



UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍAS EN CIENCIAS APLICADAS
CARRERA DE INGENIERÍA EN ELECTRÓNICA Y REDES DE
COMUNICACIÓN

**“TESTBED PARA EL ESTUDIO DE LA TECNOLOGÍA DE REDES DEFINIDAS
POR SOFTWARE PARA LA CARRERA DE TELECOMUNICACIONES DE LA
UNIVERSIDAD TÉCNICA DEL NORTE”**

**TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE
INGENIERO EN ELECTRÓNICA Y REDES DE COMUNICACIÓN**

AUTOR: JUAN CARLOS TAPIA VIVERO
DIRECTOR: MSC. HERNÁN MAURICIO DOMÍNGUEZ LIMAICO

IBARRA-ECUADOR

2022



UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA
UNIVERSIDAD
TÉCNICA DEL NORTE

IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información.

DATOS DEL CONTACTO	
CÉDULA DE IDENTIDAD	1003434410
APELLIDOS Y NOMBRES	Tapia Vivero Juan Carlos
DIRECCIÓN	Río Lita y Río Aguarico, Los Ceibos.
E-MAIL	jctapiav@utn.edu.ec
TELÉFONO MÓVIL	0990338447
DATOS DE LA OBRA	
TEMA	“Testbed para el estudio de la tecnología de Redes Definidas por Software para la carrera de Telecomunicaciones de la Universidad Técnica del Norte”
AUTOR	Tapia Vivero Juan Carlos
FECHA	17/05/2022
PROGRAMA	PREGRADO <u>X</u> POSTGRADO <u> </u>
TÍTULO	Ingeniero en Electrónica y Redes de Comunicación
DIRECTOR	MSc. Hernán Mauricio Domínguez Limaico



UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

CONSTANCIA

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 6 días del mes de Julio del 2022

EL AUTOR:

Juan Carlos Tapia Vivero

C.I.: 1003434410



UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

MAGISTER MAURICIO DOMÍNGUEZ, DIRECTOR DEL PRESENTE TRABAJO DE TITULACIÓN CERTIFICA:

Que, el presente Trabajo de Titulación "TESTBED PARA EL ESTUDIO DE LA TECNOLOGÍA DE REDES DEFINIDAS POR SOFTWARE PARA LA CARRERA DE TELECOMUNICACIONES DE LA UNIVERSIDAD TÉCNICA DEL NORTE ". Ha sido desarrollado por el señor Tapia Vivero Juan Carlos, bajo mi supervisión.

Es todo en cuanto puedo certificar en honor de la verdad.

MSc. Mauricio Domínguez

DIRECTOR

DEDICATORIA

A mi familia y amigos.

AGRADECIMIENTOS

Quiero agradecer a mis padres por su apoyo incondicional y sobre todo paciencia durante todos mis años de formación académica, es gracias a sus enseñanzas y ejemplo que me he mantenido firme hasta culminar otra etapa de mi vida.

A las maravillosas personas que tuve la suerte de conocer durante mi paso por la universidad. A Adri, Joha, Mateo y Paul, amigos con quienes compartí de cerca cada uno de los buenos y malos momentos de la carrera.

Por último, agradecer a mi tutor del proyecto, quien dedicó su tiempo y guía durante el desarrollo de este trabajo de titulación.

Gracias a todos.

CONTENIDO

1.	Capítulo I: Planteamiento del Proyecto	17
1.1	Antecedentes.....	17
1.2	Definición del Problema.....	18
1.3	Justificación del Problema.....	19
1.4	Objetivos.....	20
1.4.1	Objetivo General.....	20
1.4.2	Objetivos Específicos	21
1.5	Alcance	21
2.	Capítulo II: Fundamento Teórico	23
2.1	Introducción a SDN	23
2.1.1	Arquitectura SDN	26
2.1.2	Plano de Control	29
2.1.3	Plano de Datos	31
2.2	Protocolo OpenFlow	33
2.2.1	Conmutador OpenFlow	36
2.3	Comunicación entre controlador y conmutador	38
2.3.1	Mensajes de controlador a conmutador	39
2.3.2	Mensajes Simétricos	41
2.3.3	Mensajes Asíncronos	41
2.4	Controladores SDN	42
2.4.1	POX	43
2.4.2	Floodlight	44
2.5	Mininet	45
2.5.1	Emulación sobre Mininet para SDN.....	47
2.5.1.1	Clases utilizadas en una red Mininet.....	48
2.5.1.2	Métodos utilizados en una red Mininet	49

3.	Capítulo III: Diseño e Implementación de la red SDN y Testbed.....	51
3.1	Diseño de la red y emulación de escenarios SDN.....	51
3.1.1	Diseño SDN del escenario 1	52
3.1.2	Diseño SDN del escenario 2.....	55
3.1.3	Creación de la red SDN sobre python	57
3.2	Ejecución de escenarios SDN.....	59
3.2.1	Escenario 1 - Pox.....	59
3.2.2	Escenario 1 - Floodlight	62
3.2.3	Escenario 2 - Pox.....	66
3.2.4	Escenario 2 - Floodlight	68
3.3	Implementación del Testbed SDN en el servidor FICA UTN.....	71
3.3.1	Situación actual del centro de datos	71
3.3.2	Instalación y configuración del Testbed SDN.....	74
3.4	Ambientes de práctica SDN a través del Testbed.....	85
3.4.1	Manejo de comandos mininet y tablas de flujo con el controlador POX.	85
3.4.2	Uso de listas de control de acceso mediante controlador Floodlight.	90
3.4.3	Ejecución de firewall y reglas de tráfico mediante controlador Floodlight.	94
4.	Capítulo IV: Obtención de Métricas y Análisis de Resultados	101
4.1	Evaluación de métricas en la red SDN	101
4.1.1	Latencia	101
4.1.2	Jitter	105
4.1.3	Throughput	110
4.2	Análisis de los ambientes de práctica SDN	119
	Conclusiones.....	123
	Recomendaciones	126
6.	Anexos.....	131

ÍNDICE DE FIGURAS

Figura 1 Arquitectura SDN.....	27
Figura 2 Interfaces del Plano de Control	29
Figura 3 Plano de datos de un dispositivo de reenvío	32
Figura 4 Arquitectura básica de OpenFlow Fuente	35
Figura 5 Componentes Conmutador OpenFlow	36
Figura 6 Flujo de paquetes en un conmutador OpenFlow	38
Figura 7 Diseño SDN del escenario 1	53
Figura 8 Topología lógica del diseño de red SDN del escenario 1	55
Figura 9 Topología lógica del diseño de red SDN del escenario 2	57
Figura 10 a) Código Python de la red SDN escenario 1 b) Código Python de la red SDN escenario 2	58
Figura 11 Configuración de Interfaz de red para direccionamiento estático.....	59
Figura 12 Ejecución escenario 1 con controlador POX	60
Figura 13 Conexión exitosa entre topología y controlador POX	60
Figura 14 Mensaje OFPT_HELLO de inicio de conexión	60
Figura 15 Prueba de conectividad controlador POX escenario 1	61
Figura 16 Mensajes OpenFlow de conexión exitosa	62
Figura 17 Ejecución escenario 1 con controlador Floodlight.....	63
Figura 18 Interfaz gráfica controlador Floodlight	63
Figura 19 Información página principal interfaz Floodlight	64
Figura 20 Prueba de conectividad controlador Floodlight escenario 1	65
Figura 21 Topología de activa de red SDN	66
Figura 22 Ejecución escenario 2 con controlador POX	66
Figura 23 Conexión exitosa entre topología y controlador POX	67
Figura 24 Mensaje OFPT_HELLO de inicio de conexión	67
Figura 25 Prueba de conectividad controlador POX escenario 2	67
Figura 26 Mensajes OpenFlow de conexión exitosa	68
Figura 27 Ejecución escenario 2 con controlador Floodlight.....	68
Figura 28 Interfaz gráfica Floodlight.....	69
Figura 29 Información página principal interfaz Floodlight	69
Figura 30 Pruebas de conectividad controlador Floodlight escenario 2.....	70
Figura 31 Topología activa de la red SDN	71

Figura 32 Infraestructura del centro de datos FICA	72
Figura 33 Topología lógica del centro de datos FICA	73
Figura 34 Máquina virtual 110 juantapia creada en PVE2.....	74
Figura 35 Archivo .ova en nuestra máquina.....	75
Figura 36 Archivo .ova copiado a la máquina Proxmox	75
Figura 37 Archivo .ova se mueve al directorio /var/lib/vz	75
Figura 38 Archivo .ova descomprimido	75
Figura 39 Eliminación disco duro existente	76
Figura 40 Importación disco duro Testbed-001.vmdk	76
Figura 41 Disco sin usar	77
Figura 42 Configuración de disco sin usar	77
Figura 43 Verificación de disco duro y encendido de MV	78
Figura 44 Arranque a partir de BIOS	78
Figura 45 MV encendida sin errores	79
Figura 46 Direccinamiento IPv4	79
Figura 47 Acceso router MikroTik 1100.....	80
Figura 48 Configuración de NAT.....	81
Figura 49 Configuraciones generales de NAT primera regla.....	81
Figura 50 Configuraciones de acción de NAT	82
Figura 51 Verificación de Nateo	82
Figura 52 Configuraciones generales de NAT segunda regla	83
Figura 53 Configuraciones de acción de NAT	83
Figura 54 Ingreso al Testbed SDN mediante SSH	84
Figura 55 Testbed SDN funcionando correctamente	84
Figura 56 Nodos de la topología SDN	86
Figura 57 Información detallada de cada nodo	86
Figura 58 Enlaces activos de la red SDN	86
Figura 59 Información de la interfaz red del Host1.....	87
Figura 60 Tablas de flujo vacías de los conmutadores s1, s2 y s3.	87
Figura 61 Tabla de flujo del conmutador s1	88
Figura 62 Eliminación de la información de una tabla de flujo	89
Figura 63 Entradas de tablas de flujo en estado drop.	89
Figura 64 Conexión exitosa a partir de nuevas tablas de flujo.....	90
Figura 65 ACL que permite el tráfico entre 2 dispositivos	91

Figura 66 Verificación de conexión	91
Figura 67 Ingreso de regla ACL.....	91
Figura 68 Verificación de funcionamiento de regla	92
Figura 69 ACL para denegar el tráfico entre Host 2 y Host 3.....	92
Figura 70 Verificación de ACL	92
Figura 71 ACL's agregadas.....	93
Figura 72 Eliminación de ACL	93
Figura 73 Verificación de ACLS's activas	94
Figura 74 Eliminación de todas las reglas ACL.....	94
Figura 75 Estado de Firewall.....	95
Figura 76 Firewall activado.....	95
Figura 77 Verificación de conectividad.....	95
Figura 78 Información de conmutadores a partir de la interfaz Floodlight.....	96
Figura 79 Regla para conexión a través de un solo conmutador	96
Figura 80 Verificación de regla	96
Figura 81 Regla que permite una conexión ICMP a partir de direcciones IP.	97
Figura 82 Verificación de la regla aplicada.....	98
Figura 83 Nueva regla para conexión entre Host 1 y 4	98
Figura 84 Verificación de nueva regla	98
Figura 85 Direcciones MAC de hosts	99
Figura 86 Regla de comunicación a través de direcciones MAC.....	99
Figura 87 Verificación de regla	100
Figura 88 Comparación tiempos de latencia media entre controladores escenario 1... 103	103
Figura 89 Comparación tiempos de latencia media entre controladores escenario 1... 103	103
Figura 90 Comparación de tiempos de latencia máxima entre controladores escenario 1	104
Figura 91 Comparación de tiempos de latencia máxima entre controladores escenario 2	104
Figura 92 Comparación de jitter entre controladores en el escenario 1 SDN	109
Figura 93 Comparación de jitter entre controladores en el escenario 2 SDN	109
Figura 94 Ejecución de iPerf en los host de la red SDN	111
Figura 95 Iniciando nodo h2 como servidor TCP	111
Figura 96 Ejecución de transmisión de cada nodo para el escenario 1 en los controladores SDN.....	111

Figura 97 Ejecución de transmisión de cada nodo para el escenario 2 en los controladores SDN	112
Figura 98 Cantidad máxima de datos transferidos entre nodos en el escenario 1	114
Figura 99 Cantidad máxima de datos transferidos entre nodos en el escenario 2	114
Figura 100 Ancho de banda entre nodos de la red para el escenario 1	115
Figura 101 Ancho de banda entre nodos de la red para el escenario 2	116
Figura 102 Throughput entre nodos de la red para el escenario 1	118
Figura 103 Throughput entre nodos de la red para el escenario 2	118
Figura 104 Filtrado de paquetes de la dirección IPv4 192.168.10.2	120
Figura 105 Firewall en una red tradicional	120
Figura 106 Firewall en una red SDN	121
Figura 107 Mensajes de request por parte del Host 4 hacia Host 1 sin respuesta	122
Figura 108 Mensajes de request por parte del Host 1 hacia Host 2 con respuesta	122
Figura 109 Instalación del paquete git	131
Figura 110 Descarga paquete Mininet desde GitHub	132
Figura 111 Seleccionando la versión de Mininet a ejecutar	132
Figura 112 Instalación Mininet	133
Figura 113 Ejecución Mininet	133
Figura 114 Verificación de carpeta Pox	134
Figura 115 Inicialización del controlador POX	134
Figura 116 Ejecución de Mininet con POX	135
Figura 117 Controlador POX conectado correctamente	135
Figura 118 Inicialización de Floodlight	136
Figura 119 Ejecución de la topología SDN con el controlador Floodlight	136
Figura 120 Interfaz gráfica Floodlight	136
Figura 121 Ejemplo topología Sencilla	137
Figura 122 Prueba de conectividad entre hosts	137
Figura 123 Ejemplo topología linear	138
Figura 124 Prueba de conectividad	138
Figura 125 Ejemplo topología Árbol	139
Figura 126 Prueba de conectividad	139
Figura 127 Interfaz Miniedit	140
Figura 128 Parámetros modificables del enlace o link	140
Figura 129 Topología en Mininet a partir de Miniedit	141

Figura 130 Prueba de conectividad	141
Figura 131 Inicio topología básica personalizada	142
Figura 132 Creación de Switches, hosts y links	142
Figura 133 Ejecución de la función	143
Figura 134 Topología personalizada a partir de Python.....	143
Figura 135 Prueba de conectividad topología personalizada	144
Figura 136 Ejecución de tcpdump para captura de paquetes de forma remota	144
Figura 137 Detención de captura de paquetes	145
Figura 138 Conexión SFTP para copia de archivos de forma remota.....	145
Figura 139 Paquete capturados de forma remota en Wireshark.....	146

ÍNDICE DE TABLAS

Tabla 1 Campos de paquetes usados para comparar con entradas de flujo.....	32
Tabla 2 Requerimientos de Máquina Virtual para el Escenario 1.....	54
Tabla 3 Plan de Direccionamiento IPv4 escenario 1.....	54
Tabla 4 Plan de Direccionamiento IPv4 escenario 2.....	56
Tabla 5 Tiempos de latencia mínimos, medios y máximos del escenario 1.....	102
Tabla 6 Tiempos de latencia mínimos, medios y máximos del escenario 2.....	102
Tabla 7 Tiempo de envío de paquetes en cada secuencia en el escenario 1.....	106
Tabla 8 Tiempo de envío de paquetes en cada secuencia en el escenario 2.....	107
Tabla 9 Jitter del escenario 1 SDN.....	108
Tabla 10 Jitter del escenario 2 SDN.....	108
Tabla 11 Valores de throughput en los controlares SDN del escenario 1.....	112
Tabla 12 Valores de throughput en los controlares SDN del escenario 2.....	113
Tabla 13 Valores de throughput en los controlares SDN del escenario 1 para 15 segundos.	117
Tabla 14 Valores de throughput en los controlares SDN del escenario 2 para 15 segundos.	117

ÍNDICE DE ECUACIONES

Ecuación 1 Ecuación de cálculo de jitter.....	105
Ecuación 2 Cálculo de jitter en el escenario 1 controlador POX ping Host 1-Host 2..	105
Ecuación 3 Ecuación de cálculo del throughput.....	110
Ecuación 4 Cálculo del jitter del escenario 1 controlador Floodlight transmisión Host 1- Host 2.....	116

RESUMEN

El presente trabajo describe el diseño e implementación de un banco de pruebas (Testbed), con el fin de entender el comportamiento y aplicación de las Redes Definidas por software versus las redes tradicionales. Para lo cual, se ha tomado tres métricas de referencia a partir del RFC 2544 para su posterior análisis en la red SDN, así como se ha propuesto tres ambientes de práctica los cuales cumplen con el objetivo de proporcionar herramientas que permitirán a los estudiantes de la carrera el conocer, entrenar e investigar acerca de la tecnología SDN.

Como punto de inicio se realiza una investigación bibliográfica de la tecnología SDN, su arquitectura básica, comunicación, controladores y protocolo OpenFlow. Además, se presenta el software de emulación Mininet como una herramienta para el uso e investigación de esta tecnología.

A continuación, el trabajo se enfoca en la creación, verificación y emulación de cada uno de los escenarios SDN planteados a través de los controladores Pox y Floodlight. Además, se presenta paso a paso la implementación, configuración y ejecución del Testbed SDN en el centro de datos de la carrera, donde posteriormente se procederá con la aplicación de tres ambientes de práctica, cada uno con planteamiento y objetivos distintos.

Para finalmente, analizar el funcionamiento del Testbed SDN a partir de la evaluación de las métricas latencia, jitter y throughput de cada escenario de prueba realizado. Además, se realiza un estudio de los resultados obtenidos en cada uno de los ambientes de práctica ejecutados para comparar el funcionamiento de estos en una red SDN versus una red tradicional.

ABSTRACT

This work describes the design and implementation of a Testbed SDN, the point of this is understand the behavior and application of the Software Defined Networks against Traditional Networks. For which, three reference metrics have been taken from RFC 2544 for subsequent analysis in the SDN network, as well as three practice environments which meet the objective of providing tools that will allow students of the career to know, train and research about SDN technology.

As a starting point, a bibliographical investigation of SDN technology, its basic architecture, communication, controllers and OpenFlow protocol is carried out. In addition, the Mininet emulation software is presented as a tool for the use and investigation of this technology.

Next, the work focuses on the creation, verification and emulation of each of the SDN scenarios proposed through the Pox and Floodlight controllers. In addition, the implementation, configuration and execution of the Testbed SDN in the data center of the career is presented step by step, where three practice environments will subsequently be applied, each with a different approach and objectives.

To finally, analyze the operation of the Testbed SDN from the evaluation of the latency, jitter and throughput metrics of each test scenario carried out. In addition, a study of the results obtained in each of the executed practice environments is carried out to compare their operation in an SDN network versus a traditional network.

1. CAPÍTULO I: PLANTEAMIENTO DEL PROYECTO

Este capítulo detalla las bases para el desarrollo del presente trabajo de titulación, siendo éstos: el tema, la problemática, los objetivos, el alcance y la justificación, con el fin de denotar la importancia de la realización de este trabajo.

1.1 ANTECEDENTES

En las redes tradicionales siempre ha existido una barrera en los equipos de red, esta trata de que en muchos casos es necesario trabajar con la misma marca de hardware en todos los equipos, por lo que el paradigma de las SDN provee una opción para extender la infraestructura en un dominio virtual quitando estas ataduras mientras se mantiene la integración con el hardware, a partir de esto y según un estudio de (Muhammad , Samineni, & Robertson, 2016) incluso es posible construir un prototipo portable de un Testbed SDN el cual se presenta como una solución de bajo costo de diseño que además puede ser levantado funcionalmente sin complejidad, por lo que se puede entender que SDN va más allá de ser sólo un dominio virtual de red.

Una de las ventajas de SDN es que se integra naturalmente a diferentes instancias de las redes, por ejemplo, en redes SAN un estudio realizado por (Sadov, Grudin, Vlasov, & Titov, 2015) demostró que el desarrollo de un Testbed basado en OpenFlow puede ser usado para probar configuraciones dinámicas de la red y para los ajustes de parámetros para diferentes tipos de tráfico. De esto se puede aprender que un Testbed es apto no sólo para una red pequeña, sino que también lo es incluso para una red más compleja.

Lo antes mencionado prueba que el uso y aplicación de redes SDN son numerosas, pero lo más importante que vale recalcar es el enfoque que tiene un prototipo de Testbed de esta tecnología ya que estos prototipos proveen una herramienta para la investigación de arquitecturas de red donde

se puede establecer las diferentes ventajas y desventajas que tienen las redes SDN con las redes tradicionales

En el contexto local, la Universidad Técnica del Norte a la fecha de presentación del anteproyecto no cuenta con trabajos de grado relacionados acerca de SDN, por lo cual esto marca una línea base del estudio de la tecnología de Redes Definidas por Software en la carrera de Telecomunicaciones.

1.2 DEFINICIÓN DEL PROBLEMA

Una solución emergente, las redes definidas por software (SDN), promete cambiar el panorama de la topología y gestión de redes tradicionales. Tanto los investigadores como los primeros usuarios en el estudio de redes necesitan instalaciones de prueba SDN adecuadas para sus estudios, pero las opciones son limitadas. La industria ha ido respondiendo lentamente con soporte para SDN, pero tiene un costo elevado para muchos entornos de prueba con un solo conmutador SDN que cuesta miles de dólares (Ramanathan, Kannan, Mirkovic, & Sklower, 2017). Existen varios bancos de prueba SDN independientes los cuales pueden ser utilizados, pero el problema radica en que siempre es necesaria una membresía o un pago adicional para el uso de estos, además que poseen pruebas ya establecidas por lo que una investigación flexible sería imposible. Una alternativa gratuita e indispensable a una SDN de hardware dedicada es utilizar herramientas de emulación de red. Estas herramientas de software se utilizan ampliamente y son invaluable para la investigación de SDN (Alcorn & Melton, 2017).

La carrera CIERCOM/CITEL de la Universidad Técnica del Norte no cuenta con una herramienta para realizar pruebas de funcionamiento de la tecnología SDN. La implementación de un prototipo de banco de pruebas (Testbed), permitirá estudiar el comportamiento de una red de

manera dinámica, lo cual es cada vez más necesario debido al avance tecnológico que presenta el networking, es por ello que el estudio de estas tecnologías se vuelve más fundamental. Tomando en cuenta estas consideraciones se evidencia la carencia de una arquitectura de pruebas para el estudio de la operación y funcionamiento las Redes Definidas por Software en la carrera.

El presente proyecto realizará una investigación y diseño de la arquitectura de redes definidas por software (SDN), protocolo OpenFlow, herramienta de simulación Mininet y controladores SDN. Adicionalmente se desplegará un banco de pruebas(Testbed) SDN el cual contendrá un número específico de nodos donde además se desplegarán varias aplicaciones las cuales se reflejarán en el entorno de simulación Mininet. Posteriormente, se implementará la estructura física simulada la cual necesita equipos de red los cuales sean aptos para la utilización del protocolo OpenFlow. Finalmente se evaluará el funcionamiento de la red y las aplicaciones integradas mediante métricas como tasas de transferencia, Jitter o retardo, latencia, uso del ancho de banda y pérdida o errores en la transmisión de paquetes

1.3 JUSTIFICACIÓN DEL PROBLEMA

La carrera CIERCOM/CITEL de la Universidad Técnica del Norte no posee actualmente un prototipo para el estudio de redes definidas por software (SDN), por lo que se ha considerado en el diseño de un banco de pruebas (Testbed) SDN, el cual servirá de base para el estudio de esta tecnología las cuales tratan de solucionar las limitaciones que presentan las redes tradicionales de hoy en día como son dependencias de fabricantes, soporte de nuevos servicios, configuración individual de elementos de red, entre otros

La importancia del estudio y práctica de las SDN es que estas migran el plano de control de los elementos de la red a un controlador central el cual gestionará todos estos recursos, no sólo

facilitando el control y gestión de la red al administrador de esta, sino que permitirá modificar el comportamiento de la red dinámicamente mediante software.

La carrera, al poseer un banco de pruebas (Testbed) de estas características solo necesitará de elementos de red que soporten el protocolo OpenFlow y sus configuraciones para realizar prácticas de laboratorio o investigaciones que permitan entender el funcionamiento y operación de la tecnología SDN.

La implementación de un Testbed SDN lo que busca es garantizar una herramienta a disposición de estudiantes y docentes la cual permita e impulse el estudio y desarrollo de investigaciones relacionadas con esta tecnología, evitando sistemas de redes cerrados aprovechando la utilización de software libre, además que este prototipo quedará implementado en la carrera y servirá como línea base para futuras investigaciones de la tecnología de las Redes Definidas por Software.

1.4 OBJETIVOS

1.4.1 OBJETIVO GENERAL

Diseñar un Banco de Pruebas (Testbed) para la enseñanza e investigación de la tecnología de Redes Definidas por Software (SDN) en la carrera CIERCOM/CITEL de la Universidad Técnica del Norte que permita el análisis del comportamiento de una red de datos tradicional versus una SDN.

1.4.2 OBJETIVOS ESPECÍFICOS

- Estudiar la tecnología de Redes Definidas por Software (SDN) a fin de elaborar un estado del arte de la operación y funcionamiento de SDN.
- Diseñar e implementar el Testbed SDN para un ambiente académico de enseñanza e investigación mediante el uso de herramientas que permitan la emulación de una red definida por software.
- Verificar la operatividad y funcionamiento tanto del prototipo como de emulación dentro un ambiente controlado.
- Evaluar la funcionalidad del Testbed implementado en la carrera CIERCOM/CITEL de la Universidad técnica del Norte.
- Proponer un banco de pruebas (Testbed) sobre la operación y manejo de una SDN con el fin de reforzar los conocimientos teóricos acerca de la tecnología.

1.5 ALCANCE

El presente proyecto, con el objetivo de diseñar y evaluar el Testbed SDN propuesto ha tomado como base la metodología de desarrollo Prince 2. Esta fundamentalmente se enfoca en etapas que se centran en la obtención de un resultado en concreto. Esta metodología sigue las etapas de iniciación, planificación, control de fases, administración de entregables y un cierre.

La primera etapa se enfoca en la investigación documental que permita a partir de referencias bibliográficas conocer el funcionamiento de las redes definidas por software, de los diferentes controladores SDN que existen comercialmente y sobre el protocolo OpenFlow de tal manera que podamos obtener un estado del arte sobre Redes Definidas por Software. A partir de

esta etapa, y partiendo de las características necesarias para el proyecto se procederá al diseño de un esquema de banco de pruebas SDN y sus elementos necesarios; inmediatamente se tiene una fase de entrenamiento donde se aplicará la teoría para realizar prácticas de ambientación y funcionamiento del protocolo OpenFlow y como se controla un conmutador a través de controladores SDN. Además, nos centraremos en una fase de diseño que consiste en emular una topología de red en el software de emulación Mininet, donde posteriormente se realizará pruebas de funcionamiento y se entenderá de una manera abstracta los diferentes problemas iniciales que pueden existir en este diseño.

Finalmente, en la etapa de cierre el ambiente emulado se lo implementará en un ambiente real, el cual ya vendrá a formar nuestro Testbed, y en donde se realizarán diferentes pruebas de funcionamiento acompañadas de la medición de al menos 2 diferentes métricas como son la latencia y el retardo que nos ayudarán a entender el comportamiento y aplicación de las Redes Definidas por Software versus las redes tradicionales.

2. CAPÍTULO II: FUNDAMENTO TEÓRICO

En este capítulo se examina acerca la tecnología SDN, su arquitectura básica, comunicación, controladores y protocolo OpenFlow. Además, se presenta el software de emulación Mininet como una herramienta para el uso e investigación de esta tecnología.

2.1 INTRODUCCIÓN A SDN

El networking tradicional, que en su infraestructura física se compone de conmutadores, routers y otros dispositivos los cuales crean conexiones y manejan la red, se ha caracterizado porque cada equipo de red tiene localmente un plano de control y un plano de datos. Estos planos son los encargados de realizar una separación lógica en el tráfico de red destinado a los diferentes servicios y equipos, para así gestionar, mantener y modificar el estado de la red; en otras palabras, el plano de datos se refiere al manejo del reenvío de datos mientras que el plano de control se encarga de generar las tablas utilizadas para este proceso.

Por otra parte, de acuerdo con (IBM Cloud Education, 2021), las redes tradicionales presentan debilidades tanto en su operación como en su administración frente a la presencia y comportamiento de nuevas tecnologías de comunicaciones, a continuación, se listan las principales debilidades:

- Una red tradicional no permite establecer reglas que funcionen dinámicamente para el flujo de datos, sino que estas se presentan de manera estática en cada equipo conmutador o enrutador, por lo que una red tradicional carece de flexibilidad al momento de gestionar el tráfico en la red.

- Una red tradicional está sostenida por las características de hardware de un equipo, esto quiere decir que cada dispositivo de red sólo posee la capacidad de realizar funciones predeterminadas que le fueron configuradas de fábrica. Esta debilidad se hace notoria cuando varios equipos de una misma red realizan funciones similares, donde una centralización permitiera no sólo disminuir la adquisición de equipos, sino que su gestión sería mucho más fácil para el administrador de la red.

