PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR SDN/OPENFLOW PARA UNA RED DE CAMPUS ACADÉMICA

Tesis para optar el Título de Ingeniero de las Telecomunicaciones, que presentan los bachilleres:

GABRIEL JOSÍAS CUBA ESPINOZA

JUAN MANUEL AUGUSTO BECERRA AVILA

ASESOR: César Augusto Santiváñez Guarniz, PhD

Lima, diciembre de 2015

RESUMEN

La presente tesis se encuentra dividida en 5 capítulos:

El primer capítulo contiene una introducción a las redes de campus académicas, como es el caso de la red de la PUCP, y a la problemática presentada a través de la evolución de la tecnología Ethernet en las redes de área local (LAN)desde sus inicios, hasta la aparición del paradigma de redes SDN.

El segundo capítulo contiene la definición del paradigma de redes SDN, así como también las bases para poder entender las tecnologías aplicadas y las plataformas utilizadas en el despliegue de este.

En el tercer capítulo, se desarrollan los requerimientos a considerar en el diseño del controlador y las principales limitaciones y dificultades que se pueden encontrar. Finalmente, se explica la operación del controlador, detallando los mecanismos que se implementarán.

En el cuarto capítulo se detalla los factores determinantes para la elección de la plataforma base de controlador y el diseño del mismo, en base a los requerimientos planteados en el capítulo 3.

FInalmente, en el quinto capítulo, se presentan las distintas pruebas de concepto utilizadas para poder determinar la funcionalidad de los módulos más importantes del controlador diseñado; así como también el modelo analítico, y el análisis respectivo, utilizado para poder medir la escalabilidad del mismo.

FACULTAD DE
CIENCIAS E INGENIERÍA

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO DE TELECOMUNICACIONES

Título : Diseño e implementación de un controlador SDN/OpenFlow para una red de campus universitaria.
Área : Telecomunicaciones 4307
Asesor : César Augusto Santiváñez Guarniz, Ph.D.
Alumnos : Juan Manuel Augusto Becerra Avila
Gabriel Josías Cuba Espinoza
Códigos : 20090417
20095824
Fecha : 14-09-2015

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Facultad de Ciencias e Ingeniería
15 DIC. 2015
MIGUEL MEJIA PUENTE
DECANO

Descripción y Objetivos

Las redes de campus de universidades de investigación cuentan con miles de usuarios con requerimientos de red muy diversos: desde alta seguridad (tesorería, registro de notas), hasta volúmenes muy altos de flujo de datos (herramientas de colaboración, aplicaciones científicas como computo de alta performance ó HPC). Esta diversidad de requerimientos presenta un reto para las tecnologías de red actuales. Soluciones capa 2 (p.ej., *switching*) proveen flexibilidad y escalabilidad a altos volúmenes de tráfico, pero presentan problemas frente a un numero grande de dispositivos (direcciones MAC), sufren de vulnerabilidades frente a atacantes o equipos mal configurados (tormenta de *broadcast*), y tienden a elegir rutas sub-optimas. Soluciones capa 3 (p.ej., *routing* sobre VLANs) permiten resolver algunos de estos problemas, pero a un muy alto costo de equipos. A medida que la PUCP migra a ser una universidad de investigación de primer nivel, su red interna deberá adaptarse para poder responder a estos retos.

El paradigma de redes SDN permite extraer el plano de control (inteligencia) de los switches de la red y centralizarlo en un elemento (Controlador) que tiene conocimiento global. Al ser este controlador programable, provee una gran flexibilidad para hacer un mejor uso de los recursos de la red, permitiendo asignar rutas optimas (capa 2) a flujos de alta velocidad (p.ej., HPC) y aplicar procesamiento por paquete para *broadcasts*, *network requests*. paquetes de inicio de sesión, etc.. Al aplicar a cada flujo el trato más adecuado, se aumenta la escalabilidad de la red. Además, un controlador SDN programable permite definir políticas (*policies*) dinámicas, que respondan a la flexibilidad extrema en los roles de los usuarios (investigador, docente, administrativo, miembro de comité, etc.) y dispositivos propios de la naturaleza de investigación.

El objetivo de la presente tesis es el diseño e implementación de un controlador SDN para una red de campus universitario, como la red PUCP, de forma que esta sea escalable a grandes demandas de trafico (del orden de los Tbps), confiable, y flexible. El diseño es modular, y toma como base la *plataforma de controlador Floodlight*, a la que se le añaden módulos específicos para mejorar la escalabilidad de una red Ethernet de capa 2, y considera una implementación gradual (por etapas), permitiendo la coexistencia de elementos (o islas) de red SDN/OpenFlow y redes existentes (*legacy*).

El diseño e implementación del controlador implica la recopilación de información acerca de la red de campus de la universidad, caracterización y clasificación de los distintos tipos de tráfico presentes en ella, determinación de estrategias de manejo para cada tipo de trafico, desarrollo de módulos que implementen estas estrategias, y la evaluación del rendimiento del controlador (escalabilidad, throughput, etc.) a través de pruebas realizadas en un emulador y en switches SDN/OpenFlow disponibles en el laboratorio del grupo GIRA de la PUCP.

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Especialidad de Ingeniería de las Telecomunicaciones
Ing. GUMERCINDO BARTRA GARDINI
Coordinador

FACULTAD DE
CIENCIAS E INGENIERÍA

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO DE TELECOMUNICACIONES

Título : Diseño e implementación de un controlador SDN OpenFlow para una red de campus universitaria

Índice

Introducción

1. Redes de Campus
2. SDN/OpenFlow
3. Consideraciones de diseño
4. Diseño e integración de las aplicaciones
5. Pruebas y análisis

Conclusiones

Recomendaciones

Bibliografía

Anexos

Máximo: 100 páginas

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Especialidad de Ingeniería de las Telecomunicaciones
Ing. OMERCINDO BARTRA GARDINI
Coordinador

DEDICATORIA

A nuestros padres.

A nuestro asesor.

AGRADECIMIENTOS

A Dios, por haberme amado y brindado salvación. A mis padres, por haberme apoyado sin dudar en estos largos 7 años de carrera. A mis familiares, aunque no sepan qué es lo que hago. A mis amigos, con lo que he vivido tantas cosas tanto dentro como fuera de la universidad. Finalmente, a mi compañero de tesis, Gabriel, por soportarme en este año y haber perseverado… ¡Lo logramos!

Juan Manuel

A Dios, por todo. A mis padres, por enseñarme lo más importante que he aprendido. A mis familiares y amigos, por su apoyo, a pesar de que a veces no entiendan qué es lo que hago. A cada profesor que tuve durante esta carrera. A Manuel, por el excelente trabajo en equipo.

Gabriel

Un agradecimiento especial a nuestro asesor, Cesar Santiváñez Guarniz, PhD, por la orientación y exigencia durante el desarrollo de este trabajo.

INDICE

INDICE vii
FIGURAS x
TABLAS xii
INTRODUCCIÓN xiii
1. REDES DE CAMPUS 1
1.1. La Red PUCP 1
1.1.1. Presente 1
2. SDN/OPENFLOW 19
2.3.3. Service modules 37
2.3.4. Module Applications 38
2.3.5. REST Applications 39
2.4. Implementaciones más conocidas de Controladores open source 40
3. CONSIDERACIONES DE DISEÑO 43
3.1. Requerimientos generales de la solución 43
3.1.1. Subnetting 43
3.1.2. Open Source 43
3.1.3. Amplia comunidad de soporte 43
3.1.4. Rapid prototyping 43
3.1.5. Performance 44
3.1.6. Escalabilidad 44
3.2. Limitaciones y requerimientos de escalabilidad en OpenFlow 44
3.2.1. Plano de Datos 44
3.2.2. Plano de Control en base al controlador y al control cannel 45
3.3. Segmentación de la red física 46
3.3.1. Redes lógicas con OpenFlow 46
3.3.2. Requisitos de la segmentación lógica 46
3.4. Caracterización y clasificación de tráfico 48
3.4.1. Tráfico Broadcast 48
3.4.2. Tráfico Multicast 48
3.4.3. Tráfico Unicast 48
3.5. Concepto de operaciones del controlador 49
3.5.1. Manejo del broadcast 49
3.5.2. Manejo de multicast 50
3.5.3. Enrutamiento de unicast. 50

3.5.3.1. Baseline: Clustering 50
3.5.3.2. Re-enrutamiento de Flows elefantes 52
3.6. Interoperabilidad y portabilidad 54
3.6.1. Conectividad hacia internet 54
3.6.2. Coexistencia con Redes legacy 54
3.6.3. Portabilidad a otros controladores 55
4. DISEÑO E INTEGRACIÓN DE LAS APLICACIONES 56
4.1. Elección de la plataforma base del controlador 56
4.1.1. Estudios 56
4.1.2. Criterios para elección de la plataforma base 57
4.2.1. Selección de la plataforma 58
4.3. High level design del controlador 59
4.3.1. Diagrama de bloques del controlador 59
4.3.2. Módulos de segmentación lógica 60
4.3.3. Módulos de manejo de broadcast 60
4.3.4. Módulos de manejo de tráfico unicast 60
4.4. Diseño e integración de módulos de segmentación lógica 61
4.4.1. Logical Network Manager 61
4.4.2. Forwarding 64
4.4.3. Circuit Tree 64
4.5. Análisis e Integración de módulos de manejo de Broadcast 65
4.5.1. Módulo DHCP 65
4.5.2. Módulo ARP 65
4.5.3. Dominios de interés 67
4.6. Diseño e integración de módulos de Ingeniería de tráfico 71
4.6.1. Elephant Re-Routing 71
4.6.2. Elephant detector 74
4.6.3. Módulo Custom Cost 74
5. PRUEBAS Y ANÁLISIS 75
5.1. Pruebas de concepto 75
5.1.1. Pruebas en simulador 75
5.1.2. Pruebas de funcionalidad en Hardware Real 77
5.1.3. Pruebas de esfuerzo 80
5.1.4. Pruebas de Interoperabilidad 83
5.2. Modelamiento del sistema y análisis de escalabilidad 84
5.2.1. Consumo de recursos asociado a un destino d 86
5.2.2. Consumo de recursos para todos los destinos 90

5.3. Consideraciones adicionales 97
5.3.1. Seguridad 97
5.3.2. Alta disponibilidad 98
TRABAJO FUTURO 102
REFERENCIAS 103

FIGURAS

Figura 1-1. Mapa del campus PUCP....1
Figura 1-2. Topología típica de la red de la PUCP....2
Figura 1-3. Distribución de los enlaces de fibra óptica en el campus....3
Figura 1-4. Distribución de Access Points inalámbricos en el campus....3
Figura 1-5. Velocidades de los enlaces externos al campus....4
Figura 1-6. Topología de la RedCLARA....5
Figura 1-7. Topología de la Red GENI....5
Figura 1-8. Diagrama de Legión-PUCP....7
Figura 1-9. Primera generación de redes LAN Ethernet....8
Figura 1-10. Segunda generación de redes LAN Ethernet....9
Figura 1-11. Tercera generación de redes LAN Ethernet....9
Figura 1-12. Estructura básica de un Switch....10
Figura 1-13. Elección del Spanning Tree....12
Figura 1-14. Primera generación de VLANs: Single Switch....13
Figura 1-15. Segunda generación de VLANs: Multi-Switch....14
Figura 1-16. Enrutamiento inter-VLAN....14
Figura 1-17. Estructura básica de un Router....15
Figura 1-18. Topología tipo árbol con Switch L3 centralizado y STP....16
Figura 1-19. Funcionalidades de red implementadas en hardware....17
Figura 2-1. Arquitectura en capas de SDN....21
Figura 2-2. Esquema tradicional vs Arquitectura SDN....22
Figura 2-3. Componentes de SDN/OpenFlow....25
Figura 2-4. Flow Entry....29
Figura 2-5. Ejemplo de Flow Table....30
Figura 2-6. Pipeline de un Switch OpenFlow....32
Figura 2-7. In-band vs Out-of-band Control....32
Figura 2-8. Evolución del protocolo OpenFlow hasta la versión 1.4....34
Figura 2-9. Arquitectura de NOX....35
Figura 2-10. Arquitectura de Floodlight....35
Figura 2-11. Arquitectura de OpenDaylight....36
Figura 2-12. Arquitectura de un controlador idealizado....37
Figura 3-1. Envío del primer paquete....51
Figura 3-2. Establecimiento de ruta....51
Figura 3-3. Elephant Monitor....52

Figura 3-4. Elephant Detection 53
Figura 3-5. Elephant Rerouting 53
Figura 4-1. Diagrama de módulos del controlador implementado 59
Figura 4-2. Estructura de una regla de Red Lógica 62
Figura 4-3. ClusterDatabase 63
Figura 4-4. Diagrama de flujos de ARPProxy 67
Figura 4-5. Dominios de interés. Topología base 68
Figura 4-6. Dominios de interés. Paso 1 69
Figura 4-7. Dominios de interés. Paso 2 69
Figura 4-8. Dominios de interés. Paso 2 70
Figura 4-9. Dominios de interés. Paso 3 70
Figura 4-10. Flow Entry para Elephant Rerouting 73
Figura 5-1. Topología de prueba en Mininet 75
Figura 5-2. Reglas activas en el Switch de acceso utilizando el Clustering 77
Figura 5-3. Topología física de prueba 78
Figura 5-4. Topología descubierta por el controlador 78
Figura 5-5. Pruebas de Ping desde el Cliente-1 79
Figura 5-6. Pruebas de Ping desde el Cliente-2 79
Figura 5-7. Pruebas de Ping desde el Cliente-3 79
Figura 5-8. Limitantes físicas (TCAM) de los Switches Pica8 80
Figura 5-9. Medición del ancho de banda del enlace con iperf 81
Figura 5-10. Dashboard del controlador 82
Figura 5-11. Flujos instalados en el Switch del Cliente-1 82
Figura 5-12. Acceso a Internet desde la red OpenFlow 83
Figura 5-13. Topology de la red PUCP conocida a través de un Switch OpenFlow 84
Figura 5-14. Flujo típico, representando sus estados 86
Figura 5-15. Cadena de Márkov de N estados 87

TABLAS

Tabla 2-1. Capacidad de las tablas de Switches 26
Tabla 2-2. Controladores y características principals 42
Tabla 4-1. Comparación de controladores según casos de uso 57
Tabla 5-1. Resultados de las pruebas con CBENCH 81
Tabla 5-2. Parámetros usados en el análisis numérico de escalabilidad del sistema. 94
Tabla 5-3. Resultados de las simulaciones 96

INTRODUCCIÓN

Las redes de campus académicas, como es el caso de la Red PUCP, cuentan con una gran cantidad de usuarios, los cuales tienen diferentes roles y requerimientos de red, crecientes y tan diversos, que van desde el uso de plataformas educativas y correo electrónico, hasta el uso de las redes sociales como motivo de recreación; lo cual, junto con un ineficiente manejo de los recursos de red, genera una degradación en la calidad de los servicios percibida por los usuarios, y una congestión de tráfico dentro de la red misma, que implica un gran reto para los administradores, ya que la hace inestable, poco escalable y difícil de mantener, desde el punto de vista económico y de consumo de energía.

El paradigma de redes SDN permite quitar el Plano de Control (inteligencia) de los equipos de red y centralizarlo en un elemento llamado Controlador, el cual tiene conocimiento de toda la red, por lo que hace posible un mejor uso de los recursos de esta, haciendo que sea flexible y escalable en el Plano de Datos, y disminuyendo así los gastos de OPEX; además, debido a que está basado en estándares abiertos, permite desligarnos de comprar soluciones propietarias a un solo vendedor, y más bien trabajar dentro de un ambiente multi-vendor, haciendo que los gastos de CAPEX se vean disminuidos también.

Cabe mencionar que SDN representa una oportunidad de impulsar el desarrollo del sector TI en el Perú. Mientras que con el paradigma tradicional el país se ha convertido en un simple consumidor de tecnología, adaptándose a lo ofrecido por los proveedores de tecnologías y soluciones, con SDN es factible la creación de una industria local en telecomunicaciones, que desarrolle tecnología propia y de calidad.

Este trabajo pretende ser el inicio de una serie de investigaciones académicas en este ámbito en el país, labor que ha sido apoyada completamente en la infraestructura SDN/OpenFlow disponible en el laboratorio del Grupo de Investigación en Redes Avanzadas de la Pontificia Universidad Católica del Perú (GIRA-PUCP).

El objetivo de la presente tesis es el diseño e implementación de un controlador SDN para una red de campus académico, como la Red PUCP, de forma que esta sea escalable a grandes demandas de tráfico (del orden de los Tbps) y una gran cantidad de usuarios; además de ser confiable y flexible.

El diseño es modular, y toma como base la plataforma de controlador Floodlight, a la que se le añaden, y en algunos casos se modifican, módulos específicos para mejorar la escalabilidad de una red Ethernet de Capa 2; además, considera una implementación gradual (por etapas), permitiendo la coexistencia de elementos (o islas) de red SDN/OpenFlow y Legacy.

1. REDES DE CAMPUS

El presente capítulo contiene una introducción a las redes de campus académicas, como es el caso de la red de la PUCP, y a la problemática presentada a través de la evolución de la tecnología Ethernet en las redes de área local (LAN) desde sus inicios, hasta la aparición del paradigma de redes SDN.

1.1. La Red PUCP

1.1.1. Presente

En la actualidad, la Pontificia Universidad Católica del Perú (PUCP) tiene una población de aproximadamente 26,866 alumnos, 2,825 profesores y 2,925 administrativos y obreros, a quienes recibe día a día en los 15 departamentos académicos, 13 facultades, 8 bibliotecas, 47 laboratorios, 17 centros e institutos de investigación y edificios administrativos, pertenecientes a su campus principal, los cuales se conectan a la red de campus de la universidad para realizar sus actividades diarias. [10]

Figura 1-1. Mapa del campus PUCP

PUCP (2015)

Esta red de campus tiene un diseño jerárquico, el cual consta de 1 Switch L3 Cisco Catalyst de la serie 6500, que cumple las funciones de Core. Es modular, por lo que cuenta con tarjetas especiales que permiten la habilitación de funcionalidades extra

como por ejemplo Firewall, NAT, entre otras.

Cuenta también con un Data Center ubicado en las oficinas de la Dirección de Informática de la universidad (DIRINFO), en el cual se encuentran los servidores que brindan los servicios de Intranet, correo, video, como por ejemplo la plataforma Educast; servidores de la Dirección de Informática Académica (DIA), como la plataforma de aprendizaje PAIDEIA; entre otros. Adicionalmente, se encuentra también ubicado el Wireless LAN Controller (WLC), que es el elemento encargado de la gestión de los Access Point de la red inalámbrica de la universidad. Aunque la mayoría de los enlaces que van hacia el Data Center son de 1 Gbps, se tiene algunos con velocidad de 10 Gbps.

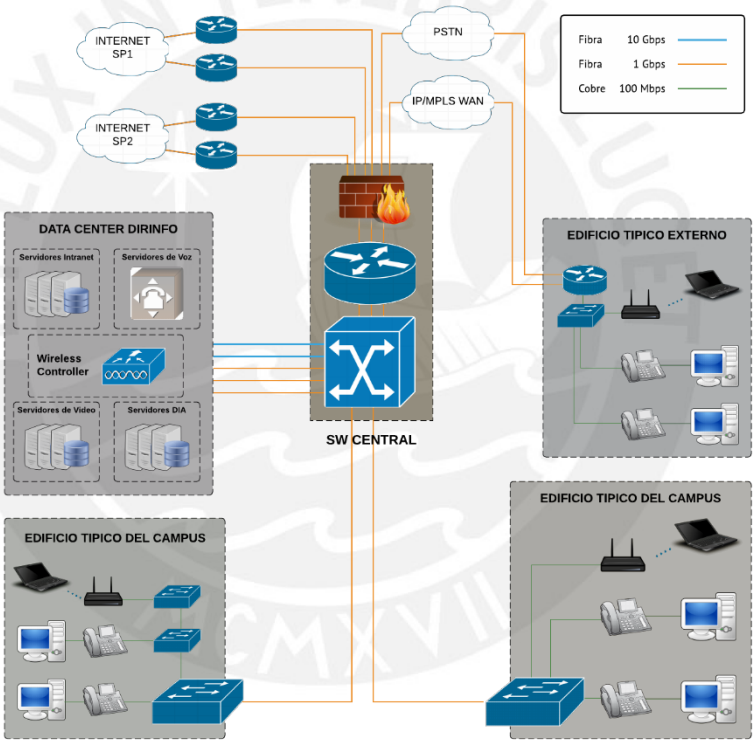

Figura 1-2. Topología típica de la red de la PUCP
ELABORACIÓN PROPIA (2015)

Adicionalmente, hacia el Switch central van conectados, a través de 64 enlaces de fibra óptica (con velocidad de 1 Gbps) tendidos por todo el campus, Switches de Distribución Cisco Catalyst de 48 puertos, ubicados en los diferentes pabellones, y que facilitan la interconexión con los Switches de Acceso Cisco Catalyst de la serie 2900, de 24 y 48 puertos, que se encuentran ubicados en los cuartos de equipos de cada piso, siempre a través de enlaces únicos de cobre con capacidad de 100 Mbps formando una topología de tipo árbol.

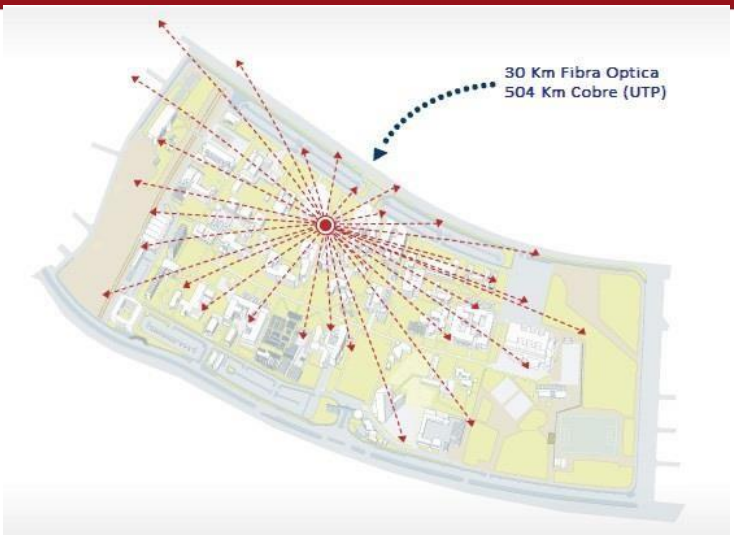

Figura 1-3. Distribución de los enlaces de fibra óptica en el campus
LA PUCP EN CIFRAS (2015) [10]

Los servicios que ofrece la red se encuentran separados por VLANs específicas e independientes para cada uno de ellos (VoIP, datos, management, entre otros), las cuales cuentan con un único *default Gateway*, configurado en el Switch de Core, el único elemento dentro de toda la red capaz de realizar enrutamiento a nivel de Capa 3.

Adicionalmente a los Switches de Acceso, se tiene 802 Access Points Cisco Aironet de la serie 3600, que brindan servicios inalámbricos indoor y outdoor, distribuidos a través de todo el campus, y cuyo acceso es a través de una clave WPA2 única y universal.

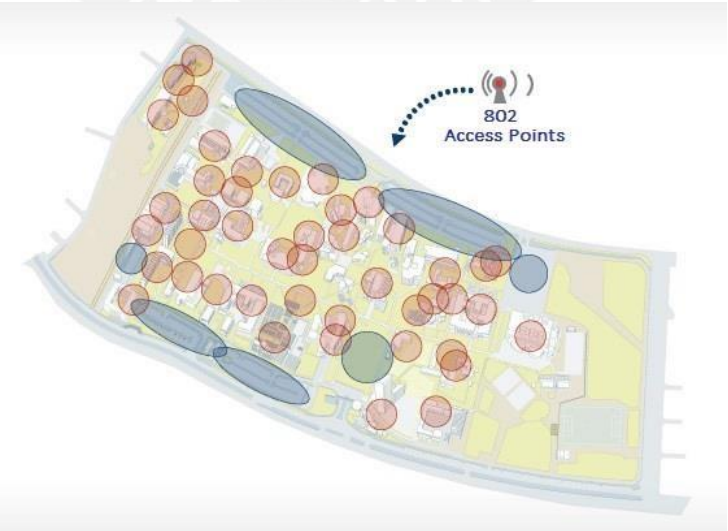

Figura 1-4. Distribución de Access Points inalámbricos en el campus
LA PUCP EN CIFRAS (2015) [10]

Esta red de Access Points se encuentra gestionada a través de un WLC de la marca Cisco, el cual genera túneles dedicados para el tráfico en el Plano de Datos hacia todos ellos; es decir, todo el tráfico inalámbrico proveniente de los usuarios pasa a través del WLC, haciendo de este un posible cuello de botella y/o punto de falla.

La red cuenta, además, con interconexiones hacia los edificios externos al campus principal, mediante enlaces dedicados y servicios de MPLS VPN contratados a través de un proveedor de Internet.

Finalmente, la red de campus cuenta con una salida a Internet con una capacidad de 900 Mbps, a través de dos proveedores, por motivos de seguridad y respaldo, cuyo tráfico es balanceado de forma manual por los administradores de la red.

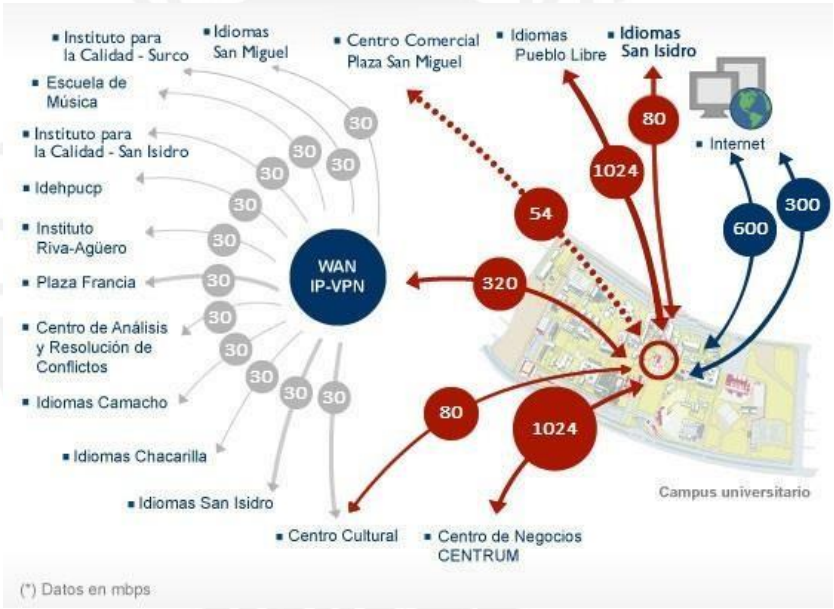

Figura 1-5. Velocidades de los enlaces externos al campus
LA PUCP EN CIFRAS (2015) [10]

1.1.2. Futuro

Debido a que la PUCP tiene como uno de sus principales objetivos el ser una universidad líder en investigación, es partícipe de redes regionales avanzadas, como:

- RAAP (Red Académica Peruana), y
- RedCLARA (Cooperación Latino Americana de Redes Avanzadas)

Figura 1-6. Topología de la RedCLARA

REDCLARA (2014)

Además, en un futuro, se planea que sea partícipe de la red de investigación GENI (Global Environment for Network Innovations) de los EEUU; a través de la federación de un ExoGENI rack, el cual se encuentra actualmente en proceso de implementación, para compartir recursos con los otros racks ubicados alrededor del mundo.

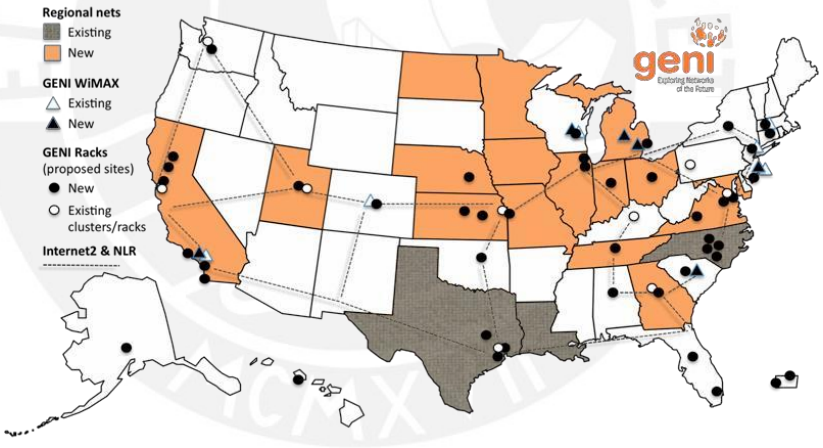

Figura 1-7. Topología de la Red GENI

GENI (2015)

DIRINFO tiene planeado hacer, en los próximos años, un cambio en la topología, teniendo un Core distribuido en 4 Switches ubicados en puntos estratégicos del campus; conectados de tal manera que brinden una mayor tolerancia ante fallas, y una mayor calidad de servicio, así como también la ampliación de los enlaces de salida hacia Internet y la mejor gestión de estos, a través del balanceo automático de carga, para darles un uso óptimo.

Se planea también la compra y el uso de un appliance con funciones de NAC

(Network Access Control), para el acceso a la red inalámbrica a través de cuentas personales y perfiles de usuario, mediante un servidor de autenticación que brinde diferentes niveles de calidad de servicio y de acceso a los recursos, dependiendo del rol que asuma cada usuario dentro de la universidad (alumno, docente, investigador, etc).

Además, se tiene pensado implementar la solución FlexConnect de Cisco, que permite que el tráfico de plano de datos del usuario sea conmutado directamente en la red de acceso, eliminando la necesidad de túneles dedicados hacia el WLC, con lo cual se haría solo sincronización en el plano de control; esto generaría una gran cantidad de direcciones MAC a ser conocidas en la red, las cuales al pertenecer a dispositivos inalámbricos que posiblemente se encuentren en movimiento, generarían hand-overs que deben ser atendidos.

Otros cambios adicionales a considerar dentro de la Red PUCP son los siguientes:

- El crecimiento del tráfico debido a las nuevas aplicaciones y contenidos, como streaming de video, video conferencias, multimedia como Punto.edu, entre otros.
- El crecimiento del tráfico interno entre servidores o centros de investigación, ya sea por el uso de HPC, Griding computing, entre otros.
- Ampliación de los enlaces entre los Switches de Core y Switches de Distribución en los pabellones a velocidades de 10 Gbps.
- Ampliación de 1 a 4 Data Centers, distribuidos por el campus.
- Expansión de la red inalámbrica de WiFi PUCP, para brindar mayor cobertura y mayores velocidades de cara al usuario a través del uso de los protocolos 801.11n/ac (en las bandas de 2.4 y 5GHz).

Finalmente, un punto importante a considerar sería el uso de la virtualización, tendencia que se ha ido extendiendo en los últimos años, para satisfacer necesidades como:

- Aprovisionamiento rápido de redes virtuales a grupos dentro de la PUCP, ya sea para eventos, experimentos, etc.

- Habilitación de enlaces on-demand desde cualquier punto del campus hacia el gateway de salida hacia las redes académicas (RAAP, RedCLARA, Geni).

- Interacción con máquinas y Switches virtuales ubicados dentro de los Data Centers de la universidad que provean servicios.

Por estos motivos y por el crecimiento de la cantidad de alumnos, se planea el aprovisionamiento de una red cada vez más robusta, que no tenga problemas en soportar los nuevos requerimientos de grandes ancho de banda y las bajas latencias que exigen las aplicaciones de vanguardia como transmisión de video y videoconferencias de alta calidad, junto con las ya mencionadas anteriormente.

