Entre Ciencia e Ingeniería, ISSN 1909-8367
Año 9 No. 17 - Primer Semestre de 2015, página 62 - 70

Mininet: una herramienta versátil para emulación y prototipado de Redes Definidas por Software[1]

Mininet: a versatile tool for emulation and prototyping of Software Defined Networking

B. Valencia, S. Santacruz, L.Y. Becerra y J.J. Padilla

Recibido Abril 20 de 2015 – Aceptado Mayo 29 de 2015

Resumen **- Las Redes Definidas por Software (SDN) son un nuevo paradigma en redes de datos que está siendo foco de estudio e investigación en los últimos años, debido a las diferentes ventajas que tiene con respecto al funcionamiento de las redes convencionales. Las Redes Definidas por Software separan el plano de control de un dispositivo de red, del plano de datos, permitiendo controlar, monitorizar y gestionar una red desde un nodo centralizado o controlador, lo cual promete simplificar la gestión de red e incluir innovación a través de su programación.**

Teniendo en cuenta que las SDNs son un tema de investigación vigente y conscientes de la importancia que tienen las herramientas que permiten simular o emular diferentes dispositivos de red para probar las nuevas soluciones propuestas por los investigadores, este artículo presenta en primera instancia una conceptualización general de Redes Definidas por Software, luego describe las diferentes herramientas de simulación y emulación más utilizadas y finalmente se enfoca en el estudio de una herramienta de emulación denominada Mininet, mostrando modos de instalación, creación de topologías de red, controladores y principales funcionalidades, revelando su versatilidad en el prototipado con este novedoso paradigma.

Palabras Clave **- Mininet, OpenFlow, Redes Definidas por Software (SDN).**

Abstract **- Software Defined Networks is a new paradigm in data networks, which in recent years is being focus of study and research because of the different advantages regarding the operation of conventional networks. Software Defined Networks separates the data plane from the control plane of a network device, allowing control, monitoring and management of a network from a central node or controller, which promises to simplify network management. It also includes innovation through its programmability.**

Given the fact that Software Defined Networks is a subject of current research, and given the importance of taking the tools to simulate or emulate network devices to test the solutions proposed by researchers and new features; this paper presents primarily a general conceptualization of Software Defined Networks, then describes different tools for simulation and emulation current used, and finally it focuses on the study of emulation tool called Mininet, showing installation modes, creating topologies, network controllers and main features revealing his versatility in this new paradigm prototyping.

Descriptors **- Mininet, OpenFlow, Software Defined Networking.**

I. Introducción

Las Redes Definidas por Software [1] son un nuevo paradigma en redes de datos que representan una gran oportunidad para la administración de las redes, ya que permiten, entre muchas otras cosas, el control de toda una red desde un único punto, la posibilidad de programar los dispositivos de red y en general la automatización de las redes de datos, todo gracias a la separación del plano de control y el plano de datos.

[1]Este artículo muestra resultados parciales del proyecto 'Prototipo de redes IPv6 definidas por software mediante Mininet' residencia en línea de investigación del proyecto "Contribución al soporte de ingeniería de tráfico en redes IPv6" registrado en el grupo de investigación e innovación en ingeniería de la Universidad Católica de Pereira.

B. Valencia y S. Santacruz son estudiantes de décimo semestre de Ingeniería de Sistemas y Telecomunicaciones de la Universidad Católica de Pereira, Pereira Colombia. bryan.valencia@ucp.edu.co y santiago.santacruz@ucp.edu.co.

L.Y. Becerra Sánchez, Docente Universidad Católica de Pereira, Pereira, Colombia, line.becerra@ucp.edu.co.

J. J. Padilla Aguilar, Docente Universidad Pontificia Bolivariana, Bucaramanga, Colombia, jhon.padilla@upb.edu.co.

Actualmente se están desarrollando muchas investigaciones alrededor de las redes definidas por software en diferentes áreas, lo cual implica una gran necesidad de contar con herramientas que permitan simular o emular los diferentes dispositivos de red con el fin de probar las soluciones propuestas por investigadores. Hoy existen algunas herramientas como NS-3 [2], EstiNet [3] y Mininet [4], que permiten simular o emular SDNs. Este artículo se enfoca principalmente en Mininet, un emulador de SDNs que incluye una colección de host, switches, controladores y enlaces virtuales; y puede ser usado para hacer pruebas sin usar dispositivos reales. Mininet ha sido ampliamente utilizado por muchos investigadores en varias instituciones debido a su flexibilidad, interactividad y escalabilidad entre otras características que la hacen una herramienta atractiva para probar soluciones en redes definidas por software. Algunos trabajos como en [5] [6] [7] [8], presentan las características de Mininet como plataforma de pruebas y su importancia en la construcción de prototipos de redes definidas por software. Y [9] [10] [11] [12] son algunos ejemplos de investigaciones en enrutamiento e interoperabilidad entre protocolos, que han usado Mininet como herramienta de emulación.