De la misma manera que IBM, organizaciones como Open Networking Foundation (ONF), a partir de investigaciones realizadas en el año 2012 (ONF, Software-Defined Networking: The New Norm for Networks, 2012) ha llegado a determinar diferentes limitantes que se pueden encontrar en una arquitectura de red tradicional, las cuales son:

- **Arquitectura estática:** Para aumentar, quitar o cambiar algún equipo de red se deben configurar múltiples conmutadores, routers, firewalls, además de ACL (Listas de Control de Acceso), VLANs (Red de Área Local Virtual) o QoS (Calidad de Servicio) para que funcionen correctamente con nuevas topologías de red implementadas.
- **Políticas inconsistentes:** Al momento de implementar políticas de seguridad los administradores de red usualmente deben configurar los cambios en cada equipo que lo necesita, por lo que, si una red es extensa, estas configuraciones tomarían mucho tiempo en realizarse.
- **Dependencia de proveedores:** Existen muchas marcas en el mercado que pueden no funcionar adecuadamente con equipos que no sean del mismo fabricante (CISCO, Arista, Huawei, entre otros), por lo que existe una atadura para trabajar siempre con una gama de

dispositivos de un mismo proveedor y de las ofertas que este ofrece para cada necesidad tecnológica que presente un administrador de red.

Con esta visión general sobre las limitantes de una red tradicional, y acorde a la Open Data Center Alliance (por sus siglas en inglés ODCA) (Stallings, 2016) se puede tener una lista de requerimientos que debería poseer una red moderna, en donde la autonomía funciona como base, pero que además trata de generalizar las diferentes necesidades que puede presentar una red en la actualidad. Estos requerimientos son:

- **Adaptabilidad:** La capacidad de responder dinámicamente a partir de las diferentes condiciones de red y servicios.
- **Automatización:** Los cambios de una política deben ser propagados automáticamente para evitar errores de configuraciones manuales.
- **Mantenimiento:** La adición de nuevas características debe ser lo más imperceptible posible y que no interrumpa ninguna operación.
- **Seguridad integrada:** Aplicaciones de red deben integrar seguridad como un servicio central y no como una solución extra del servicio.
- **Escalabilidad en demanda:** Debe existir la posibilidad de escalar o disminuir la red y sus servicios según la demanda.

Tomando en consideración los diferentes requerimientos presentados por la ODCA para redes modernas, y las limitantes que IBM y ONF presentan sobre arquitecturas tradicionales, es fácil entender como el apareamiento de las SDN ha tomado notoriedad conforme transcurren los años, y como su capacidad de control y gestión total desde un punto central o su ventaja al poseer

una función de programabilidad sobre equipos a diferencia de redes tradicionales han ubicado a esta tecnología como el siguiente paso a seguir al momento de implementar una red (Shaghghi, Mohamed, & Rajkumar , 2018)

Como resultado, y tomando como referencia a (IRTF, 2015), se define a las SDN como un conjunto de técnicas utilizadas para facilitar el diseño, entrega y funcionamiento de los servicios de red en una forma determinista, dinámica y escalable. Donde, la capacidad de control sobre la estructura, diseño y entrega de servicios de red además del reenvío de tráfico, se produce debido a la búsqueda de una opción que permita optimizar los recursos utilizados en estos procesos.

2.1.1 ARQUITECTURA SDN

La arquitectura de las SDN, y en referencia a (Shaghghi, Mohamed, & Rajkumar , 2018) se compone de tres capas principales: el plano de datos, el plano de control y el plano de aplicación. Cada plano posee sus propias funciones, pero además existen componentes que siempre están presentes en una implementación de SDN, como lo son la API en dirección sur (Southbound), el controlador SDN, la API en dirección norte (Northbound) y las aplicaciones de red.

El concepto detrás de SDN es permitir que los desarrolladores y administradores de red tengan un completo control sobre los diferentes equipos de red. El enfoque SDN divide la función de conmutación entre un plano de datos y un plano de control. El plano de datos es simplemente responsable de reenviar paquetes, mientras que el plano de control proporciona la "inteligencia" para diseñar rutas, establecer parámetros de política de enrutamiento y prioridad para cumplir con los requisitos de QoS y QoE.

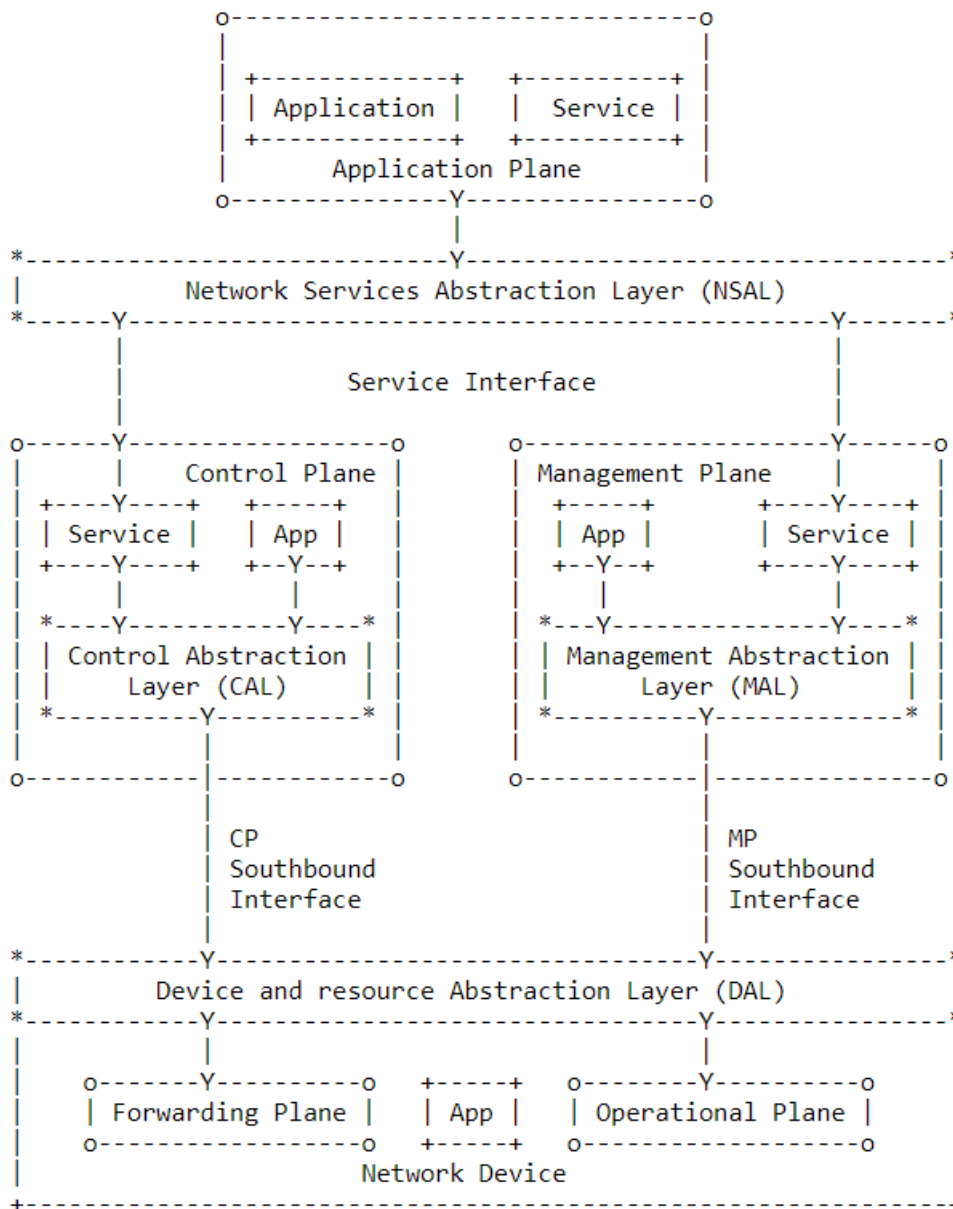


Figure 1: SDN Layer Architecture

Figura 1 Arquitectura SDN
Fuente: (IRTF, 2015)

La figura 1 muestra el enfoque que proporciona una red SDN. El plano de datos o plano de reenvío (Forwarding Plane) está constituido de switches o conmutadores físicos y virtuales; en ambos casos, estos switches son los responsables de reenviar paquetes basado en instrucciones recibidas por el plano de control (Control Plane), donde, el estado y funcionamiento de estos

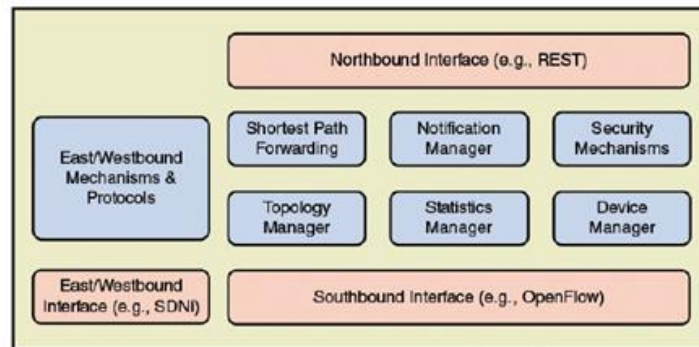
dispositivos es responsabilidad del plano de operación (Operational Plane). Sin embargo, cada switch debe implementar un modelo de reenvío de paquetes que esté abierto a los controladores SDN, los cuales son el cerebro principal de la red. Este modelo se define en términos de una interfaz de programación de aplicaciones (API: Application Programming Interface) abierta entre el plano de control y el plano de datos.

Los controladores SDN se pueden implementar directamente en un servidor físico o en un servidor virtual. Se utiliza OpenFlow (protocolo de control SDN) o alguna otra API abierta para controlar los conmutadores en el plano de datos. Además, los controladores utilizan información sobre la capacidad y la demanda obtenida del equipo de red a través del cual fluye el tráfico. Los controladores SDN también exponen las API, que permiten a los desarrolladores y administradores de red implementar una amplia gama de aplicaciones de red listas para usar y ser personalizadas, muchas de las cuales no eran factibles antes de la llegada de SDN. Hasta el momento, no hay una API estandarizada ni un consenso sobre una API abierta.

En el plano de aplicación (Application Plane) hay una variedad de aplicaciones que interactúan con los controladores SDN. Las aplicaciones SDN son programas que pueden utilizar una vista abstracta de la red para sus objetivos de toma de decisiones. Estas aplicaciones transmiten sus requisitos de red y el comportamiento de red deseado al controlador SDN a través de la API. Algunos ejemplos de aplicaciones son las redes energéticamente eficientes, la supervisión de la seguridad, el control de acceso y la gestión de redes.

2.1.2 PLANO DE CONTROL

El plano de control está involucrado en muchas actividades. Su función principal es mantener actualizada la información en la tabla de forwarding o reenvío para que este pueda manejar de mejor forma todo el tráfico que le sea posible. El plano de control es el responsable de manejar cualquier proceso (por ejemplo, ICMP) que pueda afectar la tabla de reenvío. Estos procesos son los responsables de gestionar la topología activa de la red y proporcionar a las aplicaciones información sobre esta y la actividad del plano de datos.



*Figura 2 Interfaces del Plano de Control
Fuente: (Stallings, 2016)*

Para la comunicación de los diferentes servicios de aplicación y la red SDN existen las llamadas interfaces de programación de aplicaciones o API, las que según (Córdoba López, 2019) no son más que un conjunto de definiciones y protocolos que funcionan como intermediarias para que 2 aplicaciones se comuniquen entre ellas. La Figura 2, en referencia a (Stallings, 2016) ilustra las diferentes interfaces que el plano de control posee, las cuales son:

- **Southbound Interface (Interfaz dirección sur):** Su función principal es permitir la comunicación entre el controlador SDN y los nodos de la red (conmutadores y enrutadores físicos y virtuales) para que el enrutador pueda descubrir la topología de la red, definir los

flujos de red e implementar las solicitudes que se le transmiten a través de las API en dirección norte o Northbound.

- **Northbound Interface (Interfaz dirección norte):** Contrariamente a la interfaz en dirección sur, las interfaces en dirección norte permiten la comunicación entre los componentes de nivel superior (servicios) y el controlador SDN. Mientras que las redes tradicionales usan firewalls o balanceadores de carga para controlar el comportamiento del plano de datos, SDN instala aplicaciones que usan el controlador y estas aplicaciones se comunican con el controlador a través de su interfaz en dirección norte.
- **Reenvío de ruta más corta (Shortest Path Forwarding) :** Utiliza información de enrutamiento recopilada de los conmutadores para establecer rutas preferidas.
- **Administrador de notificaciones (Notification Manager) :** Recibe, procesa y reenvía a los eventos de una aplicación, como notificaciones de alarma, alarmas de seguridad y cambios de estado.
- **(Mecanismos de seguridad) (Security Mechanisms):** Proporciona aislamiento y aplicación de seguridad entre aplicaciones y servicios.
- **(Administrador de topología) (Topology Manager):** Crea y mantiene información de topología de interconexión de conmutadores.
- **(Administrador de estadísticas) (Statistics Manager):** Recopila datos sobre el tráfico a través de los conmutadores.
- **(Administrador de dispositivos) (Device Manager):** Configura los parámetros y atributos del conmutador y administra las tablas de flujo.

Finalmente, en un nivel ligeramente inferior, existen procesos de control específicos para ciertos tipos de redes que se utilizan para aumentar el conocimiento del plano de control. Estos procesos incluyen verificación o notificación de disponibilidad y calidad de enlaces, descubrimiento de vecinos y resolución de direcciones.

2.1.3 PLANO DE DATOS

El plano de datos está compuesto por equipos de red llamados conmutadores, los cuales están especializados en el reenvío de paquetes. Sin embargo, a diferencia de las redes tradicionales, estos son simples elementos de reenvío sin inteligencia incorporada para tomar decisiones autónomas. Estos dispositivos (conmutadores) se comunican con el controlador a través de interfaces (Southbound), lo que garantiza la compatibilidad e interoperabilidad entre los diferentes dispositivos.

Si bien la única vista que se tiene de la red es a través del plano de control, el reenvío de tráfico real ocurre en el plano de datos; este es donde los dispositivos realizan el transporte y procesamiento de datos de acuerdo con las decisiones tomadas por el plano de control SDN; donde además está presente la característica más importante del plano de datos, el realizar estas funciones de reenvío sin un software integrado para tomar decisiones autónomas (Azodolmolky, 2013).

Un dispositivo de reenvío posee una tabla de flujos, la cual según (Azodolmolky, 2013) se constituye de tres partes: coincidencia de reglas, acciones a ejecutar para paquetes coincidentes y un contador para las estadísticas de estos paquetes. Además, estos campos de coincidencia de reglas incluyen puertos de un conmutador, MAC de origen y destino, tipo de Ethernet, ID de

VLANs, dirección IP de origen y destino, puerto TCP de origen y destino, tal como se indica en la Tabla 1 como referencia a (ONF, OpenFlow Switch Specification, 2013).

Tabla 1 Campos de paquetes usados para comparar con entradas de flujo

Puerto Ingreso	MAC Origen	MAC Destino	Tipo Ethernet	ID VLAN	Prioridad VLAN	IP Origen	IP Destino	TCP/UDP Puerto Origen	TCP/UDP Puerto Destino
----------------	------------	-------------	---------------	---------	----------------	-----------	------------	-----------------------	------------------------

Estas reglas de reenvío indican a la tabla de flujos cuál debería ser el siguiente salto en la ruta. Además, aparte de la acción de reenviar un paquete, el dispositivo de reenvío puede alterar el encabezado del paquete antes de reenviarlo o descartarlo. Por lo tanto, y como se muestra en la Figura 2, los paquetes entrantes pueden colocarse en una cola de entrada, en espera de ser procesados por el dispositivo de reenvío, y los paquetes reenviados generalmente se colocan en una cola de salida, en espera de su transmisión.

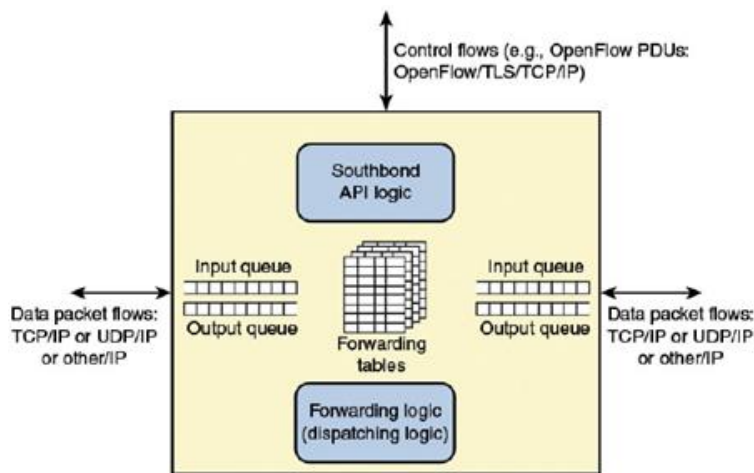


Figura 3 Plano de datos de un dispositivo de reenvío.
Fuente: (Stallings, 2016)

En este sentido, y cómo se ilustra en la Figura 2, un dispositivo de reenvío puede presentar diferentes tipos de puertos lógicos de entrada/salida; uno el cual proporciona comunicación de control mediante el protocolo OpenFlow hacia el controlador SDN, y otro para la entrada y salida de paquetes de datos.

Del mismo modo que el plano de control, el cual posee funciones específicas a través de las interfaces de Northbound y Southbound, el plano de datos posee sus propias funciones características, las cuales en referencia a (Stallings, 2016) son las siguientes:

- **Función de soporte de control:** Interactúa con la capa de control SDN para respaldar su programabilidad a través de interfaces de control. Además, esta función permite que el conmutador se comunique con el controlador, y que este, a través del protocolo OpenFlow, gestione esta comunicación.
- **Función de reenvío de datos:** Acepta los flujos de datos entrantes de otros dispositivos de red y los reenvía a lo largo de las rutas de reenvío que se han calculado y establecido de acuerdo con las reglas definidas por las aplicaciones SDN.

2.2 PROTOCOLO OPENFLOW

OpenFlow, definido por la ONF (Huang, Chowdary, & Pishadory, 2019), es un protocolo entre las capas de control y datos de una arquitectura SDN. OpenFlow, de manera básica, consta de hosts finales, un controlador y conmutadores habilitados para el funcionamiento de este protocolo. Además, hay que tener en consideración que contrariamente a un conmutador de red tradicional, uno de OpenFlow no se limita a ser un dispositivo de capa 2 el cual trabaja solamente con una tabla de direcciones MAC, sino que estos switches también son capaces de manejar direcciones IP y realizar trabajos de enrutamiento.

Cuando se implementa el protocolo OpenFlow, se lo realiza en ambos lados de la interfaz, es decir, este se aplica entre los dispositivos de la infraestructura de red (host finales) y el software de control SDN (controlador). Para identificar el tráfico de la red, OpenFlow utiliza el concepto de flujos, el cual no es más que una serie de instrucciones transmitidas desde el controlador al conmutador que se aplican basadas en las reglas de coincidencia predefinidas que pueden programarse de forma estática o dinámica mediante el software de control SDN. Estas reglas de forma general indican las acciones a tomar sobre los diferentes paquetes entrantes en la red. (Morrelae & Anderson, 2015).

El protocolo OpenFlow trabaja sobre el protocolo TCP en su puerto 6653 para las versiones v1.0 y v1.3 de OpenFlow (Huang, Chowdary, & Pishadory, 2019). Además, hay que tener en cuenta que antes de realizar cualquier acción por parte del protocolo, este primero debe crear un canal para el intercambio de instrucciones entre controlador y conmutador llamado canal OpenFlow, y el cual sólo se forma al establecer una conexión TCP satisfactoria mediante el enlace de tres vías (llamado así por constar de 3 pasos para el establecimiento de una conexión TCP). A partir de aquí, el protocolo OpenFlow trabaja distribuyendo sus funciones en 2 partes, las cuales según (Paul Göransson, 2017) son las siguientes:

- La primera parte establece una sesión de control, aquí se define una estructura de mensaje para intercambiar modificaciones de flujo y recopilar estadísticas. De esta manera, se crea un proceso de canalización lógica dentro de un conmutador para manejar diferentes flujos de paquetes en una red.
- La segunda parte define un proceso de configuración y administración, el cual está basado a partir del Protocolo de Configuración de Red (NETCONF) para asignar puertos de un

conmutador a un controlador en particular. Este proceso también define los comportamientos en caso de falla de conexión del controlador.

En resumen, OpenFlow especifica las reglas usadas entre el controlador y conmutador, así como las acciones que el conmutador debe realizar. En la Figura 4 se presenta de forma general la arquitectura de este protocolo, aquí se observa como el canal OpenFlow es bidireccional y se encarga de la comunicación entre controlador y conmutador. También se puede observar que cada conmutador OpenFlow posee su propia tabla de flujos (o instrucciones) designada por el controlador, así como su propio plano de datos el cual es el encargado del reenvío de paquetes.

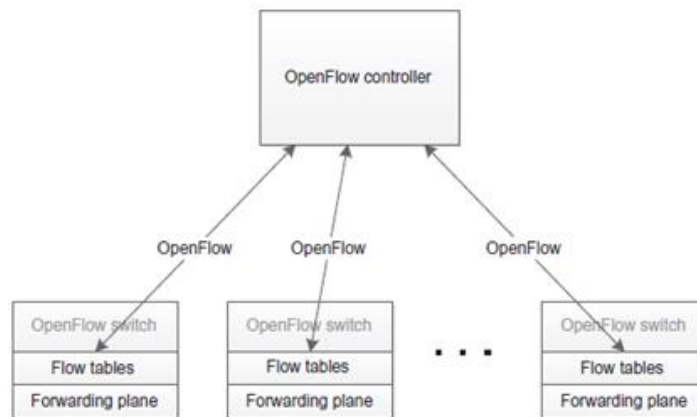


Figura 4 Arquitectura básica de OpenFlow
Fuente: (Paul Göransson, 2017)

Finalmente, existe un proceso de cómo cada conmutador evalúa el comportamiento de cada paquete entrante según su tabla de flujos. Este funcionamiento, según (Paul Göransson, 2017) es el siguiente:

- El conmutador evalúa los diferentes campos coincidentes (MAC de origen/destino, ID de VLANs, dirección IP de origen/destino, puerto TCP de origen/destino) en cada paquete entrante y define un flujo coincidente, el cual tendrá una acción asociada de acuerdo con cada paquete.

- Si no se encuentra ninguna coincidencia, el conmutador reenvía el paquete al controlador para obtener instrucciones sobre cómo tratar con el paquete.
- Normalmente, el controlador actualizará el conmutador con nuevas entradas de flujo a medida que se reciban nuevos patrones de paquetes, de modo que el conmutador pueda tratar con ellos.

2.2.1 CONMUTADOR OPENFLOW

Un conmutador OpenFlow, y como se muestra en la Figura 5, consta de tres componentes principales: un canal OpenFlow, una tabla de grupo y una o más tablas de flujo. Las tablas de flujo y la tabla de grupo son las responsables de realizar la búsqueda y reenvío de paquetes, con la diferencia que una tabla de grupo se enfoca solamente en paquetes complejos los cuales no pueden ser procesados fácilmente por una tabla de flujos (agregación de link, multipath, etc) y el canal OpenFlow se utiliza para comunicarse con el controlador mediante el protocolo OpenFlow.

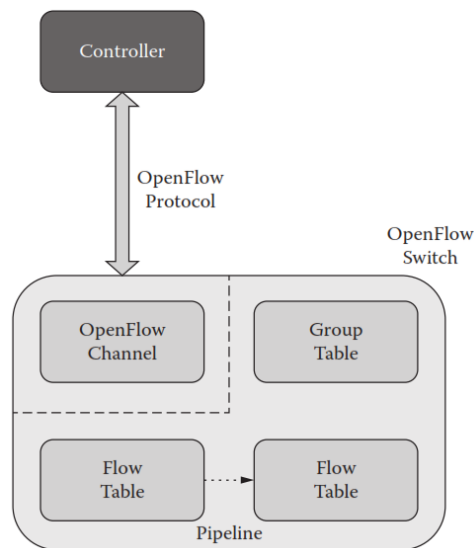


Figura 5 Componentes Conmutador OpenFlow
Fuente: (Morrelae & Anderson, 2015)

Cada tabla de flujo en el conmutador contiene un conjunto de entradas de flujo, que según (Azodolmolky, 2013) constan de:

- **Campos de encabezado:** Con información que se encuentra en el encabezado del paquete y el puerto de entrada, la cual se utiliza para hacer coincidir los paquetes entrantes.
- **Contadores:** Se utilizan para recopilar estadísticas del flujo de datos, como el número de paquetes recibidos, el número de bytes y la duración de este.
- **Conjunto de instrucciones:** Acciones que se aplicarán después de una coincidencia o match que dicta cómo manejar los paquetes coincidentes. Por ejemplo, la acción podría ser reenviar un paquete a un puerto específico.

Tras la llegada de un paquete al conmutador OpenFlow, como se muestra en la Figura 6, los campos (MAC de origen/destino, ID de VLANs, dirección IP de origen/destino, puerto TCP de origen/destino) del encabezado del paquete entrante se extraen y se comparan con las entradas de la tabla de flujo. Este emparejamiento comienza en la primera entrada de la tabla y continúa a través de las entradas siguientes. Si se encuentra una entrada coincidente, el conmutador aplica el conjunto de instrucciones asociadas con la entrada de flujo coincidente.

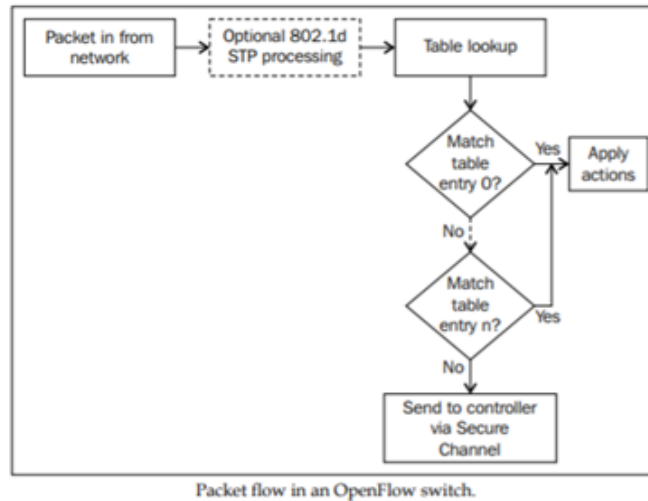


Figura 6 Flujo de paquetes en un conmutador OpenFlow
Fuente: (Azodolmolky, 2013)

Además, y continuando con la Figura 6, por cada paquete que realice match, se actualizarán los contadores asociados a esa entrada, igualmente, si el procedimiento de búsqueda de la tabla de flujo no da como resultado una coincidencia, la acción realizada por el conmutador dependerá de las instrucciones definidas en la entrada de fallas que existe para manejar este tipo de casos. Estas acciones incluyen descartar el paquete, enviar el paquete a todas las interfaces o reenviar el paquete al controlador a través del canal seguro OpenFlow (Azodolmolky, 2013).

2.3 COMUNICACIÓN ENTRE CONTROLADOR Y CONMUTADOR

Esta comunicación, donde se realiza el intercambio de un conjunto de mensajes definidos, se produce a través de un canal seguro. Este canal es la interfaz (Southbound) que conecta cada conmutador OpenFlow a un controlador. Cuando se inicia esta comunicación, primero se produce una conexión de seguridad de la capa de transporte (TLS) al controlador definido por el usuario. El conmutador y el controlador se autentican mutuamente intercambiando certificados firmados por una clave privada. Cada conmutador debe ser configurable por el usuario con un certificado

para autenticar el controlador (certificado de controlador) y el otro para autenticarse en el controlador (certificado de conmutador).

En el caso de que un conmutador pierda contacto con el controlador, como resultado de un tiempo de espera de solicitud, tiempo de espera de sesión TLS (Seguridad de la capa transporte) u otra desconexión, debe intentar comunicarse con uno o más controladores de respaldo; si estos intentos de contactar a otro controlador fallan, el conmutador debe ingresar al modo de emergencia y restablecer inmediatamente la conexión TCP actual mediante una tabla de flujos de emergencia (Azodolmolky, 2013).

Los mensajes de modificación de flujo de emergencia deben tener un valor de tiempo de espera establecido en cero. De lo contrario, el conmutador debe responder con un mensaje de error (Azodolmolky, 2013). Todas las entradas normales se eliminan al entrar en el modo de emergencia, además, al conectarse nuevamente a un controlador, estas entradas de flujo permanecen; por lo que, el controlador tiene entonces, la opción de borrar todas las entradas de flujo si lo desea.

Por seguridad, OpenFlow en su versión 1.3 proporciona soporte para la comunicación TLS cifrada con un intercambio de certificados entre los conmutadores y controladores. El protocolo OpenFlow define tres tipos de mensajes: De controlador a conmutador, Simétricos y Asíncronos.

