SDN Redes Definidas por *Software* usando MiniNet.

SDN Software Defined *Networks* using MiniNet

Luis Miguel Amaya Fariño[1] https://orcid.org/0000-0002-9680-127X
Juan Fernando Arroyo Pizarro[1] https://orcid.org/0000-0002-5907-7593
Mónica Jaramillo Infante[1] https://orcid.org/0000-0001-9718-4122
Alfredo Tumbaco Reyes[1] https://orcid.org/0000-0001-6469-7191
Bolívar Mendoza Morán[1] https://orcid.org/0000-0002-7680-7586

[1]Universidad Estatal Península de Santa Elena, UPSE - FACSISTEL | La Libertad - Ecuador | CP 240350

✉ lamaya@upse.edu.ec

http://dx.doi.org/10.26423/rctu.v9i1.489
Páginas: 48- 56

Resumen

En la era digital, las redes IP tradicionales aún son complicadas y tediosas de administrar. Existe dificultad para configurar la red de acuerdo con los procedimientos predefinidos y responder a las modificaciones de carga y fallas a través de la reconfiguración de la red. Están integradas verticalmente para complicar mucho más las cosas: los planos de control y datos están agrupados juntos, por esa razón las empresas de telecomunicaciones han creado la evolución en comunicación de datos, conocidas como SDN o redes manipuladas por software donde no es necesario el modelo jerárquico de tres capas clásico modelo OSI. El presente artículo tuvo como objetivo presentar el modelo SDN y sus características, se muestra una pequeña práctica desplegando una red virtual utilizando Virtual Box de cuatro máquinas virtuales, un emulador de red MiniNet y con un script Python para la MiniNet en las reglas de comunicación. Se pudo observar que la administración del SDN es fácil y sencilla a través de un lenguaje de programación, la definición de reglas va directamente entre equipos y no a través de las rutas de la red.

Palabras clave: SDN, OpenFlow, POX, MiniNet, Python, OSI

Abstract

In the digital times where almost everything is connected and available from anywhere, traditional IP networks are still complicated and tedious to manage. There is difficulty in configuring the network according to predefined procedures and responding to load changes and failures through network reconfiguration. They are vertically integrated to make things much more complicated: the control and data planes are grouped together, which is why telecommunications companies have created the evolution in data communication, known as SDN or software-driven networks where the classic three-layer hierarchical model of the OSI model is not necessary. The objective of this paper was to present the SDN model and its characteristics, a small practice is shown deploying a virtual network using Virtual Box of four virtual machines, a MiniNet network emulator and with a Python script for the MiniNet in the communication rules. It could be observed that the administration of the SDN is easy and simple through a programming language, the definition of rules goes directly between machines and not through the network routes.

Keywords: SDN, OpenFlow, POX, MiniNet, Python, OSI

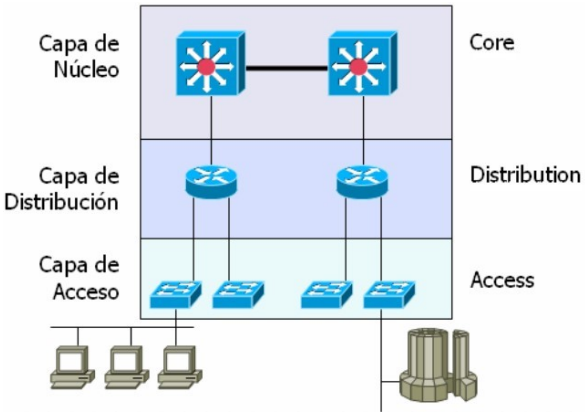
Recepción: 25 marzo 2022 | Aprobación: 12 mayo 2022 | Publicación: 30 junio 2022

1. Introducción

A medida que las redes crecen en tamaño y requisitos, navegar por los conmutadores de hardware se ha convertido en un desafío. La configuración manual de conmutadores de software de red individuales ha sido muy complicada y requiere mucho tiempo para las empresas que ejecutan sistemas altamente virtuales junto con redes grandes. Aquí es donde SDN entra en juego [1, 2].

SDN se puede describir como un enfoque de red que permite a los operadores de red configurar, rastrear, cambiar y controlar la operación de la red mediante programación a través de interfaces abiertas como el protocolo OpenFlow [3]. La SDN transforma la operación, gestión y configuración de las infraestructuras de red. La vista de SDN se basa en separar el plano de datos del plano de control [4]. SDN propone concentrar la inteligencia de red ,n un solo componente de red al distinguir el mecanismo de reenvío de paquetes de datos (plano de datos) del proceso de enrutamiento (plano de control) Figura 1. [5, 6].

En el caso de tener que dar permisos de comunicación entre equipos que están de extremo a extremo de la red y que las comunicaciones pasan a través de muchos switches, no sería un trabajo sencillo, ya que se debería analizar la red y determinar por cuales switches viaja la comunicación entre estos dos equipos, este tipo de trabajos tardaría algunas semanas para un administrador de red experimentado. Todo esto sucedería con el modelo de administración tradicional usando Cisco (Figura 1) con su sistema IOS (Internetwork Operating System) el cual es un sistema cerrado de código propietario.



