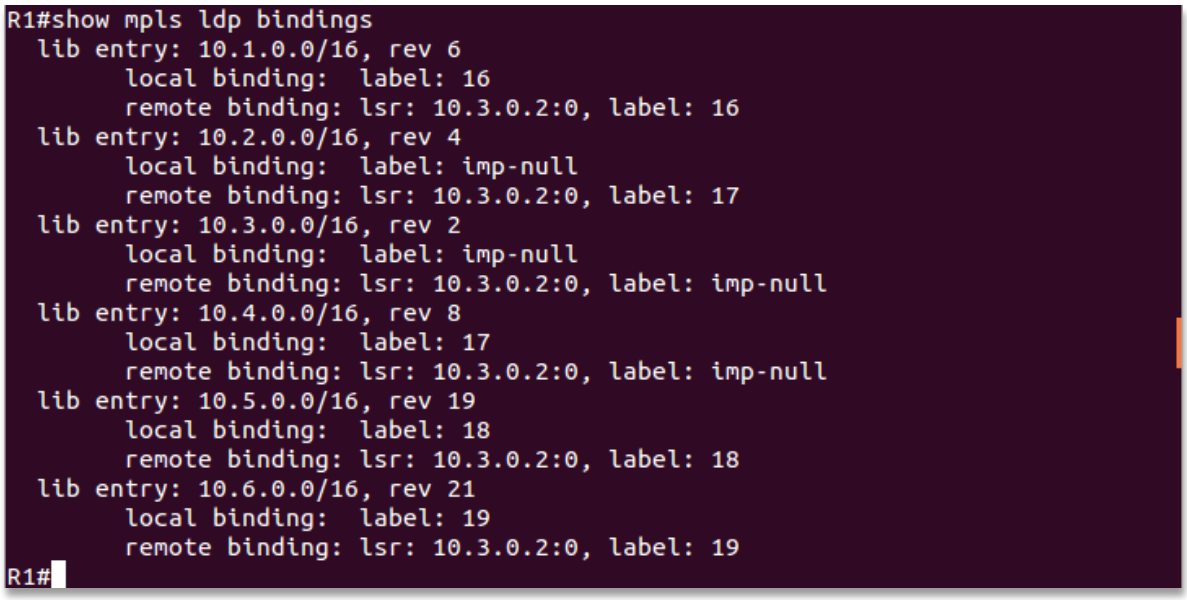


Figura 54 Tabla de Información de Etiquetas LSR1

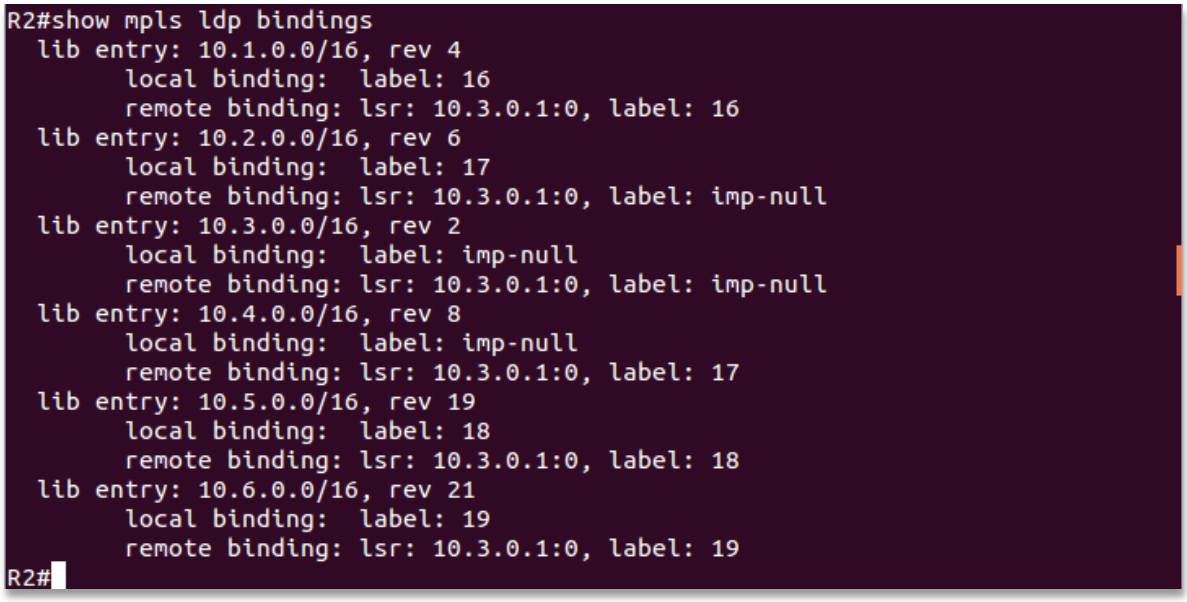


Figura 55 Tabla de Información de Etiquetas LSR2

```
R3#show mpls ldp bindings
  lib entry: 10.1.0.0/16, rev 28
        local binding:  label: 16
        remote binding: lsr: 10.5.0.1:0, label: 16
  lib entry: 10.2.0.0/16, rev 30
        local binding:  label: 19
        remote binding: lsr: 10.5.0.1:0, label: 17
  lib entry: 10.3.0.0/16, rev 26
        local binding:  label: 18
        remote binding: lsr: 10.5.0.1:0, label: 18
  lib entry: 10.4.0.0/16, rev 24
        local binding:  label: 17
        remote binding: lsr: 10.5.0.1:0, label: imp-null
  lib entry: 10.5.0.0/16, rev 2
        local binding:  label: imp-null
        remote binding: lsr: 10.5.0.1:0, label: imp-null
  lib entry: 10.6.0.0/16, rev 32
        local binding:  label: imp-null
        remote binding: lsr: 10.5.0.1:0, label: 19
R3#
```