Este documento está organizado de la siguiente manera: en la sección II se da una introducción sobre Redes Definidas por Software donde se resaltan antecedentes, arquitectura y elementos que las componen; en la sección III se listan diferentes herramientas para la emulación y/o simulación de SDN. Posteriormente en la parte IV se realiza una explicación de Mininet destacando procesos de instalación, creación de topologías, modos de uso y particularidades del emulador. En la sección V se presenta una discusión sobre las ventajas y retos de SDN y Mininet.

II. Redes definidas por software, un nuevo paradigma

La gran cantidad de datos que circula por las redes y las actuales características de los protocolos y dispositivos de red, hacen las redes IP tradicionales complejas y difíciles de manejar. Uno de los obstáculos que genera estas problemáticas obedece al hecho de que los dispositivos de red son cajas negras, éstos contienen en su firmware las configuraciones de enrutamiento predefinidas por el fabricante, lo que imposibilita a los administradores y operadores de red acciones de gestión y optimización como la distribución de cargas, los privilegios de transmisión, cambios de métricas en los protocolos de enrutamiento, entre otros [13].

Además de la imposibilidad en la personalización de este tipo de configuraciones, se encuentra el hecho de que las redes actuales están integradas verticalmente: el plano de control y de datos estan acoplados. Para contribuir a la corrección de los inconvenientes mencionados anteriormente, ha surgido un paradigma llamado Redes Definidas por Software (SDN: Software Defined Networking), que permite 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.

En SDN, la inteligencia de la red está centralizada lógicamente en los controladores basados en software (el plano de control), y los dispositivos de red se convierten en simples dispositivos de reenvío de paquetes (el plano de datos) que se puede programar a través de una interfaz abierta como OpenFlow [14].

A. Redes programables anteriores a SDN

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 [15], una red multisalto, semejante a las 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.

Active Networks. No hubo desarrollo posterior a SOFTNET, pero su idea fue el embrión de las posteriores Active Networks [16]. Las Active Networks presentaban una arquitectura que consistía en llevar, embebido en los paquetes, pequeños programas que podían ser ejecutados por los nodos que estos atraviesan, lo que hacía posible que los switches y routers de la red procesaran los paquetes de datos, haciéndoles partícipes de los mensajes y dejando de ser solamente espectadores que se limitaban a enviar mensajes de un puerto a otro, de una forma "pasiva". De ahí el nombre de Active Networks.

DCAN. Otra iniciativa que tuvo lugar a mediados de la década de 1990 es el "Devolved Control of ATM Networks" (DCAN) [17]. 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.

NETCONF. En 2006, el Grupo de Trabajo de Ingeniería de Internet IETF propuso NETCONF [18] 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 de la cual los datos de configuración extensibles podían ser enviados y recuperados.

NETCONF cumple con el objetivo de simplificar la reconfiguración del dispositivo y actúa como un bloque de construcción para la gestión, pero realmente no permite la separación entre los planos de datos y de control. 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.

Ethane. El antecedente inmediato de OpenFlow fue el proyecto SANE/Ethane [19], que, en 2006, definió una nueva arquitectura para redes empresariales. El enfoque de Ethane es 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.

B. Arquitectura y elementos de las SDN

Como se muestra en las Fig. 1 y 2, la arquitectura de SDN consta conceptualmente de tres capas: capa de infraestructura, capa lógica y capa de aplicación [15].

La capa de infraestructura está conformada 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.

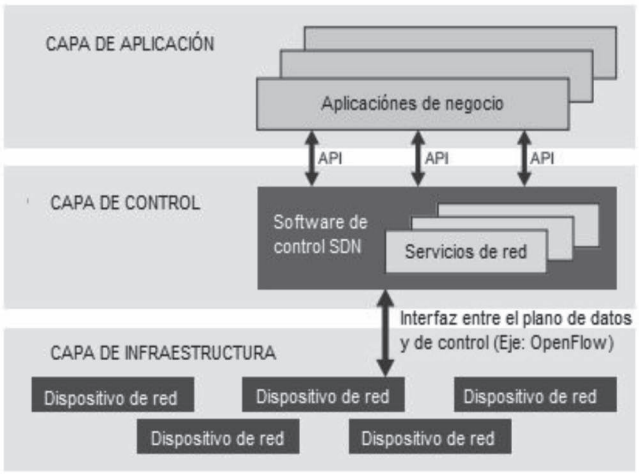

Fig. 1. Arquitectura de SDN [20]

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.

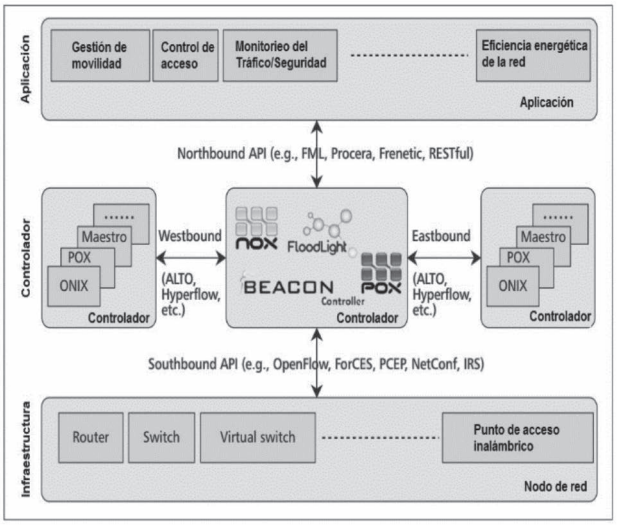

Fig. 2. Arquitectura funcional de SDN [21].