Fuente: [7].

Figura 1: Modelo de Administración Tradicional de Redes.

Este modelo consiste en conmutadores de capa central que se conectan a conmutadores de capa de distribución (a veces llamados conmutadores de agregación), que a su vez se conectan a conmutadores de capa de acceso. La mayoría de la infraestructura de red todavía se presenta de esta manera hoy.

Ante un crecimiento de las redes informáticas y la necesidad de administración más sencilla aparecen las redes definidas por software (SDN) donde los cambios se realizan en un único sistema operativo conocido como controlador, en donde se definen las reglas y ya no se debe configurar cada equipo de red. Las SDN son la evolución de las redes tradicionales [8]. El objetivo del presente trabajo fue mostrar las características de esta nueva tecnología, que desplazará los diseños de red tradicionales.

2. Marco teórico

SDN

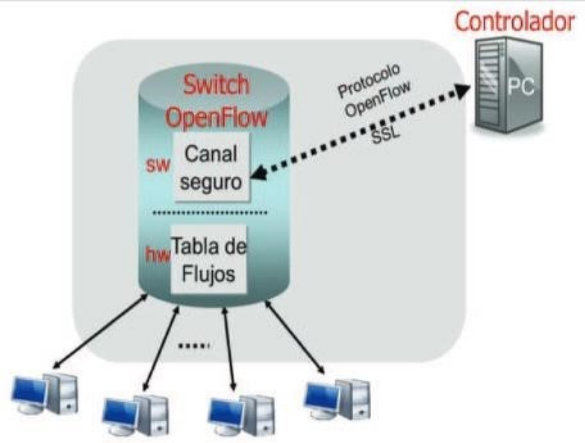
Las redes definidas por software (SDN) son una manera de abordar la creación de redes en la cual el control se desprende del hardware y se le da a una aplicación de software llamada controlador [9].

Las SDN tienen una arquitectura de red cuya característica fundamental es desacoplar físicamente el plano de control (inteligencia) del plano de datos, derivando el control a una computadora (controlador) y con dispositivos en las tareas de conmutación, aunque con limitada inteligencia, Figura 2. Esto facilita un mayor control y nivel de gestión sobre los dispositivos de red, flexibilizando el cambio de su funcionalidad y el de la red en general, empleando para ello un control centralizado, dado que el controlador tiene asociados muchos dispositivos [10].

Cuando un paquete llega a un conmutador en una red convencional, las reglas integradas al firmware propietario del conmutador le dicen al conmutador adónde transferir el paquete. El conmutador envía cada paquete al mismo destino por la misma trayectoria – y trata a todos los paquetes de la exacta misma manera. En la empresa, los conmutadores inteligentes diseñados con circuitos integrados de aplicación específica (ASICs, por sus siglas en inglés) son lo suficientemente sofisticados para reconocer diferentes tipos de paquetes y tratarlos de forma diferente, pero estos conmutadores pueden ser bastante costosos [9].

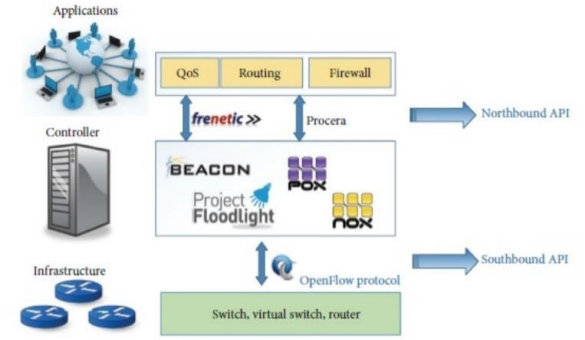
En una red definida por software, un administrador de red puede darle forma al tráfico desde una consola de control centralizada sin tener que tocar conmutadores

individuales [11]. El administrador puede cambiar cualquier regla de los conmutadores de red cuando sea necesario, dando o quitando prioridad, o hasta bloqueando tipos específicos de paquetes con un nivel de control muy detallado, Figura 3 [12].



Fuente: [13].

Figura 2: Arquitectura SDN.



Fuente: [14]

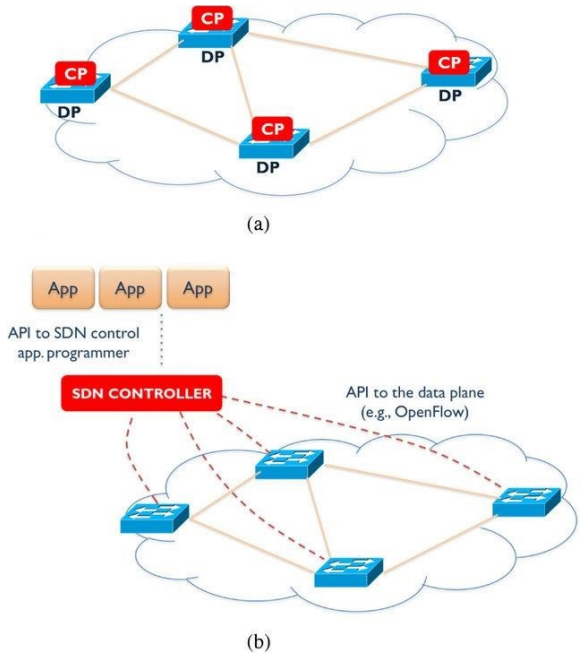
Figura 3: Arquitectura Paradigma SDN.

