Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

ESTUDIO DE REDES DEFINIDAS POR SOFTWARE E IMPLEMENTACIÓN DE ESCENARIOS VIRTUALES DE PRUEBA

TRABAJO FIN DE MÁSTER

Gabriela Alejandra Salinas Jardón

2017

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

ESTUDIO DE REDES DEFINIDAS POR SOFTWARE E IMPLEMENTACIÓN DE ESCENARIOS VIRTUALES DE PRUEBA

Autor
Gabriela Alejandra Salinas Jardón

Director
David Fernández Cambronero

Departamento de Ingeniería de Sistemas Telemáticos

2017

Resumen

Las redes definidas por software son un nuevo paradigma en la implementación de redes de telecomunicaciones con menos de 10 años en el mercado y que rompen con el esquema actual en dicha área. Pese a tener casi una década, la adopción de estas ha sido lenta y se sigue trabajando para desarrollar interfaces que faciliten la transición y convivencia entre la tecnología convencional (y ampliamente extendida) y este nuevo enfoque.

En esta memoria se revisarán las bases técnicas detrás de esta tecnología como lo es la separación del plano de datos y el plano de control, el control centralizado y la programación de la red de acuerdo a las necesidades del cliente. En cada uno de esas directrices operan protocolos y equipos/sistemas especializados que cumplen con tareas bien definidas y que en conjunto forman plataformas robustas y eficaces, pues uno de los objetivos de esta nueva forma de operar es el manejo eficiente de los recursos y la no dependencia a fabricantes, quienes mantienen el mercado cautivo y limitan la investigación y desarrollo en esta área debido al beneficio económico que obtienen al comercializar soluciones propietarias.

En la primera mitad de este texto se revisará el estado del arte de las redes definidas por software, qué es y en que consiste la virtualización de redes, el papel de OpenFlow en este esquema, qué es y cuáles son las funciones del controlador y en específico Floodlight como controlador de código abierto y por último OpenvSwitch como una herramienta para la virtualización de switches OpenFlow. Lo anterior con el objetivo de introducir al lector a un panorama general de los aspectos que intervienen en la implementación de una solución como esta y prepararlo para la segunda mitad del documento donde se plantean escenarios virtualizados que permitan comprobar que los aspectos técnicos revisados se trasladan fielmente a los ambientes en producción.

Los escenarios planteados tienen la finalidad de mostrar todos los aspectos a considerar en el diseño e implementación de las redes definidas por software, observar su comportamiento e instaurar aplicaciones de alto nivel que permitan gestionar de manera eficiente los recursos de la red. Las conclusiones obtenidas con base en los resultados de los escenarios se indicarán en el último capítulo.

Para finalizar, se incluyen apéndices útiles para que, si el lector así lo desea, pueda reproducir los escenarios mostrados y constatar por sí mismo la forma de implementar y de actuar de una red de este tipo.

Palabras clave: Redes definidas por software, OpenFlow, virtualización, Floodlight, OpenvSwitch.

Abstract

Software-defined networks are a new paradigm in the implementation of telecommunication networks with less than 10 years in the market and that break with the current scheme in that area. Despite having almost a decade, the adoption of these has been slow and work continues to develop interfaces that facilitate the transition and coexistence between conventional (and widely extended) technology and this new approach.

In this report, the technical bases behind this technology will be reviewed, such as the separation of the data plane and the control plane, the centralized control and the programming of the network according to the needs of the client. In each of these guidelines, protocols and specialized equipment / systems operate that fulfill well-defined tasks and together form robust and effective platforms, since one of the objectives of this new way of operating is the efficient management of resources. This is precisely one of the objectives: to eliminate dependence on manufacturers, who maintain the market captive and limit the research and development in this area due to the economic benefit they obtain when marketing proprietary solutions.

In the first half of this text will review the state of the art of software-defined networks, what is network virtualization, the role of OpenFlow in this scheme, what is and what are the functions of the controller and Floodlight as an open source driver and finally OpenvSwitch as a tool for virtualization of OpenFlow switches. The above with the aim of introducing the reader to a general overview of the aspects involved in the implementation of a solution such as this and prepare it for the second half of the document where they pose virtualized scenarios that allow to verify that the revised technical aspects are faithfully translated to the environments in production.

The scenarios presented are intended to show all aspects to be considered in the design and implementation of software-defined networks, to observe their behavior and to establish high-level applications that allow efficient management of network resources. The conclusions obtained on the basis of the results of the scenarios will be indicated in the last chapter.

Finally, useful appendices are included so that, if the reader so wishes, he can reproduce the scenarios shown and verify for himself how to implement and act on such a network.

Key Words: Software Defined Networks, OpenFlow, virtualization, Floodlight, OpenvSwitch.

Índice general

Resumen i
Abstract iii
Índice general v
Índice de figuras viii
Índice de tablas x
1 Introducción 11
1.1 Objetivos 12
1.2 Estructura de la memoria 12
2 Redes definidas por software 13
2.1 Operación 15
2.1.1 Tablas de reenvío 16
3 Virtualización 18
3.1 Virtualización de redes 18
3.2 Virtualización de funciones de red 19
3.2.1 NFV y SDN 21
3.3 VNX 22
3.4 Mininet 23
3.5 OpenvSwitch 24
4 OpenFlow 27
4.1 Canal seguro 27
4.2 Puertos 28
4.3 Tablas de flujo 28
4.3.1 Instrucciones 30
4.3.2 Mensajes 31
4.4 Tabla *Group* 32
4.5 Tabla *Meter* 33
5 Controladores SDN 34
5.1 NOX/POX 35

5.2 OpenDaylight 35
5.3 Floodlight 37
6 Diseño e implementación de los escenarios de pruebas. 40
6.1 Escenario #1: Topología básica 40
6.2 Escenario #2: *Hardening:* listas de acceso y firewall 45
6.2.1 Listas de acceso estándar 46
6.2.2 Implementando un firewall 47
6.3 Escenario #3: Conexión a Internet, calidad de servicio y administración del ancho de banda. 50
6.4 Escenario #4: Respuesta reactiva a cambios en la topología 54
6.5 Escenario #5: *Tunneling* 58
7 Conclusiones y trabajos futuros 62
7.1 Conclusiones 62
7.2 Trabajos futuros 64
Apéndice I: Descripción e instalación de herramientas 66
A) VirtualBox 66
B) Sistema operativo invitado 67
B.1 Configuración y puesta en marcha del controlador Floodlight 69
C) cURL 70
D) Wireshark 71
E) Mininet 72
F) MiniEdit 73
G) OpenvSwitch 74
Apéndice II: Guía básica de uso MiniEdit 75
1. Interfaz 75
1.1 Barra de opciones 75
1.2 Barra de herramientas 76
1.3 Espacio de trabajo 76
2. Creación de un escenario básico 76
2.1 Definición de la topología 77
2.2 Configurar preferencias del escenario 77

2.3 Ejecutar el escenario 77
2.4 Manipulación del escenario 78
3. Configuración de controlador externo 81
Apéndice III: Scripts de los escenarios 82
1. Escenario #1: Topología básica 82
2. Escenario #2: *Hardening*: listas de acceso y firewall 83
3. Escenario #3: Conexión a Internet, calidad de servicio y administración del ancho de banda. 84
4. Escenario #4: Respuesta reactiva a cambios en la topología 86
5. Escenario #5: *Tunneling* 88
5.1 Máquina virtual 1 88
5.2 Máquina virtual 2 90
5.3 Máquina virtual 3 91
Bibliografía 93

Índice de figuras

Figura 1. Arquitectura tradicional de un conmutador 13
Figura 2. Arquitectura SDN: Separación de planos 14
Figura 3. Esquema de operación SDN 15
Figura 4. Ejemplo de APIs hacia el norte y hacia el sur 16
Figura 5. Actualización reactiva de tablas de flujo 17
Figura 6. Actualización proactiva de tablas de flujo 17
Figura 7. Virtualización: desacoplamiento de recursos físicos y lógicos 18
Figura 8. Hipervisor y su papel en el proceso de virtualización 18
Figura 9. Arquitectura VNF 20
Figura 10. Arquitectura VNX 23
Figura 11. Módulos de OpenvSwitch 25
Figura 12. Componentes de un switch OpenFlow 27
Figura 13. Puertos reservados OpenFlow 28
Figura 14. Estructura tabla de flujo 29
Figura 15. Proceso de *packet matching* 30
Figura 16. Estructura tabla *group* 32
Figura 17. Estructura tabla *meter* 33
Figura 18. Estructura y operación del controlador OpenDaylight 37
Figura 19. Estructura y operación del controlador Floodlight 38
Figura 21. Tabla de flujo de s1 40
Figura 20. Topología escenario #1, comandos dump y pingall 41
Figura 22. Logs del controlador: Conexión con s1 41
Figura 23. Captura de mensajes hello 42
Figura 24. Captura de mensajes FEATURES REQUEST y FEATURES REPLY 42
Figura 25. Captura de mensajes GET CONFIG REQUEST y GET CONFIG REPLY 42
Figura 26. Captura de mensajes ECHO_REQUEST y ECHO_REPLY 43
Figura 27. Captura de mensajes *PACKET_IN* 43
Figura 28. Captura de mensajes FLOW_MOD 44
Figura 29. Captura de mensajes PACKET_OUT 44
Figura 30. Captura de mensajes secuenciales: PACKET_IN, FLOW_MOD y PACKET_OUT 44
Figura 31. Ping de h1 a h2 y viceversa, intento #1 45
Figura 32. Ping de h1 a h2 y viceversa, intento #2 45
Figura 33. Tabla de flujo de s1 45
Figura 34. Topología escenario #2 y comando pingall 46
Figura 35. Test de conectividad hacia h5 y h6 después de aplicar ACL 46
Figura 36. Log del controlador referente a las ACL agregadas 47

Figura 37. Listando ACLs 47
Figura 38. Estatus del módulo del firewall 48
Figura 39. Logs de activación del firewall 48
Figura 40. Conexión SSH desde h1 y h3 hacia h6 49
Figura 41. Topología escenario #3 y comando pingall 50
Figura 42. Evidencia de comunicación hacia Internet 51
Figura 43. Habilitación del servicio QoS en el controlador 51
Figura 44. Resultado del comando sh ovs-ofctl show s1 52
Figura 45. Resultado del comando iperf entre h1-nat0 y h1-h3 antes de la aplicación de QoS 52
Figura 46. Resultado del comando iperf entre h1-nat0 y h1-h3 después de la aplicación de QoS 54
Figura 47. Topología escenario #4 y comando pingall 54
Figura 48. Ruta de los paquetes enviados de h5 a h11 55
Figura 49. Ruta de los paquetes enviados de h5 a h11 después de los enlaces caídos 56
Figura 50. Ping extendido entre h5 a h11 56
Figura 51. Captura de mensajes OpenFlow para modificación de tablas de flujo 57
Figura 52. Topología escenario #4 59
Figura 53. Evidencia de comunicación entre VM1 y VM2 60
Figura 54. Evidencia de comunicación entre VM1, VM2 y VM3 61
Figura 55. Captura de tráfico relacionado al encapsulamiento de datos 61
Figura 56.Interfaz de VirtualBox 66
Figura 57. Creación de máquina virtual 67
Figura 58.Asignación de memoria RAM a la máquina virtual 68
Figura 59.Asociación del disco duro virtual con la máquina virtual 68
Figura 60.GUI de Floodlight 70
Figura 61. Interfaz de Wireshark 71
Figura 62. Interfaz de Mininet 72
Figura 63. Interfaz de MiniEdit 73
Figura 64. Interfaz de MiniEdit 75
Figura 65. Submenú Preferences 75
Figura 66. Detalle de la barra de opciones de MiniEdit 76
Figura 67. Creación de un escenario básico con MiniEdit 77
Figura 68. Configuración de Preferencias del escenario en MiniEdit 77
Figura 69. Interfaz de Mininet en segundo plano 78
Figura 70. Comando *dump* 78
Figura 71. Comando *net* 78
Figura 72. Comando *nodes* 78
Figura 73. Comando *xterm* <*hostname*> 79

Figura 74. Comando *pingall* 79
Figura 75. Comando sh ovs-ofctl dump-flows <hostname> 79
Figura 76. Comando *help* 79
Figura 77. Ejecución de escenario Mininet desde script 80
Figura 78. Ejemplo de script de Mininet 80
Figura 79. Configuración de controlador remoto a través de MiniEdit 81
Figura 80. Configuración de controlador remoto a través del script del escenario 81

Índice de tablas

Tabla 1: Ciclo de vida escenario VNX 22
Tabla 2. Comparación entre controladores NOX/POX, ODL y Floodlight 35
Tabla 3. Listado de switches del escenario 5, su DPID y número de puerto 57
Tabla 4. Características de la aplicación instalada VirtualBox 66
Tabla. 5 Características de la aplicación instalada Sistema Operativo invitado 67
Tabla 6. Características de la aplicación instalada cURL 70
Tabla 7. Características de la aplicación instalada Wireshark 71
Tabla 8. Características de la aplicación instalada Mininet 72
Tabla 9. Características de la aplicación instalada MiniEdit 73
Tabla 10. Características de la aplicación instalada OpenvSwitch 74

1 Introducción

En los últimos años, el crecimiento que ha tenido Internet ha sido acelerado. Detrás de ésta y de todos los servicios que se ofrecen en ella se encuentran las redes de comunicaciones, mismas que con la finalidad de adaptarse a las crecientes necesidades de los usuarios también ha crecido en tamaño y complejidad. Sin embargo, dichas redes en cuanto más grandes y más complejas se tornan, conllevan una reducción importante en la funcionalidad, que a la postre deriva en una considerable baja del rendimiento y por ende de la velocidad de procesamiento la misma.

La arquitectura tradicional de las redes de comunicaciones concentra en los equipos de red los planos de control y de datos, lo que implica, entre otras cosas, dificultad para adaptarse a cambios en la red y por ende, escasa flexibilidad, mayores puntos de fallo, equipo especializado costoso (y en ocasiones, por encima de las necesidades del usuario), grupo líder de fabricantes con tecnología y protocolos propietarios que conlleva a dependencia a cierto fabricante, etc. El contar con sistemas cerrados dentro de Internet impacta en la innovación que el área pudiera tener, aunado a que durante un cuarto de siglo los principales fabricantes han mantenido el control del mercado y dada la poca competencia, el encarecimiento de las soluciones ofrecidas por estos.

Adicional a lo anterior, la arquitectura tradicional de las redes es estática, es decir, no se puede adaptar de manera dinámica a las crecientes necesidades de los sistemas finales. Tratando de solventar estas deficiencias se ha introducido un nuevo paradigma en cuanto a esta área se refiere: las redes definidas por software o SDN[1].

Hoy en día los sistemas distribuidos son ampliamente explotados debido a la gran cantidad de ventajas que ofrecen, como son: disponibilidad, tolerancia a fallos, escalabilidad, elasticidad, reducción de costos, etc. Dentro de las redes de comunicaciones, las SDN tornan en un sistema de control distribuido la propia red de comunicaciones al separar el plano de control y de datos, lo que facilita la personalización de las redes, reduce tiempos de configuración y de despliegue, centraliza el control, reduce costos, simplifica y facilita la administración, optimiza recursos, etc. Todas estas características conllevan a que las SDN permitan redes más rápidas, fiables, ágiles, seguras, independientes de fabricantes y que promueven la investigación y el desarrollo al basarse en estándares y protocolos de código abierto como lo es OpenFlow.

Los controladores SDN son la parte de la red que operará el plano de control, de éstos existen gran variedad de opciones tanto comerciales como de código abierto. Contar con opciones gratuitas y de código abierto permite probar y tener un primer acercamiento con esta tecnología sin invertir el capital que implicaría un controlador SDN comercial. Entre

[1] SDN: Software-Defined Networks

los controladores de código abierto se destacan: OpenDaylight, OpenContrail, Floodlight, Ryu, FlowVisor, etc.

Actualmente, las redes definidas por software ya están siendo implementadas en diversas redes corporativas, de proveedores de servicios y en centros de datos, así mismo se prevé una mayor adopción en los próximos años que se acompaña del apoyo de grandes corporaciones como IBM, Cisco, Juniper, Ericsson, HP, etc. Quienes han sido capaces de observar la oportunidad de negocio que conlleva la adopción de este nuevo enfoque.

1.1 Objetivos

Esta memoria tiene como objetivo realizar un estudio de las tecnologías SDN analizando la terminología asociada, estructura, componentes y los procesos implementación y operación de las redes en este contexto. Así mismo, se propondrán escenarios de pruebas virtualizados y donde se hará uso de protocolos y controladores de software libre, lo anterior con la finalidad de poder observar de manera práctica los conceptos revisados en el marco teórico de este trabajo.

1.2 Estructura de la memoria

Los capítulos 2, 3 y 4 de este trabajo se concentrarán en la revisión del estado del arte de tres temas: redes definidas por software, virtualización y el protocolo OpenFlow, mientras que en el capítulo 5 se revisarán los aspectos referentes a los controladores SDN. Estos antecedentes son fundamentales para el desarrollo del capítulo 6 donde se diseñaran e implementaran escenarios de prueba que permitan corroborar los aspectos técnicos de este paradigma en ambientes en producción. En el capítulo 7 se expondrán las conclusiones a las que se llegaron tras las pruebas ejecutadas en los escenarios y se indicarán algunos proyectos y líneas de investigación a futuro.

Por último, se incluyen apéndices que permitan al lector reproducir los escenarios descritos en este trabajo: En el apéndice I se listaran las herramientas utilizadas en el desarrollo de este escrito, sus características y los pasos a seguir para su instalación, en el apéndice II, se presenta una guía para el uso de una de las herramientas utilizadas: MiniEdit y en el apéndice III se detallan los scripts de los escenarios propuestos.

2 Redes definidas por software

La base en la que se fundamentan las redes actuales y en general la Internet, tuvo su concepción en el último cuarto del siglo XX, sin embargo el crecimiento que se ha tenido en cuanto a usuarios y generación de datos en los últimos años ha puesto en evidencia las limitaciones de dicho esquema: direcciones IPv4[2] escasas, dispositivos de red (switches, routers, balanceadores de carga, etc.) cursando cada vez más tráfico, necesidad de calidad de servicio con base en servicios diferenciados, redes estáticas (no son capaces de adaptarse dinámicamente a las necesidades de recursos en cuanto a tráfico o usuarios) y de administración manual (necesidad de reconfiguración de la red frente a cambios en la topología), soluciones comerciales cerradas y con software propietario, que limita el control que tiene el usuario sobre la red y la innovación en cuanto a desarrollo e investigación en el área.

Dentro de la arquitectura tradicional de la redes de comunicaciones existen dispositivos encargados de recibir paquetes de datos en alguno de sus puertos y determinar, con base en la información con la que cuenta sobre la propia red, el o los puertos a través de los cuales lo retransmitirá. Los dispositivos encargados de llevar a cabo esta función son los conmutadores de tráfico, que internamente se componen de dos capas o planos (Ver Fig. 1):

1. Plano de datos: Compuesto por los puertos del dispositivo, a través de los cuales recibe y reenvía los paquetes, además del búfer de almacenamiento de los mismos. Se encarga del manejo de colas de salida, modificación de cabeceras de control y del reenvío de paquetes con base en la tabla de reenvío.
2. Plano de control: A cargo del sistema operativo del dispositivo, cuya función principal es mantener actualizada la tabla de reenvío con base en los protocolos de ruteo y control que maneja (software específico).

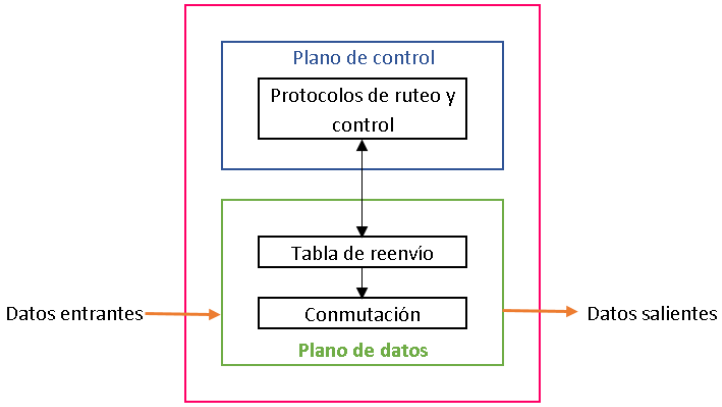


Figura 1. Arquitectura tradicional de un conmutador

[2] IPv4: Internet Protocol version 4

Las SDNs son un paradigma que rompe el esquema tradicional de las redes y que se basan en las siguientes características:

1. Separación del plano de datos y del plano de control: Implementación de planos en hardware independiente.
2. Control centralizado: El equipo donde se implementa el plano de control topológicamente forma una red estrella junto con los equipos que implementan el plano de datos.
3. Red programable: Ofrece la flexibilidad de programar la red de acuerdo a las necesidades del negocio.
4. Automatización de la red: Busca evitar la configuración manual de la red ante los cambios en esta.
5. Redundancia: Intenta disminuir la necesidad de implementar redundancia en todos los niveles de la red: acceso, distribución y núcleo.
6. Infraestructura genérica: Liberar al cliente de adquirir dispositivos de fabricantes o de características muy específicas, privilegiando el uso de soluciones de código abierto.

La división en planos hace más eficiente la conmutación de los paquetes, ya que sí la información necesaria para procesarlos se encuentra en la tabla de reenvío, no es necesario la intervención del plano de control, además el desacoplamiento de los planos implica que cada uno resida en dispositivos independientes, mientras el plano de datos residirá en el dispositivo denominado switch SDN, el plano de control residirá en el controlador (Ver Fig. 2). Invariablemente, se deberá contar con una API[3] que permita la comunicación entre ambos planos, este punto se abordará con mayor detalle en el capítulo 4.

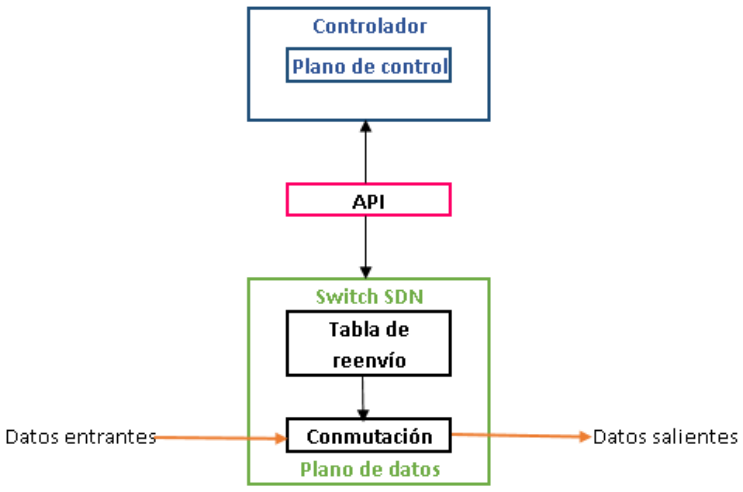


Figura 2. Arquitectura SDN: Separación de planos

[3] API: Application Programming Interface

Una de las principales ventajas del control centralizado es que ante cambios en la red, solo se deberán realizar ajustes necesarios en el controlador, además la programabilidad permite incluir las funcionalidades que la red requiera: redes privadas virtuales, balanceo de carga, controles de acceso, políticas de seguridad, etc. Teniendo así una solución flexible, que se amolde a las necesidades del negocio.

2.1 Operación

En la figura 3 se muestra el esquema de operación de una red SDN, donde podemos apreciar con mayor detalle la separación de planos:

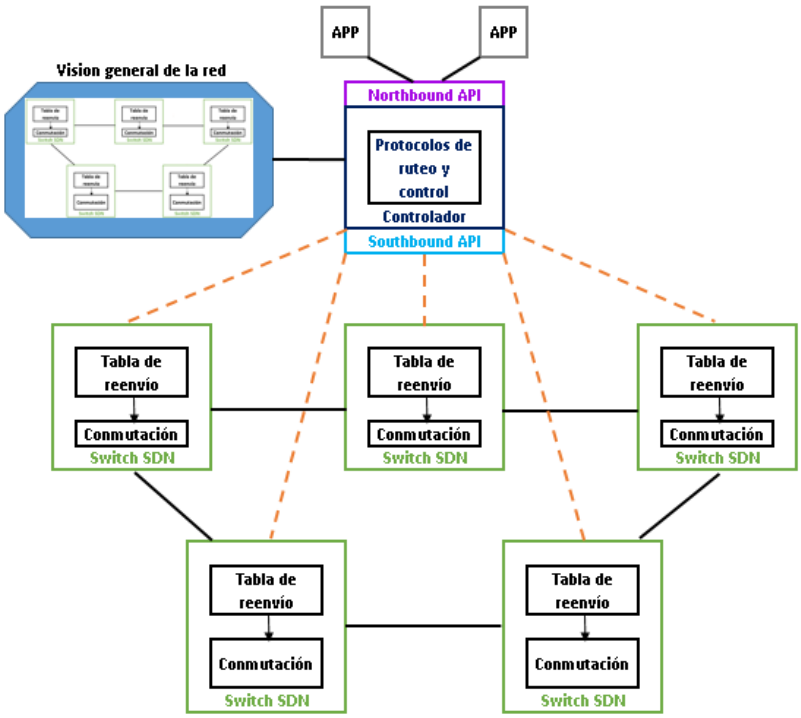


Figura 3. Esquema de operación SDN

Haciendo un acercamiento al controlador, se muestran tres elementos adicionales:

1. Vista global de la red: El controlador cuenta con una visión general de la red, lo que le permite tomar decisiones con base en el panorama general.
2. *Northbound* API (API hacia el norte): Compuesta por aplicaciones de alto nivel que permiten darle a la red la funcionalidad deseada: ingeniería de tráfico, balanceo de carga, administración de *cloud*, etc. Es también conocida como la capa de aplicación.
3. *Southbound* API (API hacia el sur): Compuesta por interfaces de software que permiten la comunicación entre el controlador y los switches SDN.

Existen gran variedad de APIs hacia el norte y hacia el sur en el mercado, tanto comerciales como de código abierto, como ejemplo de estas últimas tenemos las descritas en la siguiente figura:

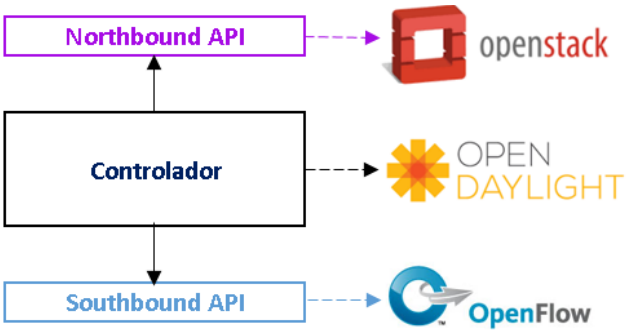


Figura 4. Ejemplo de APIs hacia el norte y hacia el sur

A) Openstack: Es una plataforma de orquestación que permite implementar servicios de computación en la nube bajo un esquema IaaS[4] . La red SDN abstrae la complejidad de la infraestructura de comunicaciones a aplicaciones como esta, para que la interacción sea lo más transparente posible. Ejemplos de APIs hacia el norte similares a Openstack tenemos: vCloud Director de IBM y CloudStack de Apache. Por otro lado, APIs relacionadas a la gestión de infraestructura están: Puppet, Chef y CFEngine, mientras que Ansible es una herramienta para autoconfiguración de dispositivos de red.

B) OpenDaylight: Es un software que se instala en el equipo que fungirá como controlador, permitiéndole llevar a cabo las funciones de gobierno del plano de control. Otros controladores de código abierto son: OpenContrail, Floodlight, FlowVisor, etc. Mientras que del lado de las soluciones comerciales se encuentran: APIC[5] de Cisco, NSX de VMware, VAN[6] de HP, etc.

C) OpenFlow: Es un protocolo que permite la comunicación entre los controladores y switches SDN. Además de este, existen otros protocolos que se pueden implementar como API hacia el sur: OvSDB[7] de VMWare, NETCONF del IETF[8], SNMP[9], BGP[10], etc.

ONF[11] es el organismo regulador en cuanto a SDN se refiere, mismo que trabaja arduamente para normalizar y estandarizar el desarrollo de APIs y controladores, esto con la finalidad de asegurar uno de los pilares de este esquema: la interoperablidad entre las capas sin problemas de compatibilidad entre fabricantes.

2.1.1 Tablas de reenvío

Las tablas de reenvío en redes SDN se basan en el tratamiento de flujos de datos. Un flujo de datos es un conjunto de paquetes transferidos desde el origen a su destino. Con base en las reglas definidas por el administrador de red, se describen las acciones a tomar

[4] IaaS: Infrastructure as a Service
[5] APIC: Application Policy Infrastructure Controller
[6] VAN: Virtual Application Networks
[7] OvSDB: Open vSwitch Data Base
[8] IETF: Internet Engineering Task Force
[9] SNMP: Simple Network Management Protocol
[10] BGP: Border Gateway Protocol
[11] ONF: Open Networking Foundation

para cada flujo; es importante indicar que los flujos son unidireccionales, en caso de requerir comunicación en sentido contrario se tendrán que definir las reglas necesarias para ese otro flujo de datos.