Se espera que estas aplicaciones generen un crecimiento exponencial del tráfico interno (entre pabellones del campus), debido a los flujos de gran velocidad que se presenten entre las distintas áreas de investigación ubicadas en el campus (INRAS-PUCP, GIRA- PUCP, Laboratorios de Física), el uso de la plataforma de grid computing Legión-PUCP, y la integración a las redes avanzadas ya mencionadas anteriormente.

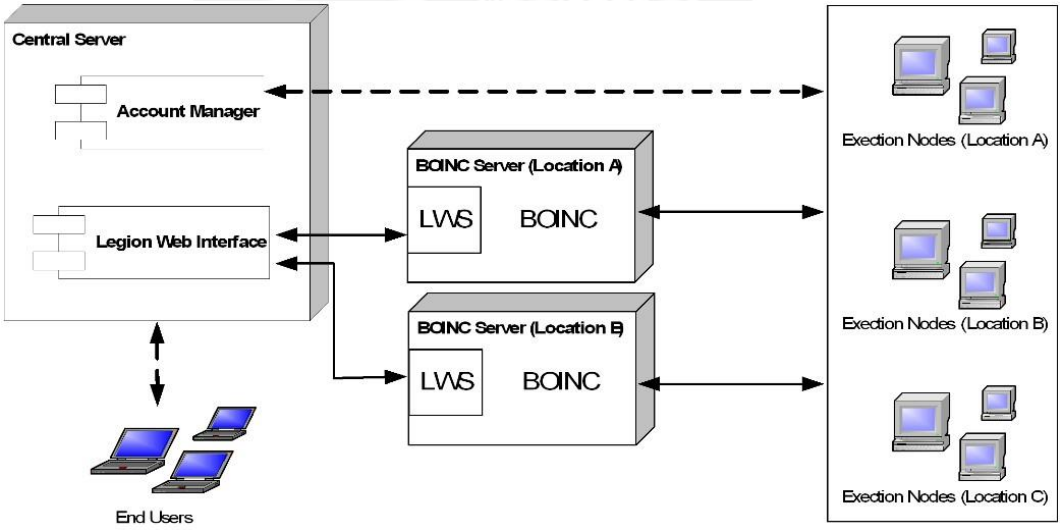

Figura 1-8. Diagrama de Legión-PUCP
BLOG GRID LEGIÓN (2011)

1.2. Evolución y problemática del esquema Layer 2 / Layer 3

En la década de 1970, predominaban los grandes cuartos de cómputo y las mainframes, y es en esta época cuando hacen su aparición las minicomputadoras,

equipos que ocupaban menos espacio y tenían un precio mucho menor; y que, aunque no contaban con las capacidades de cómputo de las primeras, eran capaces de realizar tareas sencillas con rapidez, por lo que permitían dividir grandes tareas en tareas parciales y ejecutarlas en paralelo entre ellas para igualar a las mainframes.

Debido a este nuevo ambiente de distribución de los recursos, surgió la necesidad de interconectar las minicomputadoras y es entonces que comenzaron a aparecer una gran cantidad de soluciones propietarias (a nivel de hardware, de software y de protocolos), las cuales no permitían la interoperabilidad entre redes de distintos fabricantes; es así que en la década de 1980, luego de grandes esfuerzos por estandarizar las tecnologías de networking a nivel de arquitecturas, protocolos y modelos, se da inicio el nacimiento de la tecnologías LAN como Token Ring, FDDI y Ethernet, siendo esta última la que sobrevive hasta la actualidad, y sobre la cual se tratará.

La primera generación de redes de computadoras de área local (LAN) Ethernet tenían como característica principal el tener un solo segmento de red, en el cual los hosts compartían el mismo medio físico y ancho de banda. Debido a que el uso de cables muy largos representa altos niveles de atenuación de extremo a extremo, se hizo necesario utilizar elementos llamados Repetidores, los cuales trabajan regenerando la señal, permitiendo así extender el alcance de las redes.

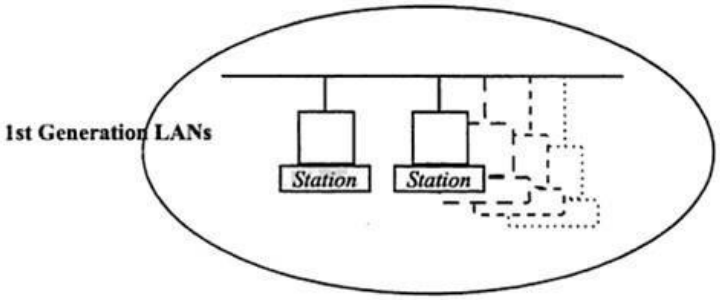

Figura 1-9. Primera generación de redes LAN Ethernet
LOCAL AREA NETWORK HANDBOOK (1999) [24]

En la segunda generación, aparecen los Hubs o Concentradores, nuevos elementos que le daban a la red la característica de tener un medio físico dedicado para cada host, aunque el ancho de banda seguía siendo compartido.

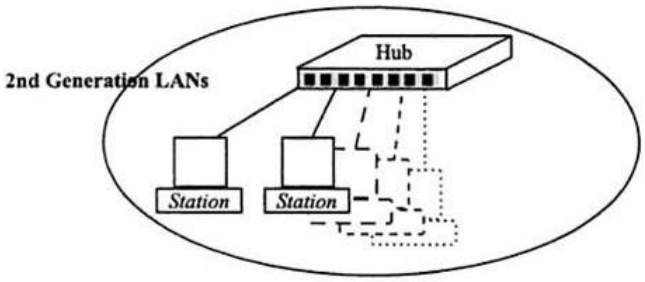

Figura 1-10. Segunda generación de redes LAN Ethernet
LOCAL AREA NETWORK HANDBOOK (1999) [24]

En estas dos primeras generaciones, los elementos (Repetidores y Hubs) trabajan en la Capa 1 del modelo OSI (Capa Física), formando de esta manera un solo dominio de colisión y un solo dominio de broadcast.

Se llama colisión al choque o interferencia entre dos tramas enviadas por diferentes hosts dentro de un solo segmento físico de red, las cuales son descartadas producto de este evento. Es así que, un dominio de colisión está definido como un grupo de hosts que comparten un solo segmento físico, donde es posible que ocurra una colisión. Es por este motivo que no se tiene un aprovechamiento adecuado del ancho de banda del segmento, ya que solo uno de los hosts puede transmitir a la vez, mientras que los otros 0'deben permanecer escuchando para disminuir la probabilidad de que exista una colisión.

1.2.1. Layer 2 networks

La tercera generación de redes permitió, con la aparición de los Bridges y los Switches, que cada uno de los hosts de la red posea un medio físico y ancho de banda independientes, dividiendo la red en varios dominios de colisión (uno por cada puerto del Switch), pudiendo utilizar totalmente el ancho de banda de cada enlace.

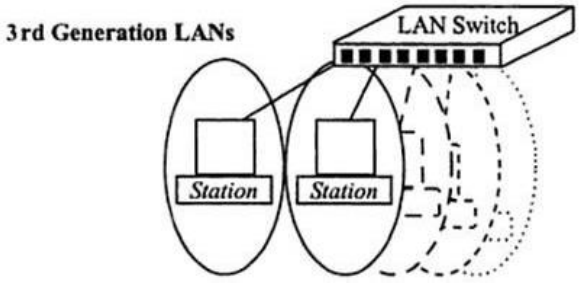

Figura 1-11. Tercera generación de redes LAN Ethernet
LOCAL AREA NETWORK HANDBOOK (1999) [24]

Los Switches trabajan en la Capa 2 del modelo OSI (Capa de Enlace de Datos) y hacen un análisis de los campos de direcciones MAC de origen y destino de las tramas, sin realizar una inspección en capas superiores, lo que los lleva a alcanzar mayores velocidades en el reenvío de estas (line-rate forwarding).

Este análisis en line-rate se realiza gracias al uso de circuitos especializados llamados ASIC (Application-Specific Integrated Circuit), los cuales se activan al ingresar tráfico por los puertos; además del uso de matrices de conmutación (Switch Fabrics) para el reenvío de las tramas entre los puertos.

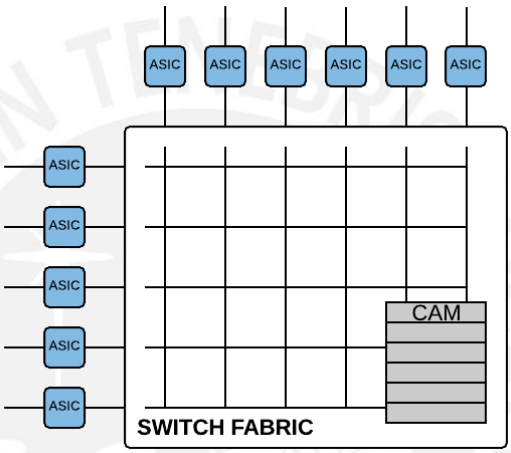

Figura 1-12. Estructura básica de un Switch
ELABORACIÓN PROPIA (2015)

Debido a que el Switch realiza el análisis de las tramas a una velocidad muy grande, las memorias RAM convenciones con que contaban inicialmente, las cuales realizan búsquedas secuenciales (tomando un ciclo de reloj por cada elemento guardado en memoria), tuvieron que ser reemplazadas por un nuevo tipo de memorias, llamadas CAM (Content-Addressable Memory), que realizan la búsqueda de datos a una mayor velocidad (en un solo ciclo de reloj), haciendo match a partir de coincidencias exactas (estados '1' y '0').

En estas CAM se almacenan duplas conformadas por las direcciones MAC aprendidas, y el puerto por el que se aprendieron, las cuales son consultadas para realizar el mapeo de las direcciones MAC fuente y destino, y la retransmisión de las tramas.

1.2.2. El problema del broadcast

Se llama dominio de broadcast al grupo de hosts lógicamente conectados (en una subred) que pueden recibir mensajes de broadcast. Estos mensajes de broadcast son generados por un host y al llegar al Switch son enviados por todos los demás puertos del mismo.

Algunos protocolos utilizan mensajes de broadcast para iniciar la negociación y/o comunicación, como por ejemplo ARP (ARP Request) y DHCP (DHCP Request), los cuales representan la gran mayoría de los mensajes de broadcast que puedan circular por una red.

El principal problema que se presenta en la transmisión de paquetes de broadcast es el de las llamadas tormentas de broadcast, las cuales merman el rendimiento de la red, llegando al punto de deshabilitarla completamente debido a la saturación de la misma, estas pueden ser producidas de dos formas: (i) por uno o más hosts que envíen mensajes de broadcast a la red de manera continua, debido a algún fallo de configuración, o (ii) por uno o más paquetes de broadcast que, al llegar a búcles en la topología, son retransmitidos de manera infinita.

1.2.3. Supervivencia en el data plane y el problema del Spanning Tree

Con la segmentación física lograda a través de los Switches, partiendo los dominios de colisión, se pudo innovar con topologías que ofrezcan un cierto grado de redundancia, para poder mitigar los efectos de la caída de uno o más enlaces dentro de la red.

Sin embargo, estos nuevos enlaces redundantes generan búcles físicos dentro de la topología, lo que unido al problema de las tormentas de broadcast, podría llegar a saturar la red hasta el punto de deshabilitarla eventualmente.

Debido a esto es que nace STP (Spanning Tree Protocol), método que permite tener topologías con redundancia física, pero que a nivel lógico no contengan bucles; ya que corre un algoritmo que elige qué puertos son los que van a permanecer activos y qué puertos van a ser bloqueados formando así un árbol topológico para cada instancia que se maneje.

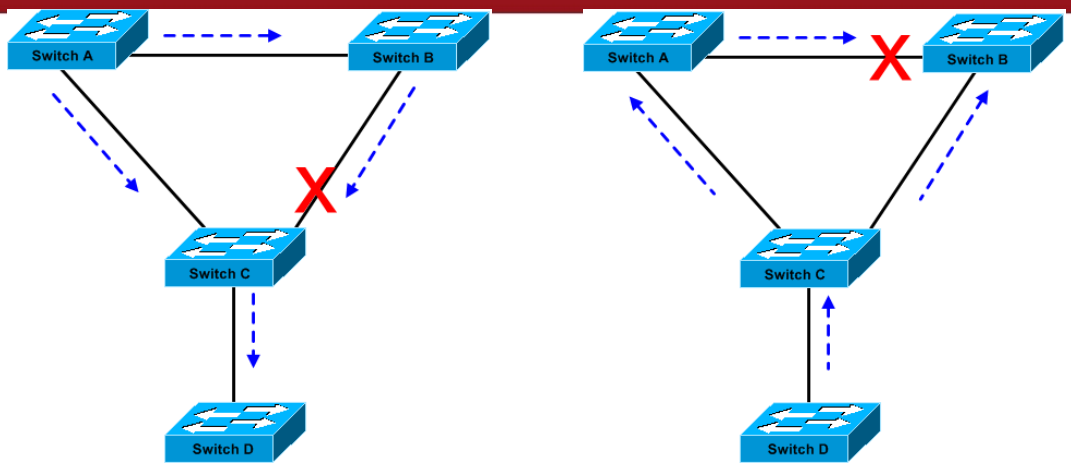

Figura 1-13. Elección del Spanning Tree
CISCO (2005) [26]

Sin embargo, cuando se tiene redes con grandes demandas de tráfico, este árbol representa un cuello de botella, ya que subutiliza el ancho de banda de la red; asimismo, cada vez que existe un cambio en alguno de los enlaces activos, se debe hacer un recalculo de las rutas y puertos activos, en el cual el árbol topológico vuelve a armarse. A este recalculo de los cambios en la topología se le llama convergencia, y el tiempo que toma este proceso es un factor crítico en el diseño de la red.

1.2.4. El problema de escalabilidad

Cuando se tiene topologías grandes, como es el caso de una red de campus, los problemas mencionados anteriormente se intensifican debido a la cantidad de elementos de red y de hosts con los que cuentan.

Un host que se comporta de manera inadecuada y envía mensajes de broadcast a la red puede saturarla, debido a que este generaría una tormenta de broadcast que termine bloqueando enlaces, equipos e incluso toda la red.

De la misma manera, cualquier cambio en la topología física, ya sea por la caída o recuperación de algún enlace, o por la adición de uno nuevo, generaría un nuevo cálculo de las ramas involucradas del árbol topológico de la red con un tiempo de convergencia grande, lo cual no es deseable.

Adicionalmente, debido a que las direcciones MAC de cada uno de los elementos de la red deben ser conocidas por los Switches para realizar en reenvío de tramas entre ellos, y a que estos cuentan con una cantidad finita de memoria, se considera este

valor limitante de los equipos como un factor de diseño importante.

1.2.5. El problema de seguridad

Al tener un solo dominio de broadcast, todos los hosts tienen la capacidad de poder conocer a los demás, ya sea a través de protocolos que utilicen Service Advertisement para hacer conocer sus servicios (como por ejemplo SSDP), o de mensajes no solicitados anunciando cambios en el estado de los hosts (como por ejemplo gratuitous ARP), lo cual podría generar un problema, ya que se puede tener expuestos recursos que sean críticos, haciéndolos vulnerables ante cualquier ataque, surge la necesidad de segmentar la red en distintos dominios de broadcast, dependiendo de la criticidad de los hosts, de las tareas que estos realicen o de los recursos a los que estos deban acceder, llevándonos a la aparición las VLAN (Virtual Local Area Network).

1.2.6. Las VLAN y la necesidad del Layer 3

Inicialmente, las VLAN aparecieron para poder segmentar los puertos de un solo Switch, dividiendo el dominio de broadcast en dominios más pequeños, para poder obtener segmentos de red cuyos recursos se encuentren separados (que pertenezcan a diferentes subnets).

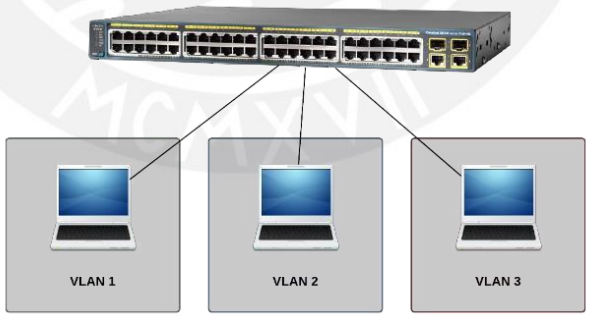

Figura 1-14. Primera generación de VLANs: Single Switch
ELABORACION PROPIA (2015)

Posteriormente, se extendió el concepto de las VLAN para poder tener comunicación entre hosts conectados a múltiples Switches, logrando que hosts alejados geográficamente puedan encontrarse dentro de la misma subred.

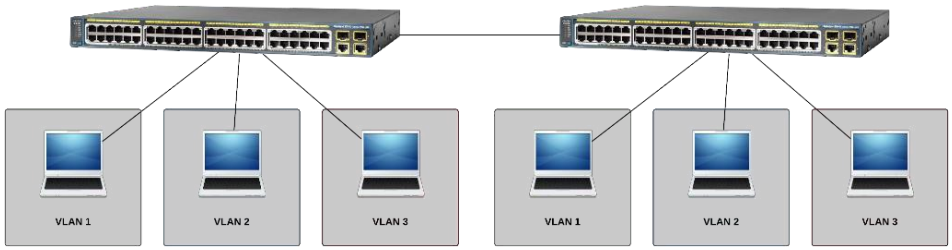

Figura 1-15. Segunda generación de VLANs: Multi-Switch
ELABORACIÓN PROPIA (2015)

Este esquema cumplía con el objetivo inicial de poder aislar segmentos de la red; sin embargo, cuando se requiere comunicación o compartición de recursos entre hosts que se encuentran en diferentes VLAN, se debe agregar un nuevo elemento, llamado Router.

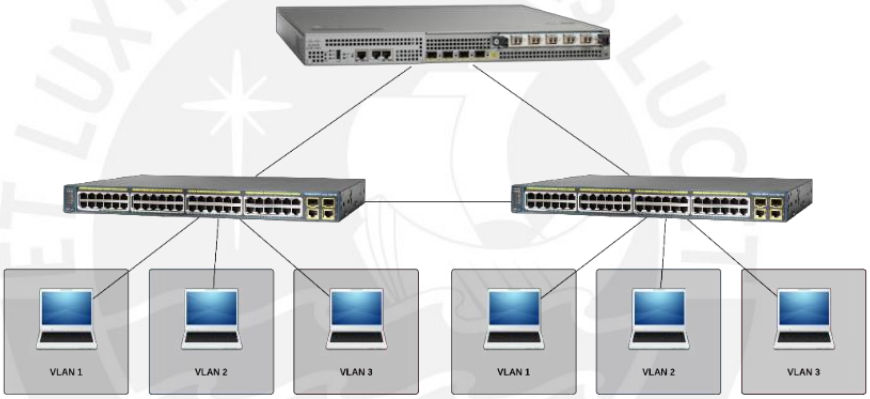

Figura 1-16. Enrutamiento inter-VLAN
ELABORACION PROPIA (2015)

Los Routers son los encargados de realizar la comunicación entre hosts de dominios de broadcast (subnets) diferentes, trabajan en la Capa 3 del modelo OSI (Capa de Red) y analizan las direcciones IP de origen y destino dentro del encabezado del paquete IP para poder tomar la decisión de a dónde enviarlo, a través de una búsqueda en la tabla de rutas que tiene almacenado en su memoria.

Tradicionalmente, los Routers contenían los algoritmos de análisis de rutas, así como del reenvío de los paquetes escritos en un procesador multipropósito; sin embargo, los equipos más actuales tienen divididas estas tareas en módulos de reenvío de paquetes (Forwarding Engine) y de análisis derutas (Routing Engine).

Esta división es con el objetivo de poder optimizar el rendimiento del Router, y se hizo

inicialmente a nivel de software; sin embargo, con la demanda de menor delay en el procesamiento de los paquetes, y el uso de ASICs por este motivo, las memorias RAM convencionales tuvieron que ser reemplazadas por TCAM (Ternary Content-Adressable Memory), las cuales a diferencia de las CAM hacen match a partir de coincidencias parciales, ya que añaden un tercer estado (estado 'X' o 'DON'T CARE') que da una mayor flexibilidad en las búsquedas.

Las memorias TCAM almacenan información como prefijos IP, ACL (Access Lists), QoS, entre otros datos necesarios para protocolos de capas superiores, así como también las acciones que deben seguirse (reenviar, descartar el paquete). Debido a la complejidad de estas, en comparación con las CAM o incluso con tablas en memoria RAM, resultan en un aumento en el precio de los equipos, haciendo que estos tengan un costo elevado.

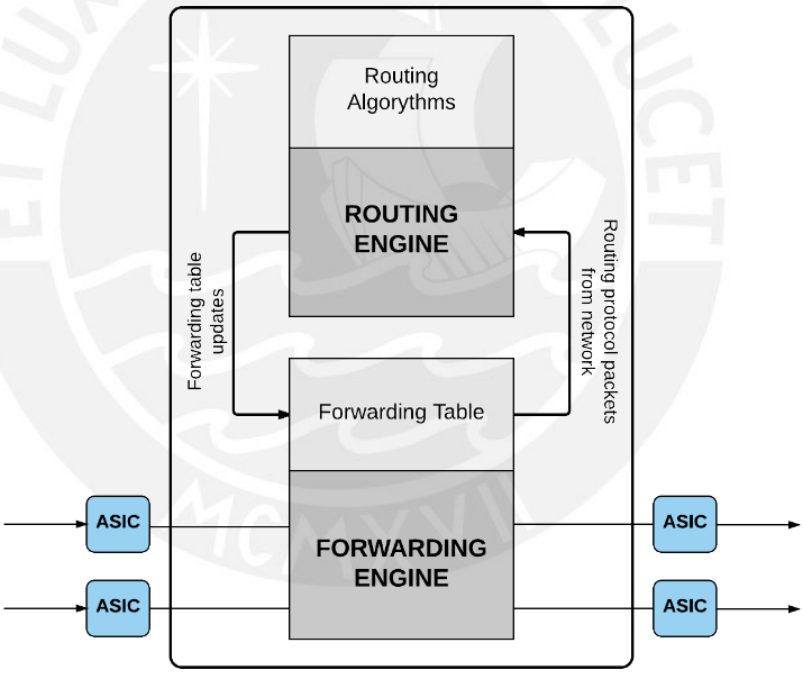

Figura 1-17. Estructura básica de un Router
ELABORACIÓN PROPIA (2015)

Ya que los Routers típicamente no solían contar con una gran densidad de puertos, aparecieron equipos que podían realizar la mayoría de tareas que realizan estos, implementándolas también en hardware (mediante el uso de TCAMs, ASICs y FPGAs), y que tenían una gran densidad de puertos disponibles para poder usar en entornos LAN, llamados Switches de capa 3 (o L3 Switches).

1.2.7. Uso del L3 Switch centralizado

El uso de L3 Switches está muy extendido en entornos de redes LAN con topologías grandes (como es el caso de la red de campus de la PUCP), ya que presenta las siguientes ventajas, tanto de Capa 2 (Velocidades line-rate para el reenvío de paquetes, menor latencia, distribución de los servicios por VLANs, etc) como de Capa 3 (Uso de TCAMs para poder tomar las decisiones de enrutamiento, partición de los dominios de broadcast, direccionamiento lógico, etc).

Sin embargo, el tener como esquema de red un Switch L3 centralizado, tiene las siguientes desventajas:

- Enrutamiento basado en Spanning Tree, lo cual no resulta óptimo, ya que crea cuellos de botella en el Switch mismo, siendo también un punto único de falla para la comunicación inter-VLANs.

- Problemas de escalabilidad, debido al tamaño finito de las memorias de TCAM cuando hay una gran cantidad de usuarios, y al procesamiento de grandes cantidades de tráfico.

- Finalmente, al costo elevado que representa como inversión inicial, debido a la cantidad decomponentes especializados con que cuenta.

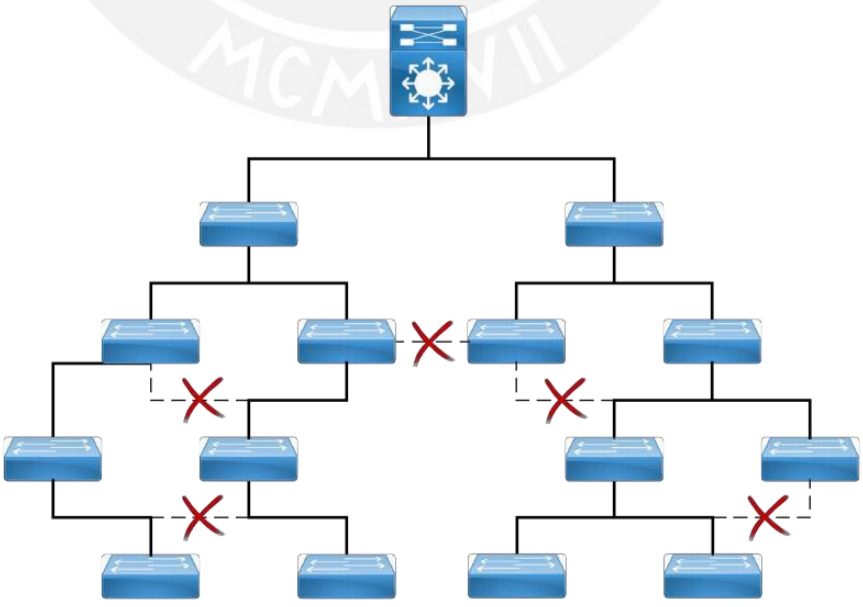

Figura 1-18. Topología tipo árbol con Switch L3 centralizado y STP

ELABORACIÓN PROPIA (2015)

1.2.8. Hacia SDN

Como podemos ver en la figura 1-19, la tendencia en el tiempo ha sido a implementar cada vez más funcionalidades relacionadas al Plano de Datos (forwarding y filtrado de paquetes) en hardware, dejando en software el Plano de Control (inteligencia de la red).

El paradigma de redes SDN surge como consecuencia natural de esta tendencia, y de la necesidad de reducir el costo de los equipos, así como la complejidad de la red, y facilitar el mantenimiento de la misma, brindándole una mayor flexibilidad.

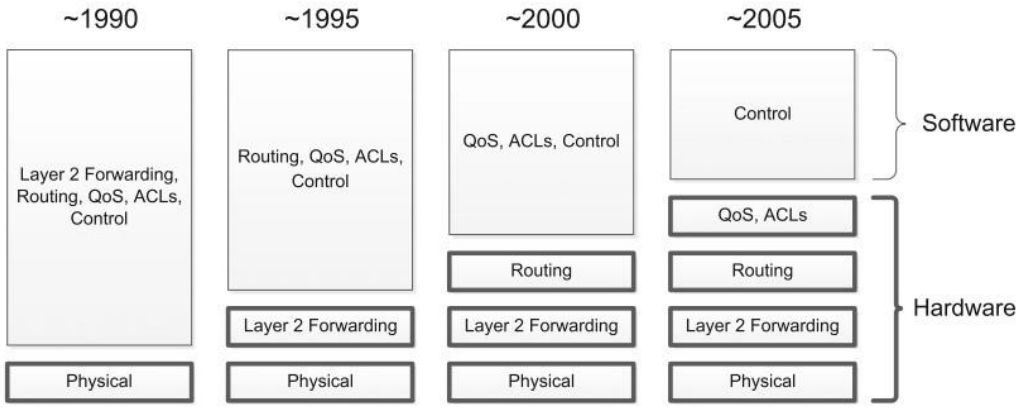

Figura 1-19. Funcionalidades de red implementadas en hardware
SOFTWARE DEFINED NETWORKING. A COMPREHENSIVE APROACH (2014) [5]

Esto se logra a través de la extracción del Plano de Control de los equipos y de su centralización, en un elemento llamado Controlador, el cual tiene un conocimiento completo de toda la red, lo que le permite ser quien tome las decisiones que los elementos de la red (Switches) deben ejecutar en el Plano de Datos.

1.3. Mecanismos de protección de enlaces

La Internet Engineering Task Force (IETF) define la propiedad de supervivencia de una red como la capacidad de una red de mantener la continuidad de los servicios existentes, luego de la presencia de un número dado de fallas en lamisma.

Existen dos tipos de mecanismos para asegurar la supervivencia de una red: mecanismos de restauración (que cuentan con tiempos de respuesta típicos en el orden de los segundos) y de protección (con tiempos de respuesta típicos en el orden de los milisegundos).

Los mecanismos de restauración, son aquellos basados en una recuperación y recálculo en caliente de la red ante una falla. Estos pueden ser implementados de tres maneras: en base a nodos, enlaces o caminos. Por ejemplo, el protocolo de Spanning Tree (STP) utiliza un mecanismo de restauración basado en enlaces, ya que cada vez que existe una falla no vuelve a calcularse el árbol topológico completamente, si no solo la parte de la red que ha cambiado de estado.

Por otro lado, los métodos de protección, o también llamados de conmutación, son aquellos basados en una recuperación de fallas predeterminada, estableciendo no solo una entidad de tráfico activa, sino también una entidad de protección.

Pueden ser implementados en diferentes arquitecturas:

- 1+1 (un camino principal y un camino de respaldo, ambos activos),
- 1:1 (un camino principal, que se encuentra activo, y un camino de respaldo),
- 1:N (1 camino de respaldo para N caminos activos) y
- M:N (M caminos de respaldo para N caminos activos).

Debido a lo ya comentado acerca de la red de campus de la universidad, se considera necesario que esta cuente con un mecanismo que le ofrezca una recuperación rápida ante cualquier falla, por lo que se plantea y se diseña un método de protección basado en caminos, el cual prioriza ciertos tipos de tráfico caracterizados como importantes, como se podrá ver en la sección 4.6, de Ingeniería de Tráfico.

2. SDN/OPENFLOW

El presente capítulo contiene la definición del paradigma de redes SDN, así como también las bases para poder entender las tecnologías aplicadas y las plataformas utilizadas en el despliegue de este.

2.1. SDN

SDN, es un nuevo paradigma de redes de datos, que pretende superar las limitaciones de las mismas, descritas en el capítulo uno. Para esto, se deja de lado la integración vertical y propietaria de los planos de control y usuario de los equipos de red, y se centraliza lógicamente el plano de control. A continuación, se amplía el concepto de SDN, sus antecedentes, su arquitectura, y el concepto actual.

2.1.1. Antecedentes

Algunas iniciativas que guiaron hacia el concepto actual de SDN son:

- ForCES:

  Propuesto en el 2004 por la IETF para separar el Plano de Datos del Plano de Control, pero sin centralización de esteúltimo.

- 4D:

  Propuesto en la conferencia SIGCOMM del 2005, plantea la centralización del Plano de Control, y la visualización total de la red.

- Ethane:

  Propuesto en el 2007, el Forwarding está gobernado por políticas que incluyen niveles de acceso de usuarios, autenticación y cuarentena. El comportamiento de los Switches Ethane, es similar a OpenFlow.

2.1.2. Arquitectura

La IETF emitió el RFC 7426, donde describe los lineamientos principales de SDN y sus principales implementaciones. En dicho documento se describe la arquitectura de SDN

A continuación se lista la terminología básica de la arquitectura SDN, de acuerdo a dicho documento.

- Application Plane:

  Es aquel que comprende las aplicaciones y servicios que definen el comportamiento de la red.

- Control Plane:

  Es aquel que toma decisiones en cuanto a cómo deben ser reenviados los paquetes por cada elemento de red, y comunica dichas decisiones a los dispositivos para su ejecución. Más exactamente, el plano de control configura el Plano de Datos.