Figura 56 Tabla de Información de Etiquetas LSR3

```
ldpd# show mpls ldp bindings
  lib entry: 10.1.0.0/16
        local binding:  label: 16
        remote binding: lsr: 10.3.0.2:0, label: 16
        remote binding: lsr: 10.5.0.2:0, label: 16
  lib entry: 10.2.0.0/16
        local binding:  label: 17
        remote binding: lsr: 10.3.0.2:0, label: 17
        remote binding: lsr: 10.5.0.2:0, label: 19
  lib entry: 10.3.0.0/16
        local binding:  label: 18
        remote binding: lsr: 10.3.0.2:0, label: imp-null
        remote binding: lsr: 10.5.0.2:0, label: 18
  lib entry: 10.4.0.0/16
        local binding:  label: imp-null
        remote binding: lsr: 10.3.0.2:0, label: imp-null
        remote binding: lsr: 10.5.0.2:0, label: 17
  lib entry: 10.5.0.0/16
        local binding:  label: imp-null
        remote binding: lsr: 10.3.0.2:0, label: 18
        remote binding: lsr: 10.5.0.2:0, label: imp-null
  lib entry: 10.6.0.0/16
        local binding:  label: 19
        remote binding: lsr: 10.3.0.2:0, label: 19
        remote binding: lsr: 10.5.0.2:0, label: imp-null
ldpd#
```

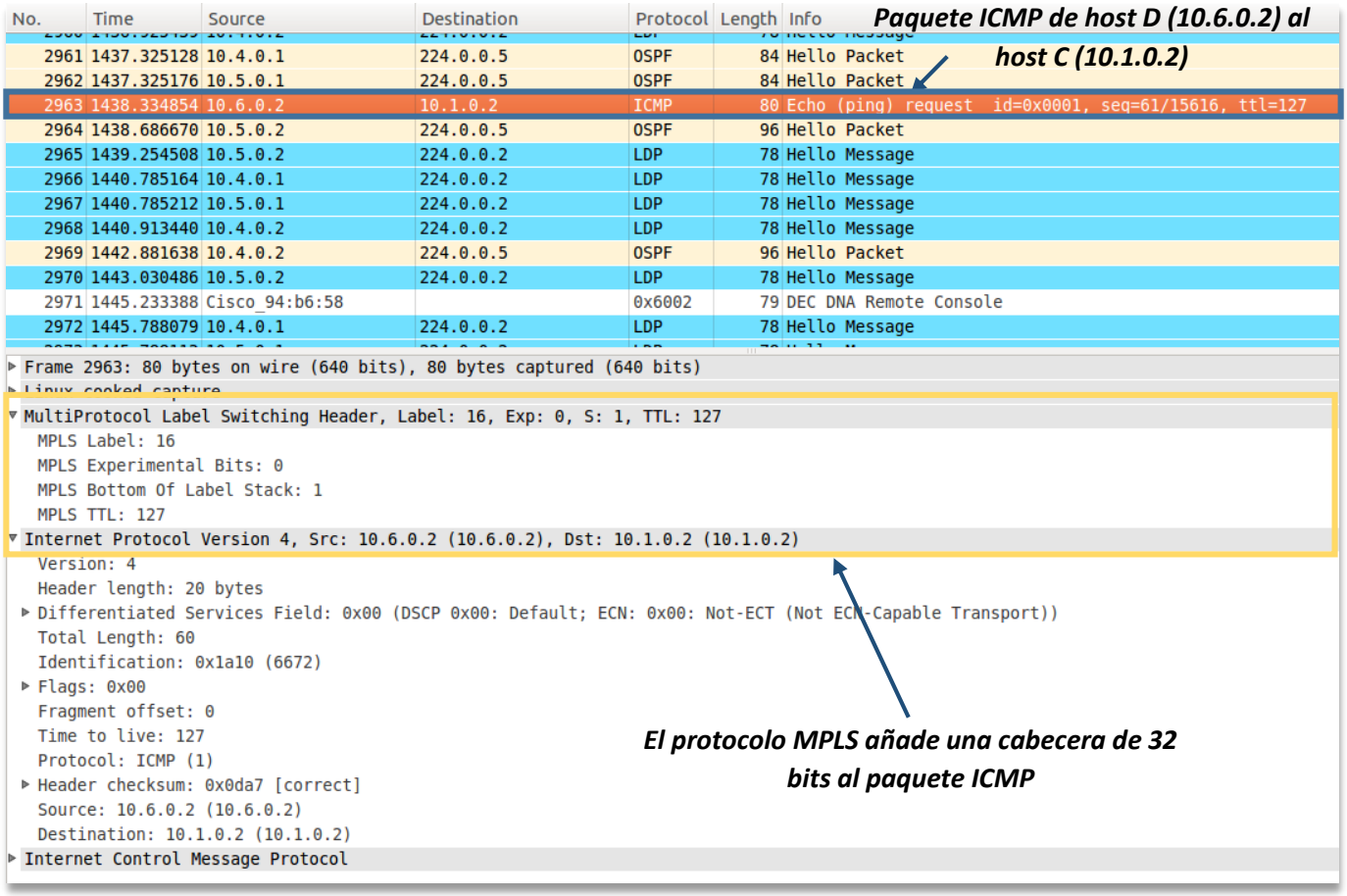
Figura 57 Tabla de Información de Etiquetas LSRLinux