La capa superior es la capa de aplicación [15]. Esta capa la componen las aplicaciones realizadas por los usuarios, que son las que permiten personalizar las redes. Estas incorporan las NorthBound APIs, tal como se observa en la Fig. 2. Es aquí donde hay un cambio significativo 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. Por tanto, de manera general, el funcionamiento es así: 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, Fig 2.

Estas SouthBound [1] consisten en un protocolo o interfaz de programación bien definida entre los elementos de la capa de datos y la capa de control. La interfaz que má se utiliza y la más notable es OpenFlow [22]. 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 [13].

III. Herramientas de simulación y emulación para SDN

En una SDN, debido a que la operación y la inteligencia de la red están totalmente centralizadas en un controlador, la exactitud y la eficiencia de las funciones implementadas por el controlador deben estar completamente probadas antes de su uso en una red real [23].

Probar, corregir y evaluar el rendimiento de un protocolo de red puede realizarse a través de varios enfoques. Una manera es sobre un banco de pruebas experimental (por

ejemplo, PlanetLab [24], Emulab [25], [26] y OFELIA [27], [28]). Debido a que este enfoque utiliza dispositivos físicos que ejecutan sistemas operativos y aplicaciones reales, puede generar resultados más realistas, pero el costo de la construcción de un gran banco de pruebas experimental es enorme, y no es fácilmente accesible a muchos usuarios.

Otro enfoque común es a través de la simulación, en el que las operaciones de los dispositivos reales y sus interacciones se modelan y ejecutan en un programa de software. El enfoque de simulación tiene ventajas como bajo costo, ser flexible y controlable, escalable, repetible y es accesible a muchos usuarios. Sin embargo, los resultados de la simulación pueden no ser lo suficientemente precisos. Para superar este problema, se puede utilizar el método de emulación que difiere de la simulación en que éste es como un experimento y por lo tanto debe ser ejecutado en tiempo real, mientras que la velocidad de la simulación puede ser más rápida o más lenta que en tiempo real. Además, en una emulación algunos dispositivos ejecutan sistemas operativos y programas de aplicación reales que interactúan con algunos dispositivos simulados. En contraste, en una simulación generalmente no están involucrados ni sistemas operativos ni aplicaciones reales [23].

En la actualidad los emuladores y/o simuladores que permiten el prototipado y la experimentación con Redes Definidas por Software más reconocidos son ns-3 [2], EstiNet [3] y Mininet [29].

A. ns-3

Actualmente, muy pocos simuladores de red son compatibles con el protocolo OpenFlow; el más notable es ns-3. La herramienta de software libre ns-3 es el simulador de red más ampliamente utilizado en el mundo [23]. Ns-3 simula el funcionamiento de un switch OpenFlow compilando y enlazando un módulo C++ de un switch OpenFlow con su código de motor de simulación. Para simular un controlador OpenFlow real, ns-3 también implementa un módulo de C ++, que compila y vincula con su código de motor de simulación. De hecho, todos los dispositivos y objetos simulados en ns-3 se implementan como módulos C ++ compilados y enlazados con su código de motor de simulación para formar un programa ejecutable a nivel de usuario, es decir, ns-3. La Tabla I, muestra algunas diferencias entre ns-3, EstiNet y Mininet, entre ellas que ns-3 es únicamente un simulador y por tanto no tiene capacidades de emulación.

B. EstiNet

Con esta herramienta puede simularse o emularse una red OpenFlow. Una buena propiedad de EstiNet es que utiliza la metodología del "kernel re-entering" [30] para permitirle a los programas de aplicación real no modificados funcionar en máquinas simuladas. Debido a esta capacidad, los resultados de la simulación de EstiNet son tan precisos como los resultados obtenidos de un emulador. EstiNet utiliza su propio reloj de simulación para controlar el orden de ejecución de eventos de simulación, pero generando resultados muy precisos. En una red OpenFlow simulada por EstiNet, programas de controlador OpenFlow reales como NOX/POX [31], [32], Ryu [33], y Foodlight [34] se pueden ejecutar directamente en un host simulado para controlar switches OpenFlow simulados sin ninguna modificación. En cuanto a EstiNet como emulador, como cualquier emulador debe hacer, éste necesita llevar a cabo la emulación en tiempo real y permite que el programa de aplicación del controlador se ejecute en una máquina externa para controlar switches OpenFlow emulados. Debido a la utilización de la metodología del kernel re-entering, EstiNet también permite al programa del controlador y a los switches OpenFlow emulados ejecutarse en la misma máquina.

A. Mininet

El emulador Mininet es una plataforma de pruebas de red de código abierto y rápidamente configurable. Hasta ahora, es la herramienta de apoyo a la investigación de las SDN OpenFlow más conocida, como se denotó en la conferencia ONS 2013 [35]. Mininet utiliza hosts virtuales, switches y enlaces para crear una red en un solo núcleo del sistema operativo, y utiliza la pila de red real para procesar paquetes y conectarse a las redes reales. Además, las aplicaciones de red basadas en Unix/Linux, también se pueden ejecutar en los hosts virtuales. En una red OpenFlow emulada por Mininet, una aplicación de controlador real OpenFlow se puede ejecutar en una máquina externa o en el mismo equipo en el que se emulan los hosts virtuales [36].

