

**PROTOTIPO DE REDES IPV6 DEFINIDAS POR SOFTWARE MEDIANTE
MININET**

**BRYAN VALENCIA SUÁREZ
SANTIAGO SANTACRUZ PAREJA**

**UNIVERSIDAD CATÓLICA DE PEREIRA
FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA
INGENIERÍA DE SISTEMAS Y TELECOMUNICACIONES
PEREIRA
2015**

**PROTOTIPO DE REDES IPV6 DEFINIDAS POR SOFTWARE MEDIANTE
MININET**

**BRYAN VALENCIA SUÁREZ
SANTIAGO SANTACRUZ PAREJA**

Informe Final de Trabajo de Grado

**Directora:
Ing. Line Yasmín Becerra Sánchez**

**UNIVERSIDAD CATÓLICA DE PEREIRA
FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA
INGENIERÍA DE SISTEMAS Y TELECOMUNICACIONES
PEREIRA
2015**

***A Dios, por hacer tangible este nuevo logro
y permitirnos día a día hacer de
nuestras vidas caminos de felicidad.***

AGRADECIMIENTOS

Que maravilloso es poder resaltar el valioso papel de nuestros padres en este sueño que hicieron también suyo. Mil gracias a ellos por su apoyo y constancia sinceros, por creer en nosotros y estar de manera incondicional.

Gracias también de corazón a la docente Line Yasmín Becerra Sánchez, no sólo por brindarnos sus conocimientos y capacidades como tutora sino, más importante aún, por guiarnos y mostrarnos lo enriquecedor que puede ser adentrarse un poco más en los conocimientos de esta disciplina.

También agradecemos a todas las personas alrededor del mundo que en las diferentes comunidades colaborativas hicieron su aporte y estuvieron dispuestos a resolver nuestros interrogantes, en especial a Eder Leão Fernandes.

Y por último, gracias a cada uno de los compañeros y amigos que hicieron que nuestro recorrido por la universidad fuera una aventura inolvidable.

TABLA DE CONTENIDOS

	Pág.
TABLA DE CONTENIDOS.....	5
LISTA DE TABLAS	7
LISTA DE FIGURAS	8
LISTA DE ANEXOS	9
LISTA DE ACRÓNIMOS Y ABREVIACIONES	10
RESUMEN	12
INTRODUCCIÓN	14
1. OBJETIVOS.....	16
1.1. OBJETIVO GENERAL	16
1.2. OBJETIVOS ESPECÍFICOS.....	16
2. METODOLOGÍA.....	17
2.1. DISEÑO METODOLÓGICO	17
2.2. RECOLECCIÓN DE DATOS	18
3. MARCO TEÓRICO	20
3.1. ANTECEDENTES.....	20
3.2. REDES DEFINIDAS POR SOFTWARE (SDN)	24
3.2.1. Arquitectura	24
3.3. OPENFLOW	26
3.4. MININET	29
3.5. IPv4 e IPv6	30
3.5.1. Encabezado IPv4 e IPv6	31
3.5.2. Tipos de direcciones IPv6	34
3.5.3. Representación de una dirección IPv6	34
3.5.4. Estructura de direccionamiento IPv6	35
4. DESARROLLO DEL PROYECTO	40
4.1. Capacidades de la herramienta Mininet.....	40
4.2. Selección de la versión de OpenFlow.....	41
4.3. Pruebas con diferentes controladores	42
4.3.1. NOX13OFLIB	42

4.3.2. RYU.....	44
4.3.2.1. Prueba de enrutamiento usando RYU e IPv4	44
4.4. Pruebas en IPv4	48
4.4.1. Prueba usando iperf	48
4.4.2. Prueba de tráfico con servidor HTTP	49
4.5. Emulación de Redes Definidas por Software con IPv6.....	49
4.5.1. Prototipo No. 1	49
4.5.2. Prototipo No. 2	51
5. PRESENTACIÓN Y ANÁLISIS DE LOS RESULTADOS.....	56
5.1. Pruebas de conectividad con ping6	56
5.2. Pruebas con IPERF e IPv6	58
5.2.1. Prueba IPERF en Prototipo No. 1	59
5.2.2. Prueba IPERF en Prototipo No. 2	60
5.3. Comparativa entre IPv4 e IPv6 en diferentes controladores.....	61
5.4. Análisis de resultados y observaciones generales	63
6. RECOMENDACIONES Y CONCLUSIONES.....	66
LISTA DE REFERENCIAS.....	68
ANEXOS.....	71

LISTA DE TABLAS

	Pág.
Tabla 1. Descripción de los campos de la cabecera básica de IPv6	33
Tabla 2. Formatos comprimidos de direcciones IPv6.	35
Tabla 3. Protocolos y campos soportados por las diferentes versiones de OpenFlow	42
Tabla 4. Direcciones IPv6 del prototipo No. 2	53
Tabla 5. Direcciones IPv4 e IPv6 utilizadas para prueba comparativa	61
Tabla 6. Resultados de prueba comparativa entre IPv4 e IPv6	62
Tabla 7. Campos de IPv6 soportados por OpenFlow 1.3.	64

LISTA DE FIGURAS

	Pág.
Figura 1. Diseño de la investigación	17
Figura 2. Arquitectura de una SDN.	25
Figura 3. Ejemplo de un conjunto de instrucciones OpenFlow.	27
Figura 4. Comunicación entre el controlador y el switch a través de OpenFlow.	28
Figura 5. Emulación de red en MiniNet.....	29
Figura 6. Encabezado IPv4.....	32
Figura 7. Encabezado IPv6.....	32
Figura 8. Formato de direcciones unicast global.....	36
Figura 9. Estructura de Dirección Local Única.....	37
Figura 10. Formato de dirección de enlace local	37
Figura 11. Estructura de Dirección anycast	38
Figura 12. Formato de dirección multicast IPv6	39
Figura 13. Comando para iniciar el controlador nox13oflib.	43
Figura 14. Utilización de controlador remoto.	43
Figura 15. Topología emulada con OpenFlow 1.3 y el controlador RYU.	45
Figura 16. Eliminación y asignación de IP a host 1 en MiniNet	46
Figura 17. Ejecución del controlador RYU	47
Figura 18. Ajuste de direcciones mediante la API REST de RYU.....	48
Figura 19. Prueba con Iperf de SDN con OpenFlow 1.3.....	48
Figura 20. Creación y uso de un servidor HTTP en una red emulada con RYU.	49
Figura 21. Topología IPv6 No. 1	50
Figura 22. PING IPv6 entre dos host usando Nox13oflib y ofsoftswitch13	51
Figura 23. Topología IPv6 No. 2	52
Figura 24. Ejecución en Mininet de topología IPv6 No. 2	54
Figura 25. Ejecución RYU con la aplicación simple_switch_13.py.	54
Figura 26. Asignación de direcciones IPv6 en Mininet.....	55
Figura 27. Comprobación de conexiones entre dispositivos con función “links”	55
Figura 28. Ping IPv6 en Topología No. 2	56
Figura 29. Análisis de paquetes de datos para OpenFlow 1.3.....	57
Figura 30. ICMPv6 en emulación de SDN.	58
Figura 31. Prueba IPERF en Prototipo No. 1	59
Figura 32. Prueba IPERF en Prototipo No. 2.....	60

LISTA DE ANEXOS

	Pág.
Anexo A. Comandos utilizados en Mininet.....	71
Anexo B. Tutorial para la instalación Ubuntu, Mininet y OpenFlow 1.3 con NOX13OFLIB.....	74
Anexo C. Instalación del controlador RYU.....	88
Anexo D. Utilización de aplicación para el controlador RYU que da comportamiento de routers a los switches.....	91
Anexo E. Emulación de una topología simple configurada con IPv6 con el controlador Nox13oflib.....	98
Anexo F. Creación y utilización de servidor HTTP en una Red Definida por Software usando RYU.	101

LISTA DE ACRÓNIMOS Y ABREVIACIONES

SIGLAS	ESPAÑOL	INGLES
CLI	Interfaz de Línea de Comandos	Command line interface
DCAN	Control Descentralizado de las Redes ATM	Devolved Control of ATM Networks
HTTP	Protocolo de Transferencia de Hipertexto	Hypertext Transfer Protocol
IANA	Autoridad para la Asignación de Direcciones de Internet	Internet Assigned Numbers Authority
ICMPv6	Protocolo de Mensajes de Control de Internet Versión	Internet Control Message Protocol version 6
IEEE	Instituto de Ingeniería Eléctrica y Electrónica	Institute of Electrical and Electronics Engineers
IETF	Grupo de Trabajo de Ingeniería de Internet	The Internet Engineering Task Force
IPFIX	Protocolo de Flujo y Exportación Información de Internet	Internet Protocol Flow Information Export
IPv4	Protocolo de Internet versión 4	Internet Protocol version 4
IPv6	Protocolo de Internet versión 6	Internet Protocol version 6
JSON	Notación de objetos de JavaScript	JavaScript Object Notation
MAC	Control de acceso al medio	Media Access Control
MIB	Base de Información de Administración	Management Information Base
MPLS	Conmutación Multi-Protocolo mediante Etiquetas	Multiprotocol Label Switching

NETCONF	Protocolo de Configuración de Red	Network Configuration Protocol
NOS	Sistema Operativo de Red	Network Operating System
OJS	Sistemas de Publicaciones Periódicas	Open Journal Systems
ONF	Fundación de Redes Abiertas	Open Networking Foundation
OSI	Modelo de Interconexión de Sistemas Abiertos	Open System Interconnection
OSPF	El camino más corto primero	Open Shortest Path First
QoS	Calidad de Servicio	Quality of Service
RIR	Registro Regional de Internet	Regional Internet Registry
SDN	Redes Definidas por Software	Software Defined Networking
SLA	Acuerdo de Nivel de Servicio	Service Level Agreement
SNMP	Protocolo Simple de Administración de Red	Simple Network Management Protocol
TCP	Protocolo de Control de Transmisión	Transmission Control Protocol
UDP	Protocolo de Datagrama de Usuario	User Datagram Protocol
VLAN	Red de área local virtual	Virtual local area network
WSGI	Interfaz de Entrada de un Servidor Web	Web Server Gateway Interface
WSN	Redes de Sensores Inalámbricos	Wireless Sensor Networks
XML	Lenguaje de marcado extensible	Extensible markup language

RESUMEN

Las Redes Definidas por Software (SDN, Software Defined Network) son un nuevo paradigma que permite separar el plano de datos y el plano de control, esto significa, entre otras cosas, que las redes pueden ser programadas a través de aplicaciones software que se ejecutan en un elemento llamado controlador y éste de manera centralizada comunica las reglas de enrutamiento, seguridad y demás a los dispositivos de la red. Por otro lado en las redes convencionales se está migrando del protocolo IPv4 a IPv6 lo que muestra la importancia de probar e investigar el uso de nuevos protocolos, tecnologías y herramientas en las Redes Definidas por Software, en este caso específico, el funcionamiento de IPv6.

En este documento se muestra el desarrollo de un prototipo de redes IPv6 definidas por software y los procesos de emulación llevados a cabo a través de la herramienta Mininet y algunos controladores como RYU con el fin de fortalecer y dar aportes importantes alrededor de este tema. También se provee documentación detallada que permite llevar a cabo de manera sencilla el seguimiento del proceso con tutoriales y explicaciones de las instalaciones, emulaciones, integraciones y, en general, de todas las pruebas llevadas a cabo para el desarrollo de este prototipo en SDN.

Descriptores: Redes Definidas por Software, IPv6, Mininet, OpenFlow, RYU, NOX13OFLIB.

ABSTRACT

Software Defined Networks (SDN) are a new paradigm that separates the data plane and the control plane, this means, among other things, that networks can be programmed via software applications running in an element called controller and this centrally communicates routing and security rules to devices on the network. In addition to conventional networking is migrating from IPv4 to IPv6 which shows the importance of testing and research using new protocols, technologies and tools in software-defined networking, in this specific case, the operation of IPv6.

This document describes the development of a prototype IPv6 software defined networks and emulation processes conducted through Mininet tool and some controllers as RYU in order to strengthen and important contributions around the topic is displayed. Detailed documentation can be performed easily track the process with tutorials and explanations of the facilities, emulations, integrations

and, in general, all the tests carried out for the development of this prototype in SDN is also provided.

Keywords: Software Defined Networks, IPv6, Mininet, OpenFlow, RYU, NOX13OFLIB.

INTRODUCCIÓN

Actualmente la gran cantidad de datos que circula por las redes está causando problemáticas muy serias en el mundo de las telecomunicaciones, esto se debe, entre otras razones, por la falta de una administración y control adecuados, las bajas velocidades de transferencia, las congestiones y los inconvenientes en el envío de información. Uno de los obstáculos que genera estas dificultades obedece al hecho de que los equipos de enrutamiento no permiten ser configurados por el administrador de la red. Estos dispositivos contienen en su firmware las configuraciones de enrutamiento predefinidas por el fabricante, lo que imposibilita acciones como la distribución de cargas, los privilegios de transmisión, cambios de métricas en los protocolos de enrutamiento, etc.

Además de la imposibilidad en la personalización de este tipo de configuraciones, se encuentra el hecho de la descentralización de la administración de las redes, ya que cada router, por ejemplo, debe configurarse para determinar parámetros de diferente índole para el funcionamiento de una red. Para contribuir a la corrección de los inconvenientes mencionados anteriormente, ha surgido un paradigma llamado Redes Definidas por Software (SDN: Software Defined Networking), que permiten una configuración avanzada que mejora la gestión de las redes de datos. Con las Redes Definidas por Software la administración es completamente centralizada porque puede programarse, desde el controlador, la funcionalidad de toda una serie de enrutadores.

Este nuevo paradigma en redes genera entonces la posibilidad de investigar su aplicación en diferentes contextos y con diferentes protocolos de red existentes y en desarrollo. IPv6 está postulada a ser la versión del protocolo de Internet utilizada en el futuro, entonces es importante evaluar el funcionamiento en conjunto de las Redes Definidas por Software y la migración del protocolo IPv4 a IPv6 porque son las tecnologías que posiblemente marcarán el funcionamiento de las redes de datos.

La simulación y emulación de diferentes tecnologías y redes en el área de las telecomunicaciones es de vital importancia porque permite descubrir elementos y características a tener en cuenta en una implementación real.

Por esta razón en el presente proyecto se pretende dar aportes importantes para la integración del protocolo IPv6 y las Redes Definidas por Software. Para esto se exponen las características de IPv6 y se realiza una descripción detallada de los aspectos más relevantes en este nuevo paradigma como los protocolos, controladores y herramientas de simulación y emulación más utilizados.

Además se desarrolla un prototipo de una red definida por software que implementa el protocolo IPv6 y se describen los procedimientos de construcción y evaluación realizados mediante la herramienta Mininet, la utilización de varios controladores e instrumentos como Iperf, servidores HTTP y demás, para la realización de diferentes pruebas. Al final se anexan todos los documentos que fueron producto de los procesos realizados para la instalación de protocolos, controladores y software en general necesarios para el desarrollo del proyecto. Estos anexos son presentados y organizados de tal manera que puedan ser seguidos y entendidos por cualquier persona interesada en continuar o seguir los procesos realizados.

Este documento está organizado de la siguiente manera: en el capítulo uno se exponen los objetivos en los que se enmarca el trabajo, posteriormente en el capítulo dos se presenta el diseño metodológico y las estrategias de recolección de información tenidas en cuenta. En el tercer capítulo se hace una introducción al marco teórico que permitió adquirir los conceptos y definiciones necesarias para enfrentar la investigación. Este marco teórico despliega antecedentes, conceptos alrededor de las Redes Definidas por Software, el emulador Mininet y por último información sobre los protocolos IPv4 e IPv6.

En el cuarto capítulo está el desarrollo del proyecto, allí se exponen las capacidades de Mininet, la selección de las herramientas, las pruebas con diferentes controladores, emulaciones con IPv4 y por último varios prototipos emulados para probar IPv6.

Para finalizar el trabajo en los capítulos cinco y seis se presentan y analizan los resultados obtenidos y se plantean algunas conclusiones, reflexiones y recomendaciones finales producto de la actividad investigativa.

1. OBJETIVOS

1.1. OBJETIVO GENERAL

Diseñar y emular una Red Definida por Software mediante la herramienta Mininet con el fin de evaluar el funcionamiento del protocolo IPv6 en este tipo de redes.

1.2. OBJETIVOS ESPECÍFICOS

- Establecer mediante exploración bibliográfica cuáles son las capacidades de la herramienta Mininet de tal manera que permita conocer su funcionamiento y alcances.
- Diseñar una red definida por software habilitada para redes IPv6 que permita la realización de diferentes evaluaciones.
- Emular el diseño de red en la herramienta Mininet y evaluar su funcionamiento.
- Analizar los resultados obtenidos con el fin de dar aportes importantes en lo que se refiere a Redes IPv6 Definidas por Software.
- Realizar un artículo publicable con los resultados y enviarlo a una revista a evaluación.
- Realizar la documentación final.

2. METODOLOGÍA

El presente es un trabajo realizado bajo la modalidad de residencia en línea de investigación y hace parte de un proyecto para soportar Ingeniería de Tráfico en redes IPv6 perteneciente al grupo de investigación GIII de la Facultad de Ciencias Básicas e Ingeniería. En éste se utiliza una metodología acorde a una investigación exploratoria, ya que es producto de una indagación en un nuevo paradigma en redes que está en fase de experimentación y pruebas denominado Redes Definidas por Software (SDN).

Para ser más específico, el propósito del presente estudio es destacar los aspectos fundamentales de la implementación de IPv6 en SDN, y su enfoque es principalmente cualitativo.

2.1. DISEÑO METODOLÓGICO

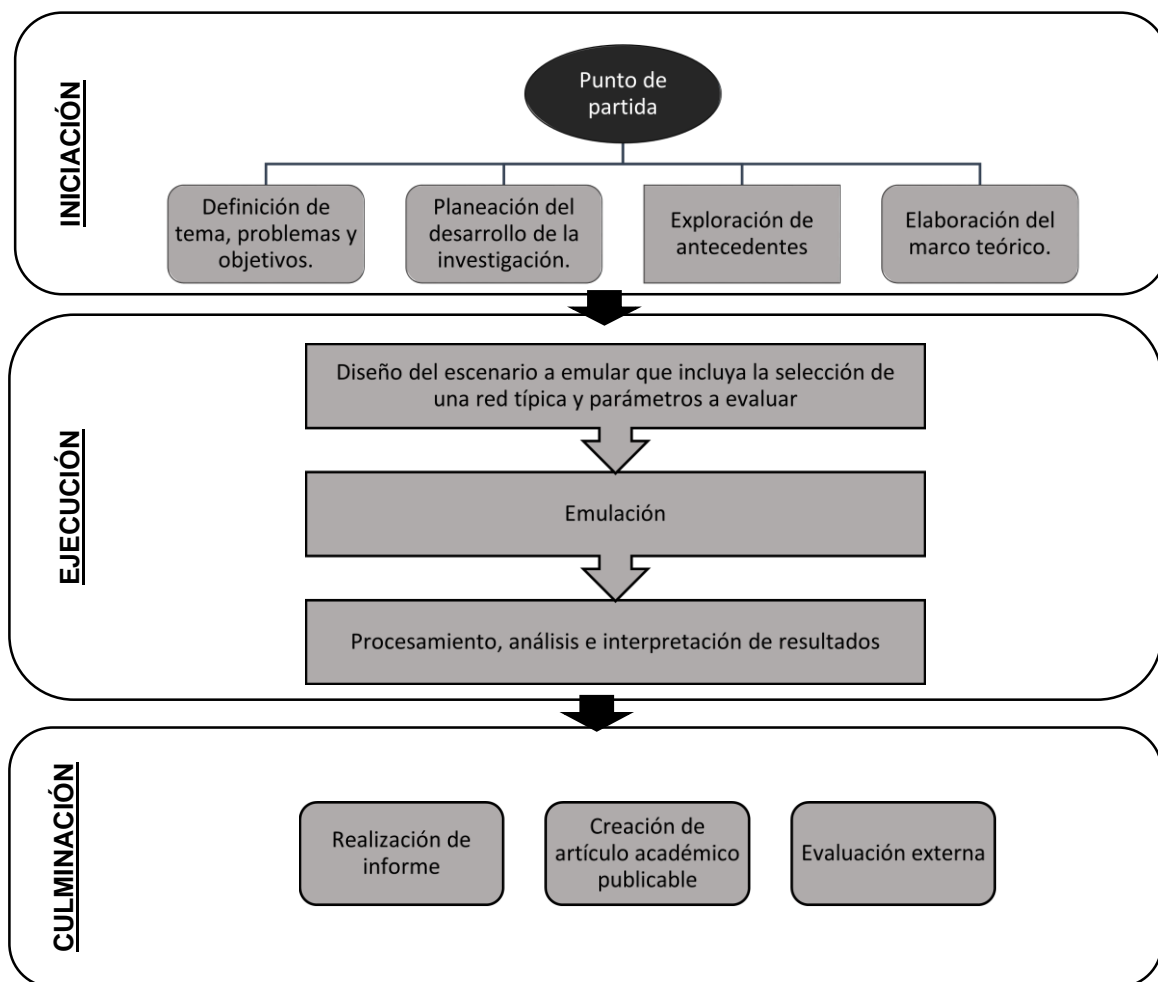


Figura 1. Diseño de la investigación

Como ilustra la Figura 1, el presente trabajo está compuesto por tres fases principales:

1. INICIACIÓN

En la etapa de iniciación se da comienzo al proyecto. Aquí se definen el tema, los problemas a tratar y los objetivos de la investigación, se realiza una planeación del resto de la investigación definiendo actividades, tiempos, recursos y en general haciendo un cronograma. Además se buscan y analizan algunos antecedentes del trabajo en exploraciones bibliográficas y revisiones documentales.

Para finalizar esta etapa, se crea un marco teórico que permita exponer los temas y definiciones necesarias para el entendimiento y desarrollo de la investigación.

2. EJECUCIÓN

Se elabora el diseño del escenario a emular, que incluye la selección de red con una topología básica que sea pertinente para realizar el prototipo y establecer protocolos y controladores que permitan realizar procesos de evaluación. Después, en la herramienta Mininet, se hace la emulación de la red diseñada y al final, con ayuda de lo obtenido con la emulación, se procesan, analizan e interpretan estos resultados.

3. CULMINACIÓN

Durante todo el desarrollo del proyecto, se construirá la documentación pertinente para la creación de un informe que exponga claramente el proceso y los resultados de la investigación, pero en la etapa de culminación debe corregirse y finalizarse.

Además, en esta etapa, se elabora un artículo académico con el objetivo de publicarse en una revista y que permita divulgar el proceso y/o los resultados del trabajo, y por último el trabajo pasa a una evaluación externa

2.2. RECOLECCIÓN DE DATOS

Para todas las etapas del proyecto, como fuentes de información se utilizarán las diferentes publicaciones del Instituto de Ingeniería Eléctrica y Electrónica (IEEE), los repositorios institucionales de diferentes universidades a nivel internacional, los sitios web oficiales de MiniNet y OpenFlow y diferentes documentos publicados por la Open Networking Foundation (ONF).

Como ayuda para el entendimiento de la herramienta Mininet y del protocolo OpenFlow, se ha participado en la lista de correo de discusión de Mininet (mininet-

discuss mailing list)¹, en la comunidad de ayuda de la ONF y se ha tenido contacto con personas capacitadas en el tema de otras universidades y organizaciones.

Mininet fue la herramienta seleccionada y utilizada para todos los diseños, emulaciones y pruebas requeridas. Posterior al proceso de emulación se realizan análisis y recomendaciones relacionadas con la implementación de IPv6 en las Redes Definidas por Software.

¹ Sistema implementado por la comunidad de ayuda de Mininet similar a un foro, con la diferencia que las preguntas y respuestas se realizan a través de correos electrónicos.

3. MARCO TEÓRICO

3.1. ANTECEDENTES

Las Redes Definidas por Software y el protocolo OpenFlow son un importante foco de investigación debido a sus características particulares y renovadoras para configurar y personalizar las redes, esto ha llevado a que se cataloguen como el paradigma y el protocolo del futuro (Blandón, 2013).

Pero la idea de programar redes no es nueva, algunos avances y desarrollos anteriores aportaron a lo que hoy son las SDN:

SOFTNET

Alrededor de los años 80s surgió un proyecto llamado SOFTNET, una red multisalto, semejante a las actuales WSN (Wireless Sensor Networks) cuya innovación fue que en el campo de datos de cada paquete se incluían comandos que los nodos iban ejecutando a medida que los iban recibiendo. Fue un intento de definir una red auto-organizable destinada a permitir la experimentación y la innovación con diferentes protocolos (Roncero, 2014).

Active Networks

No hubo desarrollo posterior a SOFTNET, pero su idea fue el embrión de las posteriores Redes Activas. Las Redes Activas (Active Networks) presentaban una arquitectura consistente en llevar embebido en los paquetes pequeños programas que podían ser ejecutados por los nodos que éstos atraviesan. Esto hacía posible que los switches y routers de la red procesaran los paquetes de datos, haciéndoles partícipes de los mensajes y no únicamente espectadores que se limitaban a enviar mensajes de un puerto a otro, de una forma “pasiva”. De ahí el nombre de Active Networks (Tennenhouse & Wetherall, 1996).

DCAN

Otra iniciativa que tuvo lugar a mediados de la década de 1990 es el “Devolved Control of ATM Networks” (DCAN). El objetivo de este proyecto era diseñar y desarrollar la infraestructura necesaria para el control y gestión escalable de redes de cajeros automáticos. La premisa es que las funciones de control y gestión de los muchos dispositivos (switches ATM en el caso de DCAN) deben ser desacopladas de los propios dispositivos y delegadas a entidades externas dedicadas a tal fin, que es básicamente el concepto detrás de SDN. DCAN asume un protocolo minimalista entre el gestor y de la red, en la línea de lo que sucede hoy en día en propuestas como OpenFlow (Nunes, Mendonca, Nguyen, Obraczka, & Turetti, 2014).

NETCONF

En 2006, el Grupo de Trabajo de Ingeniería de Internet IETF propuso NETCONF como un protocolo de gestión para modificar la configuración de dispositivos de red. El protocolo permitía a los dispositivos de red exponer una API a través del cual los datos de configuración extensibles podían ser enviados y recuperados.

Otro protocolo de gestión, ampliamente desplegado en el pasado y que se sigue utilizando hoy en día, es el SNMP. SNMP se propuso a finales de los 80 y resultó ser un protocolo de gestión de red muy popular, que utiliza la interfaz de gestión estructurado (SMI) para obtener los datos contenidos en la Base de Información de Administración (MIB). Podía ser utilizado también para cambiar las variables del MIB con el fin de modificar los parámetros de configuración. Más tarde se hizo evidente que, a pesar de lo que fue pensado originalmente, SNMP no estaba siendo utilizado para configurar equipos de red, sino más bien como una herramienta de rendimiento y supervisión de fallos. Por otra parte, se detectaron múltiples deficiencias en la concepción de SNMP, el más notable de los cuales fue la falta de una fuerte seguridad. Esto se abordó en una versión posterior del protocolo.

NETCONF, en el momento en que fue propuesto por el IETF, fue visto por muchos como un nuevo enfoque para la gestión de red que solucionaba las deficiencias mencionadas en SNMP. Aunque el protocolo NETCONF cumple con el objetivo de simplificar la reconfiguración del dispositivo y actúa como un bloque de construcción para la gestión, no permite la separación entre los planos de datos y de control. Lo mismo puede afirmarse acerca de SNMP. Una red con NETCONF no debe ser considerada como totalmente programable; además, está diseñado principalmente para ayudar a la configuración automatizada y no para activar el control directo del Estado ni permite un rápido despliegue de servicios y aplicaciones innovadoras. Sin embargo, tanto NETCONF y SNMP son herramientas útiles de administración que se pueden utilizar en paralelo en conmutadores híbridos soportan otras soluciones que permiten la creación de redes programables (Nunes, Mendonca, Nguyen, Obraczka, & Turletti, 2014).

Ethane:

El antecedente inmediato de OpenFlow fue el proyecto SANE / Ethane, que, en 2006, definió una nueva arquitectura para redes empresariales. El enfoque de Ethane fue sobre el uso de un controlador centralizado para gestionar la política y la seguridad en una red. Un ejemplo notable es proporcionar control de acceso basado en la identidad. Similar a la SDN, Ethane emplea dos componentes: un controlador para decidir si un paquete debe ser enviado, y un switch Ethane que consiste en una tabla de flujo y un canal seguro al controlador. Ethane sentó las bases para lo que se convertiría en Redes Definidas por Software (Nunes, Mendonca, Nguyen, Obraczka, & Turletti, 2014).