La etiqueta *imp-null* indica que el paquete será renviado sin etiqueta MPLS. Esto ocurre cuando el enrutador tiene la red de destino directamente conectada, que es el que debería quitar la

etiqueta MPLS del paquete, pero en este caso la etiqueta la quita el enrutador anterior, ya que el siguiente enrutador tiene directamente conectada la red destino y en ese caso resulta más eficiente quitar de forma anticipada la etiqueta MPLS es decir funciona de manera PHP, Penultimate Hop Popping. De este modo se evita una consulta en la tabla LFIB del enrutador destino, ya que tiene esa red directamente conectada.

4.4.2.2 Plano de Datos

El plano de datos es donde se realizó la conmutación de paquetes a través la tabla de Información de Reenvío de Etiquetas, LFIB. En la *figura 58* se puede observar el formato de etiquetas MPLS de los paquetes trasmitidos entre los host C y D. Este etiquetado permite ofrecer servicios como la ingeniería, la calidad de servicio a través del campo "MPLS experimental", lo que puede servir como base para futuros trabajos de investigación.



No.	Time	Source	Destination	Protocol	Length	Info
2961	1437.325128	10.4.0.1	224.0.0.5	OSPF	84	Hello Packet
2962	1437.325176	10.5.0.1	224.0.0.5	OSPF	84	Hello Packet
2963	1438.334854	10.6.0.2	10.1.0.2	ICMP	80	Echo (ping) request id=0x0001, seq=61/15616, ttl=127
2964	1438.686670	10.5.0.2	224.0.0.5	OSPF	96	Hello Packet
2965	1439.254508	10.5.0.2	224.0.0.2	LDP	78	Hello Message
2966	1440.785164	10.4.0.1	224.0.0.2	LDP	78	Hello Message
2967	1440.785212	10.5.0.1	224.0.0.2	LDP	78	Hello Message
2968	1440.913440	10.4.0.2	224.0.0.2	LDP	78	Hello Message
2969	1442.881638	10.4.0.2	224.0.0.5	OSPF	96	Hello Packet
2970	1443.030486	10.5.0.2	224.0.0.2	LDP	78	Hello Message
2971	1445.233388	Cisco_94:b6:58		0x6002	79	DEC DNA Remote Console
2972	1445.788079	10.4.0.1	224.0.0.2	LDP	78	Hello Message

```
▸ Frame 2963: 80 bytes on wire (640 bits), 80 bytes captured (640 bits)
▸ Linux cooked capture
▾ MultiProtocol Label Switching Header, Label: 16, Exp: 0, S: 1, TTL: 127
    MPLS Label: 16
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 127
▾ Internet Protocol Version 4, Src: 10.6.0.2 (10.6.0.2), Dst: 10.1.0.2 (10.1.0.2)
    Version: 4
    Header length: 20 bytes
  ▸ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 60
    Identification: 0x1a10 (6672)
  ▸ Flags: 0x00
    Fragment offset: 0
    Time to live: 127
    Protocol: ICMP (1)
  ▸ Header checksum: 0x0da7 [correct]
    Source: 10.6.0.2 (10.6.0.2)
    Destination: 10.1.0.2 (10.1.0.2)
▸ Internet Control Message Protocol
```

Figura 58 Captura de Wireshark de paquetes MPLS

4.4.2.2.1 Tabla de Información de Reenvío de Etiquetas-LFIB (show mpls fowording-table)

Mediante LDP cada LSR distribuyó sus etiquetas a sus vecinos para que las utilicen cuando le mandan paquetes. Una vez que las etiquetas se distribuyeron, la conmutación se realiza utilizando la tabla LFIB que almacena la etiqueta asignada por el LSR vecino, la interfaz por donde se enviará el paquete y la acción a realizar con la etiqueta añadida (push, swap o pop).