.

2.1. Comparación de arquitecturas de red tradicionales vs SDN

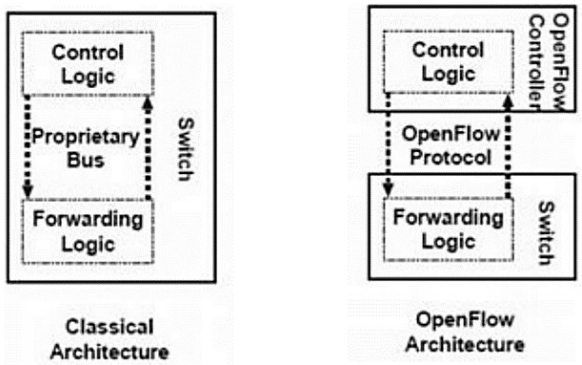
En las redes tradicionales (Figura 4 a), los planos de control (CP) y los planos de datos (DP) se ubican en los dispositivos para garantizar el control descentralizado de la red. En las SDN, los DP y los CP se separan con un controlador centralizado (Figura 4 b) que controla múltiples DP mientras se admite una API hacia el sur para los DP y una API hacia el norte para las aplicaciones SDN, (Figura 5, 6) [15].

Los dispositivos de red tradicionales utilizan firmware específico y propietario en el cual está predefinido como se van a tratar los paquetes que redireccionan por lo que son difíciles de integrar [16], Tabla 1 Figura 7 .



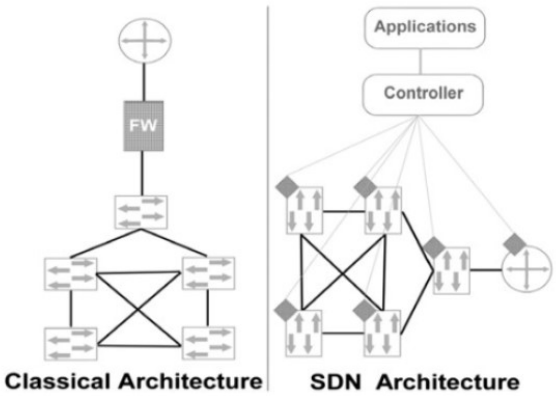
Fuente: [17].

Figura 4: Comparación entre Redes Tradicionales(a) vs SDN (b).



Fuente: [18].

Figura 5: Arquitectura clásica versus arquitectura SDN (16)



Fuente: [19].

Figura 6: Arquitectura clásica versus arquitectura SDN.

Tabla 1: Comparación de SDN vs Red Tradicional.

SDN	Tradicional
Es un enfoque de red virtual.	Enfoque de red convencional
Control centralizado.	Control distribuido
Es programable.	No es programable
Interfaz abierta	Interfaz cerrada
El plano de datos y el plano de control están desacoplados por software	El plano de datos y el plano de control se montan en el mismo plano
Puede priorizar y bloquear paquetes de red específicos.	Es compatible con la configuración estática/manual, por lo que lleva más tiempo. Conduce todos los paquetes de la misma manera sin soporte de priorización
Es fácil de programar según la necesidad	Es difícil volver a programar y reemplazar el programa existente según el uso
El costo de la red definida por software es bajo	El costo red tradicional es alto
La complejidad estructural es baja	La complejidad estructural es alta
La extensibilidad es alta	La extensibilidad es baja
Es fácil solucionar problemas e informar, ya que está controlado de forma centralizada.	Es difícil solucionar problemas e informar, ya que se distribuye de forma controlada
Su coste de mantenimiento es inferior al de la red tradicional.	El costo de mantenimiento de la red tradicional es más alto que el de SDN

Fuente: [19].

Fuente: [20].

Figura 7: Superior: Plano de datos y control distribuidos y co-localizados en una red tradicional; Inferior: Plano de control centralizado en una SDN.

2.2. Protocolos SDN

OpenFlow, es el primer protocolo para SDN [3]. Existen otros protocolos como: NETCONF, Border Gateway Protocol (BGP), Extensible Messaging and Presence Protocol (XMPP), Open vSwitch Database Management Protocol (OVSDB), MPLS Transport Profile (MPLS-TP).