Existen dos tipos de actualización de tablas de reenvío:

1. Reactivo (Ver Fig. 5): El switch SDN recibe un paquete (1) que no coincide con ninguna entrada en la tabla de flujo, por lo que notifica al controlador (2) que a su vez envía una actualización a la tabla de flujo (3) que le permita al switch manejar el paquete (4).

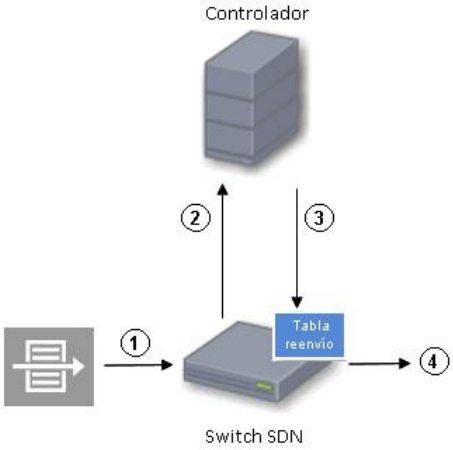


Figura 5. Actualización reactiva de tablas de flujo

2. Proactivo (Ver Fig. 6): El controlador detecta algún cambio en la red: falla de equipo, caída o saturación de enlaces, etc. (1) envía una actualización a las tablas de flujo de los switches (2) minimizando así los tiempos de procesamiento ya que la actualización se lleva a cabo previo a la llegada de paquetes (3) y el switch es capaz de procesarlo (4) adecuadamente.

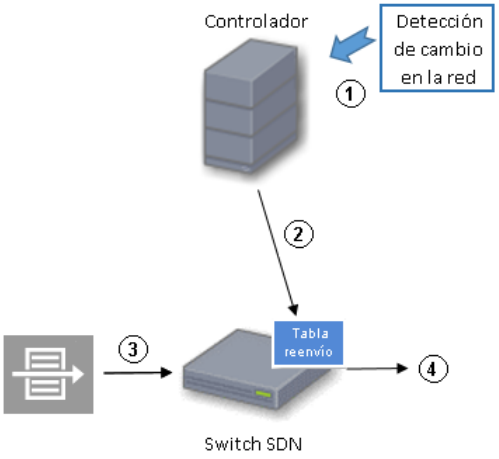


Figura 6. Actualización proactiva de tablas de flujo

3 Virtualización

La virtualización consiste en desacoplar los recursos lógicos de la infraestructura física que lo soporta (Ver Fig. 7).

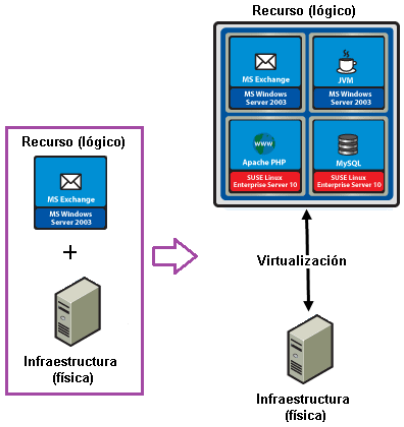


Figura 7. Virtualización: desacoplamiento de recursos físicos y lógicos

El aumento en el uso de la virtualización se debe en gran medida al impulso debido a la reducción en los costos de hardware, software cada vez más eficiente y a las limitaciones que presentan las soluciones convencionales, pues a diferencia de éstos, los servicios virtualizados buscan el uso eficiente y la compartición de recursos, la reducción de costos y la personalización de las plataformas de acuerdo a las necesidades del cliente.

Uno de los compontes básicos en las plataformas de virtualización es el hipervisor, que es un software de propósito específico que permite administrar varias instancias de servicios, inclusive distintos, dentro de una misma infraestructura física manteniéndolas aisladas entre sí (Ver Fig. 8) Es importante mencionar, que a nivel operativo, las instancias trabajan de manera secuencial y no paralela, por lo que en servicios sensibles al retardo o en tiempo real el desempeño se podría ver afectado.

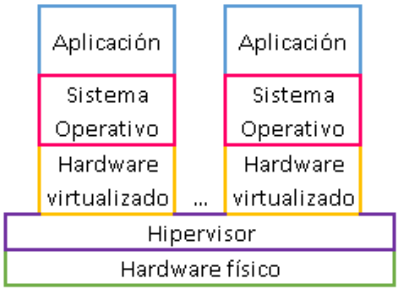


Figura 8. Hipervisor y su papel en el proceso de virtualización

3.1 Virtualización de redes

En este trabajo nos concentraremos en la virtualización de redes de comunicaciones, bajo esta premisa, podemos definir una red virtual como una entidad administrable basada en

software que combina recursos de red, tanto de hardware como de software mismo y las funcionalidades que estas proveen proporcionando recursos individuales y dedicados de un conjunto más grande de recursos, permitiendo así el control y personalización que se desee.

Existen dos tipos de virtualización en redes:

1. Externa: Se basa en la creación de redes virtuales basados en redes físicas (*Overlay*). A través de protocolos y aplicaciones se simula una única red, que está compuesta por redes heterogéneas de menor tamaño que pueden ser tanto públicas como privadas.

Ejemplos de protocolos y aplicaciones para llevar a cabo este tipo de virtualización tenemos: MPLS[12], VLAN[13], VPN[14], IPSec[15], etc.

2. Interna: A diferencia del caso anterior, en la virtualización interna se cuenta con dispositivos de propósito general a través de los cuales se implementan funcionalidades de red basadas en software.

Es de vital importancia contar con mecanismos que permitan aislar una red de otra, a través de la correcta implementación de políticas que aseguren los flujos de datos dentro del dominio correspondiente, estos mecanismos se revisarán con mayor detalle en los siguientes capítulos.

3.2 Virtualización de funciones de red

La virtualización de funciones de red (NFV[16]) consiste en virtualizar dispositivos de red dedicados, como son: routers, switches, balanceadores de carga, firewalls, etc. en infraestructura de propósito general (servidores x86).

NFV está documentado en el ETSI GS NFV 003 V1.2.1 que busca ser un marco de referencia para la creación de soluciones estándar. Así mismo, describe los elementos que conforman una solución de este tipo (Ver Fig. 9):

1. VNF[17]: Funciones de red virtuales, son los elementos que se han virtualizado, que a su vez están compuestos por VNFC[18].
2. NFVI[19]: Elementos de hardware y software sobre los que se despliegan los VNF.
3. MANO[20]: Plataforma de administración y orquestación para el aprovisionamiento de VNFs, su configuración o la gestión de la infraestructura que las soporta.

[12] MPLS: Multiprotocol Label Switching
[13] VLAN: Virtual LAN
[14] VPN: Virtual Private Network
[15] IPSec: Internet Protocol SEcurity
[16] NFV: Network Function Virtualization
[17] VNF: Virtual Network Functions
[18] VNFC: VNF Components
[19] NFVI: NFV Infraestructure
[20] MANO: Management and Orchestration

A su vez está compuesta por tres subsistemas:

- NFVO[21]: Se utiliza para el despliegue de nuevos servicios de red, la gestión del su ciclo de vida, validación y autorización de solicitudes de recursos NFVI y gestión de políticas para instancias.
- VNFM[22]: Proporciona la gestión del ciclo de vida de las instancias de VNF y la coordinación y adaptación general para la configuración y la presentación de informes de eventos entre NFVI, el EMS[23] y el NMS[24].
- VIM[25]: Controla y administra los recursos de computación, almacenamiento y red de la NFVI dentro del subdominio de infraestructura de un operador. Es responsable de la recopilación y envío de mediciones de rendimiento y eventos.

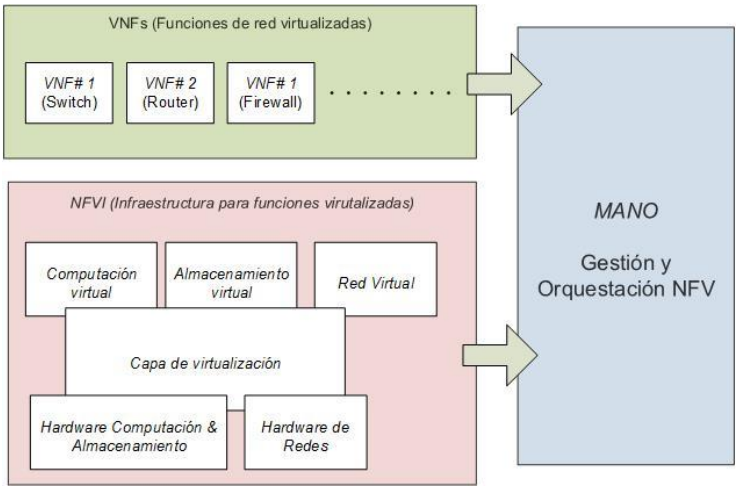


Figura 9. Arquitectura VNF

En el documento previamente citado, la ETSI[26] lista algunas de las funciones de red pueden ser virtualizadas: routers, switches, optimizadores de tráfico, firewalls, dispositivo o función de inspección profunda de paquetes, herramientas de monitoreo de performance de la red, sistemas detectores y preventores de intrusos, etc. Pareciera ser que podemos emplear NFV para virtualizar cualquier servicio, sin embargo es importante considerar, como se mencionó con anterioridad, que los servicios sensibles al retardo o en tiempo real se pueden ver afectadas en el desempeño sí aplicamos este esquema, ejemplos de dichos servicios son: seguridad extremo a extremo, telefonía IP[27], optimización en WAN[28], videoconferencia/telepresencia y aplicaciones en tiempo real.

[21] NFVO: NFV Orchestrator
[22] VNFM: VNF Manager
[23] EMS: Element Management Systems
[24] NMS: Network Management Systems
[25] VIM: Virtualized Infrastructure Manager
[26] ETSI: European Telecommunications Standards Institute
[27] IP: InternerProtocol
[28] WAN: Wide Area Network

OPNFV[29] es una plataforma de código abierto que se basa en el marco de la ETSI para el desarrollo de aplicaciones VNF, su implementación y pruebas, incluyendo además, las instancias de NFVI y VIM para su gestión. Su principal objetivo ser un factor que coadyuve a una rápida y cada vez mayor adopción y despliegue de este paradigma.

Grandes empresas, líderes en el mercado de las telecomunicaciones como son: Cisco, Juniper, HP, Dell, Brocade, Checkpoint, Fortinet, Ericcson, Citrix, IBM y un largo etcétera han denotado su interés en NFV e impulsan el desarrollo de aplicaciones de este tipo dentro de sus organizaciones, siendo que muchas de ellas ya cuentan con soluciones NFV tanto para implementarse en centros de datos como en redes de proveedores de servicios.

3.2.1 NFV y SDN

Tanto NFV como SDN han revolucionado las bases del *networking* tradicional, ofreciendo una alternativa viable a las limitaciones de las redes actuales y ofreciendo múltiples ventajas, propias de los sistemas virtualizados y distribuidos respectivamente.

Ventajas:

1. Escalabilidad: Permite crecimiento de las plataformas cuando los recursos tanto lógicos como físicos se ven rebasados, técnicamente no existen limitaciones de crecimiento.
2. Flexibilidad: Capacidad de asignar más recursos al área que lo necesita y adaptarse a las necesidades del negocio.
3. Portabilidad: Facilidad de migración a otro equipo o posición dentro de la red.
4. Alta disponibilidad, tolerancia y rápida recuperación ante fallos: Al contar con varios dispositivos capaces de realizar la misma función, es posible prescindir de equipos debido a fallas.
5. Eficiencia: Mejor utilización de los recursos.
6. Reducción de gastos: Los gastos de implementación (CAPEX) y operación (OPEX) se reducen al contar con el hardware genérico y no subutilizados.
7. Gestión: Facilita la gestión de los sistemas al desacoplar la parte lógica de la física.
8. Reduce dependencia a fabricantes: Es posible hacer uso de infraestructura genérica, evitando la adquisición de equipos especializados, que son, casi siempre, costosos.

NFV ofrece la capacidad de concurrencia y paralelización, al generar varias instancias de servicios sobre una misma infraestructura, además varios procesos se podrán ejecutar de manera simultánea.

Es importante mencionar que NFV y SDN son tecnologías distintas que contribuyen al modelo de red basada en software y que pueden implementarse en conjunto con la finalidad de complementarse.

[29] OPNFV: Open Plataform for NFV

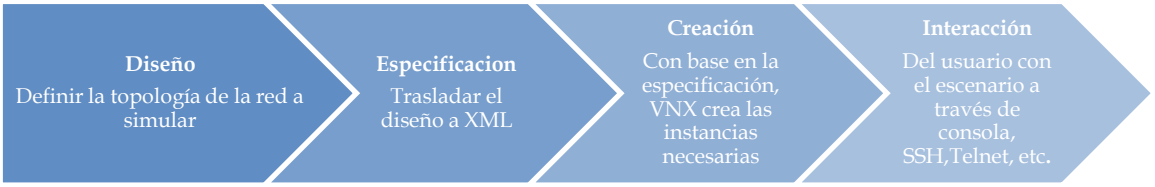
3.3 VNX

VNX[30] es una herramienta para la creación automática, ejecución e interacción de escenarios que consisten en máquinas virtuales interconectadas a través de redes virtuales que se ejecutan dentro de un único equipo físico, teniendo la posibilidad de interconectarlos con equipos o redes externas.

Fue desarrollada por el Departamento de Ingeniería Telemática de la Universidad Politécnica de Madrid, y está orientado a la docencia, pues ofrece un entorno donde es posible experimentar con protocolos, topologías, servicios y tecnologías de manera controlada y que no implica el desembolso de recursos económicos al evitar adquirir infraestructura física.

En la siguiente tabla, se describe el ciclo de vida al generar un escenario con VNX:

Tabla 1: Ciclo de vida escenario VNX



En la figura número 10 se especifica la arquitectura de VNX, donde se pueden apreciar los siguientes módulos:

- UML[31]: Es el lenguaje de modelado y al igual que su antecesor VNUML[32], se basa en la especificación del escenario en lenguaje XML[33].

El intérprete del lenguaje analiza la descripción para posteriormente generar, ejecutar y administrar el escenario.

- Libvirt: Es el API estándar para virtualización en Linux y permite la integración de otras plataformas de virtualización (Xen, KVM[34], UML, VirtualBox o VMware)
- Dynamips: Permite emular el sistema operativo IOS[35] de Cisco.

[30] VNX: Virtual Network over linuX
[31] UML: Unified Modeling Language
[32] VNUML: Virtual Network User Mode Linux
[33] XML: eXtensible Markup Language
[34] KVM: Kernel-based Virtual Machine
[35] IOS: Internetwork Operating System

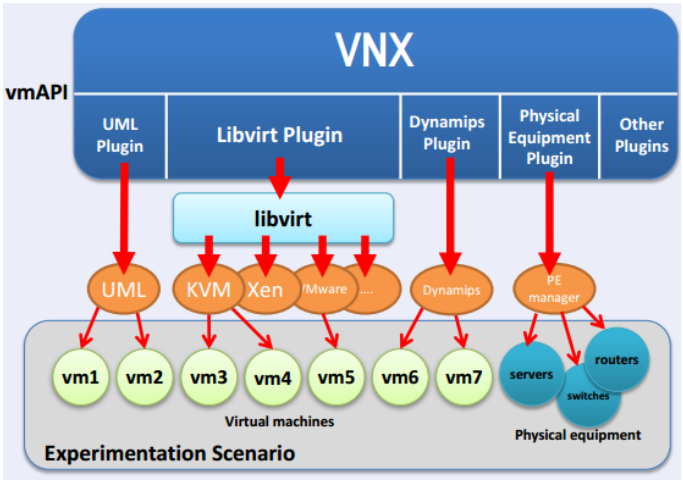


Figura 10. Arquitectura VNX

VNX se basa en VNUML, del cual ha tomado algunas características y adicionado nuevas funcionalidades que hacen de VNX una herramienta más robusta, como son:

- Olive: Permite emular el sistema operativo JunOS[36] de Juniper
- Autoconfiguración y ejecución de comandos a través de un CDROM dinámico: XML con parámetros de configuración: nombre de la máquina, interfaces, direcciones IP, capacidades de enrutamiento, etc.
- Gestión individual de máquinas virtuales.

Las operaciones básicas que se pueden ejecutar en VNX son:

- Construcción de escenarios: con base en los archivos de descripción de la red generados por el usuario en XML.
- Liberación de escenarios: liberación de recursos del dispositivo al cerrar los escenarios.
- Ejecución de secuencias comandos: se utiliza para iniciar o detener servicios en las máquinas virtuales.

3.4 Mininet

Es una herramienta de código abierto que permite emular redes de comunicaciones de una forma sencilla y cuyo objetivo es facilitar la enseñanza, investigación y desarrollo en el área de las telecomunicaciones.

Entre las principales características de esta plataforma se encuentran:

- Disponible solo para sistemas operativos Linux.

[36] JunOS: Juniper Operating System

- Cuenta con interfaz GUI[37] o CLI[38].
- Es una herramienta ligera, pues no demanda gran cantidad de recursos al sistema (CPU[39], memoria RAM[40], etc.)
- Facilita el diseño de redes complejas, lo que la convierte en una herramienta muy potente.
- Permite simular entornos híbridos, es decir, redes de datos tradicionales y redes definidas por software.
- Es posible simular dispositivos finales (computadores, servidores, máquinas virtuales, etc.), switches (convencionales y SDN), routers, controladores y enlaces.
- Los dispositivos finales están basados en Linux, por lo que en ellos se puede simular cualquier cosa y tan compleja como el propio sistema lo soporte.
- Los switches y controladores SDN soportan varias versiones de OpenFlow (desde la 1.0 hasta la 1.3)
- Ofrece varios tipos de switches SDN para su implementación.
- Incluye un controlador por defecto (POX), sin embargo, sí el usuario así lo desea, puede conectar la topología con controladores externos.
- Es posible conectar el escenario a Internet.
- Los escenarios creados pueden ser ejecutados en tiempo real, siendo posible evaluar su rendimiento y además, pueden ser accesados de manera concurrente.
- Ofrece la capacidad de interconectar los servicios simulados con infraestructura real.
- Existe amplia documentación, foros de ayuda, wikis, tutoriales, etc.

Por todo lo anterior y valorando las características frente a otras opciones de simulación, se decidió emplear esta herramienta para la generación de los escenarios que se describirán en el capítulo 6 de este texto, así mismo, los pasos a seguir para su instalación se muestran en el apéndice I.

3.5 OpenvSwitch

OpenvSwitch (OVS) es una plataforma *open source* para sistemas basados en Linux que permite la virtualización de switches multicapa con calidad de producción. Está diseñado para facilitar la automatización de la red a través de extensiones programables y el soporte a interfaces y protocolos de administración estándar como NetFlow, sFlow, LAC, SPAN[41], etc.

[37] GUI: Graphical User Interface
[38] CLI: Command Line Interface
[39] CPU: Central Processing Unit
[40] RAM: Random Access Memory
[41] SPAN: Switched Port Analyzer

La estructura de OVS es modular (Ver Fig. 11) y consta de cuatro de ellos, cada uno incluye servicios enfocados a facilitar la administración: Seguridad, Monitoreo, QoS[42] y Control automatizado.

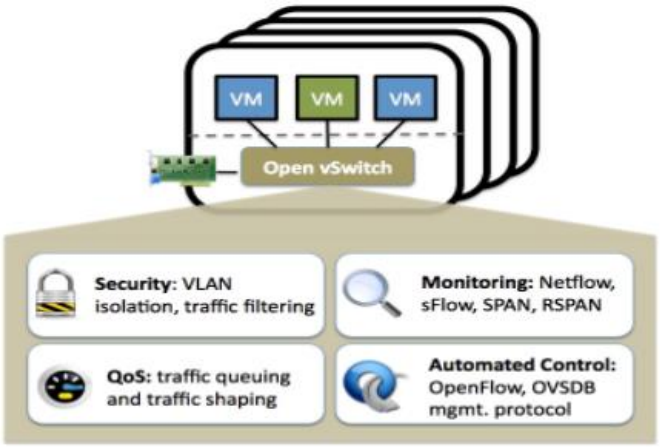


Figura 11. Módulos de OpenvSwitch

Entre las características principales de OVS están:

- Redes dinámicas: Incluye una serie de características que le permiten responder y adaptarse a cambios en la red, basándose en la información de la topología con la que cuenta y que almacena en la base de datos OVSDB.
- Etiquetas lógicas: Las etiquetas lógicas identifican de manera única a cada elemento de la red a través del etiquetado de éstas e incluye varios métodos para especificar y mantener las reglas de etiquetado.
- Integración de hardware: Permite la integración de elementos de red físicos, encargándose del reenvío del tráfico entre estos y las máquinas virtuales.
- Movilidad: Cada entidad de la red (sea máquina virtual o infraestructura física) debe ser fácilmente identificable y capaz de ser migrado sin representar mayores dificultades.

OVS se encuentra precargado de manera nativa en sistemas basados en Linux y es una de las soluciones más utilizadas al momento de implementar redes SDN ya que es una herramienta sencilla de utilizar, replica muchas de las características de un switch de capa 2 convencional y agrega potentes funcionalidades que permiten desarrollar entornos robustos capaces de ser implementados en entornos empresariales sin afectar su desempeño.

OpenvSwitch a su vez está conformado por cuatro módulos que permiten gestionarlo de manera eficiente:

[42] QoS: Quality of Service

1. **ovs-vsctl**: Se utiliza para la configuración y visualización de las operaciones de conmutación, configuración de puertos, generar y eliminar puentes, *taggeo* VLAN, etc.

Los comandos básicos de este módulo son:

- ovs-vsctl –v: Muestra la versión actual de OpenvSwitch.
- ovs-vsctl show: Imprime un resumen de la configuración del switch.
- ovs-vsctl list-br: Lista los puentes configurados en el switch.
- ovs-vsctl list-ports *switch*: Lista los puertos del switch indicado.
- ovs-vsctl list interface: Lista las interfaces del switch.
- ovs-vsctl add-br *switch*: Crea un puente en la base de datos del switch.
- ovs-vsctl add-port *switch interfaz*: Enlaza una interfaz (física o virtual) a un switch.
- ovs-vsctl add-port *switch interfaz* tag=*num_vlan*: Convierte el puerto a un puerto de acceso de la VLAN especificada.
- ovs-vsctl set interface *interfaz* type=patch options:peer= *interfaz:* Se utiliza para crear los puertos de conexión para conectar dos o más switches.

2. **ovs-ofctl**: Permite la administración y monitoreo de switches SDN.

Los comandos básicos de este módulo son:

- ovs-ofctl show *switch*: Muestra funciones OpenFlow y descripciones de los puertos para el swtich indicado.
- ovs-ofctl snoop *switch:* Muestra el tráfico hacia y desde el switch.
- ovs-ofctl dump-flows *switch flujo*: Lista los flujos correspondientes al switch indicado.
- ovs-ofctl dump-ports-desc *switch:* Imprime las estadísticas de los puertos del switch indicado.
- ovs-ofctl dump-tables-desc *switch*: Imprime las descripciones de las tablas que pertenecen al switch especificado.
- ovs-ofctl dump-ports-desc: Detalla las características de conectividad del puerto.
- ovs-ofctl add-flow *switch flujo*: Añade un flujo estático para el switch especificado.

3. ovs-ofctl del-flows *switch flujo*: Eliminar las entradas de la tabla de flujo de switch indicado. Si se omite el flujo, se eliminarán todos los flujos.

4. **ovs-dpctl**: Empleado para gestionar los flujos de la red y funciona de manera similar al módulo anterior, salvo que muestra los flujos cuya coincidencia haya sido exacta y muestra los paquetes que han atravesado el sistema.

5. **ovs-appctl**: Utilizado para listar y gestionar los *daemons* propios de OVS.

Estos dos últimos módulos son menos utilizados, ya que para la gestión básica de OVS bastan con los dos primeros y mantener la configuración por defecto de *dpctl* y *appctl* bastan para el manejo del mismo.

4 OpenFlow

Como se revisó en los capítulos anteriores, OpenFlow es un protocolo cuyo objetivo es permitir la comunicación entre el controlador y los switches SDN a través de la definición de los mensajes a intercambiar y el formato de estos. La versión más reciente es la 1.5.1 que fue lanzada en marzo del 2015, misma que se puede implementar en switches OpenFlow exclusivos o switches híbridos, es decir, switches tradicionales que además soportan este protocolo.

En la siguiente figura se muestran los componentes principales de un switch OpenFlow, entre los que destacan el canal seguro, los puertos, las tablas de flujos y las tablas *group* y *meter*.

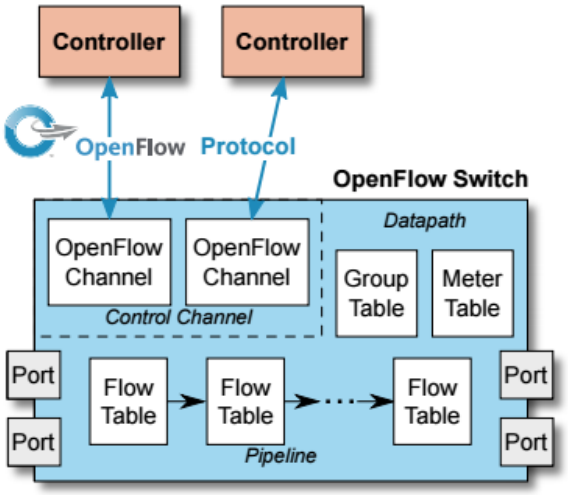


Figura 12. Componentes de un switch OpenFlow

4.1 Canal seguro

La comunicación entre el controlador y los switches SDN se lleva a través de un canal seguro, cifrado asimétricamente basado en SSL[43]/TLS [44]para autenticar ambos extremos de la comunicación, reduciendo así vulnerabilidades en cuanto a ataques hacia las comunicaciones en tránsito. Dicho canal es iniciado por el switch en cuanto es encendido y el puerto predeterminado para establecer la comunicación es el 6653 de TCP[45].

Los switches y el controlador se autentican mutuamente intercambiando certificados firmados por una clave privada. Cada switch cuenta con dos certificados: un certificado del controlador, que permite autenticar al controlador y un certificado de conmutación para autenticarse en el controlador.

[43] SSL: Secure Sockets Layer
[44] TLS: Transport Layer Security
[45] TCP: Transmission Control Protocol

El tráfico en el canal seguro no se valida en las tablas de reenvío, por lo que el switch debe ser capaz de distinguir entre tráfico conmutable y el correspondiente a la comunicación con el controlador.

4.2 Puertos

Los puertos OpenFlow son las interfaces de red utilizados para pasar paquetes entre el procesamiento de OpenFlow y el resto de la red. Existen cuatro tipos de puertos:

- Estándar: Son los puertos de entrada y salida. Estos pueden ser físicos o lógicos, (el puerto LOCAL pertenece a esta categoría) y pueden ser utilizados como puertos de entrada o salida de paquetes.
- Físicos: Son puertos de conmutación que corresponden a una interfaz de hardware del switch.
- Lógicos: Son puertos de conmutación que no corresponden explícitamente a una interfaz de hardware del switch. Estos pueden incluir encapsulación de paquetes y asignarse a varios puertos físicos.
- Reservados: Definen acciones de reenvío genéricas como son:

Puerto	Descripción
ALL	Reenviar un paquete a todos los puertos excepto por el que se recibió.
CONTROLLER	Reenviar el paquete al controlador Como puerto de salida: puerto a través del cual el switch envía paquetes encapsulados al controlador
TABLE	Enviar el paquete a la primera tabla de flujo para su procesamiento
IN_PORT	Se utiliza para enviar el paquete hacia fuera a través de su puerto de la entrada.
ANY	Puerto especial utilizado para las solicitudes OpenFlow que permite que la instancia de solicitud se aplique a cualquiera y todos los puertos.
UNSET	Se utiliza para indicar que el puerto de salida no se ha especificado en el conjunto de acciones.
LOCAL	Representa la pila de red local del switch y se utiliza para permitir que entidades remotas interactúen con él.
NORMAL	Es un puerto a través del cual se envían los paquetes procesados convencionalmente en capa dos o tres
FLOOD	Reenviar un paquete a todos los puertos excepto por el que se recibió.

Figura 13. Puertos reservados OpenFlow

4.3 Tablas de flujo

Las tablas de flujo están compuestas por siete secciones bien diferenciadas (Ver Fig. 14):

1. *Match Fields*: Conformado por los campos a través de los cuales se deberá verificar la coincidencia de los flujos entrantes: puerto de ingreso y cabeceras de paquetes. Los

campos dependerán de la función a desempeñar: ruteo (IPs origen-destino), *switching* (MAC[46]s origen-destino), firewall (puertos origen-destino), etc.