El comando *show mpls fowording-table* permitió visualizar la información de las tablas LFIB de los enrutadores MPLS y verificar la conmutación por etiquetas que se realizó en cada enrutador. Ver *Figura 59,60 y 61*.

```
R1#show mpls forwarding-table
Local  Outgoing      Prefix           Bytes Label   Outgoing    Next Hop
Label  Label or VC   or Tunnel Id     Switched      interface
16     No Label      10.1.0.0/16      24826         Fa0/0       10.2.0.2
17     Pop Label     10.4.0.0/16      0             Se0/0/0     point2point
18     18            10.5.0.0/16      0             Se0/0/0     point2point
19     19            10.6.0.0/16      0             Se0/0/0     point2point
R1#
```

Figura 59 Tabla de información de reenvío de etiquetas de R1

```
R2#show mpls forwarding-table
Local  Outgoing      Prefix           Bytes Label   Outgoing    Next Hop
Label  Label or VC   or Tunnel Id     Switched      interface
16     16            10.1.0.0/16      0             Se0/0/0     point2point
17     Pop Label     10.2.0.0/16      0             Se0/0/0     point2point
18     No Label      10.5.0.0/16      0             Fa0/0       10.4.0.1
19     No Label      10.6.0.0/16      0             Fa0/0       10.4.0.1
R2#
```

Figura 60 Tabla de información de reenvío de etiquetas de R2

```
R3#show mpls forwarding-table
Local  Outgoing      Prefix           Bytes Label   Outgoing    Next Hop
Label  Label or VC   or Tunnel Id     Switched      interface
16     16            10.1.0.0/16      0             Fa0/0       10.5.0.1
17     Pop Label     10.4.0.0/16      0             Fa0/0       10.5.0.1
18     18            10.3.0.0/16      0             Fa0/0       10.5.0.1
19     17            10.2.0.0/16      0             Fa0/0       10.5.0.1
R3#
```

Figura 61 Tabla de información de reenvío de etiquetas de R3

4.4.2.2.2 Tabla de Información de Reenvío-FIB (show ip cef)

El comando *show ip cef* permitió visualizar la información de la tabla FIB de los enrutadores Cisco y con base a ella construir la tabla LFIB. La tabla FIB se utiliza cuando la conmutación no es por etiquetas si no por destinos de direcciones IP. Esta tabla se actualiza automáticamente con los protocolos de ruteo. Ver *Figura 62, 63 y 64*.

```
R1#show ip cef
Prefix              Next Hop            Interface
0.0.0.0/0           no route
0.0.0.0/8           drop
0.0.0.0/32          receive
10.1.0.0/16         10.2.0.2            FastEthernet0/0
10.2.0.0/16         attached            FastEthernet0/0
10.2.0.0/32         receive             FastEthernet0/0
10.2.0.1/32         receive             FastEthernet0/0
10.2.0.2/32         attached            FastEthernet0/0
10.2.255.255/32     receive             FastEthernet0/0
10.3.0.0/16         attached            Serial0/0/0
10.3.0.0/32         receive             Serial0/0/0
10.3.0.1/32         receive             Serial0/0/0
10.3.255.255/32     receive             Serial0/0/0
10.4.0.0/16         10.3.0.2            Serial0/0/0
10.5.0.0/16         10.3.0.2            Serial0/0/0
10.6.0.0/16         10.3.0.2            Serial0/0/0
127.0.0.0/8         drop
224.0.0.0/4         drop
224.0.0.0/24        receive
240.0.0.0/4         drop
255.255.255.255/32  receive
R1#
```

Figura 62 Tabla de Información de Reenvío-FIB R1

```
R2#show ip cef
Prefix              Next Hop            Interface
0.0.0.0/0           no route
0.0.0.0/8           drop
0.0.0.0/32          receive
10.1.0.0/16         10.3.0.1            Serial0/0/0
10.2.0.0/16         10.3.0.1            Serial0/0/0
10.3.0.0/16         attached            Serial0/0/0
10.3.0.0/32         receive             Serial0/0/0
10.3.0.2/32         receive             Serial0/0/0
10.3.255.255/32     receive             Serial0/0/0
10.4.0.0/16         attached            FastEthernet0/0
10.4.0.0/32         receive             FastEthernet0/0
10.4.0.1/32         attached            FastEthernet0/0
10.4.0.2/32         receive             FastEthernet0/0
10.4.255.255/32     receive             FastEthernet0/0
10.5.0.0/16         10.4.0.1            FastEthernet0/0
10.6.0.0/16         10.4.0.1            FastEthernet0/0
127.0.0.0/8         drop
224.0.0.0/4         drop
224.0.0.0/24        receive
240.0.0.0/4         drop
255.255.255.255/32  receive
R2#
```

Figura 63 Tabla de Información de Reenvío-FIB R2

```
R3#show ip cef
Prefix              Next Hop            Interface
0.0.0.0/0           no route
0.0.0.0/8           drop
0.0.0.0/32          receive
10.1.0.0/16         10.5.0.1            FastEthernet0/0
10.2.0.0/16         10.5.0.1            FastEthernet0/0
10.3.0.0/16         10.5.0.1            FastEthernet0/0
10.4.0.0/16         10.5.0.1            FastEthernet0/0
10.5.0.0/16         attached            FastEthernet0/0
10.5.0.0/32         receive             FastEthernet0/0
10.5.0.1/32         attached            FastEthernet0/0
10.5.0.2/32         receive             FastEthernet0/0
10.5.255.255/32     receive             FastEthernet0/0
10.6.0.0/16         attached            FastEthernet0/1
10.6.0.0/32         receive             FastEthernet0/1
10.6.0.1/32         receive             FastEthernet0/1
10.6.0.2/32         attached            FastEthernet0/1
10.6.255.255/32     receive             FastEthernet0/1
127.0.0.0/8         drop
224.0.0.0/4         drop
224.0.0.0/24        receive
240.0.0.0/4         drop
255.255.255.255/32  receive
R3#
```