2.3.1 MENSAJES DE CONTROLADOR A CONMUTADOR

Los mensajes de controlador a conmutador son iniciados por el controlador y se utilizan para administrar o inspeccionar directamente el estado del conmutador. Este tipo de mensajes

pueden requerir o no una respuesta del conmutador y se clasifican en los siguientes subtipos (ONF, OpenFlow Switch Specification, 2013):

- **Características (Features):** Una vez establecida la sesión TLS, el controlador envía un mensaje de solicitud de función al conmutador. El conmutador debe responder con un mensaje de respuesta de funciones que especifique las funciones y capacidades que admite el conmutador.
- **Configuración (Configuration):** El controlador puede establecer y consultar parámetros de configuración en el conmutador. Este solo puede responder a una consulta del controlador.
- **Modificar estado (Modify-State):** El controlador envía estos mensajes para gestionar el estado de los conmutadores. Se utilizan para agregar, eliminar o modificar las entradas de la tabla de flujo o para establecer las prioridades de los puertos del conmutador.
- **Leer estado (Read-State):** Estos mensajes recopilan estadísticas de las tablas de flujo del conmutador, los puertos y las entradas de flujo individuales.
- **Enviar paquete (Send-Packet):** El controlador los utiliza para enviar paquetes desde un puerto especificado en el conmutador.
- **Barrera (Barrier):** El controlador utiliza los mensajes de solicitud y respuesta de barrera para para recibir notificaciones de las operaciones completadas.

2.3.2 MENSAJES SIMÉTRICOS

Los mensajes simétricos son iniciados por el conmutador o el controlador y enviados sin solicitud. Hay dos subtipos de mensajes simétricos en el protocolo OpenFlow, los cuales según (ONF, OpenFlow Switch Specification, 2013) son los siguientes:

- **Hola (Hello):** Los mensajes de saludo se intercambian entre el conmutador y el controlador al configurar la conexión.
- **Eco (Echo):** Los mensajes de solicitud o respuesta de eco se pueden enviar desde el conmutador o el controlador, y deben devolver una respuesta de eco (echo reply). Estos mensajes se pueden utilizar para indicar la latencia y el ancho de banda de una conexión de conmutador de controlador.

2.3.3 MENSAJES ASINCRÓNICOS

Los mensajes asincrónicos son iniciados por el conmutador y se utilizan para actualizar el controlador sobre eventos en la red y cambios en el estado del conmutador. Los conmutadores envían mensajes asincrónicos al controlador para indicar la llegada de un paquete, un cambio de estado del conmutador o incluso un error. Según (ONF, OpenFlow Switch Specification, 2013) existen cuatro mensajes asincrónicos principales:

- **Packet-in:** Para todos los paquetes que no tienen una entrada de flujo coincidente se envía un mensaje de entrada de paquete al controlador. Si el conmutador tiene suficiente memoria en búfer, se guardará el mensaje de entrada el cual contiene una fracción del encabezado del paquete y una ID de búfer para ser utilizada por el conmutador para reenviar el paquete. Los conmutadores que no admiten el almacenamiento en búfer interno (o que se han

quedado sin espacio de búfer interno) deben enviar el paquete completo al controlador como parte del mensaje.

- **Eliminación de flujo (Flow-Removal):** Cuando se agrega una entrada de flujo al conmutador mediante un mensaje de modificación, un valor de tiempo de espera indica cuándo debe eliminarse la entrada debido a la falta de actividad, independientemente cual sea esta.
- **Estado del puerto (Port-status):** El conmutador envía mensajes de estado del puerto al controlador a medida que cambia el estado de configuración del puerto.
- **Error:** El conmutador puede notificar al controlador de problemas mediante mensajes de error.

2.4 CONTROLADORES SDN

El controlador es el cerebro de SDN. Se encuentra entre los dispositivos del plano de datos en un extremo (Southbound) y las aplicaciones en el otro (Northbound). Un controlador SDN asume la responsabilidad de establecer cada flujo en la red mediante la instalación de entradas de flujo en los dispositivos de conmutación (Huang, Chowdary, & Pishadory, 2019).

Las entradas de flujo se pueden agregar a un dispositivo de conmutación de dos modos, proactivo o reactivo. En el modo proactivo las reglas de flujo se enviarán a los dispositivos de conmutación tan pronto como el controlador las aprenda; en cambio, en el modo reactivo, el controlador enviará las entradas de flujo a los dispositivos de conmutación solo cuando sea necesario.

Existen varios controladores SDN, los cuales en su mayoría son de libre acceso y código abierto. Estos varían en su capacidad de manejo de la red, accesibilidad en su lenguaje de programación y sistema operativo de funcionamiento. En este trabajo de grado se usarán dos de los controladores con mayor soporte y acceso del mercado, los cuales son: Pox y Floodlight.

2.4.1 POX

POX es una plataforma de desarrollo de código abierto para aplicaciones de control de SDN basadas en Python, como los controladores OpenFlow SDN. Actualmente es compatible con OpenFlow 1.0 e incluye soporte especial para las extensiones de Open vSwitch (software de código abierto que se utiliza como switch virtual) (POX, s.f.). POX se derivó y desarrolló mediante el uso del lenguaje de programación Python y tiene como objetivo proporcionar un entorno eficiente y fácil para realizar investigaciones y pruebas en redes SDN (Noman & Jasim, 2020).

POX se basa en un modelo en el que todos los elementos de la red se reconocen como componentes separados que pueden aislarse y utilizarse de la manera que sea necesaria (Noman & Jasim, 2020). La ubicación de POX es específicamente entre los componentes de red y las aplicaciones. Además, se encarga de lograr cualquier tipo de comunicación entre aplicaciones y dispositivos.

El funcionamiento el controlador POX se divide en tres características: la primera, donde se realizan las configuraciones de los dispositivos de red; la segunda, donde se selecciona la ruta óptima para las aplicaciones de tráfico; y finalmente la tercera, la cual permite que los servidores informen a los conmutadores hacia dónde dirigir los paquetes entrantes. Estas acciones se realizan

mediante módulos POX, los cuales, simplemente son programas de Python que son iniciados desde una línea de comandos en la consola del sistema (Noman & Jasim, 2020)

Para comenzar a trabajar con este controlador es necesario acceder a un repositorio GitHub, en este caso <http://github.com/noxrepo/pox>, y descargarlo. A partir de esto sólo es necesario llamar al archivo ejecutable del controlador llamado `pox.py` para una ejecución paso a paso de este. Cabe recalcar que también existen formas rápidas de ejecución con una configuración estable ya preestablecida del controlador, las cuales se muestran paso a paso en el anexo 2.

2.4.2 FLOODLIGHT

Floodlight es un controlador Java-OpenFlow de código abierto creado por una comunidad abierta de desarrolladores que admite el uso del protocolo OpenFlow desde su versión 1.0 a 1.5 para flujos de tráfico en un entorno SDN (Ribeiro, 2018). Las funciones principales de este controlador radican en la administración y asignación de las diferentes instrucciones enviadas desde el controlador al conmutador para el manejo del tráfico de red.

Este controlador, según (Fancy & Pushpalatha, 2017), resulta ventajoso sobre otros ya que ofrece un software de manejo fácil, además que permite desarrollar aplicaciones en el lenguaje de programación Java. Floodlight también funciona en una variedad de entornos de red e incluso admite redes en las cuales sus conmutadores compatibles con OpenFlow están conectados a través de conmutadores tradicionales que no son OpenFlow. Adicionalmente, el controlador Floodlight

es compatible con OpenStack, el cual no es más que un conjunto de herramientas de software que ayudan a crear y administrar plataformas de computación en la nube de forma pública y privada.

En el aspecto de la instalación, y a diferencia con POX, Floodlight nos permite acceder a un controlador ya instalado y configurado mediante sólo una descarga desde su página oficial <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview>, y aunque este ya viene con su propia máquina virtual y sistema operativo incorporado, la instalación de otros controladores en el mismo sistema operativo no presenta ningún problema para una ejecución estable de Floodlight.

2.5 MININET

Mininet es una herramienta de software que permite emular una red OpenFlow completa en una sola computadora. Mininet utiliza virtualización basada en procesos (o virtualización a nivel de sistema operativo) para lograr la ejecución de hosts y conmutadores en un solo kernel (núcleo del sistema operativo) de un sistema operativo (Azodolmolky, 2013). Este, además de crear conmutadores OpenFlow a partir del kernel, también crea controladores para manejar la comunicación de los conmutadores y hosts a través de la red emulada.

Mediante Mininet las nuevas aplicaciones de red se pueden desarrollar y probar primero en una emulación de red, para luego replicarlo en una infraestructura operativa real. De forma predeterminada, Mininet es compatible con OpenFlow v1.0; sin embargo, también puede modificarse para admitir un cambio de software que implemente una versión más reciente. Según (Azodolmolky, 2013) las características y beneficios de Mininet son:

- Mininet permite crear una red de hosts virtuales, conmutadores, controladores y enlaces la cual sólo está limitada por las capacidades físicas y virtuales de la máquina en dónde se implementa.
- Los hosts de Mininet se ejecutan en base a un software de red estándar Linux, y sus conmutadores admiten OpenFlow. Puede considerarse como un laboratorio OpenFlow sencillo para desarrollar aplicaciones, el cual adicionalmente permite realizar pruebas de topología complejas, sin la necesidad de conectar una red física.
- Mininet incluye una interfaz de línea de comandos (CLI) que tiene en cuenta la topología para depurar o ejecutar pruebas en toda la red.
- Se puede comenzar a usar Mininet de inmediato sin ninguna programación, pero también proporciona una API de Python sencilla y extensible para la creación y experimentación de redes.

El código de Mininet es casi en su totalidad Python, excepto por pequeñas utilidades en el lenguaje de programación C (Azodolmolky, 2013). Este también posee algunas limitantes las cuales se basan en rendimiento y compatibilidad, una de estas es que no se puede exceder el ancho de banda disponible (velocidad máxima que el procesador del equipo puede almacenar o ejecutar datos y procesos) en un ordenador, laptop o servidor, además que no es ejecutable en conmutadores o aplicaciones OpenFlow que no sean compatibles con Linux (Contributors, 2021).

Para comenzar a trabajar con Mininet es necesario descargar una imagen de una máquina virtual (VM) la cual se ejecuta en Ubuntu. Esta VM incluye todos los archivos necesarios de

OpenFlow, además de herramientas preinstaladas para admitir grandes redes. Este proceso se lo detalla en el Anexo 1 del documento.

2.5.1 EMULACIÓN SOBRE MININET PARA SDN

Mininet permite crear, personalizar y modificar un prototipo de red SDN. Además, este emulador posee una línea de comandos, o también llamada CLI de Mininet, que funciona para la creación de una red utilizando líneas de código específicas en el lenguaje de programación Python-Mininet (Doxygen, 2021) . Su CLI permite controlar y administrar toda una red virtual desde la consola. Además, la API de Mininet permite desarrollar aplicaciones de red personalizadas con pocas líneas de script (secuencia de comandos) en lenguaje Python, donde, si un prototipo de red funciona correctamente en Mininet, este inmediatamente se puede validar para trasladar su configuración a una red real. Una red emulada en Mininet, y según (Doxygen, 2021) posee los siguientes componentes:

- **Hosts:** Entidades a nivel de usuario que poseen interfaces y puertos propios.
- **Enlaces:** Estos poseen su propia interfaz virtual Ethernet, donde su velocidad de transmisión, retardo e incluso una característica como la pérdida de paquetes en el enlace puede ser configurados, ya sea modificando el código Python correspondiente a la red o desde su interfaz virtual.
- **Conmutadores:** Poseen su propia interfaz virtual, y de igual manera que en los enlaces, se puede utilizar código Python para configurar y cambiar todo tipo de parámetros desde el nombre del conmutador, direcciones IP, entre otras características.

2.5.1.1 CLASES UTILIZADAS EN UNA RED MININET

La programación orientada a objetos, la cual es utilizada en el diseño de redes definidas por software (SDN) en la interfaz de programación de aplicaciones Mininet, es un paradigma de programación (o estilo de programación) que se basa en el concepto de “objetos”, los cuales contienen un dato en forma de campo (mínima unidad de información) al cual se le asigna un atributo. Este concepto es usado para que estructuras o piezas reusables de códigos, en este caso las llamadas clases, sean usadas para crear instancias individuales de objetos, o dicho de otra manera, crear objetos con la capacidad de que estos se relacionen entre sí.

Una clase es un plano abstracto que se utiliza para crear objetos más específicos. Estas usualmente representan una categoría más amplia de objetos que comparten atributos. Según el manual de referencia Mininet-Python (Doxygen, 2021), las clases existentes en Mininet son las siguientes:

- **class mininet.net.Mininet(Object):** Esta clase encapsula los dispositivos emulados en una red.
- **class mininet.Topo.Topo(Object):** Esta clase representa una topología de red.
- **class mininet.node.Node(Object):** Es un nodo virtual que representa un dispositivo de red cualquiera.
- **class mininet.link.Link(Object):** Representa el enlace virtual de la conexión entre dos nodos o dispositivos de red.
- **class mininet.node.Switch(Node):** Un nodo virtual que representa a un switch operando como conmutador OpenFlow.

- **class mininet.node.Controller(Node):** Un nodo virtual que ejecuta un controlador OpenFlow.

2.5.1.2 MÉTODOS UTILIZADOS EN UNA RED MININET

Una clase también posee ciertas funciones, llamadas métodos, las cuales sólo están disponibles para objetos de un mismo atributo. Estas funciones se definen, sólo y únicamente, dentro su clase y realizan acciones útiles para un tipo de objeto en específico. En la API de Mininet-Python, y según (RAMIREZ, 2015), los métodos más utilizados en el diseño de una red SDN son:

- **addSwitch():** Agrega un switch a una topología y regresa el nombre de este.
- **addHost():** Agrega un host a una topología y regresa el nombre de este.
- **addLink():** Agrega un link bidireccional a una topología.
- **addPort():** Genera un set de puertos nuevo.
- **start():** Permite iniciar la red.
- **pingAll():** Verifica conectividad realizando un ping entre todos los nodos.
- **stop():** Detiene la red.
- **dumbNodeConnections():** Borra todas las conexiones de un set de nodos.
- **IP():** Regresa una dirección IP para un host o interfaz.
- **MAC():** Regresa una dirección MAC de un host o interfaz.
- **setARP():** Agrega una entrada ARP estática para un host o interfaz.
- **setIP():** Asigna la dirección IP para un host o interfaz.
- **setMac():** Asigna la dirección MAC a un host o interfaz.

Finalmente, existen 3 formas de generar una topología en Mininet. Una es mediante redes predefinidas (sencilla, lineal y árbol), las cuales funcionan como ejemplo y base de una red, estas se pueden ejecutar desde la línea de comandos donde está instalada la carpeta Mininet y pueden ser configurables en dos parámetros, el número de conmutadores y hosts en la topología. La segunda forma es utilizar la interfaz Miniedit, la cual se encuentra en la ruta de archivos `cd /mininet/examples` y se ejecuta con el comando `sudo ./miniedit.py`. Finalmente, y la forma más completa, es llamar un código Python previamente creado dónde está desarrollada la red y puede ser completamente modificable para cubrirlas diferentes necesidades del administrador de red. Estas 3 formas de diseño sobre el software de emulación Mininet están detalladas en el apartado de anexos.

3. CAPÍTULO III: DISEÑO E IMPLEMENTACIÓN DE LA RED SDN Y TESTBED

Este apartado del trabajo se enfoca en la creación, verificación y emulación de cada uno de los escenarios SDN planteados a través de los controladores Pox y Floodlight. Además, se presenta paso a paso la implementación, configuración y ejecución del Testbed SDN en el centro de datos de la carrera, donde posteriormente se procederá con la aplicación de tres ambientes de práctica, los cuales cumplen con el objetivo de proporcionar herramientas que permitirán a los estudiantes de la carrera el conocer, entrenar e investigar acerca de la tecnología SDN.

3.1 DISEÑO DE LA RED Y EMULACIÓN DE ESCENARIOS SDN.

El desarrollo de la emulación del presente trabajo se basa en un hypervisor tipo 2 (hosted hypervisor), para ello se utiliza el software de virtualización Oracle Virtual Box y sobre este se ejecuta una instancia de máquina virtual (VM) que poseen el sistema operativo Linux con una versión de Kernel 4.15.0-142 (Ubuntu 14.04) y adicionalmente se tienen instalado la herramienta de emulación Mininet en su versión 4.3.0.

Para la configuración de la máquina virtual se ha usado una memoria RAM base de 8GB para evitar cualquier conflicto de procesamiento debido a las simulaciones y la cantidad de procesamiento que estas necesitan. Asimismo, para soportar los requerimientos de sistema (procesamiento y almacenamiento) de la VM, se ha trabajado en un ordenador portátil Intel Core i5-8300H @ 2.30GHz de octava generación y 20 GB de memoria RAM.

En el caso de Mininet, para su instalación en la VM se ha seguido la guía desarrollada en el Anexo 1. Para el controlador POX se utilizó la distribución que la instalación de Mininet provee

cómo indica el Anexo 2; y finalmente, en el caso del controlador Floodlight, se ha usado la VM preconfigurada y lista para su descarga como se encuentra en el Anexo 3.

Este trabajo, al buscar crear una plataforma o testbed para la práctica o realización de pruebas de nuevas tecnologías en redes SDN, presenta como escenarios de emulación dos diferentes casos para su desarrollo, los cuales son:

- **Escenario 1:** Mininet y los controladores(Pox/Floodlight) se encuentran sobre una misma red y máquina virtual.
- **Escenario 2:** Mininet y los controladores(Pox/Floodlight) se encuentran sobre una misma máquina virtual, pero en redes diferentes.

3.1.1 DISEÑO SDN DEL ESCENARIO 1

Cada escenario utiliza una topología similar la cual se compone por dos controladores SDN, uno que trabaja con Floodlight y el otro con POX, además de tres conmutadores y dos hosts. Como se muestra en la Figura 7, para el primer escenario los controladores están conectados a un conmutador inicial mediante un bridge o puente de red en sus interfaces ethernet, de esta manera se permite la conectividad entre estos dispositivos. También, este conmutador está conectado a otros dos conmutadores adicionales que poseerán una interfaz ethernet extra para la comunicación entre estos y sus host. Cabe recalcar, y cómo lo indica la Figura 7, este escenario se encuentra en una misma máquina virtual.

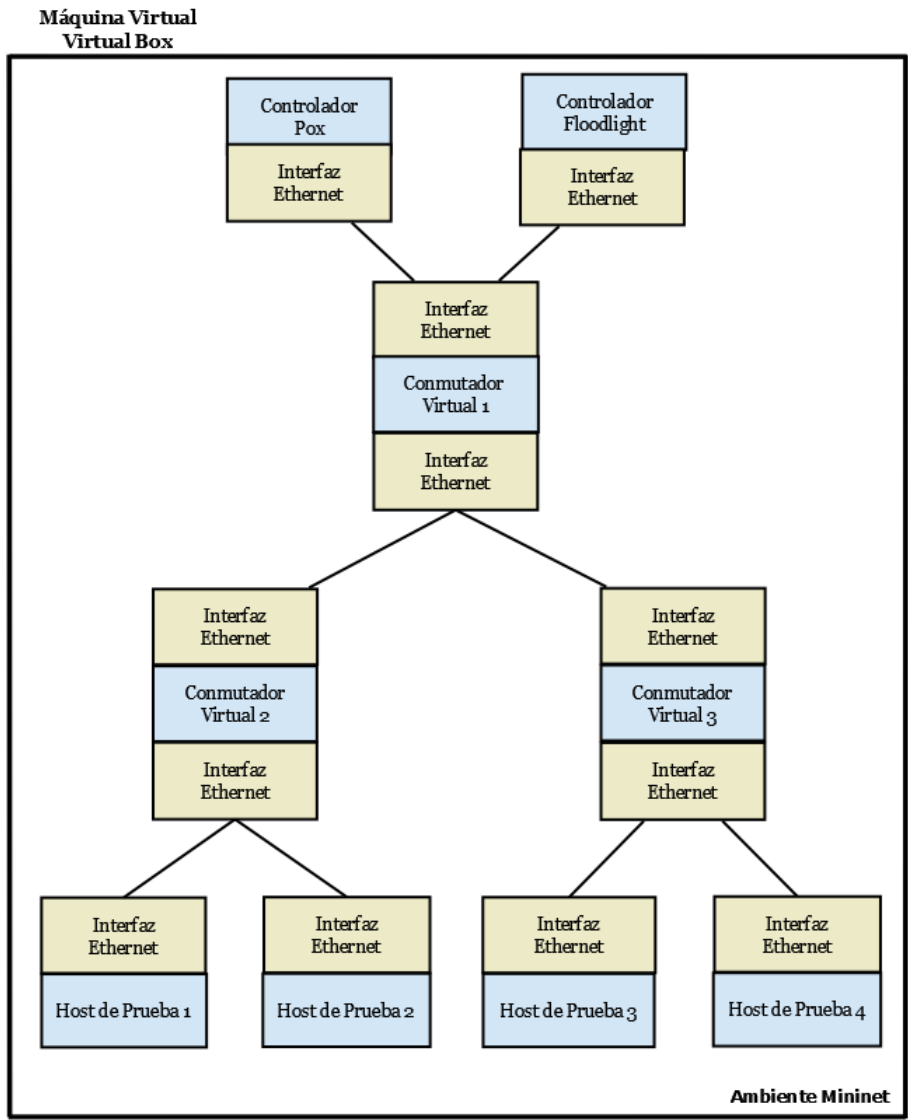


Figura 7 Diseño SDN del escenario 1

Además, con el diseño de la red SDN ya definido para el primer escenario, la Tabla 2 presenta los requerimientos de hardware necesarios para virtualizar el diseño propuesto. De la misma forma, la Tabla 3 indica el direccionamiento de red IPv4 que permite la convergencia necesaria para este primer caso.

Tabla 2 Requerimientos de Máquina Virtual para el Escenario 1

Software	Versión	Hardware Recomendado	Hardware Configurado
Linux Mininet/Pox/Floodlight	Ubuntu 14.04	8 GB RAM	8 GB RAM

Tabla 3 Plan de Direccionamiento IPv4 escenario 1

Dispositivo	Interfaz	Dirección IP	Máscara
Controladores POX/Floodlight	Eth0	10.0.2.15	255.255.255.0
Conmutador1	Eth1	_____	255.255.255.0
	Eth2	_____	255.255.255.0
Conmutador2	Eth1	_____	255.255.255.0
	Eth2	_____	255.255.255.0
Conmutador3	Eth1	_____	255.255.255.0
	Eth2	_____	255.255.255.0
Host 1	Eth1	10.0.2.1	255.255.255.0
Host 2	Eth1	10.0.2.2	255.255.255.0
Host 3	Eth1	10.0.2.3	255.255.255.0
Host 4	Eth1	10.0.2.4	255.255.255.0

Finalmente, la Figura 8 nos muestra la topología lógica de cómo se conecta cada dispositivo que compone el escenario 1 de pruebas de la red SDN propuesta. En esta se muestra al controlador SDN POX/Floodlight, así como los tres diferentes conmutadores y hosts, los cuales ya tienen asignado su respectivo direccionamiento lógico.

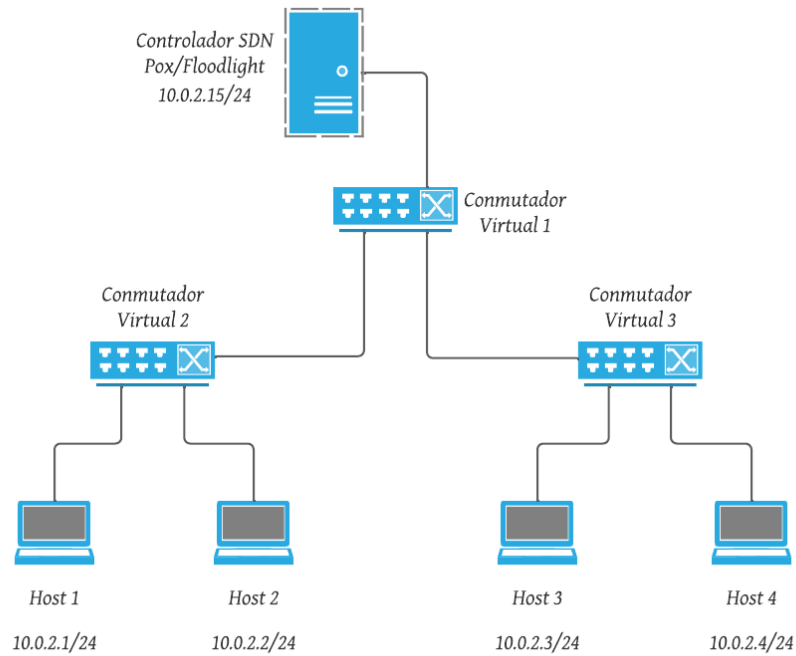


Figura 8 Topología lógica del diseño de red SDN del escenario 1

3.1.2 DISEÑO SDN DEL ESCENARIO 2

Para este caso, la única diferencia en comparación al escenario anterior radica en que los controladores tienen distinto direccionamiento IP con respecto a los conmutadores y host, es decir, se encuentran en una red diferente; por ende, y como se muestra en la Figura 7, todos los dispositivos en ambos casos se encuentran en una sola máquina virtual.

A continuación, y con el diseño de la red SDN ya definido para el escenario 2, los requerimientos necesarios de hardware para su virtualización se asemejan al escenario anterior, por lo cual se utiliza la Tabla 2 como referencia a estos. También se presenta en la Tabla 4 el direccionamiento de red IPv4 para este caso.

Tabla 4 Plan de Direccionamiento IPv4 escenario 2

Dispositivo	Interfaz	Dirección IP	Máscara
Controlador POX/Floodlight	Eth0	10.0.2.15	255.255.255.0
	Eth1	_____	255.255.255.0
Conmutador1	Eth2	_____	255.255.255.0
	Eth1	_____	255.255.255.0
Conmutador2	Eth2	_____	255.255.255.0
	Eth1	_____	255.255.255.0
Conmutador3	Eth2	_____	255.255.255.0
	Eth1	_____	255.255.255.0
Host 1	Eth1	192.168.10.1	255.255.255.0
Host 2	Eth1	192.168.10.2	255.255.255.0
Host 3	Eth1	192.168.10.3	255.255.255.0
Host 4	Eth1	192.168.10.4	255.255.255.0

Por último, la Figura 9 indica la topología lógica de conexión de cada dispositivo que compone el escenario 2 de la red SDN; además se muestra como los controladores SDN pertenecen a una red diferente de los demás dispositivos.