Teniendo en cuenta que la primera versión de OpenFlow fue lanzada en el 2011, no existen gran cantidad de estudios que aborden el tema, y menos aún el tema específico de las particularidades de la emulación de SDN con IPv6. A pesar de esto, se hallan varias investigaciones y trabajos alrededor del mundo relacionados con la utilización del emulador MiniNet, el protocolo OpenFlow y los primeros acercamientos de la implementación de IPv6 en Redes Definidas por Software. Estas investigaciones se llevan a cabo no sólo dentro de instituciones académicas sino también en organizaciones y empresas fabricantes de dispositivos mundialmente reconocidas que han visto en este nuevo modelo el futuro de las redes de datos.

OpenFlow empezó a desarrollarse en 2007 con la colaboración de los sectores académico y empresarial. Fueron las universidades de Stanford y California en Berkeley quienes llevaron las riendas en primera instancia y que junto con un grupo de empresas crearon la ONF (Open Networking Foundation).

HP (Hewlett-Packard) ha sido una de las empresas pioneras en apoyar la evolución del protocolo OpenFlow desde sus inicios, además es miembro fundador de la ONF y el primer proveedor de switches que soportan esta tecnología. Por su liderazgo en el desarrollo de este nuevo protocolo la tecnología de HP OpenFlow ha sido la opción preferida de investigadores de todo el mundo desde 2008 (Hewlett-Packard, 2014).

Así como HP, otras empresas que conforman la ONF, como Alcatel, BTI, Cisco, DCN, Google, Facebook, Hitachi, Huawei, IBM, Intel, Microsoft, Nokia, Oracle, Samsung, Yahoo y ZTE realizan investigaciones y se apropian paulatinamente de esta nueva arquitectura tanto en la fabricación de dispositivos como en su implementación.

Por su parte, algunas Universidades han hecho su tarea de explorar y experimentar en este nuevo protocolo y arquitectura de red. Desde el ámbito académico, importantes avances han permitido la evolución de las Redes Definidas por Software y contribuyendo a la mejora continua de OpenFlow a través de emuladores como MiniNet que permiten descubrir comportamientos de la red en casos específicos.

En un estudio titulado “OpenFlow Deployment and Concept Analysis” realizado en la Universidad de Žilina de la República Eslovaca, también incursionaron en el análisis de los aportes que hacen las Redes Definidas por Software en el mejoramiento de las redes de datos actuales, donde exponen, principalmente, todos los inconvenientes y/o obstáculos que pueden presentarse para su implementación, por ejemplo la falta de compatibilidad con IPv6 en dispositivos de red reales. Este análisis incluye un proceso de emulación en MiniNet. El artículo está publicado en la revista denominada “Advances in Electrical and Electronic

Engineering” en el sistema OJS (Open Journal System) de la universidad mencionada (Hegr, Bohac, Uhler, & Chlumsky, 2013).

Dentro del Laboratorio de Redes Avanzadas del Instituto Tecnológico Autónomo de México (ITAM), se han realizado una serie de aproximaciones y avances en este tema. Una que destaca es la titulada “Despliegue y Evaluación de Desempeño de una Red OpenFlow”. Rebeca Mayumi Park Campos y Elena Eunise Baack Valle, las investigadoras encargadas del trabajo, llegaron al punto de diseñar e implementar una Red Definida por Software real, con equipos físicos conectados y funcionando. Antes de realizar la instalación utilizaron el emulador MiniNet para probar su diseño (Park & Baack, 2012).

En la Universidad de Brasilia (UnB) de Brasil, se desarrolló en el año 2013 una monografía basada en los protocolos y los paradigmas de las redes definidas por software. Para su propuesta el autor utiliza el mismo emulador que se plantea para la elaboración del presente proyecto, además se exponen los conceptos básicos que se deben tener en cuenta a la hora de trabajar con Redes Definidas por Software (Rodrigues, 2013).

Son pocas las investigaciones que tratan el asunto concreto de la emulación de Redes Definidas por Software con el protocolo IPv6, pero en la Escuela de Electrónica y Ciencias de la Computación de la Universidad de Southampton, Inglaterra, se realizó un estudio denominado “Technology Validation Experiment: IPv6 and Multicast Support on OpenFlow” donde describen cómo OpenFlow soporta actualmente IPv6. El proyecto consta de 3 fases: la primera es una configuración de un entorno virtualizado usando MiniNet, la segunda es una serie de pruebas creadas en OFELIA y la tercera es una implementación en dispositivos de red físicos reales (Newman, 2014).

3.2. REDES DEFINIDAS POR SOFTWARE (SDN)

Las Redes Definidas por Software (SDN) son un paradigma de redes emergentes que da la esperanza de cambiar las limitaciones de las infraestructuras de red actuales. En primer lugar, se rompe la integración vertical mediante la separación de la lógica de control de la red (el plano de control) de los routers y switches subyacentes que reenvían el tráfico (el plano de datos). En segundo lugar, con la separación de los planos de control y de datos, los conmutadores de red se convierten en dispositivos de reenvío simples y la lógica de control se implementa en un controlador de lógica centralizado (o un sistema operativo de red), simplificando la aplicación de políticas y la reconfiguración (Kreutz et al., 2015).

Esencialmente las redes definidas por software buscan separar el plano de control del plano de datos, favoreciendo así la creación de redes más programables, automatizables y flexibles dependiendo de su prioridad gracias a la toma de decisiones por parte del servidor (Open Networking Foundation, 2015).

SDN se centra en cuatro características claves (Sezer et al., 2013):

- Separación del plano de control del plano de datos.
- Un controlador centralizado y la vista de la red.
- Interfaces abiertas entre los dispositivos en el plano de control. (controladores) y los que están en el plano de datos.
- Programación de la red por aplicaciones externas.

3.2.1. Arquitectura

Como se muestra en la Figura 2 la arquitectura de SDN consta conceptualmente de tres capas: capa de infraestructura, capa lógica y capa de aplicación.

La capa de infraestructura está compuesta por **los dispositivos de red (switches y routers)** que, en este nuevo paradigma, pueden ser configurados por los controladores a través de la implementación de reglas más sofisticadas de supervisión de tráfico para el cambio y/o políticas de seguridad. Estos elementos representan el plano de datos.

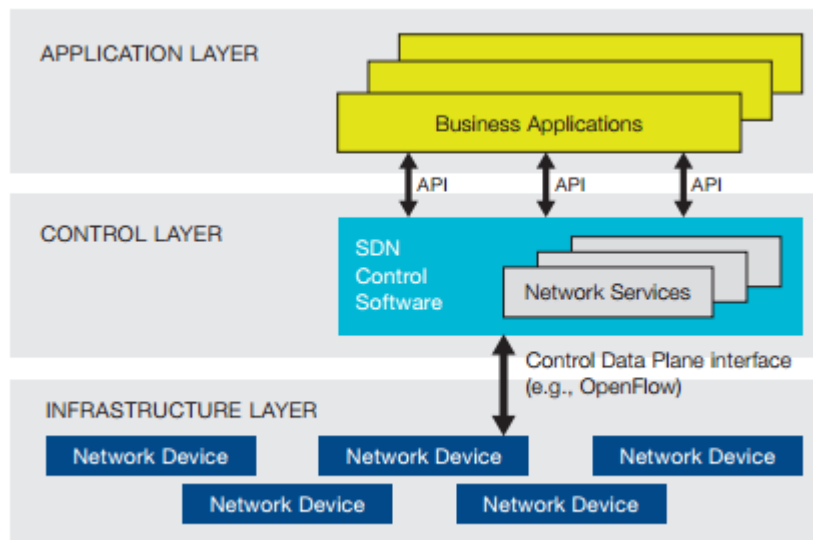


Figura 2. Arquitectura de una SDN.

Fuente: Open Networking Foundation. (2012). Software-Defined Networking: The New Norm for Networks. Recuperado el 08 de mayo de 2015, de <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>

La capa intermedia la forma el **controlador**, quien tiene una visión global de la red e incorpora el **sistema operativo de red (NOS)**. Es el encargado de tomar las decisiones y programar las tablas de flujo de los elementos de la capa inferior para controlar, entre otras cosas, el desvío y el enrutamiento de paquetes.

La capa superior es la capa de aplicación. Esta capa la componen las **aplicaciones de usuarios**, que incorporan las llamadas NorthBound APIs. Y es aquí donde cambia totalmente la filosofía respecto a las redes tradicionales. Estas Northbound APIs incorporan los patrones de uso de la red de cada aplicación, y su función es comunicar esos patrones a la capa de control, donde se toman las decisiones oportunas. Y aquí se cierra el círculo: las aplicaciones definen el uso que se va a dar a la red, lo comunican al controlador SDN, el cual toma las decisiones oportunas y las comunica a la infraestructura de red (capa inferior) mediante las SouthBound APIs (Roncero, 2014).

Estas SouthBound consisten en un **protocolo o interfaz** de programación bien definida entre los elementos de la capa de infraestructura y el controlador SDN. La interfaz mayormente utilizada y más notable es OpenFlow. Por ejemplo, un switch OpenFlow tiene una o más tablas con reglas de manejo de paquetes (tablas de flujo). Cada regla coincide con un subconjunto del tráfico y realiza ciertas acciones (descarte, reenvío, modificación, etc.) en el tráfico. Dependiendo de las reglas instaladas por una aplicación de controlador, un switch OpenFlow puede -

instruido por el controlador - comportarse como un router, switch, firewall, o llevar a cabo otras funciones (Kreutz et al., 2015).

Por otra parte la Open Networking Foundation (2012) presenta como ventajas de SDN las siguientes:

- Gestión y control centralizado de dispositivos de red desde múltiples proveedores.
- Automatización y gestión mejorada mediante el uso de APIs comunes para abstraer los detalles de redes subyacentes y los sistemas y aplicaciones de aprovisionamiento.
- Rápida innovación ya que ofrecen nuevas capacidades y servicios de red sin la necesidad de configurar dispositivos individuales o esperar las actualizaciones de los fabricantes.
- Programación de los operadores, empresas, proveedores de software independientes y usuarios (no sólo los fabricantes de equipos) utilizando entornos de programación común, que da todas las oportunidades para impulsar la diferenciación;
- Mayor fiabilidad y seguridad de la red como consecuencia de una gestión centralizada y automatizada de dispositivos de red, aplicación de una política uniforme, y un menor número de errores de configuración;
- Control de red más granular con la capacidad de aplicar políticas integrales y de gran alcance a nivel de sesión, usuarios, dispositivos y aplicaciones; y
- Mejor experiencia del usuario final como aplicaciones de red centralizada que adaptan a la perfección el comportamiento de la red a las necesidades del usuario.

3.3. OPENFLOW

OpenFlow es un protocolo abierto que se utiliza para proporcionar una comunicación entre el controlador y los dispositivos de conmutación. El controlador y los dispositivos de conmutación se conocen como plano de control y el plano de datos, respectivamente. El controlador OpenFlow es responsable de decidir qué acción se va a realizar por el switch. El enfoque de la decisión puede ser reactivo o proactivo.

Un ejemplo de un conjunto de instrucciones posibles que puede enviar un controlador a un switch a través de OpenFlow es ejemplificado en la Figura 3.

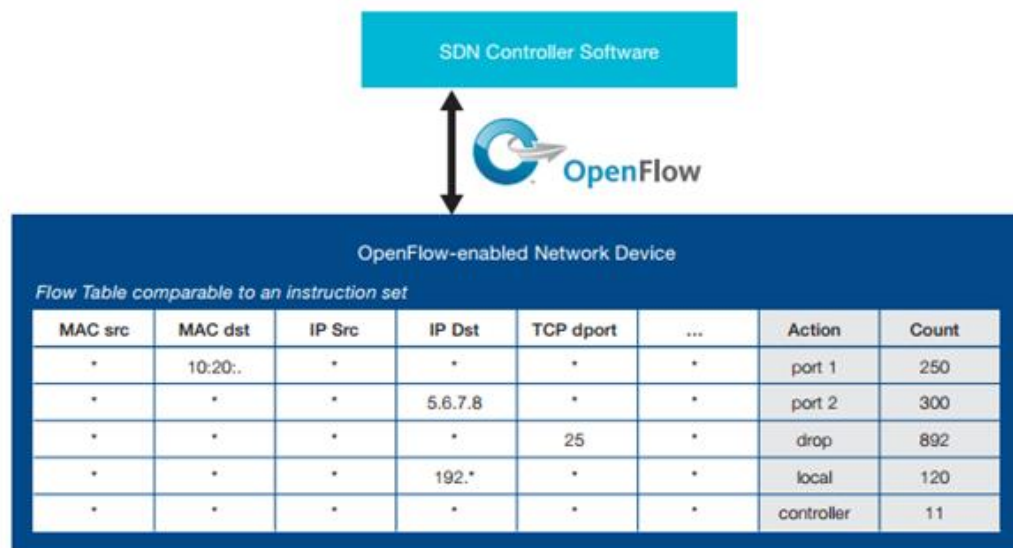


Figura 3. Ejemplo de un conjunto de instrucciones OpenFlow.

Fuente: Open Networking Foundation. (2012). Software-Defined Networking: The New Norm for Networks. Recuperado el 08 de mayo de 2015, de <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>

En el enfoque reactivo, cuando un paquete llega a un switch, el switch no sabe cómo manejar ese paquete. Por lo tanto el switch envía el paquete al controlador. El controlador es el responsable de insertar una entrada en la tabla de flujo del switch mediante el protocolo OpenFlow. La principal desventaja de este enfoque es que el switch es totalmente dependiente de la decisión del controlador. Así que cuando el switch pierde la conexión con el controlador, este no podrá manejar ese paquete.

Pero si por el contrario el enfoque es proactivo, el controlador rellena las entradas en las tablas de flujo de cada switch. Este enfoque supera la limitación del enfoque reactivo porque incluso si el switch pierde la conexión con el controlador, este no interrumpe el tráfico (Kaur, Singh, & Ghuman, 2014).

Como se muestra en la Figura 4, la comunicación entre el controlador y los dispositivos de conmutación se realiza a través de OpenFlow, en la cual el dispositivo de reenvío, o el switch de OpenFlow, contiene una o más tablas de flujo y una capa de abstracción que se comunica de forma segura con un controlador a través del protocolo OpenFlow. Las tablas de flujo consisten en entradas de flujo, cada uno determina cómo se procesarán y se enviarán los paquetes que pertenecen al flujo.

Las entradas de flujo suelen consistir en: (1) los campos coincidentes, o reglas de concordancia, que se utilizan para igualarse con los paquetes entrantes; los campos coincidentes pueden contener la información que se encuentra en la

cabecera del paquete, puerto de entrada, y metadatos; (2) contadores, utilizados para recoger las estadísticas de un flujo particular, como por ejemplo: el número de paquetes recibidos, el número de bytes y la duración del flujo; y (3) un conjunto de instrucciones o acciones, que se aplicarán a una coincidencia; dictan cómo manejar los paquetes que concuerden.

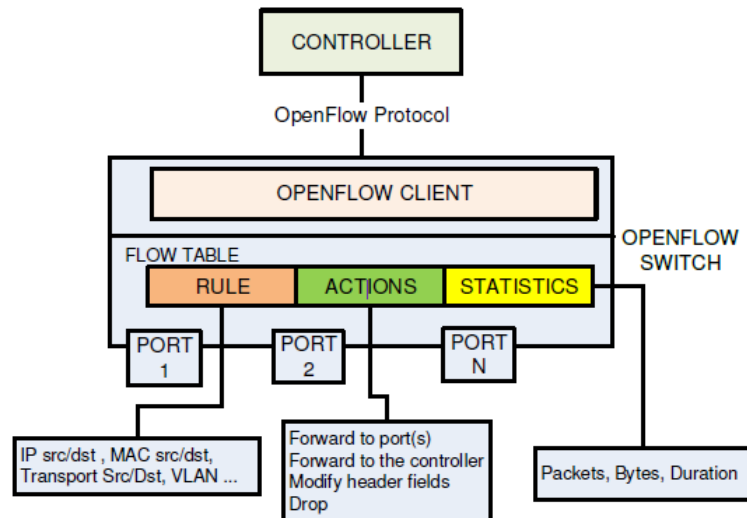


Figura 4. Comunicación entre el controlador y el switch a través de OpenFlow.

Fuente: Nunes, B. A., Mendonca, M., Nguyen, X.-N., Obraczka, K., & Turletti, T. (2014). A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. Communications Surveys and Tutorials, IEEE Communications., 2(4), 1617-1634.

Cuando llega un paquete a un switch de OpenFlow, los campos de la cabecera del paquete se extraen y se comparan con la parte de los campos coincidentes de las entradas de la tabla de flujo, es decir, si las cabeceras del paquete coinciden, el switch aplica un conjunto de instrucciones asociadas con la entrada del flujo, pero si por el contrario la cabecera del paquete no coincide, éste no podrá ser procesado.

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

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

3.4. MININET

Con el fin de crear una plataforma para probar y experimentar las características de las Redes Definidas por Software, desarrollar aplicaciones y fomentar su uso, un grupo de profesores de la Universidad de Stanford crearon el entorno MiniNet.

MiniNet (2015) es un emulador de red que ejecuta una colección de host, switches, routers y enlaces en un solo núcleo de Linux. Utiliza la virtualización ligera para hacer un único sistema que parece una red completa, ejecutándose todo en el mismo núcleo, sistema y código de usuario (ver Figura 5). Desde un host se pueden enviar paquetes a través de lo que parece ser una verdadera interfaz Ethernet, con una velocidad y retardo de enlace específico. Los paquetes se procesan por lo que parece un verdadero switch Ethernet o un enrutador.

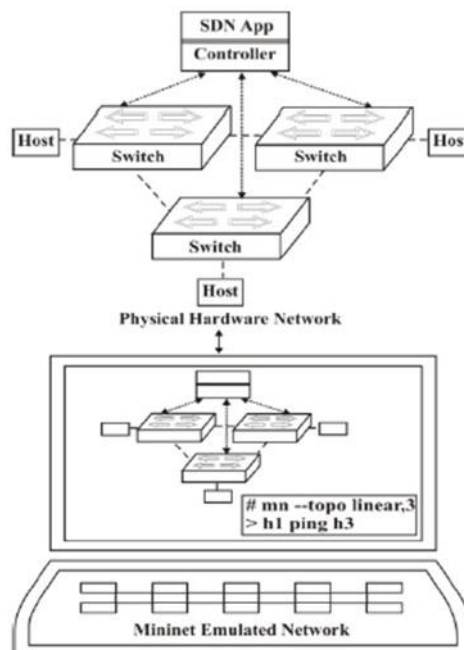


Figura 5. Emulación de red en MiniNet

Fuente: Kaur, K., Singh, J., & Ghumnan, N. (2014). MiniNet as Software Defined Networking Testing Platform. Recuperado el 08 de mayo de 2015, de <http://www.sbsstc.ac.in/icccs2014/Papers/Paper29.pdf>

Los host de MiniNet se ejecutan en el software de Linux y los switches soportan el protocolo OpenFlow, que es actualmente el protocolo mayormente usado para la investigación de SDN. MiniNet es un emulador que puede ejecutar diferentes topologías predefinidas como por ejemplo: single, lineal, árbol, pero también puede ejecutar topologías personalizadas, para ello se debe contar con algún

conocimiento en Python. Por lo tanto se puede deducir que MiniNet se crea usando código y no usando hardware (Pal, Veena, Rustagi, & Murthy, 2014).

Características y ventajas de MiniNet (MiniNet, 2015):

1. **Es rápido:** poner en marcha una red simple toma sólo unos segundos. Esto significa que el bucle de gestión de edición de depuración puede ser muy rápido.
2. **Creación topologías personalizadas:** un solo switch, grandes topologías en Internet, un centro de datos, o cualquier otra cosa.
3. **Ejecución de programas reales:** cualquier cosa que se ejecute en Linux está disponible, desde servidores web, ventanas de herramientas de monitoreo TCP hasta Wireshark.
4. **Personalización del reenvío de paquetes:** los switches de MiniNet son programables mediante el protocolo OpenFlow. Los diseños personalizados de Redes Definidas por Software que se ejecutan en MiniNet se pueden transferir fácilmente al hardware de switches OpenFlow reales para el reenvío de paquetes.
5. **Es posible ejecutar MiniNet en su computadora portátil,** en un servidor, en una máquina virtual, en una máquina Linux nativo (MiniNet se incluye con Ubuntu 12.10+!), o en la nube (por ejemplo, Amazon EC2.)
6. **Es posible compartir y replicar los resultados:** cualquier persona con una computadora puede ejecutar el código hecho por otra persona una vez ésta lo haya compartido.
7. **Es de fácil utilización:** se puede crear y ejecutar experimentos MiniNet escribiendo simples (o complejas si es necesario) scripts de Python.
8. MiniNet es un **proyecto de código abierto**, por lo que existe la posibilidad de examinar su código fuente en <https://github.com/mininet>, modificarlo, corregir los errores, problemas, hacer peticiones, etc.
9. **MiniNet está en constante desarrollo.** Así que si no funciona por alguna razón, puede informarse a MiniNet-Discuss y la comunidad de usuarios y desarrolladores MiniNet puede tratar de explicarlo, arreglarlo, o ayudar a solucionarlo.

3.5. IPv4 e IPv6

Cada computadora conectada a Internet necesita una dirección IP única, que la identifique de manera lógica y jerárquica. Las direcciones actualmente en uso están referidas al Internet Protocol version 4 (IPv4). Este protocolo pertenece a la capa de red del modelo OSI y hoy en día es el utilizado como base para las comunicaciones de Internet. En él, las direcciones son representadas por un número binario de 32 bits permitiendo una cantidad total de 2^{32} , aproximadamente 4,3 billones, direcciones posibles. También se pueden expresar como números de notación decimal dividiendo los 32 bits de la dirección en cuatro octetos. El valor

decimal de cada octeto está comprendido en el rango de 0 a 255, y están separados con un carácter único ".". Por ejemplo: 192.168.153.11 (Casero, Clemente, & Ruiz, 2011).

Desde que la Internet viene siendo utilizada en gran medida y el número de equipos conectados a la red incrementa, la forma actual del Protocolo de Internet (IPv4), se ha quedado corta, pues no cuenta con el número suficiente de direcciones IP para identificar de forma única a cada dispositivo conectado a la red. Por ejemplo, en febrero de 2011 el IANA (Internet Assignen Numbers Authority) entregó dos de los últimos siete bloques de direcciones /8 que tenía disponibles a la región Asia-Pacífico, lo que significa que este RIR (Regional Internet Registry) ya agotó su pool de IPv4.

Al notar estos problemas, la IETF (Grupo de Trabajo de Ingeniería de Internet), comenzó a trabajar en 1990 en una versión nueva del IP, una que no se quedara corta de direcciones, para esto realizó una convocatoria visible en la RFC 1550 para seleccionar la "IP de próxima generación" (Bradner, 1993). Tras análisis y revisiones se creó una combinación entre las propuestas de Deering y Francis, a la que se le dio el nombre de IPv6 (Deering & Hinden, 1998).

IPv6 tiene direcciones más grandes que el IPv4, son de 128 bits en oposición a los 32 bits respectivamente, lo que permite que existan muchas más direcciones asignables. Además de resolver el problema de la cantidad de direcciones, IPv6 simplifica el encabezado, que solo contiene 7 campos, a diferencia de IPv4 que contiene 13, esto permite que los enrutadores procesen mucho más rápido los paquetes y por tanto mejorar la velocidad de transporte (Tanenbaum, 2003).

Los cambios de IPv4 a IPv6 pueden encuadrarse en las siguientes categorías:

- Aumento de la capacidad de direccionamiento
- Simplificación del formato de encabezado
- Mejoramiento del soporte para extensiones y opciones
- Capacidad para etiquetar flujos de tráfico
- Capacidades de autenticación y privacidad

3.5.1. Encabezado IPv4 e IPv6

El encabezado básico de paquetes IPv4 tiene 12 campos con un tamaño total de 20 octetos (160 bits) (ver la Figura 6). Los 12 campos pueden ser seguidos por un campo Opciones, que, a su vez, es seguido por una porción de datos que es por lo general el paquete de la capa de transporte. La longitud variable del campo Opciones se añade al tamaño total de la cabecera del paquete IPv4.

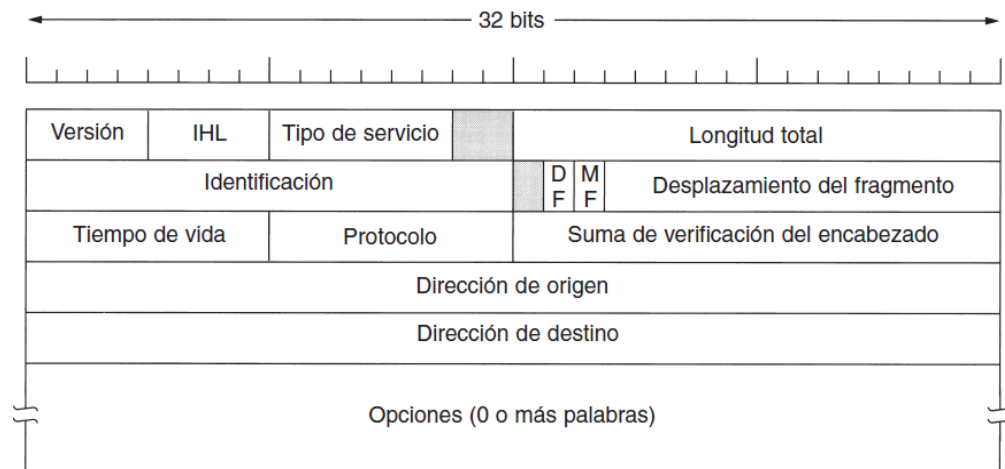


Figura 6. Encabezado IPv4

Fuente: Tanenbaum, A. (2003). *Redes de computadoras* (Cuarta ed.). México: Pearson Educación.

El encabezado básico de un paquete IPv6 tiene 8 campos con un tamaño total de 40 octetos (320 bits) (ver la Figura 7). Algunos campos de la cabecera IPv4 fueron retirados porque, en IPv6, la fragmentación no es manejada por los dispositivos y las sumas de comprobación en la capa de red no se utilizan. En su lugar, la fragmentación en IPv6 es manejada por el origen de un paquete y las sumas de comprobación son hechas en la capa de enlace de datos y capa de transporte. (En IPv4, la capa de transporte UDP utiliza una suma de control opcional. En IPv6, se requiere el uso de la suma de comprobación UDP para comprobar la integridad del paquete interno).

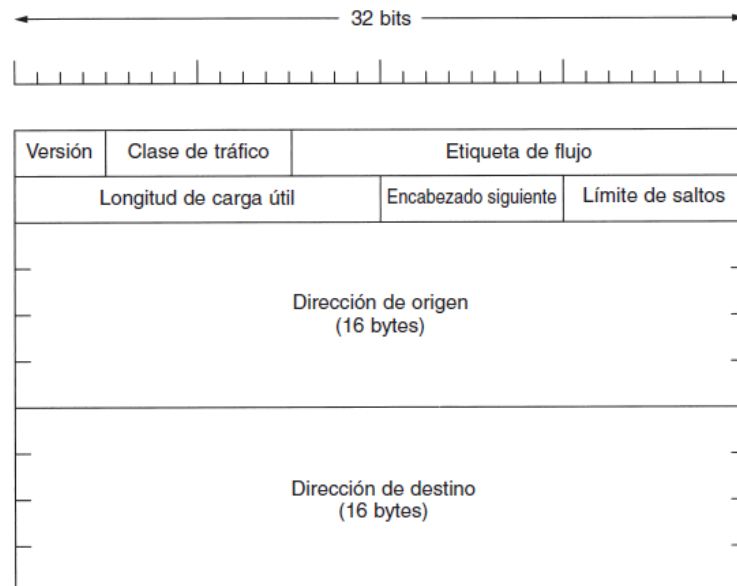


Figura 7. Encabezado IPv6

Fuente: Tanenbaum, A. (2003). *Redes de computadoras* (Cuarta ed.). México: Pearson Educación.

Los campos IHL, Identificación, Banderas: DF - MF, Desplazamiento del fragmento y suma de verificación de la cabecera del paquete IPv4 no están incluidos en la cabecera del paquete IPv6.

El campo “IHL” de IPv4 es irrelevante para IPv6 porque todas las cabeceras IPv6 son de la misma longitud. IPv4 requiere este campo porque sus cabeceras pueden ser tan cortas como de 20 bytes y tan largas como de 60 bytes para acomodar las Opciones IP.