Figura 64 Tabla de Información de Reenvío-FIB R3

4.4.3 Comprobación de encaminamiento de paquetes con OpenFlow

El escenario consistió en demostrar que se puede encaminar paquetes a través del protocolo OpenFlow en el cual las instrucciones de reenvío se basan en flujos que generan conexiones TCP en el puerto conocido para el tráfico OpenFlow 6633 entre el controlador y conmutador.

Como el conmutador OpenFlow es dedicado este no tiene inteligencia por sí mismo entonces envía el tráfico ICMP o paquetes de Ethernet enviados entre los host C y D como le indica el controlador. El conmutador conoce la dirección del controlador por lo que este inicia una conexión TCP con el controlador para redirigir el tráfico ICMP generado entre los host C y D. como se muestra en figura 65

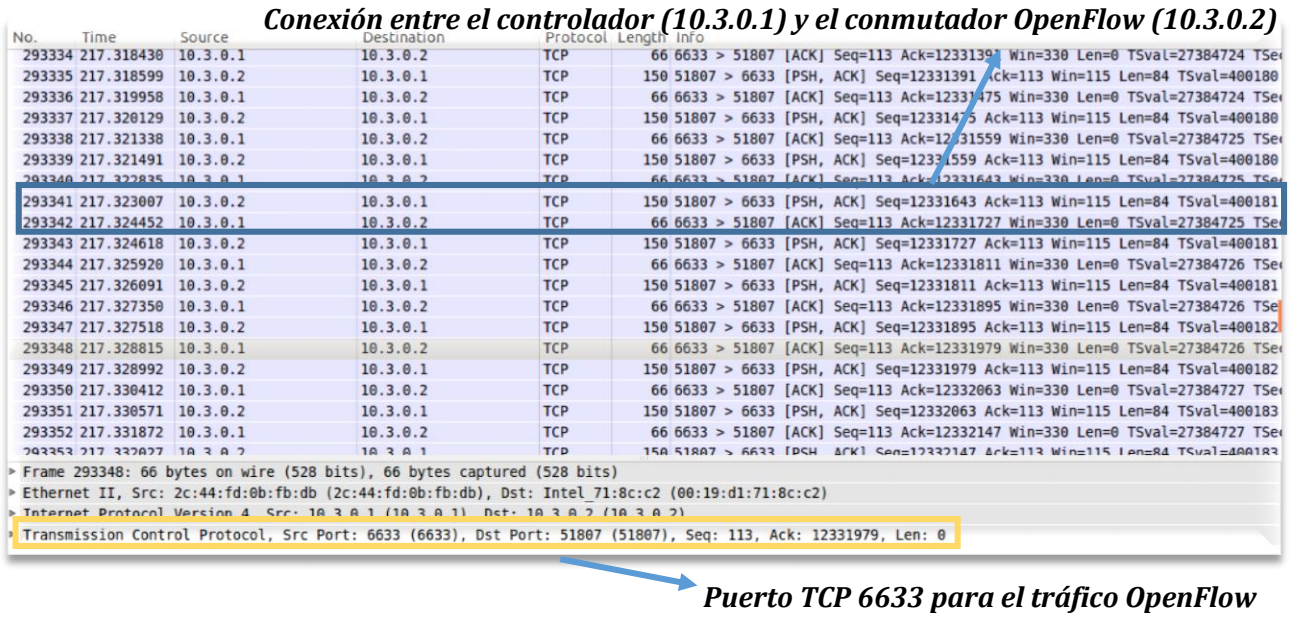


Figura 65 Captura de Wireshark: Flujo de paquetes TCP entre controlador y conmutador OpenFlow

En este escenario no se pudo apreciar los paquetes encapsulados de OpenFlow entre el conmutador y controlador para el encaminamiento de paquetes, como se obtuvo en el escenario MPLS. Debido a que OpenFlow es un protocolo en desarrollo, Wireshark no cuenta con él en su base y por la dificultad de su instalación, no se pudo realizar instalación del pluing de OpenFlow en Wireshark.

Pero los resultados fueron que se pudieron trasmitir paquetes entre host C y D con reglas simples instaladas en el control que vienen predeterminadas en el código de *L2_learning* y que desde el controlador a través del protocolo OpenFlow se puede programar reglas en la red como políticas que puedan permitir a los conmutadores establecer rutas óptimas sobre la red

para características específicas, como pueden ser: ingeniería de tráfico, enrutamiento, autentificación, control de acceso, monitoreo, diagnóstico, menor número de saltos o reducción de latencia.