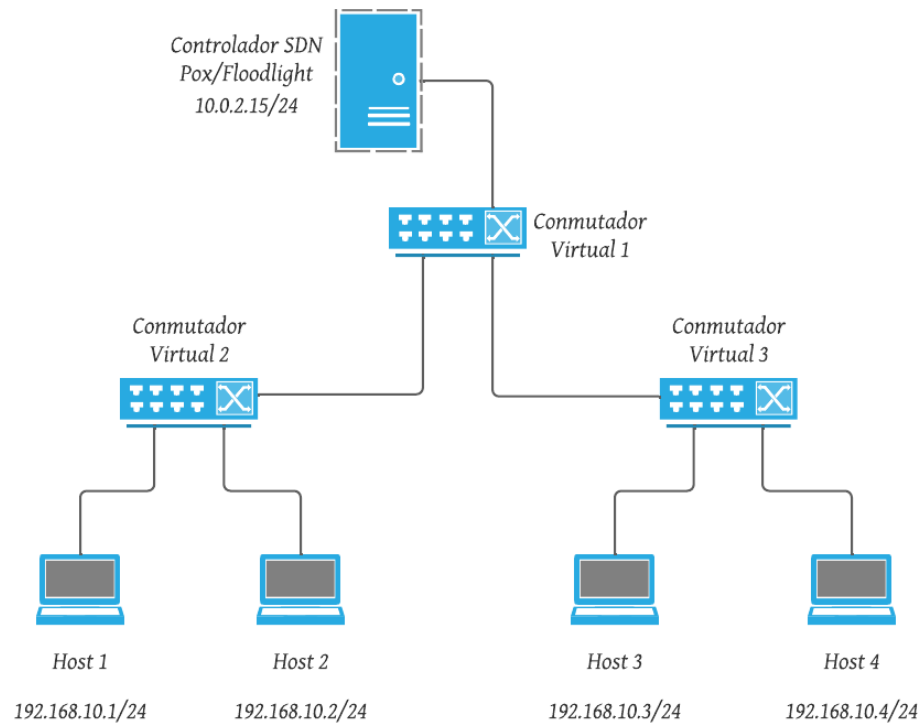


Figura 9 Topología lógica del diseño de red SDN del escenario 2

3.1.3 CREACIÓN DE LA RED SDN SOBRE PYTHON

Una vez propuesto el diseño de cada escenario, el siguiente paso a seguir es la elaboración del código python de las topologías de red planteadas. Para el escenario 1, como indica la Figura 10.a, desde un archivo python personalizado *Escenario1Testbed.py* se crea una red SDN con dos controladores, tres conmutadores y cuatro hosts. Además, para el escenario 2, el archivo python denominado *Escenario2Testbed.py* difiere en comparación al archivo del escenario 1, en el direccionamiento para cada host, tal como se muestra en la Figura 10.b

```

Escenario1Testbed.py x
## Se importa las clases necesarias
from mininet.topo import Topo
from mininet.net import Mininet

## MyTopo es el nombre de la clase que refiere a la topologia
class MyTopo( Topo ):
    def __init__( self ):
|
## Se inicializa la topologia, en este caso Topo
    Topo.__init__( self )

    ## Creación de switch o conmutadores
    s1 = self.addSwitch('s1')
    s2 = self.addSwitch('s2')
    s3 = self.addSwitch('s3')

    ## Creación de host, adicionando su direccionamiento IPv4
    h1 = self.addHost('h1',ip='10.0.2.1/24')
    h2 = self.addHost('h2',ip='10.0.2.2/24')
    h3 = self.addHost('h3',ip='10.0.2.3/24')
    h4 = self.addHost('h4',ip='10.0.2.4/24')

    ## Creación de enlaces de la red
    net.addLink(h1, s1)
    net.addLink(h2, s1)
    net.addLink(h3, s3)
    net.addLink(h4, s3)
    net.addLink(s1, s2)
    net.addLink(s2, s3)

## Función para la ejecución de la topologia
topos = {'mytopo': (lambda: MyTopo())}

Escenario2Testbed.py x
## Se importa las clases necesarias
from mininet.topo import Topo
from mininet.net import Mininet

## MyTopo es el nombre de la clase que refiere a la topologia
class MyTopo( Topo ):
    def __init__( self ):
|
## Se inicializa la topologia, en este caso Topo
    Topo.__init__( self )

    ## Creación de switch o conmutadores
    s1 = self.addSwitch('s1')
    s2 = self.addSwitch('s2')
    s3 = self.addSwitch('s3')

    ## Creación de host, adicionando su direccionamiento IPv4
    h1 = self.addHost('h1',ip='192.168.10.1/24')
    h2 = self.addHost('h2',ip='192.168.10.2/24')
    h3 = self.addHost('h3',ip='192.168.10.3/24')
    h4 = self.addHost('h4',ip='192.168.10.4/24')

    ## Creación de enlaces de la red
    net.addLink(h1, s1)
    net.addLink(h2, s1)
    net.addLink(h3, s3)
    net.addLink(h4, s3)
    net.addLink(s1, s2)
    net.addLink(s2, s3)

## Función para la ejecución de la topologia
topos = {'mytopo': (lambda: MyTopo())}

```

Figura 10 a) Código Python de la red SDN escenario 1 b) Código Python de la red SDN escenario 2

Además, es necesario realizar una configuración en el direccionamiento IPv4 de la VM, esto debido a que la dirección IP del controlador debe mantenerse estática en ambos casos dado que el ambiente Mininet, a partir del archivo python programado, es el encargado de asignar direcciones IP a los host y conmutadores de acuerdo con los lineamientos de cada escenario, mientras el controlador SDN siempre mantendrá su misma dirección IP, en cualquier caso.

Para realizar esta configuración se accede al archivo de interfaces de red de la VM que se encuentra en el directorio `/etc/network/interfaces` y se modifica tal como muestra la Figura 11.

```
GNU nano 2.2.6 Archivo: /etc/network/interfaces
# Interfaz de loopback
auto lo
iface lo inet loopback

# Interfaz estatica configurada
auto eth0

iface eth0 inet static
address 10.0.2.15
netmask 255.255.255.0
network 10.0.2.0
broadcast 10.0.2.255
gateway 10.0.2.100
dns-nameservers 8.8.8.8 8.8.4.4
```

Figura 11 Configuración de Interfaz de red para direccionamiento estático.

3.2 EJECUCIÓN DE ESCENARIOS SDN.

3.2.1 ESCENARIO 1 - POX

Para empezar con la ejecución de la red SDN del escenario 1 correspondiente al controlador Pox, y ya desde la consola de la VM, primero es necesario dirigirse al directorio donde se encuentra el archivo python con la topología programada, en este caso `/home/jtprox`, desde allí se procede a ejecutar el archivo python como indica la Figura 12. Aquí, inmediatamente después de la ejecución se observa cómo se adiciona y conecta el controlador para después proseguir con la creación de la red y sus demás dispositivos y enlaces. Además, para la visualización de paquetes en cualquier transmisión sobre la red SDN, se sigue un proceso el cual está detallado en el anexo 9, y el cual se debe aplicar antes de la ejecución de cualquier controlador.

```

jt_testbed@jtpox-VirtualBox:/home/jtpox$ sudo mn --custom Escenario1Testbed.py --topo=mytopo
--controller=remote,ip=10.0.2.15 --link=tc
[sudo] password for jt_testbed:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 10.0.2.15:6653
Connecting to remote controller at 10.0.2.15:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s3) (h4, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>

```

Figura 12 Ejecución escenario 1 con controlador POX

Después de una ejecución exitosa del archivo python, y como muestra la Figura 13, la consola del controlador POX proporciona un mensaje donde indica que los tres conmutadores están conectados. Además, mediante el software de captura de paquetes Wireshark, en la Figura 14 se observa el mensaje *OFPT_HELLO*, el cual es usado para identificar y negociar la versión de OpenFlow soportada entre conmutador y controlador cuando una conexión TCP es establecida, confirmando que ya existe un canal de comunicación entre estos dos dispositivos (ONF, OpenFlow Switch Specification, 2013).

```

jt_testbed@jtpox-VirtualBox:/home/jtpox/pox$ sudo ./pox.py controlador
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-03 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected

```

Figura 13 Conexión exitosa entre topología y controlador POX

No.	Time	Source	Destination	Protocol	Length	Info
291	34.275306154	10.0.2.15	10.0.2.15	OpenFlow	74	Type: OFPT_HELLO
293	34.291011083	10.0.2.15	10.0.2.15	OpenFlow	86	Type: OFPT_STATS_REQUEST
295	34.291066076	10.0.2.15	10.0.2.15	OpenFlow	74	Type: OFPT_HELLO
297	34.295578583	10.0.2.15	10.0.2.15	OpenFlow	290	Type: OFPT_FEATURES_REPLY
298	34.295589671	10.0.2.15	10.0.2.15	OpenFlow	1134	Type: OFPT_STATS_REPLY

Figura 14 Mensaje OFPT_HELLO de inicio de conexión

Con un enlace exitoso, se procede a verificar la conectividad de la red mediante una prueba de ping entre cada host. Esto se realiza para posteriormente analizar el comportamiento de la red mediante dos métricas: latencia, la cual indica cuántos segundos tarda el primer paquete en viajar del origen al destino (Tanenbaum, 2012), y jitter, o variación en los tiempos de llegada de los paquetes o variación del retardo (Tanenbaum, 2012).

La Figura 15 muestra las pruebas de ping realizadas desde el host 1 hacia cada extremo de la red mediante el comando `h1 ping h(2,3,4) -c 5`, donde `-c 5` especifica que en cada ping se procederá a enviar 5 paquetes.

```
mininet> h1 ping h2 -c 5
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=15.7 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.163 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.038 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.038 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.037 ms
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.037/3.213/15.793/6.290 ms
mininet> h1 ping h3 -c 5
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=60.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.239 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.229 ms
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.041/12.148/60.189/24.020 ms
mininet> h1 ping h4 -c 5
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=68.8 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.274 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.038 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.208 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.035 ms
--- 10.0.0.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/ava/max/mdev = 0.035/13.873/68.810/27.468 ms
```

Figura 15 Prueba de conectividad controlador POX escenario 1

Finalmente, Wireshark también permite verificar la correcta transmisión de los paquetes entre hosts al momento de realizar las pruebas de conectividad mediante el mensaje *OFPT_PACKET_IN* como muestra la Figura 16. Este, sólo es enviado entre dispositivos si se ha logrado de manera exitosa un intercambio de paquetes en la conexión (ONF, OpenFlow Switch Specification, 2013).

No.	Time	Source	Destination	Protocol	Length	Info
3623	273.350643034	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
3630	273.353702080	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
3699	303.546677712	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
3702	303.577307392	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
3782	334.698650896	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
No.	Time	Source	Destination	Protocol	Length	Info
4045	437.622234562	10.0.0.1	10.0.0.3	OpenFlow	182	Type: OFPT_PACKET_IN
4047	437.624167461	10.0.0.1	10.0.0.3	OpenFlow	182	Type: OFPT_PACKET_IN
4049	437.626117146	10.0.0.1	10.0.0.3	OpenFlow	182	Type: OFPT_PACKET_IN
4051	437.628058008	10.0.0.3	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
4053	437.630077828	10.0.0.3	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
No.	Time	Source	Destination	Protocol	Length	Info
4208	481.153748757	10.0.0.1	10.0.0.4	OpenFlow	182	Type: OFPT_PACKET_IN
4210	481.154451213	10.0.0.1	10.0.0.4	OpenFlow	182	Type: OFPT_PACKET_IN
4212	481.155164352	10.0.0.4	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
4214	481.155813800	10.0.0.4	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
4216	481.158618883	10.0.0.4	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN

Figura 16 Mensajes OpenFlow de conexión exitosa

3.2.2 ESCENARIO 1 - FLOODLIGHT

Para empezar con la ejecución correspondiente al controlador Floodlight, y ya desde la consola de la VM, primero es necesario dirigirse al directorio donde se encuentra el archivo python con la topología programada, en este caso */home/floodlight/floodlight*, desde allí se ejecuta el archivo python como se indica en la Figura 17. Aquí, inmediatamente después de la ejecución se observa cómo se adiciona y conecta el controlador para después proseguir con la creación de la red y sus demás dispositivos y enlaces.

```
jt_testbed@floodlight:/home/floodlight/floodlight$ sudo mn --custom Escenario1Testbed.py
--topo=mytopo --controller=remote,ip=10.0.2.15,port=6653 --link=tc
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s2) (h2, s2) (h3, s3) (h4, s3) (s1, s3) (s2, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```

Figura 17 Ejecución escenario 1 con controlador Floodlight

A diferencia del controlador POX, Floodlight proporciona una interfaz gráfica a la cual se puede acceder mediante un navegador web ingresando la dirección IP del controlador además de un puerto predeterminado, el cual en este caso es el 8080. En la Figura 18 se muestra como luce la interfaz de una conexión exitosa, donde el uptime o tiempo de ejecución indica que el controlador se encuentra en funcionamiento y además de que posee una condición “true”, que en este caso se refiere a una conexión activa y sin errores.

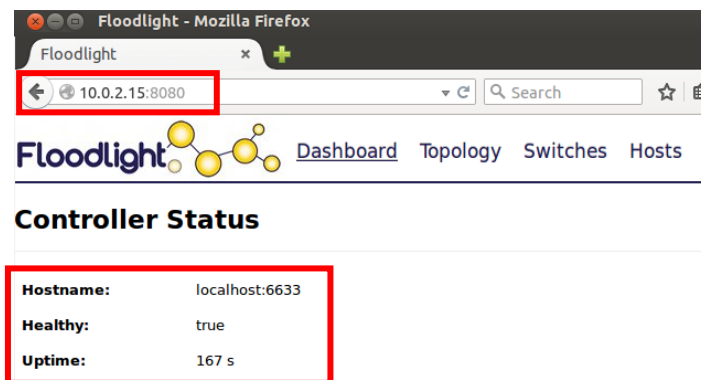


Figura 18 Interfaz gráfica controlador Floodlight

Además, en la parte inferior de la página principal de esta interfaz existe información acerca de la cantidad de conmutadores (switches) y hosts existentes en la red SDN en tiempo real. En la Figura 19 se puede observar cómo cada dispositivo posee un

tipo diferente de información básica. En el caso de los conmutadores, se muestra su dirección IP, proveedor de tecnología SDN o vendor, número de paquetes y bytes procesados, flujos existentes en la tabla de flujos o flows, inicio de conexión y un identificador de rutas de datos o DPID por sus siglas en inglés, el cual no es más que un campo que sirve para la dirección MAC del conmutador más un complemento que puede variar desde una Vlan ID o incluso simplemente estar vacío (Ribeiro, 2018). En el caso de los hosts, se indica su dirección MAC e IP, puerto del conmutador al que están conectados y su última conexión.

Switches (3)

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:00:00:00:00:03	/10.0.2.15:50440	Nicira, Inc.	36	4270	5	3/2/2022, 4:05:40 PM
00:00:00:00:00:00:02	/10.0.2.15:50441	Nicira, Inc.	45	6102	5	3/2/2022, 4:05:40 PM
00:00:00:00:00:00:01	/10.0.2.15:50439	Nicira, Inc.	42	5836	5	3/2/2022, 4:05:40 PM

Hosts (4)

MAC Address	IP Address	Switch Port	Last Seen
86:03:54:ff:3e:7e	10.0.0.1	00:00:00:00:00:00:02-1	3/2/2022, 4:06:05 PM
c6:2e:b7:7f:fc:1c	10.0.0.3	00:00:00:00:00:00:03-1	3/2/2022, 4:05:54 PM
aa:0a:4c:71:06:59	10.0.0.2	00:00:00:00:00:00:02-2	3/2/2022, 4:05:50 PM
ee:2f:a7:ec:9f:2d	0.0.0.0,10.0.0.4	00:00:00:00:00:00:03-2	3/2/2022, 4:06:05 PM

Figura 19 Información página principal interfaz Floodlight

Con un enlace exitoso, se procede a verificar la conectividad de la red mediante una prueba de ping entre cada host, de la misma forma que se realizó con el anterior controlador. La Figura 20 muestran la ejecución y resultados de conectividad realizadas desde un host hacia cada extremo de la red.


```

mininet> h1 ping h2 -c 4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.49 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.126 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.030 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.027 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.027/0.920/3.499/1.489 ms
mininet> h1 ping h3 -c 4
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=7.40 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.149 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.034 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.033 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.033/1.904/7.400/3.173 ms
mininet> h1 ping h4 -c 4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=7.44 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.159 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.035 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.033 ms

--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.033/1.917/7.443/3.190 ms

```

Figura 20 Prueba de conectividad controlador Floodlight escenario 1

Finalmente, y con cada dispositivo ya convergente en la red, la interfaz gráfica de Floodlight nos permite visualizar la topología activa de la red SDN en tiempo real. Como indica la Figura 21, la interfaz muestra cada dispositivo y su valor DPID correspondiente.

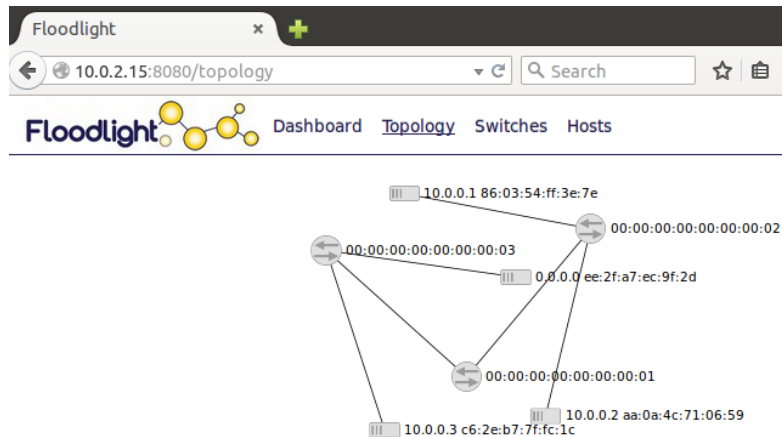


Figura 21 Topología de activa de red SDN

3.2.3 ESCENARIO 2 - POX

La ejecución de la red SDN del escenario 2 correspondiente al controlador POX es similar al escenario 1 - POX.

```

jt_testbed@jtpox-VirtualBox:/home/jtpox$ sudo mn --custom Escenario2Testbed.py --topo=mytopo
--controller=remote,ip=10.0.2.15 --link=tc
*** Creating network
*** Adding controller
Unable to contact the remote controller at 10.0.2.15:6653
Connecting to remote controller at 10.0.2.15:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s3) (h4, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>

```

Figura 22 Ejecución escenario 2 con controlador POX

Después de una ejecución exitosa del archivo python, y como indica la Figura 23, la consola del controlador POX proporciona un mensaje donde indica que los tres conmutadores están conectados. Además, y de igual manera que en el escenario anterior,

en la Figura 24 se observa el mensaje `OFPT_HELLO` confirmando la existencia del canal de comunicación entre controlador y conmutador.

```
jt_testbed@jtpox-VirtualBox:/home/jtpox/pox$ sudo ./pox.py controlador
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-03 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
```

Figura 23 Conexión exitosa entre topología y controlador POX

No.	Time	Source	Destination	Protocol	Length	Info
291	34.275306154	10.0.2.15	10.0.2.15	OpenFlow	74	Type: OFPT_HELLO
293	34.291011083	10.0.2.15	10.0.2.15	OpenFlow	86	Type: OFPT_STATS_REQUEST
295	34.291066076	10.0.2.15	10.0.2.15	OpenFlow	74	Type: OFPT_HELLO
297	34.295578583	10.0.2.15	10.0.2.15	OpenFlow	290	Type: OFPT_FEATURES_REPLY
298	34.295589671	10.0.2.15	10.0.2.15	OpenFlow	1134	Type: OFPT_STATS_REPLY

Figura 24 Mensaje `OFPT_HELLO` de inicio de conexión

Por otra parte, en cuanto a verificar la conectividad de la red, es similar a los casos anteriores, es decir mediante una prueba de ping entre cada host. En la Figura 25 se muestran las pruebas de conectividad realizadas desde un host hacia cada extremo de la red.

```
mininet> h1 ping h2 -c 4
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=57.6 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.172 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=0.046 ms
64 bytes from 192.168.10.2: icmp_seq=4 ttl=64 time=0.167 ms

--- 192.168.10.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.046/14.510/57.657/24.910 ms
mininet> h1 ping h3 -c 4
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=64 time=54.0 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=64 time=0.316 ms
64 bytes from 192.168.10.3: icmp_seq=3 ttl=64 time=0.054 ms
64 bytes from 192.168.10.3: icmp_seq=4 ttl=64 time=0.051 ms

--- 192.168.10.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.051/13.616/54.044/23.341 ms
mininet> h1 ping h4 -c 4
PING 192.168.10.4 (192.168.10.4) 56(84) bytes of data.
64 bytes from 192.168.10.4: icmp_seq=1 ttl=64 time=62.1 ms
64 bytes from 192.168.10.4: icmp_seq=2 ttl=64 time=0.319 ms
64 bytes from 192.168.10.4: icmp_seq=3 ttl=64 time=0.053 ms
64 bytes from 192.168.10.4: icmp_seq=4 ttl=64 time=0.665 ms

--- 192.168.10.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.053/15.796/62.147/26.761 ms
```

Figura 25 Prueba de conectividad controlador POX escenario 2

Finalmente, Wireshark también permite verificar la correcta transmisión de los paquetes entre hosts al momento de realizar las pruebas de conectividad mediante el mensaje `OFPT_PACKET_IN` como muestra la Figura 26.

5409	100.781962543	192.168.10.1	192.168.10.2	OpenFlow	184	Type: OFPT_PACKET_IN
5418	100.785024884	192.168.10.2	192.168.10.1	OpenFlow	184	Type: OFPT_PACKET_IN
5943	113.476282354	192.168.10.1	192.168.10.3	OpenFlow	184	Type: OFPT_PACKET_IN
5947	113.479314755	192.168.10.1	192.168.10.3	OpenFlow	184	Type: OFPT_PACKET_IN
5951	113.482184451	192.168.10.1	192.168.10.3	OpenFlow	184	Type: OFPT_PACKET_IN
5955	113.485158092	192.168.10.3	192.168.10.1	OpenFlow	184	Type: OFPT_PACKET_IN
5959	113.488018828	192.168.10.3	192.168.10.1	OpenFlow	184	Type: OFPT_PACKET_IN
5963	113.490924411	192.168.10.3	192.168.10.1	OpenFlow	184	Type: OFPT_PACKET_IN
6144	119.950095131	192.168.10.1	192.168.10.4	OpenFlow	184	Type: OFPT_PACKET_IN
6148	119.952970317	192.168.10.1	192.168.10.4	OpenFlow	184	Type: OFPT_PACKET_IN
6152	119.955912843	192.168.10.1	192.168.10.4	OpenFlow	184	Type: OFPT_PACKET_IN

Figura 26 Mensajes OpenFlow de conexión exitosa

3.2.4 ESCENARIO 2 - FLOODLIGHT

La ejecución del escenario 2 por parte de este controlador se realiza desde el directorio correspondiente donde se encuentra el archivo Python de la topología programada, en este caso `/home/floodlight/Floodlight`, desde allí se ejecuta el archivo como se indica en la Figura 27. A partir de la ejecución, se observa cómo se adiciona y conecta el controlador, así como se crean los demás dispositivos y enlaces.

```

jt_testbed@floodlight:/home/floodlight/floodlight$ sudo mn --custom Escenario2Testbed.py
--topo=mytopo --controller=remote,ip=10.0.2.15,port=6653 --link=tc
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s2) (h2, s2) (h3, s3) (h4, s3) (s1, s3) (s2, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>

```

Figura 27 Ejecución escenario 2 con controlador Floodlight

Como indica la Figura 28, mediante la interfaz gráfica de Floodlight, se puede confirmar una conexión satisfactoria. Además, en esta se observa el tiempo activo de

ejecución del controlador y su estado sin errores debido al valor *true* de la fila de salud (*Healthy*).

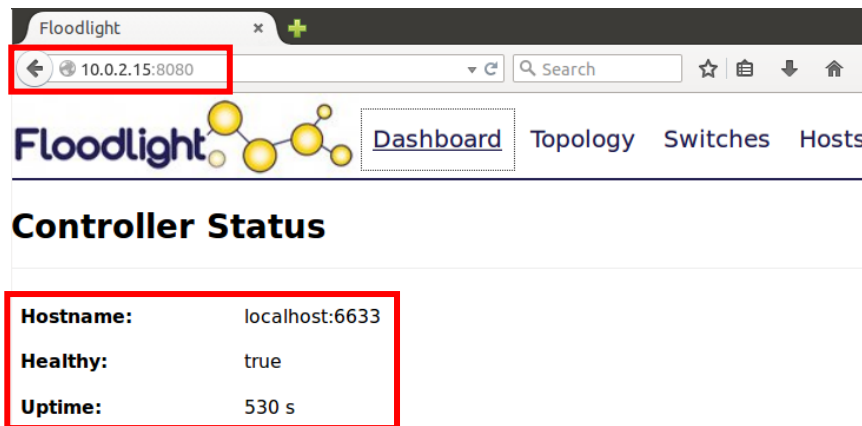


Figura 28 Interfaz gráfica Floodlight

Además, a través de la interfaz se verifica la cantidad de conmutadores y hosts existentes en la red SDN. La Figura 29 indica cada dispositivo con su información básica en tiempo real.

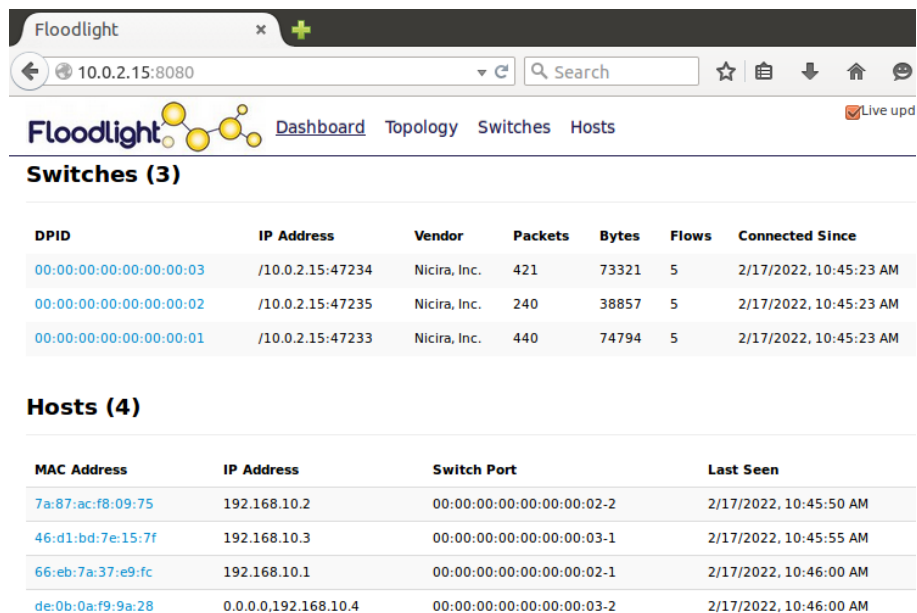


Figura 29 Información página principal interfaz Floodlight

Con un enlace exitoso, tal como se muestra en la Figura 30 se procede a verificar la conectividad de la red mediante una prueba de ping entre cada host de la misma forma cómo se realizó con el anterior controlador.