Tabla I
Comparación entre EstiNet, ns-3 y Mininet [23].

	ESTINET	**NS-3**	**MININET**
Modo de simulación	Sí	Sí	No
Modo de emulación	Sí	No	Sí
Compatible con controladores reales	Sí	No	Sí
Resultado repetible	Sí	No	Sí
Escalabilidad	Alta (para un solo proceso)	Alta (para un solo proceso)	Media (para múltiples procesos)
Exactitud en los resultados de rendimiento	Sí	No admite protocolo de árbol y ningún controlador real	Sin fidelidad de rendimiento
Soporte de GUI	Para configuración y observación	Sólo para observación	Sólo para observación

IV. Mininet

Mininet es un emulador para el despliegue de redes sobre los limitados recursos de un ordenador sencillo simple o máquina virtual [6]. Éste utiliza el kernel de Linux y otros recursos para emular elementos de la SDN como el controlador, los switches OpenFlow y los hosts. A continuación se describen las alternativas para instalar y utilizar Mininet, las topologías disponibles, formas de

acceder a controladores remotos diferentes al que viene por defecto, entre otras pautas para la utilización de esta herramienta.

A. Instalación

Mininet puede instalarse de varias maneras. Una de ellas, recomendada para empezar a interactuar con la plataforma, consiste simplemente en descargar una imagen de una máquina virtual que puede ser ejecutada posteriormente en un software como VirtualBox [37].

La segunda, es en una instalación nativa en un sistema operativo Ubuntu o Fedora (se recomienda Ubuntu). Para esto basta sólo con clonar el código fuente, abrir el directorio y ejecutar una instrucción específica para la instalación. Esta instrucción permite seleccionar, entre una serie de opciones, los elementos específicos a instalar. Por ejemplo, puede instalarse o no Wireshark [38] junto con los otros elementos del emulador. Las instrucciones para la instalación de Mininet por cualquiera de los dos métodos mencionados están disponibles en [4].

B. Topologías

Después de instalar el emulador, sólo con escribir la instrucción **$ sudo mn** en el Shell se ejecuta Mininet y se crea una topología denominada mininal. Aquí el prompt cambia a **'mininet>'** lo que indica que se está dentro del entorno.

La instrucción **sudo mn** puede venir sola o acompañada de diferentes parámetros, entre estos algunos de ayuda, para limpiar y reiniciar el emulador y otros para ejecutar topologías de otros tipos.

Las topologías disponibles son: minimal, single, linear, tree y personalizadas. En cualquiera de éstas existe un controlador, varían en el número de hosts, switches y los enlaces entre éstos. Minimal, por ejemplo, está compuesta por dos hosts conectados a un switch; single también es una topología con un único switch pero la cantidad de host puede indicarse por parámetro.

En la Fig 3, se muestra una topología Tree (o de árbol) que puede ser emulada a través de Mininet. Para ejecutarla se utiliza la instrucción sudo mn --topo tree,depth=n donde n es la profundidad en niveles de la red. En el caso de la Fig. 3, la profundad escogida es 3.

En este caso Mininet crea **8** hosts (2 conectados a cada uno de los switches del nivel más inferior) y en total 7 switches. La Fig. 4, presenta la ejecución de esta topología y el comando **links** con el que pueden comprobarse las conexiones entre todos los equipos y dispositivos emulados.

Las topologías personalizadas, por su parte, permiten el diseño de redes con las conexiones y la cantidad de dispositivos específicos deseados. Estas deben implementarse en scripts de Python utilizando las librerías y la API [39] que para este fin tiene Mininet.

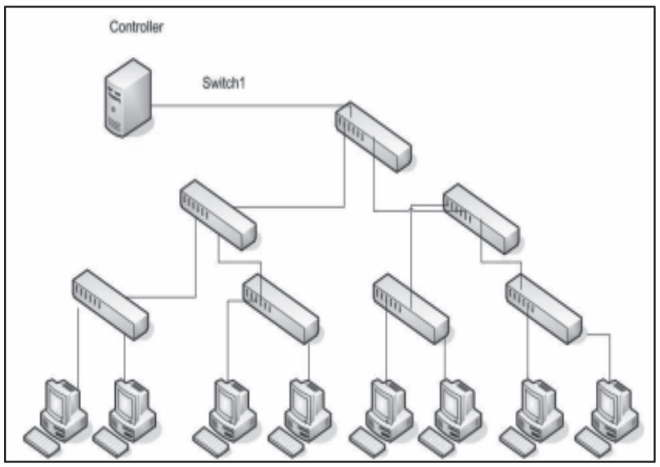

Fig. 3. Topología Tree [15].

Si se quiere crear una topología que no puede realizarse con ninguna de las topologías que Mininet ofrece por defecto, por ejemplo, una topología con dos switches interconectados, uno de ellos con dos hosts y otro con tres, podría codificarse un script en Python como el de la Fig. 5.