2.3. Protocolo OPENFLOW

OpenFlow es un protocolo cuyo propósito es administrar dispositivos de red con el controlador SDN. Proporciona una interfaz para la programación directa de equipos de red (como conmutadores y enrutadores), tanto físicos como virtuales, haciendo que la red sea más dinámica y controlada. La característica principal de OpenFlow es el uso de flujos para identificar el tráfico de red. Estas corrientes están basadas en reglas predefinidas que pueden ser estáticamente o programadas dinámicamente utilizando el control SDN [21]. Openflow se encuentra entre los protocolos más populares para este propósito y generalmente describe cómo las aplicaciones pueden programar la tabla de flujo de diferentes conmutadores [3].

2.4. Controlador SDN

El controlador SDN actúa como un sistema operativo para la red (NOS: Network Operanting System). Tiene una capa superior es una capa de aplicación, también llamada plano de administración, esta capa es responsable de las tareas relacionadas con la administración y el reenvío de datos (por ejemplo, monitoreo del tráfico de datos, administración de movilidad, enrutamiento, seguridad, equilibrador de carga, etc.), para un flujo de tráfico de datos eficiente. En la arquitectura SDN, las API hacia el sur se utilizan como protocolo de comunicación entre el plano de control (es decir, el controlador SDN) y el plano de datos (es decir, conmutadores y enrutadores) de la red [22].

El panorama actual de controladores incluye los productos comerciales de VMware (vCloud / vSphere), Nicira (NVP), NEC (Trema), Big Switch Networks (Floodlight / BNC) y Juniper / Contrail. También incluye una serie de controladores de código abierto [23].

La arquitectura del controlador SDN consta de tres niveles:

1. Complementos y protocolos en dirección sur, que forman la capa de dispositivo de red.

2. Adaptación de servicios y formación de funciones de red. La capa de coordinación y control.
3. APIs hacia el norte y aplicaciones que forman la capa de aplicación [24].

Los controladores de una SDN actúan como un çerebro"de la red. Sirven en una capa intermedia de arquitectura SDN como se muestra en la Figura 4, que proporciona un esquema de control de red unificado. Son un punto de control estratégico que transmite información de control a los conmutadores o enrutadores subyacentes a través de las API hacia el sur y se ocupa de las aplicaciones de red o aplicaciones comerciales a través de las API hacia el norte. En la arquitectura SDN, un controlador SDN (también llamado controlador OpenFlow) usa el OFP (OpenFlow Protocols) para configurar los dispositivos de red subyacentes y elegir la mejor ruta para reenviar el tráfico de datos [5].

2.5. Lista de controladores basados en OPENFLOW

Varios programas de controlador SDN (NOS) de código abierto se están utilizando actualmente para implementar la arquitectura, estos controladores son NOX [25], POX [26], Floodlight [27], MuL [28], Trema [29], Beacon [30], Maestro [6], Ryu [31], etc. En la Tabla 2 se enumera una breve descripción de estos controladores OpenFlow de código abierto.

Tabla 2: Lista de controladores basados en OpenFlow.

Nombre del controlador	Lenguaje de programación	Tipo de licencia
NOX	C++	GPL
POX	Python	Apache
Floodlight	Java	Apache
MuL	C	GPLv2
Trema	C, Ruby	GPLv2
Beacon	Java	GPLv2
Maestro	Java	LGPL
Ryu	Python	Apache

Fuente: [32]

2.6. Controlador NOX

NOX **(https://github.com/noxrepo/)** fue el primer controlador OpenFlow escrito en C ++ y también proporciona una API para Python. Ha sido la base de muchos proyectos de investigación y desarrollo en la exploración inicial del espacio OpenFlow y SDN. NOX tiene dos líneas de desarrollo separadas:

NOX es un sistema operativo de red en el que se puede construir un controlador OpenFlow. En el lado de la red, NOX interactúa directamente con los dispositivos de red abiertos mediante las instrucciones de OpenFlow. En el lado de la aplicación, NOX expone una API de alto nivel para permitir que las aplicaciones de red intercambien mensajes OpenFlow con la red [33].

2.7. Controlador POX

POX es un controlador de código abierto para desarrollar aplicaciones SDN. El cual proporciona una manera eficiente de implementar el protocolo OpenFlow, que es el protocolo de comunicación de facto entre los controladores y los conmutadores. Con el controlador POX puede ejecutar diferentes aplicaciones como concentradoras, conmutadoras, equilibrador de carga y firewall. La herramienta de captura de paquetes Tcpdump se puede utilizar para capturar y ver los paquetes que fluyen entre el controlador POX y los dispositivos OpenFlow.

POX es una plataforma de desarrollo de código abierto para aplicaciones de control de redes definidas por software (SDN) basadas en Python, como los controladores OpenFlow SDN. POX, que permite un rápido desarrollo y creación de prototipos, es cada vez más utilizado que NOX, proyecto hermano [34].

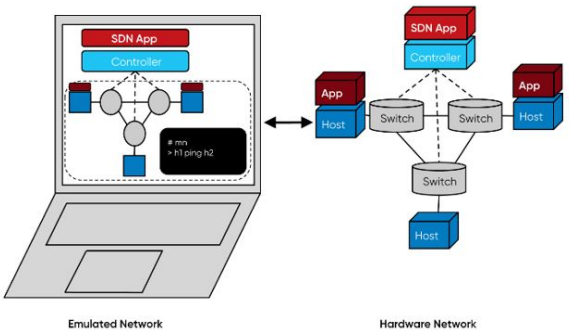
El controlador POX se puede utilizar para convertir dispositivos baratos en hubs, switches, enrutadores o middleboxes como firewall, o balanceadores de carga.