```
mininet> h1 ping h2 -c 4
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=4.74 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.167 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=0.035 ms
64 bytes from 192.168.10.2: icmp_seq=4 ttl=64 time=0.032 ms

--- 192.168.10.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.032/1.244/4.744/2.021 ms
mininet> h1 ping h3 -c 4
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=64 time=14.5 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=64 time=0.201 ms
64 bytes from 192.168.10.3: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 192.168.10.3: icmp_seq=4 ttl=64 time=0.043 ms

--- 192.168.10.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.043/3.721/14.599/6.280 ms
mininet> h1 ping h4 -c 4
PING 192.168.10.4 (192.168.10.4) 56(84) bytes of data.
64 bytes from 192.168.10.4: icmp_seq=1 ttl=64 time=14.8 ms
64 bytes from 192.168.10.4: icmp_seq=2 ttl=64 time=0.203 ms
64 bytes from 192.168.10.4: icmp_seq=3 ttl=64 time=0.040 ms
64 bytes from 192.168.10.4: icmp_seq=4 ttl=64 time=0.041 ms

--- 192.168.10.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.040/3.774/14.815/6.374 ms
```

Figura 30 Pruebas de conectividad controlador Floodlight escenario 2

Finalmente, y como muestra la Figura 31, la interfaz gráfica de Floodlight permite visualizar la topología SDN en tiempo real, así como el valor DPID correspondiente de cada dispositivo.

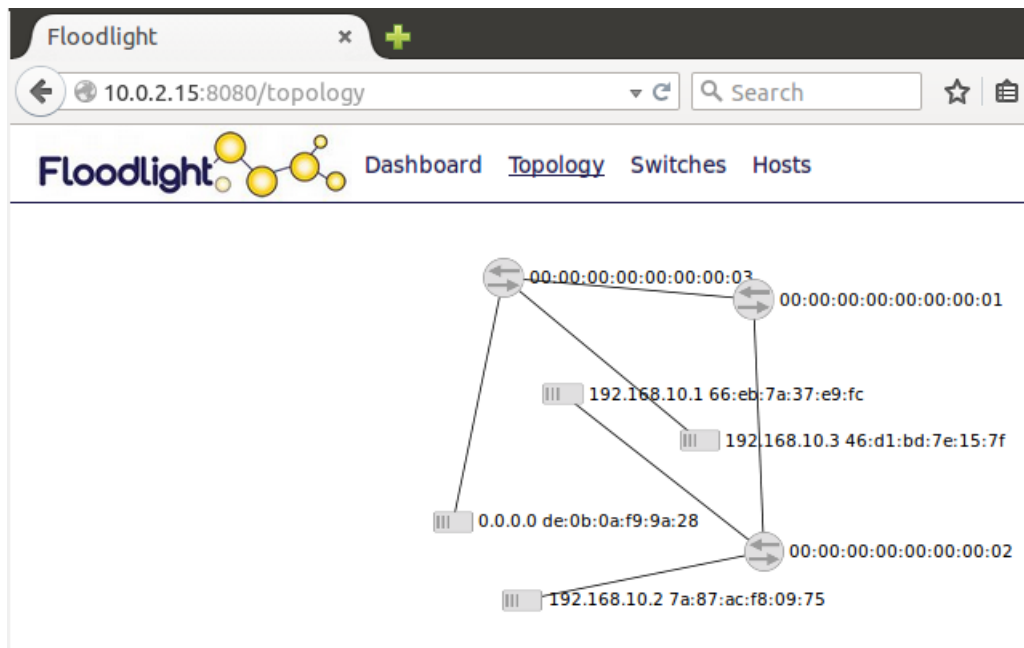


Figura 31 Topología activa de la red SDN

3.3 IMPLEMENTACIÓN DEL TESTBED SDN EN EL SERVIDOR FICA UTN.

3.3.1 SITUACIÓN ACTUAL DEL CENTRO DE DATOS

La Facultad de Ingeniería en Ciencias Aplicadas (FICA) posee un centro de datos el cual se encuentra dentro de la oficina de la Carrera de Ingeniería en Telecomunicaciones, ubicada en la planta baja del edificio FICA. La Figura 32 presenta un diagrama de la infraestructura actual del centro de datos, además de indicar las diferentes dimensiones físicas que este posee según (Nicolalde, 2021).

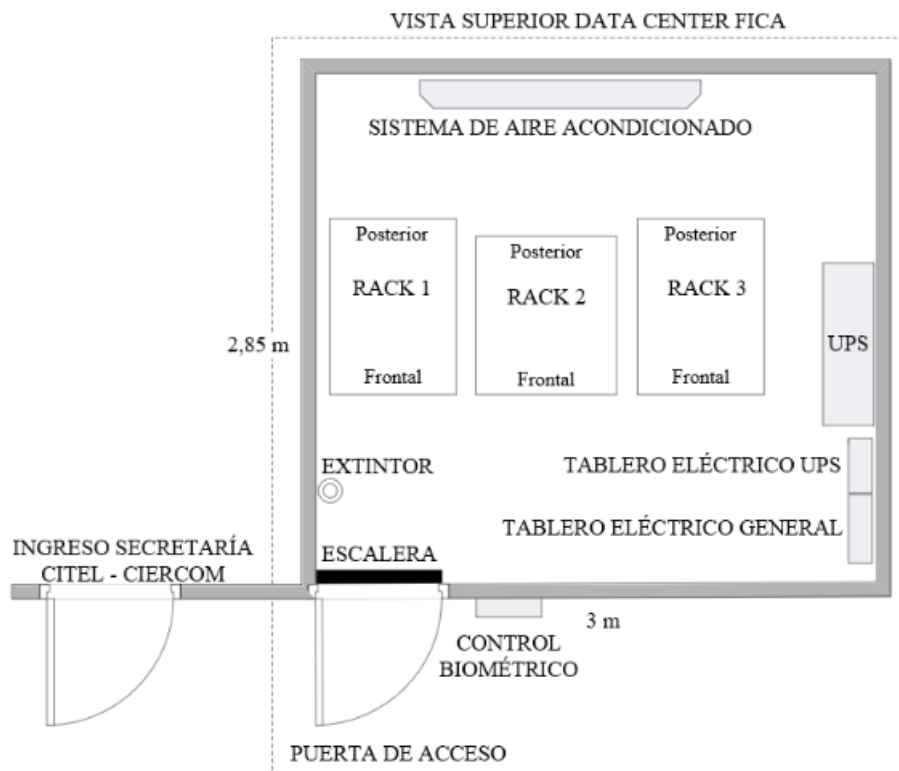


Figura 32 Infraestructura del centro de datos FICA

Fuente: (Nicolalde, 2021)

De los tres racks que se encuentran en el centro de datos, en el rack número 2 se encuentran ubicados tres servidores marca HP modelo ProLiant DL360 G9 denominados Proxmox Virtual (PV1, PV2 y PV3), los cuales brindan la capacidad para virtualizar equipos, y en dónde en uno de ellos (PV2), se implementará el Testbed SDN.

La arquitectura lógica de esta infraestructura, tomando como referencia a (Guerrero, 2019) está representada en la Figura 33. Esta posee un switch core Catalyst el cual funciona como un equipo de frontera y es el que proporciona acceso a internet a toda la red. También, existen los switch 3com 3226, Lynksys y QPcom, los cuales sólo son utilizados para la integración de los demás equipos de la red ya que estos son únicamente dispositivos de capa 2. Finalmente, el switch 3com 4500G y el router Mikrotik 1100 se

centran en la administración de la red. La arquitectura desplegada tiene el nombre de hiperconvergente, ya que esta permite la combinación de componentes físicos y virtuales en el centro de datos como lo son servidores, redes y hardware de almacenamiento, a partir del uso de un único dispositivo virtual manejado a partir de software, en este caso Proxmox (Guerrero, 2019).

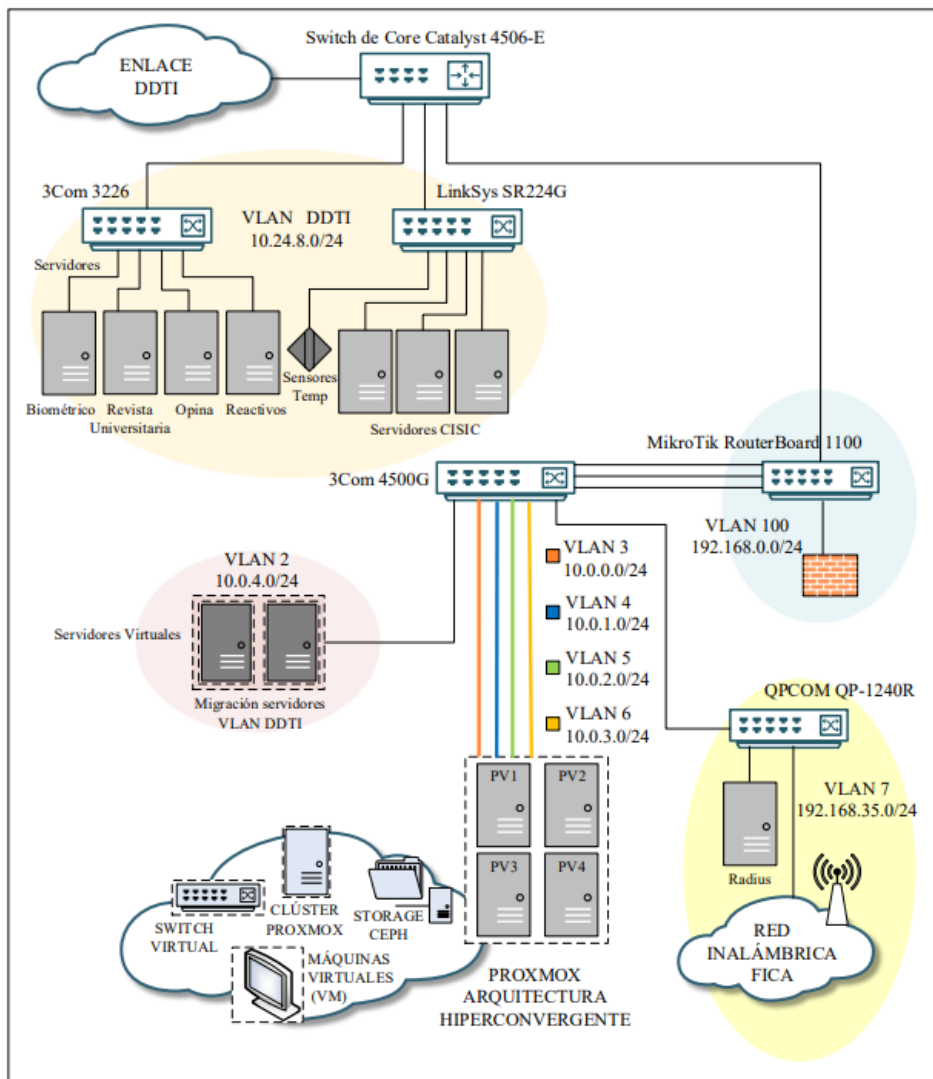


Figura 33 Topología lógica del centro de datos FICA

3.3.2 INSTALACIÓN Y CONFIGURACIÓN DEL TESTBED SDN

Para la instalación del Testbed SDN se planteó importar la MV creada y diseñada con anterioridad a partir de un archivo `.ova` proporcionado por VirtualBox, lo que brinda la posibilidad de trabajar en exactamente la misma MV que fue puesta a prueba en la fase de desarrollo descrita en el apartados 3.2 . Para esto, como se indica en la Figura 34, en el segundo servidor virtualizado Proxmox (PVE2), se crea una máquina virtual llamada *110 (juantapia)* con las mismas características en hardware que la MV creada en VirtualBox, y que posee 8.00 GB de memoria RAM y 8.00 GB de memoria en disco duro, de acuerdo con el análisis realizado en el apartado 3.1.

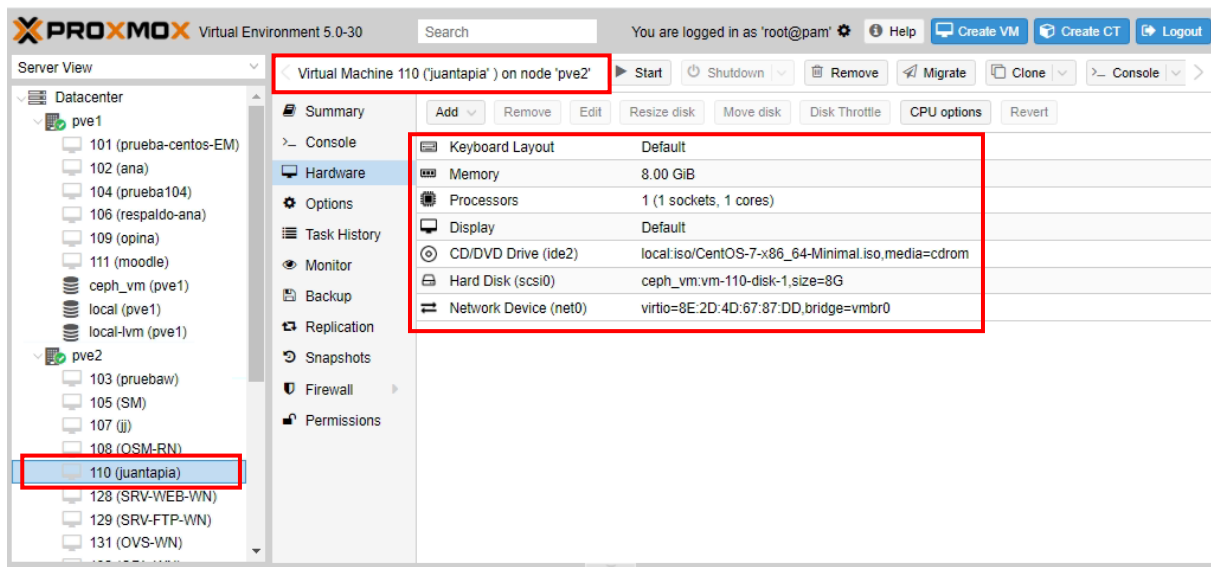


Figura 34 Máquina virtual 110 juantapia creada en PVE2

Ya que se necesita usar el archivo `.ova`, se procede a copiar este a la máquina donde se encuentra el servidor Proxmox a partir desde nuestro equipo mediante el software WinSCP, el cual es un cliente SFTP que usa SSH para Windows. Las Figura 35 y 36 muestran este proceso, dónde finalmente el archivo fue copiado a una carpeta en el directorio principal llamada `/home/juan`.

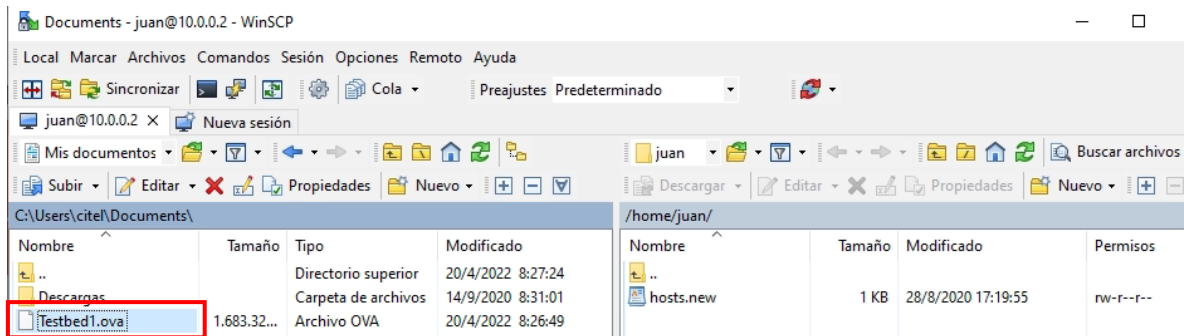


Figura 35 Archivo .ova en nuestra máquina

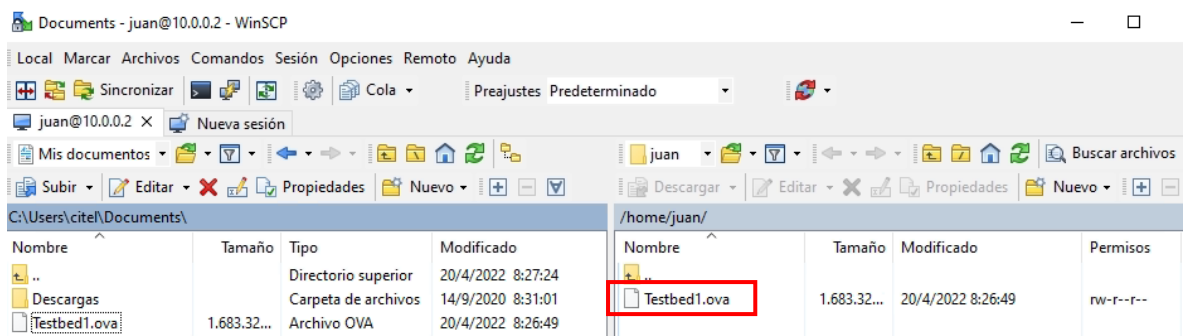


Figura 36 Archivo .ova copiado a la máquina Proxmox

Ahora, desde la consola de pve2, se mueve el archivo ubicado en la carpeta `/home/juan` al directorio `/var/lib/vz` como indica la Figura 37, y una vez aquí, a partir del comando `tar xvf Testbed.ova` se procede a descomprimir el archivo `.ova` como muestra la Figura 38.

```
root@pve2:~# cd /home/juan
root@pve2:/home/juan# ls
Testbed1.ova
root@pve2:/home/juan# mv Testbed1.ova /var/lib/vz
```

Figura 37 Archivo .ova se mueve al directorio /var/lib/vz