La ejecución de una topología personalizada en Mininet puede visualizarse en la Fig. 6. Nótese que el comando utilizado para llamar el script e iniciar la red es: $ sudo mn --custom topo-articulo.py --topo mytopo. En este comando, la palabra 'custom' va seguida de la ruta de ubicación del archivo Python, después se escribe '--topo' y por último se especifica exactamente la(s) palabra(s) que se encuentran en comillas sencillas en la última línea del script de la Fig. 5.

La Fig. 6, también muestra la forma de probar conectividad entre dos host en Mininet a través de la realización de un ping, en este caso entre el host1 y el host4.

```
bryan@bryan:~$ sudo mn --topo tree,depth=3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6)
 (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Starting CLI:
mininet> links
s1-eth1<->s2-eth3 (OK OK)
s1-eth2<->s5-eth3 (OK OK)
s2-eth1<->s3-eth3 (OK OK)
s2-eth2<->s4-eth3 (OK OK)
s3-eth1<->h1-eth0 (OK OK)
s3-eth2<->h2-eth0 (OK OK)
s4-eth1<->h3-eth0 (OK OK)
s4-eth2<->h4-eth0 (OK OK)
s5-eth1<->s6-eth3 (OK OK)
s5-eth2<->s7-eth3 (OK OK)
s6-eth1<->h5-eth0 (OK OK)
s6-eth2<->h6-eth0 (OK OK)
s7-eth1<->h7-eth0 (OK OK)
s7-eth2<->h8-eth0 (OK OK)
mininet>
```

Fig. 4. Topología Tree y comando links.

```
from mininet.topo import Topo
from mininet.topo import SingleSwitchTopo
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.node import *

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

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

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        newHost1 = self.addHost('h3')
        newHost2 = self.addHost('h4')
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( leftSwitch, rightHost )
        self.addLink( rightSwitch, newHost1)
        self.addLink( rightSwitch, newHost2)

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

Fig. 5. Script en Python de Topología personalizada.