Capítulo 5

Conclusiones

CAPÍTULO 5. CONCLUSIONES

En la actualidad, la mayoría de los operadores manejan la tecnología MPLS en su infraestructura de red por los servicios que ofrecen, lo que hace lo ideal la integración con redes definidas por software.

Openflow es un protocolo prometedor el cual se encuentra en constante desarrollo y es muy probable que sea una tecnología dominante en los próximos años.

El manejo de las tecnologías como MPLS y SDN me permitieron construir dispositivos con soporte MPLS en Linux y la capacidad para desarrollar escenarios de red en los cuales se experimente con nuevos protocolos como OpenFlow.

Las dificultades que se me presentaron a lo largo del desarrollo del proyecto se refieren a:

- Tiempo de despliegue en el escenario MPLS, ya que es una tecnología con un grado considerable de complejidad. El hecho de no contar con dispositivos MPLS dificulta en gran medida la implementación de maquetas basadas en este protocolo.
- La construcción de enrutadores de conmutación de etiquetas en Linux es un proceso largo.
- Para el escenario SDN se presentaron dificultades para el establecimiento de la conexión debido que a las aplicaciones utilizadas no son estables todavía
- Muchas de las características de estas tecnologías resultan complejas lo que dificulta su implementación

Una vez superadas estas situaciones en la implementación y manejo de las tecnologías MPLS y SDN, la configuración es robusta y demuestro la factibilidad de implementación que puede servir como punto de partida para desarrollar proyectos académicos que permitan generar talentos en estas áreas.

En cuanto a las líneas de investigación considero que debemos concentrarnos en cómo se configuran las reglas en el controlador para desarrollar cosas más versátiles y realizar pruebas de desempeño como el trabajo realizado con mi compañera Mayumi Park Campos, ***Despliegue y evaluación de desempeño de una red OpenFlow.***

BIBLOGRAFÍA

Black, U. (2002). *MPLS and Label Switching Networks.* Prentice Hall.

Create OpenFlow network with multiple PCs/NetFPGA. (2011). Retrieved from http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Deployment/HOWTO/LabSetup#PCs_for_OpenFlow_Switches

Dibildox, L. M. (2006, Mayo 16). *Colección de Tesis Digitales .* Retrieved from Investigación de Redes VPN con Tecnología MPLS: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/morales_d_l/capitulo_2.html

Fundation, O. N. (2012, 04 13). *Software-Defined Networking: The New Norm for Networks.* Retrieved 08 10, 2014, from https://www.opennetworking.org/

González, C. A. (2014, Febrero). Despliegue de una maqueta de red basada en OpenFlow. Cantambria , Santander, España : http://repositorio.unican.es/.

Hewlett-Packard Development Company, L.P. (2014). Retrieved 08 10, 2014, from Software-defined Networking: http://h17007.www1.hp.com/lamerica_nsc_cnt_amer/es/networking/solutions/technology/sdn/index.aspx#tab=TAB2

Hidalgo, D. A. (2014, Mayo). Diseño e implementación de una aplicación de red bajo la arquitectura SDN. Bogotá, Colombia: http://repository.javeriana.edu.co.

Ivana Cruza, N. A. (2013). ANÁLISIS Y CONTRASTE DE LA IMPLEMENTACIÓN DEL MULTIPROTOCOLO DE CONMUTACIÓN DE ETIQUETAS EN DIFERENTES TECNOLOGÍAS DE REDES. *XXXII*.

Juniper Networks, Inc. (2010). *Label Distribution Protocol*. (Juniper Networks , Inc) Retrieved from Juniper Networks: https://www.juniper.net/techpubs/software/junos-es/junos-es93/junos-es-swconfig-interfaces-and-routing/label-distribution-protocol.html

Juniper Networks, Inc. (2010). *LSP Establishment*. (Juniper Networks, Inc) Retrieved from Juniper Networks: https://www.juniper.net/techpubs/software/junos-es/junos-es93/junos-es-swconfig-interfaces-and-routing/lsp-establishment.html

Juniper Networks, Inc. (2010). *Resource Reservation Protocol*. (Juniper Netwoks, Inc) Retrieved from Juniper Netwoks: https://www.juniper.net/techpubs/software/junos-es/junos-es93/junos-es-swconfig-interfaces-and-routing/resource-reservation-protocol.html

Leu, J. (2013). *GitHub*. (© 2014 GitHub, Inc) Retrieved marzo 2014, from i-maravic/MPLS-Linux: https://github.com/i-maravic/MPLS-Linux

Maravic, I. S. (2012). MPLS implementation for the Linux kernel. (12).

Netwoks Juniper, InC. (2010). *Signaling Protocols Overview*. (Juniper Netwoks, Inc) Retrieved from Juniper Netwoks: https://www.juniper.net/techpubs/software/junos-es/junos-es93/junos-es-swconfig-interfaces-and-routing/signaling-protocols-overview.html

Park, M., & Baack, E. (2012). *Despliegue y evaluación de desempeño de una red OpenFlow.* Ciudad de México.