POX también es una gran herramienta para implementar y probar aplicaciones SDN [35]. Su gran fortaleza radica en que puede usarse con hardware real, en bancos de pruebas o con el emulador Mininet. El controlador POX tiene algunas características excelentes, pero no tiene interfaz GUI. Open Flow v1.0 es la versión más utilizada [36]. Open Flow versión 1.3 será la próxima versión que se supone que es ampliamente implementado en productos. POX solo admite v1.0. de OpenFlow [37].

2.8. MiniNet

MiniNet es un emulador de red que crea una red de hosts virtuales, conmutadores, controladores y enlaces (ver Figura 7). Los hosts MiniNet ejecutan software de red Linux estándar y sus conmutadores admiten OpenFlow para un enrutamiento personalizado altamente flexible y una red definida por software.

MiniNet admite investigación, desarrollo, aprendizaje, creación de prototipos, pruebas, depuración y cualquier otra tarea que pueda beneficiarse de tener una red experimental completa en una computadora portátil u otra PC [38].



Fuente: [39].

Figura 8: MiniNet

MiniNet ejecuta una colección de hosts finales, conmutadores, enrutadores y enlaces en un solo núcleo de Linux. Utiliza virtualización liviana para hacer que un solo sistema se vea como una red completa, ejecutando el mismo núcleo, sistema y código de usuario. Un host MiniNet se comporta como una máquina real; puede ingresar en él (si inicia sshd y conecta la red a su host) y ejecuta programas arbitrarios (incluido todo lo que esté instalado en el sistema Linux subyacente). Los programas que ejecuta pueden enviar paquetes a través de lo que parece una Ethernet real interfaz, con una velocidad de enlace y retraso dados. Los paquetes se procesan mediante lo que parece un conmutador, enrutador o middlebox Ethernet real, con una cantidad determinada de colas. Cuando dos programas, como un cliente y servidor iperf, se comunican a través de MiniNet, el rendimiento medido debe coincidir con el de dos máquinas nativas (más lentas) [40]

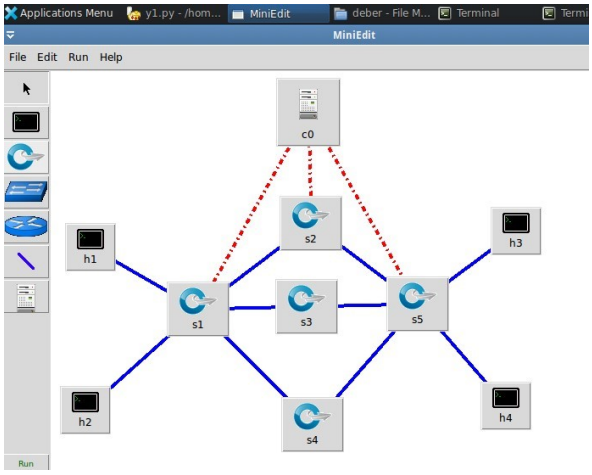
En resumen, los hosts virtuales, los conmutadores, los enlaces y los controladores de MiniNet son reales, simplemente se crean utilizando software en lugar de hardware, y en su mayor parte su comportamiento es similar al de los elementos de hardware discretos. Por lo general, es posible crear una red MiniNet que se parezca a una red de hardware, o una red de hardware que se parezca a una red MiniNet, y ejecutar el mismo código binario y aplicaciones en cualquier plataforma.

2.9. Python y MiniNet

MiniNet está escrito en Python y permite que los scripts de usuario basados en Python interactúen con él. Afortunadamente, Python es uno de los lenguajes de computadora más fáciles de entender, aprender y usar, debido a su sintaxis (principalmente) legible, similitud con otros lenguajes orientados a objetos y muchas bibliotecas útiles. Una vez que se reconcilie con sus peculiaridades (sangría significativa, uso obligatorio de uno mismo, verificación de errores de tiempo de ejecución, etc.), puede apreciarlo como una forma muy rápida (aunque a veces sucia) de crear scripts y códigos útiles [41].

2.10. MiniEdit

MiniEdit es una herramienta de interfaz gráfica para graficar redes para MiniNet (Figura 8)



Fuente: [42].

Figura 9: MiniNet.

3. Simulación de una red SDN usando MiniNet y Python

- Descargue de VM con Mininet
- Baje la imagen del sitio http://sdnhub.org/tutorials/sdn-tutorial-vm/
- Desde el siguiente link la versión exportada, incluye modo gráfico Ubuntu 14.04 y Mininet
- http://yuba.stanford.edu/~srini/tutorial/SDN_ tutorial _VM_32bit.ova la imagen viene configurada con modo grafico
- Luego importar la imagen EN VIRTUAL BOX por default (Figura 10).