```
bryan@bryan:~/mininet/custom$ sudo mn --custom topo-articulo.py --topo mytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s3 s4
*** Adding links:
(h1, s3) (s3, h2) (s3, s4) (s4, h3) (s4, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s3 s4
*** Starting CLI:
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_req=1 ttl=64 time=3.10 ms
64 bytes from 10.0.0.4: icmp_req=2 ttl=64 time=0.627 ms
64 bytes from 10.0.0.4: icmp_req=3 ttl=64 time=1.31 ms
64 bytes from 10.0.0.4: icmp_req=4 ttl=64 time=2.55 ms
64 bytes from 10.0.0.4: icmp_req=5 ttl=64 time=1.96 ms
```

Fig. 6. Topología Custom y comando ping entre h1 y h4.

C. Controladores

Mininet utiliza un controlador por defecto incorporado por la distribución OpenFlow de referencia que actúa como un switch clásico [15], lo que significa que va creando una tabla donde se asocian direcciones MAC con puertos, según va recibiendo tramas. Lo que hace después de asociar la MAC con un puerto es enviar un mensaje OpenFlow al switch para crear una nueva entrada en su tabla de flujo. OpenFlow nombra a este tipo de controlador Ethernet Learning Switch [40].

El fin primario de este controlador que trae por defecto Mininet es ayudar a entender el flujo de mensajes y la separación del plano de control y de datos, también llamado de decisión y conmutación respectivamente, por lo que no admite componentes creados por el usuario.

Como se expone en la Tabla I, Mininet admite la utilización de controladores reales. Al iniciar una red en este emulador, cada switch se puede conectar a un controlador remoto, que podría estar en la máquina virtual, fuera de la máquina virtual, en el equipo local, o en cualquier parte del mundo [41]. Para esto, cada vez que se inicie una topología debe indicarse que se usará un controlador remoto. El controlador puede ser NOX, POX, RYU, Foodlight o Beacon [42].

Usar un controlador diferente al predefinido por Mininet puede obedecer a diferentes razones, una de ellas, que ya se ha mencionado, es que puede ser necesaria la ejecución de aplicaciones desarrolladas por el usuario. Además, de la mano de la evolución de OpenFlow, puede requerirse el soporte de versiones nuevas de este protocolo.

A. Mininet y OpenFlow 1.3

A medida que OpenFlow ha avanzado y madurado, ha abierto la puerta a otras posibilidades. La Tabla II proporciona los diferentes campos y protocolos soportados por las diferentes versiones de OpenFlow.

Para usar una versión reciente de este estándar, como por ejemplo OpenFlow 1.3 es necesario utilizar controladores que la soporten. 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 [43] creó una versión de este controlador que soporta muchas de las nuevas características de OpenFlow 1.3 a la que llamó nox13oflib [44].

TABLA II

PROTOCOLOS Y CAMPOS SOPORTADOS POR OPENFLOW (OF) [45].

	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

En [46], pueden consultarse los pasos y requerimientos para instalar y utilizar nox13oflib. 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 Fig. 7, el puerto usado es el 6635.

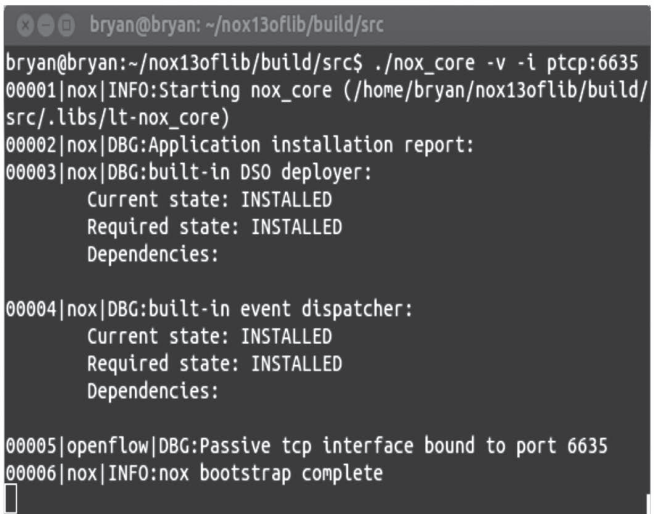

Fig. 7. Comando para iniciar el controlador nox13oflib.

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 Fig. 8, ilustra la ejecución de una topología linear que utiliza el controlador remoto nox13oflib.

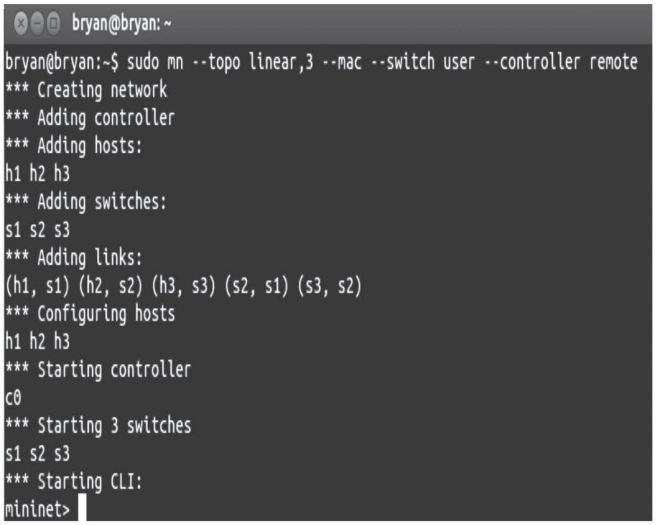

Fig. 8. Utilización de controlador remoto.

V. Conclusiones y trabajos futuros

Es claro que las Redes Definidas por Software están siendo percibidas por la industria y la academia como una revolución en el mundo de las redes de comunicaciones y a su vez, están siendo el foco de investigación en universidades de todo el mundo y campo de estudio en una gran cantidad de empresas.

Ahora bien, este es un modelo en desarrollo que está en constante experimentación y evaluación, por lo tanto las herramientas de simulación o emulación como Mininet representan una gran oportunidad para avanzar en los trabajos y estandarizaciones pendientes de este nuevo paradigma, como la seguridad y la funcionalidad del controlador para evitar los cuellos de botella y el bajo rendimiento en la red.

Mininet ha sido utilizado por más de 100 investigadores en más de 18 instituciones [47], 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 Mininet en redes de nueva generación tales como redes IPv6 [48], redes MPLS [49], protocolos de enrutamiento y señalización como OSPF-TE [50] y RSVP-TE [51] para realizar ingeniería de tráfico en Internet, con IPv6 móvil [52] 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.

Referencias

[1] B. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka and T. Turletti, “A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks.,” Communications Surveys and Tutorials, IEEE Communications., vol. 2, no. 4, pp. 1617-1634, 2014.

[2] ns-3. [Online]. Available: https://www.nsnam.org/. [Accessed 30 mayo 2015].

[3] EstiNet, “EstiNet Technolgies,” 2015. [Online]. Available: http://www.estinet.com/. [Accessed 30 mayo 2015].