Tabla 1. Descripción de los campos de la cabecera básica de IPv6

Campo	Descripción
Versión	Similar al campo Versión en la cabecera de paquete IPv4, excepto que el campo almacena el número 6 para IPv6 en vez del número 4 para IPv4.
Clase de tráfico	Similar al campo Tipo de Servicio en la cabecera del paquete IPv4. El campo Clase de Tráfico etiqueta los paquetes con una clase de tráfico que se utiliza en servicios diferenciados.
Etiqueta de flujo	Un nuevo campo en la cabecera del paquete IPv6. El campo Etiqueta de Flujo etiqueta paquetes con un flujo específico que diferencia a los paquetes en la capa de red.
Longitud de carga útil	El campo “Longitud Total” de IPv4, es el campo “Longitud de Carga útil” en IPv6. El campo “Longitud total” de IPv4 especifica la longitud del datagrama entero, incluyendo las cabeceras IP, de esta manera los routers pueden calcular la longitud de carga útil del datagrama IPv4 mediante la sustracción de la longitud de la cabecera de la longitud del datagrama. En IPv6 este cálculo es innecesario, porque la longitud de carga útil de IPv6 incluye las cabeceras de extensión.
Encabezado siguiente	Similar al campo Protocolo en la cabecera del paquete IPv4. El valor del campo Cabecera Siguiente determina el tipo de información después de la cabecera básica IPv6. El tipo de información después de la cabecera básica IPv6 puede ser un paquete de capa de transporte, por ejemplo, un paquete TCP o UDP, o una cabecera de extensión.
Límite de saltos	Igual que el campo Tiempo de Vida en la cabecera del paquete IPv4. El valor del campo Límite de Salto especifica el número máximo de dispositivos a través de los cuales un paquete IPv6

	puede pasar antes de que el paquete se considere no válido. Cada dispositivo disminuye el valor por uno. Debido a que ninguna suma de control se encuentra en la cabecera IPv6, el dispositivo puede reducir el valor sin necesidad de volver a calcular la suma de control, lo que ahorra recursos de procesamiento.
Dirección origen	Igual que el campo de dirección de origen en la cabecera del paquete IPv4, excepto que el campo contiene una dirección de origen de 128 bits para IPv6 en lugar de una dirección de origen de 32 bits para IPv4.
Dirección destino	Similar al campo de dirección de destino en la cabecera del paquete IPv4, excepto que el campo contiene una dirección de destino de 128 bits para IPv6 en lugar de una dirección de destino de 32 bits para IPv4.

Fuente: Cisco Systems, Inc. (2012). IPv6 Configuration Guide, Cisco IOS. Release 15.2MT. Recuperado el 2015 de mayo de 14, de <http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6/configuration/15-2mt/ipv6-15-2mt-book.pdf>

3.5.2. Tipos de direcciones IPv6

Como se expone en la RFC 4291 (2006), en IPv6 se han definido 3 tipos de direcciones:

- **Unicast:** un identificador para una interface. Un paquete enviado a una dirección unicast es entregado a la interface identificada por esa dirección.
- **Anycast:** es un identificador a un conjunto de interfaces (típicamente pertenecientes a diferentes nodos). Un paquete envía a una dirección anycast es entregado a uno de las interfaces identificadas por esa dirección (el más cercano, de acuerdo a la métrica de la tabla de enrutamiento).
- **Multicast:** es un identificador para un conjunto de interfaces típicamente pertenecientes a diferentes nodos). Un paquete enviado a una dirección multicast es entregado a todas las interfaces identificadas por esa dirección.

Con IPv6, la función **broadcast** es llevada a cabo mediante envío de paquetes a todos los nodos de direcciones **multicast**.

3.5.3. Representación de una dirección IPv6

La forma estándar es x : x : x : x : x : x : x : x, donde las 'x' son valores hexadecimales de 16 bits.

Ejemplo:

FEDC:BA98:7654:3210:FEDC:BA98:7654:3210

Debido a las características específicas de IPv6, es común encontrar direcciones que contengan cadenas grandes con bits '0'. Con el fin de hacer más fácil la escritura de direcciones que contengan bits cero, existe una sintaxis especial para comprimir los ceros.

El uso de ":" indica uno más grupos de 16 bits de ceros. Los "::" pueden aparecer solamente una vez en una dirección y pueden utilizarse para comprimir ceros iniciales o finales en una dirección. Además los ceros a la izquierda de cada hexeto² pueden suprimirse.

La dirección loopback que aparece en la

Tabla 2, puede ser utilizada por un nodo para enviar un paquete IPv6 a sí mismo, ésta funciona igual que la dirección de loopback en IPv4 (127.0.0.1).

Tabla 2. Formatos comprimidos de direcciones IPv6.

Tipo de dirección IPv6	Formato preferido	Formato comprimido
Unicast	2001:0:0:0:0DB8:0800:200C:417A	2001::DB8:800:200C:417A
Multicast	FF01:0:0:0:0:0:0:101	FF01::101
Loopback	0:0:0:0:0:0:0:1	::1
Unspecified	0:0:0:0:0:0:0:0	::

Fuente: Cisco Systems, Inc. (2012). IPv6 Configuration Guide, Cisco IOS. Release 15.2MT. Recuperado el 2015 de mayo de 14, de <http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6/configuration/15-2mt/ipv6-15-2mt-book.pdf>

3.5.4. Estructura de direccionamiento IPv6

Como ya se mencionó, las direcciones se clasifican en diferentes tipos: unicast, multicast y anycast. Cada uno de los tipos define valores específicos para subgrupos de los 128 bits, asociando dicho valor con las características especiales del tipo.

Direcciones Unicast

² Forma como se le conoce a cada grupo de 4 dígitos de una dirección IPv6.

Existen 3 tipos de direcciones Unicast (Cisco Systems, Inc., 2012):

1. Direcciones Unicast Global

Las direcciones IPv6 globales únicas se definen por un **Prefijo global de enrutamiento**, un **ID de subred**, y un **ID de interfaz**.

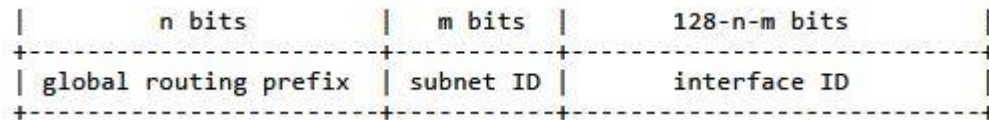


Figura 8. Formato de direcciones unicast global

Fuente: Network Working Group. (2006). *RFC 4291: IP Version 6 Addressing Architecture*. Recuperado el 12 de mayo de 2015, de <https://tools.ietf.org/html/rfc4291>

A excepción de las direcciones que comienzan con el binario 000, todas las direcciones unicast globales tienen un ID de interfaz de 64 bits. La asignación de la dirección unidifusión global IPv6 utiliza el rango de direcciones que comienzan con valor binario 001 (2000 :: / 3). La Figura 8 muestra la estructura de una dirección unicast global.

La dirección global unicast típicamente consiste en un prefijo de enrutamiento global, un ID de subred y un ID de interfaz.

Un campo de subred de 16 bits llamado el ID de subred podría ser utilizado por las organizaciones individuales para crear su propia jerarquía de direccionamiento local y para identificar las subredes. Un ID de subred es similar a una subred en IPv4, excepto que una organización con una subred ID IPv6 puede soportar hasta 65.535 subredes individuales.

2. Direcciones Locales Únicas

Una dirección única local es una dirección unicast IPv6 que es única a nivel mundial y está destinado para las comunicaciones locales, o sea que provee direcciones IP enrutables y asignables dentro de un sitio. No se espera que sean usadas para ser enrutadas en la Internet global, es enrutable dentro de un área limitada.

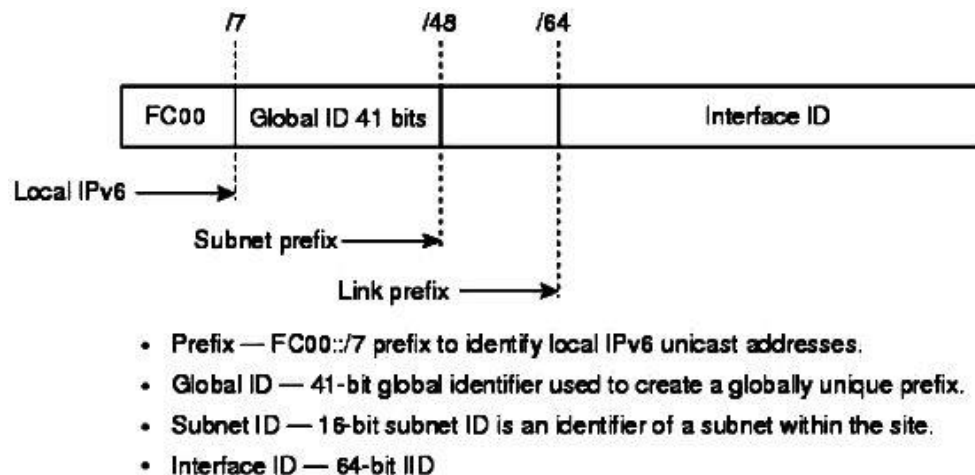


Figura 9. Estructura de Dirección Local Única

Fuente: Cisco Systems, Inc. (2012). IPv6 Configuration Guide, Cisco IOS. Release 15.2MT. Recuperado el 2015 de mayo de 14, de <http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6/configuration/15-2mt/ipv6-15-2mt-book.pdf>

3. Direcciones de enlace local

Una dirección de enlace local es una dirección unicast IPv6 que se puede configurar de forma automática en cualquier interfaz usando el prefijo de enlace local FE80 :: / 10 (1111 1110 10) y el identificador de interfaz en el formato EUI-64 modificado.

Las direcciones de los enlaces locales se utilizan en el protocolo de descubrimiento de vecinos y el proceso de configuración automática sin estado. Los nodos en un enlace local pueden usar direcciones de enlace local para comunicarse; los nodos no necesitan direcciones únicas globales para comunicarse. La Figura 10 muestra la estructura de una dirección de enlace local.

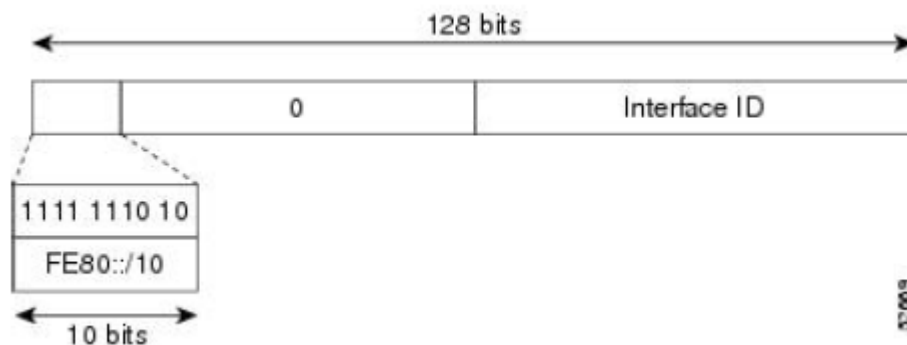


Figura 10. Formato de dirección de enlace local

Fuente: Cisco Systems, Inc. (2012). IPv6 Configuration Guide, Cisco IOS. Release 15.2MT. Recuperado el 2015 de mayo de 14, de <http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6/configuration/15-2mt/ipv6-15-2mt-book.pdf>

Direcciones Anycast

Las direcciones anycast son sintaxis indistinguibles de las direcciones unicast, porque las direcciones anycast son asignadas desde el espacio de las direcciones unicast. La asignación de una dirección unicast a más de un interfaz hace que una dirección unicast sea una dirección anycast. Los nodos a los que se asigna la dirección anycast se deben configurar explícitamente para reconocer que la dirección es una dirección anycast.

Las direcciones anycast se pueden utilizar solamente por un dispositivo, no un host, además no deben utilizarse como la dirección de origen de un paquete IPv6.

La Figura 11 muestra el formato de la dirección del dispositivo de subred anycast; la dirección tiene un prefijo concatenado por una serie de ceros (el ID de interfaz). La dirección anycast del dispositivo de subred se puede utilizar para llegar a un dispositivo en el enlace que se identifica por el prefijo de la dirección anycast dispositivo de subred.

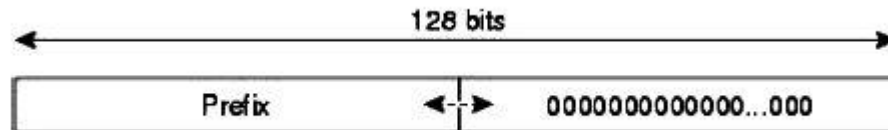


Figura 11. Estructura de Dirección anycast

Fuente: Cisco Systems, Inc. (2012). IPv6 Configuration Guide, Cisco IOS. Release 15.2MT. Recuperado el 2015 de mayo de 14, de <http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6/configuration/15-2mt/ipv6-15-2mt-book.pdf>

Direcciones Multicast

Una dirección multicast IPv6 es una dirección que tiene un prefijo de FF00 :: / 8 (1111 1111). Una dirección IPv6 multicast es un identificador para un conjunto de interfaces que normalmente pertenecen a diferentes nodos. Un paquete enviado a una dirección multicast se entrega a todas las interfaces identificadas por la dirección multicast. El segundo octeto que sigue al prefijo define el tiempo de vida y el alcance de la dirección multicast.

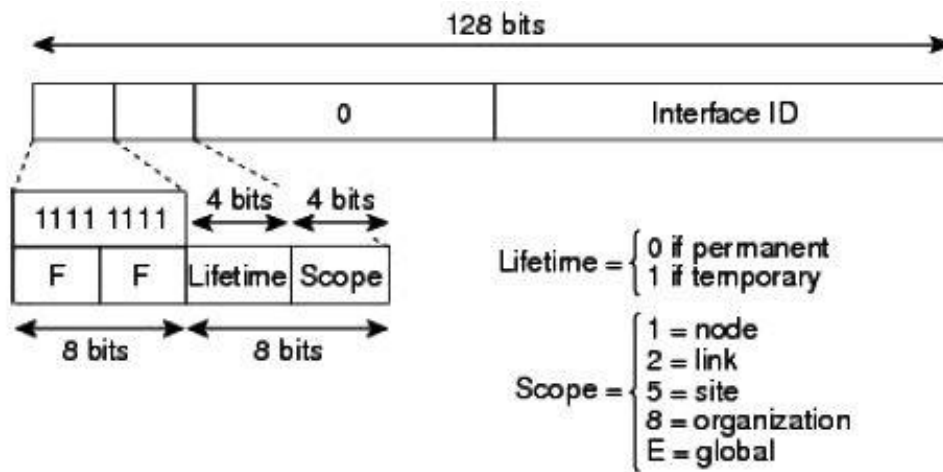


Figura 12. Formato de dirección multicast IPv6

Fuente: Cisco Systems, Inc. (2012). IPv6 Configuration Guide, Cisco IOS. Release 15.2MT. Recuperado el 2015 de mayo de 14, de <http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6/configuration/15-2mt/ipv6-15-2mt-book.pdf>

4. DESARROLLO DEL PROYECTO

A continuación se presentan las pruebas, emulaciones y documentaciones realizadas en el desarrollo del proyecto, en las que se evidencian los pasos para alcanzar los objetivos del mismo. Con el propósito no sólo de dar mayor entendimiento de la investigación, sino también para generar material que aporte a futuros trabajos relacionadas con el tema, todos los procedimientos están sustentados en anexos que comprenden ejemplos, tutoriales, explicaciones y, en general, elementos creados durante la investigación para respaldar cada detalle expuesto. Estos anexos son valor agregado importante debido a que no existen documentos publicados que acompañen y expliquen de una manera clara y sencilla el uso de muchas herramientas necesarias para la emulación de Redes Definidas por Software.

Inicialmente se realizó un reconocimiento de las herramientas software y protocolos que eran necesarios para la realización de éste, como el emulador, las Southbound y Northbound API, controladores, entre otros.

4.1. Capacidades de la herramienta Mininet

La herramienta de emulación seleccionada fue Mininet. Como se menciona anteriormente, Mininet es un emulador de SDN que funciona bajo el kernel de Linux y permite realizar pruebas con diferentes topologías tan complejas como se requiera.

Acorde con los objetivos presentados, se confirmó que efectivamente Mininet permitía la emulación de redes SDN que funcionaran con IPv6, para esto entonces se comprobó que este emulador admitiera la utilización de diferentes controladores y switches virtuales que soportaran diferentes versiones del OpenFlow, topologías, asignación de direcciones IPv6, pruebas de conectividad con esta versión de IP, entre otros.

Una de esas capacidades que permitieron realizar las emulaciones pertenecientes a esta investigación a través de Mininet, es la posibilidad que ésta ofrece de utilizar **controladores remotos**. Cuando se inicia una red en Mininet, cada switch puede ser conectado a un controlador remoto, que puede estar en la máquina donde corre Mininet, fuera de ella o en cualquier parte del mundo.

Esta opción puede ser útil si ya se tiene una versión personalizada de un framework de un controlador y las herramientas de desarrollo instaladas en la máquina virtual, o si se quiere probar un controlador que corre en una máquina diferente a la local.

La siguiente es una línea de comando base para el uso de un controlador remoto, donde [controller IP], es la dirección IP donde está corriendo el controlador remoto y [controller listening port] es el puerto en el que este controlador está escuchando.

\$ sudo mn --controller=remote,ip=[controller IP],port=[controller listening port]

Mininet también admite el uso de Wireshark para inspeccionar el tráfico entre los diferentes elementos emulados en una red, entre éstos revisar el uso de OpenFlow y las versiones de IP, como IPv6.

Con Mininet también es posible codificar topologías personalizadas a través de Python, lo que quiere decir que no se está necesariamente obligado a usar las que esta herramienta trae por defecto (tree, single, linear, etc.) sino que pueden crearse diferentes dispositivos, conectarse y distribuirse de la manera que se requiera.

Debido a que lo que se necesita en la parte inicial de este trabajo es el reconocimiento de las capacidades de esta herramienta y teniendo en cuenta que en su página oficial www.mininet.org, existe numerosa información y material que puede ser útil para reconocer muchos de los elementos de este emulador, no se ve necesario realizar una exposición detallada de sus funcionalidades.

En el Anexo A pueden consultarse algunos de los comandos más comunes que son utilizados en Mininet.

Además, como producto de todo el proceso llevado a cabo en la investigación que aquí se presenta, se construyó un artículo que está publicado en la [Revista Entre Ciencia e Ingeniería No. 17](#) donde se expone de manera detallada muchas de las características de este emulador.

En las siguientes secciones se mostrará el uso de éstas y otras funcionalidades de Mininet en ejemplos específicos.

4.2. Selección de la versión de OpenFlow

De la revisión documental se eligió OpenFlow como Southbound API para la realización de las pruebas y emulaciones siguientes por ser el protocolo más usado, compatible con diferentes controladores y con mayor documentación,

además porque, como se muestra en la Tabla 3, a partir de OpenFlow 1.3 existe soporte completo para IPv6.

El uso de una versión específica de OpenFlow requiere que tanto el controlador como el o los switches la soporten, por esto fue necesario reconocer qué controladores y qué software switch podían ser utilizados para la realización de una emulación con Mininet.

Tabla 3. Protocolos y campos soportados por las diferentes versiones de OpenFlow

	OF 1.0	OF 1.1	OF 1.2	OF 1.3 y OF 1.4
Puerto de entrada	X	X	X	X
Metadatos		X	X	X
Ethernet: src, dst, type	X	X	X	X
IPv4: src, dst, proto, ToS	X	X	X	X
TCP/UDP: src port, dst port	X	X	X	X
MPLS: label, traffic class		X	X	X
OpenFlow Extensible Match (OXM)			X	X
IPv6: src, dst, flow label, ICMPv6			X	X
IPv6 Extension Headers				X

Fuente: Braun, W., & Menth, M. (2014). Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. *Future Internet*, 6(2), 302-336.

4.3. Pruebas con diferentes controladores

Las pruebas que se presentan seguidamente fueron realizadas en una máquina virtual creada con VirtualBox 4.3.12 en la que se instaló la distribución de Linux Ubuntu en su versión 12.04. Por su parte, el equipo físico donde se ejecutó la máquina virtual cuenta con un procesador Intel Core i3 de 64 bits y una memoria RAM de 4GB.

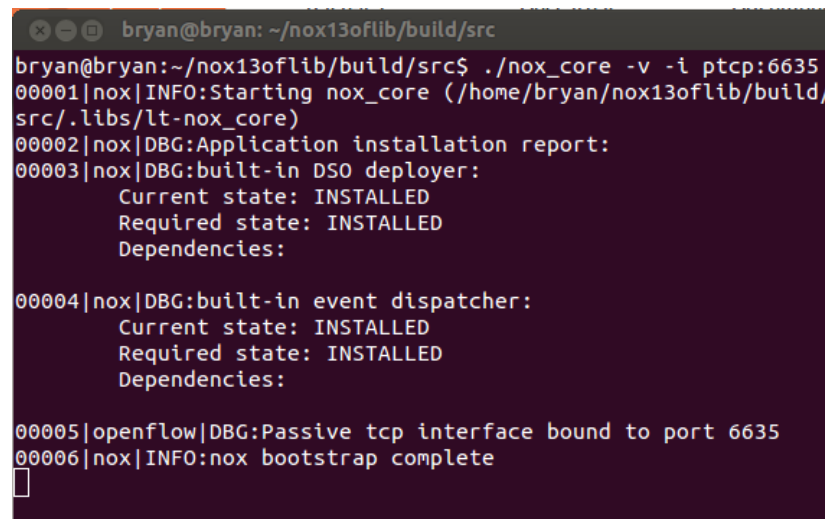
Los controladores, software switch y Mininet fueron instalados de manera nativa en Ubuntu. Para ver detalles más específicos de la instalación de estos elementos puede consultarse el Anexo B.

4.3.1. NOX13OFLIB

La primera emulación con soporte de OpenFlow 1.3 se llevó a cabo con el controlador NOX13OFLIB y ofsoftswitch13. NOX es uno de los primeros

controladores para Redes Definidas por Software, pero solamente soporta la versión 1.0 de OpenFlow. Para enmendar este problema la institución CPqD creó una versión de este controlador que soporta muchas de las nuevas características de OpenFlow 1.3 a la que llamó nox13oflib (nox13oflib, s.f.).

Para iniciar este controlador, dentro de la ruta **nox13oflib/build/src**, se ejecuta la instrucción: **./nox_core -v -i ptcp:<port>**, donde '**<port>**' se reemplaza por el puerto TCP al que se conectarán los switches. En el caso de la Figura 13, el puerto usado es el 6635.



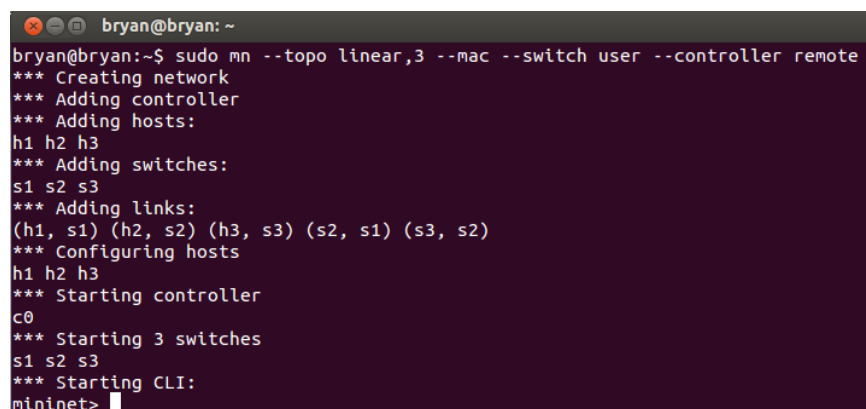
```
bryan@bryan: ~/nox13oflib/build/src
bryan@bryan:~/nox13oflib/build/src$ ./nox_core -v -i ptcp:6635
00001|nox|INFO:Starting nox_core (/home/bryan/nox13oflib/build/
src/.libs/lt-nox_core)
00002|nox|DBG:Application installation report:
00003|nox|DBG:built-in DSO deployer:
    Current state: INSTALLED
    Required state: INSTALLED
    Dependencies:

00004|nox|DBG:built-in event dispatcher:
    Current state: INSTALLED
    Required state: INSTALLED
    Dependencies:

00005|openflow|DBG:Passive tcp interface bound to port 6635
00006|nox|INFO:nox bootstrap complete
```

Figura 13. Comando para iniciar el controlador nox13oflib.

Este controlador puede instalarse e iniciarse en un equipo diferente al que utiliza Mininet o en el mismo, en este último caso, en otra ventana de Shell, puede crearse una topología que utilice este controlador. La Figura 14, ilustra la ejecución de una topología lineal que utiliza el controlador remoto nox13oflib.



```
bryan@bryan: ~
bryan@bryan:~$ sudo mn --topo linear,3 --mac --switch user --controller remote
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3
*** Starting CLI:
mininet>
```

Figura 14. Utilización de controlador remoto.

En el Anexo B puede consultarse un tutorial con el proceso completo de instalación de Ubuntu, Mininet y la emulación de una topología con este controlador.

A pesar de lograr comunicación en una topología emulada con ayuda del soporte completo de la versión 1.3 de OpenFlow con el controlador Nox13oflib y ofsoftswitch13, existen otras herramientas con muy buena documentación y comunidades de ayuda en línea que permiten ir un poco más allá en el uso de SDN e IPv6, una de éstas es el controlador RYU.

4.3.2. RYU

RYU es un framework para Redes Definidas por Software basado en componentes y escrito completamente en Python. RYU ofrece elementos software con una API bien definida que hace fácil a los desarrolladores crear nuevas aplicaciones de control y gestión de red. RYU soporta varios controladores y dispositivos de gestión de red como OpenFlow, Netconf, OF-config, entre otros. RYU también admite completamente las versiones 1.0, 1.1, 1.2, 1.3, 1.4 y 1.5 de OpenFlow (RYU, 2015).

Al ser una herramienta de código libre, cuenta con diferentes comunidades de ayuda como el [Mailing List](#), manejo del repositorio de código fuente libre [GitHub](#) y además varios libros con muy buena información no solo para el desarrollo de aplicaciones, sino también para el uso de las que ya trae por defecto este controlador.

En este sitio web <http://osrg.github.io/ryu-book/en/Ryubook.pdf>, está disponible uno de los documentos creados por el equipo RYU para el uso de aplicaciones y otros elementos de este Framework específicamente con OpenFlow 1.3.

Entre los ejercicios expuestos en este libro pueden resaltarse la implementación de un Firewall, elementos de Calidad de Servicio (QoS), VLANs y algunos relacionados con procesos de enrutamiento en SDN, en los que se prestó especial atención.

En el Anexo C pueden consultarse los pasos para la instalación de este controlador.

4.3.2.1. Prueba de enrutamiento usando RYU e IPv4

Una de las topologías utilizadas para los ejercicios de enrutamiento está representada por la Figura 15, que hace uso de una aplicación llamada rest_router, cuyo código completo está en el siguiente enlace https://github.com/osrg/ryu/blob/master/ryu/app/rest_router.py.

Como se expuso anteriormente, las Redes Definidas por Software por lo general hacen uso de un controlador, una interfaz para la comunicación entre éste y la capa de datos (en este caso OpenFlow), una aplicación que se ejecuta en la capa de control y una interfaz norte para la comunicación entre las aplicaciones y el controlador.

Ryu usa como interfaz norte REST para sus operaciones de OpenFlow. Además usa WSGI (un framework para conectar aplicaciones web y servidores web en Python), lo que permite introducir fácilmente nuevas REST APIs en una aplicación. REST está basado en una combinación de HTTP, JSON y XML (Roa, 2014).

Como se observa en la Figura 15, la topología de prueba está compuesta por 3 host y cada uno de ellos está conectado a un switch diferente, los cuales están conectados a su vez al controlador. Las interfaces están configuradas con IPv4.

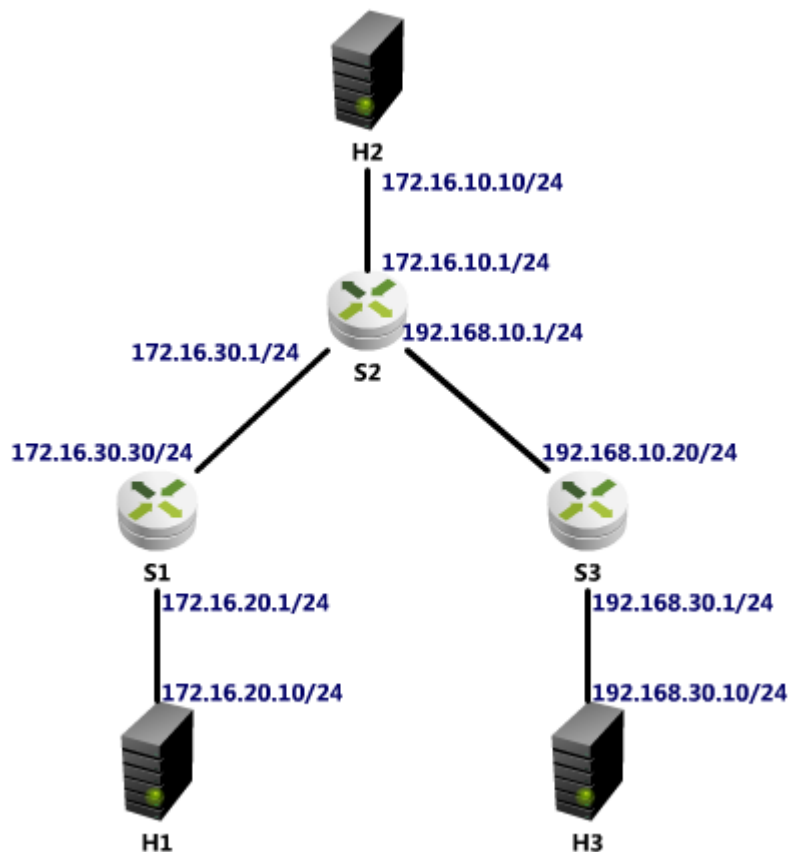


Figura 15. Topología emulada con OpenFlow 1.3 y el controlador RYU.