- User/Data Plane:

  Es aquel que realiza acciones sobre los paquetes en base a las instrucciones determinadas por el Plano de Control. Estas acciones incluyen reenviar, descartar o modificar los paquetes.

- Management Plane:

  Es aquel encargado del monitoreo, configuración y mantenimiento del Plano de Operación.

- Operation Plane:

  Es aquel que contiene el estado de los dispositivos de red y sus componentes.

En la figura 2-1, se muestra la arquitectura general de SDN y sus componentes, tales

como los describe el RFC en mención.

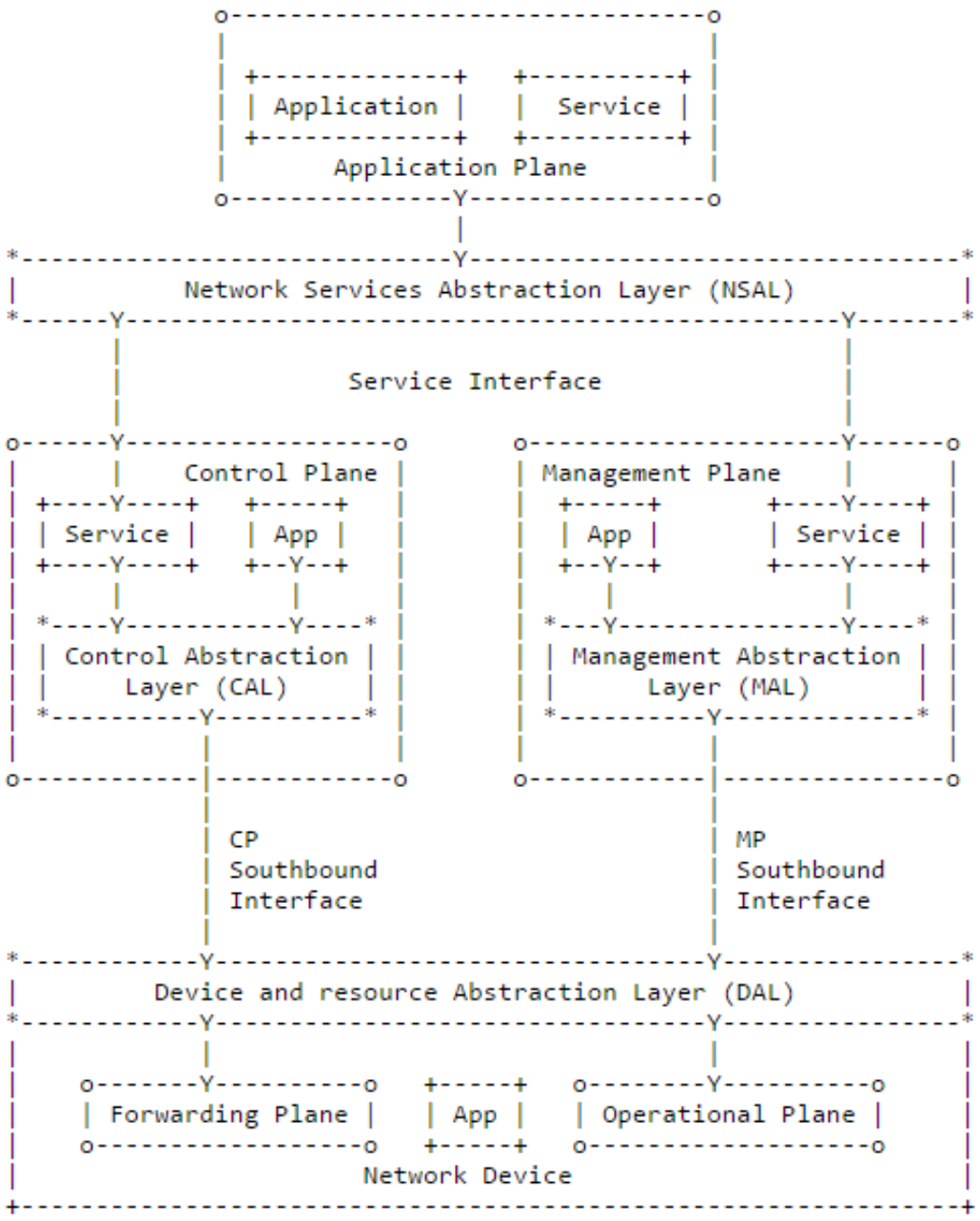

Figura 2-1. Arquitectura en capas de SDN

RFC 7426 (2015)

Si bien han aparecido diversos frameworks y protocolos para la implementación de SDN que no se ajustan necesariamente a la arquitectura propuesta por la IETF, se puede ilustrar de forma más genérica la arquitectura de este paradigma, tal como se muestra en la figura 2-2. Como se aprecia, se han desacoplado los planos de aplicación, de control y de datos de la red presentes en cada dispositivo de red, y los dos primeros se han centralizado. De esta forma, la integración de dichos planos ya

no es vertical, sino horizontal. A continuación examinaremos brevemente las dos interfaces que permiten dicha integración.

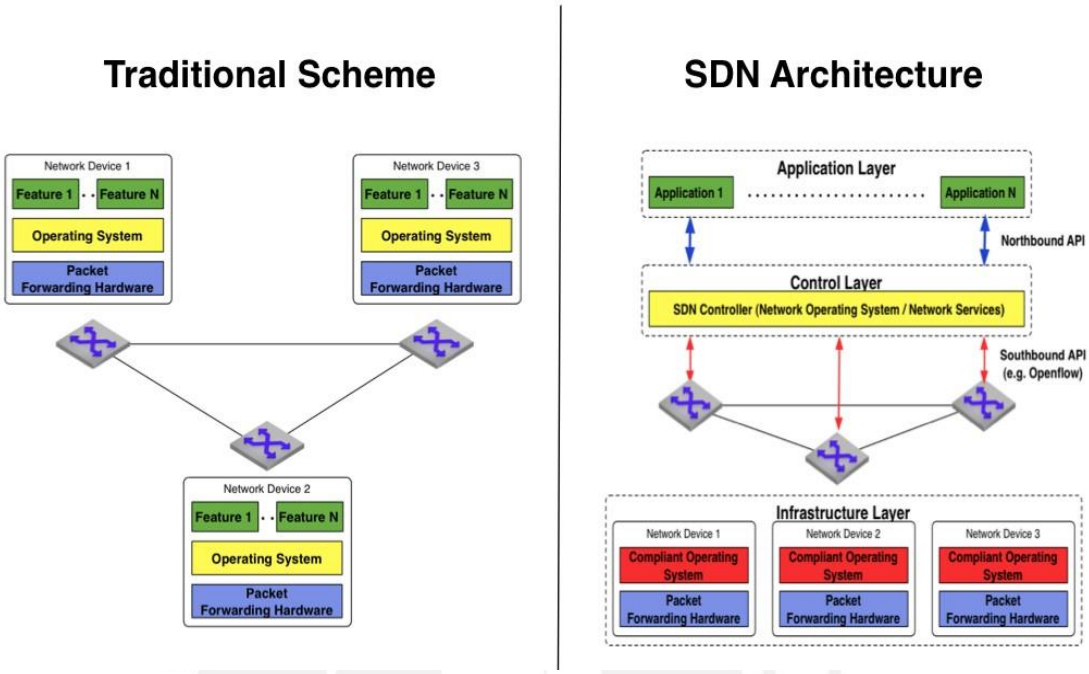

Figura 2-2. Esquema tradicional vs Arquitectura SDN
DATABLAST (2014)

2.1.2.1. Interfaces

2.1.2.1.1. Southbound Interface

Define el protocolo de comunicación entre los dispositivos de Plano de Datos y los elementos del Plano de Control, así como el conjunto de instrucciones que permitan manipular de los dispositivos de forwarding. Un ejemplo de dicha interfaz, es el protocolo OpenFlow.

2.1.2.1.2. Northbound Interface

Define una API para el desarrollo aplicaciones SDN. Típicamente, debe abstraer la complejidad de los dispositivos de red. A diferencia de la interfaz Southbound, no existe estandarización de esta interfaz, por lo que la portabilidad de Aplicaciones SDN sobre distintos controladores no está garantizada.

2.1.3. Concepto actual

Hoy en día, SDN está caracterizado por 4 aspectos:

- Separación del Plano de Control y del Plano de Datos:

  Los protocolos, algoritmos, y la lógica usada para programar las tablas de forwarding de los dispositivos de red, no residen en los mismos.

- Control centralizado:

  El Plano de Control reside en un elemento lógicamente centralizado, desde el cual se gestionan los dispositivos de red.

- Automatización y virtualización de la red:

  Aplicaciones pueden realizar cambios instantáneos y dinámicos, sin necesidad de conocer la complejidad de los dispositivos de red subyacentes.

- Openness:

  Las interfaces utilizadas deben permanecer estandarizadas, bien documentadas y no propietarias.

2.1.4. Escenarios SDN

Los principales escenarios donde se ha implementado SDN, son Data Centers, redes de Campus, y redes WAN.

- En Data Centers, las demandas se resumen en la escalabilidad de las tablas MAC, el número de VLANs y el spanning tree. Se han realizado investigaciones que muestran la capacidad de SDN para *implementar Live network migration*, *rapid deployment*, aprovisionamiento dinamico y elástico, y monitoreo y mitigación de comportamiento indebido de hosts.

- En redes WAN, los factores clave son la supervivencia a fallos y el uso más eficiente posible del ancho de banda, debido al alto costo de este último. Un caso

divulgado y documentado es el de Google.

- En redes de campus, la problemática se centra en la cantidad y variedad de dispositivos de usuario, que además pueden ser móviles, y la heterogeneidad del tráfico que generan. La aplicación de políticas y el acceso personalizado a los servicios es el reto en este escenario.

2.1.5. SDN para redes de campus

La ONF publicó en Setiembre del 2013 un *solution brief* donde se proponen los lineamientos para la implementación de SDN en redes de Campus. Dicho brief describe los objetivos a lograr en las Redes de Campus por SDN:

- Aislamiento de tráfico a través de políticas de gestión granulares aplicados a los flows, facilitando seguridad y multi-tenancy y compliance

- Optimización del ancho de banda a través de la segmentación de la red y el control centralizado de la infraestructura física y virtual. De esta manera la utilización de los dispositivos individuales y de toda la red es más eficiente

- Operación y mantenimiento más eficientes, al simplificar la configuración de la red, y reemplazar la gestión manual y riesgosa con la automatización

- Mayor confiabilidad al tener una selección de ruta y control de *failover* centralizado, para mejorar la disponibilidad de los servicios y aplicaciones

- Mayor agilidad de la red a través de la programabilidad y abstracción que proporciona SDN

- Interoperabilidad multi-vendor, promovida por la arquitectura OpenFlow, que además facilita la adopción de soluciones *Open Source*.

2.2. OpenFlow

OpenFlow es un protocolo no propietario, de propósito general, cuya función es programar el plano de datos de los dispositivos de una red SDN.

2.2.1. Arquitectura de SDN con OpenFlow

Los componentes de una arquitectura SDN/OpenFlow son: el Switch OpenFlow, el controlador, el protocolo OpenFlow y el canal switch-controlador.

La ONF es la entidad que emite las especificaciones OpenFlow, las cuales se centran en el Switch OpenFlow, definiendo sus características, así como el protocolo de comunicación con el Controlador.

En la figura 2-3 se muestra de forma simplificada los componentes de esta arquitectura, los cuales se detallarán a continuación.

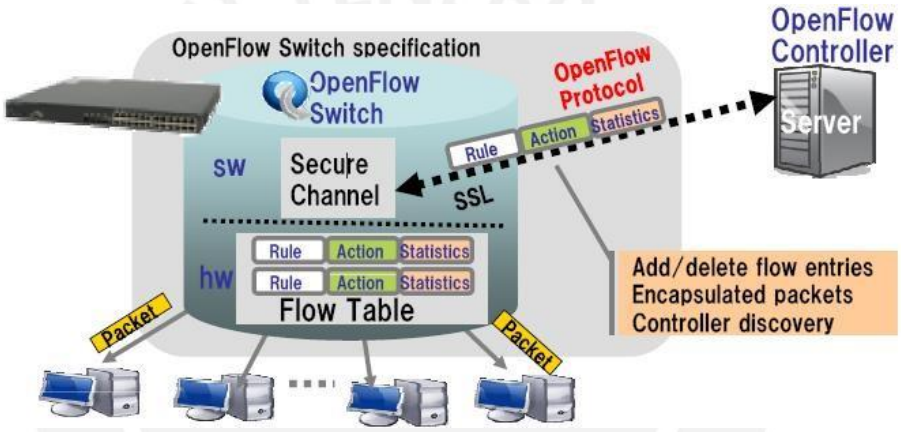

Figura 2-3. Componentes de SDN/OpenFlow
LTGJAMAICA (2015)

2.2.2. El Switch OpenFlow

El Switch OpenFlow consiste en una o más *Flow Tables*, y en una *Group Table*, los cuales realizan la inspección y reenvío de paquetes. El funcionamiento básico se explica a continuación. Cuando un paquete llega por un puerto del Switch, se compara la cabecera del paquete con el campo *Match* de cada *Flow Entry*, en un orden de prioridad. Si se encuentra una coincidencia, se ejecuta la acción correspondiente a dicho *Flow Entry*. De lo contrario, el paquete es enviado al controlador vía el canal OpenFlow o es descartado de acuerdo al comportamiento establecido por defecto.

Los Switches OpenFlow pueden implementarse en hardware o en software, y a su vez pueden ser puros (solo OpenFlow) o híbridos (OpenFlow y Legacy L2 / L3).

Las implementaciones en hardware tienen la premisa de que operan a mayor velocidad que sus contrapartes en software; sin embargo, la complejidad es mucho mayor y por lo tanto, también su precio. Esto se debe a que las *Flow Tables* deben implementarse sobre TCAM, a fin de realizar la búsqueda en line-rate sobre las diferentes tuplas que el protocolo especifica, con la posibilidad de que setear bits en *wilcard*, lo cual no se puede realizar con memorias CAM. Las TCAM no solo son costosas en precio, sino también en energía y espacio. Asimismo, el procesamiento del paquete debe ser realizado en dispositivos de uso específico, como ASIC, en lugar de CPU.

A continuación, la tabla 2-1 muestra las características de algunos modelos de Switches OpenFlow actuales; se puede ver que el uso de TCAM impone una limitación de tamaño en las tablas de forwarding, respecto a los Switches L2, que utilizan memorias de tipo CAM.

Tabla	Broadcom Trident	HP ProVision	Intel FM6000	Mellanox SwitchX
TCAM	~ 2K + 2K	1500	24K	0
L2 / Eth	~ 100K	~ 64K	64K	48K
ECMP	~ 1K	UNKNOWN	0	UNKNOWN

Tabla 2-1. Capacidad de las tablas de Switches
SCALABLE LABEL-SWITCHING FOR COMMODITY ETHERNET (2014)

Ante las dificultades descritas, muchas implementaciones recientes de Switches OpenFlow utilizan una capa de abstracción sobre distintos tipos de hardware de memoria, tales como TCAM, DRAM, SRAM, etc., con lo cual logran un mayor número de entradas en las tablas de forwarding [22]. Además, algunos Switches realizan compresión de las reglas, de tal forma que ofrecen mayor capacidad en las tablas de forwarding cuando las reglas utilizan solo ciertos campos [19].

Cabe aclarar que la ONF determinó que la implementación de algunos de los features de la especificación son mandatorios y otros son opcionales. La mayoría de los features que se revisarán en la sección 2.2.4 son obligatorios.

En el presente trabajo, utilizaremos Switches OpenFlow en software y físicos.

- Como Switch software, utilizaremos el Open vSwitch (OVS) versión 2.3.1.

Este Switch tiene compatibilidad con OpenFlow 1.3. [18].

- Como Switch físico utilizaremos Switches Pica8 modelos P3297 y P3922.

  Estos Switches tienen un modo Legacy (L2 / L3) y un modo OpenFlow, de los cuales solo utilizaremos el segundo. En el modo OpenFlow, se carga un OVS sobre un sistema UNIX. [9]

2.2.3. El controlador

Elemento que posee una visión completa de la red y puede programarse para tomar decisiones óptimas. Se encarga de realizar la creación e instalación de reglas de forwarding en los Switches.

A diferencia del Switch OpenFlow, no existe ninguna especificación referente al controlador; sin embargo, las implementaciones conocidas han dado lugar a un concepto común sobre el mismo, el cual se detallará en la sección 2.3.

2.2.4. OpenFlow Protocol

A continuación, se describen los elementos del protocolo OpenFlow que se utilizan en el presente trabajo, de acuerdo a la especificación de la versión 1.3 [29].

2.2.4.1. Flow Table

La *Flow Table* es una abstracción de la tabla de forwarding de un Switch. Consiste en un conjunto de *Flow Entries*. A partir de la versión 1.1, una Flow Table puede ser un eslabón de una cadena de procesamiento o *Pipeline*, el cual se explicará más adelante.

2.2.4.2. Flow Entry

Una *Flow Entry* es una abstracción de una regla de forwarding. Está compuesta de los siguientes campos: Match fields, Priority, Counters, Timeout Cookie, e Instructions.

Una característica resaltante del Switch OpenFlow, es que soporta la inspección de

cabeceras L2, L3, L4, en total hasta 13 tuplas en la versión OpenFlow 1.3.

A partir de la versión 1.2, se tiene a disposición los OpenFlow Extensible Match (OXM), que son descriptores de campo en forma de par type-length-value (TLV), los cuales incrementan la flexibilidad en el matching de paquetes, pudiendo soportar prácticamente cualquier protocolo. Además, algunos de estos campos soportan wilcard bits, con lo que se puede realizar forwarding en base a dirección red y máscara (enrutamiento de capa 3).

Cuando un paquete es examinado, el Switch busca la mejor coincidencia entre las cabeceras del paquete con algún flow entry. Esta búsqueda se realiza en el orden de prioridad indicado en el campo Priority (de 16 bits), y en el orden de coincidencia más exacta, es decir, con menor cantidad de campos y bits seteados como wilcard. Si esta coincidencia es encontrada, se detiene la búsqueda, se incrementan los Counters, y se ejecutan las acciones indicadas en el campo Instructions.

El campo Instructions contiene un set de *Actions* que el Switch debe realizar sobre paquete. Los Actions que una flow entry puede indicar son: reenviar el paquete por algún puerto físico o lógico, modificar el paquete, o cambiar su estado.

El campo Cookie, de 32 bits, permite al controlador relacionar un paquete recibido con el Flow Entry que tomó la decisión de enviar dicho paquete.

El campo Timeout se divide en dos valores. El Idle Timeout de un Flow Entry especifica cuanto tiempo debe pasar desde el último Match con algún paquete, para que dicho Flow Entry sea borrado automáticamente del Switch. El Hard Timeout especifica cuanto tiempo debe pasar desde la instalación del Flow Entry para que este sea borrado automáticamente del Switch, independientemente si está en uso o no. En ambos casos, el valor de cero indica un tiempo infinito (nunca expirará).

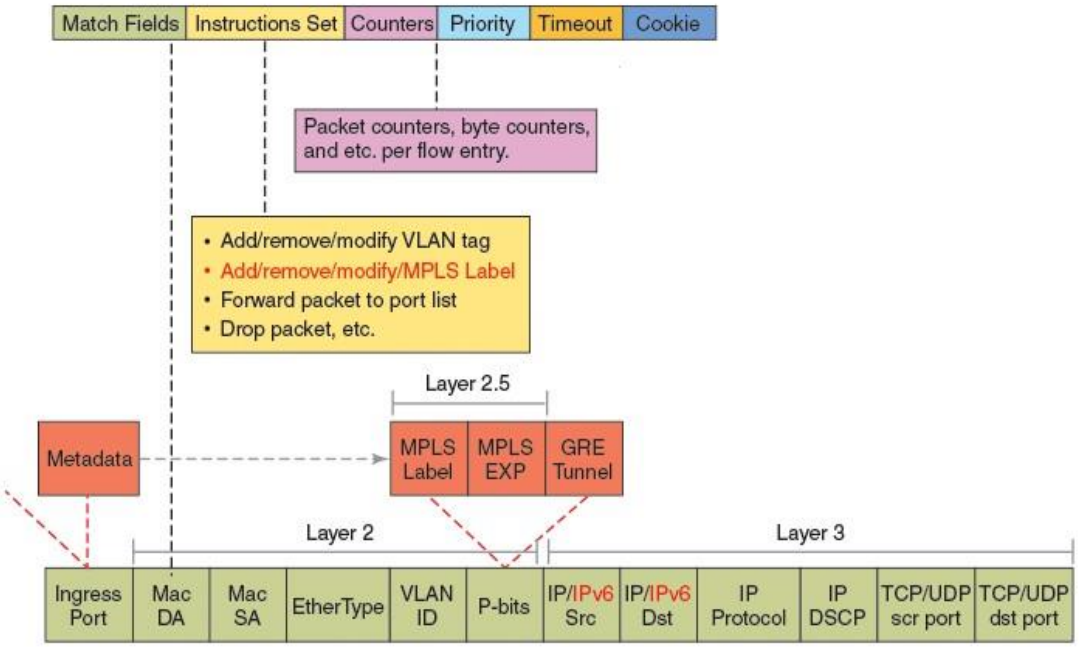

Figura 2-4. Flow Entry

OVERVIEW OF OPENFLOW 1.3 [17]

2.2.4.3. OpenFlow Ports

Abstracción de puertos físicos del Switch. Pueden ser físicos, los cuales se mapean a una interfaz física, como también pueden ser lógicos, los cuales permiten encapsulación de paquetes y el mapeo de uno o varios puertos físicos.

Los puertos reservados definen acciones especiales, como el enviar el paquete al controlador, realizar un flooding por todos los puertos del Switch, o el análisis no OpenFlow del paquete por el mismo Switch.

2.2.4.4. OpenFlow Group Table

Un *group table* consiste en una serie de group entries. Un group entry puede ser apuntado por varios flow entries y puede especificar acciones similares a las soportadas por los estos. Esto permite realizar métodos adicionales de forwarding.

Aplicaciones posibles de los group entries son multicast, fast failover en Plano de Datos, o load balancing.

2.2.4.5. Miss-Match

Cuando se finaliza la búsqueda sin encontrar coincidencia en una tabla, se produce el evento *Miss-Match*. A partir de la versión 1.3 el controlador debe necesariamente insertar una flow entry que determine si el paquete se descarta, se envía por algún puerto, o se envía al controlador para ser analizado.

2.2.4.6. Packet Matching

En la figura 2-5 se muestra un ejemplo de flow table, incluyendo *Match Fields*, *Actions* y *Counters*.

En este caso, si un paquete tiene la dirección IPv4 destino 5.6.7.8, este será enviado por el puerto 2 del Switch, y el Counter de dicha Flow Entry subirá a 301. Por otro lado, un paquete que tenga como puerto TCP destino 25, será descartado y el Counter aumentará a 893.

SRC_MAC	DST_MAC	SRC_IP	DST_IP	TCP_DPORT	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figura 2-5. Ejemplo de Flow Table

DOCKER AND OPENSTACK (2014)

2.2.4.7. Switch-Controller messages

El protocolo OpenFlow define una serie de mensajes entre el Switch y el Controlador, de los cuales los siguientes son fundamentales.

2.2.4.7.1. Switch – Controller:

- Packet-in:

  Transfiere el control de un paquete al Controlador. El paquete no se modificará o se reenviará, hasta que el controlador decida que hacer, mediante un flow mod o un packet out.

- Flow Removed:

  Informa al controlador la eliminación de algún flow entry del Switch por cualquier causa, siempre y cuando el flow tenga seteado el flag correspondiente.

- Port Status:

  Informa al controlador el cambio de estado de un puerto físico.

2.2.4.7.2. Controller – Switch:

- Packet-out:

  Envía un paquete recibido por el Controlador via Packet-in, o creado en el mismo Controlador, a determinado Switch para que sea reenviado por alguno de sus puertos.

- Flow Mod:

  Mensaje que permite añadir, modificar o eliminar flow entries.

2.2.4.8. Multiple Flow Tables – Pipeline

Cuando se tiene múltiples tablas, el paquete puede ser comparado con flow entries de diferentes tablas, si es que el *Flow Entry* coincidente especifica la acción Go To Table. Este procesamiento es conocido como Pipeline.

En cada tabla que procese el paquete y se encuentre coincidencia, se puede añadir un set de acciones, el cual se resuelve al final del Pipeline y se aplica al paquete. En la imagen 2-6 se observa la estructura de un Pipeline y las Flow Tables que la componen. Se observa que al final del Pipeline se ejecuta el Action Set.

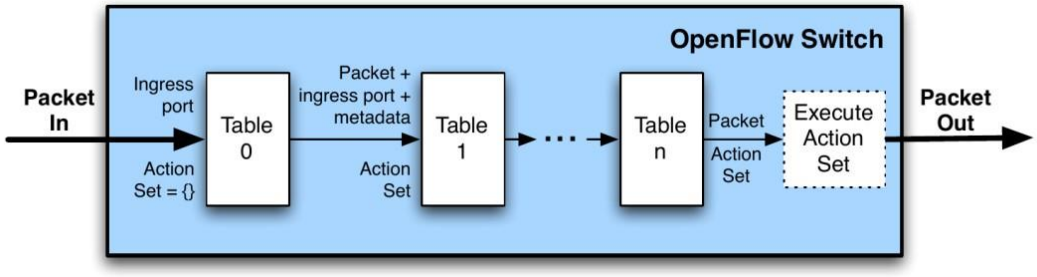

Figura 2-6. Pipeline de un Switch OpenFlow
DOCKER AND OPENSTACK (2014)

2.2.5. Controller-Switch Secure Channel

Para establecer el canal OpenFlow entre el Switch y el Controlador, puede utilizarse el protocolo TLS o TCP plano. El Switch siempre inicia la conexión.

Los canales físicos a través de los cuales se establece el canal OpenFlow, pueden ser los mismos que conforman el Plano de Datos, o puede utilizarse una red distinta y exclusiva para este fin. A continuación se detallan ambos métodos:

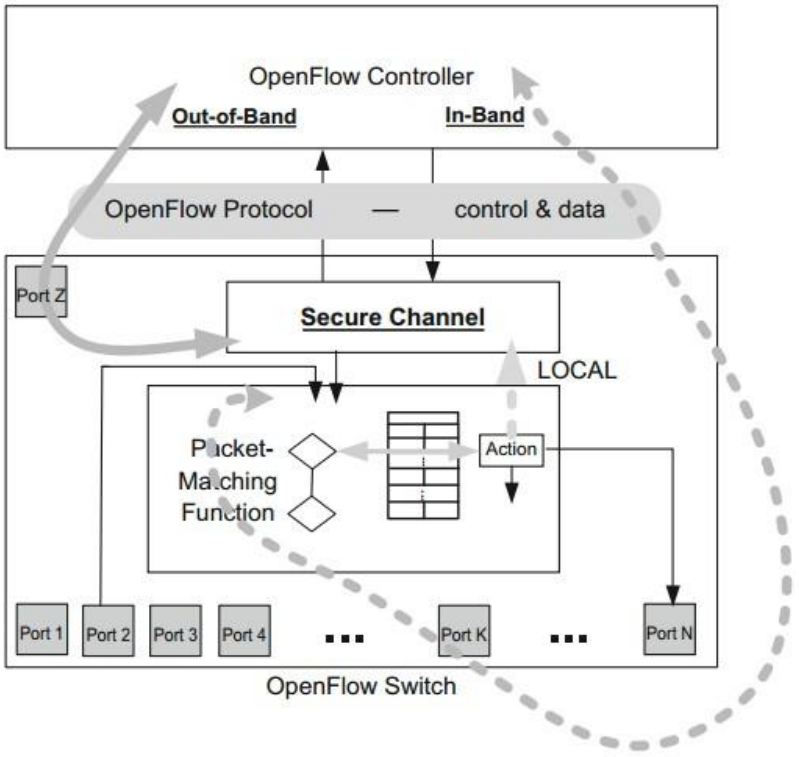

Figura 2-7. In-band vs Out-of-band Control
SOFTWARE DEFINED NETWORKING. A COMPREHENSIVE APPROACH (2014)

2.2.5.1. In-band Control

Debido a que los mensajes OpenFlow utilizarán los puertos del Plano de Datos, deberán ser procesados por las *Flow Tables*. Se requieren insertar *Flow Entries* especiales, que redirijan estos paquetes al control local, el cual se encargará de procesar los paquetes como parte del canal OpenFlow. Su configuración es compleja, pero no requiere de usar conexiones físicas dedicadas para el canal de control. Véase en la figura 2-7 la línea punteada, que muestra el paso de los mensajes del plano de control por la función de *Packet-Matching* y los puertos del plano de datos.

2.2.5.2. Out-of-band Control

En este caso, los canales OpenFlow se establecen a través de puertos de gestión, que no participan del Plano de Datos y, por lo tanto, usan conexiones dedicadas. Es el caso más sencillo de implementar, pero en redes grandes, no es posible tener tantas conexiones dedicadas al control channel, por lo que su implementación no es práctica. Véase en la imagen 2-7 la línea continua, que muestra que los mensajes del plano de control usan directamente el puerto dedicado al canal de control.

2.2.6. Múltiples Controladores

A partir de la versión 1.2 de la especificación OpenFlow, los Switches pueden configurarse con un Controlador Maestro y varios controladores Esclavos, de tal forma que se permita implementar alta disponibilidad a nivel del Controlador.

2.2.7. Evolución y actualidad

En la figura 2-8 se muestran las principales mejoras de OpenFlow a través de sus diferentes versiones.

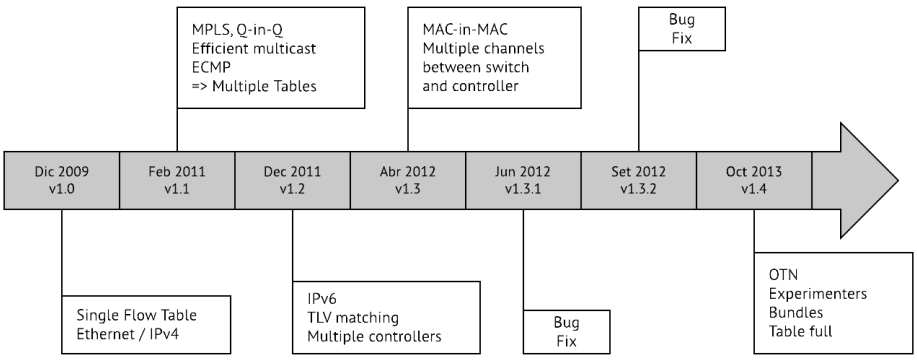

Figura 2-8. Evolución del protocolo OpenFlow hasta la versión 1.4
RAJ JAIN (2013)

La última versión de OpenFlow es la versión 1.5.1, publicada por la ONF en Marzo del 2015. Entre las diferentes mejoras que incluye esta versión, se encuentran la introducción de las Egress Tables, Meter Actions, Matching de flags TCP, entre otros.

En el presente trabajo utilizamos la versión 1.3. Esta versión, además de incrementar la funcionalidad del protocolo, no fue seguida rápidamente de otro release, por lo que dio oportunidad a los fabricantes para desarrollar sus soluciones. Por esta razón, la mayoría de productos OpenFlow en hardware y software soportan dicha versión (entre otras inferiores y superiores).

2.3. Componentes principales de un controlador SDN/OpenFlow

A continuación, se detalla la arquitectura que tienen en común varias de las implementaciones difundidas de controladores OpenFlow.

2.3.1. Diagrama de bloques

Para comprender la arquitectura que siguen los controladores OpenFlow, describiremos brevemente los casos de tres controladores, en orden de antigüedad, así como de complejidad.