```
root@pve2:~# cd /var/lib/vz
root@pve2:/var/lib/vz# ls
dump hosts.new images private root template Testbed1.ova
root@pve2:/var/lib/vz# tar xvf Testbed1.ova
Testbed1.ovf
Testbed1-disk001.vmdk
Testbed1.mf
root@pve2:/var/lib/vz#
```

Figura 38 Archivo .ova descomprimido

Inmediatamente se procede a eliminar el disco duro existente como muestra la Figura 39 debido a que es necesario reemplazar a este con el disco extraído a partir del archivo `.ova`. A continuación, se procede a importar el disco duro descomprimido a partir del archivo `.ova` llamado `Testbed1-disk001` que se encuentra en el directorio `/var/lib/vz` mediante el comando `qm import disk 110 'Testbed1-disk001.vmdk' local -format qcow2` como indica la Figura 40.

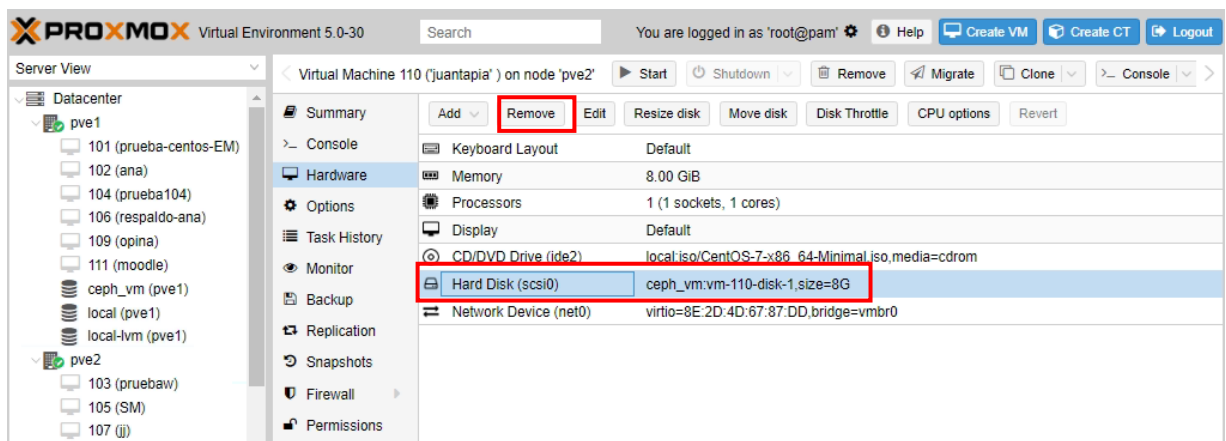


Figura 39 Eliminación disco duro existente

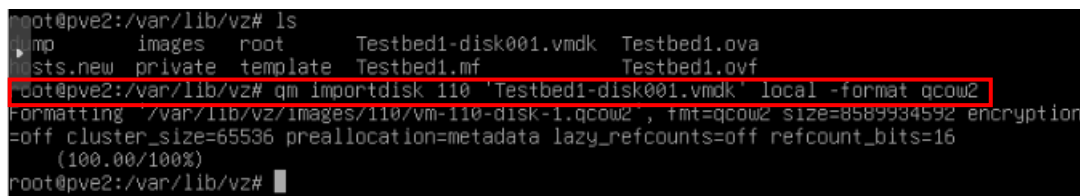


Figura 40 Importación disco duro Testbed-001.vmdk

Ahora, como indica la Figura 41, en la pestaña de hardware, la MV aparecerá con un disco sin usar, el cual se procederá a configurar mediante la opción `add` del menú principal. A continuación, como muestra la Figura 42, en la ventana emergente aparecerá un apartado de selección de Bus/Dispositivo, donde se seleccionará la opción `VirtIO Block`, y finalmente, en el apartado de imagen de disco, se usará el cual fue importado anteriormente.

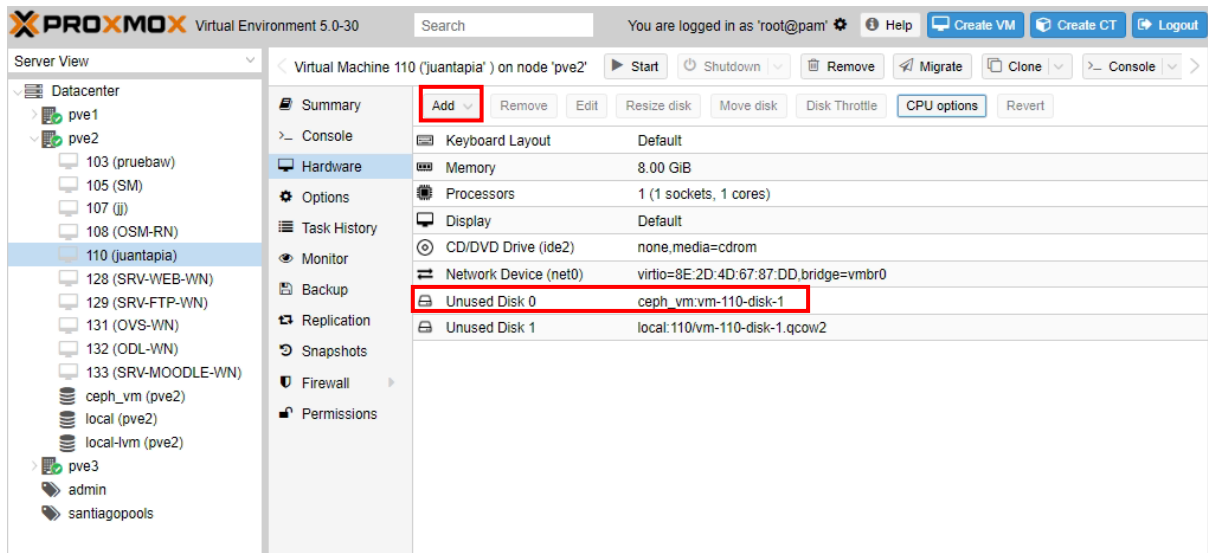


Figura 41 Disco sin usar

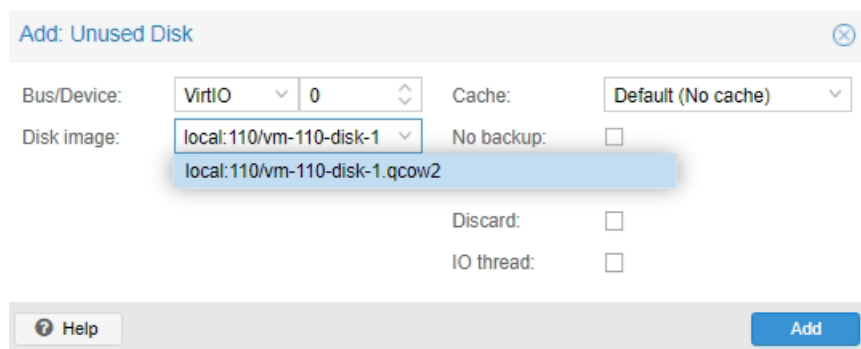


Figura 42 Configuración de disco sin usar

Finalmente, se logra la importación correcta del nuevo disco duro tal y como indica la Figura 43, para posteriormente encender la MV mediante la pestaña Start en el menú principal de la máquina.

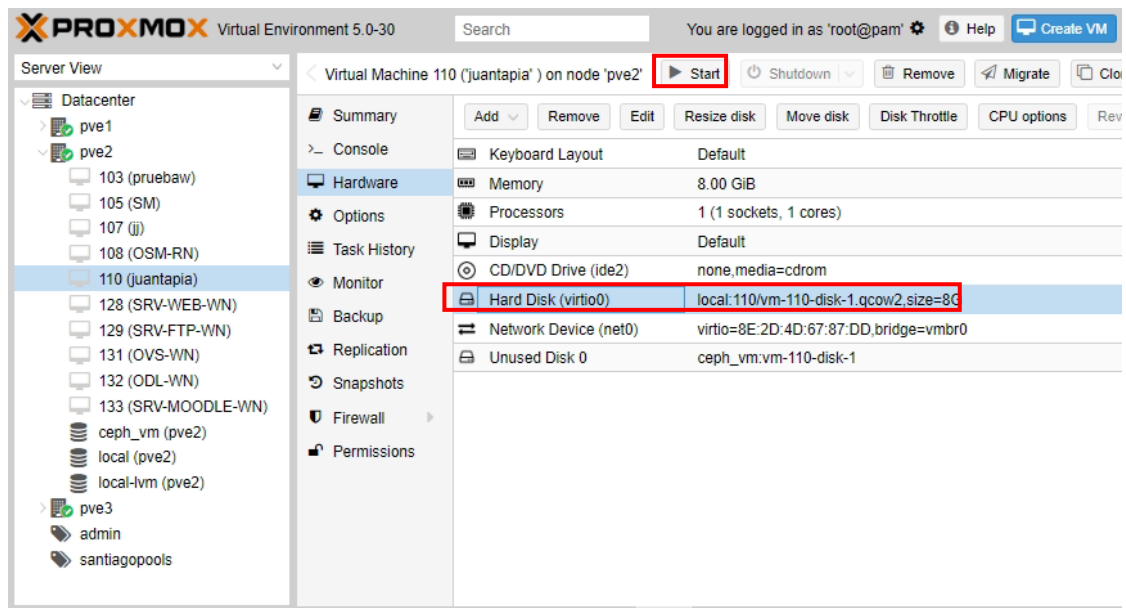


Figura 43 Verificación de disco duro y encendido de MV

Para usar la MV hay que dirigirse a la pestaña de consola como muestra la Figura 44. Mientras se enciende la MV, se accede al menú de arranque a partir del BIOS y se utiliza la opción 3, la cual permite iniciar la máquina a partir del disco que se agregó anteriormente. Si todo está correcto, la MV encenderá sin ningún problema como se observa en la Figura 45.

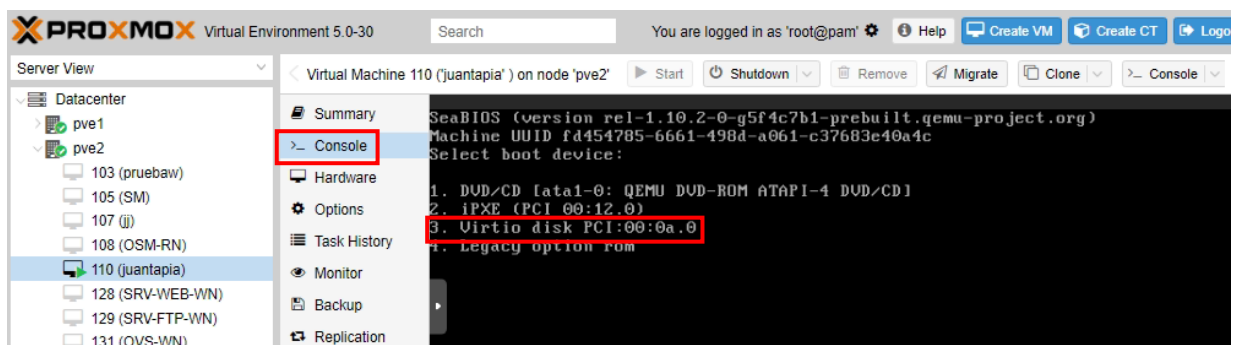


Figura 44 Arranque a partir de BIOS

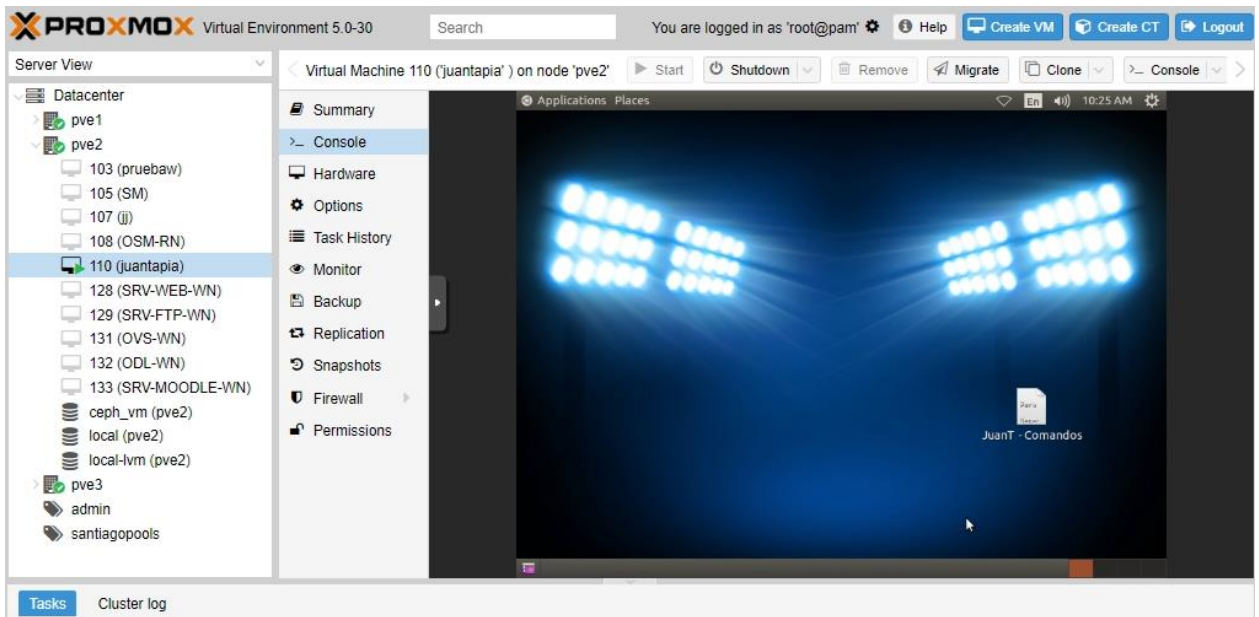


Figura 45 MV encendida sin errores

Para la configuración de red de la MV se accede y modifica al archivo *interfaces* mediante el comando `sudo nano /etc/network/interfaces` en la consola de la máquina como indica la Figura 46. En este se agrega un direccionamiento estático, donde la MV poseerá una dirección IPv4 10.0.0.10, una máscara de red 255.255.255.0, y un puerto de enlace con direccionamiento IPv4 10.0.0.254.

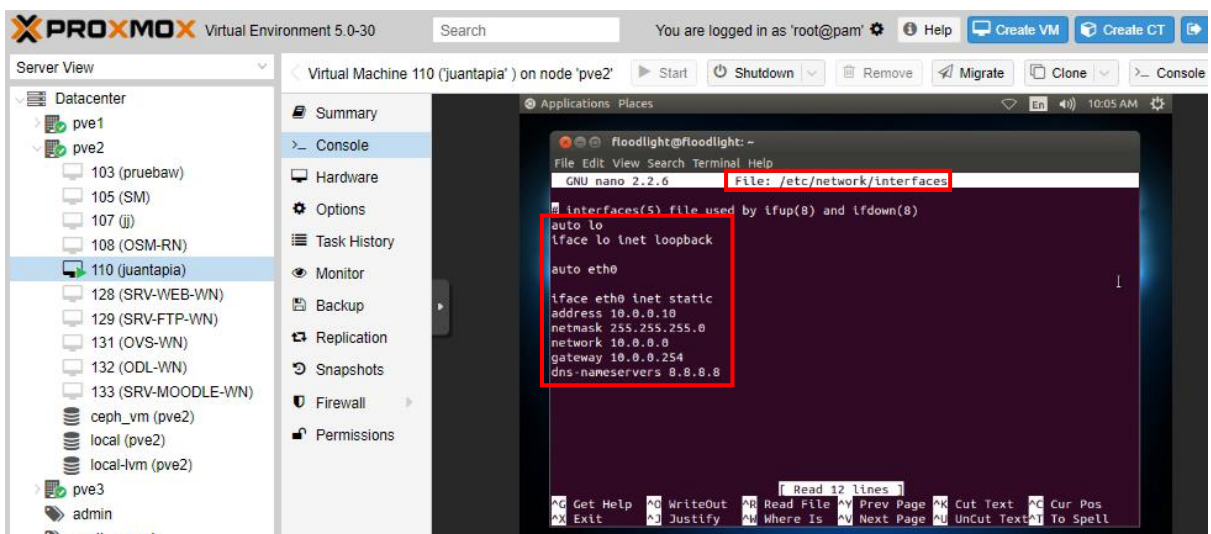


Figura 46 Direccionamiento IPv4

Con la MV y su direccionamiento IPv4 establecido, el siguiente paso es realizar un NAT para así lograr tener acceso a la máquina del Testbed SDN, la cual se encuentra en una red privada por medio de una IP pública, o, en otras palabras, permitir el acceso al Testbed a una máquina cualquiera que se conecte a la red Eduroam de la UTN. Para esto, se accede al router MikroTik 1100 del centro de datos con dirección IPv4 192.168.32.1 y se selecciona la pestaña Firewall del menú IP como muestra la Figura 47.

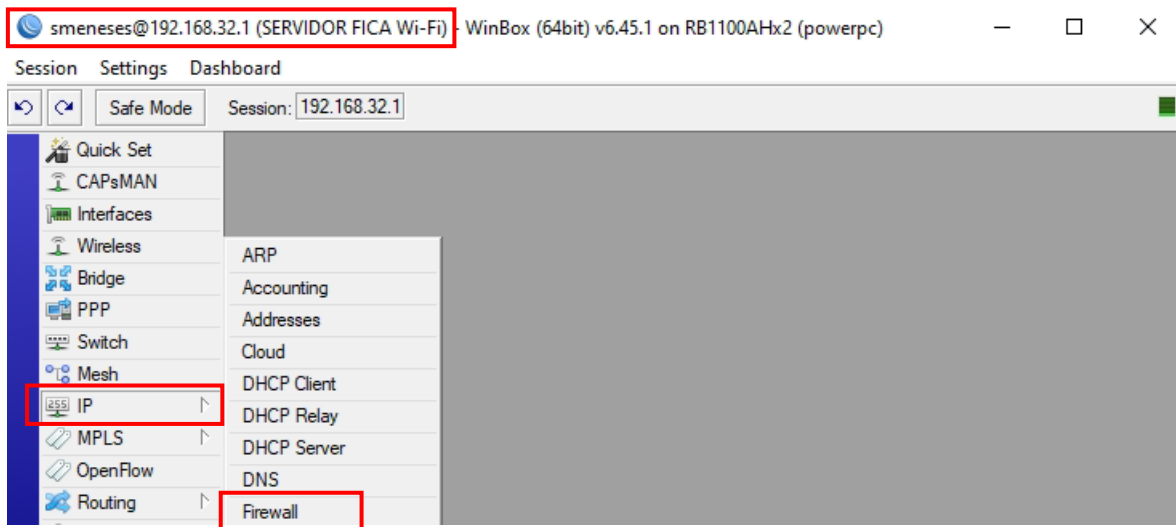


Figura 47 Acceso router MikroTik 1100

En esta ventana, en la pestaña NAT, se procede a agregar dos reglas como se muestra en la Figura 48. La primera, que se permita el acceso al servidor Proxmox pve con direccionamiento 10.0.0.3 y un puerto 8006 a partir de una IP pública 10.24.8.65 y un puerto 8089. Y una segunda regla, que permita el uso del protocolo de acceso remoto o SSH hacia la MV del Testbed con direccionamiento IPv4 10.0.0.10 a partir de una IP pública 10.24.8.65 mediante el puerto 22.

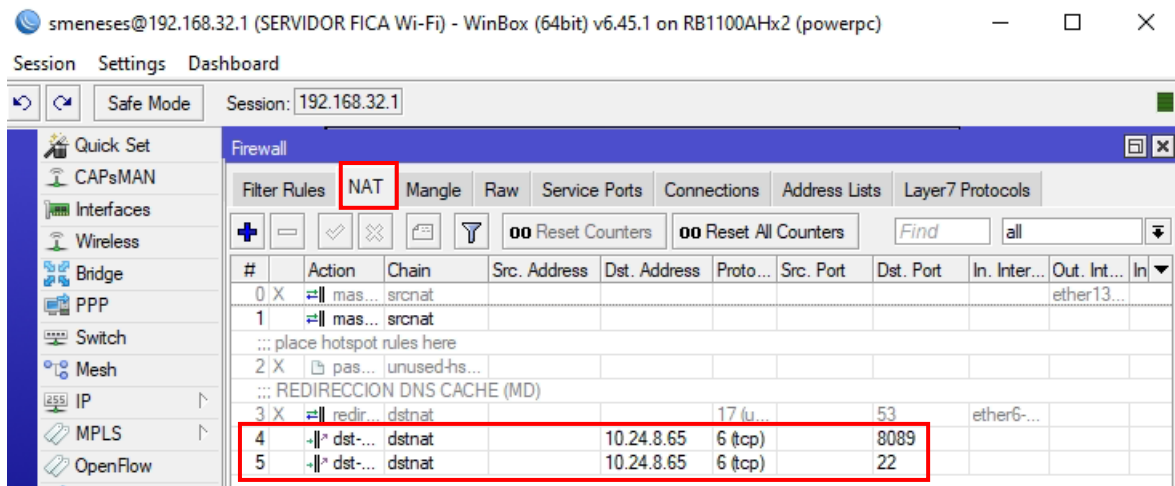


Figura 48 Configuración de NAT

Para la primera regla, las configuraciones generales se realizan como indica la Figura 49. En la pestaña de cadena (Chain) se selecciona la opción *dstnat* (destination o destino), la dirección de destino (Dst. Address) es la dirección IPv4 pública, en este caso 10.24.8.65, el protocolo utilizado es TCP, y el puerto de destino 8089, o cualquiera que no se encuentre en uso.

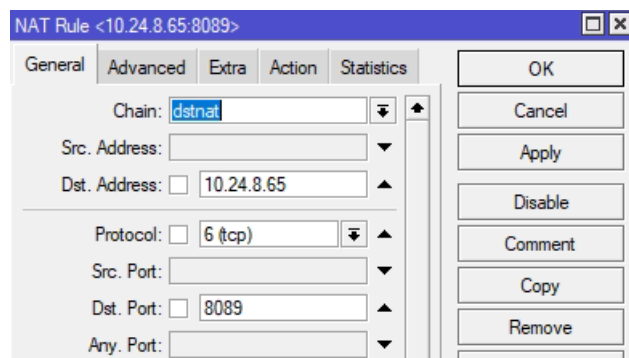


Figura 49 Configuraciones generales de NAT primera regla

Adicionalmente, también se realizan las configuraciones de acción (Action) cómo se observa en la Figura 50. En esta ventana, en la pestaña de acción, se selecciona de igual manera la opción *dest-nat*, a continuación, en la dirección "To Addresses" se utiliza el direccionamiento IPv4 del servidor Proxmox 10.0.0.3, y finalmente, el puerto predeterminado Proxmox 8006.

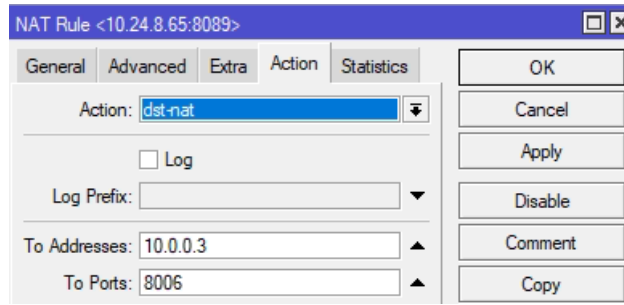


Figura 50 Configuraciones de acción de NAT

Para verificar esta configuración NAT, y a partir de una máquina cualquiera conectada a la red Eduroam UTN, se procede a acceder al servidor Proxmox mediante la dirección pública 10.24.8.65 y el puerto configurado 8089. Como se observa en la Figura 51, la conexión es exitosa ya que se logra acceder a la interfaz de Login del servidor Proxmox a partir de una máquina con direccionamiento 172.16.100.147 la cual está conectada a la red UTN.

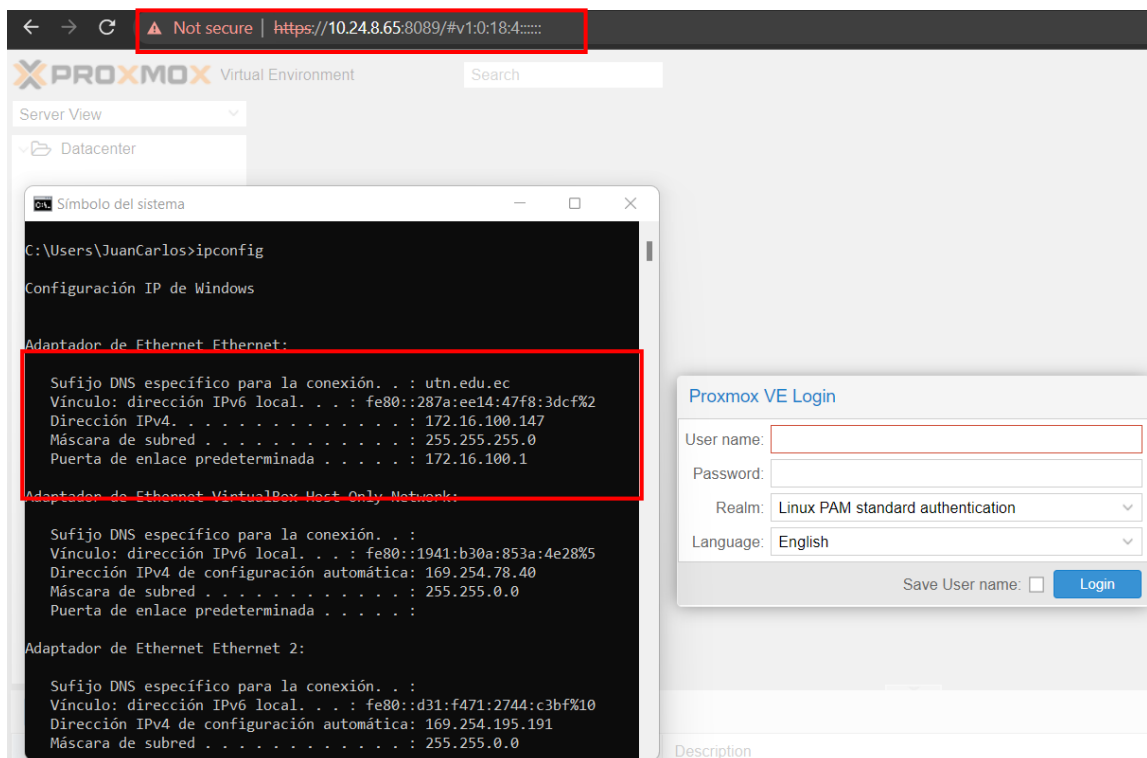


Figura 51 Verificación de Nateo

A continuación, para la segunda regla, las configuraciones generales se realizan tal y como indica la Figura 52. En la pestaña de cadena se selecciona la opción *dstnat*, la dirección de destino IPv4 es 10.24.8.65, el protocolo utilizado es TCP, y el puerto por defecto para una conexión SSH, el número 22.

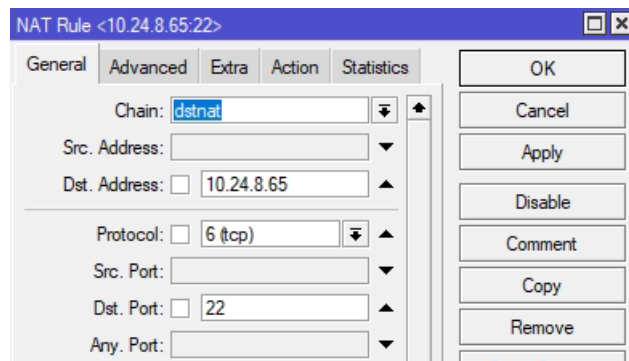


Figura 52 Configuraciones generales de NAT segunda regla

Posteriormente, también se realizan las configuraciones de acción como se muestra en la Figura 53. En esta ventana, en la pestaña de acción, se selecciona de igual manera la opción *dst-nat*, a continuación, en la dirección "To Addresses" se utiliza el direccionamiento IPv4 de la MV que contiene al Testbed SDN 10.0.0.10, y finalmente, el puerto predeterminado SSH 22.

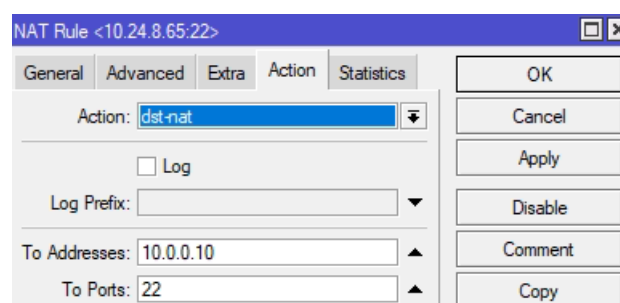


Figura 53 Configuraciones de acción de NAT

Para la verificación de esta regla, tal y como indica la Figura 54, se utiliza la plataforma de cliente SSH "PuTTY" y se accede al Testbed SDN desde una máquina cualquiera conectada a la red UTN a partir de la dirección pública 10.24.8.65 y el puerto

22, donde finalmente, y como se puede observar en la Figura 55, el acceso a la MV se realiza sin ninguna clase de error y el Testbed SDN está listo para su uso.

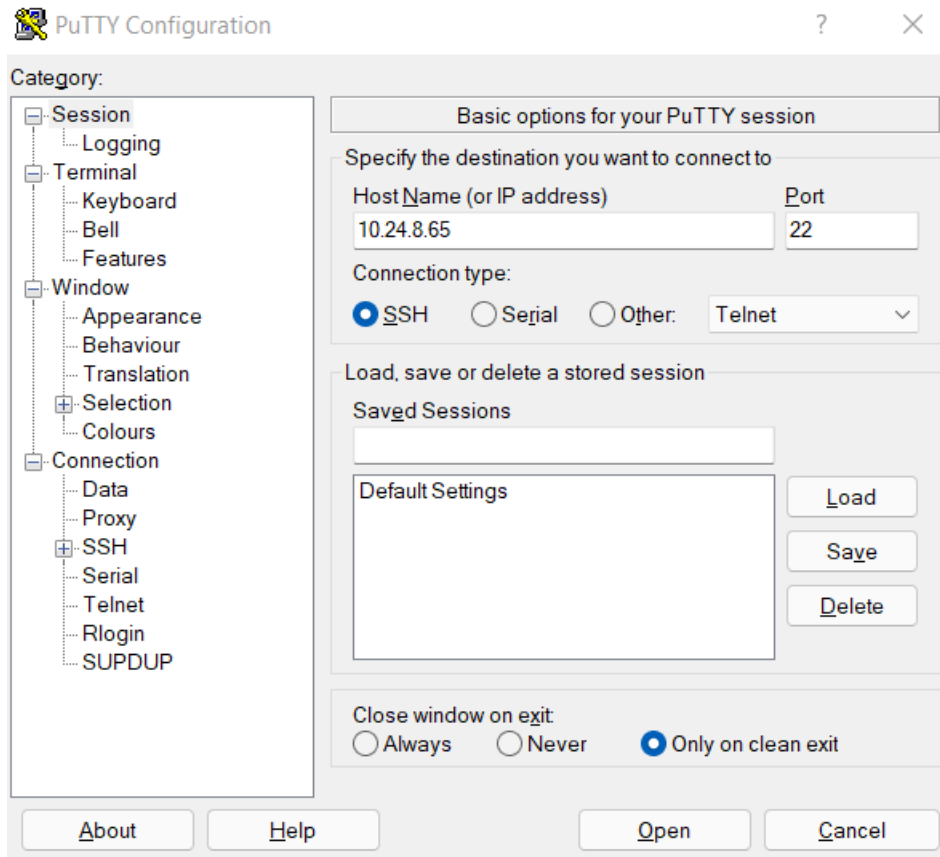


Figura 54 Ingreso al Testbed SDN mediante SSH

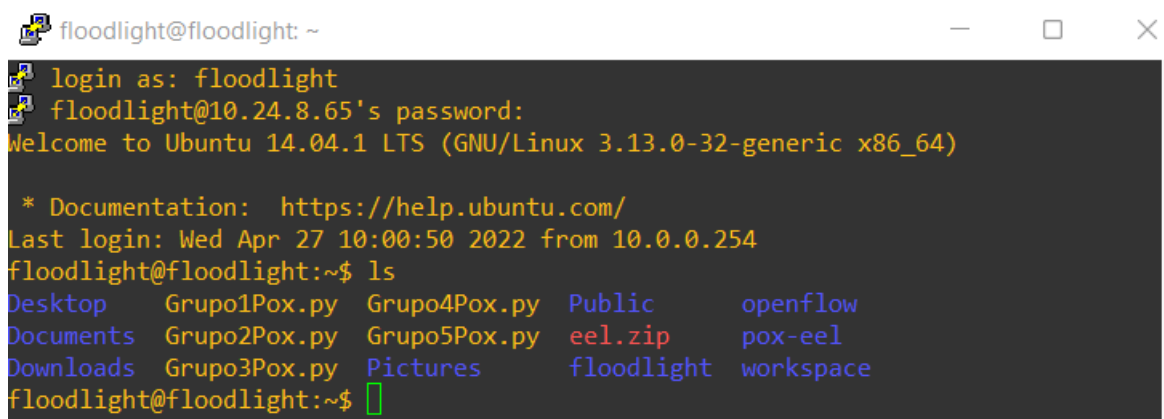


Figura 55 Testbed SDN funcionando correctamente

3.4 AMBIENTES DE PRÁCTICA SDN A TRAVÉS DEL TESTBED

La elección de los diferentes ambientes de práctica del Testbed SDN se enfocó al uso de aplicaciones, las cuales puedan actuar como entrenamiento, tanto para la familiarización de la tecnología SDN en los estudiantes, así como para su análisis y comparación del funcionamiento de estas aplicaciones con las redes tradicionales. Por consiguiente, se han propuesto tres ambientes de práctica en el ambiente SDN Mininet; el primero funciona como introducción al manejo de información y tablas de flujo a través del controlador Pox, el segundo ambiente se enfoca en la creación y aplicación de listas de control de acceso en la red SDN mediante el controlador Floodlight, y finalmente, el tercer ambiente de práctica tiene como objetivo levantar y manejar un firewall a partir de reglas sobre el controlador Floodlight.

3.4.1 MANEJO DE COMANDOS MININET Y TABLAS DE FLUJO CON EL CONTROLADOR POX.

El objetivo de este ambiente de práctica se enfoca en conocer la herramienta SDN Mininet, sus comandos, y cómo manejar la información proporcionada por las diferentes tablas de flujo una vez que la red SDN ha sido ejecutada bajo este ambiente.

Para comenzar la práctica, sólo se inicia cualquiera de los escenarios diseñados anteriormente en este capítulo, donde para esta práctica se utiliza el escenario 2 a partir del archivo python *Escenario2Testbed.py* tal y como indica la Figura 12.

Una vez ejecutado el archivo python, y ya con la topología activa, se procede a verificar la cantidad de nodos existentes en la red mediante el comando *nodes* en la terminal Mininet, tal y como indica la Figura 56, los cuales en este caso son: *c0*, *h1*,

h2, *h3*, *h4*, *s1*, *s2* y *s4*. Adicionalmente, para una información más completa (direcciones IP, pid o identificación de proceso, puerto de controlador en uso) de cada nodo, se usa el comando *dump* como muestra la Figura 57.

```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 s1 s2 s3
```

Figura 56 Nodos de la topología SDN

```
mininet> dump
<Host h1: h1-eth0:192.168.10.1 pid=3084>
<Host h2: h2-eth0:192.168.10.2 pid=3087>
<Host h3: h3-eth0:192.168.10.3 pid=3089>
<Host h4: h4-eth0:192.168.10.4 pid=3091>
<RemoteController{'ip': '10.0.2.15'} c0: 10.0.2.15:6633 pid=3076>
```

Figura 57 Información detallada de cada nodo

Mininet también permite verificar que cada enlace programado de la red SDN se ha creado correctamente. Esto se realiza ejecutando el comando *net*, el cual proporciona información acerca de los enlaces activos de la red, así como lo indica la Figura 58. Aquí se muestra como el Host1 (*h1*), a partir de su interfaz *h1-eth0*, está conectado al conmutador *s1* en su interfaz *s1-eth1*. También se puede observar como el conmutador *s1* posee una interfaz de loopback *lo*, además de la información de sus interfaces las cuales están conectadas a los diferentes hosts y conmutadores de la red.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth1
s2 lo: s2-eth1:s1-eth3 s2-eth2:s3-eth3
s3 lo: s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s2-eth2
```

Figura 58 Enlaces activos de la red SDN

El estado de las interfaces de red de cada dispositivo también está disponible en Mininet. Esta información es proporcionada mediante el comando *n ifconfig -a*, donde *n* representa el nombre del dispositivo del cual se busca información. La Figura 59

presenta la información acerca de la interfaz `s1-eth0` del Host 1, la cual está configurada con una dirección IP 192.168.10.1.

```
mininet> h1 ifconfig -a
h1-eth0  Link encap:Ethernet  direcciónHW 8a:89:11:db:3f:56
         Direc. inet:192.168.10.1  Difus.:192.168.10.255  Másc:255.255.255.0
         Dirección inet6: fe80::8889:11ff:fedb:3f56/64  Alcance:Enlace
         ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST MTU:1500  Métrica:1
         Paquetes RX:413 errores:0 perdidos:0 overruns:0 frame:0
         Paquetes TX:20 errores:0 perdidos:0 overruns:0 carrier:0
         colisiones:0 long.colatX:1000
         Bytes RX:75706 (75.7 KB)  TX bytes:1488 (1.4 KB)
```

Figura 59 Información de la interfaz red del Host1

Ahora, para verificar las tablas de flujo de cada conmutador se utiliza el comando `dpctl dump-flows`. En el caso mostrado en la Figura 60, estas tablas están vacías debido a que aún no se ha realizado ninguna transmisión de paquetes en la red SDN.

```
mininet> dpctl dump-flows
*** s1 -----
NXST_FLOW reply (xid=0x4):
*** s2 -----
NXST_FLOW reply (xid=0x4):
*** s3 -----
NXST_FLOW reply (xid=0x4):
```

Figura 60 Tablas de flujo vacías de los conmutadores s1, s2 y s3.

Por ejemplo, para observar el cambio de la tabla de flujo del conmutador `s1`, se realiza una transmisión de datos a partir de un ping entre el Host 1 y Host 2 a través del comando `h1 ping -c 4 h2 tal` y como indica la Figura 61. Posteriormente se puede observar como la tabla de flujo del conmutador `s1` ya posee información. Los datos que esta tabla proporcionan, a partir de la documentación (OvS, 2016) son los siguientes:

- **Duration:** Número de segundos los cuales la entrada de flujo ha estado en la tabla.
- **Table:** Especifica la tabla en cual la entrada de flujo se ha ubicado.

- **N_packets:** Número de paquetes que han hecho coincidencia con la entrada.
- **Idle_age:** Número de segundos desde el último paquete.
- **Priority:** Valor del número de prioridad, de forma automática es el 66535.
- **Dl_src:** Dirección MAC de origen.
- **Dl_dst:** Dirección MAC de destino.
- **Nw_src:** Dirección IP de origen.
- **Dst_src:** Dirección IP de destino.
- **Hard_timeout:** Número de segundos que la entrada persistirá antes de ser borrada.
- **Actions=output:** El paquete es satisfactoriamente procesado para la entrada.

```

mininet> h1 ping -c 4 h2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=32.7 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.240 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=0.072 ms
64 bytes from 192.168.10.2: icmp_seq=4 ttl=64 time=0.036 ms

--- 192.168.10.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.036/8.277/32.761/14.136 ms
mininet> ovsctl dump-flows
*** s1 ***
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4.717s, table=0, n_packets=4, n_bytes=392, idle_timeout=10, hard_timeout=30, idle_age=1, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=36:b3:64:d3:01:f6,dl_dst=8a:89:11:db:3f:56,nw_src=192.168.10.2,nw_dst=192.168.10.1,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
 cookie=0x0, duration=4.719s, table=0, n_packets=4, n_bytes=392, idle_timeout=10, hard_timeout=30, idle_age=1, priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=8a:89:11:db:3f:56,dl_dst=36:b3:64:d3:01:f6,nw_src=192.168.10.1,nw_dst=192.168.10.2,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2

```

Figura 61 Tabla de flujo del conmutador s1

La información de estas tablas de flujo también se puede desechar, en este caso, como se observa en la Figura 62, primero se realiza un ping entre el Host 1 y Host 3 para verificar la conectividad entre estos dos dispositivos, posteriormente, mediante el

comando `dpctl add-flow in_port=2, actions=drop` la información de la entrada de la tabla de flujo asociada al puerto 2 se borra, y por consecuencia a esto, la conectividad entre estos dos terminales se pierde.

```
mininet> h1 ping -c 3 h3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=64 time=56.6 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=64 time=0.211 ms
64 bytes from 192.168.10.3: icmp_seq=3 ttl=64 time=0.042 ms

--- 192.168.10.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.042/18.959/56.624/26.633 ms
mininet> dpctl add-flow in_port=2,actions=drop
*** s1 -----
*** s2 -----
*** s3 -----
mininet> h1 ping -c 3 h3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.

--- 192.168.10.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2014ms
```

Figura 62 Eliminación de la información de una tabla de flujo

Finalmente, se verifica el estado de las tablas de flujo mediante el comando `dpctl dump-flows` como se muestra en la Figura 63, dónde se puede observar como las entradas de flujo asociadas al puerto 2 están en estado `drop`. Para volver a tener conectividad, simplemente se borran de forma completa las tablas de flujo con el comando `dpctl del-flows`, el cual elimina la acción de drop que se encuentra en esta tabla, y permite nuevamente el ingreso de entradas de flujo. Por consecuente, la conexión procederá a reestablecerse tal y como muestra la Figura 64.

```
mininet> dpctl dump-flows
*** s1 -----
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=61.557s, table=0, n_packets=0, n_bytes=0, idle_age=61, in_port=2 actions=drop
*** s2 -----
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=61.562s, table=0, n_packets=36, n_bytes=7031, idle_age=0, in_port=2 actions=drop
*** s3 -----
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=61.562s, table=0, n_packets=0, n_bytes=0, idle_age=61, in_port=2 actions=drop
```

Figura 63 Entradas de tablas de flujo en estado drop.

```

mininet> dpctl del-flows
*** s1 -----
*** s2 -----
*** s3 -----
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)

```

Figura 64 Conexión exitosa a partir de nuevas tablas de flujo.

3.4.2 USO DE LISTAS DE CONTROL DE ACCESO MEDIANTE CONTROLADOR FLOODLIGHT.

Para comenzar este ambiente de práctica, primero se procede a ejecutar un escenario SDN cualquiera, en este caso el escenario 2, el cual se inicia mediante el comando `sudo mn -custom Escenario2Testbed.py -topo=mytopo -controller=remote,ip10.0.2.15,port=6653 -link=t`. como indica la Figura 17.