Para llevar a cabo esta prueba se utilizó Open vSwitch que es un switch virtual multicapa que está bajo la licencia de código abierto Apache 2.0. Está diseñado para permitir la automatización de red masiva a través de extensiones programáticas que soportan interfaces de gestión estándar y protocolos como NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, entre otros. Además, está diseñado para soportar múltiples servidores físicos similares a VMware's vNetwork o Cisco's Nexus 1000V (Open vSwitch, 2010).

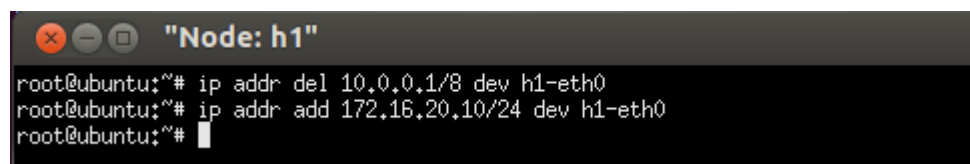
Los pasos completos de la ejecución de este ejercicio de enrutamiento están en el Anexo D.

El primer paso fue, entonces, iniciar Open vSwitch, posteriormente correr MiniNet con una topología lineal que hace uso de un controlador remoto y el switch virtual mencionado, mediante el siguiente comando:

sudo mn --topo linear,3 --mac --switch ovsk --controller remote.

Consecutivamente es necesario configurar cada switch para que éstos utilicen la versión OpenFlow 1.3. Para lograrlo se utilizó el emulador de terminal xterm para cada uno de los 3 switches de la topología y el comando: **ovs-vsctl set Bridge <nombreSwitch> protocols=OpenFlow13.**

Por defecto, Mininet asigna direcciones en el rango de 10.0.0.X a cada host; pero para lograr la emulación de la red mostrada en la Figura 15, fue necesario la asignación de direcciones IP diferentes, que se realiza eliminando la IP original y adhiriendo una nueva. La Figura 16 muestra un ejemplo de esta asignación en el host 1.



```
root@ubuntu:~# ip addr del 10.0.0.1/8 dev h1-eth0
root@ubuntu:~# ip addr add 172.16.20.10/24 dev h1-eth0
root@ubuntu:~#
```

Figura 16. Eliminación y asignación de IP a host 1 en MiniNet

El siguiente paso es correr el controlador. Dentro del xterm c0 se ejecuta el comando de la Figura 17. Nótese que la aplicación escogida es la anteriormente mencionada **rest_router.py**.

Para entender un poco más este ejercicio, es importante aclarar que lo que esta aplicación permite es convertir los switch en dispositivos de enrutamiento para que dejen de ser simples hubs que envían paquetes sin discriminación y pasen a obedecer a tablas de enrutamiento que pueden ser personalizadas y configuradas, lo que representa un enrutamiento estático.

Aquí entra en juego la interfaz norte, que es lo que permitió configurar los elementos necesarios para esta aplicación, entre éstos la asignación de direcciones a los routers, los gateways y rutas por defecto. Esta configuración se llevó a cabo a través de la API Rest que para este fin proporciona RYU.

Cada configuración genera como respuesta un JSON con los resultados del proceso y algunos datos adicionales, que pueden verse en el ejemplo de la Figura 18.

De esta manera se pudo, finalmente, comprobar la conexión entre los nodos y host de la red.

```
root@ubuntu:~# cd ryu
root@ubuntu:~/ryu# ./bin/ryu-manager --verbose ryu/app/rest_router.py
loading app ryu/app/rest_router.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu/app/rest_router.py of RestRouterAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK dpset
  PROVIDES EventDP TO {'RestRouterAPI': set(['dpset'])}
  CONSUMES EventOFPPStateChange
  CONSUMES EventOFPPSwitchFeatures
  CONSUMES EventOFPPPortStatus
BRICK RestRouterAPI
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPPStatsReply
  CONSUMES EventOFPPFlowStatsReply
  CONSUMES EventDP
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'RestRouterAPI': set(['main'])}
  PROVIDES EventOFPPFlowStatsReply TO {'RestRouterAPI': set(['main'])}
  PROVIDES EventOFPPPortStatus TO {'dpset': set(['main'])}
  PROVIDES EventOFPPStateChange TO {'dpset': set(['main', 'dead'])}
  PROVIDES EventOFPPSwitchFeatures TO {'dpset': set(['config'])}
  PROVIDES EventOFPPStatsReply TO {'RestRouterAPI': set(['main'])}
  CONSUMES EventOFPPSwitchFeatures
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPPPortDescStatsReply
  CONSUMES EventOFPHello
  CONSUMES EventOFPErrormsg
(4493) wsgi starting up on http://0.0.0.0:8080/
connected socket:<eventlet.greenio.base.GreenSocket object at 0x30f8fd0> address
:('127.0.0.1', 34774)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x2ee1a50>
move onto config mode
```

Figura 17. Ejecución del controlador RYU

```

root@ubuntu:~#
root@ubuntu:~# set -x
root@ubuntu:~#
root@ubuntu:~# curl -X POST -d '{"address":"172.16.20.1/24"}' http://localhost:
8080/router/000000000000000001 | python -mjson.tool
+ curl -X POST -d '{"address":"172.16.20.1/24"}' http://localhost:8080/router/00
0000000000000001
+ python -mjson.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
100    147    100    119    100     28   5030    1183   --:--:--  --:--:--  --:--:--  7000
[
  {
    "command_result": [
      {
        "details": "Add address [address_id=1]",
        "result": "success"
      }
    ],
    "switch_id": "000000000000000001"
  }
]

```

Figura 18. Ajuste de direcciones mediante la API REST de RYU.

4.4. Pruebas en IPv4

4.4.1. Prueba usando iperf

Iperf es una herramienta que permite crear flujos de datos TCP y UDP con el fin de hacer mediciones relacionadas con el rendimiento de una red, como ancho de banda, retardo jitter y pérdida de datagramas. En la Figura 19, puede observarse el uso de Iperf para hacer mediciones de tráfico en la emulación de la red anteriormente descrita.

```

"Node: h2"
root@ubuntu:~/Descargas/archivossh# iperf -c 172.16.20.10
-----
Client connecting to 172.16.20.10, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 19] local 172.16.10.10 port 43762 connected with 172.16.20.10 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-10.0 sec  11.0 GBytes  9.46 Gbits/sec
root@ubuntu:~/Descargas/archivossh#

"Node: h1"
root@ubuntu:~/Descargas/archivossh# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 20] local 172.16.20.10 port 5001 connected with 172.16.10.10 port 43762
[ ID] Interval      Transfer    Bandwidth
[ 20] 0.0-10.0 sec  11.0 GBytes  9.44 Gbits/sec

```

Figura 19. Prueba con Iperf de SDN con OpenFlow 1.3

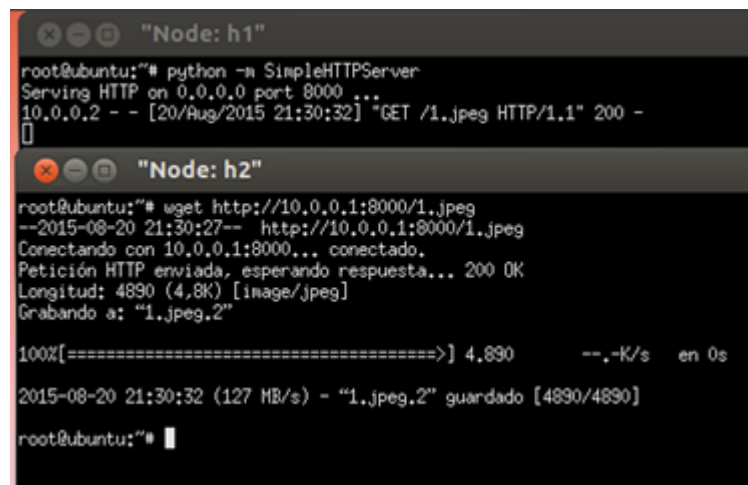
Como puede verse, la prueba se realizó con envío de paquetes TCP en la que el host2 es configurado como cliente y el host1 como servidor, al final se genera una

especie de informe con los datos registrados, en el que puede verse que el ancho de banda fue de alrededor 9.44 Gbits/sec.

4.4.2. Prueba de tráfico con servidor HTTP

Otra prueba realizada para comparar el correcto enrutamiento de los nodos y, por tanto, de la aplicación rest_router.py en RYU fue la creación de un servidor HTTP en uno de los host y el posterior envío de archivos como imágenes, documentos, audios, videos, entre otros.

Después de instalar los elementos necesarios para la creación de este tipo de servidores en Linux e iniciarlo en el host1, se solicitaron diferentes tipos de archivos desde el host2 que hizo las veces de cliente.



```
"Node: h1"
root@ubuntu:~# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
10.0.0.2 - - [20/Aug/2015 21:30:32] "GET /1.jpeg HTTP/1.1" 200 -

"Node: h2"
root@ubuntu:~# wget http://10.0.0.1:8000/1.jpeg
--2015-08-20 21:30:27-- http://10.0.0.1:8000/1.jpeg
Conectando con 10.0.0.1:8000... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 4890 (4,8K) [image/jpeg]
Grabando a: "1.jpeg.2"

100%[=====] 4,890 --.-K/s en 0s

2015-08-20 21:30:32 (127 MB/s) - "1.jpeg.2" guardado [4890/4890]

root@ubuntu:~#
```

Figura 20. Creación y uso de un servidor HTTP en una red emulada con RYU.

Como se aprecia en la Figura 20, el envío de la imagen “1.jpg” se realizó de manera exitosa, esto se comprobó ingresando en la ruta y espacio de almacenamiento usado por el host2 para ver que efectivamente había una copia de este archivo. Para ver la ejecución de esta prueba de manera más detallada puede consultarse el Anexo F.

4.5. Emulación de Redes Definidas por Software con IPv6

4.5.1. Prototipo No. 1

La primera topología emulada de una Red Definida por Software usando IPv6 es la que se muestra en la Figura 21, compuesta simplemente por un switch y dos host que fueron configurados con direcciones IPv6. Esta primera emulación se realizó utilizando el controlador expuesto anteriormente denominado nox13oflib.

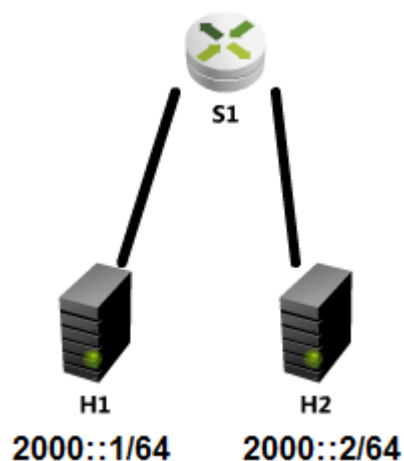


Figura 21. Topología IPv6 No. 1

Posterior a la comprobación del uso de OpenFlow 1.3 con WireShark se realizó una prueba de comunicación con IPv6 entre dos host con el uso de una aplicación que generaba patrones de comportamiento tipo Hub a los switches. La Figura 22, muestra la realización de un ping IPv6 entre un host número uno (h1) y un host número dos (h2).

```
santiago@ubuntu:~$ sudo mn --topo single,2 --mac --switch user
--controller remote
[sudo] password for santiago:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 ifconfig h1-eth0 inet6 add 2000::1/64
mininet> h2 ifconfig h2-eth0 inet6 add 2000::2/64
mininet> h1 ping6 2000::2 -I h1-eth0
PING 2000::2(2000::2) from 2000::1 h1-eth0: 56 data bytes
64 bytes from 2000::2: icmp_seq=1 ttl=64 time=3.98 ms
64 bytes from 2000::2: icmp_seq=2 ttl=64 time=2.81 ms
64 bytes from 2000::2: icmp_seq=3 ttl=64 time=2.40 ms
64 bytes from 2000::2: icmp_seq=4 ttl=64 time=3.66 ms
64 bytes from 2000::2: icmp_seq=5 ttl=64 time=3.55 ms
```

Figura 22. PING IPv6 entre dos host usando Nox13oflib y ofsoftswitch13

En el Anexo E pueden consultarse los pasos detallados para la ejecución de esta prueba.

4.5.2. Prototipo No. 2

La topología No. 2 es la que se muestra en la Figura 23, compuesta por 4 switches y 3 host. Cada una de las interfaces de los host fue configurada con una dirección IPv6 de tipo Unicast Local que se listan en la Tabla 4.

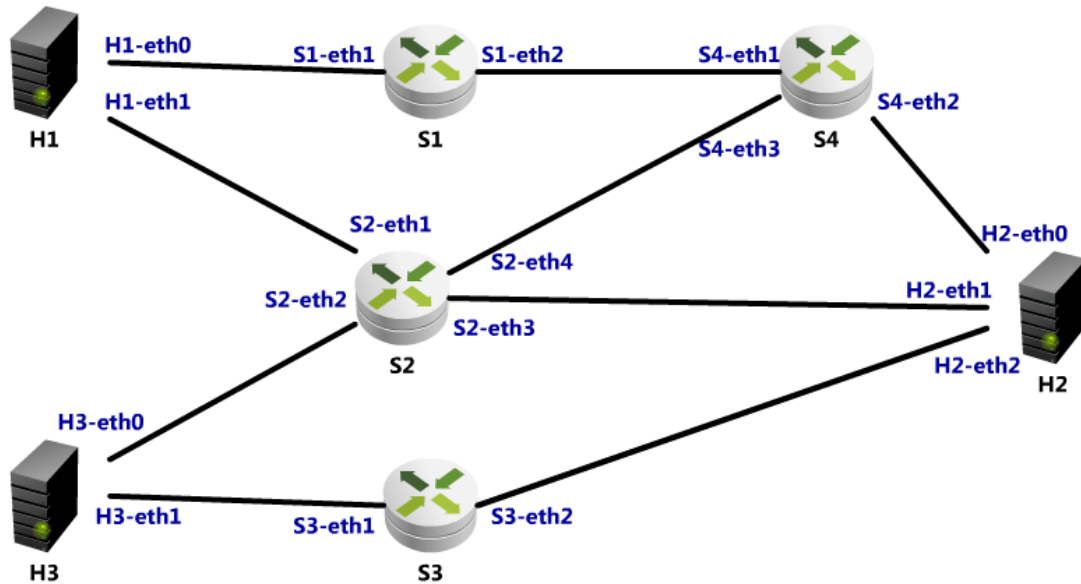


Figura 23. Topología IPv6 No. 2

Como puede notarse, esta no es una topología de las que Mininet tiene por defecto para realizar pruebas. El diseño de esta red se realizó mediante un script de Python que permite la creación de topologías personalizadas. El Script es el siguiente:

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
        s3 = self.addSwitch( 's3' )
        s4 = self.addSwitch( 's4' )
```

```
# Add links
self.addLink( h1, s1 )
self.addLink( s1, s4 )
self.addLink( h1, s2 )
self.addLink( h3, s2 )
self.addLink( h3, s3 )
self.addLink( s4, h2 )
self.addLink( s2, h2 )
self.addLink( s3, h2 )
self.addLink( s2, s4 )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Tabla 4. Direcciones IPv6 del prototipo No. 2

HOST	INTERFAZ	DIRECCIÓN IPv6
H1	h1-eth0	fc00::1/64
H1	h1-eth1	fc00::2/64
H2	h2-eth0	fc00::3/64
H2	h2-eth1	fc00::4/64
H3	h2-eth2	fc00::5/64
H3	h3-eth0	fc00::6/64
H3	h3-eth1	fc00::7/64

El controlador utilizado para esta segunda prueba fue RYU, en el que se corrió una aplicación denominada **simple_switch_13.py**. Esta aplicación hace que los switches actúen como un switch clásico. OpenFlow ha denominado a este comportamiento **Learning Switch**. Con esta aplicación el switch examinará cada paquete y aprenderá el puerto origen que le corresponde. A partir de este momento, la dirección MAC origen será asociada con ese puerto. Si el destino de un paquete ya está asociado a algún puerto, el paquete será enviado al puerto dado, de lo contrario se inundarán todos los puertos del switch con éste (Create a Learning Switch, 2015).

En general, lo que hace es que a medida que el switch va recibiendo tramas va creando una tabla donde se relacionan direcciones MAC con puertos, esto se debe a que el controlador enviará un mensaje OpenFlow al switch instaurando una nueva entrada en la tabla de flujo (Roncero, 2014).

En la Figura 24 puede observarse la ejecución de la topología personalizada que se diseñó para el despliegue de una red SDN con IPv6. Como puede notarse, se añadieron 3 hosts y 4 switches, como también los enlaces entre estos elementos correspondientes al diseño presentado previamente.

```
bryan@bryan-VirtualBox:~/mininet/custom$ sudo mn --custom topoIPv6.py --topo myt
opo --switch ovsk --controller remote
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h1, s2) (h3, s2) (h3, s3) (s1, s4) (s2, h2) (s3, h2) (s4, h2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
```

Figura 24. Ejecución en Mininet de topología IPv6 No. 2

Posterior a la ejecución de la topología en el emulador Mininet, se inició el controlador RYU (ver Figura 25), en el que, como puede verse, se utilizó la aplicación ***simple_switch_13.py***, mencionada anteriormente.

```
santiago@ubuntu:~$ cd ryu/
santiago@ubuntu:~/ryu$ ./bin/ryu-manager --verbose ryu/app/simple_switch_13.py
loading app ryu/app/simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ryu/app/simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPacketIn
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
  CONSUMES EventOFPErrormsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPPortDescStatsReply
connected socket:<eventlet.greenio.base.GreenSocket object at 0x1eb7310> address
:('127.0.0.1', 49667)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x1eb7850>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0xeeb111df OFPSwitchFeatures(au
xiliary_id=0,capabilities=79,datapath_id=1,n_buffers=256,n_tables=254)
move onto main mode
```

Figura 25. Ejecución RYU con la aplicación *simple_switch_13.py*.

Luego de iniciar la topología, se procedió con la asignación de las direcciones IPv6 a cada una de las interfaces de red de los host. Las direcciones asignadas fueron las direcciones expuestas en la Tabla 4. El procedimiento llevado a cabo para este fin es mostrado en la Figura 26.

```
*** Starting CLI:
mininet> h1 ifconfig h1-eth0 inet6 add fc00::1/64
mininet> h1 ifconfig h1-eth1 inet6 add fc00::2/64
mininet> h2 ifconfig h2-eth0 inet6 add fc00::3/64
mininet> h2 ifconfig h2-eth1 inet6 add fc00::4/64
mininet> h2 ifconfig h2-eth2 inet6 add fc00::5/64
mininet> h3 ifconfig h3-eth0 inet6 add fc00::6/64
mininet> h3 ifconfig h3-eth1 inet6 add fc00::7/64
```

Figura 26. Asignación de direcciones IPv6 en Mininet.

Con la función “links” de Mininet, pudo evidenciarse que evidentemente estaban los enlaces entre los dispositivos exactamente como se codificó la topología con Python. En la Figura 27, puede verse en la primera línea, por ejemplo la conexión entre la interfaz eth0 del host 1 y la interfaz eth1 del switch 1.

```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h1-eth1<->s2-eth1 (OK OK)
h3-eth0<->s2-eth2 (OK OK)
h3-eth1<->s3-eth1 (OK OK)
s1-eth2<->s4-eth1 (OK OK)
s2-eth3<->h2-eth1 (OK OK)
s3-eth2<->h2-eth2 (OK OK)
s4-eth2<->h2-eth0 (OK OK)
```

Figura 27. Comprobación de conexiones entre dispositivos con función “links”

5. PRESENTACIÓN Y ANÁLISIS DE LOS RESULTADOS

A continuación se exhiben pruebas de conectividad realizadas al prototipo de red No. 1 y No. 2 presentados anteriormente que muestran los resultados de la emulación de estas topologías de redes SDN configuradas con IPv6.

5.1. Pruebas de conectividad con ping6

Utilizando el prototipo de red No. 2, inicialmente se comprueba conectividad con ping IPv6 entre los dos host más distanciados, en este caso entre el host 1 y el host 2. De esta prueba en esta emulación se obtuvo que de los 17 paquetes enviados, fueron transmitidos y recibidos de vuelta en su totalidad los 17 y por tanto no hubo pérdida de paquetes.

```
mininet> h1 ping6 fc00::3 -I h1-eth0
PING fc00::3(fc00::3) from fc00::1 h1-eth0: 56 data bytes
64 bytes from fc00::3: icmp_seq=1 ttl=64 time=229 ms
64 bytes from fc00::3: icmp_seq=2 ttl=64 time=0.337 ms
64 bytes from fc00::3: icmp_seq=3 ttl=64 time=0.073 ms
64 bytes from fc00::3: icmp_seq=4 ttl=64 time=0.076 ms
64 bytes from fc00::3: icmp_seq=5 ttl=64 time=0.045 ms
64 bytes from fc00::3: icmp_seq=6 ttl=64 time=0.072 ms
64 bytes from fc00::3: icmp_seq=7 ttl=64 time=0.074 ms
64 bytes from fc00::3: icmp_seq=8 ttl=64 time=0.079 ms
64 bytes from fc00::3: icmp_seq=9 ttl=64 time=0.077 ms
64 bytes from fc00::3: icmp_seq=10 ttl=64 time=0.079 ms
64 bytes from fc00::3: icmp_seq=11 ttl=64 time=0.074 ms
64 bytes from fc00::3: icmp_seq=12 ttl=64 time=0.109 ms
64 bytes from fc00::3: icmp_seq=13 ttl=64 time=0.079 ms
64 bytes from fc00::3: icmp_seq=14 ttl=64 time=0.073 ms
64 bytes from fc00::3: icmp_seq=15 ttl=64 time=0.076 ms
64 bytes from fc00::3: icmp_seq=16 ttl=64 time=0.071 ms
64 bytes from fc00::3: icmp_seq=17 ttl=64 time=0.077 ms
^C
--- fc00::3 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16001ms
rtt min/avg/max/mdev = 0.045/13.604/229.798/54.048 ms
```

Figura 28. Ping IPv6 en Topología No. 2

Para comprobar e inspeccionar el tráfico que generó esta emulación, se utilizó el analizador de paquetes de red Wireshark. Wireshark permite capturar y mostrar los paquetes de datos que pasan por una red de la manera más detallada posible. Realizando una analogía, un analizador de paquetes es como un dispositivo de medición para examinar lo que está pasando dentro de un cable de red, así como

lo haría un voltímetro para examinar lo que está pasando por un cable eléctrico (Wireshark, 2014).

Al ser Mininet un emulador, utiliza la tarjeta de red del equipo físico para, por decirlo de alguna manera, darle vida a los dispositivos de la topología que está recreando, por lo tanto inicialmente se eligió la interfaz loopback para analizar la comunicación entre los switches y el controlador y comprobar que se estaba utilizando OpenFlow en su versión 1.3.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	OF 1.3	74	of echo request
2	0.001291	127.0.0.1	127.0.0.1	OF 1.3	74	of echo reply
3	0.001387	127.0.0.1	127.0.0.1	TCP	66	50664 > 6633 [ACK] Seq=9 Ack=9 Win=86 Len=0 TSval=4674750 TSecr=4674750
4	1.000570	127.0.0.1	127.0.0.1	OF 1.3	74	of echo request
5	1.000809	127.0.0.1	127.0.0.1	OF 1.3	74	of echo request
6	1.000917	127.0.0.1	127.0.0.1	OF 1.3	74	of echo request
7	1.002983	127.0.0.1	127.0.0.1	OF 1.3	74	of echo reply
8	1.003032	127.0.0.1	127.0.0.1	TCP	66	50663 > 6633 [ACK] Seq=9 Ack=9 Win=86 Len=0 TSval=4675000 TSecr=4675000
9	1.004866	127.0.0.1	127.0.0.1	OF 1.3	74	of echo reply
10	1.004908	127.0.0.1	127.0.0.1	TCP	66	50662 > 6633 [ACK] Seq=9 Ack=9 Win=86 Len=0 TSval=4675001 TSecr=4675001
11	1.012033	127.0.0.1	127.0.0.1	OF 1.3	74	of echo reply
12	1.012088	127.0.0.1	127.0.0.1	TCP	66	50665 > 6633 [ACK] Seq=9 Ack=9 Win=86 Len=0 TSval=4675003 TSecr=4675003

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
 Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
 Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
 Transmission Control Protocol, Src Port: 50664 (50664), Dst Port: 6633 (6633), Seq: 1, Ack: 1, Len: 8
 OpenFlow

Figura 29. Análisis de paquetes de datos para OpenFlow 1.3.

Como puede observarse en la Figura 29 esta emulación hace uso de OpenFlow 1.3 para la comunicación entre el controlador y los switches. Tanto la dirección de origen como la de destino es la loopback (127.0.0.1), ya que, como se explica previamente, Mininet utiliza esta interfaz para emular tanto el controlador como los switches.

Con ayuda de Wireshark también se evidenció el uso de ICMPv6 al momento de realizar pruebas de conectividad entre dos hosts configurados con IPv6. Como explica Conta (2006) en la RFC 443, ICMPv6 es usado por los nodos IPv6 para reportar errores encontrados en el procesamiento de paquetes, para llevar a cabo otras funciones en capa de Internet y para realizar diagnósticos como el que presenta la Figura 30 a través del ICMPv6 “ping”.

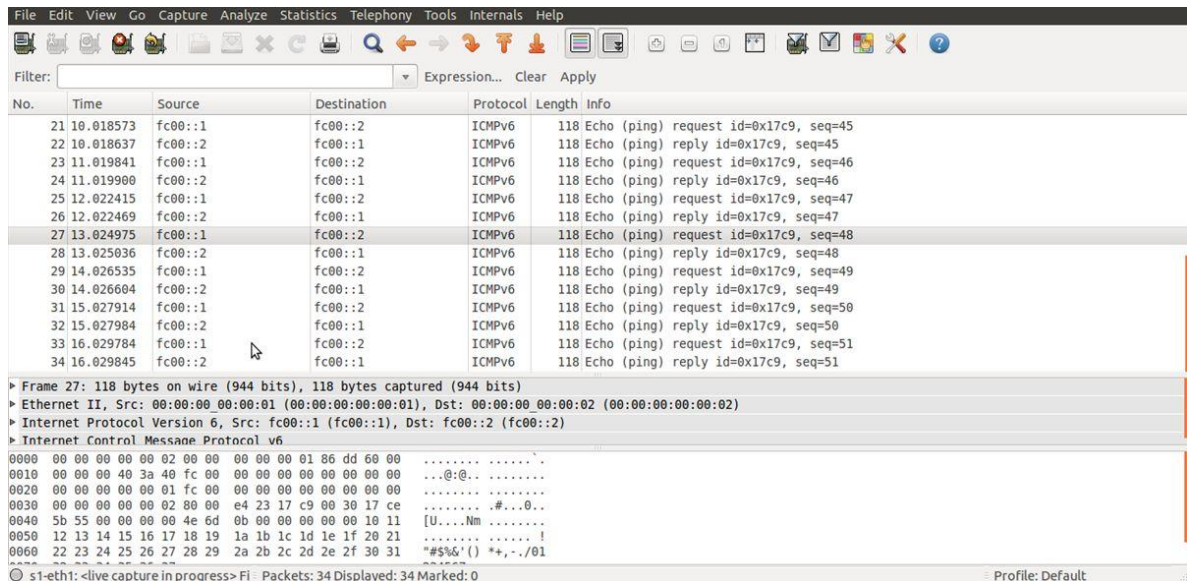


Figura 30. ICMPv6 en emulación de SDN.

5.2. Pruebas con IPERF e IPv6

Se utilizó la herramienta IPERF para realizar pruebas inyectando paquetes UDP entre un host servidor y un host cliente. Después de abrir individualmente los host con la herramienta xterm, se escribe el siguiente comando en el nodo h1:

iperf -V -s -u -B fc00::2,

donde “-V” indica la utilización del protocolo IPv6, “-s” configura el host como servidor, “-u” permite especificar que el tipo de paquetes a enviar son UDP y “-B” establece la interfaz entrante. Esto sólo es útil en hosts múltiples, que tienen varias interfaces de red (iPerf).