Pim Van Heuven, S. V. (n.d.). *RSVP-TE daemon for DiffServ over MPLS under Linux.* Retrieved from http://www.linux-kongress.org/2002/papers/lk2002-heuven.pdf

Quagga Routing Suite. (2013, 03 09). (Quagga Routing Software Suite) Retrieved 03 01, 2014, from Quagga Routing Suite: http://www.nongnu.org/quagga/index.html

Rosen, E. (2001). *Multiprotocol Label Switching Architecture.* Retrieved from http://tools.ietf.org/html/rfc3031

Rosen, E., Tappan, D., & Fedorkow, G. (2001, Febrero). *MPLS Label Stack Encoding.* Retrieved Febrero 2014, from RFC3032: http://tools.ietf.org/html/rfc3032

Rouse, M. (2012, Junio). *Definition: OpenFlow*. Retrieved from http://whatis.techtarget.com/definition/OpenFlow

Stallings, W. (2005). *Redes e Internet de Alta Velocidad: Rendimiento y Calidad de Servicio.* Prentice Hall.

Vargas, W. A. (2014). *academia.edu.* Retrieved from http://www.academia.edu/5730624/Emulaci%C3%B3n_de_una_red_definida_por_software_utilizando_MiniNet

Vieira, R. C., & Guardieiro, P. R. (2005). A proposal and evaluation of a LSP preemption policy implemented with fuzzy logic and genetic algorithms in a DiffServ/MPLS test-bed. (40).

GLOSARIO

FEC: Forwarding Equivalence Class. Clase de Equivalencia de Reenvió.

FIB: Forwarding Information Base. Base de Información de Reenvío

FTN: FEC-to-NHLFE map. Mapa de NHLFE a FEC

IETF: Internet Engineering Task Force.

IGP: Interior Gateway Protocol. Protocolo de Intercambio Interior.

ILM: Incoming Label Map. Mapa de Etiqueta de Entrada

LDP: Label Distribution Protocol. Protocolo de Distribución de Etiquetas.

LFIB: Label Forwarding Information Base. Base de Información de Reenvío de Etiquetas

LIB: Label Information Base. Base de Información de Etiquetas.

LSP: Label Switched Path. Camino de Conmutación de Etiquetas.

LSR: Label Switching Router. Enrutador de Intercambio de Etiquetas.

MPLS: Multi-Protocol Label Switching. Multiprotocolo de Conmutación de Etiquetas.

NHLFE*:* Next Hop Label Forwarding Entry. Entrada de Reenvío de Etiqueta del Siguiente Salto*.*

ONF: Open Network Fundation, Fundación de Red Abierta.

POX. Es un entorno basado en Python desarrollado con código abierto para aplicaciones de control de SDN.

QoS: Quality Of Service. Calidad de Servicio.

SDN: Software Defined Networking**.** Redes Definidas por Software**.**

ANEXOS

Instalación del Kernel Linux-MPLS

La Instalación del Kernel Linux-MPLS se basó en el tutorial [Sdn-nz] another recipe: 64-bit mpls linux with quagga-ldp and netfpga mpls support (http://ecs.victoria.ac.nz/pipermail/sdn-nz/2012-August/000003.html).

En vista de que Linux es el sistema operativo más utilizado de código abierto, la mayor necesidad de la implementación de la simulación MPLS es precisamente allí. El desarrollo de la versión para Linux de MPLS fue iniciado por James Leu en 1999. Al mismo tiempo, el Protocolo LDP fue desarrollado en el software de enrutamiento Quagga.

Desafortunadamente, LDP nunca fue terminado y tiene funcionalidades limitadas. De todos los sistemas operativos de código abierto sólo el sistema operativo NetBSD, a partir de versión 6.0, incorpora oficialmente MPLS en su código de red. Junto con esta aplicación, NetBSD también implementa el enrutamiento por el protocolo LDP para la distribución de etiquetas MPLS (Maravic, 2012).

Requerimientos:

Linux Ubuntu 12.04 LTS Server Kernel 3.5.0-49

La instalación del Kernel MPLS-Linux 3.9.0-rc3+ fue a través de la línea de comandos:

1. Descargar actualizaciones:
   apt- get update

2. Reiniciar el sistema después de instalar las actualizaciones:
   reboot

3. Instalar git:
   sudo apt-get install git

4. Clonar el Kernel MPLS-Linux a través del repositorio github:
   git clone https://github.com/i-maravic/MPLS-Linux.git
   sudo apt-get install Linux-headers-3.5.0-36-generic o ”Linux-headers-‘uname –r’ linux-headers-generic

5. Después instalar la librerías libncurses5:
   sudo apt-get install libncurses5 libncurses5-dev

6. Para habilitar el modulo correspondiente a MPLS se introduce al directorio *MPLS-Linux/* y se accede al menu del kernel:

   cd MPLS-Linux/
   sudo make menuconfig

7. Se habilita el modulo *Multiprotocol Label Switching* y se compila el kernel dentro del mismo directorio:

   make
   make modules
   make modules_install
   make install

   La duración de la compilación entre *make modules* y *make install* es de aproximadamente una hora.