[4] Mininet, “Download/Get Started With Mininet,” [Online]. Available: http://mininet.org/download/. [Accessed 30 mayo 2015].

[5] B. Lantz, B. Heller and N. McKeown, “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks,” 2010. [Online]. Available: http://klamath.stanford.edu/~nickm/papers/a19-lantz.pdf. [Accessed 1 junio 2015].

[6] K. Kaur, J. Singh and N. Ghumnan, “MiniNet as Software Defined Networking Testing Platform,” 2014. [Online]. Available: http://www.sbsstc.ac.in/icccs2014/Papers/Paper29.pdf. [Accessed 08 mayo 2015].

[7] C. Pal, S. Veena , . R. P. Rustagi and . K. Murthy, “Implementation of Simplified Custom Topology Framework in Mininet,” Asia-Pacific Conference on Computer Aided System Engineering (APCASE), pp. 48 - 53, 2014.

[8] R. . L. Santos de Oliveira, C. M. Schweitzer, A. A. Shinoda and L. Rodrigues Prete, “Using Mininet for Emulation and Prototyping,” IEEE Colombian Conference on Communications and Computing (COLCOM)., pp. 1 - 6, 2014.

[9] D. Mehmet and A. Mostafa , “Design and analysis of techniques for mapping virtual networks,” Computer Communications, 2014.

[10] J.-R. Jiang, H.-W. Huang, J.-H. Liao and S.-Y. Chen, “Extending Dijkstra’s Shortest Path Algorithm for Software Defined Networking,” Asia-Pacific Network Operation and Management Symposium (APNOMS), 2014.

[11] Sang Min Park, Seungbum Ju and Jaiyong Lee, “Efficient routing

for traffic offloading in Software-defined Network," International Workshop on Software Defined Networks for a New Generation of Applications and Services (SDN-NGAS-2014), 2014.
[12] Kannan Govindarajan, Sharipah Setapa, Kong Chee Meng and Hong Ong, "Interoperability Issue between IPv4 and IPv6 in OpenFlow Enabled Network," 2014 International Conference on Computer, Control, Informatics and Its Applications, 2014.
[13] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, 2015.
[14] N. McKeown, G. Purulkar, S. Shenker, T. Anderson, L. Peterson, J. Turner, H. Balakrishnan and J. Rexford, "OpenFlow: Enabling Innovation in Campus Networks," 2008. [Online]. Available: http://archive.openflow.org/documents/openflow-wp-latest.pdf. [Accessed 26 mayo 2015].
[15] Ó. Roncero, "Software Defined Networking," 2014. [Online]. Available: http://upcommons.upc.edu/pfc/bitstream/2099.1/21633/4/Memoria.pdf. [Accessed 2015 mayo 08].
[16] D. Tennenhouse and D. Wetherall, "Towards an active network architecture," in DARPA Active NEtworks Conference and Exposition, 2002. Proceedings, San Francisco, IEEE, 2002, pp. 2-15.
[17] University of Cambridge, "Devolved Control of ATM Networks.," [Online]. Available: http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/#intro. [Accessed 2015 mayo 2015].
[18] R. Enns, E. M. Bjorklund, J. Schoenwaelder and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241, 2011.
[19] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown and S. Shenker, "Ethane: taking control of the enterprise," ACM SIGCOMM Computer Communication Review, vol. 37, no. 4, pp. 1-12, 2007.
[20] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf. [Accessed 08 mayo 2015].
[21] S. Sezer y et. al., «Are We Ready for SDN? Implementation Challenges for Software-Defined Networks,» IEEE Communications Magazine, vol. 51, nº 7, pp. 36-43, July 2013.
[22] A. Bianco, V. Krishnamoorthi, N. Li and L. Giraudo, "OpenFlow driven ethernet traffic analysis," in 2014 IEEE Internacional Conference on Communications (ICC), Sydney, IEEE, 2014, pp. 3001-3006.
[23] S.-Y. Wang, C.-L. Chou and C.-M. Yang, "EstiNet openflow network simulator and emulator," IEEE Communications Magazine, vol. 51, no. 9, pp. 110-117, 2013.
[24] PlanetLab. [Online]. Available: https://www.planet-lab.org/. [Accessed 30 mayo 2015].
[25] Emulab, "Emulab: Total Network Testbed," 2015. [Online]. Available: https://www.emulab.net/. [Accessed 30 mayo 2015].
[26] M. Schwarz, M. Rojas, C. Miers, F. Redigolo and T. Carvalho, "Emulated and software defined networking convergence," in 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, IEEE, 2013, pp. 700-703.
[27] OFELIA, 2014. [Online]. Available: http://www.fp7-ofelia.eu/. [Accessed 30 mayo 2015].
[28] M. Suñé, B. L., W. H., T. Rothe, A. Köpsel, D. Colle, B. Puype, D. Simeonidou, R. Nejabati, M. Channegowda, M. Kind, T. Dietz and A. Autenrieth, "Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed," Computer Networks, vol. 61, p. 132–150, 2014.
[29] MiniNet, "MiniNet: An Instant Virtual Network on your Laptop (or other PC)," 2015. [Online]. Available: www.mininet.org. [Accessed 26 mayo 2015].
[30] S. Wang and H. Kung, "A Simple Methodology for Constructing extensible and high-fidelity TCP/IP network simulators.," in INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings., vol. 3, New York, IEEE, 1999, pp. 1134 - 1143.
[31] NOX, 2015. [Online]. Available: http://www.noxrepo.org/. [Accessed 30 mayo 2015].
[32] N. Gude, B. Pfaff, T. Koponen, M. Casado, S. Shenker, J. Pettit and N. McKeown, "NOX: Towards an Operating System for Networks," Computer Communication Review, vol. 38, no. 3, pp. 105-110, 2008.