2. *Priority*: Establecen prioridades en las entradas de las tablas ante paquetes coincidentes con varias de ellas. Se aplicarán las acciones definidas en la entrada de la tabla de reenvío con mayor prioridad.
3. *Counters*: Recogen información como tamaño de paquetes, número de paquetes recibidos, duración del flujo, errores, colisiones, etc. Con la finalidad de elaborar estadísticas para cada flujo en particular. Se actualizan con cada coincidencia.
4. *Instructions*: Define el tratamiento que se dará al paquete para una coincidencia dada.

Cada flujo puede tener cero o más instrucciones asociadas, que se deberán ejecutar en el orden especificado. Deberá existir una entrada por defecto para el tratamiento de los paquetes sin coincidencias.

5. *Timeouts*: Tiempo tras el cual la entrada expira en la tabla de flujo (por inactividad o por lapso de tiempo).
6. *Cookie*: Campo utilizado por el controlador para filtrar estadísticas, modificar o eliminar flujos. No se utiliza al procesar paquetes.
7. *Flags*: Son banderas que permiten modificar la forma en la que las entradas de flujo son gestionadas.

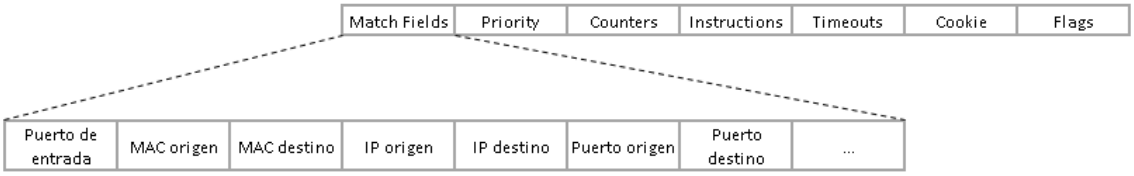


Figura 14. Estructura tabla de flujo

Cada entrada en las tablas de flujo se identifica unívocamente con base en los campos de coincidencia y su prioridad, a su vez, cada tabla de flujo es diferente a otras tablas dentro del mismo switch SDN, cada una implementa campos de coincidencia, instrucciones, acciones, etc. diferentes.

El proceso para verificar sí un paquete entrante coincide con alguna entrada de la tabla de reenvío es conocido como *packet matching* y consiste en comparar el puerto de entrada y la cabecera del paquete con la primera entrada de la tabla de flujo (entrada 0), en caso de coincidencia se aplican las instrucciones correspondientes. En caso contrario, se realiza la validación con la siguiente entrada en la tabla y así sucesivamente hasta que se encuentre una coincidencia. En caso de recorrer todas las tablas y no encontrar coincidencia alguna,

[46] MAC: Media Access Control

el paquete es enviado al controlador. Este proceso se describe de manera gráfica en la figura número 15.

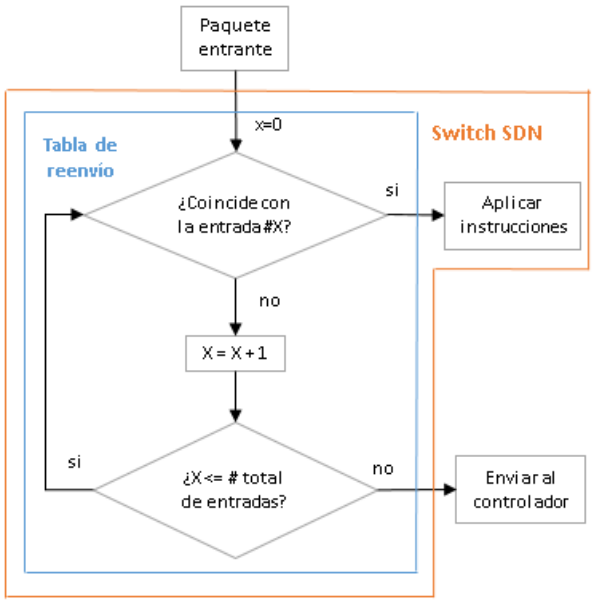


Figura 15. Proceso de *packet matching*

4.3.1 Instrucciones

Cada entrada en las tablas de flujo tiene asociado un conjunto de instrucciones que se ejecutan cuando un paquete coincide con esta. Dichas instrucciones dan lugar a cambios en el paquete, conjunto de acciones o procesamiento.

Entre las acciones que se pueden ejecutar sobre un paquete están:

1. Output *port_num*: Enviar un paquete a un puerto (físico, lógico o reservado) determinado.
2. Group *group_id*: Envía el paquete a un grupo, que es la representación de un conjunto de puertos como una sola entidad.
3. Drop: El paquete se descarta.
4. Set-Queue *queue-id*: Establece a que cola de salida se debe enviar el paquete.
5. Meter *meter-id*: Envía el paquete al medidor (meter) especificado.
6. Push-Tag/Pop-Tag: Coloca o retira etiquetas MPLS, VLAN o PBB[47].
7. Set-Field: Permiten realizar cambios a los campos de la cabecera de los paquetes, por ejemplo: IP origen, MAC origen, VLAN, etc.
8. Set-Fiel *field-type:* Permite cambiar alguno de los campos de coincidencia del paquete entrante.
9. Copy-Field *src-field-type dst-field type*: De manera análoga a la acción anterior, permite copiar alguno de los campos de coincidencia entre dos paquetes.

[47] PBB: Provider Backbone Bridge

10. Change-TTL *ttl*: Permite modificar el TTL[48] del paquete.

4.3.2 Mensajes

Mediante el protocolo OpenFlow el controlador puede agregar, actualizar o eliminar entradas en la tabla de flujo del switch, de manera reactiva o proactiva. Existen tres tipos de mensajes:

A) Controlador a switch

Son mensajes iniciados por el controlador y cuya función es gestionar e inspeccionar el estado del switch, del cual pueden o no esperar una respuesta. Existen ocho sub-tipos de este tipo de mensajes:

I. *Features*: Posterior al establecimiento del canal seguro, el controlador envía un mensaje al switch solicitando sus características.

II. *Configuration*: El controlador puede establecer y consultar parámetros de configuración en el switch. El switch sólo responderá en caso de mensajes de consulta.

III. *Modify-State*: Son los mensajes enviados por el controlador para agregar (ADD), modificar (MODIFY) o eliminar (DELETE) entradas en las tablas de flujo.

IV. *Read-State*: Recogen las estadísticas de las tablas de flujo.

V. *Packet-Out*: Se utilizan cuando el controlador desea enviar un paquete fuera de un puerto especificado en el switch.

VI. *Barrier*: Permiten al controlador asegurarse que se han cumplido las operaciones indicadas al switch.

VII. *Role-Request*: Se emplean para establecer o consultar el rol del canal OpenFlow. Esto es útil cuando el switch se conecta a más de un controlador.

VIII. *Asynchronous-Configuration*: Son utilizados por el controlador para establecer un filtro adicional en los mensajes asíncronos que desea recibir en su canal.

B) Simétricos

Los mensajes simétricos son enviados tanto por el controlador o por el switch sin previa solicitud.

- *Hello*: Enviados entre el controlador y el switch durante la configuración de la conexión.
- *Echo*: Son mensajes que pueden enviar tanto el controlador como el switch solicitando o respondiendo un eco. Sirven para realizar comprobaciones sobre el estado de la red (latencia, ancho de banda, disponibilidad, etc.)
- *Error*: Enviados por el controlador o el switch para notificar a su contraparte algún tipo de problema.

[48] TTL: Time To Live

- *Experimenter*: Son mensajes estándar que podrán ser utilizados en funcionalidades adicionales de OpenFlow en el futuro.

C) Asíncronos

Son mensajes enviados por el switch hacia el controlador y cuyo objetivo es notificar al este ultimo de eventos en la red que afectan el proceso de conmutación.

- *Packet-in*: Utilizados cuando un paquete no coincide con alguna entrada de la tabla de flujo o cuando en la tabla se especifica explícitamente el reenvío al controlador.
- *Flow-Removed*: Enviados cuando el switch remueve una entrada en la tabla de flujo debido a una previa indicación del controlador o cuando la entrada ha expirado (por inactividad o por tiempo de vida)
- *Port-Status*: Son enviados para notificar al controlador de cualquier cambio de estado en los puertos del switch (apagado, deshabilitado, activo, etc.)
- *Role-status:* Notifica al controlador de un cambio de rol.
- *Controller-status:* Utilizados para notificar al controlador cambios en el estado del canal.
- *Flow-monitor:* Son enviados para notificar de cambios en las tablas de flujos.

4.4 Tabla *Group*

Como se mencionó anteriormente, un grupo identifica como una única entidad a un conjunto de puertos, de utilidad para envíos *multicast/unicast* y es a través de la tabla de grupo que se estos son gestionados.

La tabla de grupo consta de cuatro campos (Ver Fig. 16):

Group Identifier	Group Type	Counters	Action Buckets

Figura 16. Estructura tabla *group*

- *Group Identifier:* Número de 32 bits que identifica al grupo de manera unívoca.
- *Group Type:* Determina el tipo de grupo del que se trata, que puede ser: *indirect, all, select o fast failover*.
- *Counters:* Recogen información como tamaño de paquetes, número de paquetes recibidos, duración del flujo, errores, colisiones, etc. Con la finalidad de elaborar estadísticas para cada flujo en particular. Se actualizan con cada coincidencia.
- *Action Buckets:* Contienen el conjunto de acciones a ejecutarse.

4.5 Tabla *Meter*

Un medidor mide la velocidad de los paquetes asignados y permite controlar la velocidad de estos. Los medidores se conectan directamente a las entradas de flujo y permiten a OpenFlow determinar acciones orientadas a la calidad de servicio.

La tabla medidor consta de tres campos (Ver Fig. 17):

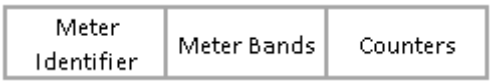


Figura 17. Estructura tabla *meter*

- *Meter Identifier:* Número de 32 bits que identifica al grupo de manera unívoca.
- *Meter Bands:* Lista no ordenada de bandas, donde cada banda especifica la velocidad y la forma de procesar el paquete.
- *Counters:* Recogen información como tamaño de paquetes, número de paquetes recibidos, duración del flujo, errores, colisiones, etc. Con la finalidad de elaborar estadísticas para cada flujo en particular. Se actualizan con cada coincidencia.

5 Controladores SDN

En los capítulos anteriores se ha denotado la importancia que tiene el controlador en el despliegue de una red SDN, pues es eje central en el manejo y administración de los recursos de la red. Si bien en el capítulo 2 se mencionaron algunas de las soluciones tanto comerciales como *open source* disponibles en el mercado, dadas las características de este documento abordaremos con mayor detalle los atributos de los controladores que pertenecen al segundo grupo.

Entre los controladores *open source* se encuentran: NOX/POX, OpenDaylight, OpenContrail, Floodlight, Ryu, FlowVisor, Atrium, Beacon, FlowER, NodeFlow, IRIS, etc. (Se podrá consultar la lista completa en la página: https://www.sdxcentral.com/sdn/definitions/sdn-controllers/open-source-sdn-controllers/). Como es posible notar, existe gran variedad de opciones entre las que se puede elegir, por lo que se deberá tomar en cuenta algunos aspectos fundamentales para decantarse por uno o por otro:

- ✓ Entorno: Se refiere al tipo de red donde se introducirá el controlador (física o virtual), la demanda de recursos informáticos que deberá soportar, el número de clientes que podrá atender, la topología y tamaño de la red, etc.
- ✓ Características propias de la plataforma: Lenguaje en el que está desarrollado, plataformas soportadas, módulos externos instalables, versiones de OpenFlow que soporta, tipo de interfaz (GUI/CLI), compatibilidad con servicios virtualizados o hardware físico, etc.
- ✓ Documentación: Por último y no menos importante, se encuentra el soporte que hay detrás de la plataforma: manuales de uso, tutoriales, foros, *datasheets*, comunidades de usuarios, casos de uso, etc.

Dada que la lista de opciones *open source* es extensa, en los siguientes párrafos nos concentraremos en tres de ellos: POX/NOX por ser el primero de los controladores SDN en ser desarrollado y OpenDaylight (ODL) y Floodlight por ser dos de los controladores más usados actualmente, al contar con el respaldo de grandes empresas y organizaciones líderes en la rama de las telecomunicaciones.

En la siguiente tabla se muestra una comparativa entre los principales atributos de los tres controladores mencionados sobre estas líneas y a continuación una breve reseña de cada uno de ellos, que permitirá tener mayores elementos que influyan en el proceso de toma de decisiones durante el diseño de las redes que los contendrán.

Tabla 2. Comparación entre controladores NOX/POX, ODL y Floodlight

	NOX/ POX	OpenDaylight	Floodlight
Año de lanzamiento	2008	2013	2014
Lenguaje de desarrollo	C++/Python	Java	Java
Plataformas	Linux/Windows/MacOs		
Versiones de OpenFlow soportadas	1.0	1.0 - 1.3	1.0 - 1.5
Compatibilidad con Mininet	Si	Si	Si
Integración recursos virtualizados y reales	Si	Si	Si
Interfaz gráfica	No/Web	Web	Web
API REST	No	Si	Si
Documentación	Baja	Media	Buena
Última versión	0.2.3	5 (Boron)	1.2

5.1 NOX/POX

NOX fue el primer controlador SDN desarrollado por un esfuerzo en conjunto entre la empresa Nicira y las universidades de Berkeley y Stanford con la finalidad de fomentar la investigación y desarrollo de nuevos protocolos de red. Fue programado en C++ y está disponible para su instalación en entornos Linux.

POX toma los aspectos destacables de NOX y a través de una plataforma puramente en Python añade algunas mejoras, como la interfaz gráfica y el soporte multiplataforma. De acuerdo al grupo desarrollador, es una plataforma ideada para la investigación, sin embargo el desempeño de la misma no es el óptimo para escenarios con gran demanda de recursos.

El proyecto es sostenido actualmente por una comunidad de desarrolladores e investigadores de distintas universidades y comunidades online (como GitHub), sin embargo, debido a que Nicira fue vendida en el año 2012, los avances con esta plataforma se han detenido y la documentación relacionada es escasa. Por lo anterior, ha dado paso a otras soluciones como las que se describen en las siguientes líneas.

5.2 OpenDaylight

ODL es un proyecto en el que interviene la Linux Foundation y empresas líderes en el área de las telecomunicaciones como son: Cisco, IBM, Citrix, HP, Intel, NEC, etc. De acuerdo a su página oficial, es la plataforma líder utilizada por los proveedores de servicios y empresas que hacen la transición hacia SDN.

El objetivo de este proyecto es ofrecer una plataforma robusta que permita implementar diversas *northbound* y *southbound* APIs, facilitando la integración de diferentes tecnologías y fabricantes en un mismo entorno. Debido a que cuenta con el apoyo de una comunidad muy amplia, la documentación relacionada es igualmente extensa y la comunidad online (foros, wikis, tutoriales, etc.) es muy activa.

La versión actual de ODL es "Boron", que surge como una colaboración entre usuarios, proveedores y desarrolladores con base en sus experiencias y experimentación con las versiones anteriores. Con respecto a sus predecesores, Boron ofrece mejoras para la integración con *cloud*, NFV e ingeniería de tráfico a gran escala, así como *frameworks* para la convivencia con Openstack y OPNFV. Además incluye mejoras considerables en cuanto a rendimiento y productividad.

En la figura 18 se muestra el esquema de operación del controlador, donde se distinguen tres capas:

1. Controlador (Servicios y aplicaciones): Compuesto por las aplicaciones de alto nivel que permiten monitorear y gestionar la red a través de:

a) Funciones del plano de control: AAA[49], switch de capa 2, servicios LISP[50], LAC[51], administrador de switches OpenFlow, procesadores de topología, etc. Cuya función es la gestión del tráfico en la red.

b) Aplicaciones embebidas: Routers Atrium, Genius, aplicaciones NAT[52], Eman, Cardinal, etc. Las cuales son aplicaciones desarrolladas por terceros y que son soportadas por el controlador.

c) Políticas de red: Son abstracciones de la red y permiten una gestión de alto nivel del sistema, como ejemplo están: Protocolo ALTO[53], FaaS[54], NEMO[55], servicio de políticas basadas en grupo, etc.

d) Aplicaciones independientes: Son aplicaciones desarrolladas por terceros que pueden ser integradas al controlador a través de APIs REST o NETCONF[56]. Por mencionar algunas están Openstack, Chef, Puppet, etc.

2. *Southbound* APIs y *plugins* de protocolos: Conformado por los módulos que permitirán la comunicación entre el controlador y los elementos del plano de datos. Como se puede observar, ofrece soporte a diversos protocolos y servicios como: OpenFlow, BGP, IoT[57], LISP, NETCONF, SXP[58], etc.

[49] AAA: Authentication, Authorization, and Accounting
[50] LISP: Locator/ID Separation Protocol
[51] LAC: Link Aggregation Control
[52] NAT: Network Address Translation
[53] ALTO: Application Layer Traffic Optimization
[54] FaaS: Fabric as a service
[55] NEMO: Network Mobility
[56] NETCONF: Network Configuration Protocol
[57] IoT: Internet of Things
[58] SXP: SGT Exchange Protocol

3. Elementos del plano de datos: Son los elementos genéricos que soportaran el plano de datos de la red y que serán programados y gestionados por el controlador: switches virtuales (OpenvSwitch) e infraestructura física, tanto SDN, convencionales e híbridos.

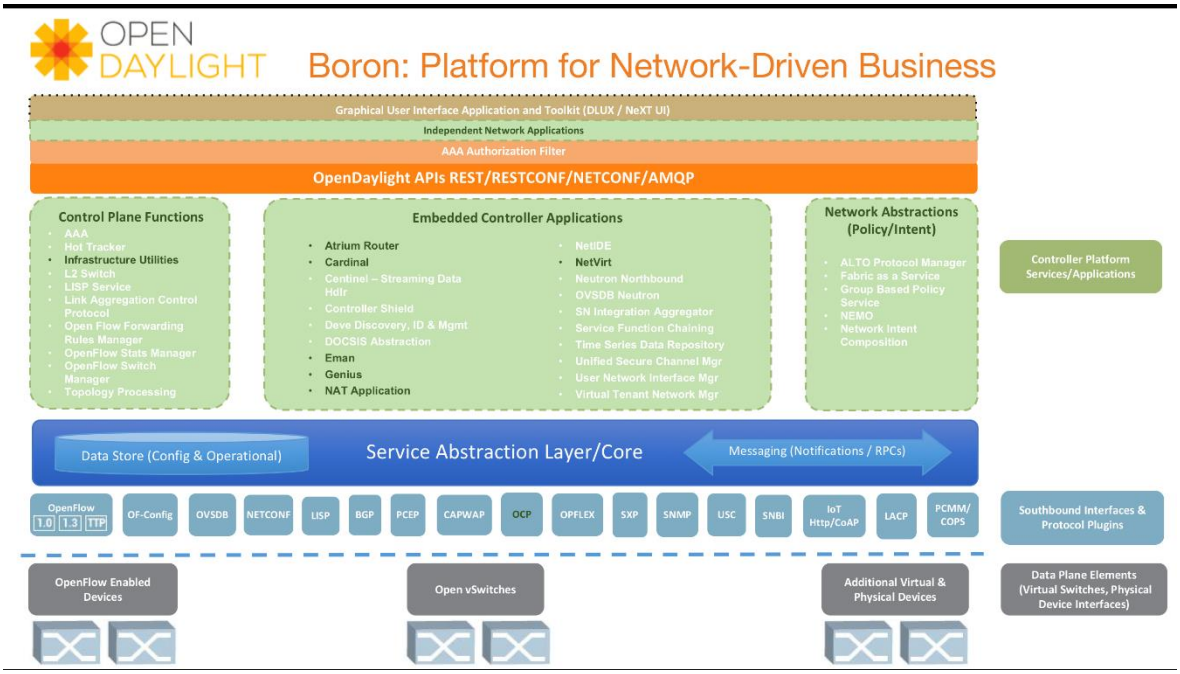


Figura 18. Estructura y operación del controlador OpenDaylight

Sobre estas capas se encuentra la API REST y la interfaz gráfica (DLUX) de OpenDaylight, a través de las cuales el usuario podrá gestionar la plataforma como una única entidad, facilitando dicha tarea y permitiendo que, gracias a su estructura modular, se desarrollen soluciones a la medida de las necesidades del usuario.

5.3 Floodlight

Todavía más reciente que OpenDaylight se encuentra Floodlight, otro controlador SDN cuya principal fortaleza es la compatibilidad con todas las versiones de OpenFlow. Al igual que ODL cuenta con una gran comunidad online que aporta continuamente mejoras a la plataforma y facilita la administración a través de su API REST e interfaz web, además e incluir módulos precargados para la administración de algunas tareas básicas. Al ser una solución más ligera consume menos recursos que ODL y por ende, se ve reflejado en la eficiencia y velocidad con la que las sentencias son procesadas.

En la siguiente figura se muestra la estructura y el diagrama de operación de Floodlight:

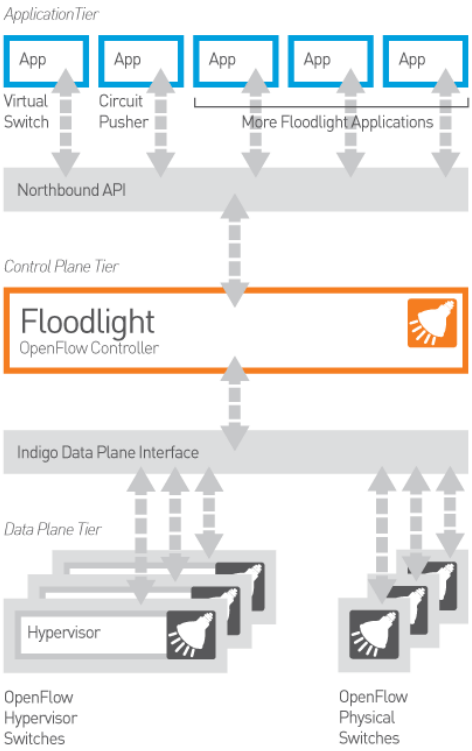


Figura 19. Estructura y operación del controlador Floodlight

Este esquema indica con mayor claridad la división en tres niveles:

1. Nivel de Aplicación: Conformado por las *northbound* APIs, que de manera modular ofrecen servicios de alto nivel. Este controlador incluye de manera preconfigurada módulos que facilitan la administración de tareas comunes en las redes de comunicaciones:

- ACL[59]: Listas de acceso estándar que permitirán o denegaran el tráfico con base en direcciones IP.
- Firewall: Al igual que las ACLs permiten o deniegan tráfico, sin embargo no solo se basan en IP, sino que es posible generar reglas tomando en cuenta direcciones MAC, números de puertos, servicios, VLANs, etc.
- Balanceador de carga: Permite configurar balanceo de carga entre los switches que componen la solución con la finalidad de evitar cuellos de botella y sobre carga en los equipos.
- Virtual Switch: Permite la creación de redes virtuales de nivel 2.
- Circuit Pusher: Permite la creación de circuitos bidireccionales entre dispositivos con base en sus direcciones IP.
- Forwarding: Es el modulo encargado de insertar los flujos que permitirá la comunicación entre el origen y el destino.

[59] ACL: Access Control List

- Static Flow Pusher: Permite enviar sentencias para modificar tablas de flujo al controlador a través de cURL.

Además de permitir la integración de aplicaciones de terceros, Openstack por ejemplo.

2. Nivel de Plano de control: Gestionado puramente por el controlador, se encarga de mantener las tablas de flujo actualizadas y sirve como puente entre el nivel de aplicación y el plano de datos.
3. Nivel de Plano de datos: A través de su interfaz Indigo, gestiona los recursos del plano de datos, tanto físicos como virtualizados.

De acuerdo a la página oficial del controlador, las fortalezas de esta plataforma son:

- ✓ Controlador multipropósito y de clase empresarial.
- ✓ Fácil de instalar y ejecutar.
- ✓ Integración de infraestructura física y virtual, siendo compatible con gran número de switches virtuales y físicos.
- ✓ Sistema modular, fácil de implementar y extender.
- ✓ Manejo de redes SDN, convencionales e hibridas.
- ✓ Alto rendimiento y multiproceso.
- ✓ Documentación extensa, que incluye tutoriales, videos, foros, manuales, *data sheets*, etc.

OpenDaylight y Floodlight son sin duda los controladores *open source* que encabezan las opciones para entornos de prueba y en producción, dado que cuentan con el respaldo de grandes comunidades que propician la generación y compartición de información, facilitando a través de esto la instalación, configuración, puesta en marcha y gestión de la plataforma elegida. Así mismo, al contar con dicho respaldo, el desarrollo de interfaces, módulos y *northbound* APIs para estas plataformas es mayor con respecto a otras opciones. Al ser muy similares en cuanto a su estructura, los puntos diferenciadores entre ODL y Floodlight se basan en el rendimiento y velocidad que pueden ofrecer, siendo indispensable valorar el tamaño y complejidad de la red a implementar para decantarse por uno u otro.

ODL es una plataforma robusta, que integra gran cantidad de módulos, utilidades y procesos, por lo que en términos de eficiencia y rendimiento se ve superado por Floodlight que solo implementa las funcionalidades básicas y es tarea del administrador de red implementar los módulos necesarios, haciéndolo más ágil y con una demanda de recursos considerablemente menor. Nuevamente resaltamos el punto sobre la valoración del entorno que los contendrá, pues para redes más complejas ODL podría ofrecer mayor utilidad mientras que para entornos menos complejos o de pruebas, Floodlight cumple muy bien con su labor.

6 Diseño e implementación de los escenarios de pruebas.

Con la finalidad de corroborar que los antecedentes técnicos de las SDNs se trasladan con fidelidad a los entornos en producción, en este capítulo se describirán algunos escenarios de prueba donde convivirán elementos de los tres niveles que componen este paradigma.

Para el plano de datos, se hará uso de OpenvSwitch para virtualizar switches SDN, para el plano de control se hará uso del controlador Floodlight que, dada la revisión que se llevó a cabo en el capítulo 5, resulto la mejor opción al ofrecer la mejor relación entre desempeño y eficiencia. Por último, referente al nivel de aplicación, se implementarán diferentes módulos que permitan gestionar diferentes aspectos de la red.

Dada la naturaleza de las aplicaciones, éstas fueron instaladas en máquinas virtuales que ejecutan el sistema operativo Ubuntu. El procedimiento de como instalar las herramientas necesarias para el desarrollo de los escenarios se detallan en el Apéndice I, mientras que en el Apéndice II se presenta un pequeño manual de Mininet/MiniEdit, que es la herramienta empleada para la emulación de los escenarios, pues permite la integración de las aplicaciones, controladores, switches y dispositivos finales de una manera sencilla. Finalmente en el Apéndice III se indican los scripts de cada uno de los escenarios desarrollados.

Los escenarios presentados van de menos a más complejos, con la finalidad de revisar detalladamente los aspectos a tomar en cuenta en el diseño e implementación de soluciones SDN.

6.1 Escenario #1: Topología básica

El objetivo de este primer escenario es visualizar a través del analizador de tráfico Wireshark los mensajes OpenFLow que se intercambian entre un controlador y un switch SDN al momento de llevar a cabo la negociación para el establecimiento de la comunicación.

En la figura 20 se muestra la topología implementada y el resultado de los comandos *dump* y *pingall*. Es posible apreciar que estamos haciendo uso de un controlador externo (Floodlight) y que hasta este momento se encuentra apagado, por lo que el resultado del test de conectividad (*pingall*) no es exitoso. En la figura 21 se muestra la tabla de flujo del switch, misma que se encuentra vacía.

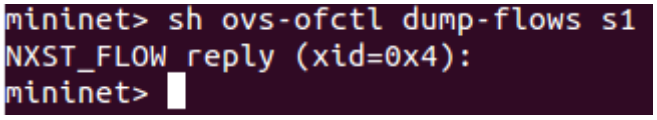


Figura 21. Tabla de flujo de s1

Figura 20. Topología escenario #1, comandos dump y pingall

Al iniciar el controlador, se visualizan los logs de este y la detección de la conexión con el switch:

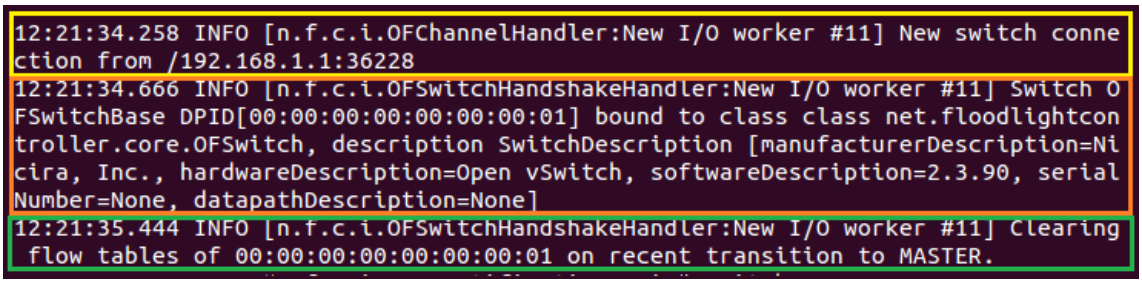


Figura 22. Logs del controlador: Conexión con s1

En la figura se indican tres apartados del log, en amarillo la detección del switch identificado a través de la IP y el puerto (36228), en naranja la negociación entre el controlador y el switch, donde este último proporciona sus características y en verde, la limpieza de las tablas de flujo del switch.