El primer caso es el del controlador NOX [6]. Se observa el diseño modular y la distribución de estos módulos en 3 niveles. El primer nivel contiene la integración con la librería OpenFlow, las estructuras de datos, y el manejo de eventos. El segundo nivel utiliza los módulos del primer nivel para crear una abstracción de la red, y ofrecer APIs a los módulos del tercer nivel, que son las aplicaciones SDN básicas.

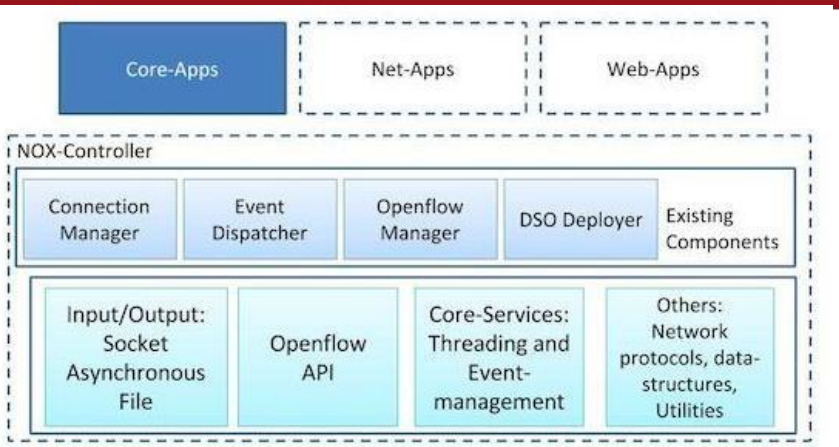

Figura 2-9. Arquitectura de NOX

THE NEW STACK (2014)

En el segundo ejemplo, tenemos el diagrama de Floodlight. Observamos los mismos 3 niveles, pero en este caso, el primer nivel se divide en Módulos de servicios internos, y módulos de servicios core. En la siguiente sección se explicará en qué consisten estos módulos. Cabe mencionar que los módulos de la columna izquierda, son utilidades de software, pero que no interactúan con la red, tales como almacenamiento, estadísticas, servers, etc.

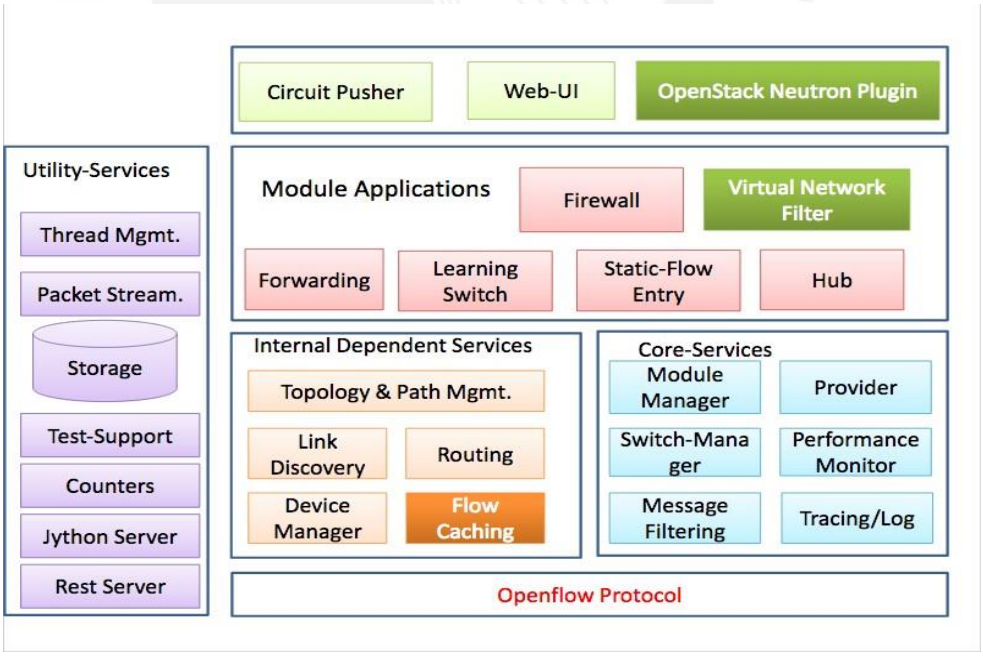

Figura 2-10. Arquitectura de Floodlight

THE NEW STACK (2015)

En el último ejemplo observamos que algunos módulos han cambiado de nivel, y que el nivel más bajo se ha convertido en una capa de abstracción. Este es el enfoque

que utilizan OpenDaylight y ONOS, dos de los controladores más avanzados a la fecha.

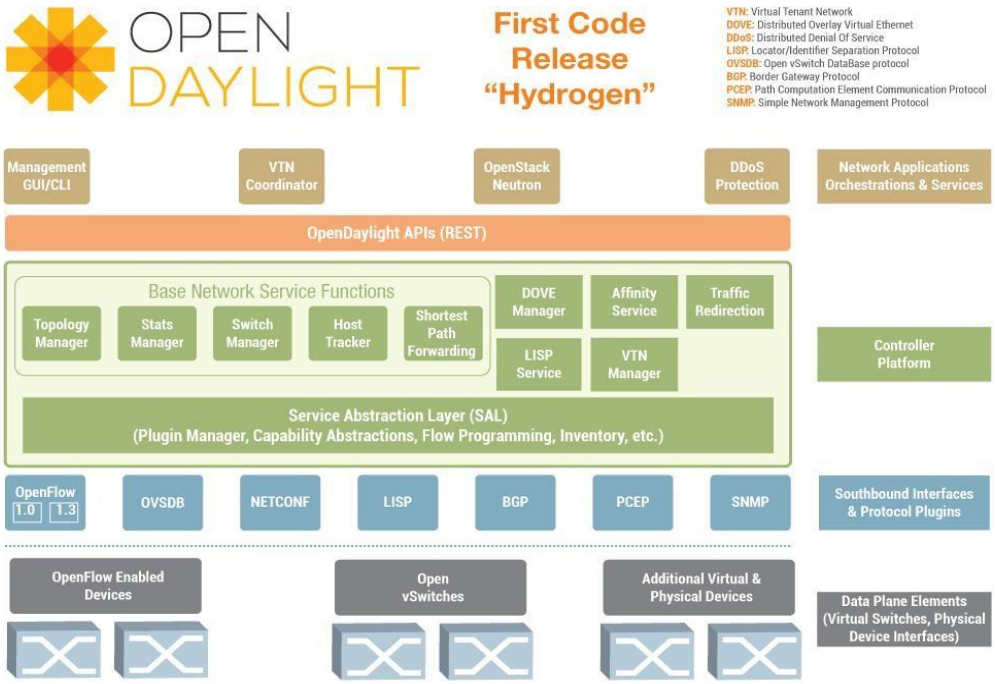

Figura 2-11. Arquitectura de OpenDaylight
ODL OFICIAL (2015) [11]

Goranson y Black, en su libro resumen la arquitectura de los controladores existentes en el siguiente diagrama. Esta arquitectura contempla los 3 niveles, sin embargo considera que la Java API es una interfaz Northbound, mientras que algunos controladores utilizan estas APIs como internas, y APIs REST como interfaz Northbound.

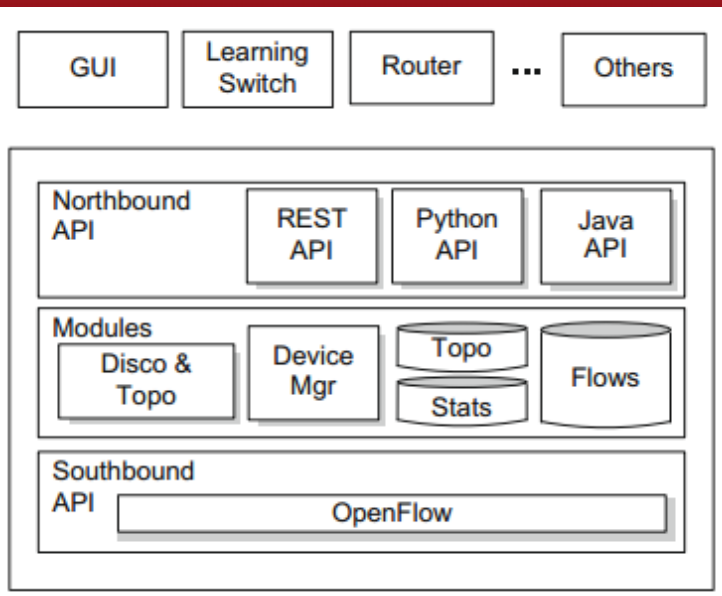

Figura 2-12. Arquitectura de un controlador idealizado
SOFTWARE DEFINED NETWORKING. A COMPREHENSIVE APPROACH [5]

Para explicar con mayor detalle estos niveles y sus módulos, nos basaremos en la arquitectura de Floodlight (Figura 2-10).

2.3.2. Core Modules

Los core modules son clases que proveen la conectividad con los dispositivos de red, proveen almacenamiento en memoria o base de datos a los módulos, gestionan la REST API y la inicialización de los módulos, así como el acceso a las estadísticas y performance interna.

Por ejemplo, el *Switch Manager* se encarga de mantener del envío y recepción de mensajes OpenFlow con los Switches de la red, a través del canal OpenFlow.

2.3.3. Service modules

Los service modules son clases que utilizan alguna librería OpenFlow para interactuar con los dispositivos OpenFlow, para luego representarlos como estados mediante estructuras en memoria, y proporcionar APIs a los Controller modules. Estos módulos no programan el Plano de Datos y por lo tanto no determinan el comportamiento de la red. A continuación, se describen dos módulos como ejemplos:

- Topology Manager:

  Se encarga de descubrir la topología, representarla mediante objetos, mantenerla actualizada y calcular rutas entre los Switches de la red.

- Device manager:

  Se encarga de descubrir los hosts de la red y su ubicación en la topología, representarlos mediante objetos y mantenerlos actualizados.

2.3.4. Module Applications

Los Module Applications son clases que, usando los objetos y métodos expuestos por los *Service* modules, pueden programar el Plano de Datos, añadiendo, modificando o eliminando *Flow Entries* de cualquier Switch OpenFlow conectado al controlador. Las aplicaciones pueden ser reactivas o proactivas, según inserten las reglas después o antes de que llegue el tráfico, respectivamente.

A continuación se listan los principales módulos encontrados en la mayoría de implementaciones:

- *Forwarding*:

  Ante la llegada de un paquete unicast, se busca la ruta shortest path, y se instala las reglas necesarias en los Switches que comprenden la ruta.

- Static flow pusher:

  Permite insertar *Flow Entries* manualmente, en cualquier Switch

- Firewall.

  Permite realizar filtrado de paquetes, en base a header de capas L2 – L4.

- ACL:

  Permite crear listas de acceso de capa 3.

- Port Down Reconciliation:

Ante la caída de un puerto, realiza la eliminación de las reglas de las rutas involucradas, y el restablecimiento de la ruta.

2.3.5. REST Applications

Los *REST Applications* son aplicaciones que utilizan APIs únicamente del tipo REST, provistas por los Controller y *Services* modules, y que representan una abstracción aun mayor de la red física.

Un ejemplo de REST Application es la Web UI (User interface) de Floodlight. Esta aplicación utiliza llamadas de la REST API de Floodlight para obtener información de estado de los diferentes elementos de la red en formato JSON, incluyendo Switches, enlaces, hosts, y utiliza algunas herramientas de software web, tales como jQuery, Bootstrap y Backbone.js para mostrar esta información de forma gráfica, dinámica y cómoda para el usuario, en un navegador web.

2.3.5.1. REST API

Una API REST (Representational State Transfer) cumple con seis principios básicos:

- Usa exclusivamente métodos HTTP
- La información transferida no se almacena en el servidor
- Puede guardarse en caché
- Utiliza modelo cliente-servidor
- Expone directorios del tipo estructura, como por ejemplo, URIs
- Transfiere información en formato XML, JSON, o ambos

2.3.5.2. Aplicaciones proactivas y reactivas

Las aplicaciones proactivas anticipan la aparición de un nuevo flujo e insertan las reglas que consideren necesarias. Estas aplicaciones pueden implementarse usando REST APIs o APIs nativas. Generalmente, lo ideal es que las aplicaciones proactivas reciban como input algún estímulo externo al plano OpenFlow, como por ejemplo los protocolos de gestión sFlow y SNMP.

Por el contrario, ante la aparición de un nuevo flujo de paquetes, las aplicaciones reactivas analizan el primer paquete y en base a este decide qué hacer con dicho flujo, insertando las *Flow Entries* necesarias. Estas aplicaciones no son eficientemente implementadas usando REST APIs, pues sería necesario que la aplicación sea el servidor y el controlador el cliente. Es por ello que para implementar estas aplicaciones, es necesario el uso de llamadas asíncronas al controlador, las cuales sólo pueden ser implementadas a través de APIs nativas (por ejemplo APIs Java).

2.4. Implementaciones más conocidas de Controladores open source

A continuación detallamos algunas de las implementaciones más usadas y documentadas al día de hoy, de controladores OpenFlow.

- NOX:

  Fue el primer controlador OpenFlow conocido, impulsado como un Network OS. Está escrito en C++ en gran parte, y soporta la versión 1.0 del protocolo. (NOX paper). Una versión de NOX en Python, llamada POX, fue usada durante mayor tiempo, aunque posee una performance menor.

- Beacon:

  Fue el primer controlador OpenFlow desarrollado en Java por David Erickson de la universidad de *Stanford*. Utiliza el framework OSGi para realizar la carga y detención de módulos en *runtime*, y *Spring* como IoC.

- Floodlight:

  Surgió como una rama de Beacon, a la cual se le quitó el framework OSGI. Su desarrollo y soporte fue continuado por Big Switch Networks, empresa que lo utilizó como base de su controlador comercial *Big Network Controller*. Soporta las versiones desde 1.0 hasta 1.3, y posee una REST Api más extensa que Beacon. Provee además, un plugin para su integración con OpenStack. Floodlight utiliza la librería *OpenFlowJ Loxi* como Java API para el protocolo OpenFlow. Es el controlador con mayor documentación, soporte, y actividad en la comunidad OpenSource.

- Ryu:

  Controlador desarrollado en Python, soportado por NTT labs. Además de OpenFlow 1.0, 1.2, 1.3, 1.4 y1.5, soporta NETCONF, OF-config, entre otros. Tiene una arquitectura de programación modular y basada en eventos, que junto con el lenguaje tipo scripting, hacen de RYU un controlador ideal para el desarrollo fácil y rápido de aplicaciones SDN. Posee admás, un plugin para una fácil integración a OpenStack.

- OpenDaylight:

  Controlador también basado en Beacon, cuyo desarrollo está dirigido por la Linux Foundation, y cuenta con un soporte de aproximadamente 40 compañías del sector, incluyendo Cisco, HP, Intel, NEC, Huawei, IBM, Microsoft, enter otras. Soporta una lista aun mayor de protocolos Southbound que RYU. Una de los features que distinguen a OpenDaylight, es que proporciona una capa de abstracción llamada SAL. La SAL permite desarrollar aplicaciones independientes no solo de la complejidad del hardware de red o de protocolos, sino también de otras aplicaciones (abstracción vertical y horizontal).

- ONOS:

  Controlador desarrollado en Java. Orientado a los *Service Provider*, posee un núcleo distribuido, lo que le otorga una escalabilidad y disponibilidad que otros controladores no tienen por defecto. Su desarrollo es dirigido por la ON.Lab, y participan empresas como Cisco, Alcatel-Lucent, Huawei, NEC, entre otros.

En la tabla 2-2 se muestra un resumen comparativo de los controladores descritos.

Características	NOX	POX	Ryu	Floodlight	OpenDaylight	ONOS
Lenguaje de Programación	C+	Python	Python	Java	Java	Java
Performance	Alta	Baja	Baja	Alta	Alta	Alta
OpenFlow	1	1	1.0 - 1.4	1.0 y 1.3	1.0 y 1.3	1.0 y 1.3
Curva de aprendizaje	Moderada	Suave	Suave	Moderada	Pronunciada	Pronunciada
Año	2008	2009	2012	2012	2013	2014

Tabla 2-2. Controladores y características principals

ELABORACIÓN PROPIA (2015)

3. CONSIDERACIONES DE DISEÑO

En este capítulo, se desarrollan los requerimientos a considerar en el diseño del controlador y las principales limitaciones y dificultades que se pueden encontrar. Finalmente, se explica la operación del controlador, detallando los mecanismos que se implementarán.

3.1. Requerimientos generales de la solución

3.1.1. Subnetting

Para aprovechar las ventajas de una red en capa 2, es necesario que todos los nodos pertenezcan a la misma subred, con la máscara de capa 3 apropiada. Esto eliminará la necesidad de enrutadores y gateways entre un origen y un destino, aumentando la escalabilidad.

3.1.2. Open Source

Una solución Open Source evita la dependencia de un fabricante en particular y facilita la modificación/actualización del sistema. El uso de software y hardware validados en entornos abiertos asegura la interoperabilidad entre los componentes hardware y software de la red.

3.1.3. Amplia comunidad de soporte

La existencia de una amplia comunidad de soporte facilita el desarrollo, optimización y troubleshooting de la solución, a pesar de no tener un contrato de soporte.

3.1.4. Rapid prototyping

El lenguaje de programación y los frameworks son determinantes en la rapidez de *prototyping* y desarrollo. Los lenguajes de alto nivel permiten lograr menores tiempos de desarrollo mediante el uso aplicaciones REST, mientras que lenguajes de bajo nivel permiten implementar programas más complejos usando directamente las librerías OpenFlow y así. Sin embargo su curva de aprendizaje es mayor.

Se busca una solución que permita un rápido desarrollo, sin sacrificar performance ni escalabilidad.

3.1.5. Performance

La performance de un controlador está directamente ligada a la implementación de sus módulos core, como también a la plataforma de hardware sobre la cual se ejecuta.

Se requiere que el controlador sea óptimo en performance, pudiendo procesar la cantidad esperada de Packet In, ejecutándose sobre plataformas de servidores comerciales.

3.1.6. Escalabilidad

El controlador debería soportar la conexión simultánea de al menos 10 000 hosts, generando hasta 40 000 conexiones simultáneas. Se buscara soluciones que permitan crecimiento a 20 000 hosts (por ejemplo, usuarios móviles) y 160 000 conexiones simultáneas.

3.2. Limitaciones y requerimientos de escalabilidad en OpenFlow

3.2.1. Plano de Datos

3.2.1.1. TCAM de los Switches

Debido al elevado precio de las TCAM, en comparación a otros tipos de memoria hardware, los actuales Switches comerciales proveen una TCAM bastante reducida, logrando aproximadamente 8 000 entradas como máximo.

En la tabla 2-1 se mostró una comparación de la cantidad de entradas de TCAM encontrados en Switches comerciales.

En el presente trabajo, se ha utilizado el Switch Pica8, el cual tiene capacidad para 8192 entradas TCAM. En el caso particular de este Switch, cuando las entradas TCAM se ocupan completamente, se puede utilizar otras formas de memoria para almacenar *Flow Entries,* de acuerdo a lo explicado en la sección 2.2.2. Por lo tanto,

en nuestro caso, el Switch soportará indefinidas entradas. Sin embargo, para efectos de aplicar nuestra solución a Switches físicos, nos limitaremos a utilizar solo las entradas en TCAM.

3.2.1.2. Ancho de banda y delay del Plano de Datos

El ancho de banda utilizado por la red, producido por los usuarios, deberá ser optimizado al reemplazar *Spanning Tree Protocol* por el *Shortest Path Forwarding*. Sin embargo, el tráfico de broadcast de una subred de varios miles de nodos, llega a generar el suficiente volumen como para ocupar un porcentaje considerable de la capacidad de los enlaces. Esto dependerá de cómo se lleve a cabo el manejo del tráfico broadcast por el controlador.

En cuanto al *delay*, una alta granularidad en las *Flow Entries* puede ocasionar que el primer paquete todos los nuevos flujos sea procesados por controlador, aumentando el delay de dicho paquete, y del flujo si está compuesto por pocos paquetes. Es por ello que el mecanismo base (*baseline)* del controlador debería considerar reglas basadas en el destino, para así disminuir la cantidad de Miss Match. Además las solicitudes ARP deben utilizarse para predecir un flujo unicast y así disminuir aún más las consultas al controlador y por lo tanto, el delay promedio de los flujos.

3.2.2. Plano de Control en base al controlador y al control cannel

El origen de las limitaciones puestas por el controlador, es el tiempo que demora en procesar un paquete, hasta que determina la acción, envía los Flow Mods y el Packet out a los Switches correspondientes. Este periodo de tiempo es conocido como *Service Time*. La inversa del *Service Time*, es el número de paquetes que el controlador puede procesar en un segundo, valor que llamaremos *Arrivals*. El *Service Time* máximo, nos indica el valor máximo de *Arrivals* que soportará el controlador en un segundo, entendiéndose que más allá de este valor, los buffers TCP se bloquearán.

Tras realizar pruebas con el controlador, se determinó que el valor máximo de *Arrivals* que soporta fue de 56 768 paquetes por segundo.

3.2.2.1. Uso del procesador y memoria del controlador

El uso de procesador y memoria del controlador bajo recursos de hardware apropiados, se debe a diversos factores, entre ellos el lenguaje de programación (compilado o interpretado), la librería OpenFlow usada, operación en Kernel o User Space, manejo de sockets I/O, implementación de Multi Threading, frameworks de programación, etc. Estos factores no estarán bajo nuestro control, pues son determinados por la implementación particular de la plataforma de controlador que se elegirá. La optimización de estos factores queda fuera del alcance de esta tesis.

3.2.2.2. Ancho de banda del Plano de Control

El canal de control impondrá limitaciones dependiendo de si se implementa *out of band* o *in band*. El primer caso no debería traer limitaciones, pues la capacidad de un enlace típico no es comparable a la ocupación que el tráfico del plano de control generará. Sin embargo, si se implementa control in-band, debe considerarse que el tráfico de Plano de Control no debe superar cierto porcentaje de la capacidad de los enlaces, a fin de garantizar un ancho de banda al tráfico de usuario. Consideraremos este valor como el 10% del total de la capacidad del enlace.

3.3. Segmentación de la red física

3.3.1. Redes lógicas con OpenFlow

Debido a que la segmentación por VLANs no es un mecanismo eficiente para lograr escalabilidad, será dejado de lado (aunque sí es soportado en OpenFlow), y no se podrá aprovechar su uso para dividir la red de forma administrativa y lograr un simple aislamiento de conectividad entre las subnets. Por este motivo, es importante emplear otro mecanismo para segmentar la red a nivel lógico, y que diferentes aplicaciones de red puedan utilizar esta segmentación para su operación.

3.3.2. Requisitos de la segmentación lógica

3.3.2.1. Segmentación de tráfico

El controlador debe permitir la existencia de diferentes redes lógicas, y un host puede pertenecer a una o a varias redes lógicas. Por defecto, el tráfico entre estas redes

lógicas debe estar habilitado, sin embargo, debe permitirse el bloqueo a solicitud del administrador o de alguna aplicación, o la creación de redes totalmente aisladas, independientemente de la infraestructura.

3.3.2.2. Movilidad y flexibilidad

Debido al creciente uso de dispositivos móviles y la tendencia BYOD, la tradicional segmentación por VLANs deja de ser eficaz. Se requiere que aplicaciones que manejan la red en alto nivel, como un NAC, realicen la segmentación de la red en base a direcciones IP, roles de usuario, entre otros. Para esto, el controlador debe proporcionar una REST API que represente una abstracción flexible de la agrupación de dispositivos. Para lograr esta flexibilidad, dicha agrupación debe hacerse en base a cualquier parámetro de red de los dispositivos, tal como se describe a continuación:

- En base a Switch y puerto:

  Emulando la forma tradicional de implementar VLANs, esto otorga cierta protección ante suplantación de direcciones MAC o IP en redes cableadas.

- En base a la dirección MAC:

  Definiendo el hardware físico que será parte de la red lógica, pero permitiendo cierta movilidad.

- En base a la dirección IP:

  Garantizando la movilidad de los dispositivos asociados a dicha red lógica.

La asociación de dispositivos descrita debe ser mantenida de forma automática y en tiempo real por el controlador.

Debe ser posible que los hosts de una misma red lógica estén definidos a partir de diferentes parámetros. Asimismo, debe ser posible que un host pertenezca a varias redes lógicas o que pertenezca a la red lógica por defecto.

3.3.2.3. Flooding flexible

El flooding de paquetes es necesario para el tráfico broadcast distinto de ARP y DHCP. La segmentación lógica debe permitir la creación de dominios de interés, que permitan hacer un flooding limitado a los dispositivos que pertenecen al dominio de interés, evitando que el flooding llegue a todos los dispositivos de la red.

3.4. Caracterización y clasificación de tráfico

3.4.1. Tráfico Broadcast

De acuerdo a datos de la red PUCP, se puede observar que la mayor cantidad de tráfico Broadcast se dirige al *Gateway*, y que este se limita gracias a la segmentación de VLANs. Debido a que en el nuevo esquema de red propuesto se requiere el uso de una sola *subnet*, el tráfico broadcast distinto a ARP y DHCP será controlado por un mecanismo que realice la distribución del tráfico mediante árboles multicast, a los nodos interesados en dicho tráfico.

3.4.2. Tráfico Multicast

El tráfico Multicast presente en la universidad proviene en su mayoría de protocolos de Capa 3, y de ciertas aplicaciones de video. Para el alcance de esta tesis, el tráfico multicast capa 3 tráfico será tratado como broadcast.

3.4.3. Tráfico Unicast

El tráfico Unicast representa el mayor volumen de datos de la red, y conviene tratarlo de forma especial. En base a los datos de la red PUCP, se llega a la conclusión de que la distribución de destinos no es uniforme, si no que algunos hosts representan la mayor cantidad de destinos, tal como se explicará acontinuación.

3.4.3.1. Modelo de elefantes y ratones

A diferencia de otras redes (como las redes telefónicas), se ha demostrado que el tráfico en Internet y en Data Centers muestra un comportamiento del tipo *Self Similar*, que puede ser modelado con mayor precisión mediante una distribución Pareto [25]. En términos prácticos, el tráfico en estas redes está compuesto por unos pocos (por

ejemplo, el 20%) flujos de larga duración y que consumen el mayor ancho de banda de la red, conocidos como *Elephant Flows*, mientras que la mayor cantidad de flujos (por ejemplo, 80%) son cortos y "*bursty*", conocidos como *Mice Flows*. En base a lo observado en capturas de tráfico dentro de la red de campus, este comportamiento se encuentra en el campus, y será considerado para el diseño del controlador y para el análisis de escalabilidad del mismo.

3.5. Concepto de operaciones del controlador

3.5.1. Manejo del broadcast

Para evitar una sobrecarga del plano de control, y además optimizar el curso del tráfico Broadcast, proponemos que el plano de control realice el envío de este tráfico de origen a destino, en tantos casos como sea posible. Cuando esto no sea posible, el flooding del tráfico debería ser lo suficientemente óptimo como para evitar que un paquete broadcast deba llegar a cada elemento de la red.

3.5.1.1. ARP

Asumiendo que en un tiempo muy largo de operación, tras el cual el número de hosts activos en la red es aproximadamente constante, el Controlador conocerá los tres parámetros de cada host. Debido a esto, será conveniente que el controlador intercepte los ARP Request, cree los ARP Reply respectivos, y los envíe directamente al host interrogante.

Este mecanismo tiene las siguientes implicaciones:

- La cantidad de broadcast se reduce, pues se convierte en tráfico unicast.
- La mayor cantidad de broadcast utilizará el canal de control.

Por otro lado, debe considerarse que en estados de transición, sea cuando el controlador inicie (o reinicie) su operación, o cuando se conozcan nuevos hosts, muchos ARP Request no podrán ser contestados por el controlador, por lo que deberá utilizarse un flooding limitado para garantizar el servicio.

3.5.1.2. DHCP

El tráfico DHCP tiene la característica especial, que tiene como objetivo implícito algún servidor DHCP que esté activo en la red. Esto puede ser aprovechado por el controlador, el cual puede capturar los paquetes DHCP, y enviarlos al servidor DHCP, para así controlar el handshake completo y evitar el flooding de los paquetes DHCP broadcast.

3.5.1.3. Otros tipos de broadcast

Existen otros tipos de broadcast, como *Service Advertisement* particulares para diversos servicios de la red, los cuales no tiene algún objetivo implícito como el caso de paquetes ARP y DHCP, sino que deben llegar a los dispositivos que estén interesados. En este caso, debe hacerse un flooding, que evite sobrecarga en el ancho de banda y en el controlador.

3.5.2. Manejo de multicast

Debido a que el tratamiento del tráfico multicast está fuera del alcance de esta tesis, será manejado por defecto como broadcast.

3.5.3. Enrutamiento de unicast.

Se ha descrito como el tráfico unicast puede ser descrito y modelado. A continuación se explica el mecanismo de operación para el tráfico unicast.

3.5.3.1. Baseline: Clustering

Se tendrá un mecanismo base, que realice un enrutamiento del *tipo Shortest Path* en base al destino, de tal manera que se consiga escalabilidad. El mejor método para lograr esto es realizar un enrutamiento en base a etiquetas (similar a MPLS) en el core de la red. El uso de etiquetas permite agrupar hosts por cada Switch de acceso, para así disminuir la cantidad de reglas en cada Switch. Este mecanismo, se denominará *Clustering*.

En los Switches de acceso, se deberá utilizar *Flow Entries* con timeout finito, que realicen el etiquetado y *Forwarding* de los paquetes hacia el core de la red.

Para explicar el mecanismo, se utiliza el siguiente escenario como ejemplo. En este caso, las rutas en el core de la red han sido establecidas previamente. En este ejemplo, el host “MAC A” envía un paquete unicast a “MAC B”. La imagen muestra como el paquete es encapsulado y enviado al controlador, el cual instala las reglas necesarias en todos los Switches y luego envía el paquete encapsulado en un *Packet Out* para ser reenviado por el puerto correspondiente.

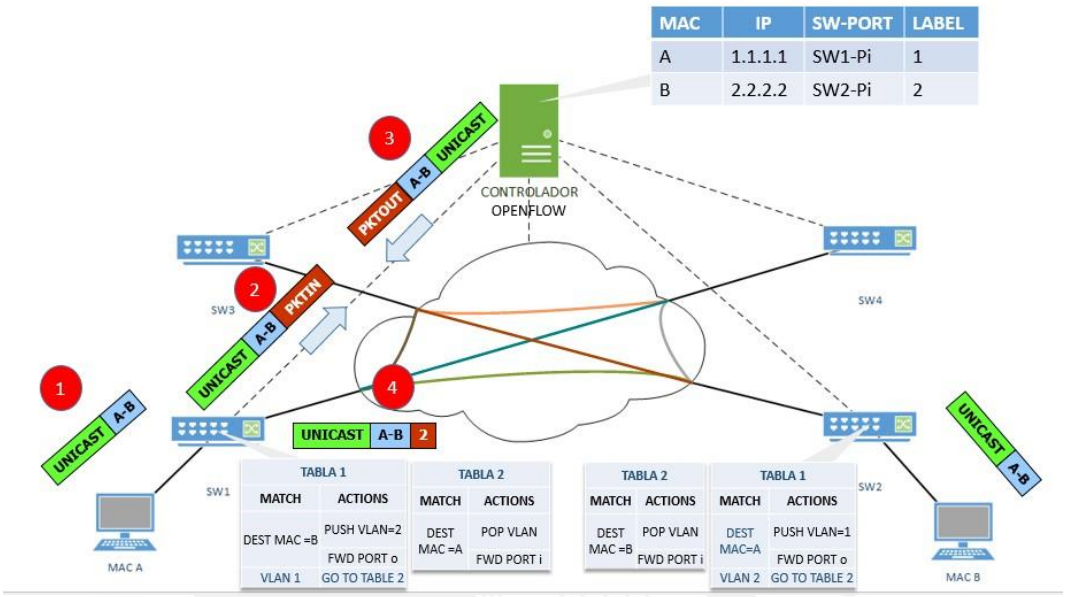

Figura 3-1. Envío del primer paquete

ELABORACIÓN PROPIA (2015)

A partir del segundo paquete, el flujo será como el mostrado en la figura 3-2.