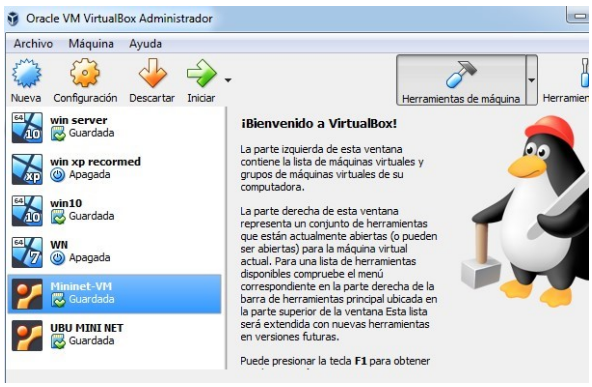


Figura 10: Ventana de Virtual Box

- Para tener internet en la máquina virtual, se debe poner el adaptador uno, conectado a NAT (Figura 11)

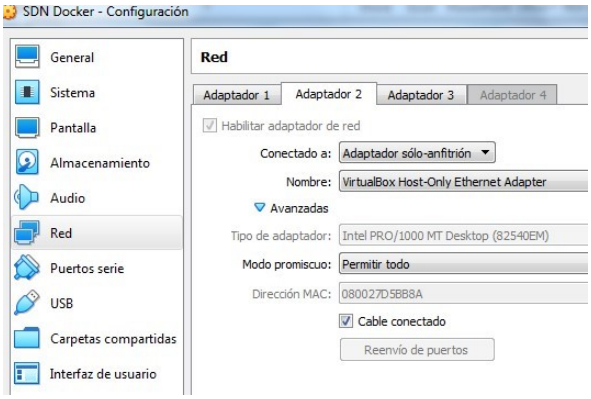


Figura 11: Ventana de Configuración

3.1. Diseño de la red usando MININET

Para diseñar la red se usa MiniEdit, dentro de MiniNet ver Figura 12:

- Ingresar a un terminal y teclear cd/MiniNet cd examples sudo./miniedit.py
- Aparecerá la herramienta y se puede dibujar y luego grabar con extensión .mn y .py para ejecutar y hacer pruebas.

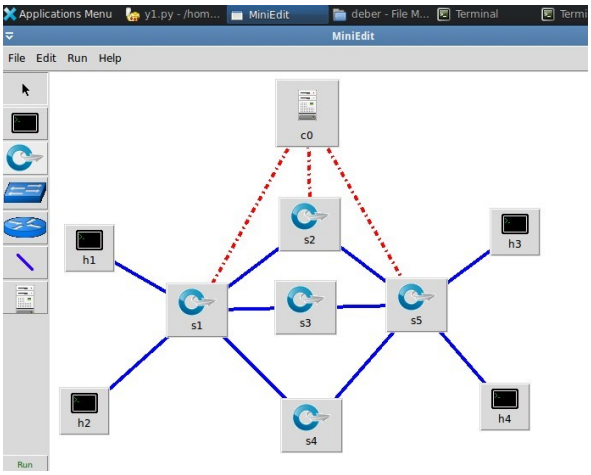


Figura 12: Diseño de una Red con MiniEdit

- En el gráfico se puede apreciar: h son los dispositivos (pc, smartphones, etc), son los switches, y el controlador.
- Al diseñar gráficamente la red con MiniEdit, se puede exportar el código Python generado automáticamente por MiniEdit
- Como MiniEdit no puede agregar direcciones, MAC se tiene que agregar código manualmente al archivo py.

info('*** Add hosts \n') h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None, mac="00:00:00:00:00:02") h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None, mac="00:00:00:00:00:01") h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None, mac="00:00:00:00:00:04") h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None, mac="00:00:00:00:00:03")

```
ubuntu@sdnhubvm:~/mininet/deber[10:43] (master)$ sudo python y1.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h2 h1 h4 h3
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
```

Figura 13: Ejecución del script p1.py

Ejecutar el comando Pingall para ver los caminos de comunicación permitidos entre host (Figura 14).

```
mininet> pingall
*** Ping: testing ping reachability
h2 -> h1 X h3
h1 -> h2 h4 h3
h4 -> X h1 h3
h3 -> h2 h1 h4
*** Results: 16% dropped (10/12 received)
mininet>
```

Figura 14: Ejecución del script p1.py

4. Conclusiones

- La evolución de las redes dio como nacimiento una SDN como una red sofisticada.
- SDN tiene la ventaja de poder hacer cambios en la programación del controlador en tiempo real, lo que es imposible en las redes tradicionales.
- La existencia de MiniNet como emulador usando POX, permite hacer prácticas de laboratorios muy similares a la realidad y es un software Open source bajo Linux, versus el software de CISCO bajo Windows.
- SDN tiene la posibilidad de ejecutar lenguajes de programación, amplía las posibilidades de manipulación de la red.
- El uso del lenguaje Python aumenta la cantidad de personas que pueden ser administradores de una red SDN.
- En una SDN, la definición de reglas va directo entre equipos y no por los caminos de red.