Los mensajes OpenFlow (descritos con mayor detalle en el apartado 4.3.2 de este documento) intercambiados entre el controlador y el switch fueron capturados con el analizador de tráfico, entre los más relevantes están:

- HELLO: Desde el controlador al switch y viceversa, indican a su contraparte de su existencia.

Se hace la distinción entre ambos mensajes a través del puerto origen-destino, pues el controlador utiliza como puerto por defecto el 6653 mientras que el switch el puerto 36228, además es importante notar que mientras el switch envía mensajes OpenFlow versión 1.3 el controlador lo hace en la versión 1.4.

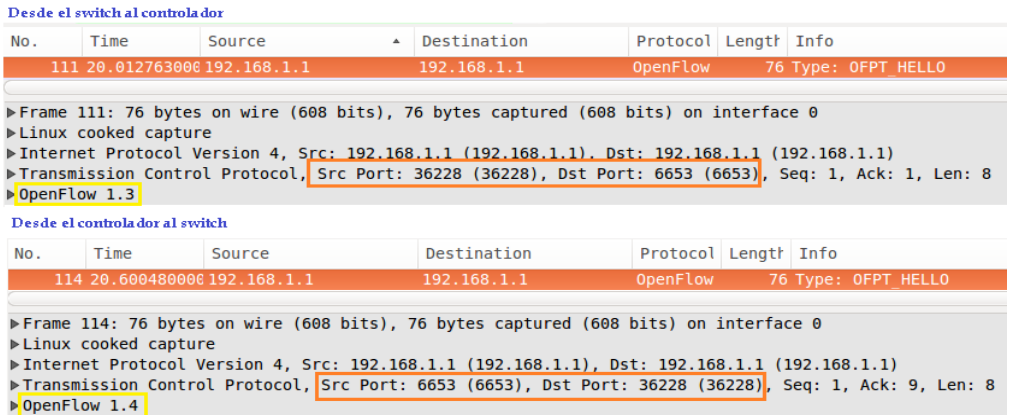


Figura 23. Captura de mensajes hello

➢ FEATURES: *FEATURES_REQUEST* solicita al switch sus características de hardware y este responde a través de *FEATURES_REPLY*.

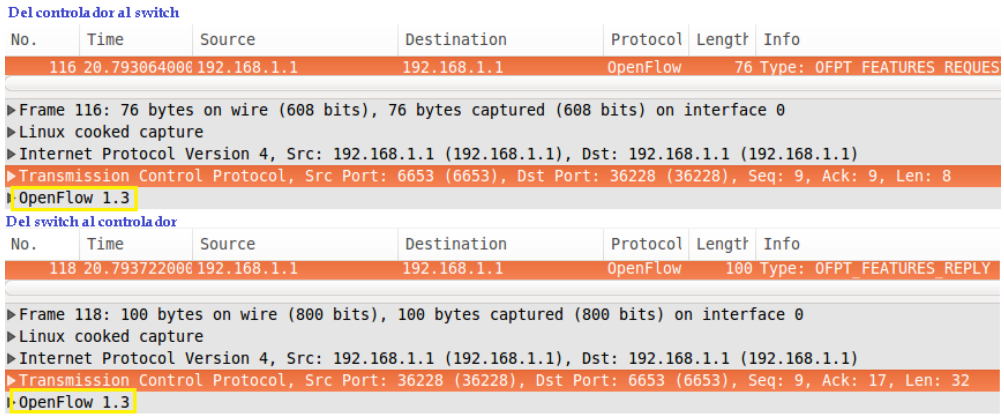


Figura 24. Captura de mensajes FEATURES REQUEST y FEATURES REPLY

En este punto, resalta el hecho que después de la negociación inicial, tanto controlador como switch envían mensajes con la versión 1.3 de OpenFlow.

➢ GET CONFIG: *GET_CONFIG_REQUEST* solicita al switch sus parámetros de configuración, el cual responde a través de un mensaje *GET_CONFIG_REPLY*.

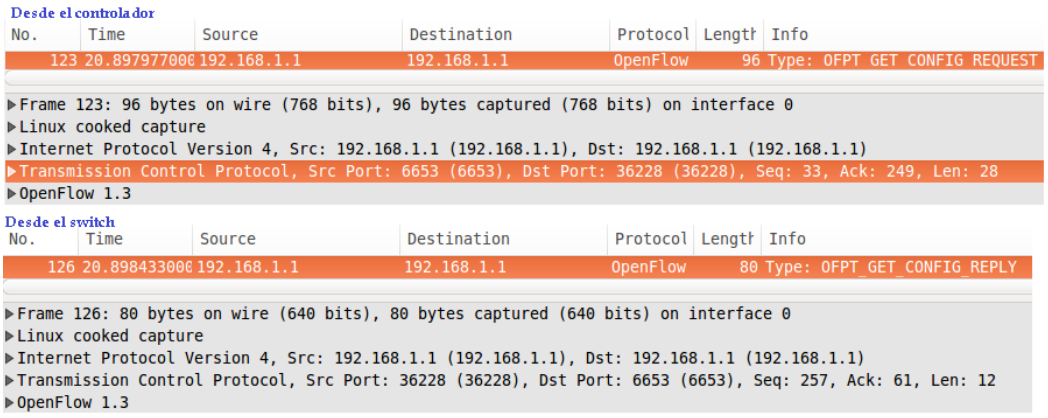


Figura 25. Captura de mensajes GET CONFIG REQUEST y GET CONFIG REPLY

➢ ECHO: A través de mensajes *ECHO_REQUEST* el controlador monitorea que el switch se encuentre activo, de ser así, éste responderá con un mensaje *ECHO_REPLY*.

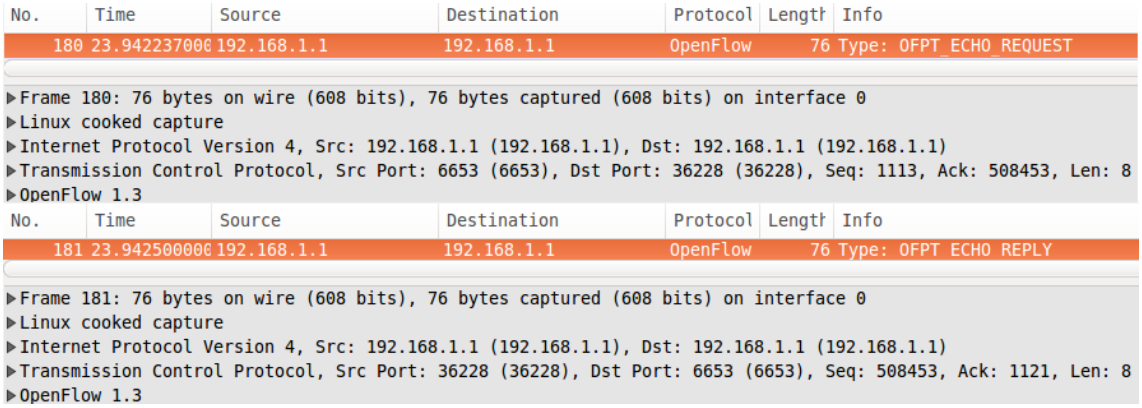


Figura 26. Captura de mensajes ECHO_REQUEST y ECHO_REPLY

➢ PACKET_IN: Cuando el switch desconoce cómo tratar un paquete, lo envía al controlador para que éste le indique como proceder a través de un mensaje *PACKET_IN* el cual incluye la IP origen y destino del paquete en cuestión.

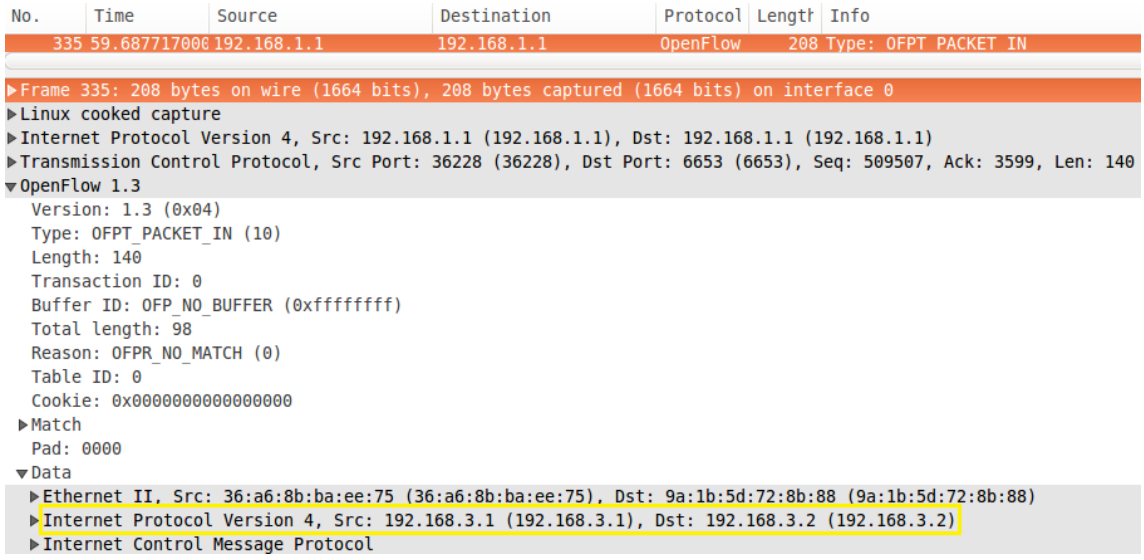


Figura 27. Captura de mensajes *PACKET_IN*

➢ FLOW_MOD: Enviados al switch para realizar alguna modificación en su tabla de flujos. Se incluyen el conjunto de instrucciones y acciones que se deberán ejecutar con los paquetes que se reciban y coincidan con la entrada que se está modificando.

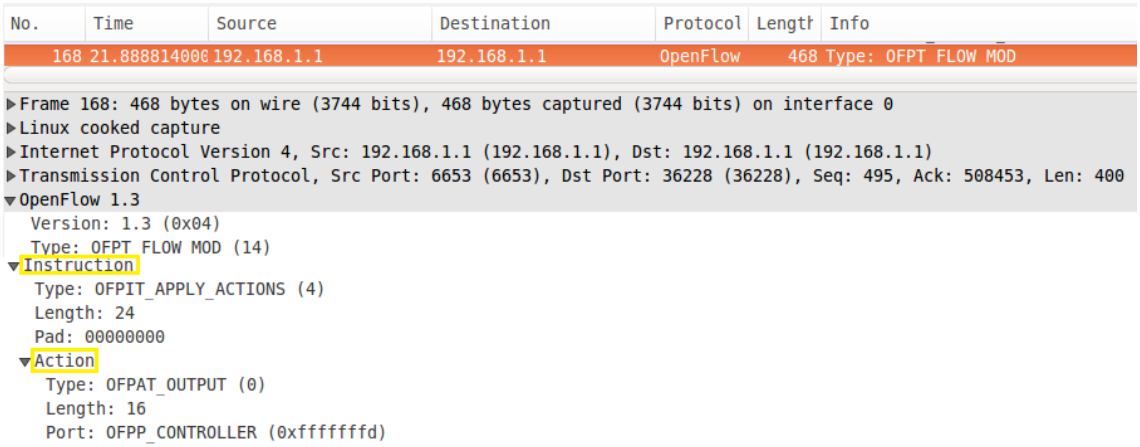


Figura 28. Captura de mensajes FLOW_MOD

➢ PACKET_OUT: Utilizado por el controlador para enviar un paquete a través del switch.

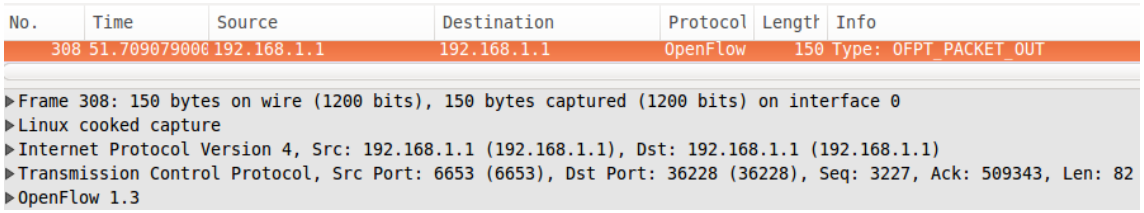


Figura 29. Captura de mensajes PACKET_OUT

Estos tres últimos conforman la terna de mensajes que se intercambian entre el switch y el controlador cuando no se cuenta con la entrada en la tabla de flujo para el tratamiento del paquete:

1. El switch indica que no sabe cómo procesar un paquete y lo envía al controlador → PACKET_IN
2. El controlador responde con una actualización a la tabla de flujo → FLOW_MOD
3. El controlador retorna al switch el paquete el cuestión para su correcto tratamiento → PACKET_OUT

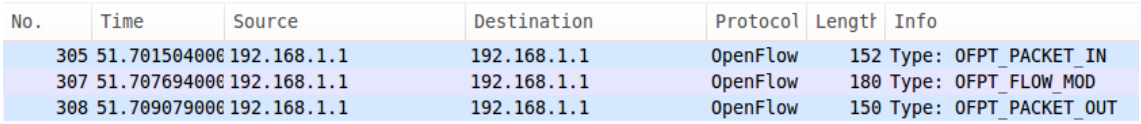


Figura 30. Captura de mensajes secuenciales: PACKET_IN, FLOW_MOD y PACKET_OUT

Como se revisó en el capítulo 4 de esta memoria, una vez que la actualización en la tabla de flujo del switch se ha realizado, éste es capaz de manejar los mensajes subsecuentes sin involucrar ya al controlador.

Una vez encendido el controlador, el test de conectividad entre h1 y h2 resulta exitoso. En la siguiente figura se muestra el resultado donde es posible apreciar que el primer

mensaje presenta tiempos de respuesta elevados, mientras que en los subsecuentes el tiempo disminuye de manera considerable:

```
mininet> h1 ping h2
PING 192.168.3.2 (192.168.3.2) 56(84) bytes of data.
64 bytes from 192.168.3.2: icmp_seq=1 ttl=64 time=462 ms
64 bytes from 192.168.3.2: icmp_seq=2 ttl=64 time=0.919 ms
64 bytes from 192.168.3.2: icmp_seq=3 ttl=64 time=0.306 ms
mininet> h2 ping h1
PING 192.168.3.1 (192.168.3.1) 56(84) bytes of data.
64 bytes from 192.168.3.1: icmp_seq=1 ttl=64 time=28.2 ms
64 bytes from 192.168.3.1: icmp_seq=2 ttl=64 time=0.922 ms
64 bytes from 192.168.3.1: icmp_seq=3 ttl=64 time=0.190 ms
```

Figura 31. Ping de h1 a h2 y viceversa, intento #1

Tras un segundo intento con el test de conectividad, se observa que al ya no requerirse la comunicación switch→controlador, el tiempo de respuesta del primer mensaje no se eleva como en el primer intento.

```
mininet> h1 ping h2 -c 3
PING 192.168.3.2 (192.168.3.2) 56(84) bytes of data.
64 bytes from 192.168.3.2: icmp_seq=1 ttl=64 time=0.233 ms
64 bytes from 192.168.3.2: icmp_seq=2 ttl=64 time=0.196 ms
64 bytes from 192.168.3.2: icmp_seq=3 ttl=64 time=0.193 ms
mininet> h2 ping h1 -c 3
PING 192.168.3.1 (192.168.3.1) 56(84) bytes of data.
64 bytes from 192.168.3.1: icmp_seq=1 ttl=64 time=0.136 ms
64 bytes from 192.168.3.1: icmp_seq=2 ttl=64 time=0.198 ms
64 bytes from 192.168.3.1: icmp_seq=3 ttl=64 time=0.199 ms
```

Figura 32. Ping de h1 a h2 y viceversa, intento #2

Para finalizar el análisis de este escenario, se muestra a continuación la tabla de flujo de s1 donde se aprecia una entrada, misma que fue gestionada por el controlador como se indicó en líneas anteriores.

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=13204.626s, table=0, n_packets=31, n_bytes=2740, idle_age=1425, priority=0 actions=CONTROLLER:65535
```

Figura 33. Tabla de flujo de s1

6.2 Escenario #2: *Hardening:* listas de acceso y firewall

Si bien el escenario anterior pretendía ser un primer acercamiento al funcionamiento de OpenFlow en la práctica, en este segundo escenario el planteamiento es un poco más complejo al interactuar con el controlador a través de cURL para indicarle acciones que nos permitan gestionar la red. En ese sentido, el objetivo de este escenario es fortalecer la seguridad de la red a través de listas de acceso estándar (ACL) y de la implementación de un firewall en el controlador.

En la siguiente figura se muestra la topología del escenario y el test de conectividad exitoso.

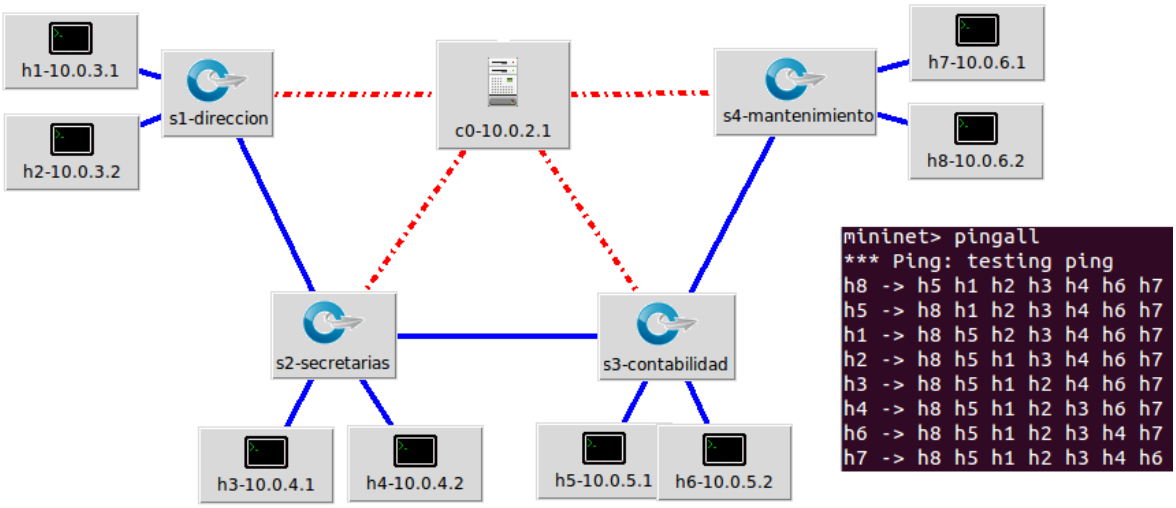


Figura 34. Topología escenario #2 y comando pingall

6.2.1 Listas de acceso estándar

A través de una lista restringiremos el acceso a los servidores ubicados en el switch s3-contabilidad para que únicamente los host de s1-direccion tengan comunicación con ellos.

Para ello, a través de una terminal en el sistema invitado y de cURL enviaremos al controlador los parámetros de la ACL, indicando la URI[60] del módulo ACL del controlador:

```
$ curl -X POST -d '{"src-ip":"10.0.4.0/24","dst-ip":"10.0.5.0/24","action":"deny"}'
http://10.0.2.10:8080/wm/acl/rules/json
$ curl -X POST -d '{"src-ip":"10.0.6.0/24","dst-ip":"10.0.5.0/24","action":"deny"}'
http://10.0.2.10:8080/wm/acl/rules/json
```

Con lo cual, solo deberá haber comunicación con h5 y h6, desde h1 y h2, como se muestra en la siguiente figura y donde se lleva a cabo nuevamente un test de conectividad:

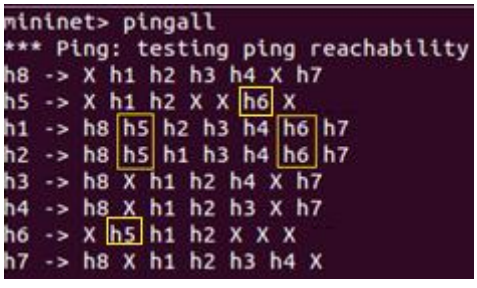


Figura 35. Test de conectividad hacia h5 y h6 después de aplicar ACL

[60] URI: Uniform Resource Identifier

Desde la consola del controlador, es posible apreciar los logs referentes a la inclusión de las listas de acceso generadas (Ver Fig. 36).

```
17:17:02.069 INFO [n.f.a.ACL:Dispatcher: Thread-21] No.1 ACL rule added.
17:17:18.128 INFO [n.f.a.ACL:Dispatcher: Thread-25] No.2 ACL rule added.
```

Figura 36. Log del controlador referente a las ACL agregadas

El siguiente comando se utiliza para listar las ACLs en nuestro controlador:

```
$ curl http://10.0.2.10:8080/wm/acl/rules/json | python -mjson.tool
```

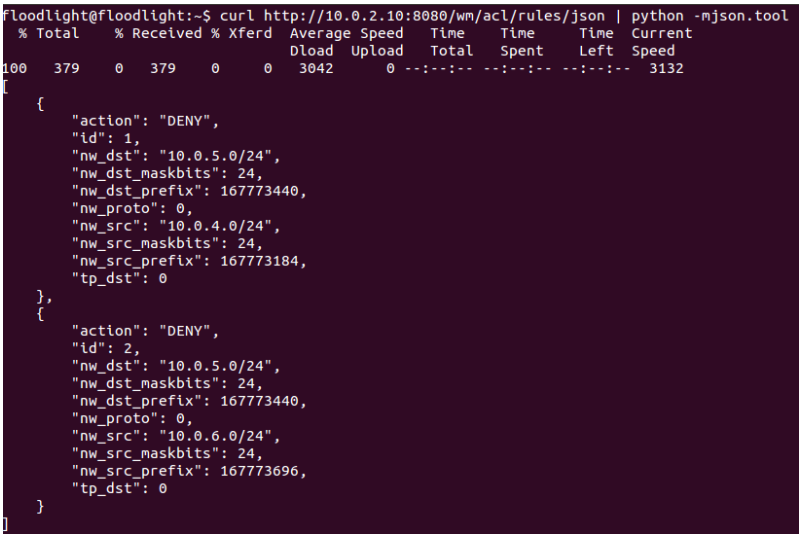


Figura 37. Listando ACLs

Y si deseamos eliminar alguna o todas las listas de acceso, deberemos utilizar:

```
//Eliminar una regla
$ curl -X DELETE -d '{"ruleid":"1" }' http://10.0.2.10:8080/wm/acl/rules/json
//Eliminar todas las reglas
$curl http://10.0.2.10:8080/wm/acl/clear/json
```

6.2.2 Implementando un firewall

Si bien las listas de acceso estándar son un método eficaz de restringir el acceso a ciertos recursos con base en las IP origen y destino, en ocasiones es necesario realizar dicha tarea con mayor detalle, como puede ser por puertos o protocolos. Para ello implementaremos reglas de firewall con las que podamos fortalecer la seguridad de la red.

En primera instancia es necesario verificar que el modulo del firewall este habilitado en nuestro controlador, a través del comando:

```
$ curl http://10.0.2.10:8080/wm/firewall/module/status/json
```

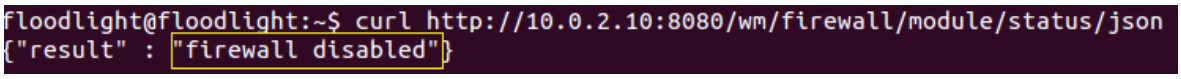


Figura 38. Estatus del módulo del firewall

Dado que el modulo esta desactivado, procedernos a ponerlo en marca con el comando:

```
$ curl http://10.0.2.10:8080/wm/firewall/module/enable/json -X PUT -d ''
```

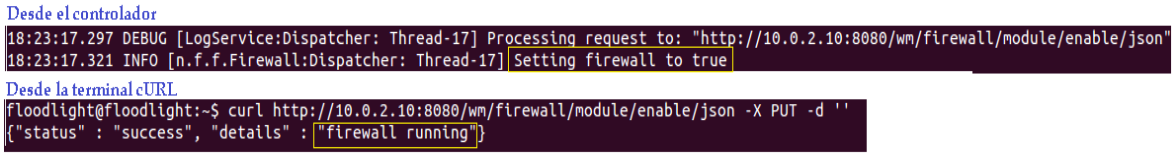


Figura 39. Logs de activación del firewall

Al activar el firewall, por defecto, todo el tráfico se bloqueara salvo aquel explícitamente permitido, por lo cual en este punto, no tenemos conectividad desde ningún host hacia los otros.

Por lo anterior, definiremos a través de qué switches se va a permitir el paso de tráfico, lo cual realizaremos con los siguientes comandos:

```
//Permitir tráfico a través de s1
$ curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:01"}'
http://10.0.2.10:8080/wm/firewall/rules/json
//Permitir tráfico a través de s2
$ curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:02"}'
http://10.0.2.10:8080/wm/firewall/rules/json
//Permitir tráfico a través de s3
$ curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:03"}'
http://10.0.2.10:8080/wm/firewall/rules/json
//Permitir tráfico a través de s4
$ curl -X POST -d '{"switchid": "00:00:00:00:00:00:00:04"}'
http://10.0.2.10:8080/wm/firewall/rules/json
```

El *switchid* corresponde a un identificador que asigna el controlador en cuanto detecta la conexión del switch, es posible visualizar este dato desde la consola de Mininet con el comando:

```
mininet> sh ovs-ofctl show switch
```

Con las reglas ingresadas estamos dejando pasar el tráfico a través de los switches sin ninguna restricción y con lo cual la conectividad perdida (al iniciar el firewall) se vuelve a recuperar.

Para puntualizar las reglas, supongamos que h6 es el servidor de nómina corporativo, al cual solo podrá tener acceso el personal de dirección (s1) y solo a través de SSH[61].

La regla a aplicar será:

```
$ curl -X POST -d '{"src-ip": "10.0.3.0/24", "dst-ip": "10.0.5.2/32", "nw-proto":"TCP", "tp-dst":"22", "action":"ALLOW" }' http://10.0.2.10:8080/wm/firewall/rules/json
```

Es importante instalar e iniciar el servidor SSH en h6, a través de la consola Mininet y de las siguientes instrucciones:

```
mininet> xterm h6
$ sudo apt-get install openssh-server
$ sudo /etc/init.d/ssh start
```

Una vez aplicada la regla y activo el servidor SSH, procederemos a probar el acceso desde h1 (permitido) y desde h3 (denegado) a h6 vía SSH. (Ver Fig. 40)

```
mininet> h1 ssh h6
root@10.0.5.2's password:
mininet> h3 ssh h6
ssh: connect to host 10.0.5.2 port 22: Connection refused
```

Figura 40. Conexión SSH desde h1 y h3 hacia h6

Los comandos para listar las reglas del firewall y eliminar alguna de ellas se muestran a continuación:

```
//Listar reglas
$ curl http://10.0.2.10:8080/wm/firewall/rules/json | python -mjson.tool
//Eliminar regla (con base en el ruleid que se puede obtener con el comando anterior
$ curl -X DELETE -d '{"ruleid":"36176203"}'
http://10.0.2.10:8080/wm/firewall/rules/json
```

Como se comentó al inicio de esta sección, el objetivo de la aplicación de estas técnicas es fortalecer la seguridad de la red a través de la inserción de reglas en el controlador que permitan denegar o permitir el tráfico con base en distintos parámetros que permitan cubrir las necesidades del cliente.

[61] SSH: Secure SHell

6.3 Escenario #3: Conexión a Internet, calidad de servicio y administración del ancho de banda.

Añadiendo mayor complejidad a la red, en este escenario se simulará la conexión hacia Internet, la implementación de calidad de servicio y administración de ancho de banda.

En la siguiente figura se muestra la topología para este escenario:

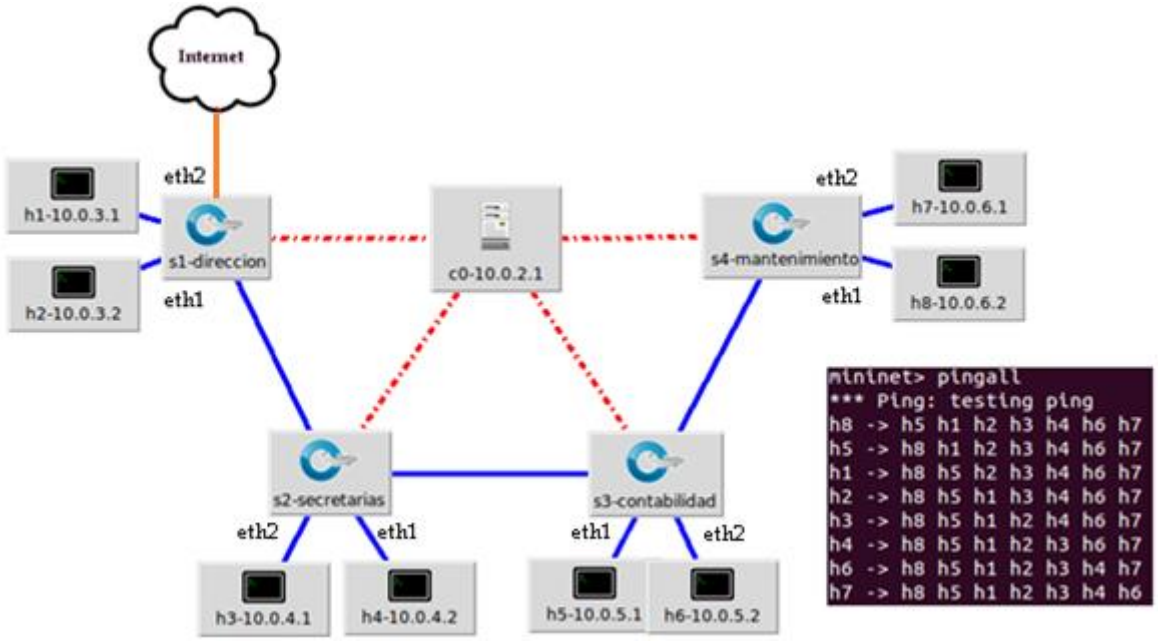


Figura 41. Topología escenario #3 y comando pingall

Como se puede observar en la imagen, el escenario cuenta con una conexión a Internet, la cual es configurada con el siguiente comando dentro del script:

```
net.addNAT().configDefault()
net.start()
```

Es importante precisar que para que el escenario tenga conexión a Internet, la máquina virtual donde lo estemos ejecutando debe tener configurado el adaptador de red en modo NAT.

Posteriormente, es necesario indicar los DNS[62] a través del comando mostrado a continuación:

```
mininet> xterm c0
$sudo nano /etc/resolv.conf
nameserver 8.8.8.8
```

[62] DNS: Domain Name System

Una vez realizado esto, todos los hosts tendrán acceso a Internet (Ver Fig. 42)

```
mininet> h1 ping google.com -c 1
PING google.com (216.58.214.174) 56(84) bytes of data.
64 bytes from mad01s26-in-f174.1e100.net (216.58.214.174): icmp_seq=1 ttl=54 time=64.3 ms
mininet> h5 ping google.com -c 1
PING google.com (216.58.214.174) 56(84) bytes of data.
64 bytes from mad01s26-in-f174.1e100.net (216.58.214.174): icmp_seq=1 ttl=54 time=56.8 ms
rtt min/avg/max/mdev = 64.307/64.307/64.307/0.000 ms
mininet> h7 ping google.com -c 1
PING google.com (216.58.214.174) 56(84) bytes of data.
64 bytes from mad01s26-in-f174.1e100.net (216.58.214.174): icmp_seq=1 ttl=54 time=67.5 ms
```

Figura 42. Evidencia de comunicación hacia Internet

A través de la implementación de calidad de servicio podremos garantizar tasas de transmisión para ciertos servicios y así priorizar tráfico para aquellos sensibles al retardo o de criticidad alta (comunicaciones hacia servidores críticos o centros de datos, por ejemplo)

Para instalar módulo de QoS en el controlador, es necesario ingresar los siguientes comandos en una terminal del sistema invitado:

```
$git clone https://github.com/wallnerryan/floodlight-qos-beta.git
$cd floodlight-qos-beta
$ant; ./floodlight.sh
```

Con lo anterior, se habilitará la sección "Tools" en la GUI del controlador y desde donde podremos verificar sí el servicio se encuentra habilitado o no (Ver Fig. 43).

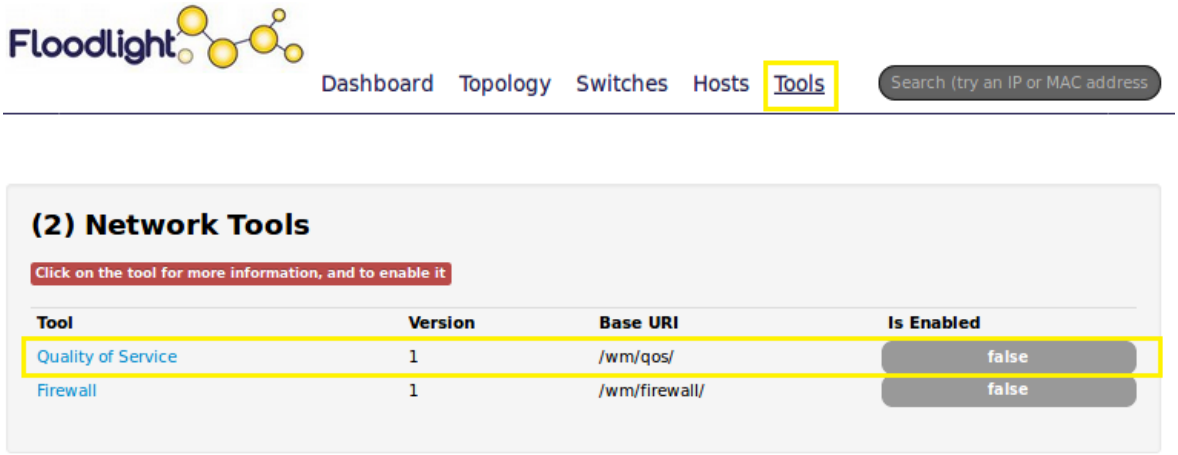


Figura 43. Habilitación del servicio QoS en el controlador

Para la habilitación del servicio, se deberá ingresar el siguiente comando:

```
$curl http://10.0.2.10:8080/wm/qos/tool/enable/json
```

.

La base sobre la que se asienta la calidad de servicio y la administración del ancho de banda es en la generación de colas que fungirán como *buffers* de almacenamiento con tasas de transmisión definidas y donde se alojaran los paquetes que hagan *match* con las políticas definidas para cada cola.

Es recomendable, antes de proseguir, ejecutar el siguiente comando que nos permitirá visualizar las conexiones activas del switch especificado y nos mostrara las velocidades máximas de cada enlace, como ejemplo, se mostrará el resultado del comando para s1:

```
mininet>sh ovs-ofctl show s1
```

```
mininet> sh ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
 1(s1-eth1): addr:3a:b2:df:4d:c9:e3
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:b6:76:58:f5:1b:b2
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:da:82:6e:73:2b:7a
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
```

Figura 44. Resultado del comando sh ovs-ofctl show s1

Iperf es una herramienta que genera flujos de datos con la finalidad de medir el rendimiento de la red. Antes de aplicar los QoS, vamos a medir las velocidades en h1 hacia internet y hacia la LAN a través de los comandos que se indican a continuación:

```
//Medición de tasas de transmisión hacia internet
mininet> iperf h1 nat0
//Medición de tasas de transmisión dentro de la LAN
mininet> iperf h1 nat0
```

```
mininet> iperf h1 nat0
*** Iperf: testing TCP bandwidth between h1 and nat0
*** Results: ['4.70 Gbits/sec', '4.71 Gbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['3.44 Gbits/sec', '3.45 Gbits/sec']
```

Figura 45. Resultado del comando iperf entre h1-nat0 y h1-h3 antes de la aplicación de QoS

Como se puede apreciar las velocidades rondan entre los 3.5 y 4.5 Gbps, ahora procederemos a generar dos colas: "VIP" para el segmento de datos 10.0.3.0/24 y 10.0.5.0/24 y "GLOBAL" para los segmentos 10.0.4.0/24 y 10.0.6.0/24. La cola VIP tendrá una tasa de transmisión mayor que la GLOBAL y por ende el tráfico procedente de los segmentos asociados será priorizado en su salida hacia Internet.

La configuración de las colas se lleva a cabo con los siguientes comandos:

```
//Cola VIP en la salida hacia internet a 9Mbps
mininet> sh ovs-vsctl -- set port s2-eth5 qos=@VIP -- --id=@VIP create qos type=linux-htb other-config:max-rate=9000000 queues=0=@q0 -- --id=@q0 create queue other-config:min-rate=9000000 other-config:max-rate=9000000
//Cola GLOBAL en la salida hacia internet a 4Mbps
mininet> sh ovs-vsctl -- set port s2-eth5 qos=@GLOBAL -- --id=@GLOBAL create qos type=linux-htb other-config:max-rate=4000000 queues=1=@q1 -- --id=@q1 create queue other-config:min-rate=4000000 other-config:max-rate=4000000
```

Una vez generadas las colas es necesario asociar los flujos a la cola VIP o a la cola GLOBAL, por lo que enviaremos al controlador las siguientes sentencias para llevar a cabo esta tarea:

```
//Asociación de flujos de s1 a la cola VIP
$ curl -d '{"switch": "00:00:00:00:00:00:00:01", "name":"VIP", "cookie":"0", "priority":"32768","ingress-port":"1","active":"true", "actions":"enqueue=0:0"}' http://10.0.2.10:8080/wm/staticflowpusher/json
//Asociacion de los flujos de s2 a la cola GLOBAL
$ curl -d '{"switch": "00:00:00:00:00:00:00:02", "name":"GLOBAL", "cookie":"0", "priority":"32768","ingress-port":"1","active":"true", "actions":"enqueue=1:1"}' http://10.0.2.10:8080/wm/staticflowpusher/json
//Asociación de flujos de s3 a la cola VIP
$ curl -d '{"switch": "00:00:00:00:00:00:00:03", "name":"VIP", "cookie":"0", "priority":"32768","ingress-port":"1","active":"true", "actions":"enqueue=0:0"}' http://10.0.2.10:8080/wm/staticflowpusher/json
//Asociación de flujos de s4 a la cola GLOBAL
$ curl -d '{"switch": "00:00:00:00:00:00:00:04", "name":"VIP", "cookie":"0", "priority":"32768","ingress-port":"1","active":"true", "actions":"enqueue=1:1"}' http://10.0.2.10:8080/wm/staticflowpusher/json
```

Tras lo cual procedemos nuevamente a testear las velocidades de salida a Internet con iperf y cuyos resultados se muestran en la figura 46. Donde se aprecia que para h1 (Cola VIP) la tasa de transmisión ronda los 9.5 Mbps, mientas que para h8 (Cola GLOBAL) ronda los 4.5 Mbps. De esta manera, logramos priorizar el uso de recursos para las áreas que lo requieran y limitar el ancho de banda en áreas menos críticas, logrando un mejor aprovechamiento de estos. De ser necesario, se podrá generar las colas que se requieran para alguna solución en particular y es aquí donde desde el proceso de diseño tienen que tomarse en cuenta aspectos como este.

```
mininet> iperf h1 nat0
*** Iperf: testing TCP bandwidth between h1 and nat0
*** Results: ['8.57 Mbits/sec', '10.5 Mbits/sec']
mininet> iperf h8 nat0
*** Iperf: testing TCP bandwidth between h8 and nat0
*** Results: ['3.83 Mbits/sec', '4.94 Mbits/sec']
```

Figura 46. Resultado del comando iperf entre h1-nat0 y h1-h3 después de la aplicación de QoS

6.4 Escenario #4: Respuesta reactiva a cambios en la topología

SDN propicia una gestión de red más sencilla debido a la separación de los planos de datos y de control. Al haber un cambio en la topología de la red, ya sea por caída de enlaces, equipos dañados u otro factor que la altere, es importante que la red pueda adaptarse a dichos cambios en el menor tiempo posible. La respuesta proactiva a estos está estrechamente vinculada al controlador, pues es el encargado de censar los dispositivos que componen la solución, mismos que también le notifican cuando detectan algún cambio.

El objetivo de este escenario es visualizar el comportamiento de la red ante un cambio en la topología y poder constatar el rol que juega cada elemento de la red en la recuperación de la comunicación, para ello el escenario presenta puntos de redundancia, con la finalidad de que existan varios caminos para un mismo destino (Ver Fig. 47).

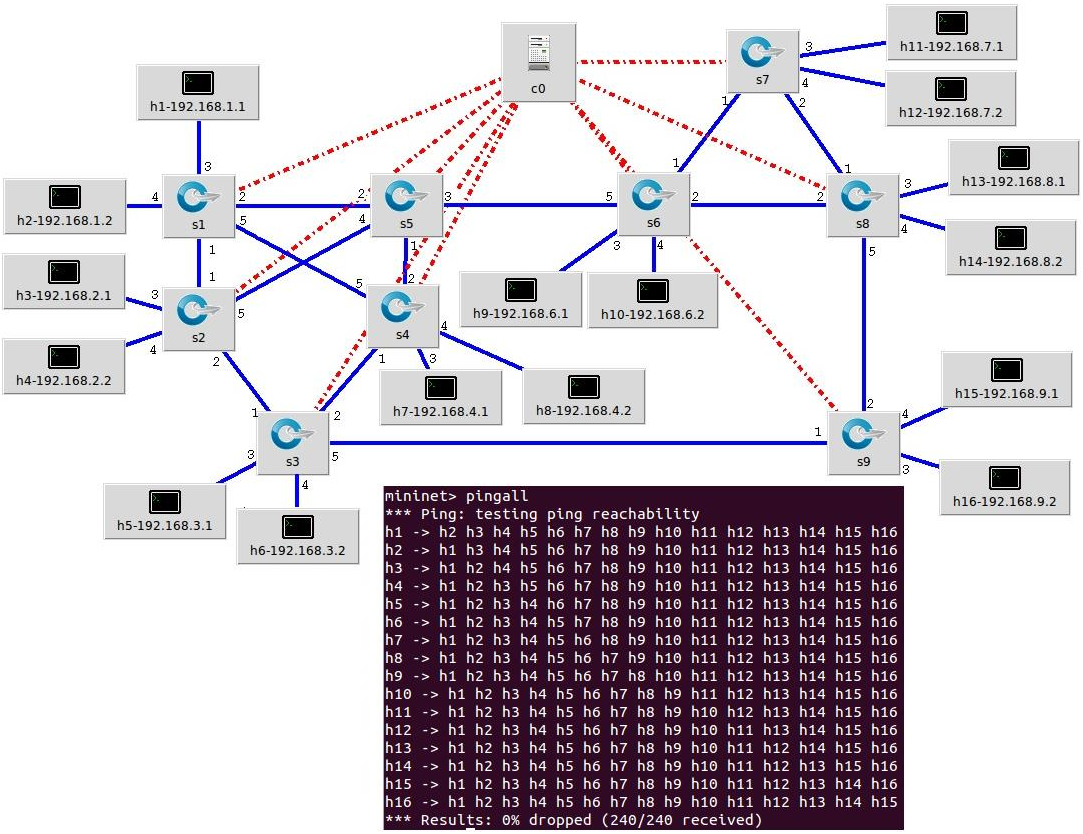


Figura 47. Topología escenario #4 y comando pingall

La figura anterior muestra que el test de conectividad entre todos los host es exitoso. Para el objetivo de este escenario procederemos a rastrear la ruta que siguen los paquetes en la comunicación entre h5 y h11.

Consultando las tablas de flujo (Ver Fig. 48), podemos observar que los paquetes intercambiados entre h5 (192.168.3.1) a h11 (192.168.7.1) siguen la ruta s3 puerto 5 → s9 puerto 2 → s8 puerto 1 → s7 puerto 3 y que en cada switch se agregan entradas por pares, con lo que se garantiza la comunicación bidireccional.

```
mininet> sh ovs-ofctl dump-flows s3
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=11.654s, table=0, n_packets=11, n_bytes=1078, idle_t
:40:b6,dl_dst=8a:84:18:3f:2b:22 nw_src=192.168.3.1,nw_dst=192.168.7.1 actions=output:5
 cookie=0x20000000000000, duration=11.637s, table=0, n_packets=12, n_bytes=1176, idle_t
:2b:22,dl_dst=ca:47:61:07:40:b6,nw_src=192.168.7.1,nw_dst=192.168.3.1 actions=output:3
mininet> sh ovs-ofctl dump-flows s9
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=29.847s, table=0, n_packets=30, n_bytes=2940, idle_t
:40:b6,dl_dst=8a:84:18:3f:2b:22 nw_src=192.168.3.1,nw_dst=192.168.7.1 actions=output:2
 cookie=0x20000000000000, duration=29.817s, table=0, n_packets=30, n_bytes=2940, idle_t
:2b:22,dl_dst=ca:47:61:07:40:b6,nw_src=192.168.7.1,nw_dst=192.168.3.1 actions=output:1
mininet> sh ovs-ofctl dump-flows s8
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=51.903s, table=0, n_packets=52, n_bytes=5096, idle_t
:40:b6,dl_dst=8a:84:18:3f:2b:22,nw_src=192.168.3.1,nw_dst=192.168.7.1 actions=output:1
 cookie=0x20000000000000, duration=51.872s, table=0, n_packets=52, n_bytes=5096, idle_t
:2b:22,dl_dst=ca:47:61:07:40:b6,nw_src=192.168.7.1,nw_dst=192.168.3.1 actions=output:5
mininet> sh ovs-ofctl dump-flows s7
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=70.154s, table=0, n_packets=71, n_bytes=6958, idle_t
:40:b6,dl_dst=8a:84:18:3f:2b:22,nw_src=192.168.3.1,nw_dst=192.168.7.1 actions=output:3
 cookie=0x20000000000000, duration=70.104s, table=0, n_packets=70, n_bytes=6860, idle_t
:2b:22,dl_dst=ca:47:61:07:40:b6,nw_src=192.168.7.1,nw_dst=192.168.3.1 actions=output:2
```

Figura 48. Ruta de los paquetes enviados de h5 a h11

Con la intención de observar el comportamiento del controlador ante algún cambio en la topología, procederemos a dar de baja los enlaces s3 → s9, s3 → s2 y s4 → s5 a través de los siguientes comandos, mientras que de manera paralela mantenemos un ping extendido entre h5 → h11 y wireshark realizando las capturas correspondientes.

```
mininet> link s3 s9 down
mininet> link s3 s2 down
mininet> link s4 s5 down
```

Además es necesario agregar los siguientes módulos en el controlador, esto desde el archivo de configuración de Floodlight.

```
net.floodlightcontroller.forwarding.Forwarding
net.floodlightcontroller.learningswitch.LearningSwitch
```

Debido a los enlaces fuera de servicio, los paquetes deben tomar una ruta alternativa: s3 puerto 2 → s4 puerto 5 → s1 puerto 2 → s5 puerto 3 → s6 puerto 1 → s7 puerto 3 (Ver Fig. 49)

En la figura 50 se muestra el resultado del ping extendido donde se aprecia incremento del tiempo de respuesta en un punto determinado, esto es debido al cambio en la topología y por ende a que existe comunicación nuevamente con el controlador debido a que se desconoce cómo procesar los paquetes.

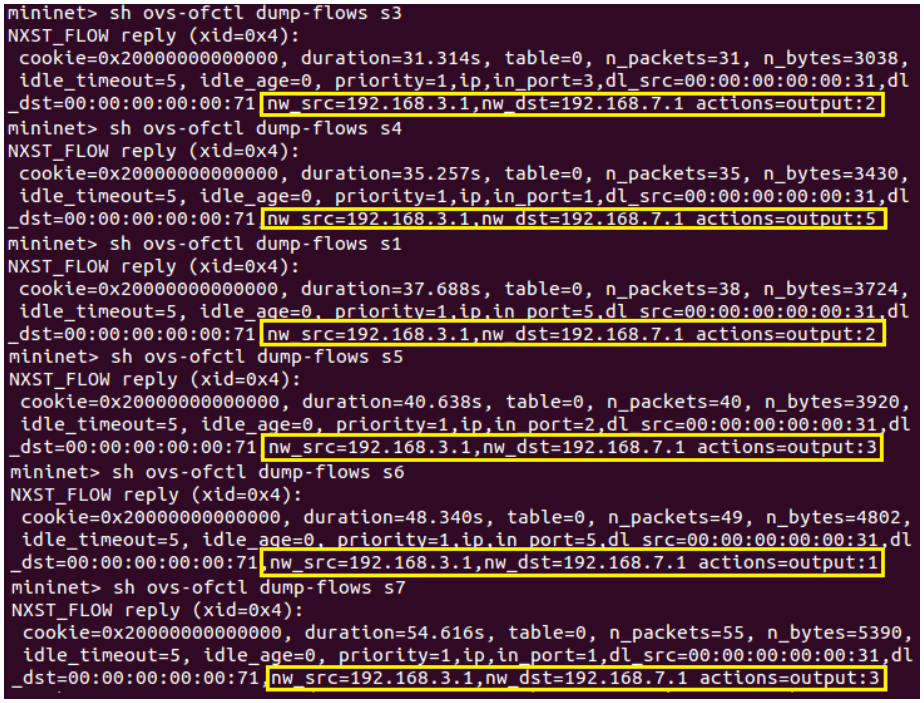


Figura 49. Ruta de los paquetes enviados de h5 a h11 después de los enlaces caídos

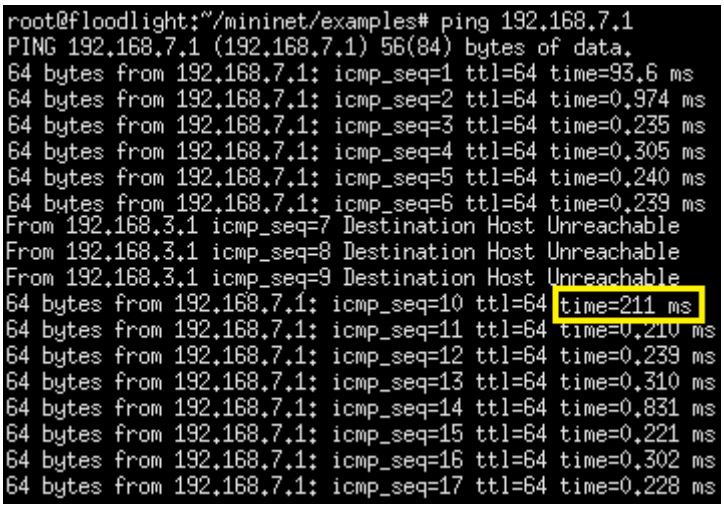


Figura 50. Ping extendido entre h5 a h11

Analizando las capturas de wireshark es posible apreciar que el controlador envía cinco instrucciones de modificación en las tablas de flujo (una por cada switch que se involucra

en la nueva ruta de comunicación). Para comprender mejor las capturas de tráfico que se realizaron, en la siguiente tabla se listan los switches que componen el escenario y el puerto que les fue asignado por el controlador:

Tabla 3. Listado de switches del escenario 5, su DPID y número de puerto

SW	DPID	PUERTO
1	00:00:00:00:00:00:00:01	51320
2	00:00:00:00:00:00:00:02	51333
3	00:00:00:00:00:00:00:03	51321
4	00:00:00:00:00:00:00:04	51332
5	00:00:00:00:00:00:00:05	51322
6	00:00:00:00:00:00:00:06	51328
7	00:00:00:00:00:00:00:07	51326
8	00:00:00:00:00:00:00:08	51323
9	00:00:00:00:00:00:00:09	51329

La captura de tráfico se muestra en la siguiente figura, donde se resalta solo uno de los cinco mensajes FLOW_MOD a manera de ejemplo, en él se observa el puerto del switch y la acción asociada para el flujo en cuestión.

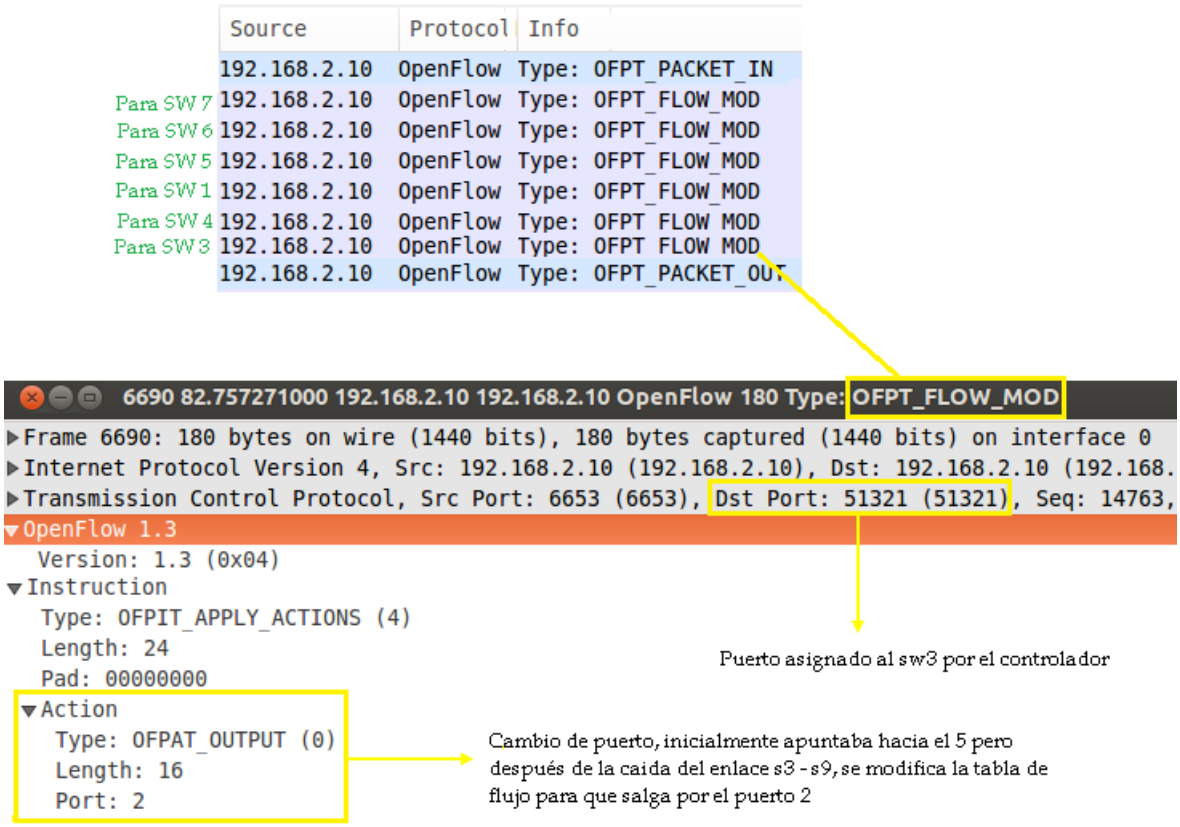


Figura 51. Captura de mensajes OpenFlow para modificación de tablas de flujo

Como es posible apreciar, el controlador toma la gestión de la red y dispara modificaciones en las tablas de flujo de acuerdo a los cambios que se presenten en la misma y facilitando así la tarea del administrador de red, pues al contar con una visión general de ella, puede calcular rutas eficaces sin tener que aplicar protocolos como STP para evitar bucles en la red.

6.5 Escenario #5: *Tunneling*

El *tunneling* es una técnica que permite encapsular un protocolo sobre otro protocolo de capa superior (encapsulador) para crear túneles que permitan la comunicación entre los dispositivos en una red de comunicaciones. Una de las aplicaciones de esta técnica es la generación de VPN[63]s, que conectan a sitios remotos a través de redes superpuestas como lo es Internet.

Existen diversos protocolos para la implementación de los túneles, entre los que destacan VXLAN[64], IPSec[65], GRE[66], etc. En este cuarto y último escenario implementaremos GRE para generar túneles que permitan la comunicación de tres sucursales de la misma empresa a través de Internet. Esta sería una aplicación de SD-WAN[67], a través de la cual se pretende programar las redes de área extensa y que solo con una conexión a Internet, puedas comunicar sitios remotos de manera sencilla.

El objetivo de este escenario es simular la conexión de tres sitios remotos a través de túneles GRE a través de Internet. En la figura 52 se muestra la topología de este escenario, donde para poder simular la conexión a Internet se hizo uso de tres máquinas virtuales independientes, cada una con un escenario propio y conectadas a Internet. El controlador reside en la máquina virtual número 2, sin embargo el plano de control de las maquinas 1 y 3 también es gestionado por este equipo, pues se buscaba un escenario con control centralizado que permitiera eficientar el uso de recursos.

Para permitir la comunicación mallada entre las tres máquinas virtuales, será necesario la generación del mismo número de túneles GRE y cuyos pasos a seguir para la configuración se mostraran a continuación (Tener en cuenta que los túneles GRE operativamente funcionan como una extensión de la LAN).

Es importante recordar que en el apéndice III de este documento se facilitan los scripts necesarios para la reproducción de los escenarios revisados en este capítulo.

[63] VPN: Virtual Private Network
[64] VXLAN: Virtual Extensible Local Area Network
[65] IPSec: Internet Protocol Security
[66] GRE: Generic Routing Encapsulation
[67] SD-WAN: Software Defined Wide Area Network

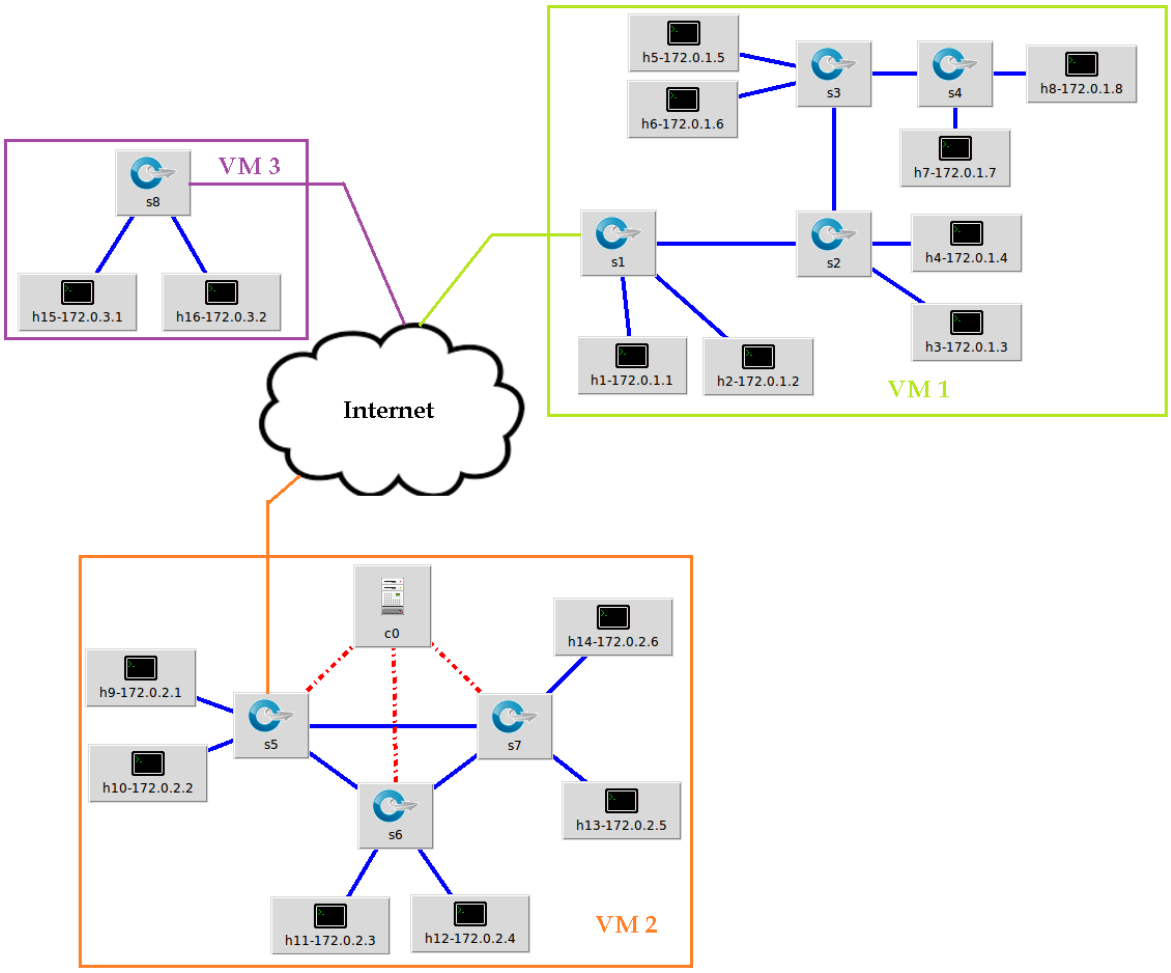


Figura 52. Topología escenario #4

Previo a la generación de los túneles, la única red que tiene comunicación dentro de la misma LAN es VM2 pues es la que cuenta con el controlador en la misma red. VM1 y VM3 no tiene comunicación con dispositivos de su misma red debido a que no hay enlace con el controlador y por ende los switches no saben cómo procesar los paquetes.

Los comandos para le generación del túnel VM 1 – VM2 son:

```
//En VM1
mininet> s1.cmd('ovs-vsctl add-port s1 s1-gre1 -- set interface s1-gre1 type=gre options:remote_ip=192.168.2.10')
//En VM2
mininet> s5.cmd('ovs-vsctl add-port s5 s5-gre1 -- set interface s5-gre1 type=gre options:remote_ip=192.168.1.10')
```

Analizando los comandos anteriores, es importante mencionar dos aspectos fundamentales: la creación de un puerto tipo GRE y la indicación de la IP remota, que es la IP del otro extremo del túnel y que debe ser de un segmento diferente al que se maneja en la LAN (en ambientes reales, la IP remota es la dirección pública del sitio contrario).

Una vez establecido el túnel, podemos apreciar que la comunicación entre VM1 y VM2 se establece con éxito (Ver Fig. 53).

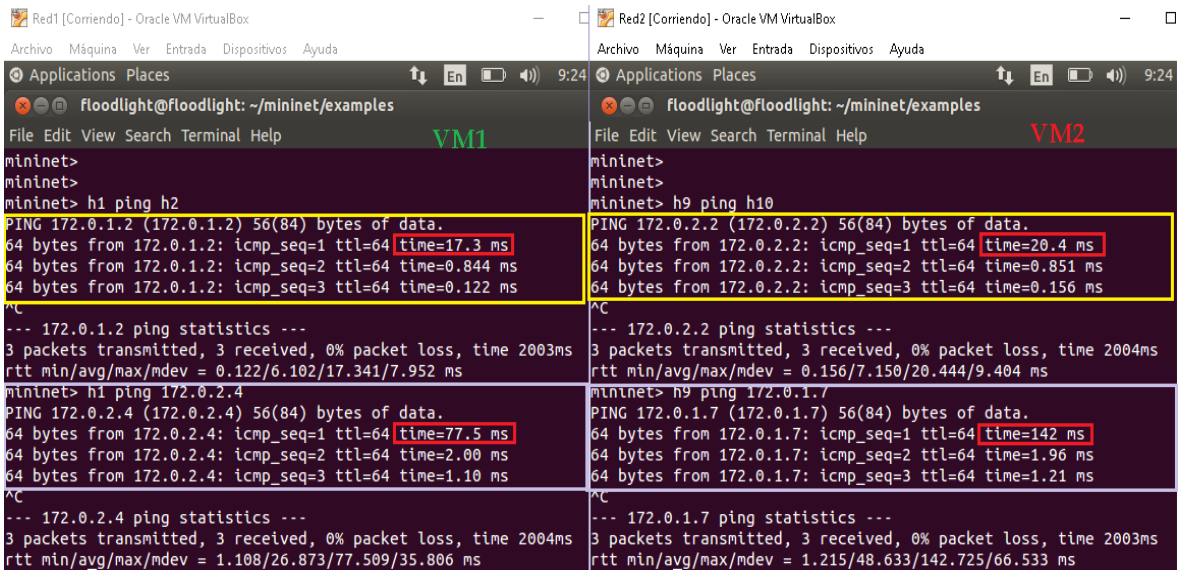


Figura 53. Evidencia de comunicación entre VM1 y VM2

En amarillo se resaltan el test de conectividad hacia otro dispositivo dentro de la misma red y en lila se indica el test de conectividad hacia un dispositivo en la otra máquina virtual. Ambos test son exitosos y nuevamente se aprecia que los primeros paquetes el tiempo de respuesta son elevados debido a la comunicación que se establece con el controlador en primera instancia.

Ahora se procederá a configurar los dos túneles restantes: el a través de los comandos:

```
// Túnel VM2 – VM3
// En VM2
mininet> s5.cmd('ovs-vsctl add-port s5 s5-gre2 -- set interface s5-gre2 type=gre options:remote_ip=192.168.3.30')
// En VM3
mininet> s8.cmd('ovs-vsctl add-port s8 s8-gre2 -- set interface s8-gre2 type=gre options:remote_ip=192.168.2.10')
// Túnel VM3 – VM1
// En VM3
mininet> s8.cmd('ovs-vsctl add-port s8 s8-gre1 -- set interface s8-gre1 type=gre options:remote_ip=192.168.1.10')
// En VM1
mininet> s1.cmd('ovs-vsctl add-port s1 s1-gre2 -- set interface s1-gre2 type=gre options:remote_ip=192.168.3.30')
```

Con lo anterior ya tenemos comunicación mallada entre las tres máquinas virtuales y cuya evidencia se presenta a continuación:

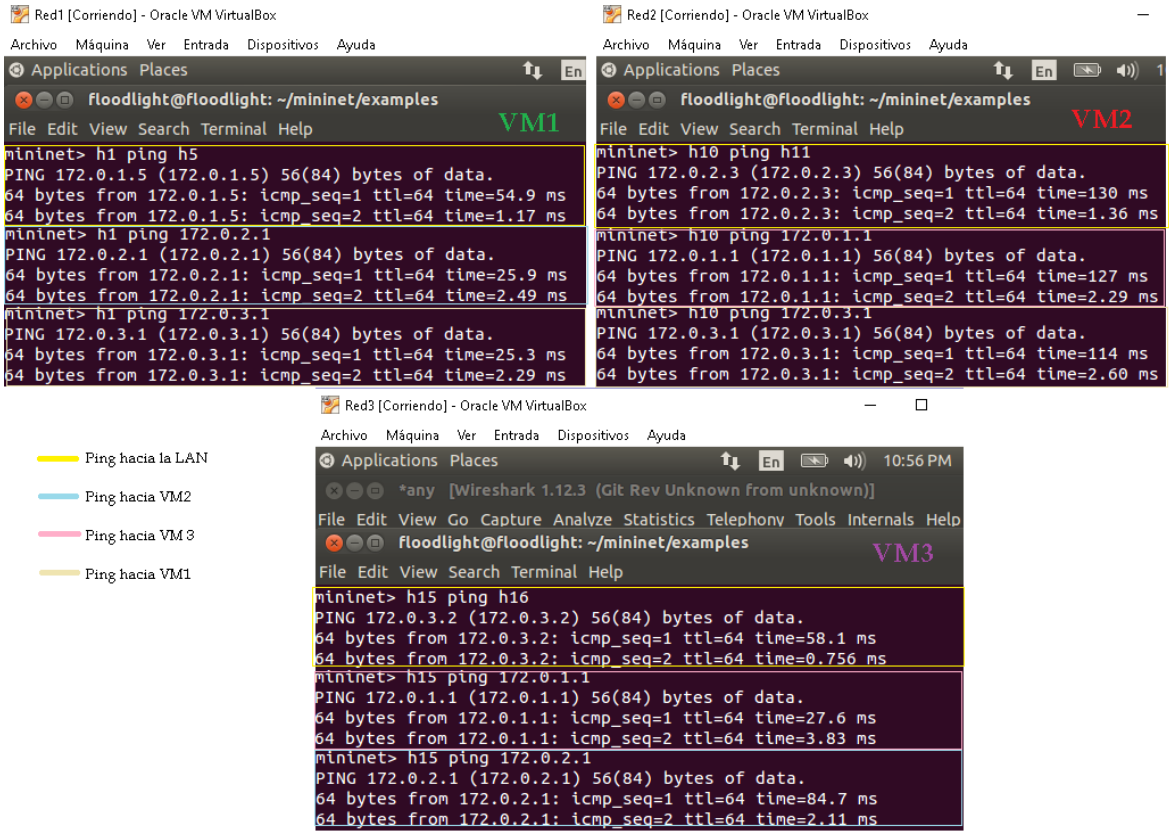


Figura 54. Evidencia de comunicación entre VM1, VM2 y VM3

A través del analizador de tráfico se observan nuevamente los mensajes intercambiados entre el controlador y los switches SDN, sin embargo en este caso en específico es importante señalar el encapsulamiento que se lleva a cabo de los paquetes de capa dos sobre el protocolo de red IP (Ver Fig. 55)

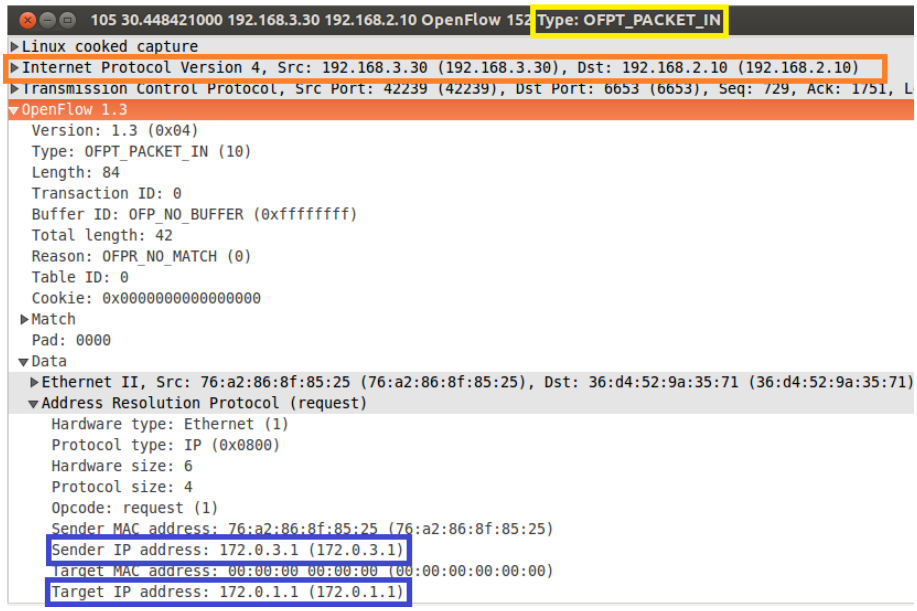


Figura 55. Captura de tráfico relacionado al encapsulamiento de datos

7 Conclusiones y trabajos futuros

Durante la elaboración de esta memoria se han abordado diversos aspectos teóricos y técnicos relacionados con las redes definidas por software, pues tener un panorama amplio y claro de las tecnologías y conceptos que intervienen en el diseño, despliegue y operación de una red basada en este paradigma así lo requiere, ya que al ser una tecnología, si bien no reciente, pero sí aún desconocida para muchos profesionales de las telecomunicaciones es indispensable aclarar que es y de que va este nuevo enfoque. En las siguientes líneas se presentarán las conclusiones de este trabajo y las líneas de investigación a futuro.

7.1 Conclusiones

El esquema actual de Internet es sencillo y simple, basa los servicios superiores e inferiores sobre la capa de red, sin embargo esto provoca un cuello de botella precisamente en este punto y que no está preparado para el crecimiento que están teniendo tecnologías como el Internet de las Cosas, *Big Data*, Inteligencia Artificial, comunicaciones máquina a máquina, Infraestructura/Plataforma/Software como servicio, etc. Pues dicho esquema no contemplaba tal cantidad de información, generándose a velocidades nunca antes pensadas y requiriendo altos niveles de servicio.

La Internet debe evolucionar para poder dar cabida a las tecnologías emergentes o podrá colapsarse, es por ello imprescindible contemplar mecanismos que permitan la gestión de los servicios de manera eficiente, eficaz y capaz de ser reactiva. SDN presenta una revolución al modelo tradicional de las redes de telecomunicaciones que es sobre lo que se basa la Internet, pues separa el plano de control y plano de datos permitiendo un control centralizado y uso más eficiente de los recursos, además facilita la gestión de la red y es capaz de ofrecer respuestas reactivas y proactivas a cambios en la topología. La programabilidad de la red es uno de sus puntos fuertes, pues facilitará el desarrollo de soluciones a la medida, evitando infrautilización de recursos y dejando de depender de fabricantes.

Los escenarios descritos en este documento son importantes desde el punto de vista de un primer acercamiento con la tecnología SDN, pues en ellos se describe en primera instancia el proceso de comunicación entre el controlador (plano de control) y los switches SDN (plano de datos). Conocer y reconocer los mensajes involucrados en la negociación resulta imprescindible para entender cómo funciona este paradigma y de ser necesario, ser capaces de realizar *troubleshooting* en caso de falla.

La seguridad es un aspecto fundamental hoy en día, tanto para proteger los activos de las organizaciones como para cumplir con los marcos regulatorios al respecto, por esa situación, contar con medidas de seguridad en la red es un aspecto importante y que desde el diseño de la misma se debe contemplar. En el segundo escenario se implementaron dos

medidas bien conocidas al respecto: listas de acceso y *firewall,* que permiten fortalecer el control de la red a través de reglas que indicarán qué tráfico se permite y cual se deniega, esto con base en IPs, números de puertos o protocolos de aplicación, así seremos capaces de definir zonas desmilitarizadas y asegurar que solo los usuarios, procesos y servicios autorizados tengan acceso a los recursos. Una de las ventajas observables en este punto, es que no fue necesario agregar un equipo adicional que llevará a cabo estas tareas de seguridad, el controlador cuenta con la capacidad de añadir módulos con funciones específicas como estas y que, reitero, facilitan la administración de la red al concentrar en un punto varias funcionalidades y reduciendo costos al solo implementar aquellas que se requieren sin adquirir hardware adicional.

Como se ha indicado, Internet es hoy en día la base de las comunicaciones, por lo que contar con una red con acceso a Internet fue el planteamiento del escenario 3. El ancho de banda es un recurso caro y limitado, por lo que la gestión que se tenga de este tiene un papel fundamental en la prestación de servicios críticos y sensibles al retardo, por ello, en este escenario se plantearon dos medidas básicas para esta tarea: la implementación de políticas de calidad de servicio y limitación en cuanto a ancho de banda. Nuevamente, este esquema nos permitió integrar módulos para estas funciones específicas y lograr desde una misma consola la administración de varios servicios a la vez.

Sin duda alguna uno de los aspectos importantes de las SDNs es que reaccionan a los cambios en la topología, permitiendo que la red converja nuevamente en tiempos menores al ser el controlador el encargado de mantener actualizadas las tablas de flujo gracias a su visión general de la red. En el escenario 4 se pudo constatar dicha funcionalidad a través del módulo de *forwarding* con el que cuenta el controlador empleado.

Por último, el escenario 5 se trató de acercar a la implementación de un servicio lo más próximo a la realidad, al conectar a través de Internet tres sucursales distintas de una organización, para ello, cada sucursal debía contar únicamente con acceso a Internet y la sucursal central además con conexión al controlador. Mediante la configuración de túneles se pudo lograr la interconexión sin necesidad de contratar enlaces dedicados y adquirir infraestructura de capa 3, simplemente con la conexión a Internet se construyeron dichos túneles a través de los cuales las sucursales tenían comunicación con el controlador y con las otras sucursales.

Es importante resaltar el hecho de que las características del controlador elegido (Floodlight) permitieron agregar funcionalidades a la red sin requerir mayor actividad que configurar el modulo destinado a dicha función, así mismo se pudo constatar la eficiencia con la que trabaja el controlador, pues la interfaz con la que se le envían instrucciones es a través de línea de comandos, que hace uso de menos recursos y por ende, con mayor velocidad.

Una de las conclusiones más relevante es el hecho de que pese a tener casi una década de haber sido desarrollado, la mayoría de las aplicaciones de SDN siguen siendo a través de línea de comandos y/o desarrollo de *scripts* en distintos lenguajes de programación. Si bien estos aspectos permiten desarrollar soluciones a la medida de los usuarios, me parece que a la vez dificulta la adopción de esta por parte de los profesionales que administran las redes de datos hoy en día, ya que, esta área ha sido liderada por muchos años por empresas con un amplio control y conocimiento del mercado que se encuentran desarrollando plataformas comerciales con este enfoque y que facilitan su adopción gracias a dos factores: interfaces amigables y como dije: amplio conocimiento del mercado. Dichas empresas conocen bien a sus clientes y saben cómo introducir soluciones novedosas como esta, saben qué es lo que buscan en una solución y que tenga el respaldo de grandes fabricantes. Adicional a ello, actualmente existe poco personal calificado para la administración de redes SDN, que si bien muchos conceptos no cambian respecto a la arquitectura tradicional, si es importante un proceso de capacitación con esta nueva tecnología y que muchas empresas no están dispuestas a asumir, pues la tecnología actual les funciona y bien, el cambiar de paradigma conlleva una inversión tanto en infraestructura como en capacitación de personal y además, se deberán asumir los riesgos asociados a un cambio de este tipo y que sí no se lleva a cabo de la mano de expertos en el área pudiera resultar desastrosa.

7.2 Trabajos futuros

SDN tiene aún muchos campos de aplicación por ser explorados y que podrían representar un impulso adicional para esta nueva tecnología como lo son:

- ✓ Integración de redes físicas y virtualizadas: Actualmente es posible integrar dispositivos físicos y virtualizados dentro de una solución SDN, sin embargo el listado de equipos físicos compatibles con entornos de virtualización es limitado, por lo que los esfuerzos por que cada vez más equipos que componen esta lista se incrementen son importantísimos.
- ✓ Redes híbridas: Dado que el modelo actual de la Internet ha sido utilizado por muchos años, es importante contar con dispositivos que puedan integrar ambos enfoques y con esto permitir una transición más sencilla y en la medida de lo posible, transparente para los usuarios.
- ✓ Voz: Hay poca documentación relacionada a la implementación de servicios de voz sobre redes SDN, sin embargo éste es un servicio que en los últimos años ha tenido un crecimiento importante por lo que se deberá considerar su integración y desarrollo de *northbound* APIs para su gestión.
- ✓ Dispositivos móviles e IoT: Con la explosión que ha habido del uso de dispositivos móviles, las comunicaciones maquina a máquina derivada del IoT y tecnologías como la *wearable technology* hacen atractiva la capacidad de poder gestionarlos a través de un

esquema SDN y explotar los beneficios que ofrece. Hoy en día existe una línea de investigación de la ONF orientada a la habilitación de OpenFlow en redes inalámbricas y dispositivos móviles, misma que se puede consultar en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-wireless-mobile.pdf

- ✓ *Streaming*: Al igual que con el servicio de voz, el servicio de *streaming* es sensible al retardo y por ello, las gestiones en cuanto a calidad de servicio y garantías en cuanto a ancho de banda son importantes para una experiencia de servicio satisfactoria. Actualmente existen proyectos como GENI CINEMA que se define como un servicio de *streaming* de video en vivo pensado para las redes SDN.
- ✓ Monitoreo: Actualmente no existen plataformas lo suficientemente robustas para el monitoreo de controladores y switches SDN como para competir con las ya existentes en las redes convencionales. Un primer proyecto, Avior, que se puede integrar con algunos controladores SDN que cuenten con interfaces REST API. En estos momentos Avior solo proporciona a través de su interfaz GUI información básica referente a los dispositivos de la red y permite gestionar flujos de manera gráfica, prescindiendo de comandos Python o del envío de documentos (por ejemplo, json para Floodlight) a través de línea de comandos.

Aún hay muchos aspectos en donde las SDNs pudieran tener un impacto positivo y relevante, pues es un esquema que fácilmente se puede adaptar al crecimiento de las redes de comunicaciones dada su naturaleza distribuida y orientada hacia el nuevo enfoque de la Internet. Sin duda el desarrollo de interfaces hacia el norte es crucial para que la adopción y transición se dé de manera menos traumática y más transparente para los usuarios.

Apéndice I: Descripción e instalación de herramientas

Para la elaboración de esta memoria se requirió la instalación de diversas herramientas sobre un equipo de cómputo con las siguientes características:

Computador portátil, 8 GB en memoria RAM y 1 TB de capacidad en disco duro, sistema operativo Windows 10 Home Edition de 64 bits.

A) VirtualBox

Tabla 4. Características de la aplicación instalada VirtualBox

Descripción	Software de virtualización de sistemas operativos
Tipo de licencia	Gratuita (uso personal o de evaluación)
Versión instalada	5.1.22
URL descarga	https://www.virtualbox.org/wiki/Downloads
Motivos	Herramienta gratuita con 10 años en el mercado, lo que implica un periodo largo de prueba y amplia documentación sobre su uso.
Observaciones	Ninguna

Pasos para la instalación:

- Dirigirse a la URL de descarga
- Seleccionar la opción correspondiente al sistema operativo anfitrión
- Una vez descargado, ejecutar la aplicación y proceder con la instalación guiada por el *wizard*. La instalación resulta muy intuitiva y es recomendable realizarla con los parámetros por default.

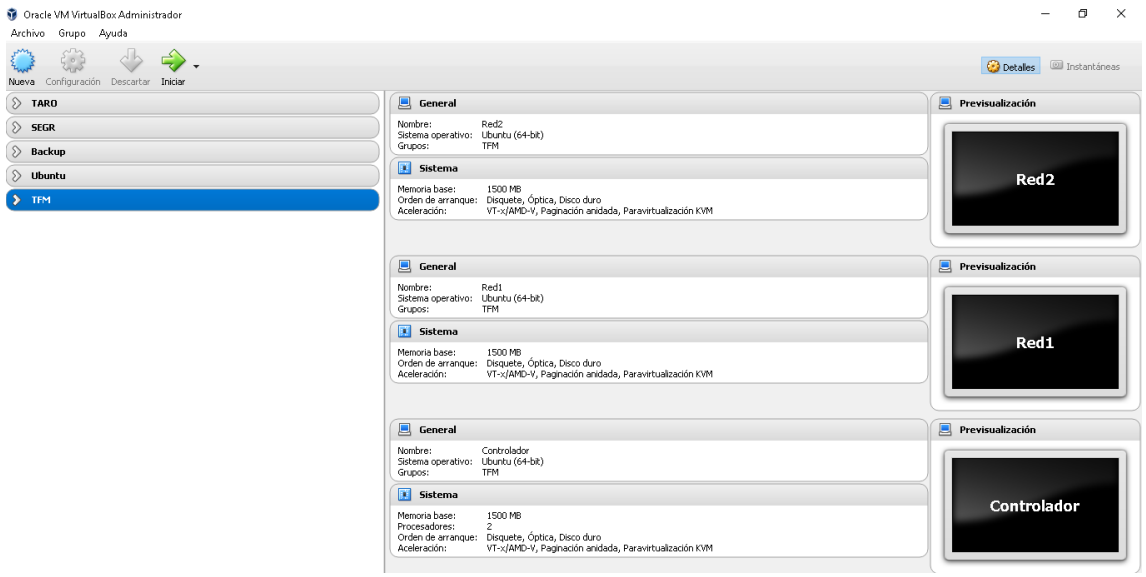


Figura 56.Interfaz de VirtualBox

B) Sistema operativo invitado

Tabla. 5 Características de la aplicación instalada Sistema Operativo invitado

Descripción	Se trata del sistema a ejecutar en las máquinas virtuales. En este caso me decanté por un sistema virtualizado que incluye Ubuntu y Floodlight (Controlador SDN)
Tipo de licencia	Software libre
Versión instalada	Ubuntu 14.04 y Floodlight 1.2
URL descarga	http://www.projectfloodlight.org/download/
Motivos	Utilizar un servicio virtualizado como este nos ahorra un paso: instalación del controlador.
Observaciones	Las credenciales de acceso son: Usuario: floodlight Contraseña: floodlight En caso de optar por la instalación por separado habrá que descargar el archivo ISO de Ubuntu (https://www.ubuntu.com/download/desktop) y la instalación nativa del controlador (Guía de instalación: https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Installation+Guide)

Pasos para la instalación:

- Acudir a la URL y descargar el archivo ZIP "Floodlight VM".
- Una vez descargado, se deberá descomprimir el archivo, con lo que obtendremos el archivo VMDK.
- Abrir VirtualBox → Dar clic en *Nueva* → Indicar el nombre a asignar y seleccionar el sistema operativo (Ubuntu 64 bits)

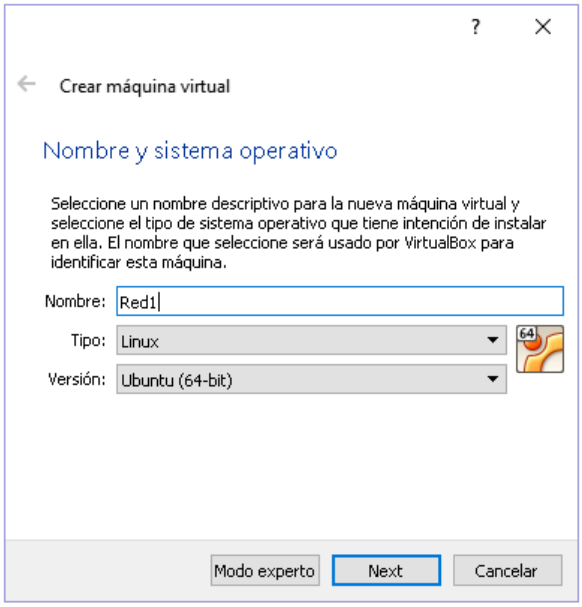


Figura 57. Creación de máquina virtual

➢ En la ventana siguiente, dejamos la asignación de memoria RAM con los valores por defecto.

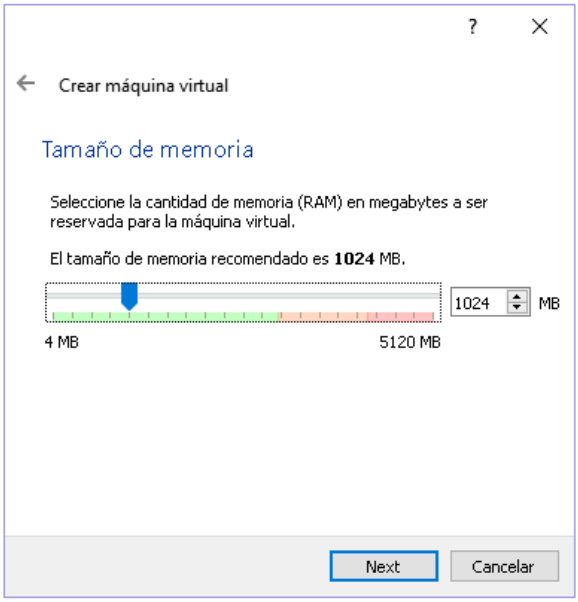


Figura 58.Asignación de memoria RAM a la máquina virtual

➢ El paso más importante viene a continuación, donde indicaremos que se utilice un disco virtual existente y que corresponde al archivo VMDK descargado previamente.

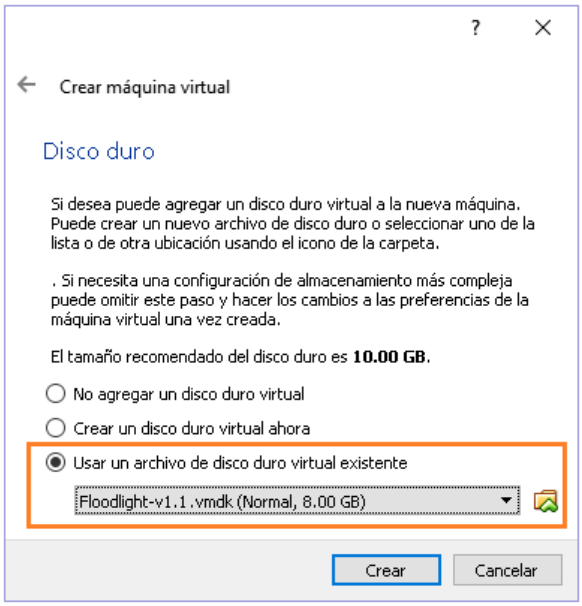


Figura 59.Asociación del disco duro virtual con la máquina virtual

➢ Al dar clic en "Crear" hemos finalizado con la configuración del primer sistema operativo invitado, dicho proceso se tendrá que repetir tantas veces como sistemas invitados requiramos.

B.1 Configuración y puesta en marcha del controlador Floodlight

Como se comentó con anterioridad, el sistema operativo invitado ya cuenta con el controlador Floodlight instalado, para iniciarlo es necesario ingresar los siguientes comandos en la terminal:

```
$cd floodlight
$ sudo java -jar target/floodlight.jar
```

Después de ejecutar los comandos anteriores, tendremos en la terminal el log del controlador, donde veremos entradas relacionadas a las acciones que lleva a cabo, como son envío de mensajes, detección de nuevos equipos en la red, modificación de tablas de flujo, etc.

Es importante identificar el archivo *floodlightdefault.properties* que se ubica en la ruta: */home/floodlight/floodlight/src/main/resources*, el cual es un documento de texto que contiene la configuración por defecto del controlador y que podemos editar en caso de así requerirlo.

Dicho documento contiene algunas líneas que vale la pena revisar:

- Campos que serán tomados en cuenta para realizar el reenvío de paquetes.