Por su parte en el host2, que hace las veces de cliente, se ejecuta el siguiente comando:

iperf -u -t 10 -i 1 -V -c fc00::2,

donde “-u” indica el envío de paquetes UDP, “-t 10” señala los segundos de duración, “-i 1” establece el intervalo de tiempo en segundos entre cada reporte periódico de ancho de banda, jitter y pérdida, “-V” indica la utilización de IPv6, “-c” configura el host como cliente.

5.2.1. Prueba IPERF en Prototipo No. 1

Los resultados de esta prueba en el prototipo de red No. 1 pueden evidenciarse en la **¡Error! No se encuentra el origen de la referencia..**

```
"Node: h1"
root@ubuntu:~# iperf -V -s -u -B 2000::1
-----
Server listening on UDP port 5001
Binding to local address 2000::1
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 13] local 2000::1 port 5001 connected with 2000::2 port 46385
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 13] 0.0-13.7 sec  8.13 MBytes  4.99 Mbits/sec  0.237 ms   0/ 5802 (0%)
^C
```

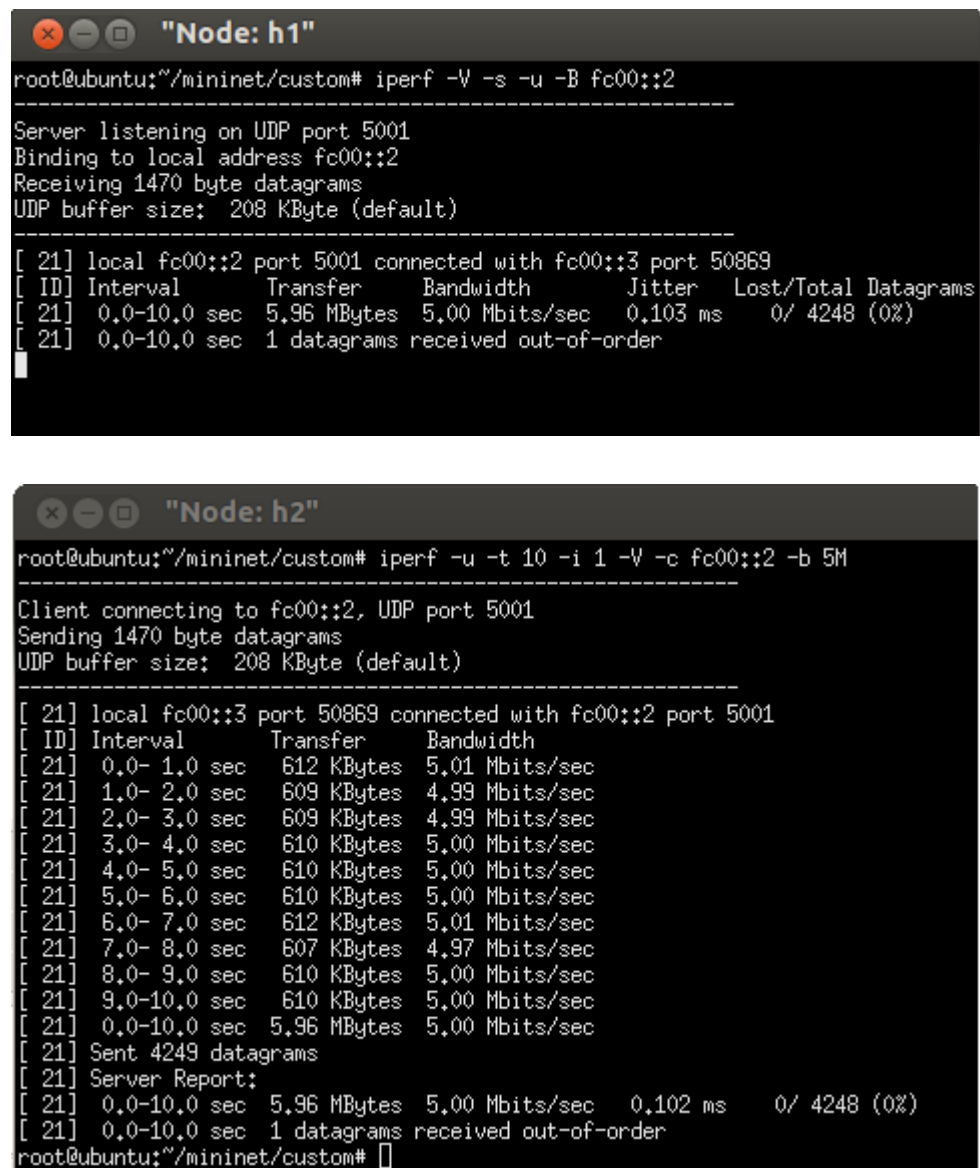
```
"Node: h2"
Client connecting to 2000::1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 13] local 2000::2 port 46385 connected with 2000::1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0- 1.0 sec   599 KBytes  4.90 Mbits/sec
[ 13] 1.0- 2.0 sec   610 KBytes  5.00 Mbits/sec
[ 13] 2.0- 3.0 sec   612 KBytes  5.01 Mbits/sec
[ 13] 3.0- 4.0 sec   596 KBytes  4.88 Mbits/sec
[ 13] 4.0- 5.0 sec   610 KBytes  5.00 Mbits/sec
[ 13] 5.0- 6.0 sec   610 KBytes  5.00 Mbits/sec
[ 13] 6.0- 7.0 sec   612 KBytes  5.01 Mbits/sec
[ 13] 7.0- 8.0 sec   610 KBytes  5.00 Mbits/sec
[ 13] 8.0- 9.0 sec   610 KBytes  5.00 Mbits/sec
[ 13] 9.0-10.0 sec   610 KBytes  5.00 Mbits/sec
[ 13] 10.0-11.0 sec   610 KBytes  5.00 Mbits/sec
[ 13] 11.0-12.0 sec   610 KBytes  5.00 Mbits/sec
[ 13] 12.0-13.0 sec   610 KBytes  5.00 Mbits/sec
^C[ 13] 0.0-13.7 sec  8.13 MBytes  4.98 Mbits/sec
[ 13] Sent 5802 datagrams
[ 13] Server Report:
[ 13] 0.0-13.7 sec  8.13 MBytes  4.99 Mbits/sec  0.236 ms   0/ 5802 (0%)
root@ubuntu:~#
```

Figura 31. Prueba IPERF en Prototipo No. 1

En la prueba con IPERF en el primer Prototipo hubo una transferencia de aproximadamente 5802 datagramas, hubo un jitter de 0.237 ms y los rangos de ancho de banda en cada uno de los 13 intervalos en los que se tomó una muestra fue de alrededor de los 5Mbps/s como se configuró inicialmente.

En el servidor fue utilizada la interfaz 2000::1 en el puerto 46385 y en el cliente la interfaz 2000::2 en el puerto 5001.

5.2.2. Prueba IPERF en Prototipo No. 2



The figure consists of two terminal windows. The top window, titled "Node: h1", shows the server configuration and results. The bottom window, titled "Node: h2", shows the client configuration and results.

```

"Node: h1"
root@ubuntu:~/mininet/custom# iperf -V -s -u -B fc00::2
-----
Server listening on UDP port 5001
Binding to local address fc00::2
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 21] local fc00::2 port 5001 connected with fc00::3 port 50869
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 21] 0.0-10.0 sec  5.96 MBytes  5.00 Mbits/sec  0.103 ms   0/ 4248 (0%)
[ 21] 0.0-10.0 sec  1 datagrams received out-of-order

"Node: h2"
root@ubuntu:~/mininet/custom# iperf -u -t 10 -i 1 -V -c fc00::2 -b 5M
-----
Client connecting to fc00::2, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 21] local fc00::3 port 50869 connected with fc00::2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0- 1.0 sec   612 KBytes  5.01 Mbits/sec
[ 21] 1.0- 2.0 sec   609 KBytes  4.99 Mbits/sec
[ 21] 2.0- 3.0 sec   609 KBytes  4.99 Mbits/sec
[ 21] 3.0- 4.0 sec   610 KBytes  5.00 Mbits/sec
[ 21] 4.0- 5.0 sec   610 KBytes  5.00 Mbits/sec
[ 21] 5.0- 6.0 sec   610 KBytes  5.00 Mbits/sec
[ 21] 6.0- 7.0 sec   612 KBytes  5.01 Mbits/sec
[ 21] 7.0- 8.0 sec   607 KBytes  4.97 Mbits/sec
[ 21] 8.0- 9.0 sec   610 KBytes  5.00 Mbits/sec
[ 21] 9.0-10.0 sec   610 KBytes  5.00 Mbits/sec
[ 21] 0.0-10.0 sec  5.96 MBytes  5.00 Mbits/sec
[ 21] Sent 4249 datagrams
[ 21] Server Report:
[ 21] 0.0-10.0 sec  5.96 MBytes  5.00 Mbits/sec  0.102 ms   0/ 4248 (0%)
[ 21] 0.0-10.0 sec  1 datagrams received out-of-order
root@ubuntu:~/mininet/custom#

```

Figura 32. Prueba IPERF en Prototipo No. 2

En la prueba con IPERF en el segundo Prototipo hubo una transferencia de aproximadamente 4249 datagramas, hubo un jitter de 0.103 ms y los rangos de ancho de banda en cada uno de los 10 intervalos en los que se tomó una muestra fue de alrededor de los 5Mbps/s como se configuró inicialmente.

En el servidor fue utilizada la interfaz fc00::2 en el puerto 5001 y en el cliente la interfaz fc00::3 en el puerto 50869.

Estos resultados comprueban la implementación y rendimiento de un Prototipo de una red IPv6 Definida por Software emulada mediante la herramienta Mininet configurada inicialmente con una aplicación simple, pero que demuestra que efectivamente las SDN ofrecen los elementos necesarios para la implementación de la versión 6 del protocolo de Internet.

5.3. Comparativa entre IPv4 e IPv6 en diferentes controladores

Con el objetivo de realizar algunas comparaciones entre el funcionamiento de IPv4 e IPv6 en Redes Definidas por Software que permitieran distinguir diferencias en parámetros como el retardo se planteó y realizó una prueba utilizando la topología de la Figura 23. Esta prueba también permitió contrastar RYU y NOX13OFLIB ya que cada procedimiento se realizó en estos dos controladores.

La prueba consistió básicamente en realizar 4 test con la herramienta IPERF entre dos de los host más alejados de la topología (los mismos para cada prueba). Dos de esas pruebas se hicieron con RYU y las otras dos con NOXOFLIB. Para la primera prueba con cada controlador las interfaces de la topología se configuraron con IPv4 y para la segunda con IPv6 para obtener los datos comparativos entre estas dos versiones del protocolo de Internet en SDN.

La Tabla 5, muestra las direcciones utilizadas en cada interfaz para la prueba con IPv4 e IPv6.

Tabla 5. Direcciones IPv4 e IPv6 utilizadas para prueba comparativa

Host	Interfaz	Dirección IPv4	Dirección IPv6
H1	h1-eth0	176.168.16.1	2000::1/64
H1	h1-eth1	176.168.16.2	2000::2/64
H2	h2-eth0	176.168.16.3	2000::3/64
H2	h2-eth1	176.168.16.4	2000::4/64
H3	h2-eth2	176.168.16.5	2000::5/64

H3	h3-eth0	176.168.16.6	2000::6/64
H3	h3-eth1	176.168.16.7	2000::7/64

Para el caso específico de estas muestras, IPERF se configuró para utilizar UDP con un ancho de banda de 10 Mbits/s y con un total de muestras en 10 intervalos. Los resultados de estas pruebas se pueden ver en la Tabla 6.

Tabla 6. Resultados de prueba comparativa entre IPv4 e IPv6

Versión IP	Estado tablas de flujo	Parámetros evaluado	Controlador	
			NOX13OFLIB	RYU
IPv4	Incompletas	Transferencia	4.02 MBytes	5.15 MBytes
		Jitter	7.268 ms	2234.239 ms
		Porcentaje de datagramas perdidos	1006 (26%)	531 (13%)
		Total datagramas	3877	4206
	Completas	Transferencia	4.66 MBytes	5.96 MBytes
		Jitter	7.212 ms	0.009 ms
		Porcentaje de datagramas perdidos	753 (18%)	0 (0%)
		Total Datagramas	4077	4251
IPv6	Incompletas	Transferencia	3.35 MBytes	4.31 MBytes
		Jitter	13.671 ms	1432.044 ms
		Porcentaje de datagramas perdidos	1425 (35%)	1082 (26%)
		Total Datagramas	4076	4155
	Completas	Transferencia	3.13 MBytes	5.96 MBytes
		Jitter	3.820 ms	0.032 msn
		Porcentaje de datagramas perdidos	1018 (25%)	0 (0%)
		Total Datagramas	4074	4249

Las pruebas se realizaron con la aplicación ***simple_switch.py*** tanto en RYU como en NOX13OFLIB que como ya se explicó crea lo que se denomina Learning Switch. Debido al comportamiento de esta aplicación las pruebas se ejecutaron sin rutas aprendidas en los switch y posteriormente cuando los switch ya habían creado tablas de flujo para encaminar los datagramas.

En el primer intento (cuando los switches no tenían completas sus tablas de flujo) hubo una pérdida de datagramas de 13% y un Jitter 2234.239 ms en la prueba IPv4 en RYU y en NOX13OFLIB se perdieron 26% de datagramas y hubo un Jitter de 7.268ms. En el caso de IPv6, con RYU hubo un 26% de datagramas perdidos y un Jitter de 1432.044ms. Con NOX13OFLIB se perdieron el 35% de datagramas y el Jitter fue de 13.671 ms.

En el segundo intento (después de que se completaron las tablas de flujo) con RYU no hubo pérdida de paquetes ni con IPv4 ni con IPv6 y el Jitter no es superior en ninguna prueba a *0.1ms*. Por su parte con NOX13OFLIB hubo una reducción considerable en la pérdida de paquetes pasó del 26% al 18% en IPv4 y en IPv6 del 35% al 25%. El Jitter con este último controlador fue de *7.212 ms* para IPv4 y *3.820 ms* para IPv6.

5.4. Análisis de resultados y observaciones generales

De las pruebas comparativas entre IPv4 e IPv6 se evidenció que cuando los dispositivos de red no tienen aún llenas sus tablas de flujo y deben inundar con paquetes la red para conocer la ruta para enviar los datos, se genera una pérdida de datagramas considerable y un aumento del Jitter con las dos versiones de IP.

En relación con IPv4 e IPv6 en el caso del controlador RYU los parámetros evaluados no arrojaron diferencias significativas cuando las tablas de flujo ya están completas, pero sí es muy notable una diferencia en la pérdida de datagramas cuando las rutas no están determinadas en los dispositivos de red ya que con IPv6 la pérdida de paquetes fue el doble (26%) en comparación con IPv4 (13%).

Hay una diferencia también en pérdida de paquetes entre IPv4 e IPv6 con el uso de NOX13OFLIB ya que en las pruebas tanto con las tablas de flujo incompletas como completas es mayor la pérdida de paquetes con el uso de IPv6.

Este primer acercamiento es una de las muchas opciones que pueden ser probadas en emulaciones e incluso en dispositivos reales para iniciar con la experimentación e investigación de la utilización de IPv6 en SDN.

Estas pruebas demostraron que sí es posible trabajar con IPv6 en Redes Definidas por Software pero revelaron sólo una de las muchas opciones que este nuevo paradigma trae consigo para manipular y gestionar las redes. Esa “variedad de opciones” que se acaba de mencionar dependerá drásticamente del desarrollo de nuevas aplicaciones que están en la capa más superior de la arquitectura de las SDN llamada Capa de Aplicación que puede verse en la Figura 2.

A la fecha existen muy pocas aplicaciones para SDN que hagan uso de IPv6, inclusive no existe ninguna publicada que permita dar comportamiento de router a los dispositivos de red, lo que significa que crear tablas de enrutamiento con el protocolo IPv6 desde el controlador aún no es posible.

A pesar de que Mininet es una herramienta completa y muy útil, no tiene hasta el momento instrumentos para hacer pruebas más específicas como la inyección de diferentes tipos de tráfico, que es de vital importancia para evaluar el funcionamiento de una red y comprobar sus capacidades antes de instalarla en equipos reales.

Además de esto se evidencia que Mininet no presenta material que ayude a la emulación y pruebas de SDN e IPv6 ya que dentro de todos sus documentos sólo se realizan implementaciones con IPv4 lo que imposibilita un poco el desarrollo y el avance de esta integración.

Por otra parte, del proceso y del reconocimiento de las capacidades de los controladores expuestos y, en general, de lo ofrecido por las Redes Definidas por Software, la personalización no sólo del enrutamiento, sino de las políticas de seguridad, rendimiento, ingeniería de tráfico, etc., es realmente posible pero dependerá de la construcción de aplicaciones propias y ajustadas a las necesidades de una organización o a procesos de investigación.

Después de entender que existen elementos necesarios para soportar IPv6 en SDN como versiones específicas del protocolo que comunica la capa de infraestructura con el controlador, en el caso de OpenFlow, las versiones superiores o iguales a la 1.3, y que éste debe ser soportado por los elementos de estas dos capas, es notable que su papel en la realización de nuevas aplicaciones es fundamental para aprovechar las diferentes ventajas que esta versión de IP trae consigo.

La Tabla 7 muestra los campos concretos del protocolo IPv6 soportados por OpenFlow 1.3. El reconocimiento de estos elementos y otros, presentes en la documentación oficial de la Open Networking Foundation, hacen posible la creación de aplicaciones que innoven y generen diferentes comportamientos de una red a partir de las características nuevas ofrecidas por IPv6.

Estos elementos hacen parte de la API de OpenFlow que son necesarios para la creación de aplicaciones que sean entendidas por el controlador y que puedan dar instrucciones a los dispositivos de la red.

Tabla 7. Campos de IPv6 soportados por OpenFlow 1.3.

Campo	Bits	Bytes	Descripción
OXM_OF_IPV6_SRC	128	16	Dirección IPv6 de origen.
OXM_OF_IPV6_DST			Dirección IPv6 de destino.
OXM_OF_IPV6_FLABEL	20	4	Etiqueta de flujo de IPv6
OXM_OF_ICMPV6_TYPE	8	1	Tipo de ICMPv6

OXM_OF_ICMPV6_CODE	8	1	Código de ICMPv6
OXM_OF_IPV6_ND_TARGET	128	16	La dirección objetivo en un mensaje de descubrimiento de vecinos en IPv6.
OXM_OF_IPV6_ND_SLL	48	6	La opción de dirección de capa de enlace del origen en un mensaje de descubrimiento de vecinos en IPv6.
OXM_OF_IPV6_ND_TLL	48	6	La opción de dirección de capa de enlace de destino en un mensaje de descubrimiento de vecinos en IPv6.
OXM_OF_IPV6_EXTHDR	9	2	Pseudo-campo de cabecera de extensión IPv6.

Fuente: Open Networking Foundation. (2013). *OpenFlow Switch Specification*. Recuperado el 9 de noviembre de 2015, de <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>

6. RECOMENDACIONES Y CONCLUSIONES

A la fecha, existe muy poca información publicada relacionada específicamente con la implementación de IPv6 en Redes Definidas por Software, una característica que notablemente, además de motivar la investigación, hace que los resultados que aquí se presentaron puedan aportar al esclarecimiento de algunos interrogantes y, más importante aún, a mostrar y abrir puertas a otras oportunidades de estudio.

Las emulaciones de Redes Definidas por Software e IPv6 que se presentaron fueron realizadas con aplicaciones que los controladores como NOX13OFLIB y RYU traen por defecto, que permiten darle a los dispositivos de la capa de infraestructura un comportamiento de un switch normal, o como los nombra OpenFlow de Learning Switch. Para generar otro tipo de comportamientos en la red y personalizar otros elementos es necesaria la codificación de otras aplicaciones que soporten IPv6.

Con esta aplicación que acaba de mencionarse y utilizando IPERF para realizar algunas pruebas y análisis comparativos entre IPv4 e IPv6 se evidenció que en el momento de completar las tablas de flujo en los switch hay una diferencia en la pérdida de datagramas considerable, ya que en IPv6 se presentaron porcentajes mayores de pérdidas que con IPv4.

De igual manera es esencial considerar la integración entre el controlador y OpenFlow. Tanto el controlador, por ejemplo RYU, como OpenFlow generan documentación detallada que especifica los elementos necesarios para llevar a cabo esta integración. Es allí, en el estudio de estas especificaciones, nuevos elementos, APIs y demás, donde está la clave para hacer aplicaciones que realmente personalicen las redes para aprovechar elementos propios de un protocolo (como IPv6) o para ajustar una red a las verdaderas necesidades de un contexto.

Mininet ha sido utilizado por más de 100 investigadores en más de 18 instituciones, incluyendo la Universidad de Princeton, Berkely, Purdue, ICSI, UMass, Universidad de Alabama, NEC, NASA, Deutsche Telekom Labs, y la Universidad de Stanford. Es una herramienta al alcance de un gran número de usuarios que avanza y mejora día a día con la colaboración de investigadores de todo mundo, lo que ha permitido que actualmente sea un entorno que proporciona los elementos necesarios para crear prototipos de redes SDN, emular topologías grandes y ejecutar casi cualquier controlador externo de manera local o remota.

Por otro lado, la experimentación de tecnologías y protocolos existentes o modificaciones a los mismos sobre herramientas de simulación o emulación e

incluso en dispositivos reales en redes de nueva generación tales como redes MPLS, protocolos de enrutamiento y señalización como OSPF-TE y RSVP-TE para realizar ingeniería de tráfico en Internet, con IPv6 móvil para el soporte de movilidad IP, podrían ser trabajos futuros de interés para investigadores que trabajen alrededor del tema, permitiendo proveer soluciones versátiles y eficientes en este nuevo paradigma.

Además la posibilidad de crear aplicaciones para programar y personalizar las redes, que es lo que ofrece SDN, no es un elemento que puede ser contemplado únicamente como una herramienta adicional, es el núcleo de este nuevo paradigma y la oportunidad para que miles de desarrolladores, administradores de red, expertos en telecomunicaciones y en ingeniería de tráfico puedan transformar las redes de datos y crear un mundo de posibilidades a la mano de miles de usuarios interesados en hacer que sus redes funcionen exactamente como ellos y su entorno lo requieren.

Finalmente la realización del trabajo de grado en la modalidad de residencia en línea de investigación posibilitó aumentar las capacidades investigativas y descubrir formas de estar a la vanguardia del desarrollo de nuevas tecnologías, además permitió integrar los grandes componentes de la Ingeniería de Sistemas y Telecomunicaciones ya que las Redes Definidas por Software requieren conocimientos en Redes y al mismo tiempo en el desarrollo y codificación de aplicaciones.

LISTA DE REFERENCIAS

- Blandón, D. (2013). OpenFlow: el protocolo del futuro. *Páginas*, 1(93), 61-72.
- Bradner, S. (1993). *RFC 1550*. Recuperado el 03 de junio de 2015, de <http://tools.ietf.org/html/rfc1550>
- Braun, W., & Menth, M. (2014). Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. *Future Internet*, 6(2), 302-336.
- Casero, E., Clemente, A., & Ruiz, S. (2011). *IPv6*. Recuperado el 12 de mayo de 2015, de https://sysadmin.wibidei.com/wp-content/uploads/2011/12/PI_IPV6.pdf
- Cisco Systems, Inc. (2012). *IPv6 Configuration Guide, Cisco IOS. Release 15.2MT*. Recuperado el 2015 de mayo de 14, de <http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6/configuration/15-2mt/ipv6-15-2mt-book.pdf>
- Conta, A. (2006). *RFC 4443*. Recuperado el 20 de octubre de 2015, de <https://tools.ietf.org/pdf/rfc4443.pdf>
- CPqD. (s.f.). Obtenido de [http://www.cpqd.com.br/es/](http://www.cpqd.com.br/es/Create%20a%20Learning%20Switch)
- Create a Learning Switch. (2015). Recuperado el 29 de Septiembre de 2015, de <https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch>
- Deering, S., & Hinden, R. (1998). *RFC 2460*. Recuperado el 3 de junio de 2015, de <http://tools.ietf.org/html/rfc2460>
- Hegr, T., Bohac, L., Uhlir, V., & Chlumsky, P. (2013). OpenFlow Deployment and Concept Analysis. *Information and Communication Technologies and Services*, 327-335.
- iPerf. (s.f.). *iPerf 2 user documentation*. Recuperado el 24 de octubre de 2015, de iPerf - The network bandwidth measurement tool: <https://iperf.fr/iperf-doc.php#doc>
- Kaur, K., Singh, J., & Ghuman, N. (2014). *MiniNet as Software Defined Networking Testing Platform*. Recuperado el 08 de mayo de 2015, de <http://www.sbsstc.ac.in/icccs2014/Papers/Paper29.pdf>
- Kreutz, D., Ramos, F., Esteves, P., Esteve, C., Azodolmolky, S., & Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 113(1), 14-76.
- MiniNet. (2015). *Introduction to MiniNet*. Recuperado el 09 de mayo de 2015, de <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>
- Network Working Group. (2006). *RFC 4291: IP Version 6 Addressing Architecture*. Recuperado el 12 de mayo de 2015, de <https://tools.ietf.org/html/rfc4291>
- Newman, D. (2014). *Technology Validation Experiment: IPv6 and Multicast Support on OpenFlow*. Recuperado el 08 de mayo de 2015, de http://users.ecs.soton.ac.uk/drn/ofertie/tve_ipv6_and_multicast.pdf

- nox13oflib. (s.f.). Recuperado el 31 de mayo de 2015, de <https://github.com/CPqD/nox13oflib>
- Nunes, B. A., Mendonca, M., Nguyen, X.-N., Obraczka, K., & Turletti, T. (2014). A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *Communications Surveys and Tutorials, IEEE Communications.*, 2(4), 1617-1634.
- Open Networking Foundation. (2012). *Software-Defined Networking: The New Norm for Networks*. Recuperado el 08 de mayo de 2015, de <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>
- Open Networking Foundation. (2013). *OpenFlow Switch Specification*. Recuperado el 9 de noviembre de 2015, de <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>
- Open Networking Foundation. (2015). *Open Networking Foundation*. Recuperado el 08 de mayo de 2015, de Open Networking Foundation: <https://www.opennetworking.org/sdn-resources/sdn-definition>
- Open vSwitch. (2010). Recuperado el 27 de agosto de 2015, de <http://openvswitch.org/>
- Pal, C., Veena, S., Rustagi, R., & Murthy, k. (2014). Implementatation of Simplified Custom Topology Framework in MiniNet. En Z. Chaczko, F. Lumban, F. Pichler, & C. Chiu, *2014 Asia-Pasific Conference on Computer Aided System Engeneering (APCASE)* (págs. 48-53). Bali: IEEE.
- Park, R., & Baack, E. (2012). *Despliegue y evaluación de desempeño de una red OpenFlow*. Recuperado el 08 de 05 de 2015, de Instituto Tecnológico Autónomo de México: <http://www.lrav.itam.mx/publicacion/RepTecYus2012.pdf>
- Roa, S. (2014). *SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs*. Recuperado el 25 de agosto de 2015, de <http://thenewstack.io/sdn-series-part-iv-ryu-a-rich-featured-open-source-sdn-controller-supported-by-ntt-labs/>
- Rodrigues, L. (2013). *OpenFlow e o Paradigma de Redes Definidas por Software*. Recuperado el 08 de mayo de 2015, de http://monografias.cic.unb.br/dspace/bitstream/123456789/391/1/Monografia_Vesao_Leitura_em_PC.pdf
- Roncero, Ó. (2014). *Software Defined Networking*. Recuperado el 2015 de mayo de 08, de <http://upcommons.upc.edu/pfc/bitstream/2099.1/21633/4/Memoria.pdf>
- RYU. (25 de agosto de 2015). *RYU*. Recuperado el 25 de agosto de 2015, de <http://osrg.github.io/ryu/>
- Santos, R., Schweitzer, C., Shinoda, A., & Rodrigues, L. (2014). Using MiniNet for Emulation and Prototyping Software-Defined Networks. En Y. Rodríguez,

- 2014 *IEEE Colombian Conference on Communications and Computing (COLCOM)* (págs. 1-6). Bogotá: IEEE.
- Sezer, S., Scott, S., Chouhan, P., Fraser, B., Lake, D., Finnegan, J., . . . Rao, N. (2013). Are We Ready for SDN? Implementation Challenges for Software-Defined Networks. *IEEE Communications Magazine*, 36-43.
- Tanenbaum, A. (2003). *Redes de computadoras* (Cuarta ed.). México: Pearson Educación.
- Tennenhouse, D., & Wetherall, D. (1996). *Towards an Active Network Architecture*. Recuperado el 08 de mayo de 2015, de <http://ccr.sigcomm.org/archive/1996/apr96/ccr-9604-tennenhouse.pdf>
- Wireshark. (2014). *Wireshark User's Guide*. Recuperado el 20 de octubre de 2015, de https://www.wireshark.org/docs/wsug_html_chunked/index.html

ANEXOS

Anexo A. Comandos utilizados en Mininet		
Versión de Ubuntu	12.04	
Herramientas utilizadas	<ul style="list-style-type: none">- Terminal de Ubuntu- Mininet	
Objetivo	<ul style="list-style-type: none">- Mencionar y exponer algunos comandos usados en el emulador Mininet.	