[33] RYU, "Component-based software defined networking framework. Build SDN Agilely," 2014. [Online]. Available: http://osrg.github.io/ryu/. [Accessed 30 mayo 2015].
[34] Floodlight, "Floodlight OpenFlow Controller," [Online]. Available: http://www.projectfloodlight.org/floodlight/. [Accessed 30 mayo 2015].
[35] "Open Networking Summit 2013," Santa Clara, CA, 2013.
[36] S.-Y. Wang, "Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet," in IEEE Symposium on Computers and Communications (ISCC), Funchal, 2014.
[37] "VirtualBox," [Online]. Available: https://www.virtualbox.org/.
[38] "Wireshark," [Online]. Available: https://www.wireshark.org/.
[39] "Mininet Python API Refence Manual," [Online]. Available: http://mininet.org/api/hierarchy.html. [Accessed 31 mayo 2015].
[40] A. Bagewadi and R. M. Babu, "Towards an Ethernet Learning Switch and Bandwidth Optimization using POX Controller," International Journal of Advanced Research in Computer and Communication Engineering, vol. 3, no. 7, pp. 7531-7535, 2014.
[41] Mininet, "Using a Remote Controller," [Online]. Available: http://mininet.org/walkthrough/#using-a-remote-controller. [Accessed 31 mayo 2015].
[42] D. Erickson, "The Beacon OpenFlow Controller," [Online]. Available: http://yuba.stanford.edu/~derickso/docs/hotsdn15-erickson.pdf. [Accessed 31 mayo 2015].
[43] "CPqD," [Online]. Available: http://www.cpqd.com.br/es/.
[44] "nox13oflib," [Online]. Available: https://github.com/CPqD/nox13oflib. [Accessed 31 mayo 2015].
[45] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," Future Internet, vol. 6, no. 2, pp. 302-336, 2014.
[46] "OpenFlow 1.3 Tutorial," 2013. [Online]. Available: https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow-1.3-Tutorial. [Accessed 31 mayo 2015].
[47] "Mininet-discuss mailing list," [Online]. Available: https://mailman.stanford.edu/mailman/listinfo/mininet-discuss.
[48] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6)," RFC2460, p. 39, 1998.
[49] E. Rosen, A. Viswanathan and R. Callon, "Multiprotocol Label Switching Architecture," RFC3031, 2001.
[50] D. Katz, K. Kompella and D. Yeung, "Traffic Engineering (TE) Extensions to OSPF Version 2," RFC3630, p. 14, 2003.
[51] D. Awduche, L. Berger, D. Gan, T. Li and V. Srinivasan, "RSVP-TE: Extensions to RSVP for LSP Tunnels," RFC3209, 2001.
[52] C. Perkins, D. Jhonson and J. Arkko, "Mobility Support in IPv6," RFC6275, 2011.

Bryan Valencia Suárez. Nació en Belén de Umbría,Risaralda, Colombia el 22 de septiembre de 1994. Es estudiante de décimo semestre de Ingeniería de Sistemas y Telecomunicaciones en la Universidad Católica de Pereira-UCP. Fue integrante del semillero de investigación inscrito al Grupo de Investigación GEMA-UCP en el cual, participó en un proyecto sobre la utilización de Realidad Aumentada como herramienta en la Enseñanza-Aprendizaje de Geometría. Actualmente hace parte del Semillero de Investigación en Telecomunicaciones (SIT) inscrito al Grupo de Investigación e Innovación en Ingenierías de la Universidad Católica de Pereira (GIII-UCP) donde lleva a cabo un trabajo de grado relacionado con la implementación del protocolo IPv6 en Redes Definidas por Software (SDN).

Santiago Santacruz Pareja. Nació en Pereira, Risaralda, el 3 de junio de 1992. Estudiante de décimo semestre Ingeniería de sistemas y telecomunicaciones de la Universidad Católica de Pereira. Fue integrante del semillero de investigación inscrito al Grupo de Investigación GEMA participó en un proyecto sobre la utilización de Realidad Aumentada como herramienta

en la Enseñanza-Aprendizaje de Geometría. Actualmente hace parte del Semillero de Investigación en Telecomunicaciones (SIT) inscrito al Grupo de Investigación e Innovación en Ingenierías de la Universidad Católica de Pereira (GIII-UCP) donde lleva a cabo un trabajo de grado relacionado con la implementación del protocolo IPv6 en Redes Definidas por Software (SDN)

Line Yasmín Becerra Sánchez. Es Ingeniera Electrónica de la Universidad Pontificia Bolivariana (1999). Especialista en Telecomunicaciones de la Universidad Pontifica Bolivariana (2005). Magíster de la Universidad Pontificia Bolivariana (2009). Actualmente es estudiante de doctorado en ingeniería en el área de telecomunicaciones de la misma universidad, es docente de la Universidad Católica de Pereira y Pertenece al Grupo de Investigación GIII-UCP. Sus áreas de interés son: Ingeniería de tráfico, Enrutamiento, Redes Móviles, Simulación de Redes, Internet, IPv6, MIPv6, HMIPv6.

Jhon Jairo Padilla Aguilar. Es ingeniero Electrónico de la Universidad del Cauca (1993). Obtuvo su grado de Maestría en Informática de la Universidad Industrial de Santander (1998) y es Doctor en Ingeniería Telemática por la Universidad Politécnica de Cataluña (2008). Actualmente es docente de la Facultad de Ingeniería Electrónica de la Universidad Pontificia Bolivariana y coordina el Grupo de Investigación en Telecomunicaciones (GITEL) de dicha universidad. Sus áreas de interés son: Ingeniería de tráfico, Internet, Calidad de Servicio en Internet, redes inalámbricas, IPv6.