```
net.floodlightcontroller.forwarding.Forwarding.match=vlan, mac, ip, transport
```

- Puerto a través del cual los switches tendrán comunicación con el controlador (por defecto es el 6653)

```
net.floodlightcontroller.core.internal.FloodlightProvider.openflowPort=6653
```

- Puerto e IP para el acceso a la interfaz gráfica, cuya URL es: http://10.0.2.10:8080/ui/index.html

```
net.net.floodlightcontroller.restserver.RestApiServer.httpPort=8080
net.floodlightcontroller.restserver.RestApiServer.host=10.0.2.10
```

- Indican sí se requiere autenticación al usuario para ingresar a la GUI y sí ésta puede ser accesada a través de los protocolos HTTP y/o HTTPS.

```
net.floodlightcontroller.restserver.RestApiServer.httpsNeedClientAuthentication=NO
net.floodlightcontroller.restserver.RestApiServer.useHttps=NO
net.floodlightcontroller.restserver.RestApiServer.useHttp=YES
```

Dicha interfaz está compuesta por tres secciones (Ver Fig. 55):

Controller status: Proporciona información referente al tiempo en línea y módulos cargados en el controlador.

Topology: A manera de mapa muestra la topología implementada.

Switches y *Hosts*: Proporciona el listado de dispositivos que conforman la solución y sus datos más relevantes.

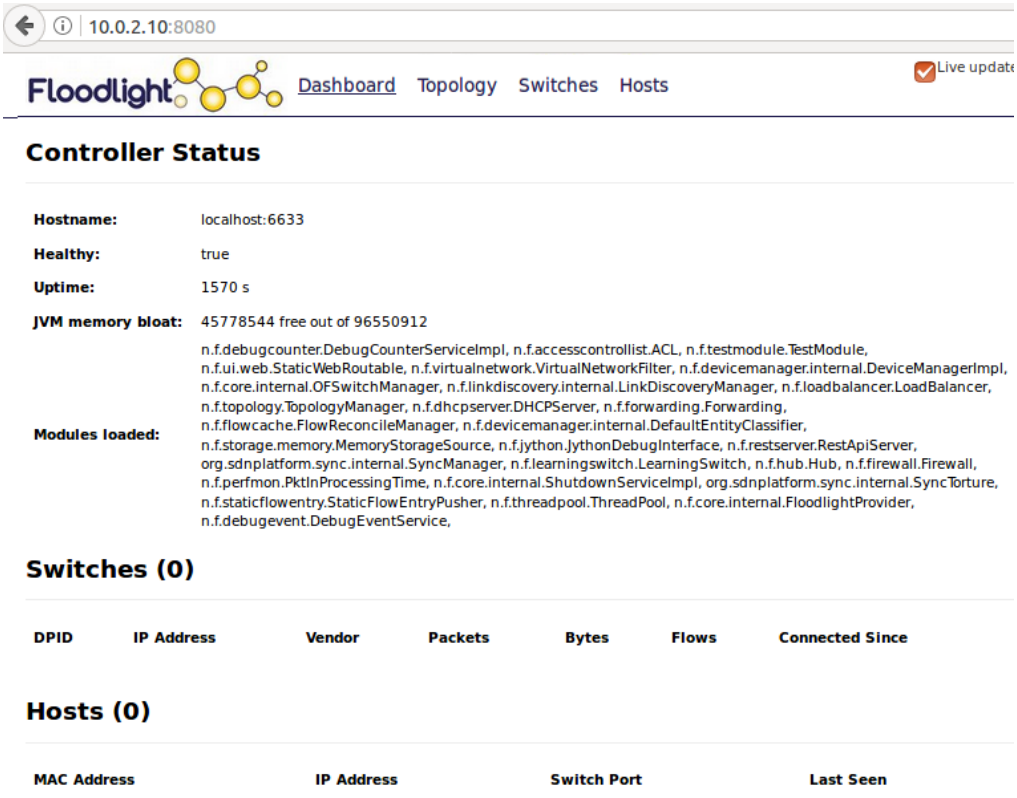


Figura 60.GUI de Floodlight

C) cURL

Tabla 6. Características de la aplicación instalada cURL

Descripción	Software para la transferencia de archivos vía HTTP, FTP, Telnet, etc.
Tipo de licencia	Software libre
Versión instalada	7.30
URL descarga	https://curl.haxx.se/download.html
Motivos	A través de esta herramienta podremos interactuar con el controlador Floodlight para la inserción o eliminación de flujos.
Observaciones	No hay.

Pasos para la instalación:

- Ingresamos los siguientes comandos en una terminal en el sistema operativo invitado:

```
$ sudo apt-get install curl
```

D) Wireshark

Tabla 7. Características de la aplicación instalada Wireshark

Descripción	Analizador de protocolos
Tipo de licencia	Software libre
Versión instalada	2.2.6
URL descarga	https://www.wireshark.org/#download
Motivos	Herramienta potente y de gran popularidad en el mercado. Manejo sencillo.
Observaciones	La instalación se realizó de manera nativa dentro de las máquinas virtuales generadas previamente.

Pasos para la instalación y ejecución:

➢ Ingresamos los siguientes comandos en una terminal en el sistema operativo invitado:

```
$ apt-get install wireshark
$ sudo wireshark
```

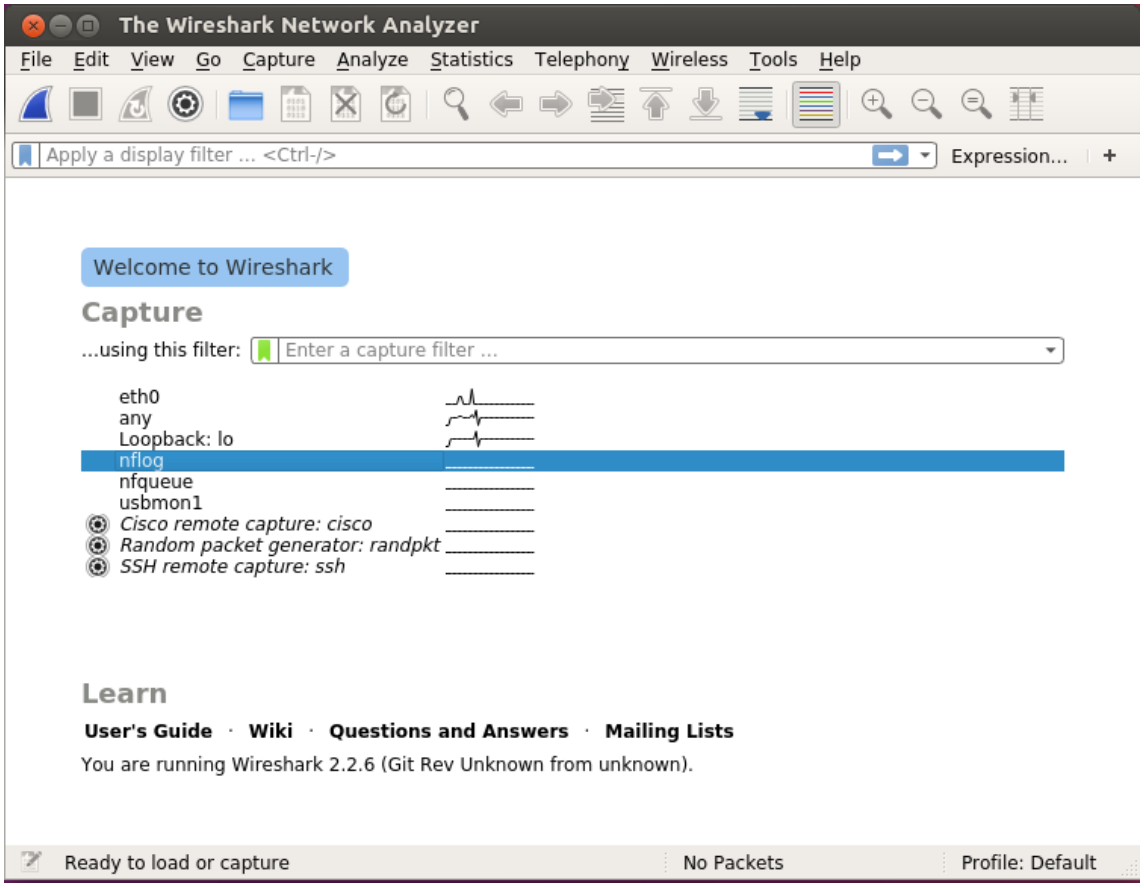


Figura 61. Interfaz de Wireshark

E) Mininet

Tabla 8. Características de la aplicación instalada Mininet

Descripción	Emulador de redes de datos
Tipo de licencia	Software libre
Versión instalada	2.2.1rc1
URL descarga	http://mininet.org/download/
Motivos	Es una herramienta sencilla, ligera y potente para emular encaminadores, conmutadores, dispositivos finales y los enlaces entre ellos. Permite generar topologías complejas totalmente personalizables, concurrentes, óptimo rendimiento y cambios en tiempo real.
Observaciones	De la misma manera que floodlight, se ofrece un servicio virtualizado que tiene embebida esta herramienta (Guía de instalación: http://mininet.org/download/#option-1-mininet-vm-installation-easy-recommended). Sin embargo en este caso ya se cuenta con el sistema invitado, por lo que se procedió a instalar de manera nativa.

Pasos para la instalación y ejecución:

- Ingresamos los siguientes comandos en una terminal en el sistema operativo invitado:

```
$ sudo apt-get install git
$ git clone git://github.com/mininet/mininet
$ cd mininet
$ git tag
$ git checkout –b 2.2.2rc1 2.2.2rc1
$ cd mininet/util/install.sh
$ sudo mn
```

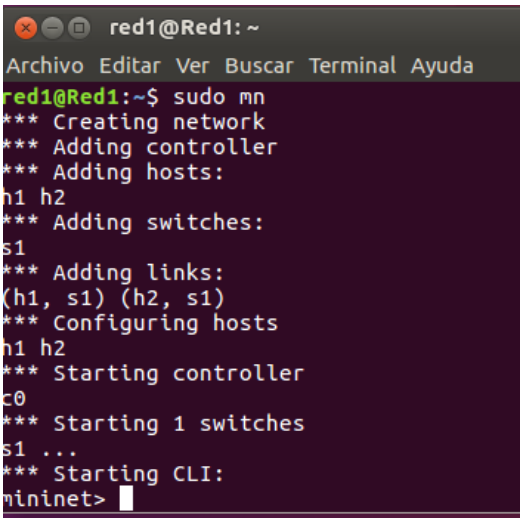


Figura 62. Interfaz de Mininet

F) MiniEdit

Tabla 9. Características de la aplicación instalada MiniEdit

Descripción	Emulador de redes de datos vía interfaz gráfica
Tipo de licencia	Software libre
Versión instalada	2.2.0.1
URL descarga	No aplica
Motivos	Es un buen primer acercamiento con Mininet, ya que ofrece la posibilidad de generar topologías sencillas a través de la interfaz gráfica.
Observaciones	Permite generar topologías sencillas, mismas que pueden ser guardadas en archivos *.py, los cuales se pueden editar para agregar mayor complejidad según se requiera.

Pasos para la instalación y ejecución:

- Ingresamos los siguientes comandos en una terminal en el sistema operativo invitado:

```
$ cd /mininet/examples
$ sudo python miniedit.py
```

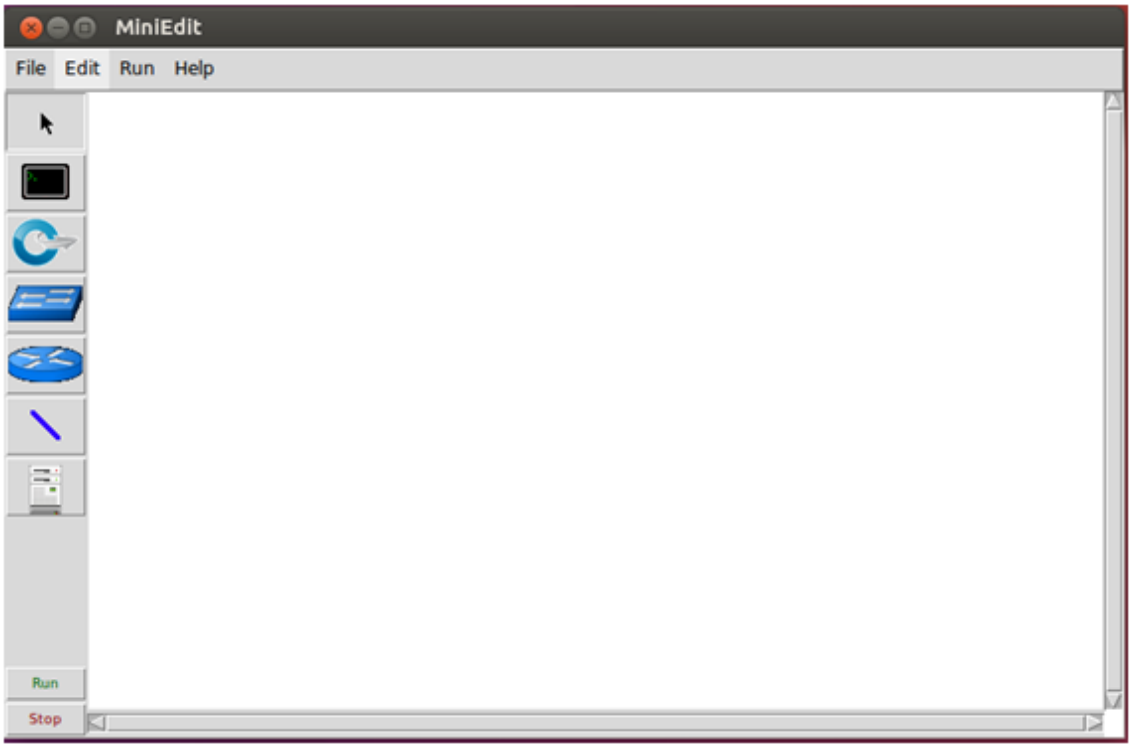


Figura 63. Interfaz de MiniEdit

G) OpenvSwitch

Tabla 10. Características de la aplicación instalada OpenvSwitch

Descripción	Servicio para la virtualización de switches
Tipo de licencia	Software libre
Versión instalada	2.7.0
URL descarga	http://openvswitch.org/releases/openvswitch-2.7.0.tar.gz
Motivos	Es la herramienta de virtualización de switches OpenFlow más utilizada y por ende, con mayor documentación disponible.
Observaciones	No hay.

Pasos para la instalación y ejecución:

- Ingresamos los siguientes comandos en una terminal en el sistema operativo invitado:

```
$ wget http://openvswitch.org/releases/openvswitch-2.7.0.tar.gz
$ tar zxvf openvswitch-2.7.0.tar.gz
$ cd openvswitch-2.7.0/
$ ./configure --prefix=/usr --with-linux=/lib/modules/`uname -r`/build
$ make
$ make install
$ make modules_install
$ rmmod openvswitch
$ depmod -a
$ /etc/init.d/openvswitch-controller stop
$ update-rc.d openvswitch-controller disable
$ /etc/init.d/openvswitch-switch start
```

Apéndice II: Guía básica de uso MiniEdit

La interfaz de usuario es bastante sencilla e intuitiva, sin embargo es conveniente dar una revisión rápida al entorno que ofrece dicha herramienta.

1. Interfaz

En la siguiente figura se muestran las principales áreas de la interfaz de MiniEdit:

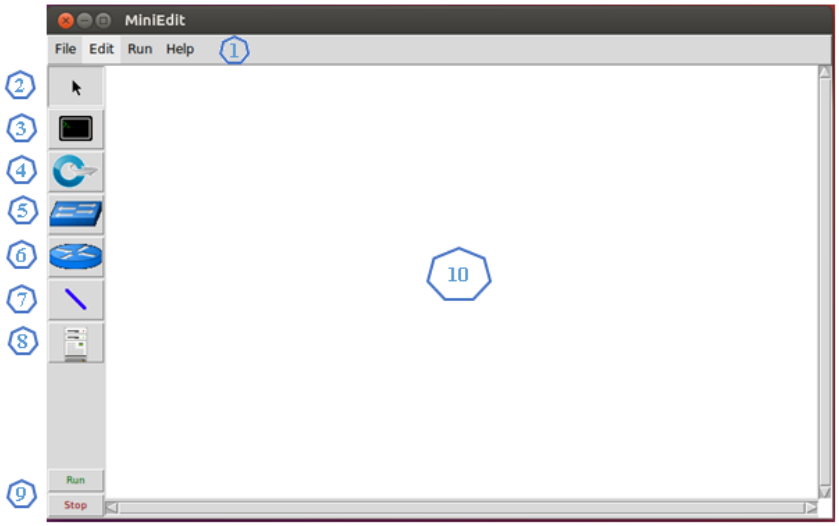


Figura 64. Interfaz de MiniEdit

1.1 Barra de opciones

Con el identificador número 1, consta de las opciones (Ver Fig. 61):

- File: Permite crear (New), abrir (Open) o guardar (Save) la topología MiniEdit, en un archivo con extensión *.mn, además permite exportarla (Export Level 2 Script) como script, con extensión *.py, el cual podremos ejecutar posteriormente siempre y cuando contemos con el intérprete de Python correspondiente.
- Edit: Permite modificar las preferencias del escenario (Preferences), como son: el segmento base, habilitar línea de comandos Mininet (Start CLI) al ejecutar el escenario, el tipo switch SDN por default, la versión de OpenFlow, etc.

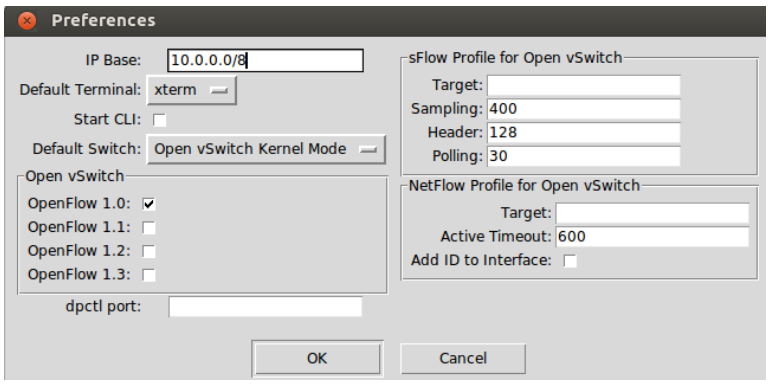


Figura 65. Submenú Preferences

➢ Run: Dado que MiniEdit (al igual que Mininet) permiten visualizar en tiempo real el comportamiento de la topología, tenemos opciones para ejecutar (Run) y detener (Stop) el escenario. Además con la opción "Show OVS Summary" muestra un resumen de los switches del escenario (IP, estado, interfaces, etc.), por otro lado con "Root Terminal" se abrirá una consola con privilegios de superusuario en caso de que requiramos realizar modificaciones en el sistema a través de línea de comandos.

➢ Help: En esta opción, encontraremos información relacionada a la versión que se está ejecutando.

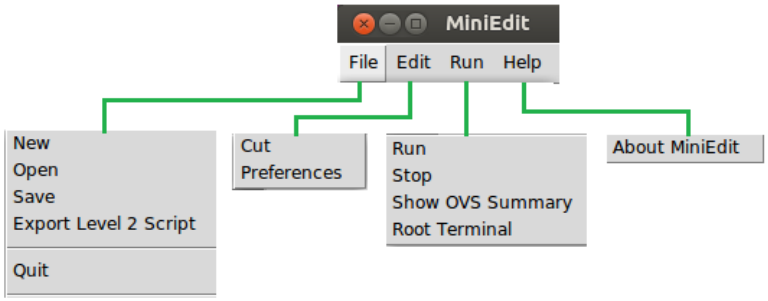


Figura 66. Detalle de la barra de opciones de MiniEdit

1.2 Barra de herramientas

Con el identificador 2, la herramienta de Selección que nos permite mover los elementos de la topología.

Con los identificadores 3 al 8, están los elementos que podemos utilizar en el escenario:

➢ Host (3): Representa los dispositivos finales de la red, que pueden ser computadores, servidores, dispositivos móviles, etc.

➢ Switch SDN (4): Dispositivo que, dentro del paradigma SDN, lleva a cabo las funciones del plano de datos.

➢ Legacy Switch (5): Dispositivo tradicional de conmutación de fragmentos.

➢ Legacy Router (6): Dispositivo tradicional de conmutación de paquetes.

➢ NetLink (7): Representan el cableado que une a los dispositivos de la topología.

➢ Controller (8): Dispositivo que, dentro del paradigma SDN, lleva a cabo las funciones del plano de control.

Con el identificador 9 tenemos nuevamente botones para ejecutar y detener el escenario.

1.3 Espacio de trabajo

Con el identificador 10, es el área sobre la cual podremos construir el escenario.

2. Creación de un escenario básico

En las siguientes páginas se mostrará paso a paso como generar un escenario básico utilizando esta herramienta.

2.1 Definición de la topología

Desde la barra de herramientas, seleccionaremos un controlador, un switch SDN y un host los cuales arrastraremos hacia el área de trabajo y los cuales uniremos a través de enlaces, como se muestra en la siguiente figura:

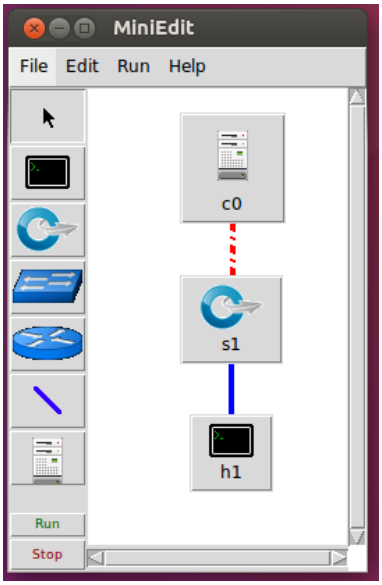


Figura 67. Creación de un escenario básico con MiniEdit

2.2 Configurar preferencias del escenario

En el menú Edit → Preferences, indicaremos las opciones del escenario, en este caso solo definiremos el segmento a utilizar, activaremos la consola e indicaremos la versión 1.3 de OpenFlow.

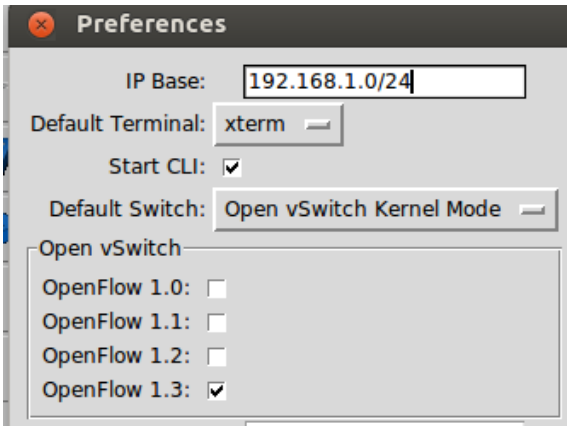


Figura 68. Configuración de Preferencias del escenario en MiniEdit

2.3 Ejecutar el escenario

A través del botón "Run" en la barra de herramientas ejecutaremos el escenario, con lo cual en segundo plano se desplegará en una consola la interfaz de Mininet.

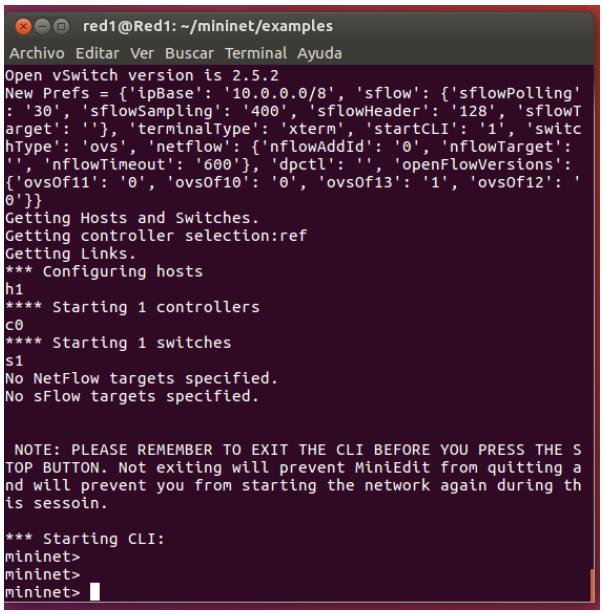


Figura 69. Interfaz de Mininet en segundo plano

2.4 Manipulación del escenario

Desde la interfaz de Mininet podemos ejecutar comandos que permiten estudiar el escenario:

- dump: Muestra el resumen de los dispositivos que conforman el escenario, su IP y el ID de proceso relacionado.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=12578>
<customOvs s1: lo:127.0.0.1,s1-eth1:None pid=12580>
<Controller c0: 127.0.0.1:6633 pid=12583>
```

Figura 70. Comando *dump*

- net: Muestra las conexiones existentes los dispositivos.

```
mininet> net
h1 h1-eth0:s1-eth1
s1 lo:  s1-eth1:h1-eth0
c0
```

Figura 71. Comando *net*

- nodes: Muestra el listado de nodos que componen el escenario.

```
mininet> nodes
available nodes are:
c0 h1 s1
```

Figura 72. Comando *nodes*

- xterm <hostname>: Abre una terminal de usuario para el dispositivo con el hostname indicado.