COMANDOS UTILIZADOS EN MININET

Actualmente la herramienta Mininet tiene integrado una serie de comandos, a continuación se exponen los comandos más utilizados:

- **Help:** este comando muestra una lista con todos los comandos disponibles de Mininet.

```
mininet> help

Documented commands (type help <topic>):
=====
EOF      gterm  iperfudp  nodes      pingpair    py      switch
dpctl    help   link      noecho     pingpairfull  quit    time
dump     intf  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

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
```

- **Nodes:** Enumera los componentes que forman parte de la red.

```
mininet> nodes
available nodes are:
h1 h2 s1
mininet> 
```

- **Net:** Despliega los enlaces de la topología de la red

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0
```

- **Dump:** Aporta la información básica de cada nodo, como por ejemplo la dirección IP y el código PID.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=13837>
<Host h2: h2-eth0:10.0.0.2 pid=13839>
<OVSBridge s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=13844>
```

- **Host ifconfig:** Permite ver la configuración específica del host de la red.

```
mininet> h1 ifconfig
h1-eth0  Link encap:Ethernet  direcciónHW 00:00:00:00:00:01
        Direc. inet:10.0.0.1  Difus.:10.255.255.255  Másc:255.0.0.0
        Dirección inet6: fe80::200:ff:fe00:1/64 Alcance:Enlace
        ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
        Paquetes RX:7 errores:0 perdidos:0 overruns:0 frame:0
        Paquetes TX:9 errores:0 perdidos:0 overruns:0 carrier:0
        colisiones:0 long.colaTX:1000
        Bytes RX:558 (558.0 B)  TX bytes:738 (738.0 B)

lo       Link encap:Bucle local
        Direc. inet:127.0.0.1  Másc:255.0.0.0
        Dirección inet6: ::1/128 Alcance:Anfitrión
        ACTIVO BUCLE FUNCIONANDO  MTU:65536  Métrica:1
        Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
        Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
        colisiones:0 long.colaTX:0
        Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)
```

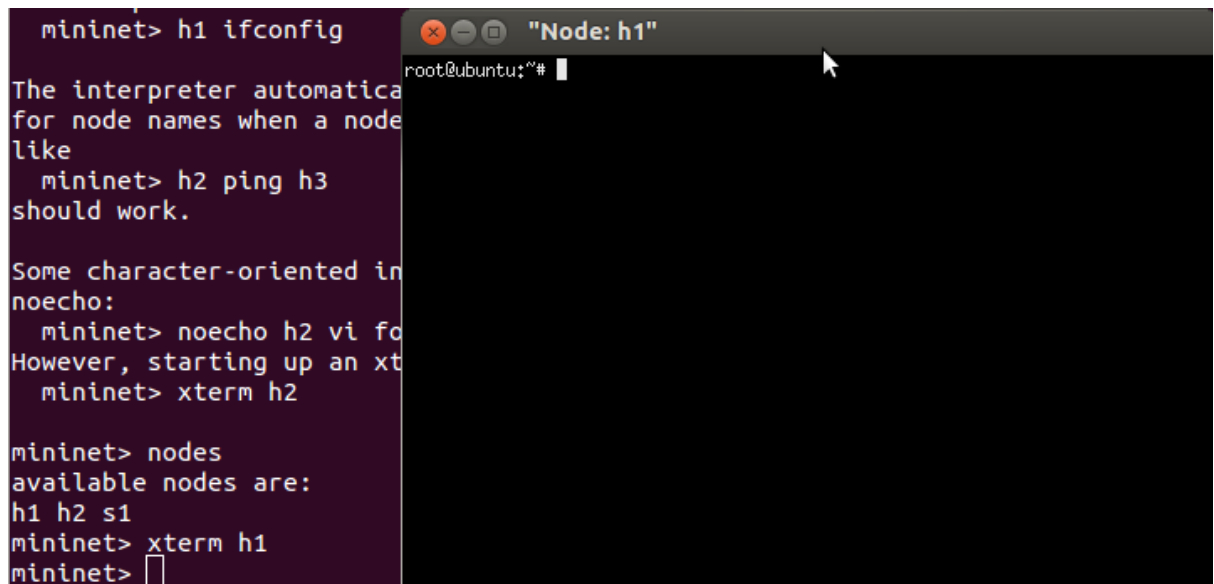
- **Ping:** Esto se utiliza para comprobar la conectividad entre los nodos asignados.


```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.359 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.140 ms
```

- **PingAll:** Esto se utiliza para comprobar la conectividad entre todos los nodos.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

- **xterm host:** Crea un terminal virtual de un host.



The image shows a terminal window with mininet commands and a new xterm terminal window titled "Node: h1" that has been opened. The mininet terminal shows the following commands and output:

```
mininet> h1 ifconfig
The interpreter automatically
for node names when a node
like
mininet> h2 ping h3
should work.

Some character-oriented in
noecho:
mininet> noecho h2 vi fo
However, starting up an xt
mininet> xterm h2

mininet> nodes
available nodes are:
h1 h2 s1
mininet> xterm h1
mininet>
```

The xterm window titled "Node: h1" shows a root prompt at root@ubuntu:~#.

- **Iperf:** Test que ejecuta un servidor iperf en un host y un cliente iperf en el segundo host, para caracterizar el ancho de banda entre ambos.

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['10.9 Gbits/sec', '11.0 Gbits/sec']
```

Anexo B. Tutorial para la instalación Ubuntu, Mininet y OpenFlow 1.3 con NOX13OFLIB.	
Versión de Ubuntu	12.04
Herramientas utilizadas	<ul style="list-style-type: none"> - Virtual Box - Ubuntu - Mininet - NOX13OFLIB - Wireshark
Objetivos	<ul style="list-style-type: none"> - Explicar de manera detallada la forma como se instalan y se ejecutan los siguientes programas (Ubuntu, Mininet, NOX13OFLIB y Wireshark.) - Configurar las herramientas con OpenFlow 1.3. - Ejecutar una topología en Mininet utilizando el controlador NOX13OFLIB.

TUTORIAL PARA LA INSTALACIÓN DE UBUNTU, MININET Y OPENFLOW 1.3 CON NOX13OFLIB

En el presente anexo se hace una explicación detallada sobre el proceso de instalación de los elementos necesarios para utilizar OpenFlow 1.3 en el emulador de Mininet a través del controlador NOX13OFLIB. Para esto es indispensable contar con el sistema operativo Ubuntu 12.04 de Linux.

INSTALACIÓN DE UBUNTU 12.04

La instalación de Mininet y NOX13OFLIB requieren de un Sistema Operativo Ubuntu, en este caso para soportar el controlador NOX13OFLIB se instala la versión 12.04. La instalación puede realizarse de 2 formas: nativa o en una máquina virtual. A continuación se mencionan los pasos necesarios para instalar Ubuntu 12.04 como máquina virtual.

Requerimientos mínimos para instalar del sistema

- Procesador 1 GHz x86 (Pentium o superior)
- 1 GB de memoria (RAM)
- 6 GB de espacio en disco
- Vídeo capaz de soportar una resolución de 1024×768
- Soporte de audio

- Una conexión a Internet

PROCESO DE DESCARGA:

El mecanismo más fácil para descargar Ubuntu 12.04 es a través de una imagen ISO, está puede descargarse de la siguiente página <http://releases.ubuntu.com/12.04/>

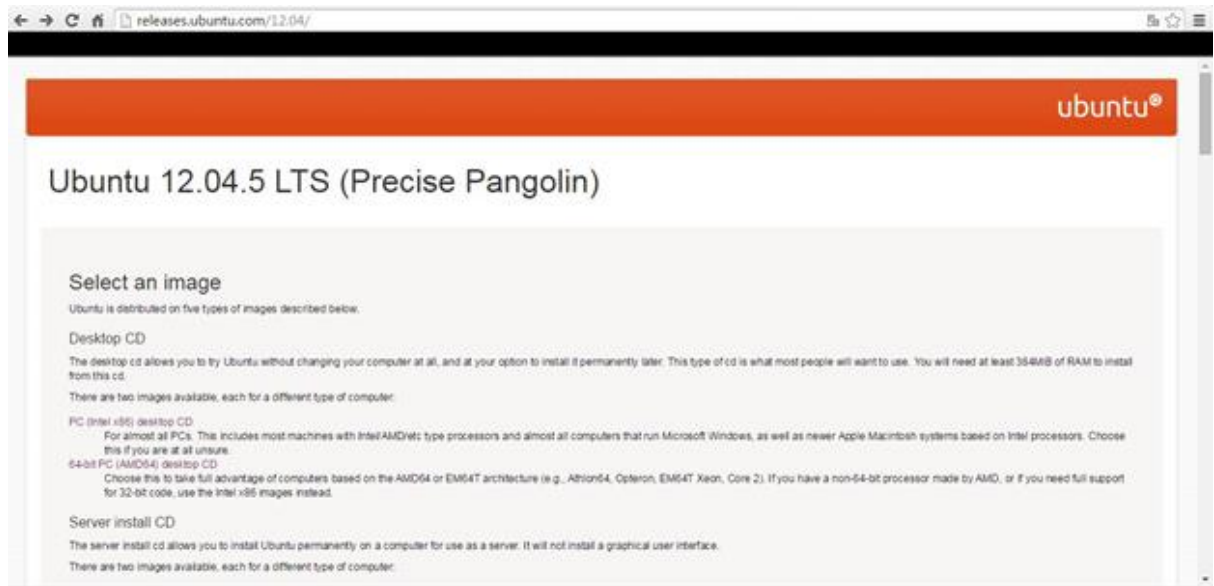


Figura 1. Descarga de Ubuntu 12.04

Una vez ingresada en la página web, el usuario debe seleccionar la opción que dice: “64-bit PC (AMD64) desktop CD o 32-bit PC (i386) desktop CD” dependiendo del tipo de sistema que su ordenador utilice.

INSTALACIÓN DE LA MÁQUINA VIRTUAL

1. Instalar un software de virtualización como VirtualBox, este permite la creación de una máquina virtual, en la página <https://www.virtualbox.org/wiki/Downloads> puede descargarse este programa seleccionando el link que se encuentra subrayado en azul en la Figura 2.



Figura 2. Descarga del VirtualBox

2. Una vez descargado el instalador se procede de la siguiente manera:
Se ejecuta el archivo descargado para iniciar la instalación. Aparecerá la ventana mostrada en la Figura 3.



Figura 3. Instalación de VirtualBox

Posteriormente el programa empezará su proceso de instalación como se puede observar en la Figura 4, en determinado punto de la instalación aparecerá otra ventana notificando que se deben aceptar ciertos permisos para así culminar con el proceso de instalación de VirtualBox.



Figura 4. Instalación del software y los permisos de dispositivo de VirtualBox

- Una vez ha finalizado la instalación del programa se ejecuta el programa instalado “VirtualBox”, después de iniciar el programa se selecciona la opción “Nueva” que se encuentra en la parte superior izquierda de la ventana, como se observa en la Figura 5.

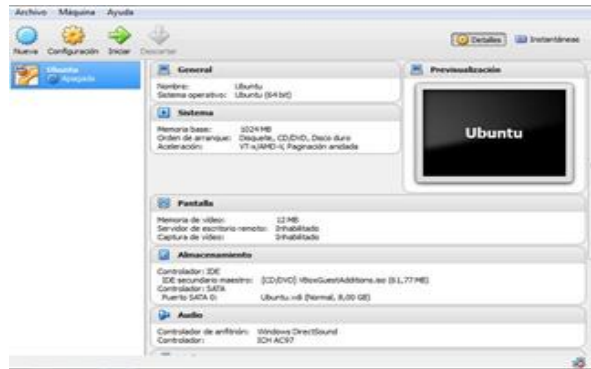


Figura 5. Ventana principal de VirtualBox

- Después de haber dado clic en dicho botón aparecerá una nueva ventana, ahí se debe ingresar un nombre, el tipo de sistema operativo a instalar y la versión, en este caso se debe seleccionar el sistema operativo de Linux e inmediatamente aparecerá por defecto la versión de Ubuntu como se observa en la figura 6.

Nota: se debe tener en cuenta que los tipos de sistema de su ordenador pueden variar.

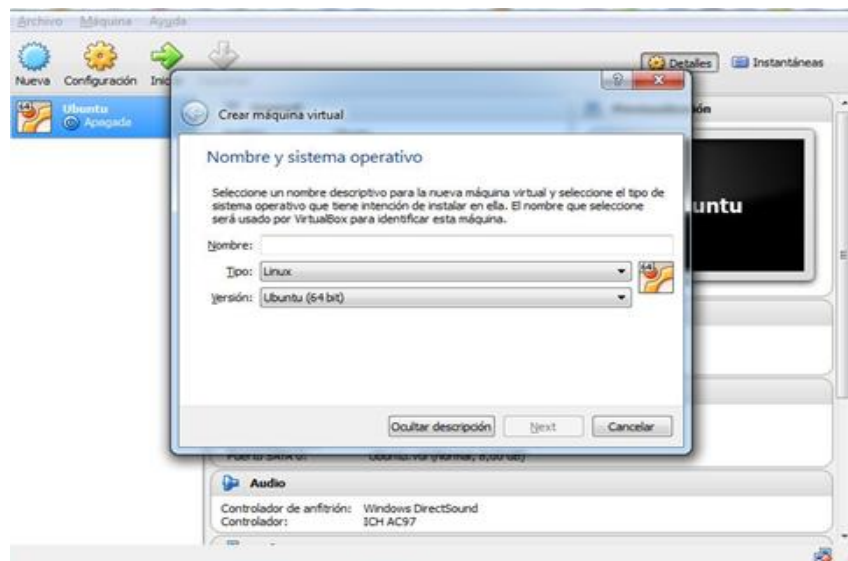


Figura 6. Creación de máquina virtual.

5. A continuación se selecciona el tipo de memoria (RAM) que va a hacer utilizada en la máquina virtual, para ello se recomienda emplear un tamaño mínimo de 1GB (1024MB). Una vez se ha seleccionado el tipo de memoria RAM y ejecutado el botón Next, aparecerá una ventana nueva, en ella se debe seleccionar el tipo de sistema, a su vez se debe elegir la opción VDI, posteriormente se asigna el tipo de almacenamiento del disco ya sea de manera estática o dinámica.
6. Una vez creada la partición se procede a la instalación del sistema operativo, para ello se debe ejecutar el icono que se ha creado durante el proceso de instalación, este se ubica en la parte izquierda del programa VirtualBox. Al ejecutar el icono debe aparecer una nueva ventana notificando el ingreso de la imagen ISO descargada anteriormente, una vez a finalizado el proceso, se procede a la selección del idioma, luego se da clic en el botón “siguiente”, en determinado punto de la instalación se pedirá al usuario que ingrese un nombre y una contraseña, para así concluir con el proceso de instalación del sistema operativo.
7. Una vez ha finalizado el proceso de instalación, se debe reiniciar el sistema operativo instalado en la máquina virtual, de esta forma se verá el entorno de Ubuntu. Una vez que ejecutado el sistema operativo, se debe proceder a instalación del complemento **git**. Para realizar este procedimiento se debe seleccionar el botón que se encuentra localizado en la parte superior izquierda de la pantalla, se da clic en el icono de (Ubuntu), en la opción de buscar se debe escribir la palabra “terminal”, como se refleja en la Figura 8.



Figura 8. Búsqueda del terminal

8. Después de ejecutar el terminal de Ubuntu, aparecerá una nueva ventana, es ahí donde se realiza la instalación del complemento **git**, para ello se debe escribir el siguiente comando: **sudo apt-get install git**. Una vez ha culminado el proceso de instalación se procede con la ejecución de los siguientes comandos como se observa en la figura 9.

```
bryan@bryan:~$ sudo apt-get install git
[sudo] password for bryan:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  git-man liberror-perl
Paquetes sugeridos:
  git-daemon-run git-daemon-sysvinit git-doc git-el git-arch git-cvs git-svn
  git-email git-gui gitk gitweb
Se instalarán los siguientes paquetes NUEVOS:
  git git-man liberror-perl
0 actualizados, 3 se instalarán, 0 para eliminar y 276 no actualizados.
Necesito descargar 6.751 kB de archivos.
Se utilizarán 15,2 MB de espacio de disco adicional después de esta operación.
¿Desea continuar [S/n]? S
Des:1 http://co.archive.ubuntu.com/ubuntu/ precise/main liberror-perl all 0.17-
[23,8 kB]
Des:2 http://co.archive.ubuntu.com/ubuntu/ precise-updates/main git-man all 1:1
7.9.5-1ubuntu0.1 [631 kB]
Des:3 http://co.archive.ubuntu.com/ubuntu/ precise-updates/main git amd64 1:1.7
9.5-1ubuntu0.1 [6.097 kB]
Descargados 6.751 kB en 1min. 29seg. (75,3 kB/s)
Seleccionando el paquete liberror-perl previamente no seleccionado.
```

Figura 9. Instalación del complemento git del terminal

PROCESO DE INSTALACIÓN DE MININET Y NOX13OFLIB PARA EL SOPORTE DE OPENFLOW 1.3

El mecanismo más fácil para descargar Mininet y los paquetes necesarios para instalar OpenFlow 1.3, se describen en la siguiente página <https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow-1.3-Tutorial> como se observa en la figura 10.



Figura 10. Página oficial para descargar Mininet y OpenFlow 1.3

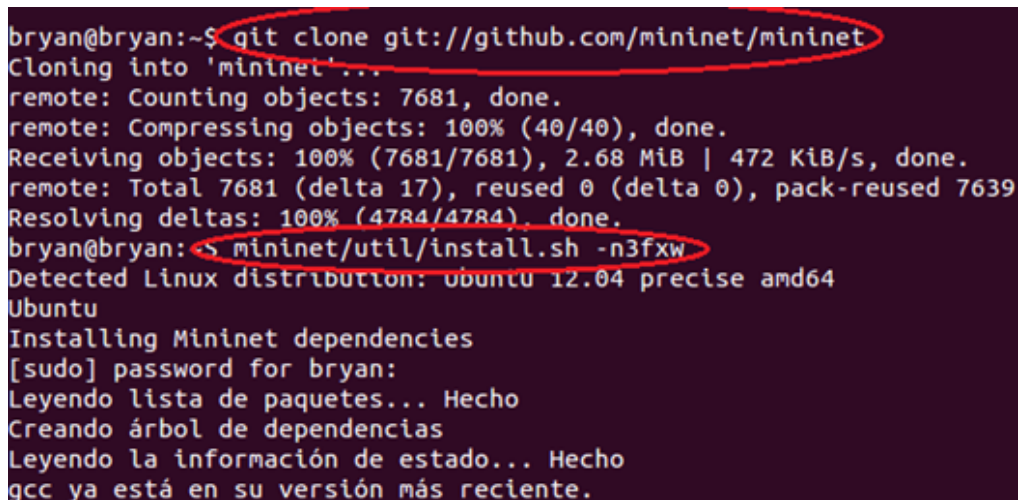
La instalación de Mininet y OpenFlow 1.3 se realiza de la siguiente manera:

PROCESO DE INSTALACIÓN

- Abrir el “terminal”
- Con el siguiente comando se instalan los siguientes elementos: Mininet, Switch del software, Nox13oflib y Wireshark.

A continuación se explica el procedimiento necesario para instalar los paquetes necesarios de OpenFlow 1.3, como se observa en la figura 11.

```
cd $HOME/  
git clone git://github.com/Mininet/Mininet  
Mininet/util/install.sh -n3fxw
```



```
bryan@bryan:~$ git clone git://github.com/mininet/mininet  
Cloning into 'mininet'...  
remote: Counting objects: 7681, done.  
remote: Compressing objects: 100% (40/40), done.  
Receiving objects: 100% (7681/7681), 2.68 MiB | 472 KiB/s, done.  
remote: Total 7681 (delta 17), reused 0 (delta 0), pack-reused 7639  
Resolving deltas: 100% (4784/4784), done.  
bryan@bryan:~$ Mininet/util/install.sh -n3fxw  
Detected Linux distribution: ubuntu 12.04 precise amd64  
Ubuntu  
Installing Mininet dependencies  
[sudo] password for bryan:  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
gcc ya está en su versión más reciente.
```

Figura 11. Pasos para instalar el paquete completo de OpenFlow 1.3

- Una vez culminado todo el proceso de instalación, se procede a instalar el plugin de OpenFlow 1.3 en Wireshark, para ello se debe ejecutar el siguiente comando como se observa en la Figura 12:

```
sudo apt-get install scons  
git clone https://github.com/CPqD/ofdissector  
cd ofdissector  
cd src  
export WIRESHARK=/usr/include/wireshark  
scons install
```

```

bryan@bryan:~$ git clone https://github.com/CPqD/ofdissector
Cloning into 'ofdissector'...
remote: Counting objects: 273, done.
remote: Total 273 (delta 0), reused 0 (delta 0), pack-reused 273
Receiving objects: 100% (273/273), 161.41 KiB, done.
Resolving deltas: 100% (137/137), done.
bryan@bryan:~$ ls
Descargas  Escritorio  Imágenes  mininet  nbeesrc-jan-10-2013  nox13ofl1b  ofsoftswitch13  Público
Documents  examples.desktop  loxigen  Música  nbeesrc-jan-10-2013.zip  ofdissector  Plantillas  Vídeos
bryan@bryan:~$ cd ofdissector
bryan@bryan:~/ofdissector$ ls
ACKS  conf  COPYING  doc  gen  README.md  src  test
bryan@bryan:~/ofdissector$ cd src
bryan@bryan:~/ofdissector/src$ export WIRESHARK=/usr/include/wireshark
bryan@bryan:~/ofdissector/src$ scons install

```

Figura 12. Plugin de Wireshark

- Una vez efectuado el comando **scons install**, el sistema comenzará el proceso de instalación, como se observa en la Figura 13, en cierta parte del proceso aparecerá un error.

```

root@santiago-VirtualBox: /home/ofdissector/src
La orden «cd» del paquete «lrpas» (multiverse)
cd.: no se encontró la orden
root@santiago-VirtualBox:~# cd ..
root@santiago-VirtualBox:~/ofdissector$ git clone https://github.com/CPqD/ofdissector
Clonar en «ofdissector»...
remote: Counting objects: 273, done.
remote: Total 273 (delta 0), reused 0 (delta 0), pack-reused 273
Receiving objects: 100% (273/273), 161.41 KiB | 152.00 KiB/s, done.
Resolving deltas: 100% (137/137), done.
Checking connectivity... hecho.
root@santiago-VirtualBox:~/ofdissector$ cd src
bash: cd: src: No existe el archivo o el directorio
root@santiago-VirtualBox:~/ofdissector$ cd Documents
bash: cd: Documents: No existe el archivo o el directorio
root@santiago-VirtualBox:~/ofdissector$ ls
ofdissector  santiago
root@santiago-VirtualBox:~/ofdissector$ cd ofdissector
root@santiago-VirtualBox:~/ofdissector$ ls
ACKS  conf  COPYING  doc  gen  README.md  src  test
root@santiago-VirtualBox:~/ofdissector$ cd src
root@santiago-VirtualBox:~/ofdissector/src$ export WIRESHARK=/usr/include/wireshark
root@santiago-VirtualBox:~/ofdissector/src$ scons install
scons: Reading SConscript files ...
Package glib-2.0 was not found in the pkg-config search path.
Perhaps you should add the directory containing 'glib-2.0.pc'
to the PKG_CONFIG_PATH environment variable
No package 'glib-2.0' found
OSError: 'pkg-config --cflags --libs glib-2.0' exited 1:
File "/home/ofdissector/src/Sconstruct", line 40:
env.ParseConfig('pkg-config --cflags --libs glib-2.0')
File "/usr/lib/scons/SCons/Environment.py", line 1554:
return Function(self, self.backtick(command))
File "/usr/lib/scons/SCons/Environment.py", line 596:
raise OSError("'%s' exited %d" % (command, status))
root@santiago-VirtualBox:~/ofdissector/src$

```

Figura 13. Instalación de scons y error.

- Una posible solución para resolver este inconveniente es salirse de **ofdissector/src** a la raíz (Home) y una vez estando ahí se procede a la ejecución del siguiente comando logrando así la instalación de la librería correspondiente: **sudo apt-get install libglib2.0-dev**, como se observa en la figura 14.

```

bryan@bryan:~/ofdissector/src$ cd /
bryan@bryan:/$ sudo apt-get install libglib2.0-dev
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Paquetes sugeridos:
  libglib2.0-doc
Se instalarán los siguientes paquetes NUEVOS:
  libglib2.0-dev
0 actualizados, 1 se instalarán, 0 para eliminar y 265 no actualizados.
Necesito descargar 1.812 kB de archivos.
Se utilizarán 8.662 kB de espacio de disco adicional después de esta operación.
Des:1 http://co.archive.ubuntu.com/ubuntu/ precise-updates/main libglib2.0-dev amd64 2.32.4-0ubuntu1 [1.812 kB]
7% [1 libglib2.0-dev 119 kB/1.812 kB 7%]

```

Figura 14. Solución al problema del scon.

- Una vez ha culminado la instalación del paso anterior, se prosigue instalando los plugins necesarios, para ello se debe ejecutar los siguientes pasos, como se observa en la Figura 15.1:

cd home

ls

cd bryan (se debe ingresar el nombre de la Carpeta personal)

cd ofdissector

cd src

export WIRESHARK=/usr/include/wireshark

scons install

```

Configurando libglib2.0-dev (2.32.4-0ubuntu1) ...
bryan@bryan:/$ cd home/
bryan@bryan:/home$ ls
bryan
bryan@bryan:/home$ cd bryan
bryan@bryan:~$ ls
Descargas  Escritorio  Imágenes  mininet  nbeesrc-jan-10-2013  nox13oflib  ofsoftswitch13  Público
Documentos examples.desktop loxigen  Música  nbeesrc-jan-10-2013.zip  ofdissector  Plantillas  Vídeos
bryan@bryan:~$ cd ofdissector
bryan@bryan:~/ofdissector$ cd src
bryan@bryan:~/ofdissector/src$ export WIRESHARK=/usr/include/wireshark
bryan@bryan:~/ofdissector/src$ scons install

```

Figura 15-1. Continuación del plugin de Wireshark

Nota: Al finalizar este tutorial, el usuario ya podrá disponer de todas las herramientas necesarias para la utilización de OpenFlow 1.3.

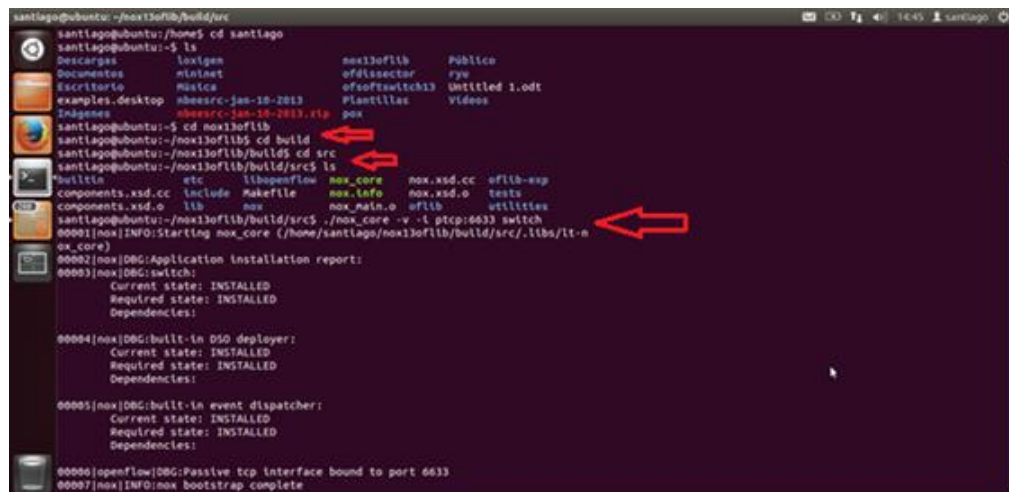
PASOS PARA EJECUTAR UNA TOPOLOGÍA CON NOX13OFLIB

Los pasos necesarios para ejecutar una topología en Mininet es la siguiente:

1. Abrir un terminal
2. Ejecutar el controlador **NOX13OFLIB**, para ello se debe ejecutar el siguiente comando estando en el directorio home:

```
cd nox13oflib
ls
cd build
ls
cd src
./nox_core -v -i ptcp:6633 switch
```

- Una vez ejecutado el último comando, aparecerá una serie de mensajes notificando la activación del controlador **NOX13OFLIB**, como se observa en la Figura 17.