5. Referencias

1. CHOWDHURY, N. y BOUTABA, R. Network virtualization: state of the art and research challenges [En línea]. 2009. Disp. desde DOI: 10.1109/MCOM.2009.5183468.

2. ALBERT, E. y GÓMEZ, Z. SDN-Actors: Modeling and Verification of SDN Programs [En línea]. 2018. Disp. desde DOI: 10.1007/978-3-319-95582-7_33.
3. ANDRIOLI, L.; DA ROSA, R. y AUBIN, T. Analisando métodos y oportunidades em redes definidas por software (SDN) para otimizações de tráfego de dados [En línea]. 2017. Disp. desde DOI: 10.5335/rbca.v9i4.6948.
4. AZODOLMOLKY, S. y COKER, O. Software Defined Networks with OpenFlow - Second Edition [En línea]. 10 enero 2017. Disponible en: https://learning.oreilly.com/library/view/software-defined-networking-with/9781783984282/b0f4c4c4-dab7-4361-ad34-28e227ed8f15.xhtml.
5. BHOLEBAWA, I. y UPENA, D. Performance Analysis of SDN/OpenFlow Controllers: POX Versus Floodlight [En línea]. 2017. Disp. desde DOI: 10.1007/s11277-017-4939-z.
6. CAI, Z. y COX, A. *Maestro: A System for Scalable OpenFlow Control* [En línea]. Rice University, Tech. Rep., 2011. Disponible en: https://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf.
7. Modelo Jerárquico de redes escalables con Mikrotik [En línea]. Disponible en: https://mum.mikrotik.com/presentations/EC13/morocho.pdf.
8. DARABSEH, A. y AL-AYYOUB, M. M. SDSecurity: A Software Defined Security experimental framework [En línea]. 2015. Disp. desde DOI: 10.1109/ICCW.2015.7247453.
9. ROUSE, M. *Redes definidas por software (SDN)* [En línea]. 2012. Disponible en: https://searchdatacenter.techtarget.com/es/definicion/Redes-definidas-por-software-SDN.
10. BERNAL, I. y MEJÍA, D. *Las Redes Definidas por Software y los Desarrollos Sobre Esta Temática en la Escuela Politécnica Nacional* [En línea]. 2015. Disponible en: https://revistapolitecnica.epn.edu.ec/ojs2/index.php/revista_politecnica2/article/view/610.
11. AGUACÍA, Douglas. *Implementación De Un Módulo De Comunicaciones Openflow Para SmartNet* [En línea]. 2014. https://repository.javeriana.edu.co/bitstream/handle/10554/16509/AguaciaFiscoDouglasAlexander2015.pdf?sequence=1&rut=9106cc78f6cec044e9e65113674b66074abe\fb06c1a4412754deda65f7622c8f.
12. FIGUEROA, Noberto. *SDN - Redes Definidas por Software* [En línea]. 2008. Disponible en: https://www.academia.edu/8885994/SDN_Redes_definidas_por_Software&rut=c42cef0b9e931a5d0615e13d3b7bd27f6b53%5C%2081d90d1ff31f7a0c79f9e71c0b04.
13. CARLOS, Juan. *Implementación de un Prototipo de una Red Definida por Software (SDN) Empleando una Solución Basada en Hardware* [En línea]. 2014. Disponible en: https://www.researchgate.net/publication/301341630_Implementacion_de_un_Prototipo_de_una_Red_Definida_por_Software_SDN_Empleando_una_Solucion_Basada_en_Hardware.
14. VALDIVIESO, Leonardo. SDN: Evolution and Opportunities in the Development IoT Applications [En línea]. 2014. Disponible en: https://www.researchgate.net/publication/273876653_SDN_Evolution_and_Opportunities_in_the_Development_IoT_Applications.
15. QADIR, A. y AHAD, N. Building Programmable Wireless Networks: An Architectural Survey [En línea]. 2013. Disp. desde DOI: 10.1186/1687-1499-2014-172.
16. YAGÜES, P. Programación de redes SDN mediante el controlador POX [En línea]. 2015. Disponible en: http://repositorio.upct.es/bitstream/handle/10317/5254/tfg729.pdf?sequence=1&isAllowed=y.
17. QADIR, Junaid. Building Programmable Wireless Networks: An Architectural Survey [En línea]. 2013. Disp. desde DOI: 10.1186/1687-1499-2014-172.
18. YAGÜES, Pablo. Programación de redes SDN mediante el controlador POX [En línea]. 2015. Disponible en: http://repositorio.upct.es/bitstream/handle/10317/5254/.
19. HAJI, H. *Comparison of Software Defined Networking with Traditional Networking* [En línea]. 2021. Disp. desde DOI: 10.9734/ajrcos/2021/v9i230216.
20. BERNAL, Ivan. Las Redes Definidas por Software y los Desarrollos Sobre Esta Temática en la Escuela Politécnica Nacional [En línea]. 2016 [Consulta: 3 ene. 2016]. Disponible en: https://www.insst.es/documents/94886/175731/N%C3%BAmero+73+%28versi%C3%B3n+pdf%29/0f307c04-fc06-4bd3-8bc9-f82de8702656.
21. SKULYSH, M. The Method of Computing Organization in High Loaded SDN Controller System [En línea]. 2017. Disp. desde DOI: 10.1109/CADSM.2017.7916080.