8. Después de la instalación se actualiza el orden de arranque del kernel a través del comando:

   sudo ***update-grub2.***

9. Se reinicia el sistema y antes de iniciar se elige el kernel:

   MPLS-Linux 3.9.0-rc3+ x86_64 x86_64 x86_64 GNU/Linux

Instalación de iproute2

1. Para la instalación de iproute2 se necesita la librería libdb4o:
   sudo apt-get install libdb4o-cil-dev libdb-dev bison flex

2. Para obtener iproute2 se clona desde el repositorio github a través de git:
   git clone https://github.com/i-maravic/iproute2.git

3. se instala iproute, libtool, texinfo:
   sudo apt-get install iproute-dev autoconf automake libtool texinfo

4. Para la instalación de iproute2 se introduce al directorio:
   cd iproute2/
   sudo make
   sudo make install

Instalación de Quagga- LDP

Se instala el paquete gawk:

sudo apt-get install gawk

1. Para la obtener Quagga se clona a través de git:
   git clone http://git.savannah.gnu.org/r/quagga.git

2. Se introduce al directorio quagga:
   cd quagga/

3. Se introduce:
   ./bootstrap.sh

4. Se instala el parch ldpd.path:
   patch -p1 < ../quagga-ldpd.patch

5. Se configura:
   ./configure

6. Se instala:
   sudo Make
   sudo make install

Configuración del parche LDP

El parche ldpd.path no se aplica completamente en todos los archivos de configuración de quagga por lo que se corrigieron algunos errores:

1. Makefile.am.rej
   - Se modifica Makefile.am

2. Configure.ac.rej --- configure.ac
   Línea #1746 agrega a redhat/Makefile = redhat/Makefile ldpd/Makefile

3. lib/memtypes.c.rej agrega --> lib/memtypes.c
   Struct memory_list memory_list_zebra[] =

   línea 87
   { MTYPE_RIB_TABLE_INFO, “RIB table info” },

se le agrega:

```
#ifdef HAVE_MPLS
{ MTYPE_MPLS_BINDINGS,              "MPLS Route Bindings"          },
{ MTYPE_MPLS_LSP,                   "MPLS Label Switched Patch"    },
{ MTYPE_MPLS_CROSSCONNECT,          "Static MPLS crossconnect"     },

#endif /*HAVE_MPLS*/

{ -1, NULL },

};
```

4. Table.h.rej -→ table.h
   Línea #89
   Agregar :

```
 /* MPLS label bindings.  */
     Void *mpls;
```

   Y ponerle tres \\\ desde void *aggregate; hasta /* MPLS label bindings. */

5. Zebra/Makefile.am.rej → zebra/Makefile.am se agrega en **noinst_PROGRAMS = testzebra**

```
Mpls_sources=
Mpls_headers=
If MPLS_ENABLED
Mpls_source+mpls_vty.c mpls_lib.c
Mpls_headers+=mpls_lib.h
endif
```

   - **zebra_SOURCES = \ agrega: $(mpls_sources)**
   - **testzebra_SOURCES = agregar al final mpls_null.c**
   - **noinst_HEADERS = \ agrega $(mpls_headers)**

6. zebra/main.c.rej → zebra/main.c
   agregar → :
   #ifdef HAVE_MPLS
   #include "zebra/mpls_lib.h"
   #endif

7. zebra/zebra_rib.c.rej → zebra/zebra_rib.c
   línea #42 agregar:
   #ifdef HAVE_MPLS

#include 2zebra/mpls_lib.h”
#endif

8. zebra/rt_netlink.c
   comentar las líneas 1249 con // o /* ….*/
   addattr_l
   rta_addattr_l
   addattr32

9. lib/sigevent.h
   agregar #define Q_SIGC(sig) (sizeof)(sig)/(sig[0]))

Configuración de Quagga-LDP

Para que puedan funcionar los servicios de zebra y ldpd se agrega en etc/services el puerto 646-LDP para TCP y UDP, y el puerto 2610-ldpd en UDP. Zebra y ldpd son la plataforma para hacer funcionar el Kernel MPLS-Linux.

Se accede al directorio /usr/local/etc donde se copian los siguientes archivos:

```
#cp zebra.conf.sample zebra.conf

 # cp ldpd.conf.sample ldpd.conf
```

Figura 66 copia de archivos zebra y ldpd

Después de copiar los archivos se agrega el usuario quagga para poder crear el archivo .pid y se puedan invocar los servicios de zebra y ldpd. Se añade quagga al final del archivo /etc/group para que pueda pertenecer al grupo root:

```
# adduser quagga
```

Figura 67 Agregación del usuario quagga al grupo root

Se cambian los permisos a /run

```
# chmod 775 /run

#chmod 777 usr/local/etc //permisos de escritura
```

Figura 68 Cambio de permisos a los directorios /run y /usr/local/etc

Para que funcione el servicio de zebra y ldpd se agrega el archivo libzebra.so en el directorio ld.conf:

```
# echo /usr/local/lib > /etc/ld.so.conf.d/libzebra.conf
```

Figura 69 Agregación del archivo libzebra al directorio ld.conf

Reconfigurar ldconfig:

```
# ldconfig
```

Figura 70 reconfiguración del ldconfig