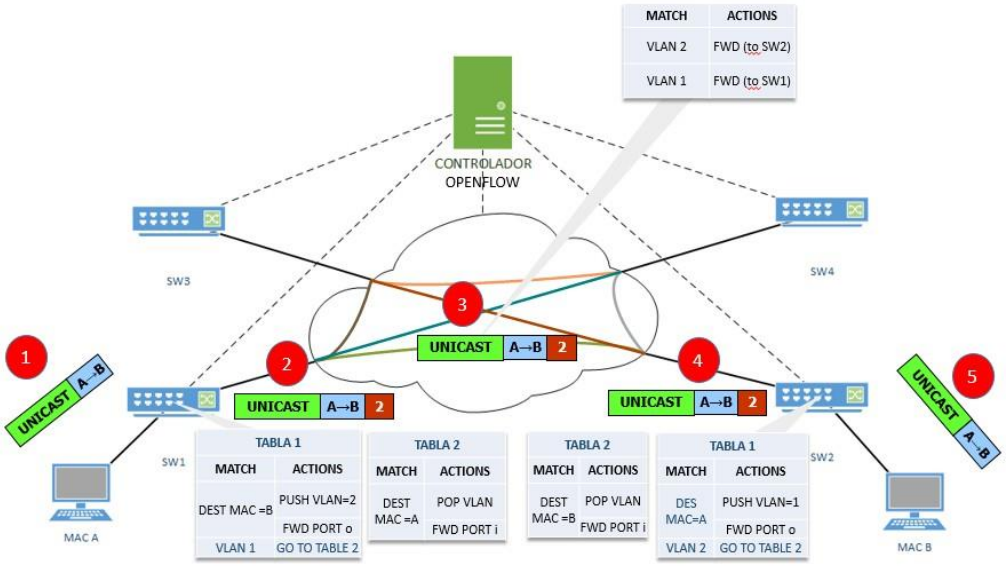

Figura 3-2. Establecimiento de ruta

ELABORACIÓN PROPIA (2015)

De esta forma se ha creado una ruta para los paquetes que vayan entre Host A y Host B o viceversa. Se resalta que estas reglas son basadas en el destino. Si un host "MAC C" conectado al SW1 envía paquetes al host "MAC B", utilizará las mismas Flow Entries, y el primer paquete no tendrá que ser analizado por el controlador, ni este tendrá que instalar reglas adicionales.

3.5.3.2. Re-enrutamiento de Flows elefantes

Adicionalmente a los mecanismos anteriores, debe utilizarse un mecanismo que permita un enrutamiento con mayor granularidad, garantizando una ruta con menor congestión para los flujos que se consideren "elefantes", y de forma simultánea, evitando la congestión, Para lograr esto, debe utilizarse algún mecanismo de monitoreo de ocupación de enlaces, y calcular las rutas en base a los costos determinados por el mecanismo de monitoreo, en nuestro caso, un colector de estadísticas que usa el protocolo sFlow (sFlow Collector).

Para ilustrar como operan estos módulos de forma conjunta para lograr el re enrutamiento de los Flows Elefantes, véase el siguiente ejemplo. En la figura 3-3. El host A establece un flujo con el Host B usando alguno de los mecanismos descritos anteriormente (las líneas punteadas son los tramos de la ruta). Debido a que la ruta usada no contempla el uso de costos en base a alguna métrica, dicha ruta tiene un enlace congestionado en el core (línea punteada en rojo).

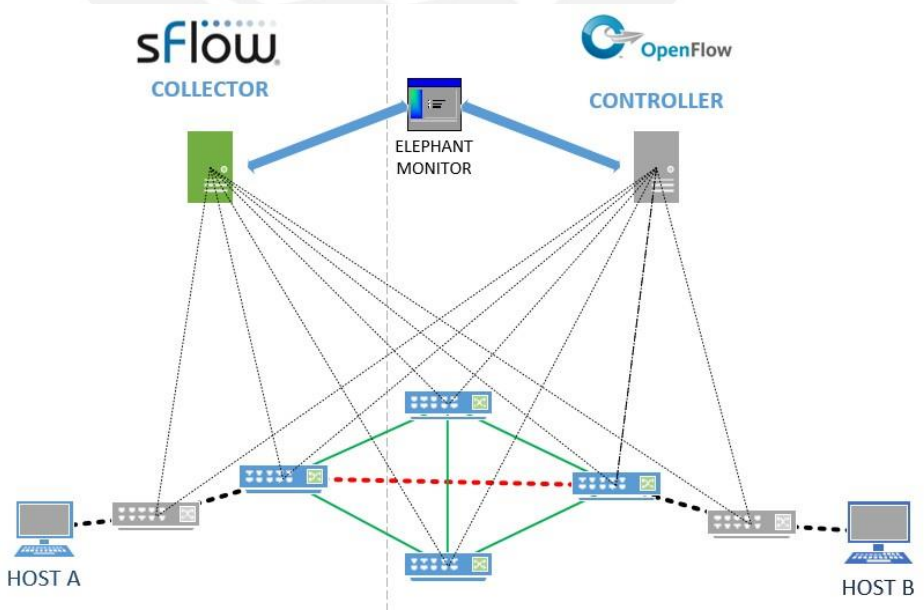

Figura 3-3. Elephant Monitor
ELABORACIÓN PROPIA (2015)

Ante esta situación, el colector sFlow detecta que se establece el flujo y que este ocupa gran ancho de banda en el puerto de entrada del Switch de acceso, e informa al mecanismo de monitoreo (Elephant Monitor). El *Elephant Monitor* ordenará al controlador que se reasigne una ruta que esté descongestionada.

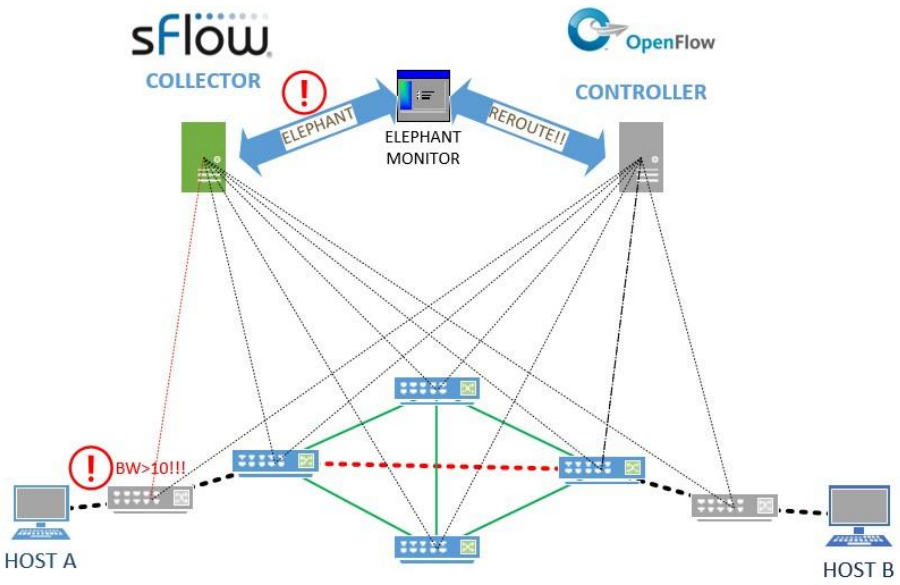

Figura 3-4. Elephant Detection
ELABORACIÓN PROPIA (2015)

Luego de haber insertado las reglas, el Flow Elefante es cursado por enlaces no congestionados (enlaces de verde).

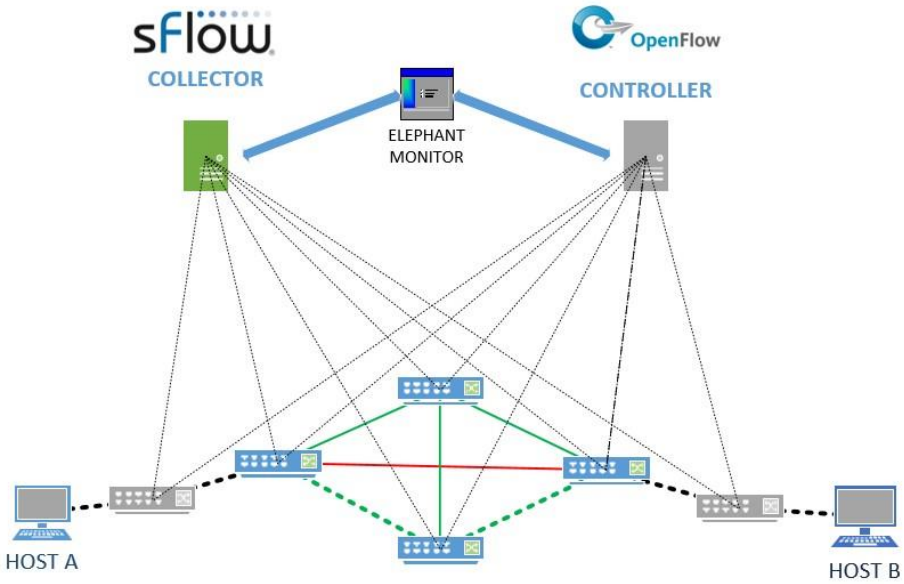

Figura 3-5. Elephant Rerouting
ELABORACIÓN PROPIA (2015)

3.6. Interoperabilidad y portabilidad

3.6.1. Conectividad hacia internet

La solución debe ser transparente para los equipos de red encargados de dar conectividad a internet, sin necesidad de realizar alguna modificación o configuración extra en dichos equipos.

3.6.2. Coexistencia con Redes legacy

La coexistencia de redes legacy y redes OpenFlow, introduce el concepto de islas OpenFlow. Una isla OpenFlow es un segmento de red, conformado por un grupo de Switches conectados a un mismo controlador. Debe analizarse diversos escenarios de interoperabilidad entre segmentos legacy e islas OpenFlow, con el fin de posibilitar una migración gradual hacia SDN.

- Coexistencia de una red legacy con una isla OpenFlow cuando la isla OpenFlow se encuentra como capa de acceso y agregación, y el segmento legacy como Core.

  Este escenario representa una posible forma de migración. En cada etapa de migración, los Switches de acceso y agregación van siendo reemplazados por Switches OpenFlow. A su vez, a medida que se reduce la cantidad de Switches legacy, se puede habilitar más enlaces entre los swicthes de distribución y agregar Switches de core. En este caso, todos los Switches OpenFlow deben estar conectados para formar una sola isla OpenFlow, y los hosts directamente conectados a Switches OpenFlow deben pertenecer a una sola subnet.

- Isla OpenFlow utilizada como Core de la red, mientras que el resto de la red es legacy.

  Este escenario representa una forma alternativa de migración. Se comienza por reemplazar el Switch de core central, y se reemplaza el resto de Switches gradualmente.

3.6.3. Portabilidad a otros controladores

Debido al carácter Open Source de las plataformas de controlador OpenFlow, estas son muy cambiantes. Conviene que el diseño del controlador pueda ser aplicado a cualquier plataforma y para esto se requiere que el diseño sea modular, y que la implementación se realice sobre una plataforma modular.

4. DISEÑO E INTEGRACIÓN DE LAS APLICACIONES

En el presente capítulo se detalla los factores determinantes para la elección de la plataforma base de controlador y el diseño del mismo, en base a los requerimientos planteados en el capítulo 3.

4.1. Elección de la plataforma base del controlador

La comparación relativa de plataformas de controlador es un tema de intensa discusión en la comunidad. A continuación se muestran las fuentes de información y los criterios para el criterio de la elección de la plataforma.

4.1.1. Estudios

Se han revisado diversos estudios de comparación de controladores. El primero de ellos es *On controller performance in software-defined networks* [31]. En este paper se realiza la comparación de los controladores NOX, Beacon, Maestro y una versión multi- threading de NOX, NOX-MT. La comparación se realiza usando una herramienta llamada *cbench*, la cual simula conexiones de N Switches con el controlador, y envía Packet In al controlador. Dispone de dos modos de prueba, *Latency*, en el cual se envía Packet In y se espera la respuesta del controlador antes de volver a enviar otro, y *Throughput*, en el cual se envían Packet In a la tasa más alta que el controlador puede soportar. Estos modos permiten hallar la mínima latencia y la máxima cantidad de Packet Ins que el controlador puede procesar, respectivamente. Se concluye que NOX- MT incrementa altamente la performance al implementar multi threading en la etapa de I/O.

El siguiente estudio revisado es [23], donde además de Cbench, se utiliza la herramienta HCPROBE para evaluar la confiabilidad y seguridad de los controladores. En este estudio, se concluye que Beacon es el controlador que soporta mayor *throughput*. Sin embargo las comparaciones de performance no son de utilidad en estos dos trabajos, pues los controladores evaluados trabajan en modo Learning Switch, funcionalidad que no es de utilidad en esta tesis.

Un tercer estudio analizado es [9], el cual no analiza *Performance*, si no que utiliza un método del tipo *Multi-Criteria Decision Making* llamado *Analytic Hierarchy Process*

(AHP), que permite comparar de forma cualitativa las propiedades de los controladores y en base a prioridades determinar el más óptimo. En este estudio, el mejor controlador fue Ryu, y en segundo lugar, Floodlight.

De forma adicional, se revisó un análisis que Sridhar Rao hizo para el sitio web "The New Stack". En este análisis, Rao compara las características en común de los controladores, y la aplicabilidad de cada controlador a casos de uso específicos.

En la tabla 4-1 se muestra un resumen de la comparación hecha por Rao, que sintetiza la utilidad de los controladores actuales para diferentes casos de uso.

Use-case	Trema	NOX / POX	RYU	Floodlight	ODL	ONOS
Network Virtualization by Virtual Overlays	YES	YES	YES	PARTIAL	YES	NO
Hop-by-hop Network Virtualization	NO	NO	NO	YES	YES	YES
OpenStack Neutron Support	NO	NO	YES	YES	YES	NO
Legacy network Interoperability	NO	NO	NO	NO	YES	PARTIAL
Service Insertion and Chaining	NO	NO	PARTIAL	NO	YES	PARTIAL
Network Monitoring	PARTIAL	PARTIAL	YES	YES	YES	YES
Policy Enforcement	NO	NO	NO	PARTIAL	YES	PARTIAL
Load Balancing	NO	NO	NO	NO	YES	NO
Traffic Engineering	PARTIAL	PARTIAL	PARTIAL	PARTIAL	YES	PARTIAL
Dynamic Network Taps	NO	NO	YES	YES	YES	NO
Multi-layer Network Optimization	NO	NO	NO	NO	PARTIAL	PARTIAL
Transport Networks - NV, Traffic-Rerouting, Interconnecting DCs, etc	NO	NO	PARTIAL	NO	PARTIAL	PARTIAL
Campus Networks	PARTIAL	PARTIAL	PARTIAL	PARTIAL	PARTIAL	NO
Rerouting	YES	NO	YES	YES	YES	YES

Tabla 4-1. Comparación de controladores según casos de uso
COMPARISON OF OPEN SOURCE SDN CONTROLLERS (2015) [30]

4.1.2. Criterios para elección de la plataforma base

4.1.2.1. Lenguaje de programación

En la elección del lenguaje de programación, fue determinante el requerimiento de Rapid Prototyping y de Performance. Se prefirió un lenguaje como Java ya que, en general, posee una performance mayor que lenguajes como Python o Ruby. Además, la abstracción disponible en un lenguaje orientado a objetos como Java proporciona ventajas para el Rapid Prototyping de los módulos a implementar.

Para la implementación de aplicaciones REST, se prefiere el uso de Python, por la rapidez de desarrollo que permite para este tipo de aplicaciones.

4.1.2.2. Soporte

Muchas de las primeras plataformas de controladores tienen un bajo soporte y documentación accesible desde Internet, por ejemplo, NOX, POX, Trema, Maestro, Beacon. Los proyectos de mayor envergadura y más recientes, como Floodlight, OpenDaylight, Onos y Ryu superan a sus predecesores en este aspecto. Véase, por ejemplo, el mailing list de Floodlight, de OpenDaylight o Ryu, y las páginas de soporte oficiales [4].

4.1.2.3. Curva de aprendizaje

A pesar que la mayoría de controladores mantiene una estructura similar, algunos de estos utilizan frameworks de programación, que si bien otorgan características destacables, acentúan la curva de aprendizaje. Tal era el caso de los frameworks Karaf y OSGI, utilizados en Opendaylight para la carga de módulos en runtime; Maven para la automatización del Build; o la utilización de Yang como lenguaje de modelamiento de datos.

Estas herramientas, no son requisitos para la implementación de la prueba de concepto de nuestro diseño.

4.1.2.4. Complejidad

Debido a que algunos de los requerimientos de diseño presentan cierta complejidad, la plataforma debe otorgar librerías y módulos que permitan realizarlos. Por ejemplo, Floodlight y OpenDaylight tienen mecanismos de manejo de Hosts y manejo de la topología bastante flexibles, implementados sobre una librería OpenFlow de desarrollo independiente, y un manejo de eventos en base a *Java Listeners*.

4.2.1. Selección de la plataforma

En la tabla 2-2, se puede visualizar un resumen de las características evaluadas en los controladores, a través de los estudios analizados y los criterios mencionados.

Dada la evaluación, se determinó que Floodlight es la plataforma que satisface con mayor amplitud los requerimientos, y se ha implementado la prueba de concepto sobre esta plataforma.

4.3. High level design del controlador

Para un estudio más detallado de Floodlight, y de los componentes básicos que son utilizados en nuestro controlador, refiérase el ANEXO C. En el resto del capítulo se explica el diseño en software del controlador, que representa la contribución de este trabajo.

A continuación, se describirá el High Level Design del controlador, detallando los módulos y sus interacciones.

4.3.1. Diagrama de bloques del controlador

En la imagen 4-1 se observa el diagrama de módulos de nuestro controlador y las interacciones entre estos. Refiérase a la imagen 2-10 del diagrama de bloques de Floodlight para observar y comparar la ubicación de los módulos a implementar dentro de la arquitectura del controlador, así como los módulos que serán removidos o no son considerados en esta implementación. Las flechas negras indican las llamadas de las APIs Java expuestas en los *Service* de cada módulo.

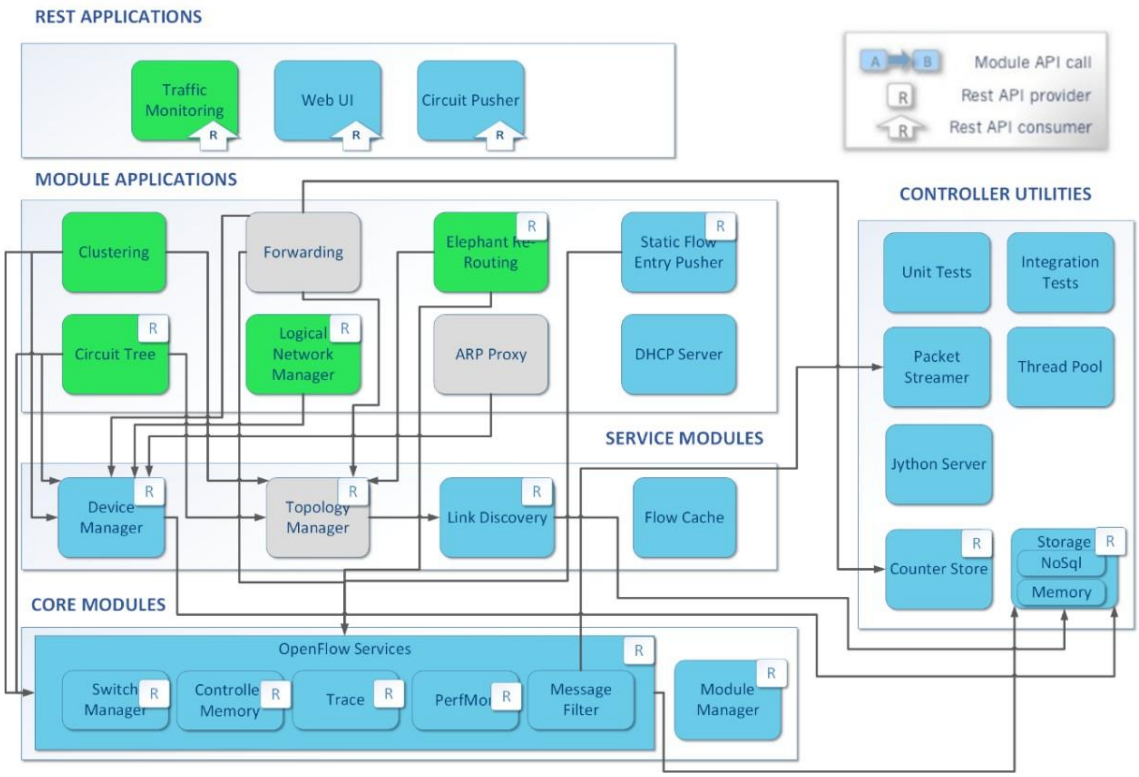

Figura 4-1. Diagrama de módulos del controlador implementado
ELABORACIÓN PROPIA (2015)

A continuación, se describe de forma general el diseño de los módulos a implementar.

4.3.2. Módulos de segmentación lógica

Para lograr la segmentación lógica especificada en la sección 3.3.2 con todos sus requerimientos, se implementará el módulo Logical Network Manager y su REST API.

4.3.3. Módulos de manejo de broadcast

Como se explicó en la sección 3.5.1, se requiere utilizar los mecanismos ARP Proxy, DHCP Handler y Dominios de interés. El mecanismo *ARP Proxy* será implementado en un Módulo de aplicación. El mecanismo *DHCP Handler* está implementado en la versión oficial de Floodlight como *DHCP server*, teniendo la funcionalidad de Handler y de DHCP Server. El mecanismo Dominio de interés estará implementado como parte del módulo *Forwarding*.

4.3.4. Módulos de manejo de tráfico unicast

Como se señaló en la sección 3.5.3, se requiere implementar un mecanismo que realice Shortest Path Forwarding, utilizando dos niveles de enrutamiento. Esto se hará como un módulo de Aplicación llamado Clustering.

Sobre el módulo Clustering, funcionará un mecanismo de reglas proactivas. Este será implementado como un módulo de aplicación llamado *Circuit Tree*.

De forma general, se definen las siguientes características del módulo *Circuit Tree*:

- Aplicación proactiva:

  No se analizan Packet In.

- Flows de baja prioridad:

  Debido a que son flows con Match muy generales, se desea tener la posibilidad de utilizar otras reglas más específicas por medio de otras aplicaciones.

Asimismo, se requería implementar un mecanismo que realice el re-enrutamiento de

los Flows elefantes. Este será implementado mediante la modificación de la clase *TopologyInstance*, la adición de la clase *CustomCostBalancer* creado por Luca Prete, Simone Visconti, Andrea Biancini, Fabio Farina [12], y la creación de la clase principal del módulo de aplicación llamado *Elephant Rerouting*

Este módulo tiene como objetivo re-enrutar aquellos flows que se hayan establecido por medio de *Forwarding*, Clustering o de *Circuit Tree* y que cumplan la característica de ser elefantes. Esta última característica será detectada mediante una aplicación en python, llamado *Elephant Detector*, que utilizara la REST API provista por *CustomCostBalancer* y *Elephant Rerouting*.

En la presente sección, se detallará el diseño e implementación de los módulos y su integración en la plataforma Floodlight. El código fuente en Java del controlador completo se puede obtener y descargar en la web en la siguiente URL:

https://github.com/telestack/Floodlight.

4.4. Diseño e integración de módulos de segmentación lógica

4.4.1. Logical Network Manager

Creación y mantenimiento de las redes lógicas.

Se define una REST API para crear y eliminar una Red lógica, especificando un nombre y un ID. Este último sirve de valor Cookie ID para las *Flow Entries* que serán instaladas por este módulo. Al crearse una Red Lógica, se crea una tabla *Net* donde se guardarán los hosts que pertenezcan a dicha red lógica; esta será la tabla de consulta para las aplicaciones que requieran utilizar las redes lógicas.

Para añadir hosts a una red lógica, deben crearse reglas que permitan la gestión automática y flexible de dichas redes lógicas. La creación de las reglas se realiza mediante una llamada de la REST API.

Las reglas deben tener la siguiente estructura:

Campo	Descripción
Tipo	Dirección MAC, dirección IP o punto de conexión
Valor	De acuerdo al tipo
ID Red lógica	Del 2 al 1024 (pueden ser múltiples valores)

Figura 4-2. Estructura de una regla de Red Lógica
ELABORACIÓN PROPIA (2015)

Cuando se añade una regla, esta se guarda en una tabla de reglas relacionadas al tipo de atributo. Luego, se busca en la lista de dispositivos los dispositivos que tenga el parámetro que tenga especificado en dicha regla. Si se encuentra alguna coincidencia, se extrae la dirección MAC de dicho dispositivo y esta se guarda en la tabla Net asociada al ID de la red lógica especificada en la regla.

Como se puede apreciar, no hay ninguna restricción a nivel lógico para que un mismo host pueda pertenecer a varias redes lógicas, lo cual otorga una gran flexibilidad.

Para que ante el cambio de alguno de los parámetros de red, un host pueda cambiar o mantener una o más reglas aplicadas, el módulo implementa *Device Listeners*, en los cuales se gestionan estos eventos.

Cuando un dispositivo se añade (nuevo u olvidado por el controlador), se realiza la búsqueda de una regla que coincida con alguna característica del dispositivo. De encontrarse, la MAC del dispositivo se añade a la tabla Net.

Cuando un dispositivo cambia de dirección IP, se remueve la MAC de dicho dispositivo de la tabla Net. Luego, se procede a realizar la búsqueda en la tabla de reglas de direcciones IP, una regla que coincida con la nueva dirección IP.

Cuando un dispositivo cambia de punto de conexión, se remueve la MAC de dicho dispositivo de la tabla Net. Luego, se procede a realizar la búsqueda en la Tabla de reglas de puntos de conexión una regla que coincida con el nuevo punto de conexión.

Es de esperar que el parámetro especificado en la regla no se encuentre en ningún dispositivo activo en la red, por lo que la regla será guardada en la tabla de reglas respectiva, pero ningún dispositivo se asociará en la tabla Net. Sin embargo, es necesario que cuando aparezca un dispositivo con las características señaladas, este

se agregue automáticamente a la tabla Net correspondiente.

En cada Switch donde existe al menos un nodo asociado a una red lógica, se crea una Flow Entry que funcione como Table Miss. Esta especifica como match únicamente a un puerto de entrada, con la instrucción enviarlo al controlador encapsulado en un Packet In, con un cookie ID de la Red Lógica asociada al host que está conectado a dicho puerto. El controlador usa el cookie ID para determinar la Red Lógica de origen, sin necesidad de buscarla, y realiza únicamente la búsqueda de la Red Lógica asociada a la dirección MAC destino.

Si el tráfico entre las Redes Lógicas origen y destino está habilitado, el módulo setea la Acción a realizar como Forward y el paquete continúa en el Pipeline, para ser procesado por Clustering o *Forwarding* y de esta manera se establece del flujo. De lo contrario la Acción se setea en *drop*, y los módulos que continúen el procesamiento del paquete setean las reglas respectivas para descartar los paquetes de dicho flujo.

4.4. Diseño e integración de módulos Baseline 4.4.1.Clustering

El módulo Clustering se implementa como una clase que extiende de *Forwarding* Base. Se implementa un arreglo de datos *clusterDatabase* usando la clase *Java Map*, para alojar el mapeo de cada Switch destino con cada posible Switch origen, su Label ID y el puerto de salida, usando la siguiente estructura:

Switch Destino	Label ID	Switch Origen	Output Port
A	46	B	3
		C	5
		D	15
B	63	A	6
		C	9
		D	12

Figura 4-3. ClusterDatabase
ELABORACIÓN PROPIA (2015)

El módulo procesa los eventos Packet In. Cuando se dispara algún evento, se valida en primer lugar que el *Ethertype* del paquete sea IPv4 (o IPv6). Luego, se obtiene el punto de conexión (Switch y puerto) de los host origen y destino. A continuación, se consulta en el *clusterDatabase* si existe una ruta entre ambos Switches. De cumplirse

lo anterior, se procede únicamente a crear las reglas correspondientes en los Switches de acceso, utilizando la información del host destino mediante el *Device Service*.

De no existir la ruta del core en el mapeo, se procede a crearla. Para esto, se obtiene la ruta del *Topology Service*, y se instala en cada uno de los Switches que conforman la ruta reglas basadas solo en la etiqueta VLAN asociada al Switch destino, tanto en los caminos de ida como de vuelta.

Podemos deducir que, en un determinado instante, en un Switch, si se tienen D destinos en uso, se tendrán D reglas ocupadas. De forma permanente, si se tienen 48 hosts por Switch de acceso, se tendrá 48 reglas ocupadas por cada host, más 1 para el VLAN ID local, en total, 49 reglas. Para los Switches de core, si se tienen N Switches de acceso, se tendrá como máximo N entradas ocupadas permanentemente. Analizaremos la escalabilidad de este mecanismo en el capítulo 5.

4.4.2. Forwarding

Si por algún motivo excepcional, el paquete no puede ser enrutado por el módulo Clustering, el pipeline de procesamiento del Packet In llamará al módulo *Forwarding,* para lo cual se realiza las modificaciones necesarias en la implementación de otros módulos involucrados.

Forwarding encuentra la ruta entre los dos Switches, y realiza la instalación de reglas en cada uno de ellos. Estas reglas son específicas en dirección MAC e IP, origen y destino y tienen un timeout de 5 segundos. Para mayores referencias, consultar con la documentación oficial de Floodlight.

4.4.3. Circuit Tree

El módulo *Circuit Tree* requiere que se registre la dirección MAC del Host que será Circuit Root. Para esto, se crean REST API calls con la dirección MAC del host como parámetro de entrada.

Para cada *Circuit Root* registrado, se obtiene las Rutas entre el Switch de Acceso del *Circuit Root* y el resto de Switches de acceso de la red (es decir, Switches que tienen

al menos un host conectados). A este conjunto de rutas se le conoce como árbol de circuitos (o *Circuit Tree*).

En base a las rutas obtenidas, en los Switch número 2 hasta el Switch n-1 (siendo los Switches 1 y n los Switches de acceso) se colocan reglas cuyo match sea para la parte de cluster ID de la Opaque Label del DST MAC. Para esto se utiliza una máscara en la dirección MAC destino.

El valor de Priority asignado a las *Flow Entries* de este módulo debe ser mayor que los correspondientes a *Forwarding*, y menores que los correspondientes a Clustering.

De forma similar a Clustering, podemos deducir el número de entradas ocupadas en los Switches. Si se tienen 'R*' Circuit Roots* registrados en el módulo distribuidos en 'N' Switches, en cada Switch de acceso se tendrán 'N+1' reglas ocupadas en cada Switch de acceso, más las 48 reglas de mapeo inverso para cada host.