22. BHOLEBAWA, I. y UPENA, D. Performance Analysis of SDN/OpenFlow Controllers: POX Versus Floodlight. Obtenido de Wireless Personal [En línea]. 2017. Disp. desde DOI: 10.1007/s11277-017-4939-z.
23. NADEAU, T. y GRAY, K. SDN: Software Defined Networks. Sebastopol, CA 95472.: O'Reilly Media [En línea]. 2013. Disponible en: https://learning.oreilly.com/library/view/sdn-software-defined/9781449342425/.
24. SAIKIA, D. MuL OpenFlow controller [En línea]. 2013. Disponible en: http://sourceforge.net/projects/mul/.
25. GUDE, Natasha.; KOPONEN, Teemu.; PETTIT, Justin.; PFAFF, Ben.; CASADO, Casado.; MCKEOWN, Nick. y SHENKER, Scott. Towards an operating system for networks [En línea]. 2008. Disponible en: http://pintos.benpfaff.org/papers/nox.pdf.
26. MCCAULEY, M. POX [En línea]. 2009. Disponible en: http://www.noxrepo.org/.
27. FLOODLIGHT. Floodlight is a Java-based OpenFlow controller [En línea]. 2012. Disponible en: http://floodlight.openflowhub.org/.
28. SAIKIA, D. MuL OpenFlow controller [En línea]. 2013. Disponible en: http://sourceforge.net/projects/mul/.
29. TAKAMIYA, Y. y KARANATSIOS, N. Trema OpenFlow controller framework [En línea]. 2012. Disponible en: https://github.com/trema/trema.
30. ERICKSON, D. D. The Beacon OpenFlow controller. *In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* [En línea]. 2013.
31. CORPORATION. Nippon Telegraph and Telephone. *Ryu network operating system* [En línea]. 2012. Disponible en: http://osrg.github.com/ryu/.
32. ISLAM, Md. Node to Node Performance Evaluation through RYU SDN Controller [En línea]. 2020. Disp. desde DOI: 10.1007/s11277-020-07060-4.
33. AZODOLMOLKY, S. y COKER, O. Software Defined Networks with OpenFlow - Second Edition [En línea]. 2017.
34. ROUSE, M. Redes definidas por software (SDN) [En línea]. 2012. Disponible en: https://searchdatacenter.techtarget.com/es/definicion/Redes-definidas-por-software-SDN.
35. MARÍN, Yanko. y FÉLIX, PALIZA. Evaluación Del Desempeño De Redes Definidas Por Software Basadas En La API P4runtime y El Protocolo OpenFlow. *Informática Habanaat: Havana, CubaVolume: IX Simposio Internacional de Telecomunicaciones* [En línea]. 2020. Disponible en: https://www.researchgate.net/publication/343725197_EVALUACION_DEL_DESEMPENO_DE_REDES_DEFINIDAS_POR_SOFTWARE_BASADAS_EN_LA_API_P4RUNTIME_Y_EL_PROTOCOLO_OPENFLOW/citations#fullTextFileContent.
36. TOMASI, Susana. Impacto del Procesamiento Electrónico de Datos en la Actuación del Contador Como Perito Judicial [En línea]. [Consulta: 16 jul. 2015]. Disponible en: https://www.academia.edu/es/14263920/IMPACTO_DEL_PROCESAMIENTO_ELECTR%5C%C3%5C%93NICO_DE_DATOS_EN_LA_ACTUACI%5C%C3%5C%93N_DEL_CONTADOR_COMO_PERITO_JUDICIAL&rut=722e210dfdb1371afa394c0c7f2e73d30f17%5C%5Cc02c256800cb1a5048299064f04c..
37. KAUR, S.; SINGH, J. y SINGH, N. Network Programmability Using POX Controller [En línea]. 2014. Disponible en: http://www.baburd.com.np/material/NG/Paper2-SDN-POX-Controller.pdf.
38. MININET.ORG. *MiniNet* [En línea]. 2019. Disponible en: http://mininet.org/overview/.
39. MININET. *MiniNet* [En línea]. 2021. Disponible en: https://opennetworking.org/mininet/.
40. LANTZ, B.; HANDIGOL, N.; HELLER, B. y JEYAKUMAR, V. *Introduction to Mininet* [En línea]. 2018. Disponible en: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet.
41. Introduction to Mininet [En línea]. 2018. Disponible en: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet.
42. MiniEdit 2.1.0.8 [En línea]. 2014. Disponible en: https://techandtrains.com/2014/02/07/miniedit-2-1-0-8/.