Para la creación de una lista de control de acceso, que permita el tráfico entre dos dispositivos a partir de su IP, se utiliza el comando `curl -X POST -d '{"src-ip":"192.168.10.1/32","dst-ip":"192.168.10.4/32","action":"allow"}' http://localhost:8080/wm/acl/rules/json`. *Curl*, no es más que un comando que permite enviar solicitudes HTTP, además, el campo `-X POST -d` es el que permite enviar los datos especificados de la solicitud al servidor (Curl, 2022), el cual en este caso es el controlador. Adicionalmente, se especifica la dirección IP origen(`src-ip`), la dirección IP destino(`dst-ip`), y la acción la cual permite el flujo de tráfico entre estos dos dispositivos (`action:allow`). Este comando debe ser ingresado en el directorio donde se encuentra el controlador Floodlight. En este caso, la ACL configurada habilita el tráfico entre el Host1 y Host4 tal como se indica la Figura 65.

```
floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-ip":"192.168.10.1/32",
,"dst-ip":"192.168.10.4/32","action":"allow"}' http://localhost:8080/wm/acl/rules/ison
{"status" : "Success! New rule added."}floodlight@floodlight:~/floodlight$
```

Figura 65 ACL que permite el tráfico entre 2 dispositivos

De igual manera, se puede denegar el tráfico entre dos dispositivos a partir de su IP, en este caso, se utiliza el comando `curl -X POST -d '{"src-ip":"192.168.10.1/32", "dst-ip":"192.168.10.4/32", "action":"deny"}'` `http://localhost:8080/wm/acl/rules/json`, donde la acción necesaria para este caso está configurada como `action:deny`. En este ejemplo primero se procede a verificar la conectividad de todos los hosts de la red SDN como se observa en la Figura 66.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Figura 66 Verificación de conexión

Una vez hecha la verificación de las conexiones de red, se procede a ingresar la ACL de denegación como muestra la Figura 67, y por consecuencia, como indica la Figura 68, esta regla ha denegado la conectividad entre los Host 1 y 4.

```
floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-ip":"192.168.10.1/32",
,"dst-ip":"192.168.10.4/32","action":"deny"}' http://localhost:8080/wm/acl/rules/ison
{"status" : "Success! New rule added."}floodlight@floodlight:~/floodlight$
```

Figura 67 Ingreso de regla ACL

```

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

```

Figura 68 Verificación de funcionamiento de regla

Además, se puede ingresar más de una regla como se muestra en la Figura 69. Aquí, esta regla deniega el tráfico entre Host 2 ($IP=192.168.10.2$) y Host 3 ($IP=192.168.10.3$), lo cual se constata al momento de verificar la conectividad de toda la red mediante el comando *pingall*, lo que hace notar como la conexión entre estos 2 hosts se ha perdido, tal y como se observa en la Figura 70.

```

floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-ip":"192.168.10.2/32",
,"dst-ip":"192.168.10.3/32","action":"deny"}' http://localhost:8080/wm/acl/rules
/json
{"status" : "Success! New rule added."}
floodlight@floodlight:~/floodlight$

```

Figura 69 ACL para denegar el tráfico entre Host 2 y Host 3

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X
h2 -> h1 X h4
h3 -> h1 X h4
h4 -> X h2 h3
*** Results: 33% dropped (8/12 received)

```

Figura 70 Verificación de ACL

Las ACL's agregadas también pueden ser enlistadas mediante el comando *curl http://localhost:8080/wm/acl/rules/json* | *python -m json.tool* como indica la Figura 71. Esta información detalla qué tipo de acción cada regla está realizando, su id respectiva y la dirección IP de los dispositivos origen/destino.

```

    "action": "DENY",
    "id": 1,
    "nw_dst": "192.168.10.4/32",
    "nw_dst_maskbits": 32,
    "nw_dst_prefix": -1062729212,
    "nw_proto": 0,
    "nw_src": "192.168.10.1/32",
    "nw_src_maskbits": 32,
    "nw_src_prefix": -1062729215,
    "tp_dst": 0
  },
  {
    "action": "DENY",
    "id": 2,
    "nw_dst": "192.168.10.3/32",
    "nw_dst_maskbits": 32,
    "nw_dst_prefix": -1062729213,
    "nw_proto": 0,
    "nw_src": "192.168.10.2/32",
    "nw_src_maskbits": 32,
    "nw_src_prefix": -1062729214,
    "tp_dst": 0
  }
}

```

Figura 71 ACL's agregadas

También es posible eliminar ACL's según su id mediante el comando `curl -X DELETE -d '{"ruleid":"ID"}' http://localhost:8080/wm/acl/rules/json`, como indica la Figura 72. Este proceso se lo verifica revisando las ACL's activas tal y como muestra la Figura 73, dónde se puede apreciar que únicamente se encuentra la regla número 2.

```

floodlight@floodlight:~/floodlight$ curl -X DELETE -d '{"ruleid":"1"}' http://localhost:8080/wm/acl/rules/json
{"status": "Success! Rule deleted"}floodlight@floodlight:~/floodlight$

```

Figura 72 Eliminación de ACL

```
floodlight@floodlight:~/floodlight$ curl http://localhost:8080/wm/acl/rules/json
| python -mjson.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100    202      0   202    0     0    8599      0  --:--:--  --:--:--  --:--:--   9619
[
  {
    "action": "DENY",
    "id": 2,
    "nw_dst": "192.168.10.3/32",
    "nw_dst_maskbits": 32,
    "nw_dst_prefix": -1062729213,
    "nw_proto": 0,
    "nw_src": "192.168.10.2/32",
    "nw_src_maskbits": 32,
    "nw_src_prefix": -1062729214,
    "tp_dst": 0
  }
]
```

Figura 73 Verificación de ACL's activas

Finalmente, también es posible eliminar absolutamente todas las ACL's configuradas a través del comando `curl http://localhost:8080/wm/acl/clear/json` tal y como indica la Figura 74.

```
floodlight@floodlight:~/floodlight$ curl http://localhost:8080/wm/acl/clear/json
floodlight@floodlight:~/floodlight$ curl http://localhost:8080/wm/acl/rules/json
| python -mjson.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100     2      0     2    0     0    109      0  --:--:--  --:--:--  --:--:--   133
[]
floodlight@floodlight:~/floodlight$
```

Figura 74 Eliminación de todas las reglas ACL

3.4.3 EJECUCIÓN DE FIREWALL Y REGLAS DE TRÁFICO MEDIANTE CONTROLADOR FLOODLIGHT.

El controlador Floodlight, en uno de sus módulos de instalación, posee la capacidad de levantar un Firewall para la red que se ejecute a través de este. Para este ambiente de práctica, se utiliza el mismo escenario usado en la práctica anterior como indica la Figura 17.

El estado del firewall se puede constatar a partir del comando `curl http://localhost:8080/wm/firewall/module/status/json` como indica la Figura 75. Ya que este se encuentra inactivo, para activarlo se utiliza el comando `curl http://localhost:8080/wm/firewall/module/enable/json -X PUT -d ''` así como se muestra en la Figura 76.

```
floodlight@floodlight:~/floodlight$ curl http://localhost:8080/wm/firewall/module/status/json
{"result" : "firewall disabled"}floodlight@floodlight:~/floodlight$
```

Figura 75 Estado de Firewall

```
floodlight@floodlight:~/floodlight$ curl http://localhost:8080/wm/firewall/module/enable/json -X PUT -d ''
{"status" : "success", "details" : "firewall running"}floodlight@floodlight:~/floodlight$
```

Figura 76 Firewall activado

Una de las reglas configurables del firewall se enfoca en permitir la transmisión de datos sólo a través de cierto conmutador de la red SDN. Para este caso, primero se verifica la conexión de toda la red, la cual debido a la activación del firewall se encuentra totalmente caída, tal y como se ve en la Figura 77. De forma adicional, se hace una consulta de la id del conmutador a partir de la interfaz gráfica Floodlight como indica la Figura 78.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
```

Figura 77 Verificación de conectividad

The screenshot shows the Floodlight web interface at the URL 10.0.2.15:8080/switches. The page title is 'Switches (3)'. Below the title is a table with the following columns: DPID, IP Address, Vendor, Packets, Bytes, Flows, and Connected Since. The first row of the table has its DPID value, '00:00:00:00:00:00:01', highlighted with a red rectangular box.

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:00:00:00:00:01	/10.0.2.15:58695	Nicira, Inc.	1216	197969	5	3/17/2022, 12:01:48 AM
00:00:00:00:00:00:02	/10.0.2.15:58697	Nicira, Inc.	1178	195351	7	3/17/2022, 12:01:48 AM
00:00:00:00:00:00:03	/10.0.2.15:58696	Nicira, Inc.	1184	194744	5	3/17/2022, 12:01:48 AM

Figura 78 Información de conmutadores a partir de la interfaz Floodlight

Una vez obtenida la id del conmutador, y como indica la Figura 79, se utiliza el comando `curl -X POST -d '{"switchid": "00:00:00:00:00:00:01"}'` `http://localhost:8080/wm/firewall/rules/json`, el cual especifica mediante el campo `switchid` qué conmutador estamos seleccionando para la regla. En este ejemplo, sólo se ha permitido la transmisión de paquetes que viajen a través del conmutador s1 con id 00:00:00:00:00:00:01, por lo que en la Figura 80 se muestra como únicamente hay conexión entre Host 1 y Host 2.

```
floodlight@floodlight:~/floodlight$ curl -X POST -d '{"switchid": "00:00:00:00:00:00:01"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "1830900057"}floodlight@floodlight:~/floodlight$
floodlight@floodlight:~/floodlight$
```

Figura 79 Regla para conexión a través de un solo conmutador

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X
h2 -> h1 X X
h3 -> X X X
h4 -> X X X
*** Results: 83% dropped (2/12 received)
```

Figura 80 Verificación de regla

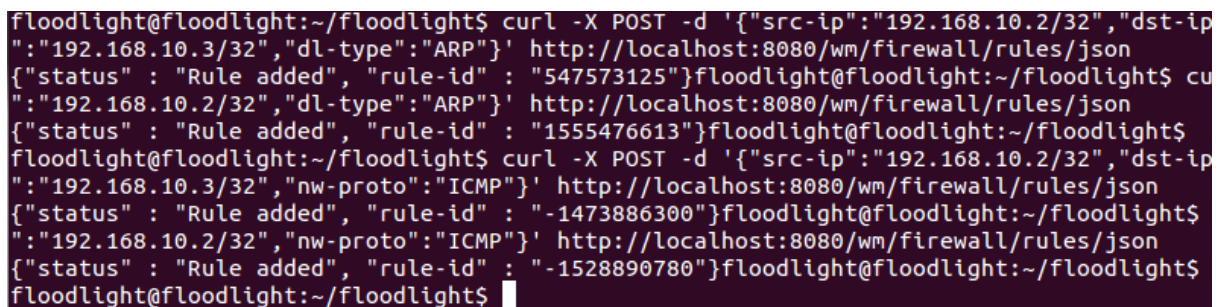
También es posible crear reglas que permitan sólo la comunicación ICMP entre dos dispositivos a partir de su IP. En esta regla los campos necesarios son la dirección IP origen(`src-ip`), la dirección IP destino(`dst-ip`), el protocolo ARP(`dl-type:ARP`) y el protocolo ICMP(`nw_proto:ICMP`). los comandos se aplican tal y como muestra la Figura 81 y son los siguientes:

```
curl -X POST -d '{"src-ip":"192.168.10.1/32","dst-ip":"192.168.10.4/32","dl-type":"ARP"}' http://localhost:8080/wm/firewall/rules/json
```

```
curl -X POST -d '{"src-ip":"192.168.10.4/32","dst-ip":"192.168.10.1/32","dl-type":"ARP"}' http://localhost:8080/wm/firewall/rules/json
```

```
curl -X POST -d '{"src-ip":"192.168.10.1/32","dst-ip":"192.168.10.4/32","nw-proto":"ICMP"}' http://localhost:8080/wm/firewall/rules/json
```

```
curl -X POST -d '{"src-ip":"192.168.10.4/32","dst-ip":"192.168.10.1/32","nw-proto":"ICMP"}' http://localhost:8080/wm/firewall/rules/json
```



```
floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-ip":"192.168.10.2/32","dst-ip":"192.168.10.3/32","dl-type":"ARP"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "547573125"}floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-ip":"192.168.10.2/32","dl-type":"ARP"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "1555476613"}floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-ip":"192.168.10.2/32","dst-ip":"192.168.10.3/32","nw-proto":"ICMP"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "-1473886300"}floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-ip":"192.168.10.2/32","nw-proto":"ICMP"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "-1528890780"}floodlight@floodlight:~/floodlight$
```

Figura 81 Regla que permite una conexión ICMP a partir de direcciones IP.

Es posible verificar la regla aplicada con anterioridad a partir de un ping entre cada host como muestra la Figura 82. Adicionalmente, se pueden seguir agregando más reglas que permitan a más hosts conectarse, en este caso, se agrega una regla que permite la comunicación ICMP para el Host 1 y Host 4 como indica la Figura 83. Finalmente,

como en el ejemplo anterior, esta regla se puede verificar mediante un ping entre todos los terminales como se ve en la Figura 84.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X h3 X
h3 -> X h2 X
h4 -> X X X
*** Results: 83% dropped (2/12 received)
```

Figura 82 Verificación de la regla aplicada

```
floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-ip":"192.168.10.1/32","dst-ip":"192.168.10.4/32","dl-type":"ARP"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "-230677767"}floodlight@floodlight:~/floodlight$ c
{"src-ip":"192.168.10.1/32","dl-type":"ARP"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "388895225"}floodlight@floodlight:~/floodlight$
floodlight@floodlight:~/floodlight$
floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-ip":"192.168.10.1/32","dst-ip":"192.168.10.4/32","nw-proto":"ICMP"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "839266008"}floodlight@floodlight:~/floodlight$ cu
{"src-ip":"192.168.10.1/32","nw-proto":"ICMP"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "814908696"}floodlight@floodlight:~/floodlight$
```

Figura 83 Nueva regla para conexión entre Host 1 y 4

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X h4
h2 -> X h3 X
h3 -> X h2 X
h4 -> h1 X X
*** Results: 66% dropped (4/12 received)
```

Figura 84 Verificación de nueva regla

También es posible crear reglas que permitan el tráfico entre dos dispositivos mediante su dirección MAC. Para esto, primero se busca la información de la MAC de cada Host en la interfaz Floodlight como indica la Figura 85.

The screenshot shows the Floodlight web interface with the 'Hosts' tab selected. A table titled 'Hosts (4)' displays the following data:

MAC Address	IP Address	Switch Port	Last Seen
a2:61:84:f6:ba:85	192.168.10.2	00:00:00:00:00:00:02-2	3/17/2022, 12:50:11 AM
fe:26:d6:5e:04:ad	192.168.10.1	00:00:00:00:00:00:02-1	3/17/2022, 12:50:37 AM
3a:0c:89:d8:53:4d	192.168.10.3	00:00:00:00:00:00:03-1	3/17/2022, 12:50:34 AM
66:b2:b9:b6:c4:9f	0.0.0.0,192.168.10.4	00:00:00:00:00:00:03-2	3/17/2022, 12:51:00 AM

Figura 85 Direcciones MAC de hosts

Ya con la información de las direcciones MAC necesarias, se procede a crear una regla que permita la comunicación entre el Host 1 con dirección MAC fe:26:d6:5e:04:ad y el Host 4 con dirección 66:b2:b9:b6:c4:9f como indica la Figura 86 mediante el comando a continuación:

```
curl -X POST -d '{"src-mac": "fe:26:d6:5e:04:ad", "dst-mac": "66:b2:b9:b6:c4:9f"}'
```

```
http://localhost:8080/wm/firewall/rules/json
```

```
curl -X POST -d '{"src-mac": "66:b2:b9:b6:c4:9f", "dst-mac": "fe:26:d6:5e:04:ad"}'
```

```
http://localhost:8080/wm/firewall/rules/json
```

```
floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-mac": "fe:26:d6:5e:04:ad", "dst-mac": "66:b2:b9:b6:c4:9f"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "55589278"}floodlight@floodlight:~/floodlight$
floodlight@floodlight:~/floodlight$ curl -X POST -d '{"src-mac": "66:b2:b9:b6:c4:9f", "dst-mac": "fe:26:d6:5e:04:ad"}' http://localhost:8080/wm/firewall/rules/json
{"status": "Rule added", "rule-id": "497524642"}floodlight@floodlight:~/floodlight$
floodlight@floodlight:~/floodlight$
```

Figura 86 Regla de comunicación a través de direcciones MAC

Finalmente, se verifica la aplicación correcta de esta regla mediante un ping entre cada host como indica la Figura 87. En esta se observa cómo sólo los 2 terminales correspondientes tienen comunicación.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X h4
h2 -> X X X
h3 -> X X X
h4 -> h1 X X
*** Results: 83% dropped (2/12 received)
```

Figura 87 Verificación de regla

4. CAPÍTULO IV: OBTENCIÓN DE MÉTRICAS Y ANÁLISIS DE RESULTADOS

Este apartado del trabajo se enfoca en el análisis de funcionamiento del Testbed SDN a partir de la evaluación de las métricas latencia, jitter y throughput de cada escenario de prueba realizado en la sección 3.2. Además, se realiza un estudio de los resultados obtenidos en cada uno de los ambientes de práctica ejecutados en la sección 3.4.

4.1 EVALUACIÓN DE MÉTRICAS EN LA RED SDN

El rendimiento de una red se ve influenciada por algunas métricas, donde tres de las más importantes según (Tanenbaum, 2012) son la latencia, jitter y throughput. A partir de estas métricas es posible realizar un análisis del rendimiento de una red y evaluar su funcionamiento, por ende, en este trabajo de grado se han tomado estos tres parámetros para el estudio del comportamiento de las redes SDN en los escenarios anteriormente propuestos para los controladores Floodlight y Pox.

4.1.1 LATENCIA

Para encontrar la latencia en cada escenario, sólo es necesario observar los resultados de las pruebas de conectividad efectuadas anteriormente en la sección 3.2, donde, la consola mininet proporciona la información de latencia mínima, media y máxima (*rtt min/avg/max*) después de cada ping realizado. Esta información ha sido compilada en las Tablas 5 y 6 para cada escenario y comparada tal como indican las Figuras 88 y 89.

Tabla 5 Tiempos de latencia mínimos, medios y máximos del escenario 1

Controlador	Host Origen	Host Destino	Latencia mínima (ms)	Latencia media (ms)	Latencia máxima (ms)
POX	Host 1	Host 2	0.037	3.213	15.793
		Host 3	0.041	12.148	60.189
		Host 4	0.035	13.873	68.810
Floodlight	Host 1	Host 2	0.027	0.920	3.499
		Host 3	0.033	1.904	7.400
		Host 4	0.033	1.917	7.443

Tabla 6 Tiempos de latencia mínimos, medios y máximos del escenario 2

Controlador	Host Origen	Host Destino	Latencia mínima (ms)	Latencia media (ms)	Latencia máxima (ms)
POX	Host 1	Host 2	0.046	14.510	57.657
		Host 3	0.051	13.616	54.044
		Host 4	0.053	15.796	62.147
Floodlight	Host 1	Host 2	0.032	1.244	4.744
		Host 3	0.043	3.721	14.599
		Host 4	0.040	3.774	14.815

El primer detalle en sobresalir de los tiempos de latencia media comparados de las pruebas realizadas se refleja en las Figuras 88 y 89, en estas se muestra cómo existe una menor latencia cuando hay menos conmutadores en la ruta de transmisión de paquetes en cada conexión. Este resultado se debe a que el flujo de paquetes entre Host 1 y Host 2 sólo lo ejecuta un conmutador, a diferencia de una conexión entre Host 1 y Host 3 o Host 1 y Host 4 donde existen dos y hasta tres conmutadores efectuando este proceso, donde por cada conexión entrante, estos dispositivos deben realizar un proceso de coincidencia

(match) y actualización de flujo de paquetes en sus tablas de flujo, lo cual explica por qué la diferencia de latencia entre pruebas.

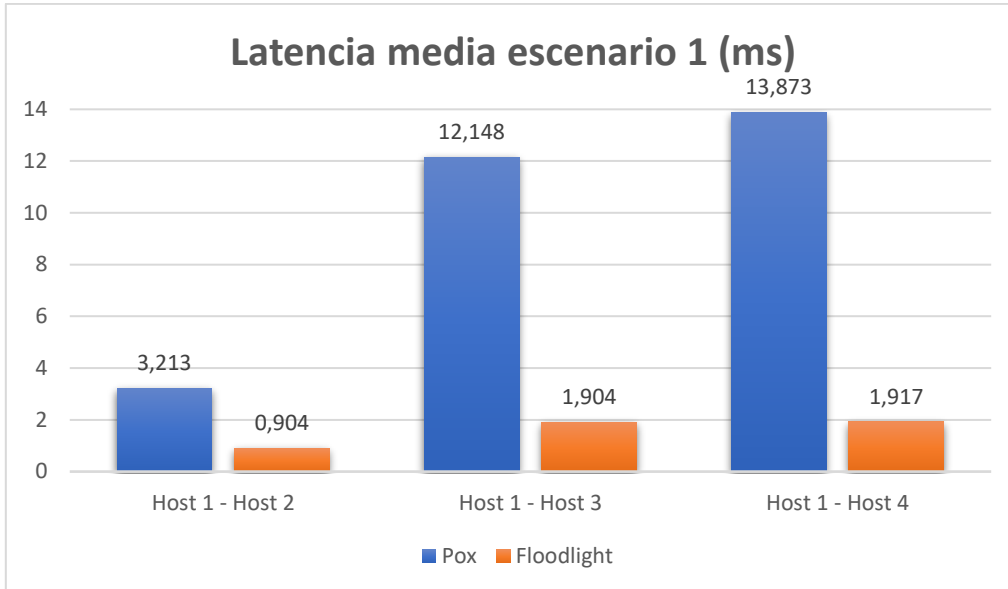


Figura 88 Comparación tiempos de latencia media entre controladores escenario 1

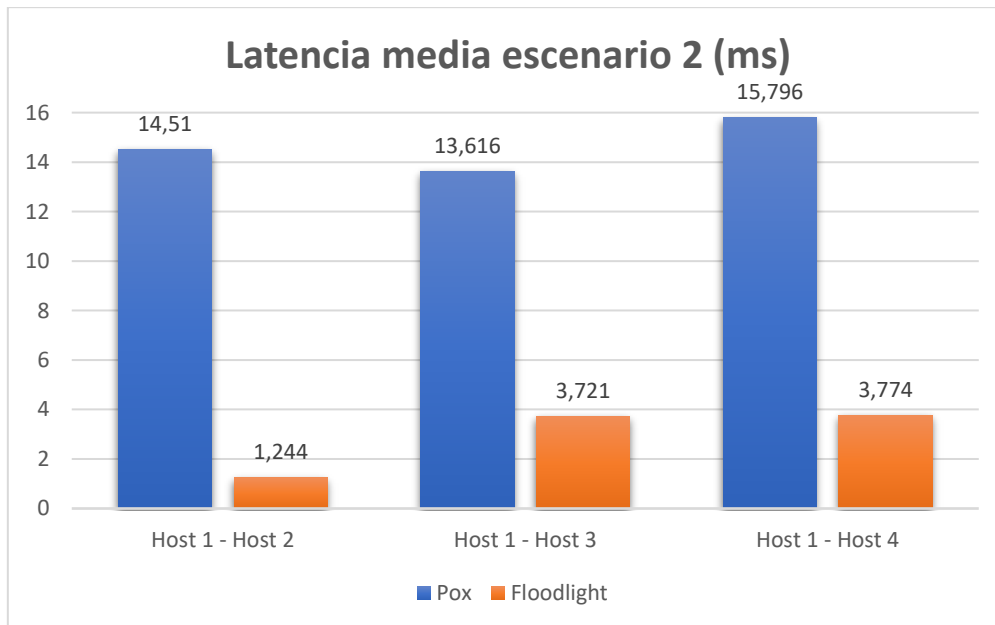


Figura 89 Comparación tiempos de latencia media entre controladores escenario 1

Otra consideración es la diferencia de latencia máxima entre los controladores Pox y Floodlight. En el escenario 1, donde todos los dispositivos se encuentran en la misma red, el controlador Floodlight, como muestra la Figura 90, presenta menor latencia que el controlador Pox, y de la misma manera, como indica la Figura 91 para el escenario 2, la diferencia en el tiempo de latencia entre controladores se mantiene.

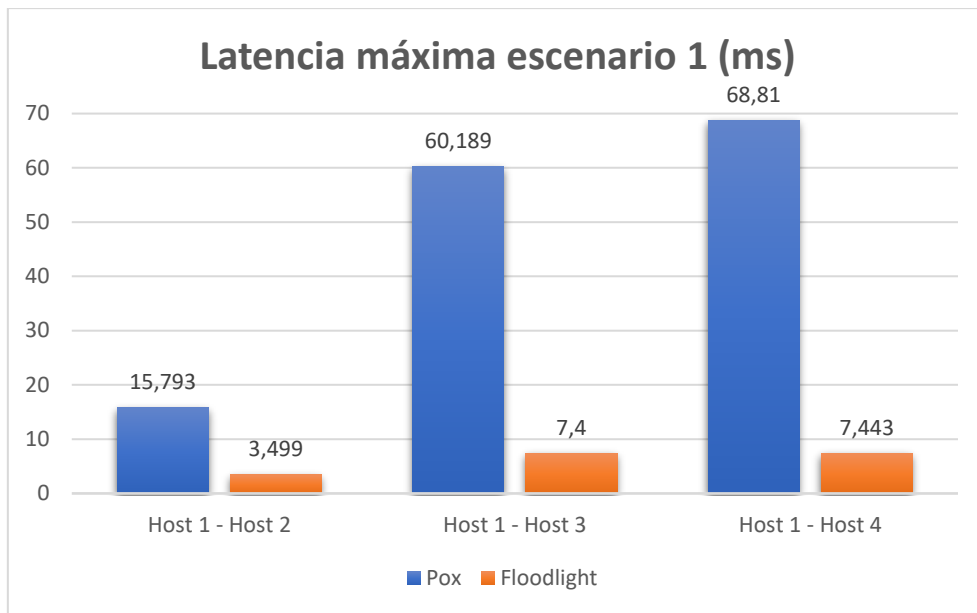


Figura 90 Comparación de tiempos de latencia máxima entre controladores escenario 1

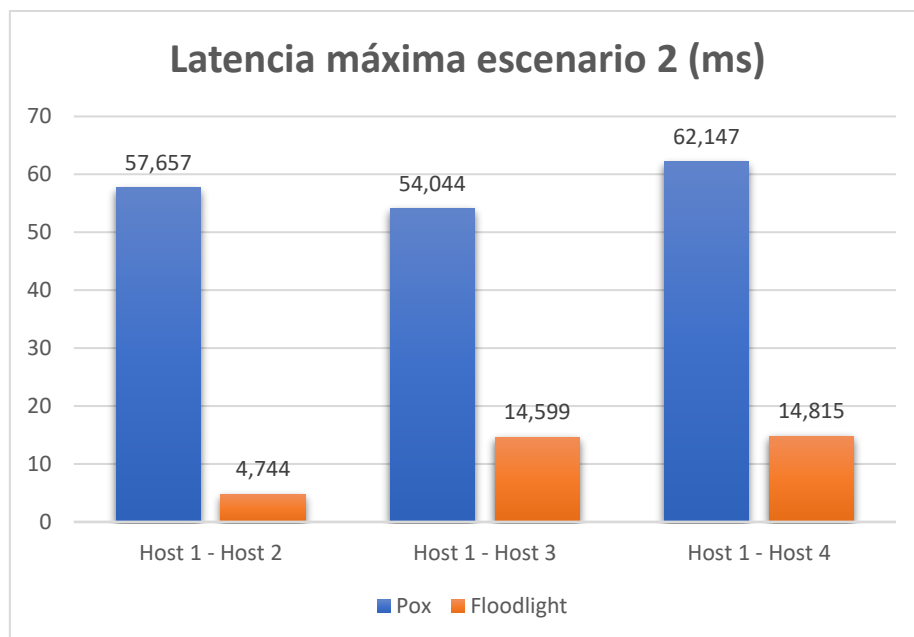


Figura 91 Comparación de tiempos de latencia máxima entre controladores escenario 2

Estas diferencias entre controladores principalmente ocurren debido a que cada controlador presenta un módulo de acción diferente, donde el lenguaje de programación usado en cada uno juega un papel importante. En el caso de Floodlight, el uso adicional de java permite que este realice actividades de ejecución y depuración mucho más rápido que el controlador Pox, donde el único lenguaje usado es python, por ende, la diferencia en los resultados en los tiempos de latencia es tan notorio entre controladores.

4.1.2 JITTER

El jitter, a diferencia de la latencia, se lo debe calcular debido a que su información no es proporcionada por la consola después de realizar las pruebas de conectividad. El cálculo de esta métrica, como indica (LiveAction, 2022), se muestra a continuación en la Ec. 1:

Ecuación 1 Ecuación de cálculo de jitter

$$jitter = \frac{\sum_i^n |D_i|}{n}$$

Donde $\sum_i^n |D_i|$ se refiere a la suma de la diferencia del tiempo de latencia medio entre cada secuencia de paquetes, y n al número de secuencias existentes.

A continuación, tomando la Ec.1 como referencia, se procede a calcular el valor de jitter entre cada conexión realizada entre hosts para cada controlador y escenario correspondiente a partir de los valores de latencia de las Tablas 5 y 6.

Ecuación 2 Cálculo de jitter en el escenario 1 controlador POX ping Host 1-Host 2

$$jitter = \frac{|(15.7 - 0.163) + (0.163 - 0.038) + (0.038 - 0.038)|}{3} = \mathbf{3.915\ ms}$$

En este sentido, a manera de ejemplo se presenta los cálculos realizados para obtener el valor del jitter entre el host 1 y host 2 a partir de los valores de transmisión de

paquetes mostrados en las Tablas 7 y 8. La compilación del valor de jitter entre los diferentes hosts y escenarios se indican en las Tablas 9 y 10, donde a partir de estos se procederá a realizar una comparación entre controladores y escenarios como lo muestran las Figuras 92 y 93 para su posterior análisis.