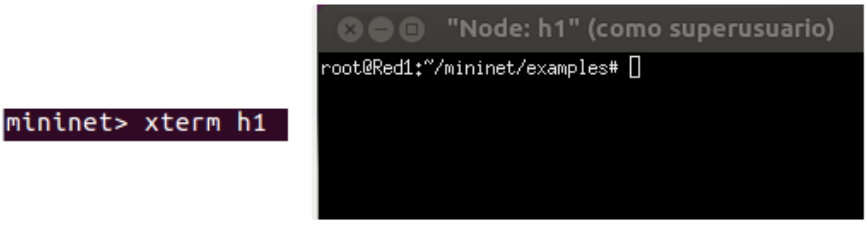


Figura 73. Comando *xterm <hostname>*

- pingall: Test de conectividad entre los dispositivos finales del escenario.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 nat0
h2 -> h1 h3 h4 nat0
h3 -> h1 h2 h4 nat0
h4 -> h1 h2 h3 nat0
nat0 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

Figura 74. Comando *pingall*

- sh ovs-ofctl dump-flows <hostaname>: Muestra las tablas de flujo para el switch indicado.

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.990s, table=0, n_packets=1, n_bytes=42, idl
 idle_age=2, priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=26
e3,dl_dst=6e:7e:53:0d:40:fd,arp_spa=10.0.2.2,arp_tpa=10.0.2.11,arp_
output:1
 cookie=0x0, duration=2.984s, table=0, n_packets=1, n_bytes=98, idl
 idle_age=2, priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=6
:fd,dl_dst=26:81:77:8b:b3:e3,nw_src=10.0.2.11,nw_dst=10.0.2.2,nw_to
=8,icmp_code=0 actions=output:2
```

Figura 75. Comando sh ovs-ofctl dump-flows <hostname>

- help: Lista los comandos de los cuales se puede hacer uso.

```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig
```

Figura 76. Comando *help*

- quit/exit: Cierra la terminal de Mininet. En este punto es importante mencionar que al estar trabajando con Mininet +MiniEdit para detener el escenario primero se deberá ingresar el comando "quit" en la terminal Mininet y posteriormente procederemos a dar clic en el botón "Stop" para detener completamente el escenario.

Como se mencionó en líneas anteriores, es posible guardar el escenario como un script, que podrá ser ejecutado en cualquier otro dispositivo siempre y cuando cuente con el intérprete de Python. Para obtener el script debemos dirigirnos a la pestaña *File* → *Export Level 2 Script* y posteriormente indicar la ruta donde deseamos sea almacenado el archivo con extensión *.py

Para ejecutar el escenario desde el script, es necesario abrir una terminal, dirigirse al directorio donde se encuentra el archivo e ingresar el siguiente comando:

```
$ sudo python archivo.py
```

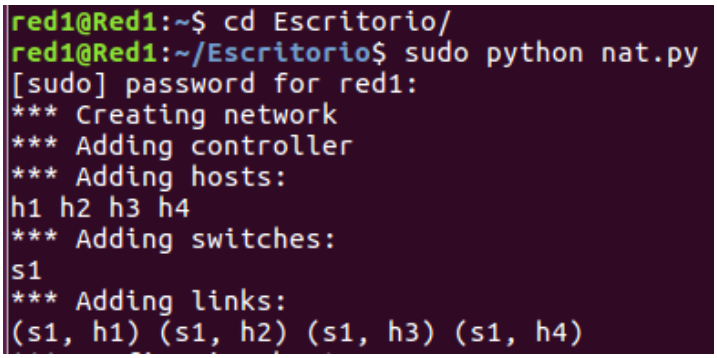


Figura 77. Ejecución de escenario Mininet desde script

En la siguiente figura se muestra un ejemplo de script y donde se comentan (indicado con el símbolo #) la función que lleva a cabo cada bloque del código:

```
# Se importan librerias a utilizar
from mininet.net import Mininet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.log import setLogLevel, info
# Definiendo la topologia
def Net():
    #Se indica que la topologia requerida es de tipo ''arbol'' con 1 switch
    #y 2 hosts por switch
    net = TreeTopo( depth=1, fanout=2 )
    #Se agrega en controlador, entre parentesis <hostname>
    net.addController( 'c0' )
    #Se agregan los host, entre parentesis <hostname>,<ip>
    h1 = net.addHost( 'h1', ip='10.0.0.1' )
    h2 = net.addHost( 'h2', ip='10.0.0.2' )
    #Se agrega el switch, entre parentesis <hostname>
    s3 = net.addSwitch( 's3' )
    #Se crean los enlaces entre el switch y los host
    net.addLink( h1, s3 )
    net.addLink( h2, s3 )
    #Se inicia el escenario
    net.start()
    #Se habilita la linea de comandos de Mininet
    CLI( net )
    #Secuencia de escape para detener el escenario
    net.stop()
#Loop para la ejecucion del escenario
if __name__ == '__main__':
    Net()
```

Figura 78. Ejemplo de script de Mininet

Por último, tras cerrar el escenario es recomendable limpiar los archivos temporales de la herramienta, para evitar conflictos en la ejecución de escenarios posteriores con el siguiente comando:

```
$ sudo mn -c
```

3. Configuración de controlador externo

Existen dos formas de indicar un controlador externo en la topología:

- MiniEdit: Dando clic derecho sobre el icono del controlador →*Properties* accederemos la configuración básica del controlador, donde deberemos indicar el puerto escucha del controlador en *Controller Port* (dependiendo del controlador, el puerto es distinto), seleccionar *Remote Controller* en *Controller Type* e indicar la IP del controlador. En caso de que la comunicación con el controlador no sea a través de TCP, se deberá indicar el protocolo correcto en el campo *Protocol*.

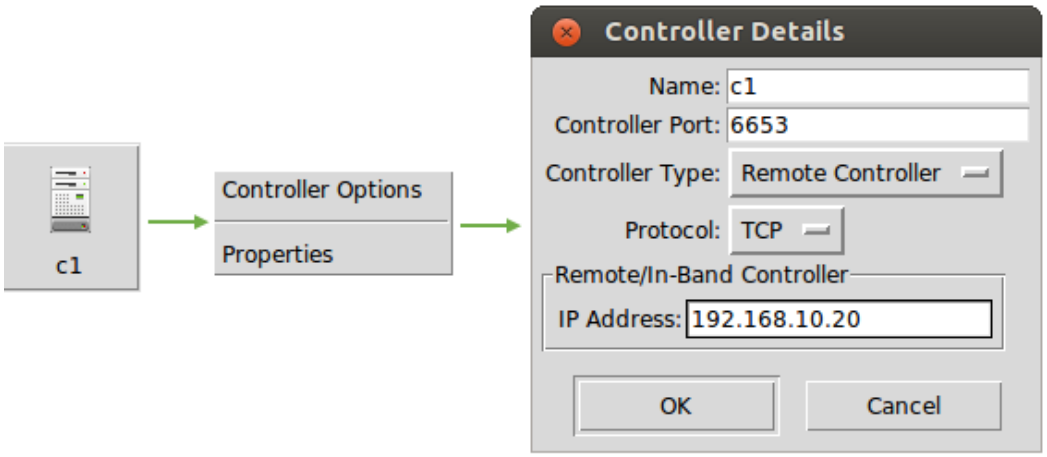


Figura 79. Configuración de controlador remoto a través de MiniEdit

- Script: Editando el archivo *.py de nuestro proyecto es posible indicar el uso de un controlador externo, como se muestra en la siguiente imagen:

```
c1=net.addController(name='c1',
                     controller=RemoteController,
                     ip='192.168.10.20',
                     protocol='tcp',
                     port=6653)
```

Figura 80. Configuración de controlador remoto a través del script del escenario

Apéndice III: Scripts de los escenarios

En este apartado, se presentan los scripts de los escenarios descritos en el capítulo 6 de esta memoria.

1. Escenario #1: Topología básica

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def escenario1():
    net = Mininet(topo=None, build=False, ipBase='192.168.0.0/16')

    info('*** Agregando controlador ***\n')
    c0=net.addController(name='c0',      controller=RemoteController,      ip='192.168.1.1',
protocol='tcp', port=6653)

    info('*** Agregando switch ***\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info('*** Agregando hosts ***\n')
    h1 = net.addHost('h1', cls=Host, ip='192.168.3.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='192.168.3.2', defaultRoute=None)

    info('*** Agregando enlaces ***\n')
    net.addLink(s1, h1)
    net.addLink(s1, h2)

    info('*** Iniciando topologia ***\n')
    net.build()

    info('*** Iniciando controlador ***\n')
    for controller in net.controllers:controller.start()

    info( '*** Iniciando switch***\n')
    net.get('s1').start([c0])

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    escenario1()
```

2. Escenario #2: *Hardening*: listas de acceso y firewall

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def escenario2():
   net = Mininet(topo=None, build=False, ipBase='10.0.0.0/16')

   info('*** Agregando controlador ***\n')
   c0=net.addController(name='c0',      controller=RemoteController,      ip='10.0.2.10',
protocol='tcp', port=6653)

   info('*** Agregando switches ***\n')
   s1 = net.addSwitch('direccion', cls=OVSKernelSwitch)
   s2 = net.addSwitch('secretarias', cls=OVSKernelSwitch)
   s3 = net.addSwitch('contabilidad', cls=OVSKernelSwitch)
   s4 = net.addSwitch('mantenimiento', cls=OVSKernelSwitch)

   info('*** Agregando hosts ***\n')
   h1 = net.addHost('h1', cls=Host, ip='10.0.3.1', defaultRoute=None)
   h2 = net.addHost('h2', cls=Host, ip='10.0.3.2', defaultRoute=None)
   h3 = net.addHost('h3', cls=Host, ip='10.0.4.1', defaultRoute=None)
   h4 = net.addHost('h4', cls=Host, ip='10.0.4.2', defaultRoute=None)
   h5 = net.addHost('h5', cls=Host, ip='10.0.5.1', defaultRoute=None)
   h6 = net.addHost('h6', cls=Host, ip='10.0.5.2', defaultRoute=None)
   h7 = net.addHost('h7', cls=Host, ip='10.0.6.1', defaultRoute=None)
   h8 = net.addHost('h8', cls=Host, ip='10.0.6.2', defaultRoute=None)

   info('*** Agregando enlaces***\n')
   net.addLink(s1, h1)
   net.addLink(s1, h2)
   net.addLink(s1, s2)
   net.addLink(s2, h3)
   net.addLink(s2, h4)
   net.addLink(s2, s3)
   net.addLink(s3, h5)
   net.addLink(s3, h6)
   net.addLink(s3, s4)
   net.addLink(s4, h7)
   net.addLink(s4, h8)

   info('*** Iniciando topologia ***\n')
```

…continúa en la siguiente página

```
    net.build()

    info('*** Iniciando controlador ***\n')
    for controller in net.controllers: controller.start()

    info( '*** Iniciando switches ***\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])
    net.get('s4').start([c0])

    info( '*** Configurando NAT ***\n')
    net.addNAT().configDefault()
    net.start()

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    escenario2()
```

3. Escenario #3: Conexión a Internet, calidad de servicio y administración del ancho de banda.

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def escenario3():
    net = Mininet(topo=None, build=False, ipBase='10.0.0.0/16')

    info('*** Agregando controlador ***\n')
    c0=net.addController(name='c0',      controller=RemoteController,      ip='10.0.2.10',
protocol='tcp', port=6653)

    info('*** Agregando hosts ***\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.3.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.3.2', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.4.1', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='10.0.4.2', defaultRoute=None)
```

…continúa en la siguiente página

```
    h5 = net.addHost('h5', cls=Host, ip='10.0.5.1', defaultRoute=None)
    h6 = net.addHost('h6', cls=Host, ip='10.0.5.2', defaultRoute=None)
    h7 = net.addHost('h7', cls=Host, ip='10.0.6.1', defaultRoute=None)
    h8 = net.addHost('h8', cls=Host, ip='10.0.6.2', defaultRoute=None)

    info('*** Agregando enlaces ***\n')
    net.addLink(s1, h1)
    net.addLink(s1, h2)
    net.addLink(s1, s2)
    net.addLink(s2, h3)
    net.addLink(s2, h4)
    net.addLink(s2, s3)
    net.addLink(s3, h5)
    net.addLink(s3, h6)
    net.addLink(s3, s4)
    net.addLink(s4, h7)
    net.addLink(s4, h8)

    info('*** Iniciando topologia ***\n')
    net.build()

    info('*** Iniciando controlador ***\n')
    for controller in net.controllers: controller.start()

    info( '*** Iniciando switches ***\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])
    net.get('s4').start([c0])

    info( '*** Configurando NAT ***\n')
    net.addNAT().configDefault()
    net.start()

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    escenario3()
```

4. Escenario #4: Respuesta reactiva a cambios en la topología

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def escenario4():

    net = Mininet( topo=None, build=False, ipBase='192.168.1.0')

    info('*** Agregando controlador ***\n')
    c0=net.addController(name='c0', controller=RemoteController, ip='192.168.2.10',
protocol='tcp', port=6653)

    info('*** Agregando switches ***\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch, mac='00:00:00:00:00:01')
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch, mac='00:00:00:00:00:03')
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch, mac='00:00:00:00:00:05')
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch, mac='00:00:00:00:00:08')
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch, mac='00:00:00:00:00:07')
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch, mac='00:00:00:00:00:06')
    s9 = net.addSwitch('s9', cls=OVSKernelSwitch, mac='00:00:00:00:00:09')
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch, mac='00:00:00:00:00:04')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch, mac='00:00:00:00:00:02')

    info('*** Agregando hosts ***\n')
    h1 = net.addHost('h1', cls=Host, ip='192.168.1.1', defaultRoute=None,
mac='00:00:00:00:00:11')
    h2 = net.addHost('h2', cls=Host, ip='192.168.1.2', defaultRoute=None,
mac='00:00:00:00:00:12')
    h3 = net.addHost('h3', cls=Host, ip='192.168.2.1', defaultRoute=None,
mac='00:00:00:00:00:21')
    h4 = net.addHost('h4', cls=Host, ip='192.168.2.2', defaultRoute=None,
mac='00:00:00:00:00:22')
    h5 = net.addHost('h5', cls=Host, ip='192.168.3.1', defaultRoute=None,
mac='00:00:00:00:00:31')
    h6 = net.addHost('h6', cls=Host, ip='192.168.3.2', defaultRoute=None,
mac='00:00:00:00:00:32')
    h7 = net.addHost('h7', cls=Host, ip='192.168.4.1', defaultRoute=None,
mac='00:00:00:00:00:41')
    h8 = net.addHost('h8', cls=Host, ip='192.168.4.2', defaultRoute=None,
mac='00:00:00:00:00:42')
    h9 = net.addHost('h9', cls=Host, ip='192.168.6.1', defaultRoute=None,
mac='00:00:00:00:00:61')
```

…continúa en la siguiente página

```
    h10 = net.addHost('h10', cls=Host, ip='192.168.6.2', defaultRoute=None,
mac='00:00:00:00:00:62')
    h11 = net.addHost('h11', cls=Host, ip='192.168.7.1', defaultRoute=None,
mac='00:00:00:00:00:71')
    h12 = net.addHost('h12', cls=Host, ip='192.168.7.2', defaultRoute=None,
mac='00:00:00:00:00:72')
    h13 = net.addHost('h13', cls=Host, ip='192.168.8.1', defaultRoute=None,
mac='00:00:00:00:00:81')
    h14 = net.addHost('h14', cls=Host, ip='192.168.8.2', defaultRoute=None,
mac='00:00:00:00:00:82')
    h15 = net.addHost('h15', cls=Host, ip='192.168.9.1', defaultRoute=None,
mac='00:00:00:00:00:91')
    h16 = net.addHost('h16', cls=Host, ip='192.168.9.2', defaultRoute=None,
mac='00:00:00:00:00:92')

    info('*** Agregando enlaces ***\n')
    net.addLink(s1, s2)
    net.addLink(s2, s3)
    net.addLink(s3, s4)
    net.addLink(s4, s5)
    net.addLink(s5, s1)
    net.addLink(h1, s1)
    net.addLink(s1, h2)
    net.addLink(h3, s2)
    net.addLink(s2, h4)
    net.addLink(s4, h7)
    net.addLink(s4, h8)
    net.addLink(s7, s6)
    net.addLink(s8, s7)
    net.addLink(s8, s6)
    net.addLink(s6, h9)
    net.addLink(s6, h10)
    net.addLink(s8, h13)
    net.addLink(h14, s8)
    net.addLink(h11, s7)
    net.addLink(h12, s7)
    net.addLink(s3, h5)
    net.addLink(s3, h6)
    net.addLink(s5, s6)
    net.addLink(s2, s5)
    net.addLink(s1, s4)
    net.addLink(s3, s9)
    net.addLink(s9, s8)
    net.addLink(s9, h16)
    net.addLink(s9, h15)

    info('*** Iniciando topologia ***\n')
    net.build()
```

…continúa en la siguiente página

```
info('*** Iniciando controlador ***\n')
for controller in net.controllers: controller.start()

info( '*** Iniciando switches ***\n')
net.get('s1').start([c0])
net.get('s3').start([c0])
net.get('s5').start([c0])
net.get('s8').start([c0])
net.get('s7').start([c0])
net.get('s6').start([c0])
net.get('s9').start([c0])
net.get('s4').start([c0])
net.get('s2').start([c0])

CLI(net)
net.stop()

if __name__ == '__main__':
   setLogLevel( 'info' )
   escenario4()
```

5. Escenario #5: *Tunneling*

5.1 Máquina virtual 1

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def nodo1():
   net = Mininet(topo=None, build=False, ipBase='172.0.1.0/16')

   info('*** Agregando controlador ***\n')
   c0=net.addController(name='c0', controller=RemoteController, ip='192.168.2.10',
protocol='tcp', port=6653)

   info('*** Agregando switches ***\n')
   s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
   s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
   s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
   s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
```

…continúa en la siguiente página

```
    info('*** Agregando hosts ***\n')
    h1 = net.addHost('h1', cls=Host, ip='172.0.1.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='172.0.1.2', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='172.0.1.3', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='172.0.1.4', defaultRoute=None)
    h5 = net.addHost('h5', cls=Host, ip='172.0.1.5', defaultRoute=None)
    h6 = net.addHost('h6', cls=Host, ip='172.0.1.6', defaultRoute=None)
    h7 = net.addHost('h7', cls=Host, ip='172.0.1.7', defaultRoute=None)
    h8 = net.addHost('h8', cls=Host, ip='172.0.1.8', defaultRoute=None)

    info('*** Agregando enlaces ***\n')
    net.addLink(s1, h1)
    net.addLink(s1, h2)
    net.addLink(s1, s2)
    net.addLink(s2, h3)
    net.addLink(s2, h4)
    net.addLink(s2, s3)
    net.addLink(s3, h5)
    net.addLink(s3, h6)
    net.addLink(s3, s4)
    net.addLink(s4, h7)
    net.addLink(s4, h8)
    info('*** Iniciando topologia ***\n')
    net.build()

    info('*** Iniciando controlador ***\n')
    for controller in net.controllers: controller.start()

    info( '*** Iniciando switches ***\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])
    net.get('s4').start([c0])

    info('*** Configurando tunel hacia VM 2 ***\n')
    s1.cmd('ovs-vsctl add-port s1 s1-gre1 -- set interface s1-gre1 type=gre
options:remote_ip=192.168.2.10')
    info('*** Configurando tunel hacia VM 3 ***\n')
    s1.cmd('ovs-vsctl add-port s1 s1-gre2 -- set interface s1-gre2 type=gre
options:remote_ip=192.168.3.30')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    nodo1()
```

5.2 Máquina virtual 2

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def nodo2():
    net = Mininet(topo=None, build=False, ipBase='172.0.2.0/16')

    info('*** Agregando controlador ***\n')
    c0=net.addController(name='c0', controller=RemoteController, ip='192.168.2.10',
protocol='tcp', port=6653)

    info('*** Agregando switches ***\n')
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch)

    info('*** Agregando hosts ***\n')
    h9 = net.addHost('h9', cls=Host, ip='172.0.2.1', defaultRoute=None)
    h10 = net.addHost('h10', cls=Host, ip='172.0.2.2', defaultRoute=None)
    h11 = net.addHost('h11', cls=Host, ip='172.0.2.3', defaultRoute=None)
    h12 = net.addHost('h12', cls=Host, ip='172.0.2.4', defaultRoute=None)
    h13= net.addHost('h13', cls=Host, ip='172.0.2.5', defaultRoute=None)
    h14= net.addHost('h14', cls=Host, ip='172.0.2.6', defaultRoute=None)

    info('*** Agregando enlaces ***\n')
    net.addLink(s5, h9)
    net.addLink(s5, h10)
    net.addLink(s5, s6)
    net.addLink(s6, h11)
    net.addLink(s6, h12)
    net.addLink(s6, s7)
    net.addLink(s7, h13)
    net.addLink(s7, h14)
    net.addLink(s7, s5)

    info('*** Iniciando topologia ***\n')
    net.build()

    info('*** Iniciando controlador ***\n')
    for controller in net.controllers: controller.start()
```

…continúa en la siguiente página

```
    info( '*** Iniciando switches ***\n')
    net.get('s5').start([c0])
    net.get('s6').start([c0])
    net.get('s7').start([c0])

    info('*** Configurando tunel hacia VM 1 ***\n')
    s5.cmd('ovs-vsctl add-port s5 s5-gre1 -- set interface s5-gre1 type=gre
options:remote_ip=192.168.1.10')
    info('*** Configurando tunel hacia VM 3 ***\n')
    s5.cmd('ovs-vsctl add-port s5 s5-gre2 -- set interface s5-gre2 type=gre
options:remote_ip=192.168.3.30')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    nodo2()
```

5.3 Máquina virtual 3

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def nodo3():
    net = Mininet(topo=None, build=False, ipBase='172.0.3.0/16')

    info('*** Agregando controlador ***\n')
    c0=net.addController(name='c0', controller=RemoteController, ip='192.168.2.10',
protocol='tcp', port=6653)

    info('*** Agregando switches ***\n')
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch)

    info('*** Agregando hosts ***\n')
    h15 = net.addHost('h15', cls=Host, ip='172.0.3.1', defaultRoute=None)
    h16 = net.addHost('h16', cls=Host, ip='172.0.3.2', defaultRoute=None)

    info('*** Agregando enlaces ***\n')
    net.addLink(s8, h15)
    net.addLink(s8, h16)
```

…continúa en la siguiente página

```
    info('*** Iniciando topologia ***\n')
    net.build()

    info('*** Iniciando controlador ***\n')
    for controller in net.controllers: controller.start()

    info( '*** Iniciando switches ***\n')
    net.get('s8').start([c0])

    info('*** Configurando tunel hacia VM 1 ***\n')
    s8.cmd('ovs-vsctl add-port s8 s8-gre1 -- set interface s8-gre1 type=gre
options:remote_ip=192.168.1.10')
    info('*** Configurando tunel hacia VM 2 ***\n')
    s8.cmd('ovs-vsctl add-port s8 s8-gre2 -- set interface s8-gre2 type=gre
options:remote_ip=192.168.2.10')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    nodo3()
```

Bibliografía

[1]Amin, R., Shah, N., Shah, B., & Alfandi, O. (27 de Enero de 2017). Auto-Configuration of ACL Policy in Case of Topology Change in Hybrid SDN.

[2]*Aprendiendo OpenDayLight*. (20 de Enero de 2015). Obtenido de https://aprendiendoodl.wordpress.com/2015/01/20/opendaylight-api-rest/

[3]Azodolmolky, S. (2013). *Software Defined Networking with OpenFlow*. Packt Publishing.

[4]Banks, E. (09 de Septiembre de 2014). *Network Computing*. Obtenido de http://www.networkcomputing.com/networking/software-defined-wan-primer/2018665838

[5]Birkeland, S. F. (Junio de 2016). *Norwegian University of Science and Technology*. Obtenido de https://brage.bibsys.no/xmlui/bitstream/handle/11250/2406871/15443_FULLTEXT.pdf?sequence=1

[6]Celenlioglu, M. R., & Mantar, H. A. (2014). A Scalable Routing and Admission Control Model in SDN-based Networks.

[7]Chi, P.-W., Wang, M.-H., Guo, J.-W., & Lei, C.-L. (2016). SDN Migration: An Efficient Approach to Integrate OpenFlow Networks with STP-enabled Networks.

[8]*cURL*. (s.f.). Obtenido de https://curl.haxx.se/

[9]Doherty, J. (2016). *SDN and NFV Simplified: A Visual Guide to Understanding Software Defined Networks and Network Function Virtualization*. Addison-Wesley Professional.

[10]Fernández, D. (2016). Virtualización de redes y de funciones de red. Madrid, España.

Fernández, D., & Bellido, L. (2016). Software-Defined Networking (SDN). Madrid, España.

[11]Fernández, D., Bellido, L., Ruiz, J., Walid, O., Mateos, V., & Villagra, V. (2012). Virtual Networks over linuX (VNX).

[12]*Floodlight*. (s.f.). Obtenido de http://www.projectfloodlight.org/floodlight/

[13]Goransson, P., Black, C., & Culver, T. (2016). *Software Defined Networks, 2nd Edition*. Morgan Kaufmann.

[14]*Grotto Networking*. (s.f.). Obtenido de https://www.grotto-networking.com/BBSDNOverview.html

[15]Han, L., Li, Z., Liu, W., Dai, K., & Qu, W. (2016). Minimum Control Latency of SDN Controller Placement.

[16]Jacob, D. (06 de Octubre de 2016). *Packet Design*. Obtenido de http://www.packetdesign.com/blog/sd-wan-vs-wan-sdn/

[17]Karaman, M. A., Gorkemli, B., Tatlicioglu, S., Komurcuoglu, M., & Karakaya, O. (2015). Quality of Service Control and Resource Prioritization with Software Defined Networking.

[18]Kumar, H., Gharakheili, H. H., & Sivaraman, V. (2013). User Control of Quality of Experience in Home Networks using SDN.

[19]Liang, C. (s.f.). *Department of Computer Science at Duke University*. Obtenido de http://www.cs.duke.edu/courses/fall14/compsci590.4/notes/slides_floodlight_updated.pdf

[20]Luca Davoli, L. V., Ventre, P. L., Siracusano, G., & Salsano, S. (s.f.). *Networking Group, University of Rome.* Obtenido de http://netgroup.uniroma2.it/Stefano_Salsano/papers/salsano-seg-routing-short-ewsdn-2015.pdf

[21]McCauley, J. (s.f.). *Open Networking Summit*. Obtenido de http://opennetsummit.org/archives/apr12/mccauley-mon-nox.pdf

[22]*Mininet*. (s.f.). Obtenido de http://mininet.org/

[23]Morreale, P. A., & Anderson, J. M. (2015). *Software Defined Networking.* CRC Press.

[24]Nadeau, T. D., & Gray, K. (2013). *SDN: Software Defined Networks.* O'Reilly Media, Inc.

[25]Olaya, M. E., Bernal, I., & Mejía, D. (2016). Application for Load Balancing in SDN.

[26]*Open Networking Foundation.* (27 de Mayo de 2014). Obtenido de https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-of-enabled-transport-sdn.pdf

[27]*Open Networking Summit*. (s.f.). Obtenido de http://opennetsummit.org/archives/apr12/vahdat-wed-sdnstack.pdf

[28]*OpenDaylight*. (s.f.). Obtenido de https://www.opendaylight.org/

[29]*OpenFlow Stanford University*. (s.f.). Obtenido de https://openflow.stanford.edu/display/ONL/POX+Wiki

[30]Open-Source Routing and Network Simulation. (05 de Diciembre de 2013). *Open-Source Routing and Network Simulation*. Obtenido de http://www.brianlinkletter.com/vnx-linux-network-simulator-review/

[31]*OpenvSwitch*. (s.f.). Obtenido de http://openvswitch.org/

[32]Pica8. (Marzo de 2015). *University of California Santa Cruz*. Obtenido de http://pleiades.ucsc.edu/doc/pica8/ovs-commands-reference.pdf

[33]Pujolle, G. (2015). *Software Networks.* John Wiley & Sons.

[34]*Python*. (s.f.). Obtenido de https://pypi.python.org/pypi/pox

[35]Qilin, M., & WeiKang, S. (2015). A Load BalancingMethod Based on SDN.

[36]*Scott's Weblog*. (30 de Octubre de 2012). Obtenido de http://blog.scottlowe.org/2012/10/30/running-host-management-on-open-vswitch/

[37]*SDN Hub*. (s.f.). Obtenido de http://sdnhub.org/tutorials/pox/

[38]*SDX Central*. (s.f.). Obtenido de https://www.sdxcentral.com/sd-wan/definitions/

[39]Sharafat, A. R., Saurav Das, G. P., & McKeown, N. (s.f.). *Department of Computer Science, Stanford University.* Obtenido de http://klamath.stanford.edu/~nickm/papers/mpls-sigcomm11.pdf

[40]*The Random Security Guy*. (28 de Diciembre de 2014). Obtenido de http://therandomsecurityguy.com/openvswitch-cheat-sheet/

[41]Tourrilhes, J., Sharma, P., Banerjee, S., & Petti, J. (s.f.). The Evolution of SDN and OpenFlow: A Standards Perspective.

[42]Yan, S., Aguado, A., Ou, Y., Wang, R., & Simeonidou, D. (Febrero de 2017). Multilayer Network Analytics With SDN-Based Monitoring Framework.

[43]Zope, N., Pawar, S., & Saquib, Z. (2016). Firewall and Load balancing as an application of SDN.