4.5. Análisis e Integración de módulos de manejo de Broadcast

4.5.1. Módulo DHCP

Él módulo DHCP procesa eventos de PACKET INs. En primer lugar, se valida que el *Ethertype* sea IPv4, el protocolo de transporte sea UDP, y que los puertos sean los puertos estándar de DHCP para servidor y cliente. Luego, el flujo se ramifica de acuerdo a qué tipo de mensaje DHCP es, y en base a esto, realiza la función de servidor DHCP, otorgando los *leases*. Todos los mensajes DHCP que el servidor/controlador envía a los hosts son enviados encapsulados en un Packet Out y el Switch los envía por el puerto especificado, evitando el flooding.

Este módulo fue implementado y es mantenido por personal de Big Switch Networks. Para obtener los detalles de funcionamiento, se puede consultar la documentación oficial.

4.5.2. Módulo ARP

El módulo ARP Proxy procesa los eventos de Packet In.

Cuando se dispara un evento, se verifica que el Ethertype del paquete corresponda

a ARP. Si no es así, el Pipeline continua a los siguientes módulos. De ser cierto, se determina en base al código de operación si es un ARP Request o Reply, para ser analizado por el método *Handle ARP Request* o *Handle ARP Reply*, respectivamente. El método ARP Request, realiza en primer lugar una búsqueda en una cachè de ARP Requests. Si existe un ARP Request previo, se resetea el contador de dicha solicitud, y se detiene el Pipeline, pues aún no se conoce el host destino. De lo contrario, se realiza una consulta en la lista de dispositivos de la dirección IP. Si se encuentra alguna coincidencia y la dirección IP asociada es válida, se construye el ARP Reply y este es enviado encapsulado en un Packet Out al Switch de acceso del Host destino, para ser enviado por el puerto correspondiente. De lo contrario, el ARP Request se guarda en la cache de ARP Requests, se encapsula en un packet out, y se envía a todos los Switches. Sin embargo, se envía una sola copia a cada Switch, y este envía el ARP Request por todos sus puertos, utilizando el puerto lógico ALL.

Cuando se captura un ARP Reply, se realiza la búsqueda de todos los ARP Request relacionados que estén en la caché, es decir, que no hayan expirado. Para todos aquellos ARP Request que no hayan expirado, se construyen los ARP Reply, se elimina la entrada de la caché correspondiente, y se envía encapsulado en Packet Out. A continuación, se muestra el diagrama de flujo en la figura 4-4.

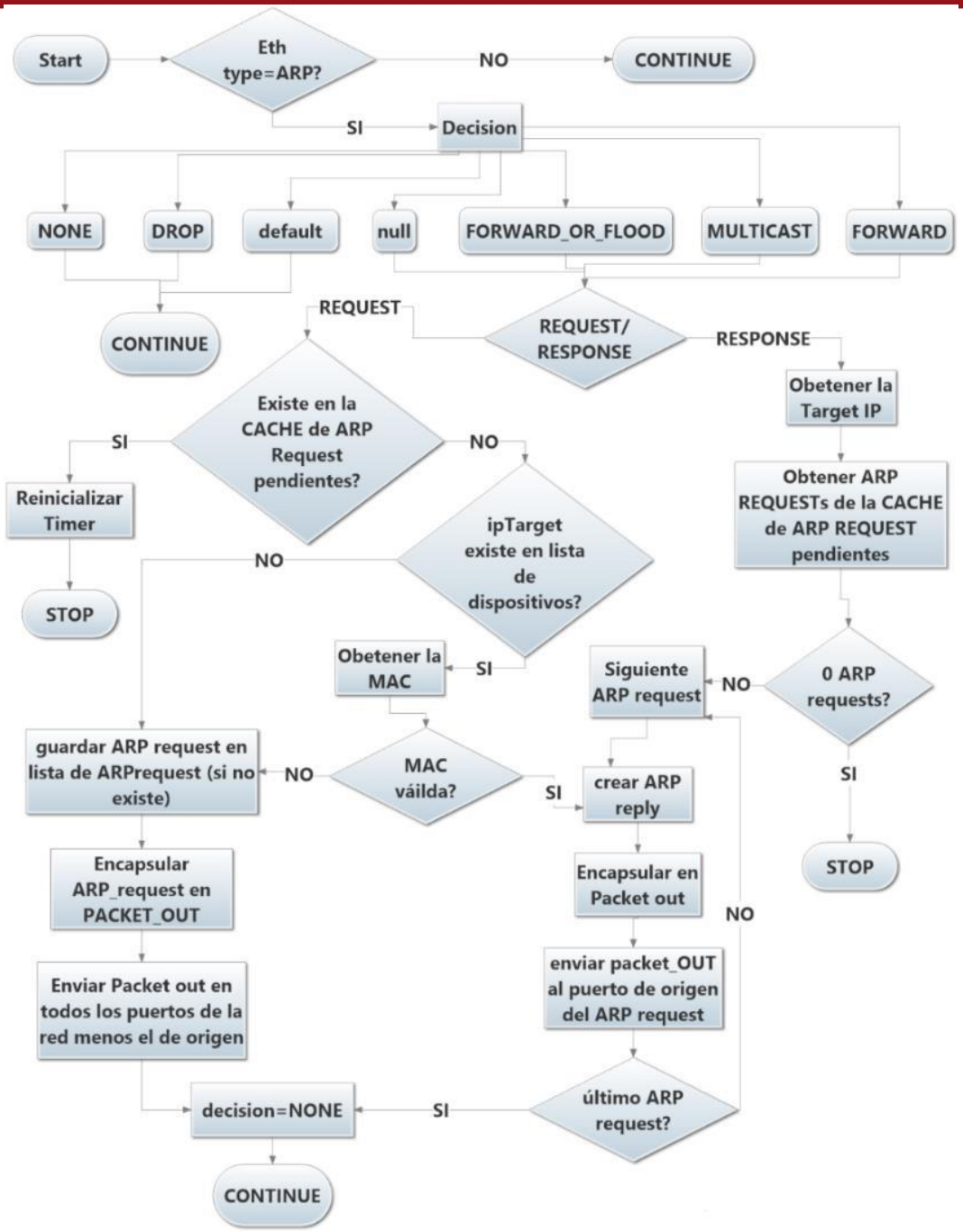

Figura 4-4. Diagrama de flujos de ARPProxy
ELABORACIÓN PROPIA (2015)

En este trabajo, se ha tomado como base la implementación de un ARP Proxy para Floodlight versión 0.9, hecho por Michael Bredel, cuyo código ha sido portado a la versión 1.1 de Floodlight, y se han corregido y adaptado a nuestros requerimientos.

4.5.3. Dominios de interés

El módulo Dominios de interés es del tipo proactivo y utiliza las REST APIs provistas

por el módulo de Logical Network Manager para la creación de los dominios y el tracking de sus respectivos hosts. Este módulo se encarga de construir, para cada domino de interés, un árbol para la distribución de las tramas broadcast. Idealmente, se construirá el árbol que minimice el número de transmisiones (número de enlaces en el árbol). Este problema, conocido como "Minimum Steiner Tree", es NP-Complete, por lo que se decidió implementar una heurística que provea un resultado eficiente en tiempo corto. Esta heurística se detalla mediante el siguiente ejemplo.

Considerando la red mostrada en la figura 4-5, la red tiene una comunidad de interés formada por las computadoras de color verde. Al inicio, no hay un árbol de rutas por lo que el tráfico broadcast no puede ser enviado a la comunidad de interés.

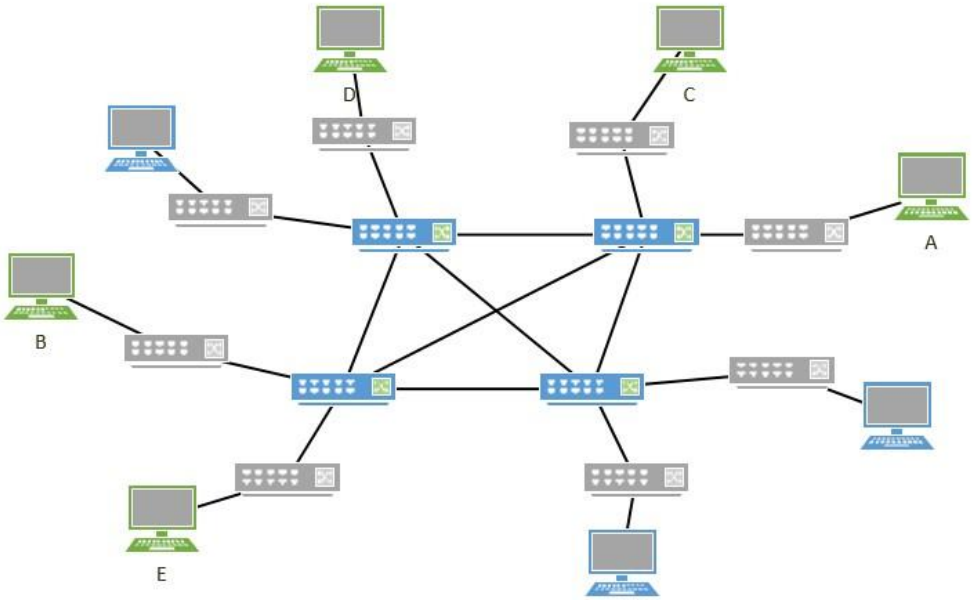

Figura 4-5. Dominios de interés. Topología base
ELABORACIÓN PROPIA (2015)

- Paso 1: Elegir 2 hosts miembros del grupo aleatoriamente. Encontrar la ruta entre ambos, utilizando el método *GetRoute* de Topology Manager. Esta será el tronco del árbol. En la figura 4-6, se observa que se eligió los nodos A y B y se estableció el tronco del árbol (de color azul) entre los switches de acceso respectivos.

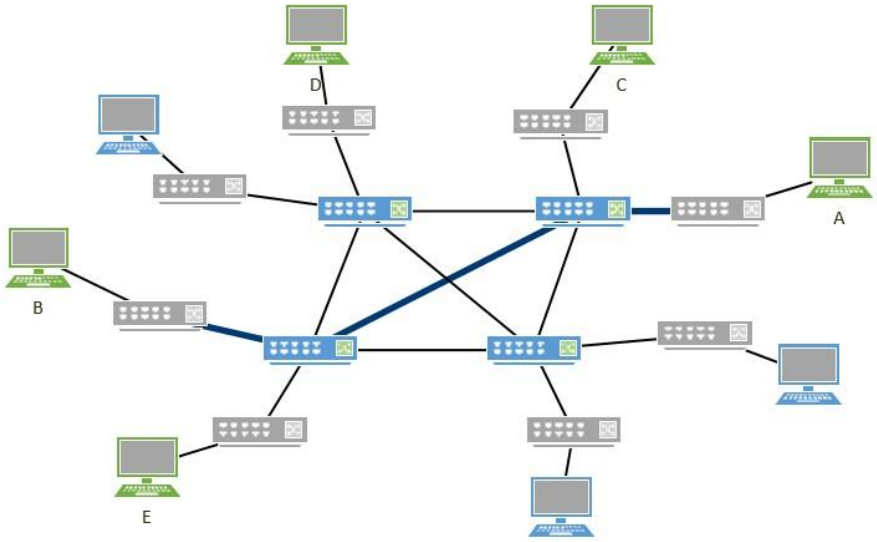

Figura 4-6. Dominios de interés. Paso 1

ELABORACIÓN PROPIA (2015)

- Paso 2: Para un host en el grupo que aún no haya sido cubierto, encontrar una ruta entre este y algún switch del tronco del árbol. Esto se realiza creando un método en Topology Instance que implemente el algoritmo de Dijkstra, modificando la condición de término de "es igual al destino" por "es igual a cualquier nodo en el tronco arbol". En las figuras 4-7 y 4-8 se puede observar que se crearon ramas desde los switches de acceso de los nodos C y D hacia el tronco (colores verde y rojo, respectivamente).

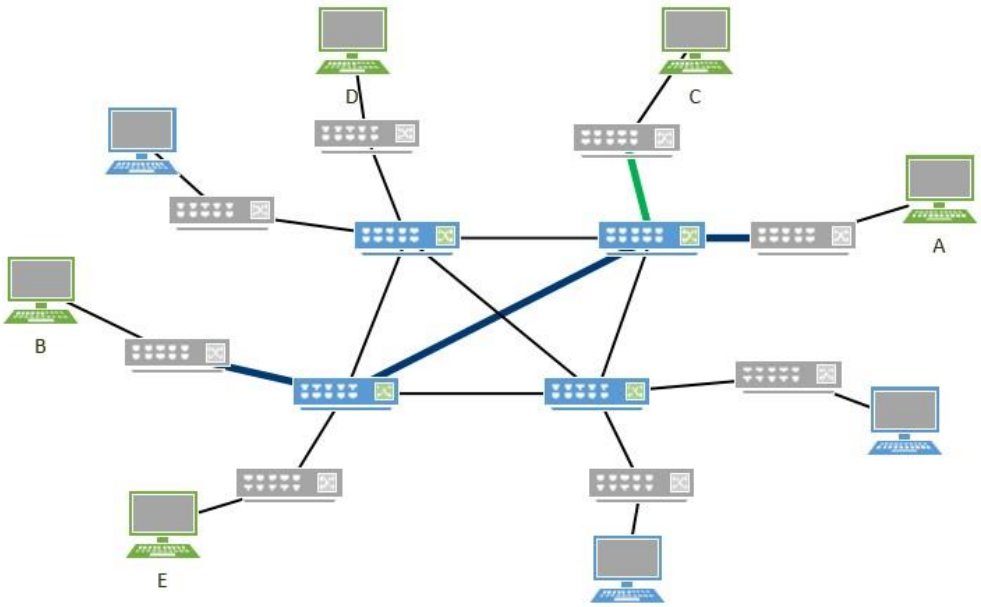

Figura 4-7. Dominios de interés. Paso 2

ELABORACIÓN PROPIA (2015)

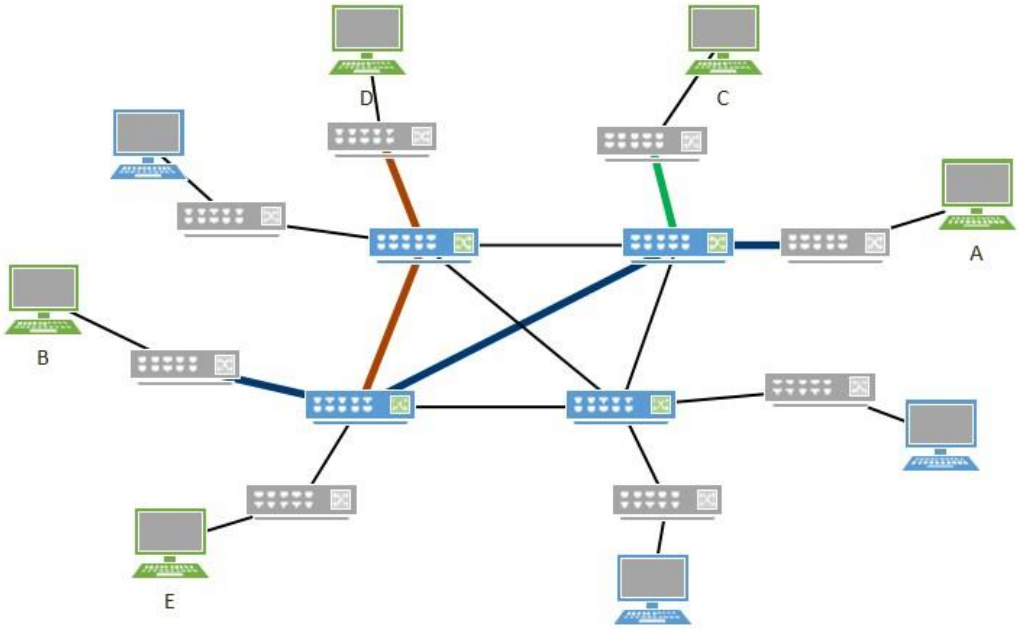

Figura 4-8. Dominios de interés. Paso 2
ELABORACIÓN PROPIA (2015)

- Paso 3: Si quedan hosts sin cubrir, repetir el paso 2. Si todos los hosts han sido cubiertos, el árbol multicast está completo. En la figura 4-9 se observa que con la rama de color marrón, el nodo E queda conectado al árbol y se completa el árbol multicast.

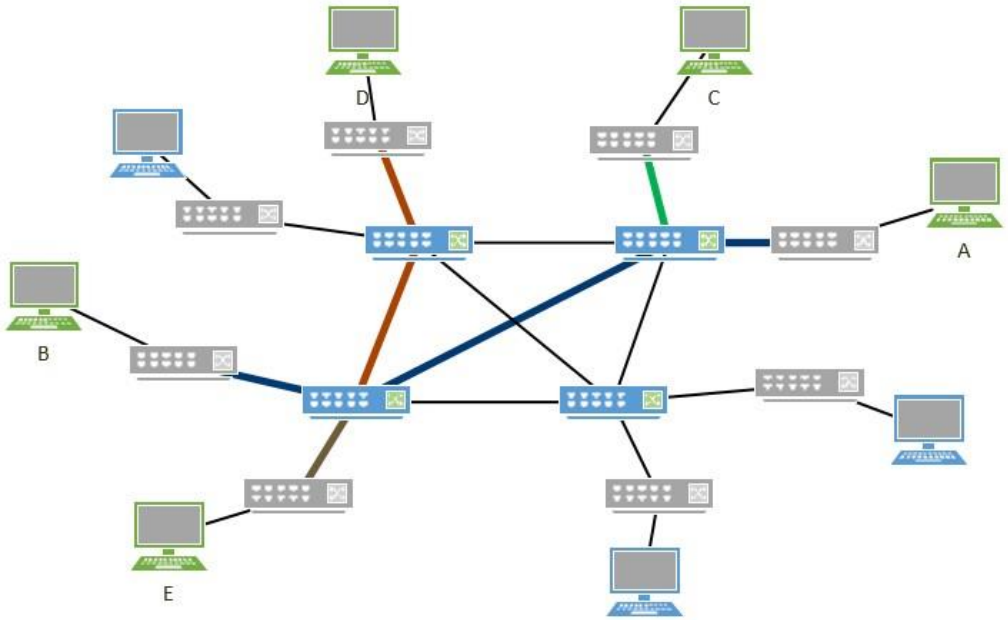

Figura 4-9. Dominios de interés. Paso 3
ELABORACIÓN PROPIA (2015)

En los switches de acceso se crean los siguientes Flow Entries:

- Flow Entries de entrada cuyo Match es igual a cada puerto asociado a la comunidad de interés. Estos Entries setean el Vlan Tag de la comunidad, insertan una cabecera MAC, escriben en el campo Source MAC Address la dirección MAC del switch de acceso, y apuntan a un Flow Group de entrada con el ID del dominio de interés, cuyos Buckets envían los paquetes por todos los puertos adyacentes al árbol multicast.

- Se crean Flow Entries de salida cuyo Match es el Vlan tag de la comunidad de interés. Estos Flow Entries remueven la cabecera MAC externa y apuntan a un Flow Group de salida con el ID del dominio de interés, cuyos Buckets envían los paquetes por todos los puertos asociados a dicha comunidad de interés.

En los switches de core que pertenecen al árbol multicast, para evitar que los paquetes regresen al mismo switch y se creen loops, se crea una Flow Entry que descarta los paquetes con el Destination MAC Address igual a la dirección MAC del mismo Switch.

Los paquetes que no son descartados, son analizados por Flow Entries que tienen como match el Vlan Tag de la comunidad. Estos Flow Entries modifican la cabecera MAC externa del paquete, colocando en la dirección Destination MAC Address la dirección MAC del switch anterior o “last hop”, para prevenir posibles loops. Estos Flow Entries apuntan al Flow Group asociado a la comunidad de interés, cuyos Buckets envían los paquetes por todos los puertos asociados a dicha comunidad de interés.

4.6. Diseño e integración de módulos de Ingeniería de tráfico

4.6.1. Elephant Re-Routing

El modulo *Elephant Re-Routing* tiene una clase principal llamada *Elephant Rerouting*, que provee los métodos para re enrutar una flujo establecido.

Para realizar el cálculo y asignación de rutas con costos dinámicos, se implementa una clase *Topology Instance CCBalancer* que extienda la clase padre *Topology Instance*. De esta forma, se aprovecha el acceso a los estados y métodos de la clase

padre, y se añaden únicamente los métodos adicionales. Entre las estructuras de datos que se necesitan añadir son:

- Un Mapeo de *Broadcast Trees*, usando Java *HashMap*: *ElephantRootedTrees*.

- Una Cache de rutas, usando la *Cache* de la librería *Guava* de Google: *ElephantRouteCache*.

Adicionalmente, se requieren añadir los métodos necesarios a la clase *Topology Manager*.

Para conseguir un equilibrio entre costo de cómputo y tiempo de respuesta, cada 5 segundos se procede a:

- Recalcular los Broadcast Trees más comunes, cuyos *Root* son determinados externamente mediante la REST API.

- Recalcular la caché de rutas, lo cual consiste en hacer únicamente un *Backtracking* a los *Broadcast Trees* correspondientes.

El método *GetElephantRoute* devuelve la ruta entre dos Switches. Para esto, determina si alguno de los dos nodos tiene una entrada en el hash de *DestinationRootedTrees*. Solo en ese caso se llamará a la cache *ElephantRootedTrees*. De lo contrario, se llama al método *NotVeryImportantElephant,* el cual recalcula el *Broadcast Tree* y hace el *Backtracking* correspondiente, para construir la ruta.

Bajo este esquema, la búsqueda de una ruta para un elefante puede dar los siguientes resultados.

- Mejor caso:

  Existe una ruta para ambos nodos en la cache de rutas. Se establece la ruta.

- Segundo caso:

No existe una ruta para ambos nodos en la cache de rutas, sin embargo, uno de los nodos tiene un *Broadcast Tree* actualizado. Se hace el *Backtracking*, y se coloca la ruta en la caché. Se establece la ruta.

- Peor caso:

No existe una ruta para ambos nodos en la cache de rutas, ni en el hash de *Broadcast Tree*. Es necesario calcular el *Broadcast Tree*, luego calcular la ruta y por último establecerla.

La clase *ElephantRerouting* tendrá los métodos necesarios para encontrar la ruta e insertará los *Flow Entries* necesarios entre dos hosts de la red. Para esto, se implementará la REST API correspondiente.

El módulo inserta las reglas granulares en toda la ruta. Estas reglas tienen las siguientes características

MATCH	ACTION	PRIORIDAD	TIMEOUT	FLAGS
SRC_MAC	FORWARD_TO_PORT X	10000	5 SEGUNDOS	FLOW_REMOVED
DST_MAC				
SRC_IP				
DST_IP				
SRC_PORT				
DST_PORT				

Figura 4-10. Flow Entry para Elephant Rerouting
ELABORACIÓN PROPIA (2015)

4.6.1.1. Mecanismo de restauración de rutas

Ante la caída de un link o de un puerto, el Switch enviará un mensaje *Port Status* al controlador, y el *FloodlightProvider* generará el evento correspondiente.

Nuestro módulo implementa el *listener* para recibir estos eventos. Cuando se produce alguno, se realiza una búsqueda del puerto y Switch involucrado en la ruta de cada Elephant que no ha expirado. Si se encuentra alguna coincidencia, inmediatamente se procede a reasignar la ruta.

4.6.2. Elephant detector

El *Elephant Detector* es una Aplicación REST escrita en Python, que utiliza la REST API del colector sFlow y las API REST del controlador. Cada 5 segundos, realiza llamadas al colector, y extrae el Throughput de cada enlace de la red. Al mismo tiempo recolecta eventos de congestión en los Switches de acceso. El módulo envía los parámetros del Flow que está causando la saturación al módulo de elefantes en el controlador.

El colector sFlow se configura para que detecte eventos llamados *Elephant Event*, en cada Switch de acceso. El *Elephant Event* se produce cuando el Throughput en un enlace es mayor al 10% de la capacidad de dicho enlace.

Cuando el *Elephant Detector* recibe el evento, crea un objeto JSON *Elephant*, definido con los parámetros dirección MAC origen, dirección MAC destino, dirección IP origen, dirección IP origen, puerto destino y puerto origen, valores tomados del paquete de muestra (*Sample Packet*) que el colector envía a la aplicación. Luego, realiza una llamada REST API al módulo *Elephant Rerouting* del controlador, para realizar el re enrutamiento de este *Elephant Flow*.

4.6.3. Módulo Custom Cost

Para obtener las métricas de estado de los enlaces que se utilizarán para calcular rutas con costos dinámicos, se utiliza el módulo Custom Costs Balancer, creado por Luca Prete, Simone Visconti, Andrea Biancini y Fabio Farina. Este módulo provee una REST API para que el módulo *Elephant Detector* pueda enviarle la ocupación de los enlaces con la frecuencia requerida, 5 segundos.

La clase *CCBalancer* utiliza esta información y la almacena en un mapeo link-costo, así como la mantiene actualizada ante cambios en la topología, mediante la implementación del *Topolgy Listener*. Esta información puede ser obtenida por *Topolgy Instance* para calcular las rutas, mediante el *Custom Cost Balancer Service*, y se informa de eventos de actualización de costo a través del *Custom Cost Balancer Listener*.

5. PRUEBAS Y ANÁLISIS

En el presente capítulo, se presentan las distintas pruebas de concepto utilizadas para poder determinar la funcionalidad de los módulos más importantes del controlador diseñado; así como también el modelo analítico, y el análisis respectivo, utilizado para poder medir la escalabilidad del mismo.

5.1. Pruebas de concepto

5.1.1. Pruebas en simulador

El escenario de pruebas simulado estuvo compuesto de 500 hosts conectados a la siguiente topología de red:

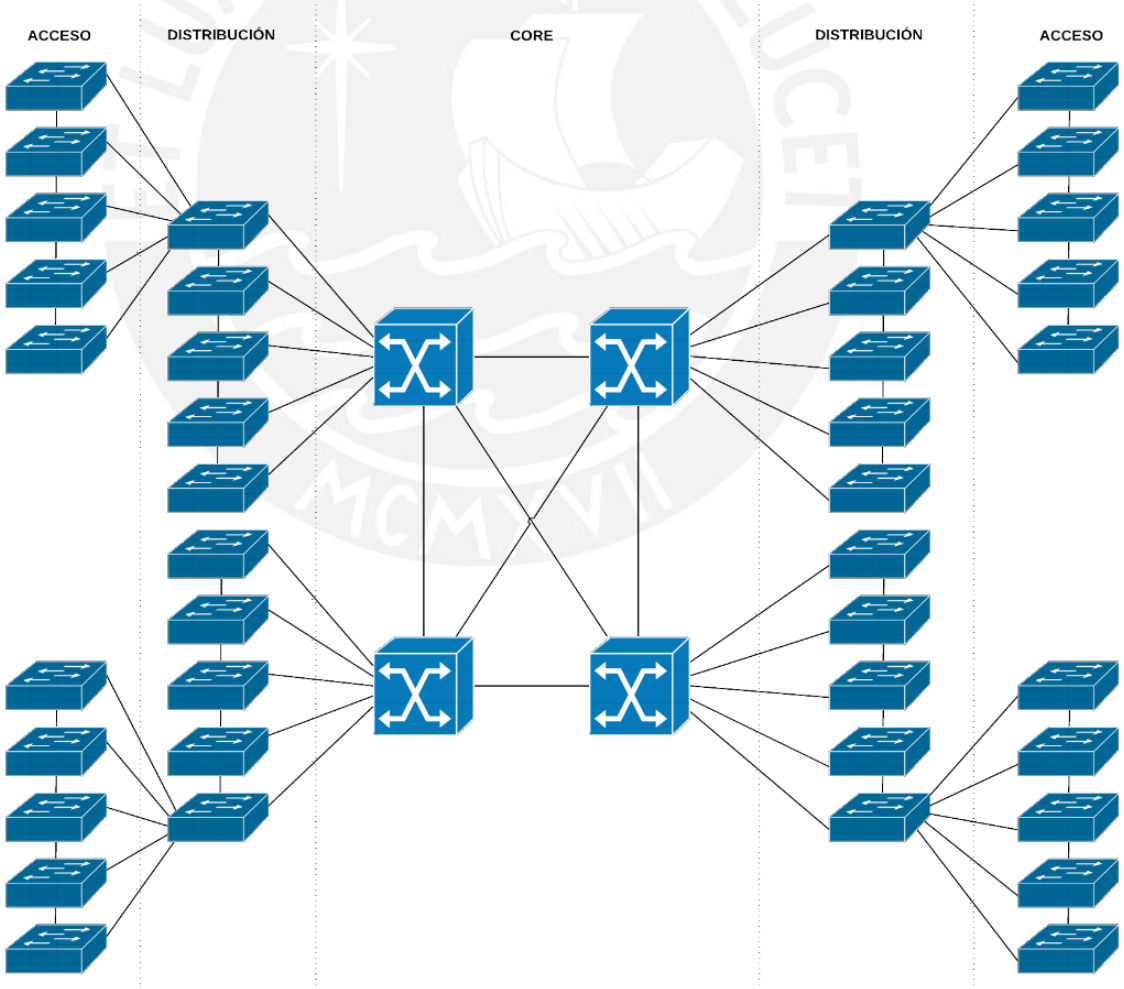

Figura 5-1. Topología de prueba en Mininet

ELABORACIÓN PROPIA (2015)

La cual tiene una estructura jerarquizada, que consta de:

- 4 Switches de CORE, conectados en full mesh
- 20 Switches de DISTRIBUCIÓN, 5 por debajo de cada Switch de CORE
- 100 Switches de ACCESO, 5 por debajo de cada Switch de DISTRIBUCIÓN
- 500 hosts, 5 conectados a cada Switch de ACCESO

Adicionalmente, los Switches de ACCESO, y de DISTRIBUCIÓN, se encuentran conectados en serie con sus vecinos (Switches pertenecientes al mismo grupo de ACCESO / DISTRIBUCIÓN).

El escenario de prueba fue simulado en una de las Workstation del laboratorio del GIRA- PUCP, utilizando el simulador de redes virtuales Mininet y se realizaron las siguientes pruebas independientes:

5.1.1.1. Broadcast (ARP y DHCP) y Clustering

Para esta prueba se tomó capturas de tráfico en la red utilizando la herramienta Wireshark en uno de los Switches de core, mientras se realizaba un flooding de paquetes ICMP a través de toda la red (utilizando la herramienta ping).

La primera captura se realizó sin utilizar los módulos de ARPProxy y DHCPHandler, por lo que se producen muchos paquetes de ARP, los cuales representan el 97% de paquetes capturados.

La segunda captura se realizó luego de habilitar los módulos ya mencionados (ARPProxy y DHCPHandler), obteniéndose un 10%.

Como se puede ver, ya no se tiene paquetes de ARP, pues estos son encapsulados en el Switch de ACCESO y son enviados al controlador como Packet In, para que este sea quien resuelva las consultas, eliminando así por completo el flooding de paquetes de ARP en la red (los cuales representan a la mayor cantidad de paquetes de broadcast que existen en las redes).

Para la prueba del módulo Clustering, se confirmó que existe conectividad a nivel ICMP entre los hosts. Asimismo, se observó que las reglas instaladas corresponden a las instaladas por el módulo Clustering, descrito en el capítulo 4.