```
santiago@santago: ~/nox13oflib/build/src
santiago@santago:~/home$ cd santago
santiago@santago:~$ ls
Descargas  loxigen  nox13oflib  Publico
Documents  mininet  ofdissector  ryu
Iscritorio  minica  ofsoftswitch13  Untitled 1.odt
examples.desktop  nbsrc-jan-10-2013  Plantillas  Videos
Inágenes  nbsrc-jan-10-2013.rtf  pox

santiago@santago:~$ cd nox13oflib
santiago@santago:~/nox13oflib$ cd build
santiago@santago:~/nox13oflib/build$ cd src
santiago@santago:~/nox13oflib/build/src$ ls
build  etc  libopenflow  nox_core  nox.xsd.cc  oflib-exp
components.xsd.cc  include  Makefile  nox.info  nox.xsd.o  tests
components.xsd.o  lib  nox  nox_main.o  oflib  utilities

santiago@santago:~/nox13oflib/build/src$ ./nox_core -v -i ptcp:6633 switch
00001[nox]INFO:Starting nox_core (/home/santiago/nox13oflib/build/src/.libs/lt-n
ox_core)
00002[nox]DBG:Application Installation report:
00003[nox]DBG:switch:
Current state: INSTALLED
Required state: INSTALLED
Dependencies:

00004[nox]DBG:built-in DSO deployer:
Current state: INSTALLED
Required state: INSTALLED
Dependencies:

00005[nox]DBG:built-in event dispatcher:
Current state: INSTALLED
Required state: INSTALLED
Dependencies:

00006openFlow[DBG:Passive tcp interface bound to port 6633
00007[nox]INFO:nx bootstrap complete
```

Figura 17. Pasos para activar el controlador NOX13OFLIB.

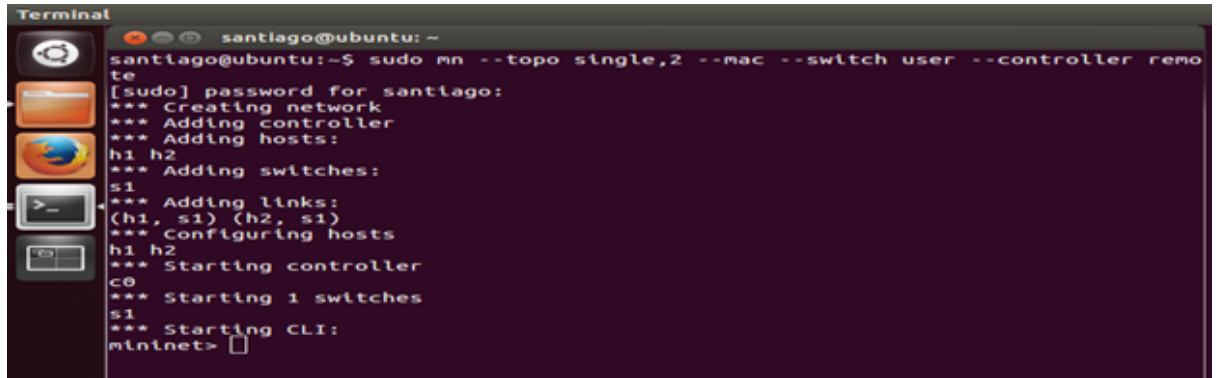
- Después de haber activado el controlador **NOX13OFLIB**, el usuario podrá ejecutar diferentes topologías de Mininet.

TOPOLOGÍAS DE MININET

El emulador de Mininet cuenta con diferentes topologías, a continuación se mencionan algunas de ellas.

Topología Single (Figura 18.1)

`sudo mn --topo single,2 --mac --switch user --controller remote`

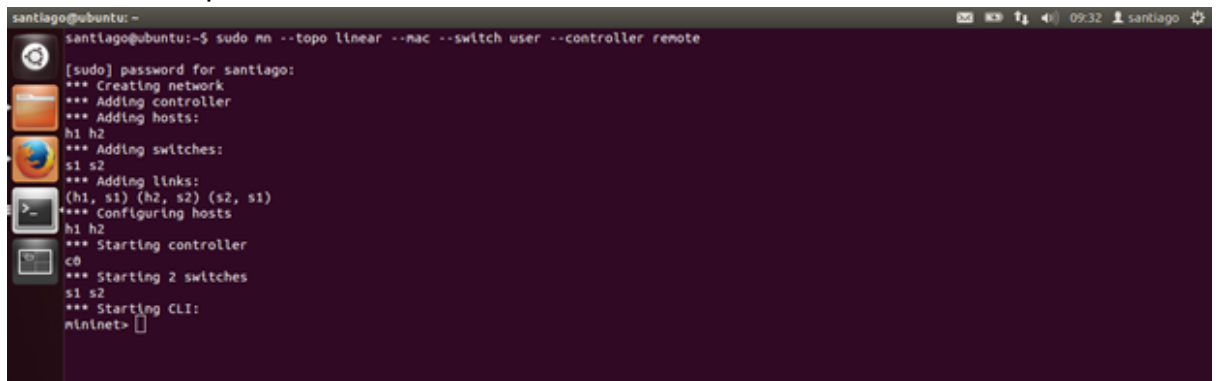


```
santiago@ubuntu: ~  
santiago@ubuntu:~$ sudo mn --topo single,2 --mac --switch user --controller remote  
[sudo] password for santiago:  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2  
*** Adding switches:  
s1  
*** Adding links:  
(h1, s1) (h2, s1)  
*** Configuring hosts  
h1 h2  
*** Starting controller  
c0  
*** Starting 1 switches  
s1  
*** Starting CLI:  
mininet>
```

Figura 18.1. Topología single de Mininet

Topología linear (Figura 18.2)

`sudo mn --topo linear --mac --switch user --controller remote`




```
santiago@ubuntu: ~  
santiago@ubuntu:~$ sudo mn --topo linear --mac --switch user --controller remote  
[sudo] password for santiago:  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2  
*** Adding switches:  
s1 s2  
*** Adding links:  
(h1, s1) (h2, s2) (s2, s1)  
*** Configuring hosts  
h1 h2  
*** Starting controller  
c0  
*** Starting 2 switches  
s1 s2  
*** Starting CLI:  
mininet>
```

Figura 18.2. Topología linear de Mininet

Nota: Cada vez que se vaya a ingresar una topología debe cerciorarse que el controlador NOX13OFLIB esté activo, además es indispensable poner controller remote como se pudo observar en las dos topologías vistas anteriormente.

- Inmediatamente se ha ejecutado la topología de Mininet, el usuario podrá realizar diferentes pruebas, como por ejemplo: la prueba de conectividad entre

los host a través del comando ping como se observa en la figura 19 conectividad.



```
Terminal
santiago@ubuntu: ~
santiago@ubuntu:~$ sudo mn --topo single,2 --mac --switch user --controller remote
[sudo] password for santiago:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=2.18 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=2.12 ms
```

Figura 19. Conectividad haciendo ping.

- Una vez ejecutada la topología y probada la conectividad, el usuario puede verificar el tráfico emitido a través de la herramienta Wireshark, para ello basta con abrir el programa y seleccionar la interfaz **lo** como se observa en la Figura 20. Posteriormente aparecerá una serie de paquetes, notificando que realmente se está trabajando bajo el protocolo de Openflow 1.3.

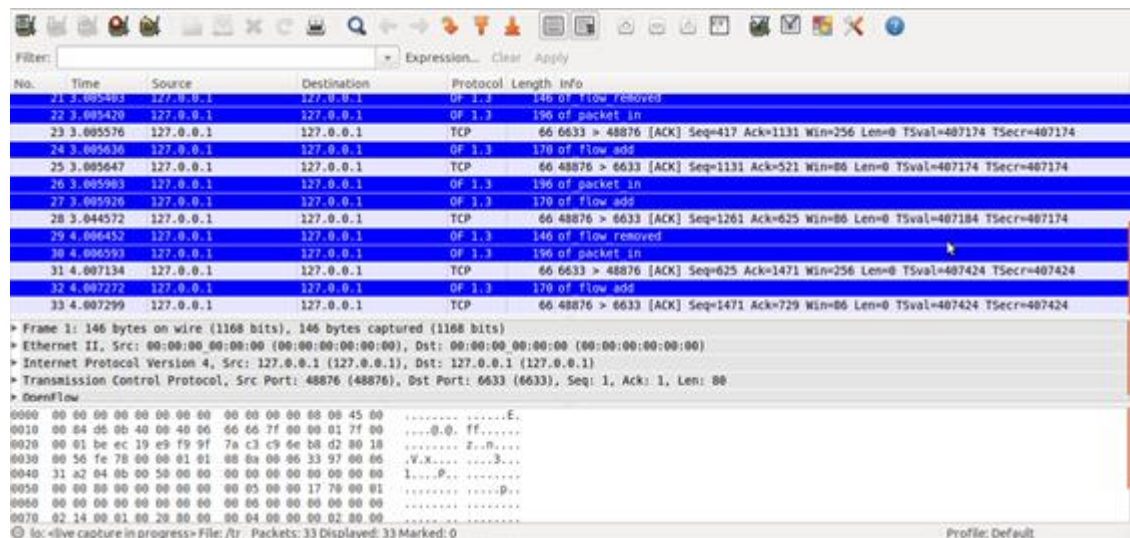


Figura 20. Verificación OpenFlow 1.3

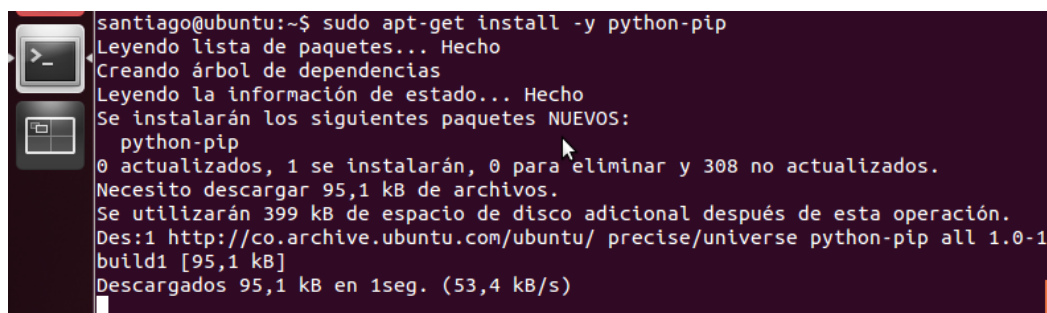
Anexo C. Instalación del controlador RYU		
Versión de Ubuntu	12.04	
Herramientas utilizadas	- Terminal	
Objetivo	- Realizar una explicación detallada de la forma como se instala el controlador RYU en Ubuntu.	

INSTALACIÓN DEL CONTROLADOR RYU

En el presente anexo se hace una explicación sobre el proceso de instalación de los elementos necesarios para utilizar el controlador RYU, a continuación se explican los pasos necesarios para la ejecución del presente controlador.

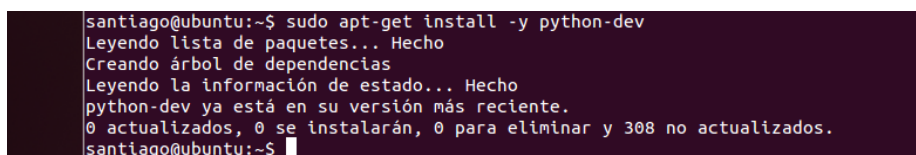
NOTA: EL PRESENTE TUTORIAL HA SIDO INSTALADO EN LA DISTRIBUCIÓN DE UBUNTU 12.04, POR LO TANTO SE GARANTIZA EL CORRECTO FUNCIONAMIENTO EN DICHA DISTRIBUCIÓN.

1. Abrir el terminal
2. Ejecutar el comando **sudo apt-get install -y python-pip**.



```
santiago@ubuntu:~$ sudo apt-get install -y python-pip
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
python-pip
0 actualizados, 1 se instalarán, 0 para eliminar y 308 no actualizados.
Necesito descargar 95,1 kB de archivos.
Se utilizarán 399 kB de espacio de disco adicional después de esta operación.
Des:1 http://co.archive.ubuntu.com/ubuntu/ precise/universe python-pip all 1.0-1
build1 [95,1 kB]
Descargados 95,1 kB en 1seg. (53,4 kB/s)
```

3. Ejecutar el comando **sudo apt-get install -y python-dev**.



```
santiago@ubuntu:~$ sudo apt-get install -y python-dev
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
python-dev ya está en su versión más reciente.
0 actualizados, 0 se instalarán, 0 para eliminar y 308 no actualizados.
santiago@ubuntu:~$
```

4. Ejecutar el comando **sudo apt-get install -y python-repoze.lru**.


```
santiago@ubuntu:~$ sudo apt-get install -y python-repoze.lru
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  python-repoze.lru
0 actualizados, 1 se instalarán, 0 para eliminar y 308 no actualizados.
```

5. Ejecutar el comando **sudo apt-get install -y python-ecdsa**.

```
santiago@ubuntu:~$ sudo apt-get install -y python-ecdsa
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
E: No se ha podido localizar el paquete python-ecdsa
```

- Una vez se ha ejecutado el comando aparecerá un error notificando que no se ha podido localizar el paquete **edcsa**, para solucionar dicho problema, es necesario ingresar los siguientes comandos.

```
sudo apt-get install python-setuptools
sudo easy_install pip
sudo pip install ecdsa
```

```
santiago@ubuntu:~$
Obj http://co.archive.ubuntu.com precise-updates/universe Translation-es
Obj http://co.archive.ubuntu.com precise-updates/universe Translation-en
Obj http://co.archive.ubuntu.com precise-backports/main Translation-en
Obj http://co.archive.ubuntu.com precise-backports/multiverse Translation-en
Obj http://co.archive.ubuntu.com precise-backports/restricted Translation-en
Obj http://co.archive.ubuntu.com precise-backports/universe Translation-en
Descargados 5.058 kB en 14seg. (338 kB/s)
Leyendo lista de paquetes... Hecho
santiago@ubuntu:~$ sudo apt-get install python-setuptools
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
python-setuptools ya está en su versión más reciente.
0 actualizados, 0 se instalarán, 0 para eliminar y 310 no actualizados.
santiago@ubuntu:~$ sudo easy_install pip
Searching for pip
Best match: pip 1.0
Adding pip 1.0 to easy-install.pth file
Installing pip script to /usr/local/bin

Using /usr/lib/python2.7/dist-packages
Processing dependencies for pip
Finished processing dependencies for pip
santiago@ubuntu:~$

santiago@ubuntu:~$ sudo pip install ecdsa
Downloading/unpacking ecdsa
  Downloading ecdsa-0.13.tar.gz (55Kb): 55Kb downloaded
  Running setup.py egg_info for package ecdsa

Installing collected packages: ecdsa
  Running setup.py install for ecdsa
```

- Posteriormente, se procede a descargar el controlador RYU, para ello es necesario ejecutar el siguiente comando.

```
git clone git://github.com/osrg/ryu.git  
cd ryu  
sudo python ./setup.py install  
cd ..
```

```
santiago@ubuntu:~$ git clone git://github.com/osrg/ryu.git  
Cloning into 'ryu'...  
remote: Counting objects: 19773, done.  
remote: Compressing objects: 100% (125/125), done.  
remote: Total 19773 (delta 97), reused 0 (delta 0), pack-reused 19648  
Receiving objects: 100% (19773/19773), 20.06 MiB | 467 KiB/s, done.  
Resolving deltas: 100% (14677/14677), done.  
santiago@ubuntu:~$ cd ryu  
santiago@ubuntu:~/ryu$ sudo python ./setup.py install  
  
Installed /home/santiago/ryu/pbr-0.11.0-py2.7.egg  
running install  
Downloading/unpacking eventlet>=0.15
```

Anexo D. Utilización de aplicación para el controlador RYU que da comportamiento de routers a los switches.	
Versión de Ubuntu	12.04
Herramientas utilizadas	<ul style="list-style-type: none"> - Mininet - RYU
Objetivo	<ul style="list-style-type: none"> - Dar comportamiento de routers a los switch de una Red Definida por Software emulada con el controlador RYU a través de una aplicación codificada en Python.

UTILIZACIÓN DE APLICACIÓN PARA EL CONTROLADOR RYU QUE DA COMPORTAMIENTO DE ROUTERS A LOS SWITCHES.

En el presente anexo se hace una explicación detallada sobre el proceso de configuración de routers utilizando el emulador Mininet, a continuación se muestra los pasos necesarios para la creación y la ejecución de una topología lineal como se observa en la figura 1, cabe resaltar que dicha ejecución está conformada por los siguientes elementos: 3 switch y 3 host. Además se deberá configurar las rutas o las direcciones de cada host y switch, a su vez se verificarán la comunicación.

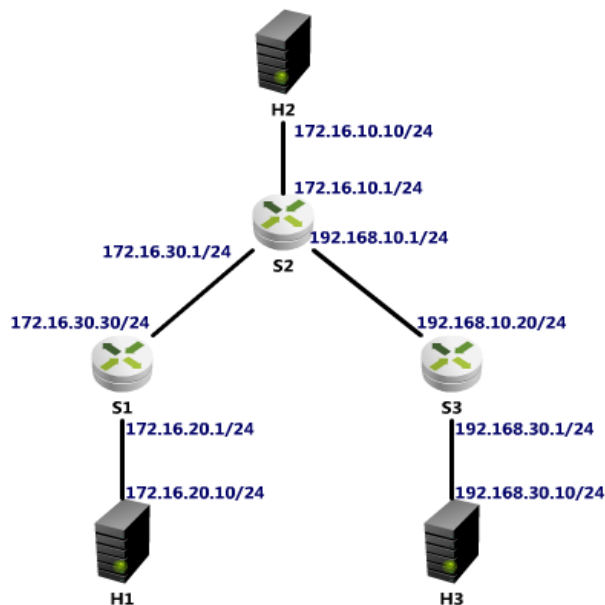


Figura 1. Visualización de la topología a ejecutar con sus respectivas direcciones.

1. Abrir el terminal.
2. Ejecutar el comando **sudo service openvswitch-switch restart**, para iniciar los servicios de Open vSwitch como se observa en la figura 2.

```
santiago@ubuntu: ~  
santiago@ubuntu:~$ sudo service openvswitch-switch restart  
[sudo] password for santiago:  
* Killing ovs-vswitchd (1111)  
* Killing ovssdb-server (1096)  
* Starting ovssdb-server  
* Configuring Open vSwitch system IDs  
* Starting ovs-vswitchd  
* Enabling remote OVSDb managers  
santiago@ubuntu:~$
```

Figura 2. Inicialización de los servicios de Open vSwitch

3. Después de haber inicializado el servicio de Openvswitch, se procede a la ejecución de la topología como se observa en la figura 3, para ello basta ejecutar el siguiente comando.

sudo mn --topo linear,3 --mac --switch ovsk --controller remote.

```
santiago@ubuntu:~$ sudo mn --topo linear,3 --mac --switch ovsk --controller remote  
te  
*** Creating network  
*** Adding controller  
Unable to contact the remote controller at 127.0.0.1:6633  
*** Adding hosts:  
h1 h2 h3  
*** Adding switches:  
s1 s2 s3  
*** Adding links:  
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)  
*** Configuring hosts  
h1 h2 h3  
*** Starting controller  
c0  
*** Starting 3 switches  
s1 s2 s3 ...  
*** Starting CLI:  
mininet>
```

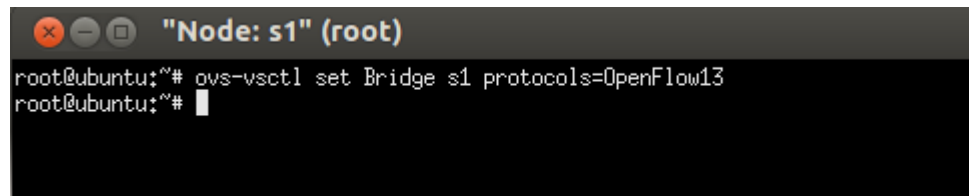
Figura 3. Ejecución de la topología lineal

4. Una vez se ha ejecutado la topología, se procede a la configuración del protocolo OpenFlow 1.3 en cada Switch, para ello se ingresa al **xterm** de cada Switch y se ejecuta las siguientes líneas.

Para ingresar el **xterm** de cada switch basta con ingresar el siguiente comando:

Mininet> xterm s1 (switch al que desea ingresar en este caso ingresamos al switch 1)

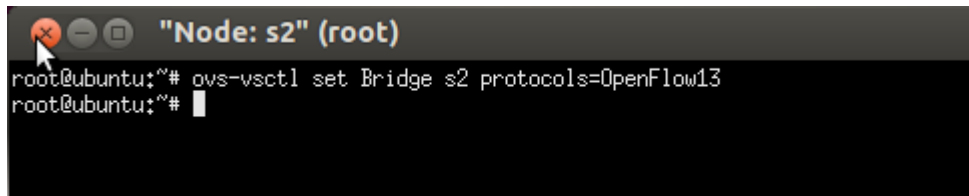
ovs-vsctl set Bridge s1 protocols=OpenFlow13



```
"Node: s1" (root)
root@ubuntu:~# ovs-vsctl set Bridge s1 protocols=OpenFlow13
root@ubuntu:~#
```

Figura 4. Configuración del protocolo OpenFlow 1.3

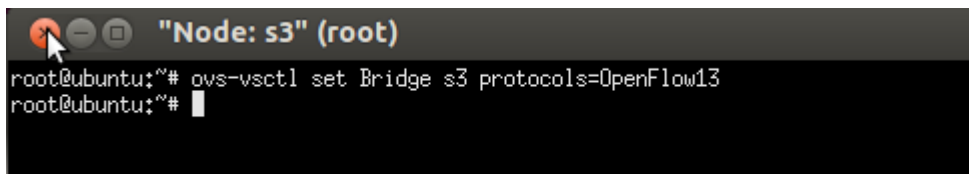
ovs-vsctl set Bridge s2 protocols=OpenFlow13



```
"Node: s2" (root)
root@ubuntu:~# ovs-vsctl set Bridge s2 protocols=OpenFlow13
root@ubuntu:~#
```

Figura 4.1. Configuración del protocolo OpenFlow 1.3

ovs-vsctl set Bridge s3 protocols=OpenFlow13

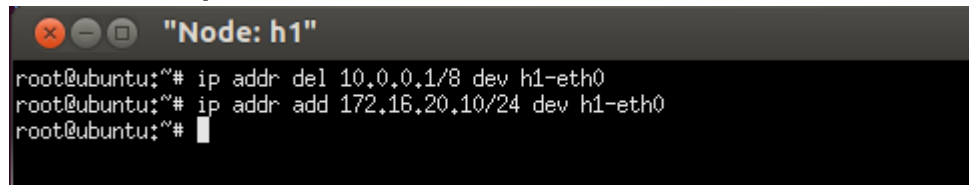


```
"Node: s3" (root)
root@ubuntu:~# ovs-vsctl set Bridge s3 protocols=OpenFlow13
root@ubuntu:~#
```

Figura 4.3. Configuración del protocolo OpenFlow 1.3

- Una vez se ha configurado el protocolo en los switches (1,2,3), se procede a la eliminación de las direcciones IP que vienen por defecto en cada Host, para ello se debe ingresar al **xterm** de cada host y se debe ejecutar el siguiente comando: **ip addr del <DirecciónIP> dev Host-eth0**; Una vez que ha concluido el proceso en cada Host, se procede a la asignación de las nuevas direcciones IP, para ello se debe escribir el siguiente comando: **ip addr add DirecciónIP dev Host-eth0**. En la figura 5 se muestra todo el proceso de eliminación y asignación de las direcciones IP en cada host.

- Host1 (H1) **ip addr del 10.0.0.1/8 dev h1-eth0**
ip addr add 172.16.20.10/24 dev h1-eth0



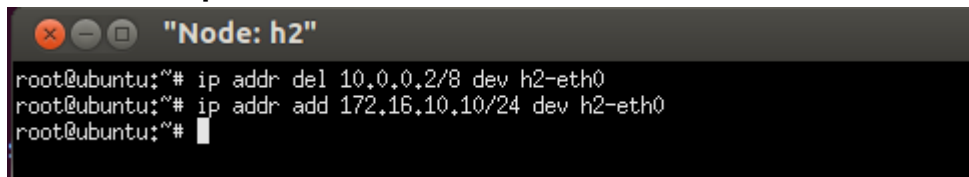
```

"Node: h1"
root@ubuntu:~# ip addr del 10.0.0.1/8 dev h1-eth0
root@ubuntu:~# ip addr add 172.16.20.10/24 dev h1-eth0
root@ubuntu:~#

```

Figura 5.1. Eliminación y asignación de la IP en el host1

- Host2 (H2) **ip addr del 10.0.0.2/8 dev h2-eth0**
ip addr add 172.16.10.10/24 dev h2-eth0



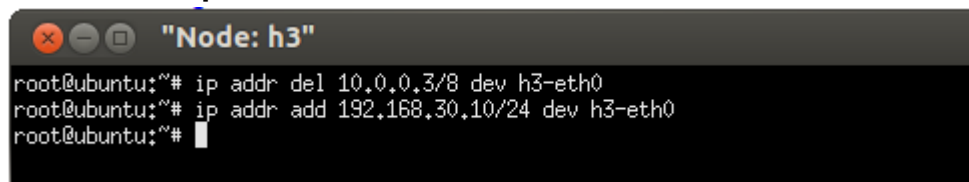
```

"Node: h2"
root@ubuntu:~# ip addr del 10.0.0.2/8 dev h2-eth0
root@ubuntu:~# ip addr add 172.16.10.10/24 dev h2-eth0
root@ubuntu:~#

```

Figura 5.2. Eliminación y asignación de la IP en el host2

- Host3 (H3) **ip addr del 10.0.0.3/8 dev h3-eth0**
ip addr add 192.168.30.10/24 dev h3-eth0



```

"Node: h3"
root@ubuntu:~# ip addr del 10.0.0.3/8 dev h3-eth0
root@ubuntu:~# ip addr add 192.168.30.10/24 dev h3-eth0
root@ubuntu:~#

```

Figura 5.3. Eliminación y asignación de la IP en el host3

- Después de haber asignado las direcciones IP en cada uno de los host, se procede a la ejecución del controlador RYU, para ello ingresamos al **xterm** C0 del controlador y ejecutamos el controlador RYU, el comando que se utiliza es: **xterm c0**, una vez ha ingresado al **xterm** se debe ejecutar el siguiente comando como se observa en la en la figura 6.

```

cd ryu
./bin/ryu-manager --verbose ryu/app/rest_router.py

```

```

root@ubuntu:~# cd ryu
root@ubuntu:~/ryu# ./bin/ryu-manager --verbose ryu/app/rest_router.py
loading app ryu/app/rest_router.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu/app/rest_router.py of RestRouterAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK dpset
  PROVIDES EventDP TO {'RestRouterAPI': set(['dpset'])}
  CONSUMES EventOFPStateChange
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPortStatus
BRICK RestRouterAPI
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPStatsReply
  CONSUMES EventOFPPFlowStatsReply
  CONSUMES EventDP
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'RestRouterAPI': set(['main'])}
  PROVIDES EventOFPFlowStatsReply TO {'RestRouterAPI': set(['main'])}

```

Figura 6. Ejecución del controlador RYU

- Una vez inicializado el controlador RYU, se procede a la elaboración de las direcciones de cada router, para ello es indispensable ingresar a un **xterm** nuevo del c0 (controlador) para ejecutar los siguientes comandos como se observa en la figura 7.

```

root@ubuntu:~#
root@ubuntu:~# set -x
root@ubuntu:~#
root@ubuntu:~# curl -X POST -d '{"address":"172.16.20.1/24"}' http://localhost:
8080/router/000000000000000001 | python -mjson.tool
+ curl -X POST -d '{"address":"172.16.20.1/24"}' http://localhost:8080/router/00
000000000000000001
+ python -mjson.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100  147  100   119  100    28   5030   1183  --:--:-- --:--:-- --:--:--   7000
[
  {
    "command_result": [
      {
        "details": "Add address [address_id=1]",
        "result": "success"
      }
    ],
    "switch_id": "000000000000000001"
  }
]

```

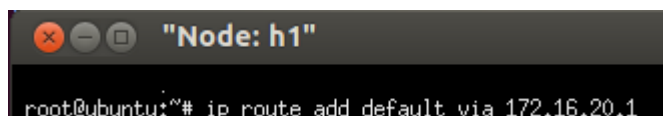
Figura 7. Ajustes de las direcciones en el controlador

- En la figura 8 se observa todas las configuraciones necesarias, las cuales deben de ser asignadas dentro del controlador c0 cómo se realizó en el paso anterior.