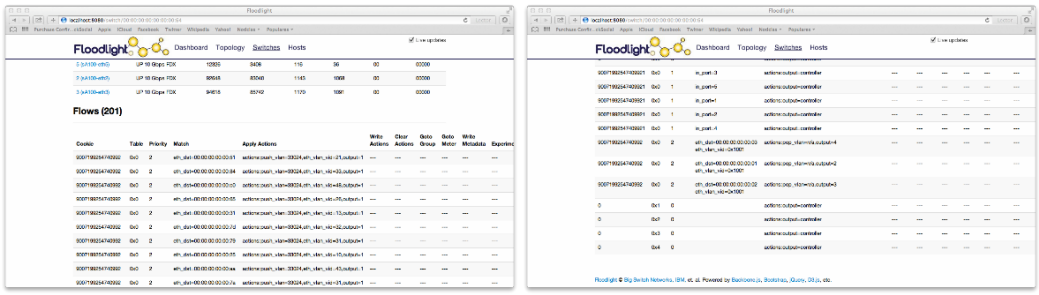

Figura 5-2. Reglas activas en el Switch de acceso utilizando el Clustering

Finalmente, se armó un *Circuit Tree* para uno de los hosts (VM 1). Se realizó una prueba ping desde diferentes hosts hacia la VM 1, con resultados exitosos, y se observó que las reglas corresponden al módulo Circuit Tree:

5.1.2. Pruebas de funcionalidad en Hardware Real

El escenario de pruebas físico estuvo compuesto por 1 Switch Pica8 3922 (Switch de CORE) y 3 Switches Pica8 3297 (Switches de ACCESO). Cada Switch de acceso fue conectado a una PC, como se puede ver en la figura 5-3.

Cada una de las 3 computadoras fue configurada con una dirección IP contenida dentro de la subred 192.168.5.0/24.

Los enlaces entre el Switch de CORE y los Switches de ACCESO tienen una capacidad máxima de 10 Gbps, mientras que los enlaces entre los Switches de ACCESO y las PC son de una capacidad máxima de 1 Gbps.

Asimismo, los enlaces del plano de control tienen una capacidad máxima de 1 Gbps dedicado, teniéndose una configuración de gestión out-of-band (plano de control independiente físicamente del plano de datos).

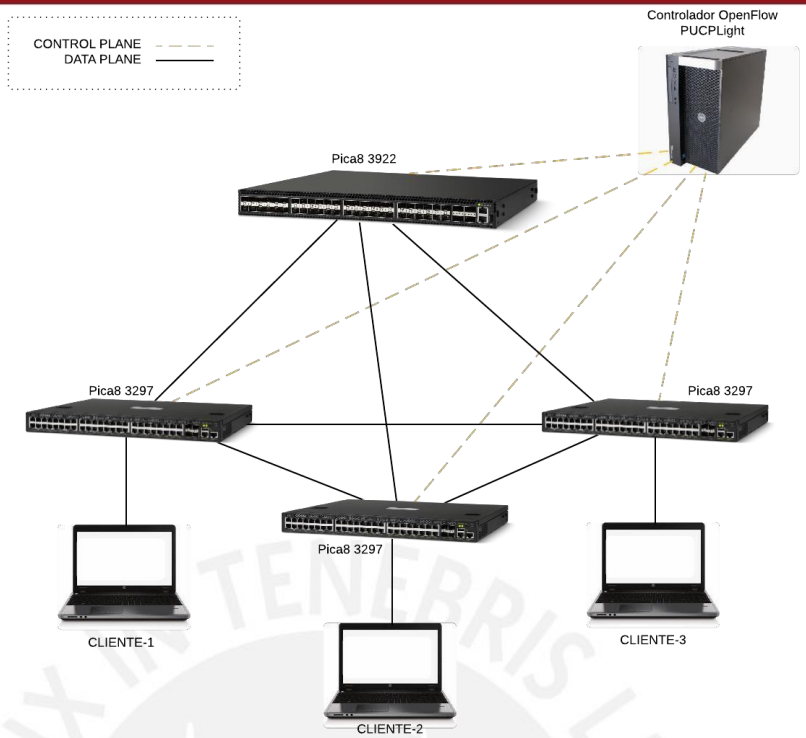

Figura 5-3. Topología física de prueba
ELABORACIÓN PROPIA (2015)

Se pudo comprobar que el controlador descubrió correctamente la topología, como se puede apreciar en la figura 5-4.

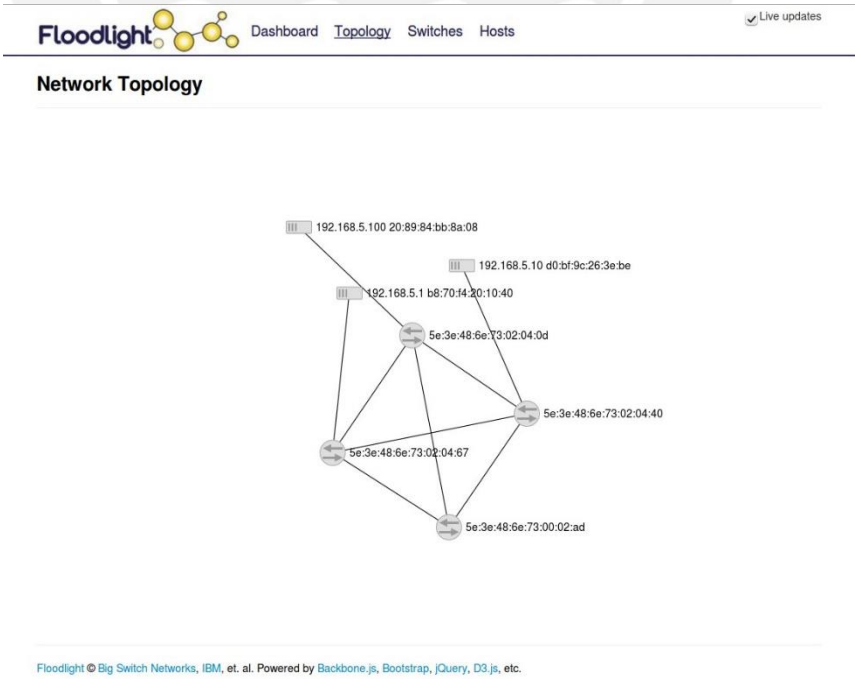

Figura 5-4. Topología descubierta por el controlador

Se realizaron pruebas de conectividad mediante el comando ping, cuyos resultados se muestran en las siguientes figuras.

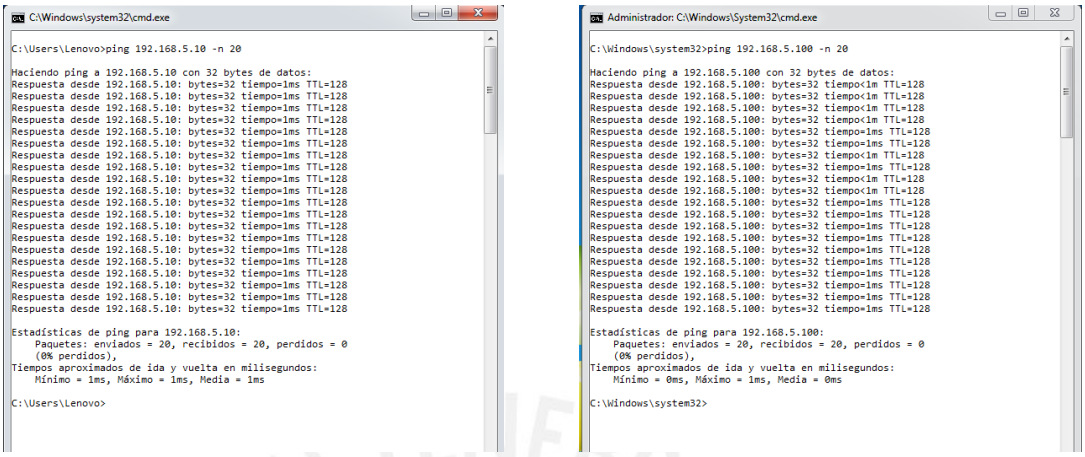

Figura 5-5. Pruebas de Ping desde el Cliente-1

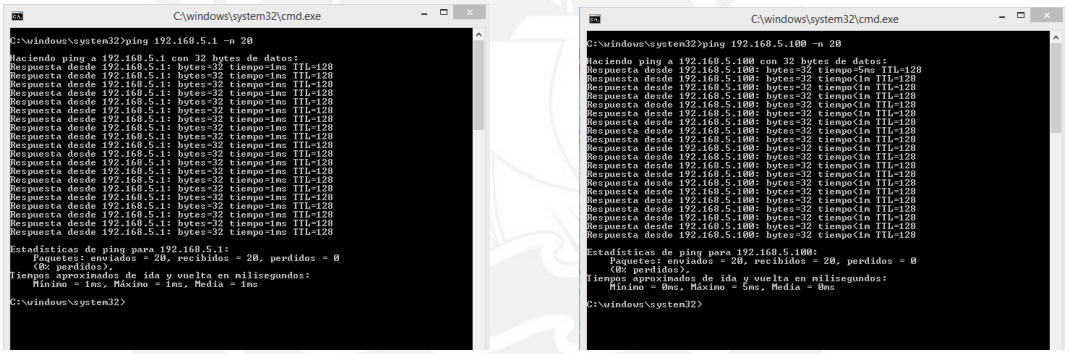

Figura 5-6. Pruebas de Ping desde el Cliente-2

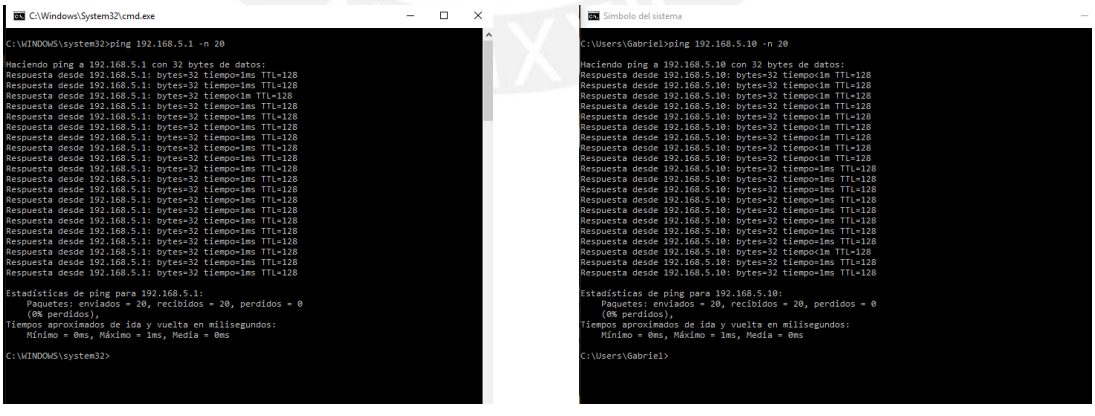

Figura 5-7. Pruebas de Ping desde el Cliente-3

5.1.3. Pruebas de esfuerzo

Las pruebas de esfuerzo realizadas se dividieron en dos: Pruebas de capacidad de los Switches, y pruebas de capacidad del controlador.

Los Switches utilizados en el laboratorio del GIRA-PUCP cuentan con la arquitectura de procesador Broadcom Triumph2 y, como podemos ver en la figura 5-8, soportan un máximo de 8192 direcciones MAC en su TCAM.

Se pudo observar que la cantidad de reglas que soportan sobrepasa grandemente a las limitantes físicas de TCAM que tienen, ya que las reglas se instalan inicialmente a nivel de Software, en Bases de Datos especiales dentro de los Switches y son descargadas a nivel de Hardware, solo cuando son utilizadas, siendo esta una forma de optimizar el uso de las TCAM en los Switches.

Chipset	TCAM Size	FIB Size	All tuples in TCAM	IPv4 in TCAM	IPv6 in TCAM	MAC in TCAM	ARP in TCAM	IPv4 in FIB	IPv6 in FIB
Firebolt3	108kB	32K	2048	4096	1024	4096	2048	12000	12000
Triumph2	216kB	32K	4096	8192	2048	8192	4096	12000	12000
Trident+	54kB	128K	1024	2048	512	2048	1024	12000	12000
Trident2	108kB	Common FIB and L2 Memory	2048	4096	1024	4096	2048	105000	105000

Figura 5-8. Limitantes físicas (TCAM) de los Switches Pica8
Pica8 HARDWARE GUIDES (2015)

Adicionalmente, se realizó pruebas para medir el Ancho de Banda de los enlaces en el Plano de Control, a través del uso de la herramienta iPerf [8], observándose un Ancho de Banda efectivo de 94.4 Mbps en enlaces FastEthernet (100 Mbps), como se puede observar en la figura 5-9.

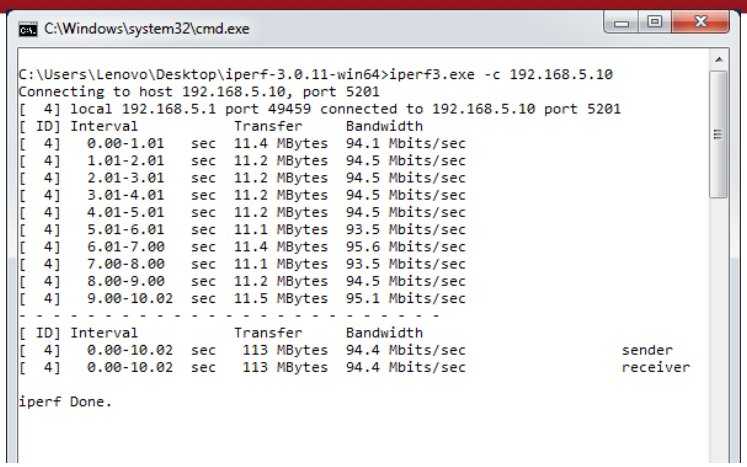

Figura 5-9. Medición del ancho de banda del enlace con iperf

Por otro lado, en cuanto a la capacidad del controlador, se realizaron pruebas de capacidad de procesamiento del mismo, a través del uso de la herramienta de benchmarking CBENCH, que forma parte de la plataforma de pruebas de elementos SDN llamada OFLOPS [15], obteniéndose los siguientes resultados:

#switches	Modo Latency		
	min	max	avg
1	6 520.99	19 580.98	18 139.50
10	45 491.95	49 838.95	48 208.09
100	41 718.42	56 228.44	53 923.55
200	36 416.89	56 767.83	51 879.78

Tabla 5-1. Resultados de las pruebas con CBENCH

Además, se pudo verificar el comportamiento correcto del controlador, de acuerdo a los módulos implementados. En la figura 5-10, se puede observar el dashboard de la interfaz web del controlador, donde se observan los módulos cargados, los Switches conectados y los hosts descubiertos.

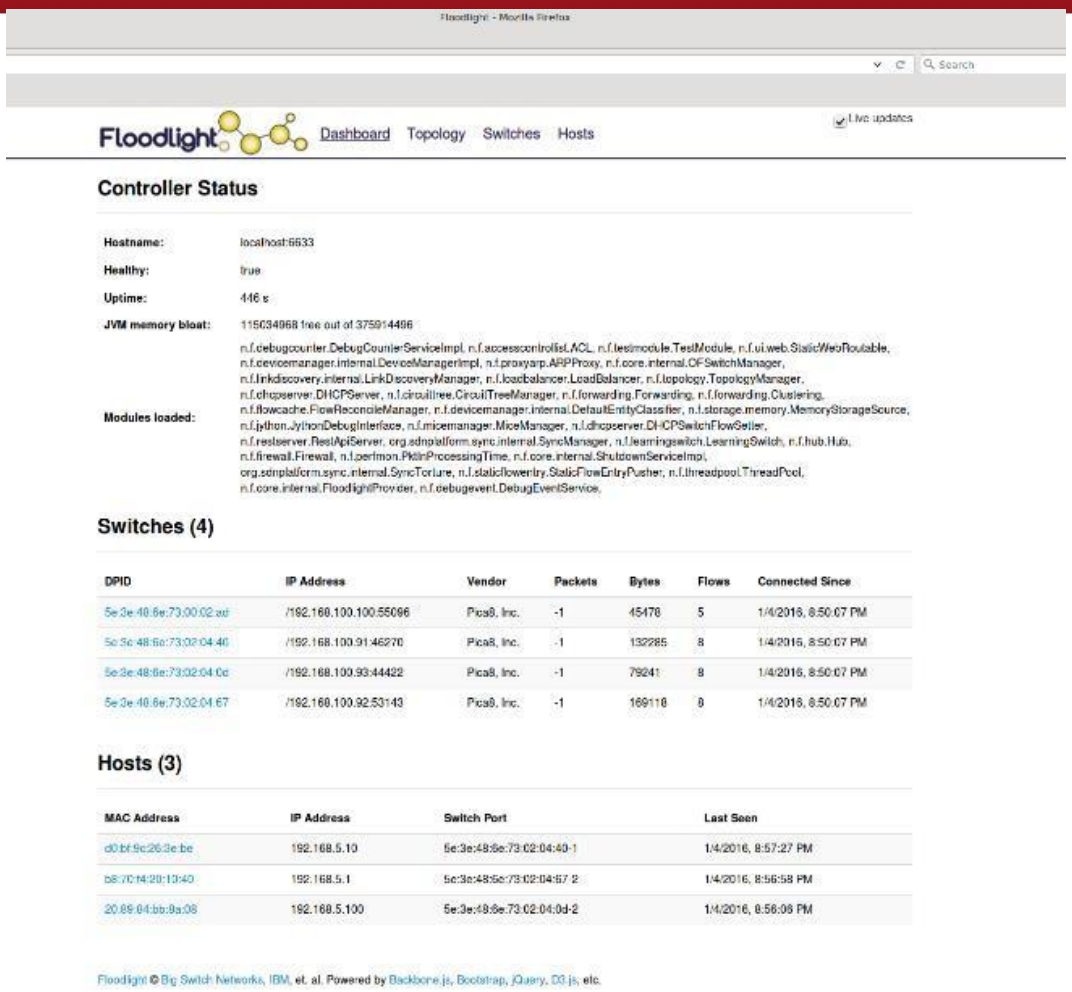

Figura 5-10. Dashboard del controlador

Además, se comprobó que las reglas instaladas corresponden al módulo Clustering (Figura 5-11).

Flows (7)

Cookie	Table	Priority	Match	Apply Actions
9007199254740992	0x0	2	eth_dst=b8:70:f4:20:10:40	actions:push_vlan=33024,eth_vlan_vid=4,output=50
0	0x0	0		actions:output=controller
9007199254740992	0x0	2	eth_dst=20:89:84:bb:8a:08 eth_vlan_vid=0x1001	actions:pop_vlan=n/a,output=2
0	0x1	0		actions:output=controller
0	0x2	0		actions:output=controller
0	0x3	0		actions:output=controller
0	0x4	0		actions:output=controller

Figura 5-11. Flujos instalados en el Switch del Cliente-1

5.1.4. Pruebas de Interoperabilidad

5.1.4.1.1. Red LAN con salida a Internet

Adicionalmente, se configuró una de las computadoras como Gateway hacia Internet, lográndose conectividad y navegación web en todos los hosts.

En la siguiente figura se puede comprobar que el ping hacia google.com fue exitoso, así como la navegación web.

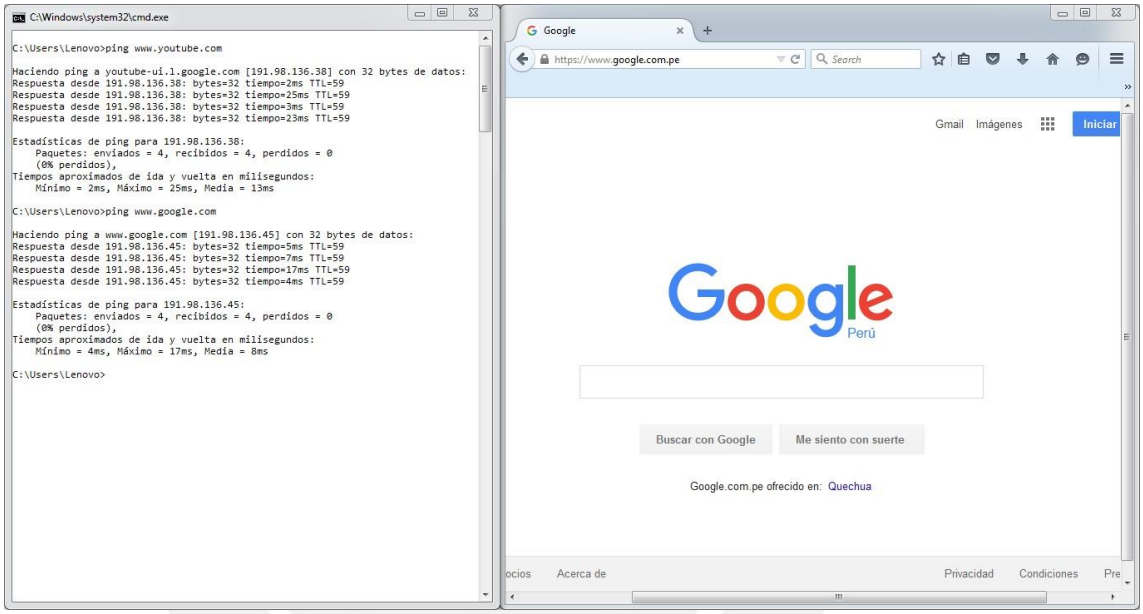

Figura 5-12. Acceso a Internet desde la red OpenFlow

5.1.4.1.2. Acceso a la Red PUCP

Para la realización de estas pruebas, se conectó un puerto del Switch de ACCESO a un punto de red asignado en el laboratorio del GIRA-PUCP.

Las pruebas de interoperabilidad con segmentos de red Legacy en el reconocimiento de las direcciones MAC, por parte del controlador, de todos los elementos de la red que se encuentren enviando paquetes de broadcast (mayoritariamente *gratuitous ARP*).

La siguiente figura es la captura del dashboard del controlador para esta prueba. Puede notarse que el controlador identificó muchos de los hosts conectados a la

subred, debido al tráfico de tipo broadcast que estos generan.

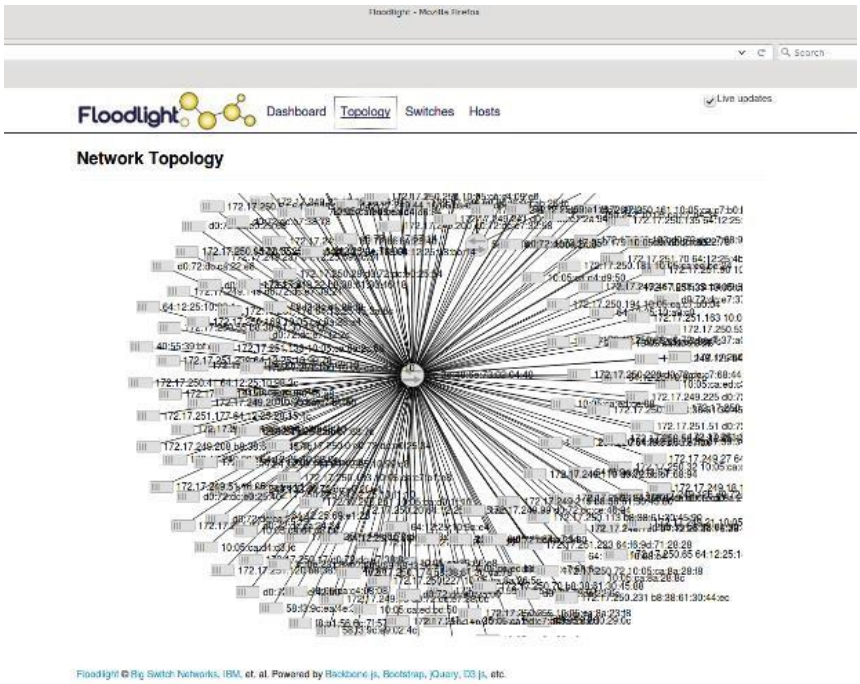

Figura 5-13. Topology de la red PUCP conocida a través de un Switch OpenFlow

5.2. Modelamiento del sistema y análisis de escalabilidad

En esta sección se analizará la escalabilidad del sistema propuesto usando una computadora COTS para albergar nuestro controlador y commodity Switches.

Este análisis asume lo siguiente:

- La red consiste de dos niveles. El primero es constituido por N nodos conectados a C Comutadores (Switches), cada uno con S puertos ($N \leq S \times C$). El segundo está constituido por una nube de Switches de Distribución y Core que interconecta los C Switches de acceso en una topología arbitraria y con alta redundancia, como se muestra en las figuras 4-5 y 5-1. El controlador PUCPLight estará también conectado a esta nube a través de una red de gestión de alta velocidad.

- El uso de optimal forwarding, así como la reducción del número de paquetes broadcast que viajan por la red, garantiza que la red tiene una mayor escalabilidad con respecto a volumen de tráfico que las redes actuales. Asimismo, el uso de forwarding en capa 2 significa que se pueden usar Switches de mayor tamaño a

un menor precio.

- Los mayores limitantes de nuestra red serán: (i) la capacidad de procesamiento del controlador, y (ii) el número de entradas disponibles en los Switches (TCAM).

- Los Switches en el segundo nivel de la jerarquía realizan decisiones de forwarding a un nivel de granularidad equivalente al número de Switches de acceso, es decir, no tienen visibilidad a las direcciones MAC/IP de los usuarios finales. En cada Switch de segundo nivel, el controlador pre-grabará una entrada por cada Switch de acceso, por lo que el Switch solo necesitará $C \ll N$ entradas, y no generará carga al controlador. Es así que los Switches de segundo nivel no constituyen un limitante a la escalabilidad, por lo que el análisis se centrará en los Switches de acceso.

- Se considera como una sesión una dupla (nodo origen, nodo destino). Para máxima generalidad, se usará la dirección MAC como identificador de un nodo. Cada sesión se considera estadísticamente independiente de las demás.

- Cada sesión se modela como un proceso de Márkov ON/OFF continuo, con parámetros λ_d y μ, donde λ_d representa el transition rate de OFF a ON, y μ es el transition rate de ON a OFF. En general, λ_d depende del destino (unos más populares que otros), mientras que μ se considera el mismo para todas las sesiones. La duración promedio de una sesión es $1/\mu$, y la probabilidad que una sesión este activa en un momento dado es igual a $\lambda_d/(\lambda_d + \mu)$. Entonces, para un destino d dado, el número promedio de sesiones activas con este nodo como destino es igual a $(N-1) \cdot \lambda_d/(\lambda_d + \mu)$.

- Cada entrada en una tabla OpenFlow de un Switch de acceso tiene un timer (holding time) que al ser activado borra la entrada de la tabla. Este timer se reinicia cada vez que un paquete hace match con esa entrada. Entonces, para la entrada correspondiente a un destino d, este timer solo se activa *holding time* segundos después que todas las sesiones destinadas al nodo d terminan. El controlador configura el valor de este timer independientemente para cada entrada. En este trabajo se asume que el controlador configura este timer usando un valor aleatorio derivado de una distribución exponencial con parámetro $1/\omega$.

En la subsección siguiente (5.2.1) determinaremos, para un destino d dado, el consumo memoria (TCAM) en el Switch y la carga (en paquetes) inducida en el Controlador. Luego, en la subsección 5.2.2, determinaremos el consumo total de memoria TCAM en un Switch y la carga total en el controlador debido a todas los destinos, para dos distribuciones de trafico: uniforme y Pareto.

5.2.1. Consumo de recursos asociado a un destino d

Empecemos considerando un Switch con solo un nodo conectado a él. En este caso, se puede representar el comportamiento de este sistema como una cadena de Márkov de 3 estados: ON, OFF, y HOLD. La figura 5-14 muestra un ejemplo en el tiempo de este sistema. Inicialmente el sistema está en OFF, lo que significa que no hay ninguna sesión activa. En un momento dado, una sesión empieza y el sistema hace la transición hacia el estado ON. Luego de un tiempo, la sesión termina, y el sistema hace la transición hacia el estado HOLD, donde el Switch mantiene la entrada OpenFlow hasta que el timer expire (transición al estado OFF) u otra sesión empiece (transición al estado ON, nuevamente). Se debe tener en cuenta que cuando el sistema está en estado OFF, el Switch no tiene una entrada OpenFlow asociada al destino. Y cuando el sistema experimenta una transición del estado OFF al estado ON el Switch envía un mensaje al controlador solicitando una entrada en la tabla OpenFlow que le indique como procesar paquetes para el destino d.

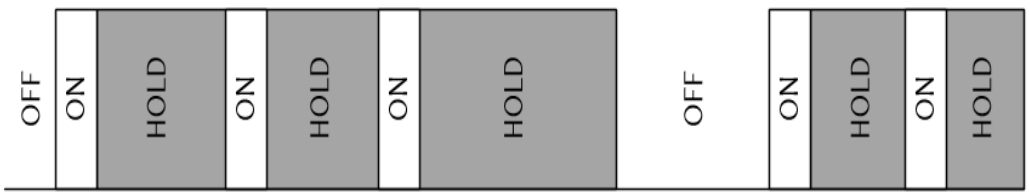

Figura 5-14. Flujo típico, representando sus estados
ELABORACIÓN PROPIA (2015)

Extendiendo este modelo para el caso de un Switch con S nodos conectados a cada uno de sus puertos de entrada, se puede modelar el sistema como una cadena de Márkov de $S+2$ estados, donde el estado ON_k representa el caso donde k nodos (orígenes) tienen su sesión al nodo d en estado ON. Esta cadena de Márkov puede ser representada por el diagrama de estados mostrado en la Figura 5-15.

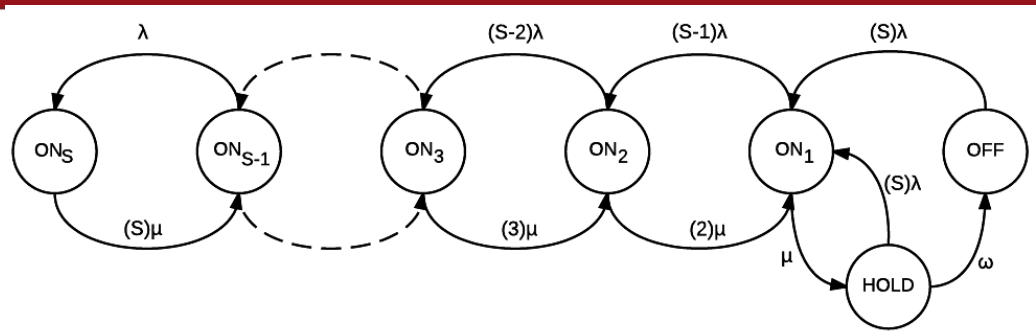

Figura 5-15. Cadena de Márkov de N estados
ELABORACIÓN PROPIA (2015)

Las ecuaciones de estado del modelo son las siguientes:

$$P_{HOLD} + P_{OFF} + \sum_{k=1}^{S} P_{ON_k} = 1 \quad \dots \text{(i)}$$

$$S \cdot \lambda \cdot P_{OFF} = \omega \cdot P_{HOLD} \quad \dots \text{(ii)}$$

$$(S \cdot \lambda + \mu) \cdot P_{HOLD} = \mu \cdot P_{ON_1} \quad \dots \text{(iii)}$$

$$\big((S-1) \cdot \lambda + \mu\big) \cdot P_{ON_1} = 2 \cdot \mu \cdot P_{ON_2} + S \cdot \lambda \cdot (P_{HOLD} + P_{OFF}) \quad \dots \text{(iv)}$$

$$\big((S-2) \cdot \lambda + 2 \cdot \mu\big) \cdot P_{ON_2} = 3 \cdot \mu \cdot P_{ON_3} + (S-1) \cdot \lambda \cdot P_{ON_1} \quad \dots \text{(v)}$$

$$\big((S-3) \cdot \lambda + 3 \cdot \mu\big) \cdot P_{ON_3} = 4 \cdot \mu \cdot P_{ON_4} + (S-3) \cdot \lambda \cdot P_{ON_2} \quad \dots \text{(vi)}$$

$$\vdots$$

$$(\lambda + (S-1) \cdot \mu) \cdot P_{ON_{S-1}} = S \cdot \mu \cdot P_{ON_S} + 2 \cdot \lambda \cdot P_{ON_{S-2}} \quad \dots \text{(vii)}$$

$$S \cdot \mu \cdot P_{ON_S} = \lambda \cdot P_{ON_{S-1}} \quad \dots \text{(viii)}$$

A partir de la deducción de una expresión generalizada para los estados ON de la cadena:

$$P_{ON_k} = \binom{S}{k} \cdot \left(\frac{\lambda}{\mu}\right)^k \cdot (P_{HOLD} + P_{OFF})$$

Podemos aplicarla en (i) de tal manera que podamos hallar una expresión que nos permita evaluar el estado inicial OFF:

$$(P_{HOLD} + P_{OFF}) \sum_{k=0}^{S} \binom{S}{k} \cdot \left(\frac{\lambda}{\mu}\right)^{k} = 1$$

Para valores de $\lambda < \mu$, la serie contenida en la expresión es conocida:

$$\sum_{k=0}^{S} \binom{S}{k} \cdot \left(\frac{\lambda}{\mu}\right)^{k} = \left(1 + \frac{\lambda}{\mu}\right)^{S}$$

Por lo que al aplicarla, junto con otras transformaciones, nos lleva a la expresión del estado inicial OFF:

$$P_{OFF} = \frac{1}{\left(1 + \frac{S \cdot \lambda}{\mu}\right) \cdot \left(1 + \frac{\lambda}{\mu}\right)^{S}}$$

Con la cual podemos despejar las expresiones que nos servirán para realizar el análisis de escalabilidad, que son el número de ARRIVALS al controlador y la OCCUPANCY de las memorias de los Switches.

Como se explicó anteriormente, un mensaje al controlador (ARRIVAL al controlador) será generado cada vez que el sistema haga una transición del estado OFF al estado ON. Es decir, el rate de arrivals al controlador provenientes de este Switch relacionados al destino d es igual al rate "out" del estado P_{OFF} ($S \cdot \lambda \cdot P_{OFF}$), es decir:

$$ARR = \frac{S \cdot \lambda}{\left(1 + \frac{S \cdot \lambda}{\mu}\right) \cdot \left(1 + \frac{\lambda}{\mu}\right)^{S}}$$

Asimismo, el Switch tendrá una entrada asociada al destino d si y solo si el sistema se encuentra en cualquier estado excepto OFF. Es decir, la probabilidad que la entrada asociada al destino d exista en la TCAM del Switch es igual a $(1 - P_{OFF})$. Entonces, la OCCUPANCY promedio de la entrada asociada al destino d es igual a:

$$OCC = S \cdot \lambda \cdot \left(1 + \frac{\lambda}{\mu}\right)^{-S} \cdot \left(1 - \frac{1}{\left(1 + \frac{\omega}{\mu} \cdot \frac{\mu}{S \cdot \lambda}\right)}\right)$$

Finalmente, debemos realizar un reordenamiento de las expresiones de manera que puedan tener una forma fácil de analizar.

$$ARR = S \cdot \lambda \cdot \left(1 + \frac{\lambda}{\mu}\right)^{-S} \left(1 - \frac{1}{\left(1 + \frac{\omega}{\mu} \cdot \frac{\mu}{S \cdot \lambda}\right)}\right)$$

$$OCC = \left(1 - \left(1 + \frac{\lambda}{\mu}\right)^{-S}\right) \cdot \left(1 + \frac{1}{\left(1 + \frac{\omega}{\mu} \cdot \frac{\mu}{S \cdot \lambda}\right)\left(\left(1 + \frac{\lambda}{\mu}\right)^{S} - 1\right)}\right)$$

Para esto, se definen los siguientes parámetros de relación $\rho_\lambda = \frac{S \cdot \lambda}{\mu}$ y $\rho_\omega = \frac{\omega}{\mu}$.

$$ARR = S \cdot \lambda \cdot \left(1 + \frac{\rho_\lambda}{S}\right)^{-S} \left(1 - \frac{1}{\left(1 + \frac{\rho_\omega}{\rho_\lambda}\right)}\right)$$

$$OCC = \left(1 - \left(1 + \frac{\rho_\lambda}{S}\right)^{-S}\right) \cdot \left(1 + \frac{1}{\left(1 + \frac{\rho_\omega}{\rho_\lambda}\right)\left(\left(1 + \frac{\rho_\lambda}{S}\right)^{S} - 1\right)}\right)$$

Y utilizando la siguiente propiedad:

$$\lim_{f(x) \to 0} \left(1 + f(x)\right)^{\frac{1}{f(x)}} = e$$

Con $\rho_\lambda \ll N$, podemos decir que si $f(x) = \frac{\rho_\lambda}{S} \Longrightarrow f(x) \longrightarrow 0$

Llegamos a las siguientes expresiones finales:

$$ARR = \mu \cdot \rho_\lambda \cdot e^{-\rho_\lambda} \left(1 - \frac{1}{\left(1 + \frac{\rho_\omega}{\rho_\lambda}\right)}\right)$$

$$OCC = (1 - e^{-\rho_\lambda}) \cdot \left(1 + \frac{1}{\left(1 + \frac{\rho_\omega}{\rho_\lambda}\right)(e^{\rho_\lambda} - 1)}\right)$$

5.2.2. Consumo de recursos para todos los destinos.

5.2.2.1. Consumo de TCAM en un Switch

Sea X_d el número de entradas en la taba TCAM de un Switch asociada al destino d en un instante t, X_d es una variable aleatoria Bernoulli con parámetro $p = OCC(\lambda_d)$. Entonces, el número total de entradas TCAM en el Switch en un instante t(OCC_{TOT}) es igual a:

$$OCC_{TOT} = \sum_{d=1}^{N} X_d$$

Como N es un número largo, se puede aplicar la ley de números grandes:

$$OCC_{TOT} \to N \cdot E_d\{X_d\}$$

Más precisamente, OCC_{TOT} se puede aproximar a una variable Gaussiana con parámetros:

$$E\{OCC_{TOT}\} = \sum_{d=1}^{N} E_d\{X_d\} = N \cdot E_d\{X_d\} = N \cdot E_d\{OCC(\lambda_d)\}$$

$$Var\{OCC_{TOT}\} = \sum_{d=1}^{N} Var\{X_d\} \leq \sum_{d=1}^{N} E_d\{X_d\} = N \cdot E_d\{OCC(\lambda_d)\}$$

Donde la segunda ecuación se cumple pues la varianza de la suma de variables independientes es igual a la suma de las varianzas, y porque la varianza de una variable Bernoulli de parámetro p es igual a $p \cdot q < p$. Cabe mencionar que para la mayoría de destinos en la sumatoria anterior, el valor de p es muy pequeño y q es muy cercano a 1, por lo que la cota superior es muy precisa.

Sea $\mathrm{OCC_{MAX}}$ el tamaño de memoria TCAM disponible en un Switch, entonces la probabilidad de que en este Switch, en algún instante se exceda este límite es igual a:

$$\mathrm{P_{OVERFLOW}} = \mathrm{Q} \cdot \left((\mathrm{OCC_{MAX}} - \mathrm{N} \cdot \mathrm{E_d}\{\mathrm{OCC}(\lambda_d)\}) / \sqrt{\mathrm{N} \cdot \mathrm{E_d}\{\mathrm{OCC}(\lambda_d)\}} \right)$$

Donde la función Q es la *tail probability* de una variable normal estándar. Ha de notarse que en caso de un overflow de la memoria TCAM, la entrada será almacenada en la memoria convencional (RAM) del Switch, lo que afectará la performance (incrementará la latencia) del Switch, y (de ser excesiva la carga) podría eventualmente resultar en pérdida de paquetes.

Entonces, si se queremos garantizar que la probabilidad de overflow de las TCAM de un Switch es menor a un valor ε, tenemos que:

$$\left((\mathrm{OCC_{MAX}} - \mathrm{N} \cdot \mathrm{E_d}\{\mathrm{OCC}(\lambda_d)\}) / \sqrt{\mathrm{N} \cdot \mathrm{E_d}\{\mathrm{OCC}(\lambda_d)\}} \right) \geq \mathrm{Q}^{-1} \cdot \varepsilon$$

O, similarmente,

$$\mathrm{N} \cdot \mathrm{E_d}\{\mathrm{OCC}(\lambda_d)\} + \mathrm{Q}^{-1} \cdot \varepsilon \cdot \sqrt{\mathrm{N} \cdot \mathrm{E_d}\{\mathrm{OCC}(\lambda_d)\}} \leq \mathrm{OCC_{MAX}}$$

De la anterior expresión se puede hallar el valor máximo de N para no exceder la probabilidad de *overflow* deseada. Denominaremos esta cantidad $\mathrm{N_{MAX_{TCAM}}}$.

5.2.2.2. Carga en el controlador

A diferencia del caso anterior, donde cada Switch es tratado en forma independiente, para la carga del controlador se debe sumar las contribuciones de todos los Switches para todos los destinos.

Sea $\mathrm{ARR}(d, c)$ el arrival rate de paquetes al controlador con respecto al destino d generado por el Switch c. Entonces, el arrival rate total al controlador es igual a la suma:

$$\mathrm{ARR_{TOT}} = \sum_{c=1}^{C} \sum_{d=1}^{N} \mathrm{ARR(d,c)}$$

Es decir, el arrival rate total es la suma de $\mathrm{N \cdot C}$ arrival process independientes, cada uno de baja intensidad. Entonces, el agregado de todos estos procesos puede aproximarse (con gran precisión) a un proceso de Poisson con media:

$$\mathrm{E\{ARR_{TOT}\}} = \mathrm{N \cdot C \cdot E_d\{ARR(d)\}} = \left(\frac{\mathrm{N^2}}{\mathrm{S}}\right) \cdot \mathrm{E_d\{ARR(d)\}}$$

Denominemos $\mathrm{ARR_{MAX}}$ al número máximo de arrivals por segundo que el controlador puede procesar. El tiempo de servicio del controlador es entonces $1/\mathrm{ARR_{MAX}}$. La distribución (pdf) de este tiempo de servicio depende de los procesos siendo ejecutados, y es probablemente cercano a un valor constante. Sin embargo, consideraremos que este tiempo de servicio tiene una distribución exponencial para obtener una aproximación conservadora del comportamiento (retardo, perdida) de los paquetes que llegan al controlador con una restricción de tiempo de respuesta.

Sea $\mathrm{T_{REPLY}}$ el tiempo máximo de espera de la respuesta de un controlador. Es decir, el tiempo de espera para que el Switch obtenga la información necesaria para empezar a transmitir los paquetes de una nueva sesión. La probabilidad que una consulta al controlador tome más de este tiempo, puede ser hallada considerando al controlador como una cola M/M/1/m donde m (el tamaño del buffer) es igual a $\mathrm{T_{REPLY}} \cdot \mathrm{ARR_{MAX}}$. En este caso, la probabilidad buscada es equivalente a la probabilidad de pérdida de paquete:

$$\mathrm{P_e} = \frac{(1-\rho) \cdot \rho^{\mathrm{m}}}{(1-\rho^{\mathrm{m}})}$$

Donde:

$$\rho = \frac{\mathrm{E\{ARR_{TOT}\}}}{\mathrm{ARR_{MAX}}}$$

Entonces, para un $\mathrm{P_e}$ y $\mathrm{T_{REPLY}}$ objetivo, se puede hallar el máximo valor de ρ que no exceda $\mathrm{P_e}$, y en base a este ρ se puede determinar el valor máximo de N. Denominamos esta cantidad como $\mathrm{N_{MAX_{ARR}}}$.

5.2.2.3. Calculo de holding time ($1/\omega$) óptimo y máximo número de nodos N_{MAX}

Como se vio en 5.2.1, las expresiones de ARR y OCC dependen del valor de ω, donde $1/\omega$ es el tiempo promedio de espera antes de borrar una entrada OpenFlow luego que un sesión expiró. Entonces, para un ω dado, el número de nodos que el sistema puede soportar es igual a:

$$N_{MAX}(\omega) = \min\{N_{MAX_{TCAM}}(\omega),\ N_{MAX_{ARR}}(\omega)\}$$

Entonces, el óptimo valor de ω puede ser hallado como:

$$\omega_{OPTIMO} = \arg\max_{\omega}\left(\min\{N_{MAX_{TCAM}}(\omega)\ , N_{MAX_{ARR}}(\omega)\}\right)$$

Finalmente, el máximo número de nodos que el sistema soporta es igual a:

$$N_{MAX} = \min\{N_{MAX_{TCAM}}(\omega_{OPTIMO}), N_{MAX_{ARR}}(\omega_{OPTIMO})\}$$

En lo que queda de esta sección evaluaremos numéricamente el valor de N_{MAX} bajo dos patrones de tráfico: (i) distribución uniforme (todos los destinos son igualmente probables) y (ii) Pareto (el número de sesiones, o la "popularidad" de un destino sigue una distribución Pareto).

El valor de los parámetros usados se muestra en la tabla 5-1, y se basa en el comportamiento observado en el campus PUCP.

En particular, examinando un trace de 30 minutos de tráfico se observó que la duración promedio de una sesión era del orden de [1, 10] segundos. Y que el número de sesiones simultaneas era del orden de 40 000, es decir 4 por destino, en una red de unos 10 000 nodos.

Es de esperar que, a medida que aumenta el número de nodos en la red (N se incrementa) el número de sesiones en las que un nodo está involucrado se mantenga constante, variando solamente la probabilidad de comunicarse con un destino en particular. Es decir, el tiempo/número de interacciones de un usuario no cambian, pero el set de posibles destinos se hace mayor.

Parámetro	Símbolo	Valor
Duración promedio de una sesión	$1/\mu$	5 s
Número de puertos/hosts por switch	S	48
Número promedio de sesiones concurrentes para un mismo destino		4
Máximo número de entradas TCAM en un Switch	OCC_{MAX}	8096
Máximo número de paquetes por segundo que el controlador puede procesar	ARR_{MAX}	51 880
Máxima probabilidad de overflow de TCAM tolerada	$P_{OVERFLOW}$	10^{-5}
Máxima probabilidad de que un paquete al controlador no regrese a tiempo (< T_{REPLY}) tolerada.	P_e	10^{-5}
Máximo tiempo de espera de un paquete enviado al controlador.	T_{REPLY}	1 ms

Tabla 5-2. Parámetros usados en el análisis numérico de escalabilidad del sistema. ELABORACIÓN PROPIA (2015)

5.2.2.4. Tráfico uniforme

Recordando que el número promedio de sesiones destinadas a un mismo nodo es igual a $(N-1)\cdot\lambda_d/(\lambda_d+\mu)$, y dado que este valor es igual a 4 (independientemente de N) y $\mu = 0.2$ (ver tabla 5-2) tenemos que $\lambda_d = 0.8/(N-5)$, valor constante y pequeño.

Por tanto,

$$E\{OCC_{TOT}\} = N\cdot E_d\{OCC(\lambda_d)\} = N\cdot OCC(0.8/(N-5))$$

Por otro lado, como $P_{OVERFLOW} = 10^{-6}$, $Q^{-1}(10^{-6}) = 4.57535$ y la condición para garantizar una probabilidad de overflow menor que $P_{OVERFLOW}$ puede ser escrita como:

$$X + 4.7535\cdot\sqrt{X} \leq 8\,192$$

O, equivalentemente

$$X \leq 7\,679$$

Donde X es igual a $E\{OCC_{TOT}\}$, por lo que finalmente tenemos que:

$$N \cdot OCC(0.8/(N-5)) \leq 7\,679$$

Similarmente, para $T_{REPLY} = 1$ ms y $ARR_{MAX} = 51\,880$ pkts/s tenemos $m = 51$, y por lo tanto, para $P_e = 10^{-6}$ tenemos que $\rho \leq 0.787$ y $E\{ARR_{TOT}\} \leq 40\,829.56$ pkts/s.

Entonces:

$$(N^2/S) \cdot ARR(0.8/(N-5)) \leq 40\,829.56$$

Evaluando numéricamente, se obtiene $\omega_{OPT} = 1/180\ s^{-1}$ y $N_{MAX} = 56\,420$ hosts.

5.2.2.5. Tráfico con distribución de Pareto

En este caso, se parte del supuesto de que la popularidad de cada destino sigue una distribución Pareto truncada. Esto es consistente con observaciones empíricas. En particular, asumiremos que el número promedio de sesiones en la que un destino está involucrado (x) tiene la siguiente pdf:

$$pdf_x(x) = \begin{cases} \dfrac{\alpha \cdot {x_m}^{\alpha}}{1 - \left(\frac{x_m}{H}\right)^{\alpha}} \cdot x^{-\alpha-1} & x_m < x < H \\ 0 & x < x_m \text{ ó } x > H \end{cases},$$

Donde x_m es el parámetro de escala, α el parámetro de forma, y $H = N - 1 \gg x_m$.

El valor esperado de x es $\sim \alpha \cdot x_m/(\alpha - 1)$. Como este valor debe ser igual a 4 (ver Tabla 5-2) y como nuestras observaciones de la distribución de popularidad de hosts en la PUCP sugieren un $\alpha \sim 1.2$, elegimos $x_m = 2/3$ y $\alpha = 1.2$ para caracterizar la distribución de popularidad de los hosts en la PUCP. Entonces:

$$f_x(x) \sim 0.7377 \cdot x^{-2.2}$$

Para un destino dado, se cumple que el promedio (en el tiempo) de sesiones simultaneas x_d es igual a $(N-1) \cdot \lambda_d/(\lambda_d + \mu)$, por lo que sigue que $x_d = h(\lambda_d)$ para todo λ_d, donde $h(z) = (N-1) \cdot z/(z + 0.2)$.

Entonces, se puede usar esta transformación $h(z)$ para hallar la pdf de λ, como sigue:

$$f_\lambda(\lambda) = f_x(x) \cdot \frac{dx}{d\lambda} = f_x\big(h(\lambda)\big) \cdot h'^{(\lambda)} = 0.7377 \cdot \left(\frac{(N-1)\cdot\lambda}{\lambda+0.2}\right)^{-2.2} \cdot \frac{0.2\cdot(N-1)}{(\lambda+0.2)^2}$$

$$f_\lambda(\lambda) = 0.1475 \cdot \frac{(\lambda+0.2)^{0.2}}{(N-1)^{1.2}\cdot\lambda^{2.2}}$$

Para $\lambda > 0.4/(3\cdot N - S) = \lambda_{min}$. Entonces:

$$E_\lambda\{OCC(\lambda)\} = \int_{\lambda_{min}}^{\infty} f_\lambda(\lambda)\cdot OCC(\lambda)\ d\lambda$$

$$E_\lambda\{ARR(\lambda)\} = \int_{\lambda_{min}}^{\infty} f_\lambda(\lambda)\cdot ARR(\lambda)\ d\lambda$$

Evaluando numéricamente las integrales para múltiples valores de N y ω, obtenemos que $E\{OCC_{TOT}\} \leq 6\,429$ TCAMs y $E\{ARR_{TOT}\} \leq 40\,725$ pkts/s, además de tener $\omega_{OPTIMO} = 1/340\ s^{-1}$ y $N_{MAX} = 100\,004$ hosts.

Los resultados de las simulaciones se muestran en la siguiente tabla:

	Tráfico uniforme	Tráfico Pareto
OCC_{IDEAL} (TCAM)	6 329	6 429
ARR_{IDEAL} (pkts/s)	40 076	40 725
ω_{IDEAL} (s^{-1})	1/180	1/340
N_{MAX}	56 420	100 004

Tabla 5-3. Resultados de las simulaciones

5.3. Consideraciones adicionales

5.3.1. Seguridad

5.3.1.1. Protección del Control Channel

Inicialmente, la comunicación en el canal de control se realiza a través de una conexión TCP entre el Switch y el controlador, iniciada por el Switch. Sin embargo, a partir de la versión 1.0 del protocolo OpenFlow se especifica la opción de protección del canal de control, a través de la encriptación del mismo, utilizando el protocolo TLS (Transport Layer Security).

La comunicación en el canal de control se realiza a través de un túnel TLS, el cual se crea por petición del Switch, al iniciar este la conexión con el controlador. Ambos elementos realizan una autenticación mutua a través del intercambio de certificados firmados por llaves privadas.

Cada Switch debe ser configurado con dos tipos de certificados:

- Un certificado para autenticar al controlador (certificado del controlador), y
- Un certificado para autenticarse en el controlador (certificado del Switch)

Los cuales deben encontrarse pre-instalados en el repositorio local de certificados del controlador (Java Keystore).

5.3.1.2. Protección de la REST API

La plataforma de controlador Floodlight soporta tres modos de seguridad para su REST API:

- HTTP no-seguro
- HTTPS confiable
- HTTPS encriptado

El uso del protocolo HTTPS debe ser habilitado manualmente, ya que este se encuentra deshabilitado por defecto.

Para habilitar la encriptación de la REST API, se debe configurar el controlador para apuntar a la ubicación del repositorio local de certificados digitales y la contraseña de acceso al mismo.

Adicionalmente, se debe tener un par de llaves públicas y privadas para la conexión, cuyos certificados deben estar pre-instalados en el repositorio de certificados locales.

5.3.1.3. Protección contra ataques DoS

El protocolo de gestión sFlow se está utilizando para recolectar datos en tiempo real de la red, como ocupación y throughput de los enlaces; adicionalmente, se utiliza para obtener estadísticas de las conexiones, para poder prevenir y/o detectar ataques de denegación de servicios, midiendo parámetros como:

- Cantidad de flujos iniciados por un host
- Cantidad de flujos que tienen como destino a un host

Una vez detectados estos eventos, se puede utilizar el módulo de StaticFlowPusher para instalar reglas en los Switches que desechen los paquetes que provienen de los hosts involucrados en el ataque.

5.3.2. Alta disponibilidad

5.3.2.1. OpenFlow multiple controller

La conexión de un Switch hacia múltiples controladores permite mejorar la confiabilidad del plano de control, ya que el Switch puede continuar recibiendo instrucciones OpenFlow si uno de los controladores, o la conexión hacia este, falla.

A pesar de que el protocolo OpenFlow, en su versión 1.0, menciona el soporte de la conexión de un Switch a múltiples controladores, es a partir de la versión 1.2 que esto es especificado.

Al iniciar la comunicación OpenFlow, el Switch debe conectarse con todos los

controladores que tenga configurados, y a partir de este momento mantener una comunicación concurrente con ellos.

Existen tres roles que puede tener un controlador:

- EQUAL

  El rol por defecto de un controlador es EQUAL. En este rol, el controlador tiene acceso de lectura y escritura al Switch y es igual que todos los otros controladores que tengan el mismo rol, pudiendo enviar comandos que modifiquen el estado del Switch.

- SLAVE

  El rol de SLAVE solo le da al controlador acceso de lectura a lo Switches, imposibilitándolo de enviar paquetes y comandos de cambio de estado al Switch.

- MASTER

  El rol de MASTER es similar al rol de EQUAL, ya que se tiene permisos de lectura y escritura al Switch, con la diferencia de que el controlador se asegura de ser el único que tiene este rol.

El hand-over (cambio) entre controladores es realizado por solicitud de ellos mismos y no por el Switch; asimismo, el cambio de rol de un controlador debe ser iniciado igualmente por un controlador, pudiendo ser en cualquier momento.

5.3.2.2. Floodlight High Availability

Floodlight soporta la alta disponibilidad de controladores de una manera similar a lo especificado en el protocolo OpenFlow, con la diferencia de que en vez de existir 3 roles (EQUAL, MASTER, SLAVE), un controlador solo puede permanecer en estado ACTIVO o en STANDBY.

Un controlador en estado ACTIVO es aquel que se encuentra gestionando a la red. En cambio, un controlador en STANDBY es aquel que se encuentra inactivo, a la espera de que el controlador en estado ACTIVO falle, para cambiar su estado.

A diferencia del estado ACTIVO, donde solo un controlador puede permanecer a la vez, puede haber múltiples controladores en STANDBY.

Este feature se encuentra implementado dentro del módulo de core HARole, el cual se encarga de definir y guardar el estado de cada uno de los controladores.

CONCLUSIONES

- Se cumplió con el objetivo principal de la tesis, se diseñó un controlador OpenFlow escalable, y se implementó una prueba de concepto sobre la plataforma Floodlight.

- El controlador implementado es escalable en para el tráfico unicast, mediante el mecanismo usado por el módulo Clustering. Se demostró que usando este mecanismo se obtiene un 25% de uso en las TCAM de los Switches y en la capacidad del controlador.

- El Timeout de las Flow Entries determina influye directamente en el porcentaje de uso de las TCAM de los Switches y de la capacidad del controlador, y por lo tanto, en la escalabilidad del sistema.

- Cuando la distribución de destinos tiene una alta concentración en unos pocos hosts, se logra escalabilidad de las TCAM de los Switches de acceso respectivos mediante el módulo Circuit Tree.

- Bajo el nuevo esquema de subnetting, el tráfico Broadcast aumentará. Ante esto, el controlador provee los módulos para evitar tormentas de broadcast y saturación de enlaces. Además se comprueba que este enfoque otorga la información necesaria para optimizar el enrutamiento.

- El controlador puede coexistir con elementos Legacy. Puede usarse para iniciar una migración a OpenFlow gradual.

- El diseño del controlador soporta una migración total de la Red PUCP a SDN, evitando así el uso de un Router centralizado y consecuentemente, las desventajas de la red actual que se indicaron en el capítulo 1.

TRABAJO FUTURO

- Análisis con duración de sesiones Pareto. En el análisis hecho en esta tesis, se consideró que la duración de una sesión es constante (parámetro mu). Sin embargo, para realizar predicciones más precisas, esta puede ser modelada con una distribución Pareto.

- Clustering con Timeout dinámicos. A partir del modelo analítico del sistema, se puede asignar un Timeout óptimo a cada flujo, de acuerdo a una clasificación previa, teniendo en cuenta que el uso de un destino sigue una distribución Pareto.

- Portar la solución a OpenDaylight y/o ONOS. OpenDaylight y ONOS son las plataformas de controlador más avanzadas. Tras haberse validado el diseño de un controlador escalable, y haberse realizado una prueba de concepto en un controlador robusto como Floodlight, es factible portar el controlador a una de estas dos plataformas, a fin de conseguir una mayor robustez, y estar a la altura de los controladores usados por los fabricantes más importantes de la industria.

- Integración con la red inalámbrica. Si bien ya se tiene hardware WLAN con soporte para OpenFlow, el problema principal radica en la gestión de la movilidad. Este problema es resuelto de forma óptima y transparente mediante el control centralizado.

- Extensión a IPv6 y multicast. Tanto Floodlight como las plataformas más avanzadas de controladores tienen soporte nativo de IPv6, por lo que para el forwarding se requieren cambios mínimos a nivel de código. Sin embargo, en IPv6 aparecen protocolos de red como SLAAC, DHCPv6, que requieren implementación de multicast. Para esto se requiere implementar módulos que construyan los árboles multicast y elijan un Rendezvous Point para que los servicios puedan funcionar eficientemente, sin saturar el plano de control.

- Alta disponibilidad. Si bien Floodlight provee interfaces para implementar un esquema Active-Stand By, estas deben desarrollarse y someterse a un riguroso set de pruebas.

REFERENCIAS

[1] Big Switch Networks. (10 de 12 de 2015). Floodlight Project. http://www.projectfloodlight.org/floodlight/

[2] CAM (Content Addressable Memory) VS TCAM (Ternary Content Addressable Memory) (08 de 12 de 2015) Obtenido de: Cisco Support Forum. https://supportforums.cisco.com/document/60831/cam-content-addressable-memory-vs-tcam-ternary-content-addressable-memory

[3] Erickson, D. (2013, August). The beacon OpenFlow controller. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (pp. 13-18). ACM.

[4] Floodlight Developers Mailing List. https://groups.google.com/a/openflowhub.org/forum/#!forum/floodlight-dev

[5] Göransson, P., & Black, C. (2014). Software Defined Networks A Comprehensive Approach. Waltham: Elsevier Inc.

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

[7] Internet Research Task Force (IRTF). (2015). Software-Defined Networking (SDN): Layers and Architecture Terminology.

[8] iPerf. https://iperf.fr/

[9] Khondoker, R., Zaalouk, A., Marx, R., & Bayarou, K. (2014, January). Feature-based comparison and selection of Software Defined Networking (SDN) controllers. In Computer Applications and Information Systems (WCCAIS), 2014 World Congress on (pp. 1-7). IEEE.

[10] LA PUCP EN CIFRAS (06 de 12 de 2015) http://www.pucp.edu.pe/la-universidad/nuestra-universidad/pucp-en-cifras/

[11] LINUX Foundation. OpenDaylight. (10 de 12 de 2015) https://www.opendaylight.org/

[12] Luca Prete. CustomCostBalancer https://github.com/LucaPrete/Custom-costs-balancer/blob/master/it/garr/ccbalancer/ICCBalancerListener.java

[13] Morreale, Patricia A.; Anderson, James A. (2015) Software Defined Networking. Design and Deployment. U.S.: CRC Press

[14] Nadeau, T., & Gray, K. (2013). Big Switch Networks/Floodlight. En T. D. Nadeau,

& K. Gray, SDN: Software Defined Networks (págs. 93, 94, 95). Sebastopol: O'Reilly Media.

[15] Oflops
http://archive.openflow.org/wk/index.php/Oflops

[16] ON.LAB. (11 de 2014). Introducing ONOS - a SDN network operating system for.
http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf

[17] Open Networking Foundation. (2012). OpenFlow Switch Specification Version 1.3.0. Open Networking Foundation.

[18] Open vSwitch. (10 de 12 de 2015). Open vSwitch.
http://openvswitch.org/

[19] OpenFlow Switching Performance: Not All TCAM is Created Equal (13 de 12 de 2015)
http://packetpushers.net/OpenFlow-switching-performance-not-all-tcam-is-created-equal/

[20] Pica8 Inc. (10 de 12 de 2015). Pre-loaded Switches.
http://www.pica8.com/products/pre-loaded-Switches

[21] Ryu SDN Framework Community. (10 de 12 de 2015). Ryu
http://osrg.github.io/ryu/

[22] SDN Open Flow TCAM Ned to Know (13 de 12 de 2015)
https://www.sdxcentral.com/articles/contributed/sdn-OpenFlow-tcam-need-to-know/2012/07/

[23] Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., & Smeliansky, R. (2013, October). Advanced study of SDN/OpenFlow controllers. In Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia (p. 1). ACM.

[24] Slone, John P. (1999) Local Area Network Handbook (6th Ed). U.S.: CRC Press

[25] Smith, R. D. (2011). The dynamics of internet traffic: self-similarity, self-organization, and complex phenomena. Advances in Complex Systems, 14(06), 905-949.

[26] Spanning Tree PortFast BPDU Guard Enhancement (06 de 12 de 2015)
http://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/10586-65.html

[27] Tanenbaum, Andrew S. (2012). Redes de Computadoras (5ta Ed). Mexico: Pearson Educación.

[28] Tavarez, Agueda S. (2011) Network Architecture for Univertsity Campus Network (Tesis de Maestría). College of Communication Engineering of

Chongqing University. Chongqing, China

[29] Thayumanavan Sridhar (1998) Layer 2 and Layer 3 Switch Evolution. In The Internet Protocol Journal, Future Communications Software (pp. 38-43). Cisco

[30] THE NEW STACK. SDN Series Part Eight: Comparison Of Open Source SDN Controllers
http://thenewstack.io/sdn-series-part-eight-comparison-of-open-source-sdn-controllers/

[31] Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., & Sherwood, R. (2012, April). On controller performance in software-defined networks. InUSENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE) (Vol. 54).