```
set -x
curl -X POST -d '{"address": "172.16.20.1/24"}' http://localhost:8080/router/0000000000000001 | python -mjson.tool
curl -X POST -d '{"address": "172.16.30.30/24"}' http://localhost:8080/router/0000000000000001 | python -mjson.tool
curl -X POST -d '{"address": "172.16.10.1/24"}' http://localhost:8080/router/0000000000000002 | python -mjson.tool
curl -X POST -d '{"address": "172.16.30.1/24"}' http://localhost:8080/router/0000000000000002 | python -mjson.tool
curl -X POST -d '{"address": "192.168.10.1/24"}' http://localhost:8080/router/0000000000000002 | python -mjson.tool
curl -X POST -d '{"address": "192.168.30.1/24"}' http://localhost:8080/router/0000000000000003 | python -mjson.tool
curl -X POST -d '{"address": "192.168.10.20/24"}' http://localhost:8080/router/0000000000000003 | python -mjson.tool
curl -X POST -d '{"gateway": "172.16.30.1"}' http://localhost:8080/router/0000000000000001 | python -mjson.tool
curl -X POST -d '{"gateway": "172.16.30.30"}' http://localhost:8080/router/0000000000000002 | python -mjson.tool
curl -X POST -d '{"gateway": "192.168.10.1"}' http://localhost:8080/router/0000000000000003 | python -mjson.tool
curl -X POST -d '{"destination": "192.168.30.0/24", "gateway": "192.168.10.20"}' http://localhost:8080/router/0000000000000002 |
```

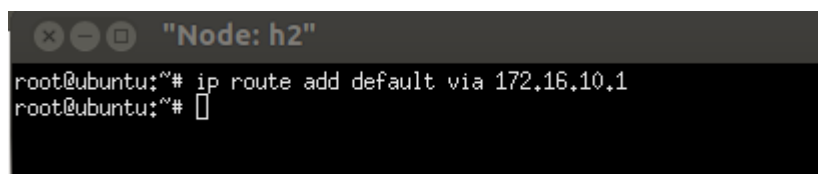
Figura 8. Lista de direcciones que deben de ser configuradas en el controlador

- Una vez ha concluido con los ajustes de las direcciones, es pertinente configurar las rutas por defecto de cada Host, para ello es necesario ingresar al **xterm** de cada host y ejecutar el siguiente comando en cada uno de ellos como se observa en la figura 9, 10 y 11.



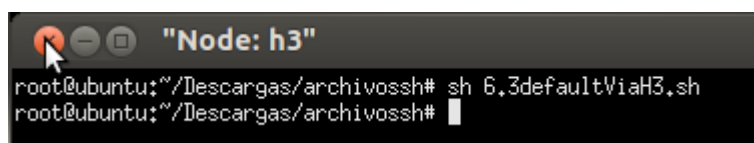
```
root@ubuntu:~# ip route add default via 172.16.20.1
```

Figura 9. Configuración de la ruta por defecto del Host 1.



```
root@ubuntu:~# ip route add default via 172.16.10.1
root@ubuntu:~#
```

Figura 10. Configuración de la ruta por defecto del Host 2.



```
root@ubuntu:~/Descargas/archivossh# sh 6.3defaultViaH3.sh
root@ubuntu:~/Descargas/archivossh#
```

Figura 11. Configuración de la ruta por defecto del Host 3.

- Una vez ha terminado de ajustar las rutas por defecto en cada Host, se puede deducir que la configuración ha sido exitosa, para verificar esto, es necesario realizar una prueba de conectividad como se observa en las figura 12 y 13.

```
mininet> h1 ping h2
PING 172.16.10.10 (172.16.10.10) 56(84) bytes of data.
64 bytes from 172.16.10.10: icmp_req=1 ttl=62 time=36.2 ms
64 bytes from 172.16.10.10: icmp_req=2 ttl=62 time=0.485 ms
64 bytes from 172.16.10.10: icmp_req=3 ttl=62 time=0.084 ms
64 bytes from 172.16.10.10: icmp_req=4 ttl=62 time=0.164 ms
```

Figura 12. Prueba de conectividad a través de Mininet

Anexo E. Emulación de una topología simple configurada con IPv6 con el controlador Nox13oflib	
Versión de Ubuntu	12.04
Herramientas utilizadas	<ul style="list-style-type: none"> - Mininet - NOX13OFLIB - Wireshark
Objetivos	<ul style="list-style-type: none"> - Realizar una explicación detallada de la forma como se configura y ejecuta las topologías basadas en IPv6 usando el emulador de Mininet. - Inicializar y utilizar el controlador Nox13oflib - Verificar a través de la herramienta Wireshark la utilización del protocolo IPv6.

EMULACIÓN DE UNA TOPOLOGÍA SIMPLE CONFIGURADA CON IPV6 CON EL CONTROLADOR NOX13OFLIB

En el presente anexo se hace una explicación detallada de los pasos utilizados para la configuración y ejecución de la topología basada en IPv6, en la figura 1 se observa la topología a realizar, que se compone de 2 host y un switch.

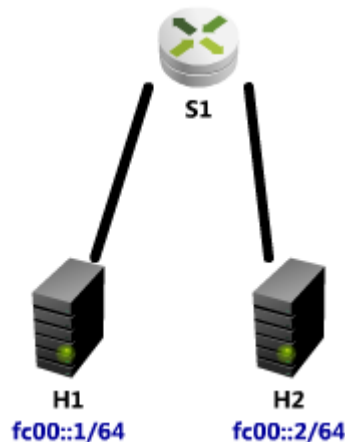


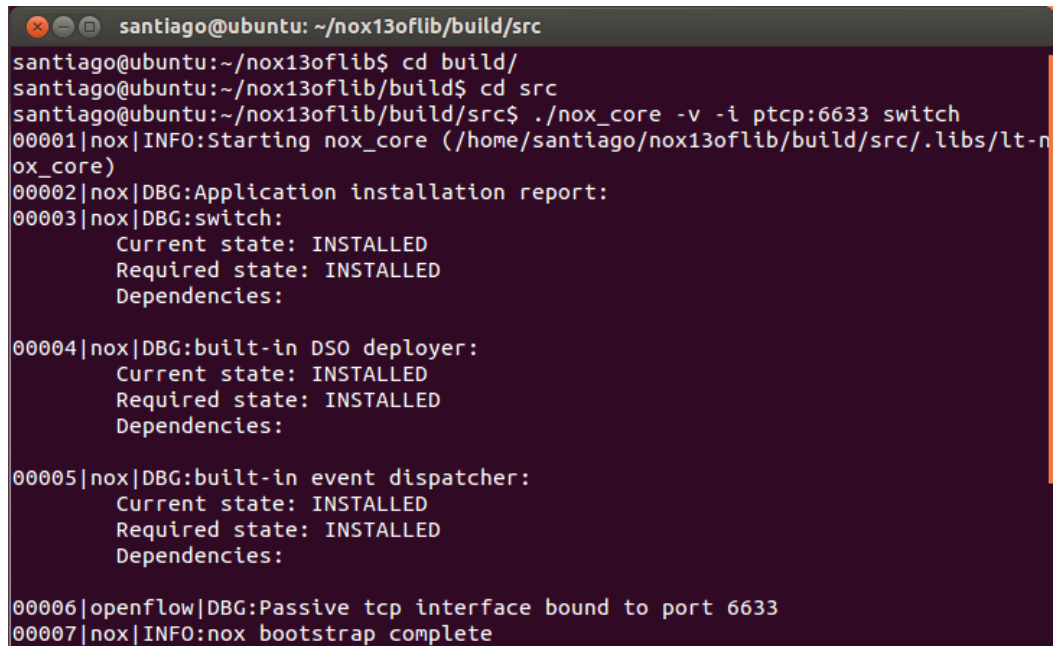
Figura 1. Topología IPv6 a realizar.

- El primer paso que el usuario debe realizar para la ejecución de la topología IPv6 es inicializar el controlador, en este caso NOX13OFLIB, para ello se ejecuta en un terminal de Linux los siguientes comandos:

```
cd nox13oflib
cd build
```

cd src

./nox_core -v -i ptcp:6633 switch



```
santiago@ubuntu: ~/nox13oflib/build/src
santiago@ubuntu:~/nox13oflib$ cd build/
santiago@ubuntu:~/nox13oflib/build$ cd src
santiago@ubuntu:~/nox13oflib/build/src$ ./nox_core -v -i ptcp:6633 switch
00001|nox|INFO:Starting nox_core (/home/santiago/nox13oflib/build/src/.libs/lt-nox_core)
00002|nox|DBG:Application installation report:
00003|nox|DBG:switch:
      Current state: INSTALLED
      Required state: INSTALLED
      Dependencies:

00004|nox|DBG:built-in DSO deployer:
      Current state: INSTALLED
      Required state: INSTALLED
      Dependencies:

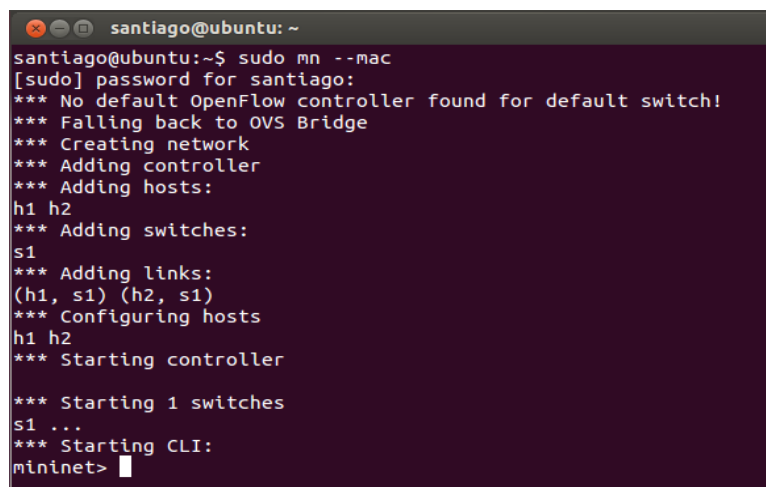
00005|nox|DBG:built-in event dispatcher:
      Current state: INSTALLED
      Required state: INSTALLED
      Dependencies:

00006|openflow|DBG:Passive tcp interface bound to port 6633
00007|nox|INFO:nox bootstrap complete
```

Figura 2. Ejecución del controlador NOX13OFLIB.

- Después de haber inicializado el controlador, en otra terminal, se procede a la ejecución de la topología, para este caso se utiliza una topología que trae por defecto el emulador Mininet.

sudo mn --mac



```
santiago@ubuntu: ~
santiago@ubuntu:~$ sudo mn --mac
[sudo] password for santiago:
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Figura 3. Ejecución de una topología

- Una vez el usuario ha ejecutado la topología, se procede a la asignación de las direcciones IPv6 a los host a través de los siguientes comandos:

```
h1 ifconfig h1-eth0 inet6 add fc00::1/64
h2 ifconfig h2-eth0 inet6 add fc00::2/64
```

```
mininet> h1 ifconfig h1-eth0 inet6 add fc00::1/64
mininet> h2 ifconfig h2-eth0 inet6 add fc00::2/64
```

Figura 4. Asignación de las direcciones IPv6

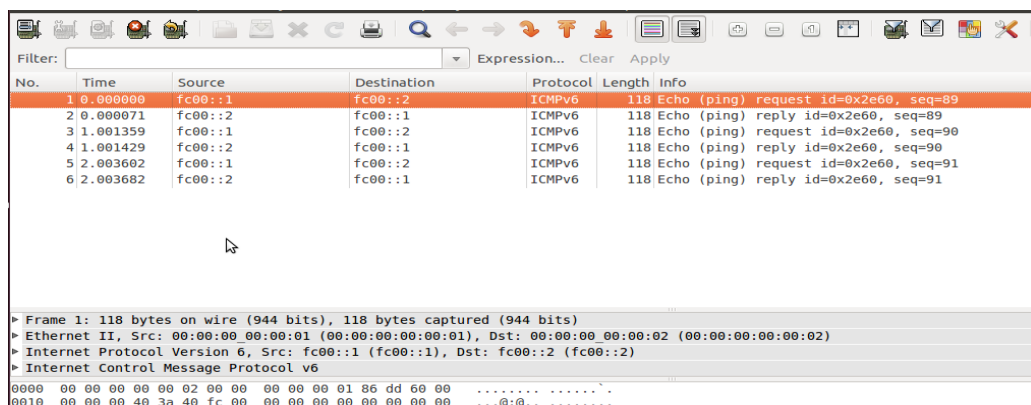
- Una vez se ha asignado las direcciones IPv6 se procede a la verificación de conectividad entre los host, para ello es necesario ejecutar el siguiente comando, como se observa en la figura 5.

```
h1 ping6 fc00::2 -I h1-eth0
```

```
mininet> h1 ip -6 neighbor show
mininet> h1 ping6 fc00::2 -I h1-eth0
PING fc00::2(fc00::2) from fc00::1 h1-eth0: 56 data bytes
64 bytes from fc00::2: icmp_seq=1 ttl=64 time=1.15 ms
64 bytes from fc00::2: icmp_seq=2 ttl=64 time=0.123 ms
64 bytes from fc00::2: icmp_seq=3 ttl=64 time=0.074 ms
```

Figura 5. Prueba de conectividad IPv6.

- Para verificar que se está trabajando bajo el protocolo IPv6, el usuario puede ejecutar el programa Wireshark a través del comando **sudo wireshark** o abriéndolo desde el ejecutable, allí puede seleccionarse la interfaz **s1-eth1** (ver Figura 6).



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fc00::1	fc00::2	ICMPv6	118	Echo (ping) request id=0x2e60, seq=89
2	0.000071	fc00::2	fc00::1	ICMPv6	118	Echo (ping) reply id=0x2e60, seq=89
3	1.001359	fc00::1	fc00::2	ICMPv6	118	Echo (ping) request id=0x2e60, seq=90
4	1.001429	fc00::2	fc00::1	ICMPv6	118	Echo (ping) reply id=0x2e60, seq=90
5	2.003602	fc00::1	fc00::2	ICMPv6	118	Echo (ping) request id=0x2e60, seq=91
6	2.003682	fc00::2	fc00::1	ICMPv6	118	Echo (ping) reply id=0x2e60, seq=91

Frame 1: 118 bytes on wire (944 bits), 118 bytes captured (944 bits)

Ethernet II, Src: 00:00:00:00:00:01 (00:00:00:00:00:01), Dst: 00:00:00:00:00:02 (00:00:00:00:00:02)

Internet Protocol Version 6, Src: fc00::1 (fc00::1), Dst: fc00::2 (fc00::2)

Internet Control Message Protocol v6

0000 00 00 00 00 00 02 00 00 00 00 00 01 86 dd 60 00@:..

0010 00 00 00 40 3a 40 fc 00 00 00 00 00 00 00 00@:..

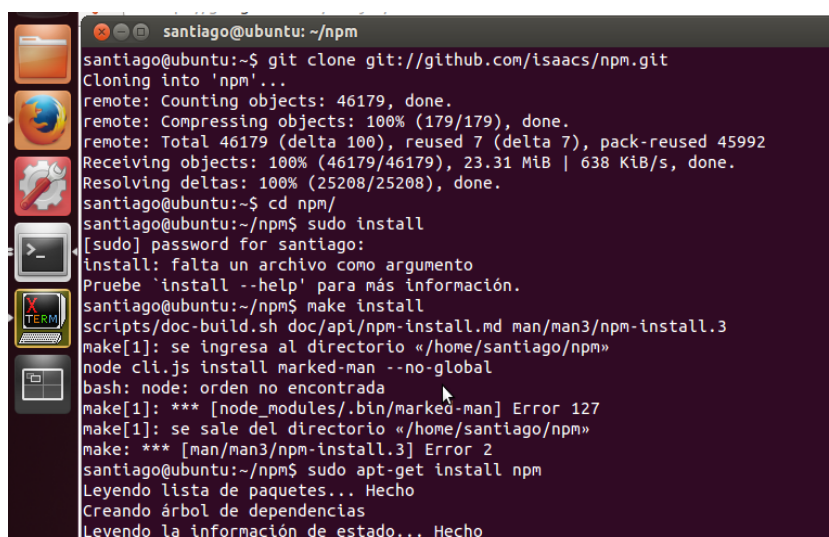
Figura 6. Verificación de IPv6 a través de Wireshark.

Anexo F. Creación y utilización de servidor HTTP en una Red Definida por Software usando RYU.	
Versión de Ubuntu	12.04
Herramientas utilizadas	<ul style="list-style-type: none"> - Mininet - RYU - Servidor HTTP
Objetivos	<ul style="list-style-type: none"> - Explicar de manera detallada la forma como se instala un servidor HTTP. - Explicar la forma como se ejecuta un servidor HTTP en el emulador Mininet.

CREACIÓN Y UTILIZACIÓN DE SERVIDOR HTTP EN UNA RED DEFINIDA POR SOFTWARE USANDO RYU.

Para la instalación del servidor HTTP es necesario ejecutar el siguiente comando **sudo apt-get install mpn**, una vez ejecutado aparecerá un error notificando que el paquete no se ha podido localizar, para dar solución puede ejecutarse el siguiente comando (ver Figura 1).

```
git clone git://github.com/isaacs/npm.git
cd npm
make install
```



```
santiago@ubuntu: ~/npm
santiago@ubuntu:~$ git clone git://github.com/isaacs/npm.git
Cloning into 'npm'...
remote: Counting objects: 46179, done.
remote: Compressing objects: 100% (179/179), done.
remote: Total 46179 (delta 100), reused 7 (delta 7), pack-reused 45992
Receiving objects: 100% (46179/46179), 23.31 MiB | 638 KiB/s, done.
Resolving deltas: 100% (25208/25208), done.
santiago@ubuntu:~$ cd npm/
santiago@ubuntu:~/npm$ sudo install
[sudo] password for santiago:
install: falta un archivo como argumento
Pruebe 'install --help' para más información.
santiago@ubuntu:~/npm$ make install
scripts/doc-build.sh doc/api/npm-install.md man/man3/npm-install.3
make[1]: se ingresa al directorio «/home/santiago/npm»
node cli.js install marked-man --no-global
bash: node: orden no encontrada
make[1]: *** [node_modules/.bin/marked-man] Error 127
make[1]: se sale del directorio «/home/santiago/npm»
make: *** [man/man3/npm-install.3] Error 2
santiago@ubuntu:~/npm$ sudo apt-get install npm
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

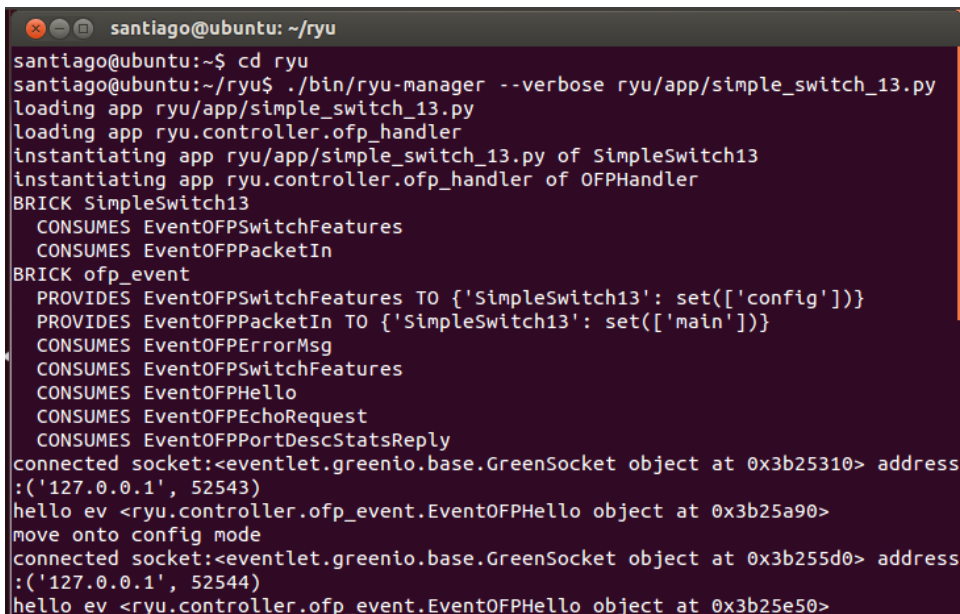
Figura 1. Instalación del servidor HTTP

Posterior a la instalación de los elementos anteriores, puede procederse así para la ejecución del servidor y la emulación de la SDN:

1. Abrir el terminal.
2. Ejecutar el controlador **RYU**, para ello solo basta ingresar el siguiente comando (Figura 2).

cd RYU

./bin/ryu-manager --verbose ryu/app/simple_switch_13.py



```
santiago@ubuntu: ~/ryu
santiago@ubuntu:~/ryu$ cd ryu
santiago@ubuntu:~/ryu$ ./bin/ryu-manager --verbose ryu/app/simple_switch_13.py
loading app ryu/app/simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ryu/app/simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPacketIn
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
  CONSUMES EventOFPErrormsg
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPPortDescStatsReply
connected socket:<eventlet.greenio.base.GreenSocket object at 0x3b25310> address
:('127.0.0.1', 52543)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x3b25a90>
move onto config mode
connected socket:<eventlet.greenio.base.GreenSocket object at 0x3b255d0> address
:('127.0.0.1', 52544)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x3b25e50>
```

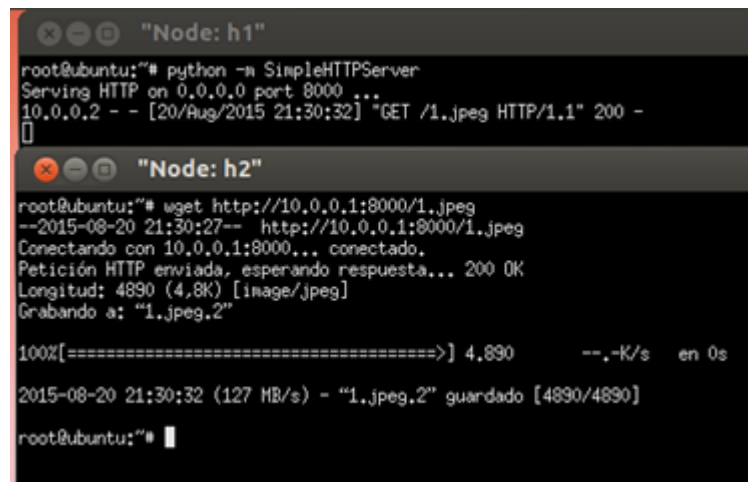
Figura 2. Ejecución del controlador RYU

3. Después de inicializar el controlador, se procede a la ejecución de la topología, para este caso se ha ejecutado una topología tipo single, esta está conformada por los siguientes elementos: 1 switch y 2 host, el comando para ejecutar dicha topología es el siguiente: **sudo mn -- topo single,2 --mac --switch user --controller remote**
4. Una vez ejecutada la topología se debe ingresar al **xterm** de cada host, ya que es ahí donde se configura el servidor HTTP, el comando que se debe escribir para ingresar al xterm es el siguiente:

- **xterm h1**
- **xterm h2**

5. Después de visualizar los xterm, se selecciona uno de los dos host para configurarlo como servidor, en este caso se selecciona el host 1 como el servidor, el comando que se utiliza para adaptar al host 1 como servidor es el siguiente: **python -m SimpleHTTPServer**; Una vez realizado este procedimiento debe ingresarse al host 2 para realizar la configuración del cliente y la transferencia del archivo, la estructura del comando para realizar este tipo de peticiones al servidor HTTP es el siguiente:

wget http://<IP donde corre el servidor>:<puerto>/<ruta, nombre y extensión del archivo a descargar>



```
"Node: h1"
root@ubuntu:~# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
10.0.0.2 - - [20/Aug/2015 21:30:32] "GET /1.jpeg HTTP/1.1" 200 -

"Node: h2"
root@ubuntu:~# wget http://10.0.0.1:8000/1.jpeg
--2015-08-20 21:30:27-- http://10.0.0.1:8000/1.jpeg
Conectando con 10.0.0.1:8000... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 4890 (4,8K) [image/jpeg]
Grabando a: "1.jpeg.2"

100%[=====] 4.890 --.-K/s en 0s

2015-08-20 21:30:32 (127 MB/s) - "1.jpeg.2" guardado [4890/4890]

root@ubuntu:~#
```

Figura 3. Ejecución de los comandos y transferencia de archivo.

Nota: es importante aclarar que la carpeta “home” fue la carpeta raíz donde se ejecutó el servidor y donde se encontraba el archivo “1.jpg” transferido en la petición del cliente.

AUTORIZACIÓN

Yo, **Bryan Valencia Suárez** mayor de edad, vecino de Pereira, identificado con la Cédula de Ciudadanía N° **1087491950** de Belén de Umbría actuando en nombre propio, en mi calidad de autor del trabajo de tesis____, monografía____, trabajo de grado____x____, informe de práctica empresarial____, denominado: **Prototipo de Redes IPv6 Definidas por Software Mediante Mininet**. Presentado como requisito para optar el título de **Ingeniero de Sistemas y Telecomunicaciones**, en el año **2016**, hago entrega del ejemplar respectivo y de sus anexos de ser el caso, en formato digital o electrónico (CD-ROM) y autorizo a LA UNIVERSIDAD CATÓLICA DE PEREIRA, para que en los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, Decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas sobre la materia, utilice y use en todas sus formas, los derechos patrimoniales de reproducción, comunicación pública, transformación y distribución (alquiler, préstamo público e importación) y los demás derechos comprendidos en aquellos, que me corresponden como creador de la obra objeto del presente documento. También autorizo a que dicha obra sea incluida en bases de datos. Esta autorización la hago siempre que mediante la correspondiente cita bibliográfica se le de crédito a mi trabajo como autor.

Con todo, en mi condición de autor me reservo los derechos morales de la obra antes citada con arreglo al artículo 30 de la Ley 23 de 1982. PARÁGRAFO: La presente autorización se hace extensiva no sólo a las facultades y derechos de uso sobre la obra en formato o soporte material, sino también para formato virtual, electrónico, digital, óptico, usos en red, internet, extranet, intranet, etc., y en general para cualquier formato conocido o por conocer.

EL AUTOR - ESTUDIANTES, manifiesta que la obra objeto de la presente autorización es original y la realizó sin violar o usurpar derechos de autor de terceros, por lo tanto la obra es de su exclusiva autoría y tiene la titularidad sobre la misma. PARÁGRAFO: En caso de presentarse cualquier reclamación o acción por parte de un tercero en cuanto a los derechos de autor sobre la obra en cuestión, EL ESTUDIANTE - AUTOR, asumirá toda la responsabilidad, y saldrá en defensa de los derechos aquí autorizados; para todos los efectos la Universidad actúa como un tercero de buena fe.

Firma (s),

A handwritten signature in purple ink, appearing to read 'Bryan Valencia Suárez', is written over a large, faint circular stamp or watermark.

CC. 1087491950

Pereira, 1 de diciembre de 2015

AUTORIZACIÓN

Yo, Santiago Santacruz Pareja mayor de edad, vecino de Pereira, identificado con la Cédula de Ciudadanía N° 1088299069 de Pereira actuando en nombre propio, en mi calidad de autor del trabajo de tesis____, monografía _____, trabajo de grado X informe de práctica empresarial _____, denominado: Prototipo de Redes IPv6 Definidas por Software mediante Mininet

Presentado como requisito para optar el título de Ingeniero de Sistemas y Telecomunicaciones en el año 2016 hago entrega del ejemplar respectivo y de sus anexos de ser el caso, en formato digital o electrónico (CD-ROM) y autorizo a LA UNIVERSIDAD CATÓLICA DE PEREIRA, para que en los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, Decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas sobre la materia, utilice y use en todas sus formas, los derechos patrimoniales de reproducción, comunicación pública, transformación y distribución (alquiler, préstamo público e importación) y los demás derechos comprendidos en aquellos, que me corresponden como creador de la obra objeto del presente documento. También autorizo a que dicha obra sea incluida en bases de datos. Esta autorización la hago siempre que mediante la correspondiente cita bibliográfica se le de crédito a mi trabajo como autor.

Con todo, en mi condición de autor me reservo los derechos morales de la obra antes citada con arreglo al artículo 30 de la Ley 23 de 1982. PARÁGRAFO: La presente autorización se hace extensiva no sólo a las facultades y derechos de uso sobre la obra en formato o soporte material, sino también para formato virtual, electrónico, digital, óptico, usos en red, internet, extranet, intranet, etc., y en general para cualquier formato conocido o por conocer.

EL AUTOR - ESTUDIANTES, manifiesta que la obra objeto de la presente autorización es original y la realizó sin violar o usurpar derechos de autor de terceros, por lo tanto la obra es de su exclusiva autoría y tiene la titularidad sobre la misma. PARÁGRAFO: En caso de presentarse cualquier reclamación o acción por parte de un tercero en cuanto a los derechos de autor sobre la obra en cuestión, EL ESTUDIANTE - AUTOR, asumirá toda la responsabilidad, y saldrá en defensa de los derechos aquí autorizados; para todos los efectos la Universidad actúa como un tercero de buena fe.

Firma (s),

Santiago Santacruz Pareja
CC. 1088299069

Pereira, 1 de diciembre de 2015