Tabla 7 Tiempo de envío de paquetes en cada secuencia en el escenario 1

Controlador	Host Origen	Host Destino	Tiempo de transmisión (ms)
POX			15.7
			0.163
			0.038
			0.038
	Host 1	Host 2	60.1
			0.239
			0.042
			0.041
	Host 1	Host 3	68.8
			0.274
			0.038
			0.208
Floodlight			3.49
			0.126
			0.030
			0.027
	Host 1	Host 2	7.40
			0.149
			0.034
			0.033
	Host 1	Host 3	7.44
			0.159
			0.035
			0.033
Host 1	Host 4	7.44	
		0.159	
		0.035	
		0.033	

Tabla 8 Tiempo de envío de paquetes en cada secuencia en el escenario 2

Controlador	Host Origen	Host Destino	Tiempo de transmisión (ms)	
POX			57.6	
			0.172	
			0.046	
			0.167	
				54.0
				0.316
				0.054
				0.051
				62.1
				0.319
				0.053
				0.665
Floodlight			4.74	
			0.167	
			0.035	
			0.032	
				14.5
				0.201
				0.043
				0.043
				14.8
				0.203
				0.040
				0.041

A simple vista, es fácil percatarse que el tiempo transcurrido del primer paquete siempre es bastante alto a diferencia de los siguientes, esto debido a que el match de coincidencia para el primer paquete entrante siempre se efectúa, para posteriormente sólo realizar el reenvío de los demás paquetes por parte del conmutador. Además, en este punto

cabe recalcar que la diferencia en tiempos de transmisión entre controladores no es muy notoria a partir del segundo paquete, en comparación con el primero, donde si existe una diferencia bastante notoria entre tiempo de transmisión en cada uno de los escenarios SDN, llegando a valores máximos de 68.8 ms en el caso del controlador Pox, y a 14.5 ms con Floodlight.

Tabla 9 Jitter del escenario 1 SDN

Controlador	Host Origen	Host Destino	Jitter (ms)
POX	Host 1	Host 2	3.915
		Host 3	15.017
		Host 4	17.276
Floodlight	Host 1	Host 2	1.154
		Host 3	2.455
		Host 4	2.469

Tabla 10 Jitter del escenario 2 SDN

Controlador	Host Origen	Host Destino	Jitter (ms)
POX	Host 1	Host 2	19.225
		Host 3	17.983
		Host 4	20.881
Floodlight	Host 1	Host 2	1.559
		Host 3	4.819
		Host 4	4.920

Como se presenta en las tablas 9 y 10, los valores máximos de jitter en las transmisiones de paquetes del controlador Pox difieren bastante en comparación con Floodlight. En el primero existen valores de hasta 20 ms, a diferencia del segundo donde

incluso nunca se llegó a 5 ms en ninguna de las pruebas realizadas. Esto sólo continúa el patrón en la diferencia de los controladores y sus resultados en cada una de las métricas obtenidas, dónde Floodlight siempre es mucho más rápido y menos propenso a retardos que el controlador Pox.

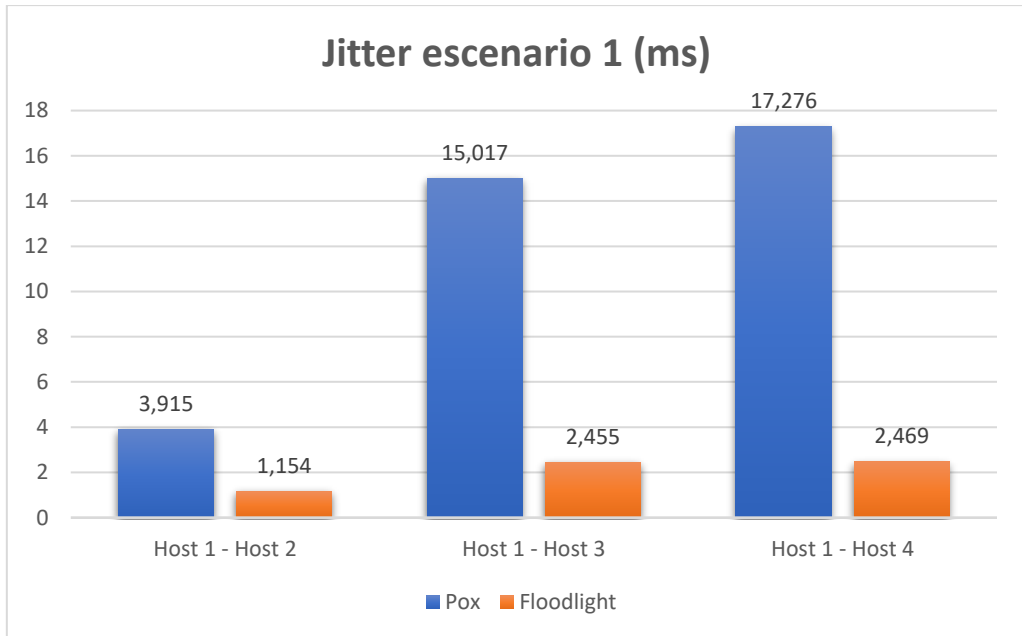


Figura 92 Comparación de jitter entre controladores en el escenario 1 SDN

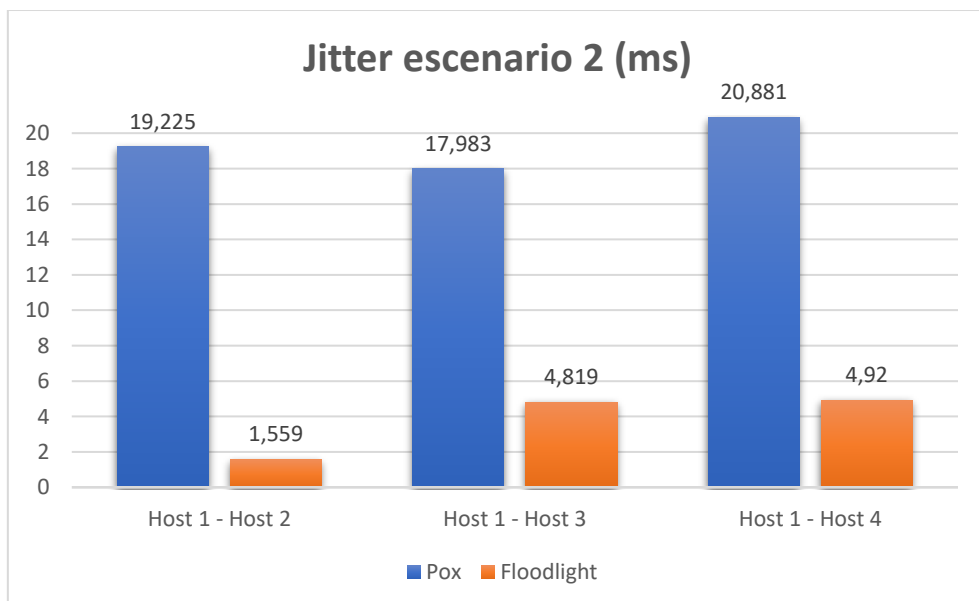


Figura 93 Comparación de jitter entre controladores en el escenario 2 SDN

Como indican las Figuras 92 y 93, de la misma forma que en la latencia, el jitter presenta valores mucho menores cuando se ejecuta la red mediante el controlador Floodlight a diferencia de Pox, por lo que se evidencia cómo el tipo de programación base para cada controlador influye para el funcionamiento de una red SDN, siendo en este caso java mucho más rápido que python. Finalmente, cabe recalcar que ningún valor de jitter excede los 30 ms, y que según (LiveAction, 2022), este se considera aceptable e incluso ideal para la mayoría de las redes, incluso para conexiones de audio y video.

4.1.3 THROUGHPUT

La tasa de transferencia efectiva, o Throughput, se refiere a la cantidad de datos que pueden transferirse de un dispositivo a otro en un cierto período de tiempo (IETF, 1999). Un test para medir esta métrica determina la velocidad a la que pueden viajar los datos manteniendo cero paquetes descartados por parte del dispositivo puesto a prueba. El cálculo de esta métrica, como indica (Contreras Higuera & Granados Acuña , 2012), se muestra a continuación en la Ec. 3, donde “*Datos*” se refiere a la cantidad de información transferida, y T_t al tiempo transcurrido en toda la transmisión :

Ecuación 3 Ecuación de cálculo del throughput

$$\mathbf{Throughput} = \frac{\mathbf{Datos}}{T_t}$$

Para encontrar el valor de esta métrica en la red SDN, se utilizará iPerf, una herramienta que permite realizar pruebas de funcionamiento en redes informáticas a partir de la creación de flujos de datos TCP y UDP (iPerf, 2022). En este caso, ya que se hará una prueba a partir del host 1 hacia los demás host de la red SDN, se abrirá iPerf desde la consola Mininet para cada host a partir del comando `xterm h1 h2 h3 h4` como se indica en la Figura 94.

```
mininet> xterm h1 h2 h3 h4
```

Figura 94 Ejecución de iPerf en los host de la red SDN

A continuación, se iniciarán los nodos h2, h3 y h4 para que actúen como servidor TCP (-s) en el puerto 5566 (-p) de forma que se monitoreen los resultados cada segundo (-i), así como se indica en la Figura 95 a manera de ejemplo para el host 2.

```
"Node: h2"
root@floodlight:~/floodlight# iperf -s -p 5566 -i 1
-----
Server listening on TCP port 5566
TCP window size: 85,3 KByte (default)
-----
```

Figura 95 Iniciando nodo h2 como servidor TCP

Ahora, se inicia el cliente TCP desde el nodo h1 y se selecciona la duración de la transmisión (-t), que en este caso será 15 segundos para todos los nodos. Además, se debe indicar la dirección IPv4 del servidor TCP al cual se realizará la transmisión (-c), en este caso, de los nodos h2, h3 y h4 como se presentan en las figuras 96 y 97 para el escenario 1 y 2 respectivamente.

```
"Node: h1"
root@floodlight:~/floodlight# iperf -c 10.0.0,2 -p 5566 -t 15
Client connecting to 10.0.0,2, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 21] local 10.0.0,1 port 51491 connected with 10.0.0,2 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15.0 sec  64,2 GBytes 36,7 Gbits/sec

"Node: h1"
root@floodlight:~/floodlight# iperf -c 10.0,0,3 -p 5566 -t 15
Client connecting to 10.0,0,3, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 21] local 10.0.0,1 port 39277 connected with 10.0,0,3 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15,0 sec  60,4 GBytes 34,6 Gbits/sec

"Node: h1"
root@floodlight:~/floodlight# iperf -c 10.0.0,4 -p 5566 -t 15
Client connecting to 10.0.0,4, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 21] local 10.0.0,1 port 47513 connected with 10.0.0,4 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15,0 sec  57,6 GBytes 33,0 Gbits/sec

"Node: h1"
root@floodlight:~/pox-eel/pox# iperf -c 10.0,0,2 -p 5566 -t 15
Client connecting to 10.0,0,2, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 21] local 10.0.0,1 port 51498 connected with 10.0,0,2 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15,0 sec  55,8 GBytes 32,0 Gbits/sec

"Node: h1"
root@floodlight:~/pox-eel/pox# iperf -c 10.0,0,3 -p 5566 -t 15
Client connecting to 10.0,0,3, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 21] local 10.0.0,1 port 39284 connected with 10.0,0,3 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15,0 sec  42,5 GBytes 24,4 Gbits/sec

"Node: h1"
root@floodlight:~/pox-eel/pox# iperf -c 10.0.0,4 -p 5566 -t 15
Client connecting to 10.0.0,4, TCP port 5566
TCP window size: 85,3 KByte (default)
-----
[ 21] local 10.0.0,1 port 47520 connected with 10.0,0,4 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15,0 sec  50,6 GBytes 23,0 Gbits/sec
```

Figura 96 Ejecución de transmisión de cada nodo para el escenario 1 en los controladores SDN.

```

root@floodlight:~/floodlight# iperf -c 192.168.10.2 -p 5566 -t 15
Client connecting to 192.168.10.2, TCP port 5566
TCP window size: 85.3 KByte (default)
[ 21] local 192.168.10.1 port 52539 connected with 192.168.10.2 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15.0 sec  67.9 GBytes 38.9 Gbits/sec

root@floodlight:~/floodlight# iperf -c 192.168.10.3 -p 5566 -t 15
Client connecting to 192.168.10.3, TCP port 5566
TCP window size: 85.3 KByte (default)
[ 21] local 192.168.10.1 port 47523 connected with 192.168.10.3 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15.0 sec  57.8 GBytes 33.1 Gbits/sec

root@floodlight:~/floodlight# iperf -c 192.168.10.4 -p 5566 -t 15
Client connecting to 192.168.10.4, TCP port 5566
TCP window size: 85.3 KByte (default)
[ 21] local 192.168.10.1 port 49358 connected with 192.168.10.4 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15.0 sec  60.0 GBytes 34.4 Gbits/sec

root@floodlight:~/pox-eel/pox# iperf -c 192.168.10.2 -p 5566 -t 15
Client connecting to 192.168.10.2, TCP port 5566
TCP window size: 85.3 KByte (default)
[ 21] local 192.168.10.1 port 45167 connected with 192.168.10.2 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15.0 sec  60.0 GBytes 34.3 Gbits/sec

root@floodlight:~/pox-eel/pox# iperf -c 192.168.10.3 -p 5566 -t 15
Client connecting to 192.168.10.3, TCP port 5566
TCP window size: 85.3 KByte (default)
[ 21] local 192.168.10.1 port 56498 connected with 192.168.10.3 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15.0 sec  46.7 GBytes 26.7 Gbits/sec

root@floodlight:~/pox-eel/pox# iperf -c 192.168.10.4 -p 5566 -t 15
Client connecting to 192.168.10.4, TCP port 5566
TCP window size: 85.3 KByte (default)
[ 21] local 192.168.10.1 port 57579 connected with 192.168.10.4 port 5566
[ ID] Interval      Transfer    Bandwidth
[ 21] 0.0-15.0 sec  57.9 GBytes 33.2 Gbits/sec

```

Figura 97 Ejecución de transmisión de cada nodo para el escenario 2 en los controladores SDN

En este sentido, la tasa de transferencia efectiva se presenta en un intervalo de 15 segundos y muestra un promedio de la capacidad máxima de transferencia de datos hecha en la transmisión, así como su ancho de banda. Estos valores se compilan en las Tablas 11 y 12, además de ser representados tal y como se indican en las Figuras 98, 99, 100 y 101.

Tabla 11 Valores de throughput en los controlares SDN del escenario 1.

	Nodo Cliente	Nodo Servidor	Datos Transferidos (Gigabytes)	Ancho de Banda (Gigabits/segundo)
Floodlight		Host 2	64.2	36.7
	Host 1	Host 3	60.4	34.6
		Host 4	57.6	33.0
Pox		Host 2	55.8	32.0
	Host 1	Host 3	42.5	24.4
		Host 4	50.6	29.0

Tabla 12 Valores de throughput en los controlares SDN del escenario 2.

	Nodo Cliente	Nodo Servidor	Datos Transferidos (Gigabytes)	Ancho de Banda (Gigabits/segundo)
Floodlight	Host 1	Host 2	67.9	38.9
		Host 3	57.8	33.1
		Host 4	60.0	34.4
Pox	Host 1	Host 2	60.0	34.3
		Host 3	46.7	26.7
		Host 4	57.9	33.2

Como se presenta en las tablas 11 y 12, los valores de throughput no cambian mucho en cada escenario, aunque si se compara resultados entre controlador, y como se viene obteniendo en las anteriores métricas, Floodlight continúa presentando mejores resultados que Pox. En este caso, el primero llega a tener valores máximos de 67.9 Gb de datos transferidos con un ancho de banda de 38.9 Gbits/seg para el escenario dos, el cual es mucho mayor en comparación de Pox, donde en el mismo escenario de pruebas SDN, presentó un valor máximo de datos transferidos de 60 Gb con un ancho de bando de 34.4 Gbits/segundo, casi 8 Gb y 4 Gbits/segundo menos que los resultados presentados por parte del controlador Floodlight.

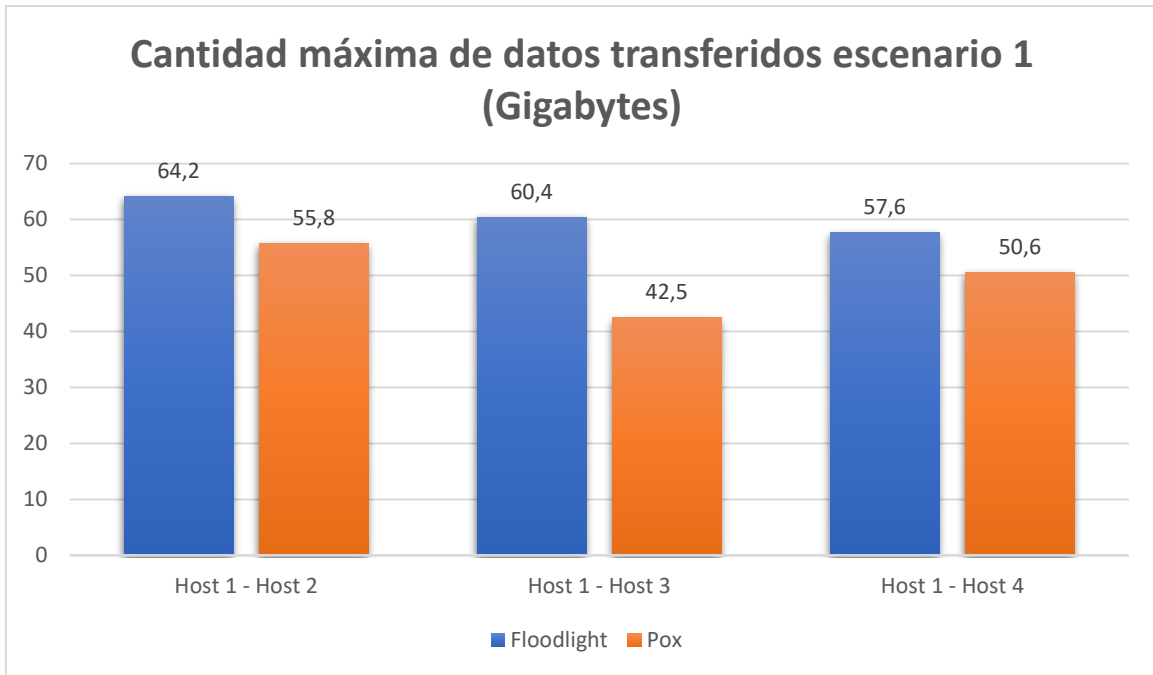


Figura 98 Cantidad máxima de datos transferidos entre nodos en el escenario 1

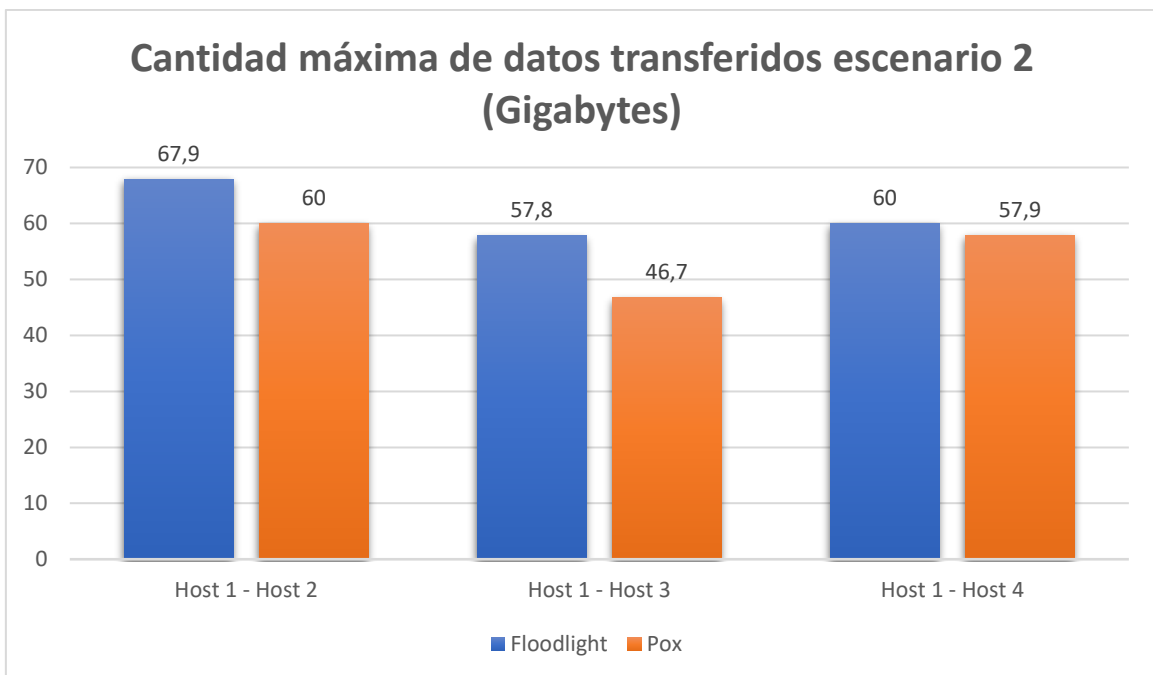


Figura 99 Cantidad máxima de datos transferidos entre nodos en el escenario 2

Con base a los resultados expuestos en las Figuras 98 y 99 para los dos escenarios SDN, se obtiene que el throughput promedio para el controlador Floodlight es 60.7 y 61.9 GB en el escenario 1 y 2 respectivamente, a diferencia de los valores un poco menores de

los obtenidos a partir del controlador Pox los cuales son 49.63 y 54.86 GB para cada escenario.

Tomando en cuenta los valores de cada controlador, es notorio como Floodlight posee una capacidad muy alta de enviar datos sin que ningún paquete se pierda en la transmisión a comparación de Pox, donde aunque a simple vista parezcan resultados no tan lejanos uno de otro, hay que recordar que estos valores se encuentran medidos en gigabytes, por lo que la diferencia de estos promedios se hace más evidente cuando se tiene en cuenta la cantidad de datos transmitidos en una transmisión que involucró 60 GB a una de 49 GB.

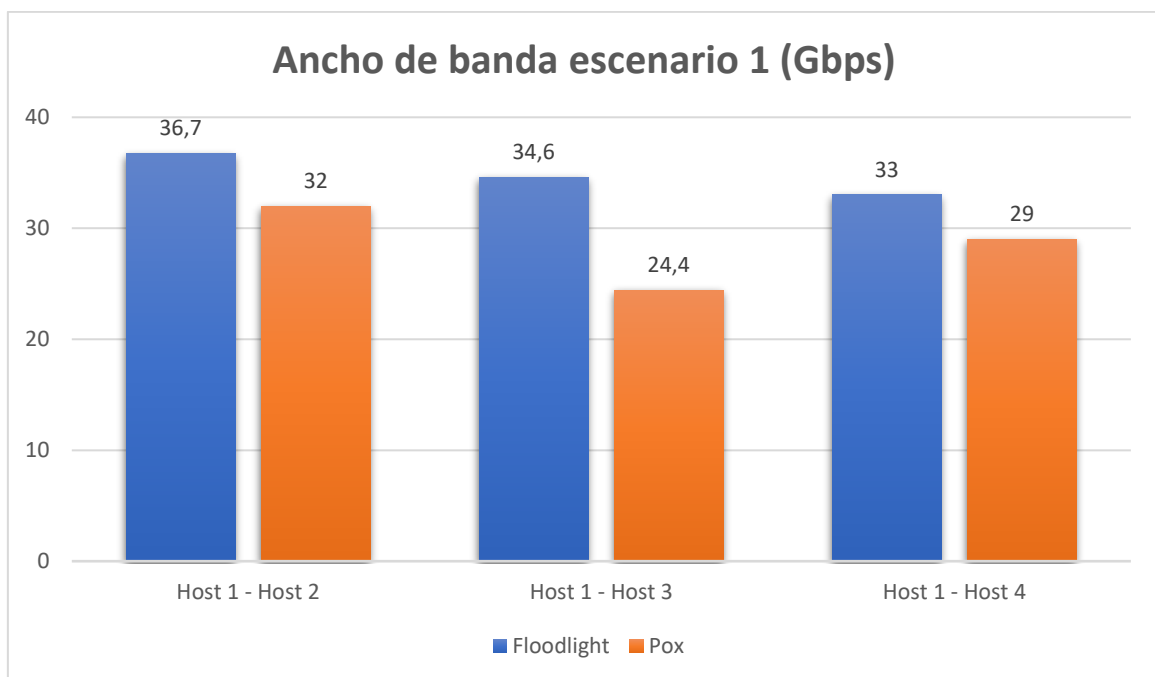


Figura 100 Ancho de banda entre nodos de la red para el escenario 1

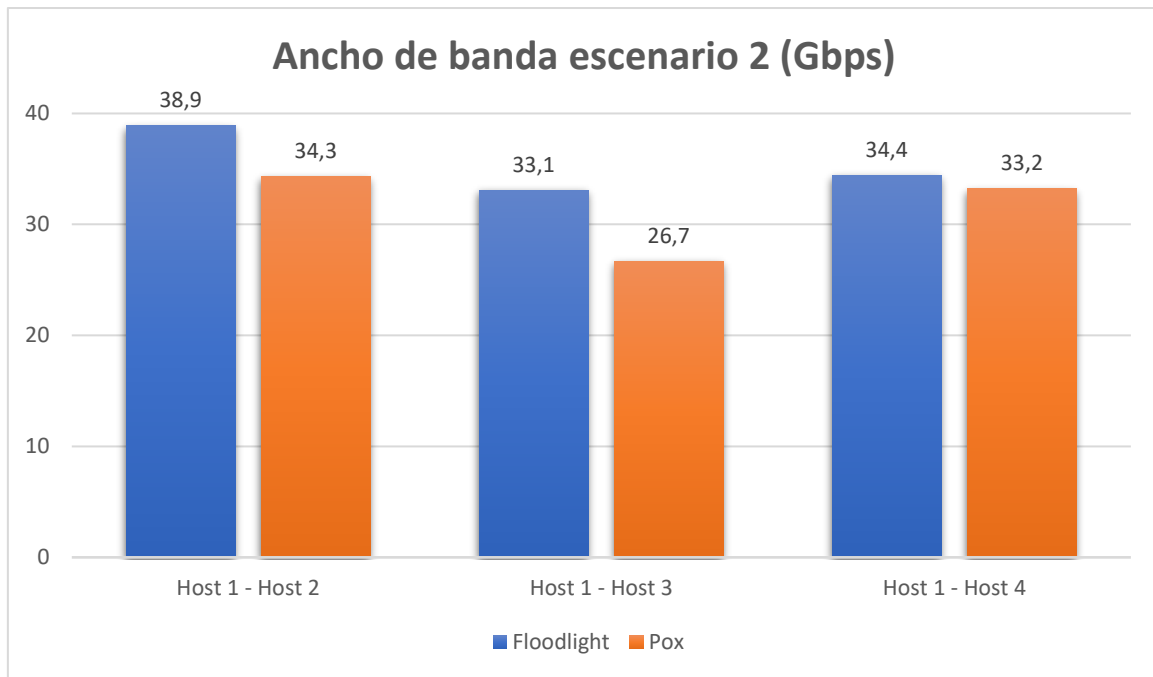


Figura 101 Ancho de banda entre nodos de la red para el escenario 2

A continuación, tomando la Ec.2 como referencia, se procede a calcular el valor de throughput de cada transmisión hecha en los controladores y escenarios correspondientes, a partir de los valores presentados en las Figuras 96 y 97.

Ecuación 4 Cálculo del jitter del escenario 1 controlador Floodlight transmisión Host 1-Host 2

$$throughput = \frac{64.2 \text{ Gigabytes}}{15 \text{ segundos}} \times \frac{8 \text{ bit}}{\text{bytes}} = 34.24 \text{ Gbps}$$

En este sentido, a manera de ejemplo se presenta los cálculos realizados para obtener el valor del throughput entre el host 1 y host 2. La compilación del valor de throughput entre los diferentes hosts y escenarios se indican en las Tablas 13 y 14, donde a partir de estos se procederá a realizar una comparación entre controladores y escenarios como lo muestran las Figuras 102 y 103 para su posterior análisis.

Tabla 13 Valores de throughput en los controlares SDN del escenario 1 para 15 segundos.

	Nodo Cliente	Nodo Servidor	Throughput (Gbps)
Floodlight	Host 1	Host 2	34.24
		Host 3	32.21
		Host 4	30.72
Pox	Host 1	Host 2	29.76
		Host 3	22.66
		Host 4	26.98

Tabla 14 Valores de throughput en los controlares SDN del escenario 2 para 15 segundos.

	Nodo Cliente	Nodo Servidor	Throughput (Gbps)
Floodlight	Host 1	Host 2	36.21
		Host 3	30.82
		Host 4	32.00
Pox	Host 1	Host 2	32.00
		Host 3	24.9
		Host 4	30.88

Debido a que las redes SDN emuladas para ambos escenarios no presentan errores de transmisión en su diseño o programación, los valores de throughput obtenidos llegan a alcanzar un valor de medición de hasta 36 Gbps. Esto no es una sorpresa ya que si se toma como referencia inicial los valores de datos transferidos y cómo estos llegan a cantidades que varían en gigabytes, el tener un valor de throughput tan alto para una medición de 15 seg es algo esperado y nada extraordinario. Ahora, para tener una mejor idea de cómo estos valores se comparan entre escenarios y controladores SDN, se ha compilado las tablas 13 y 14 en las figuras presentadas a continuación.

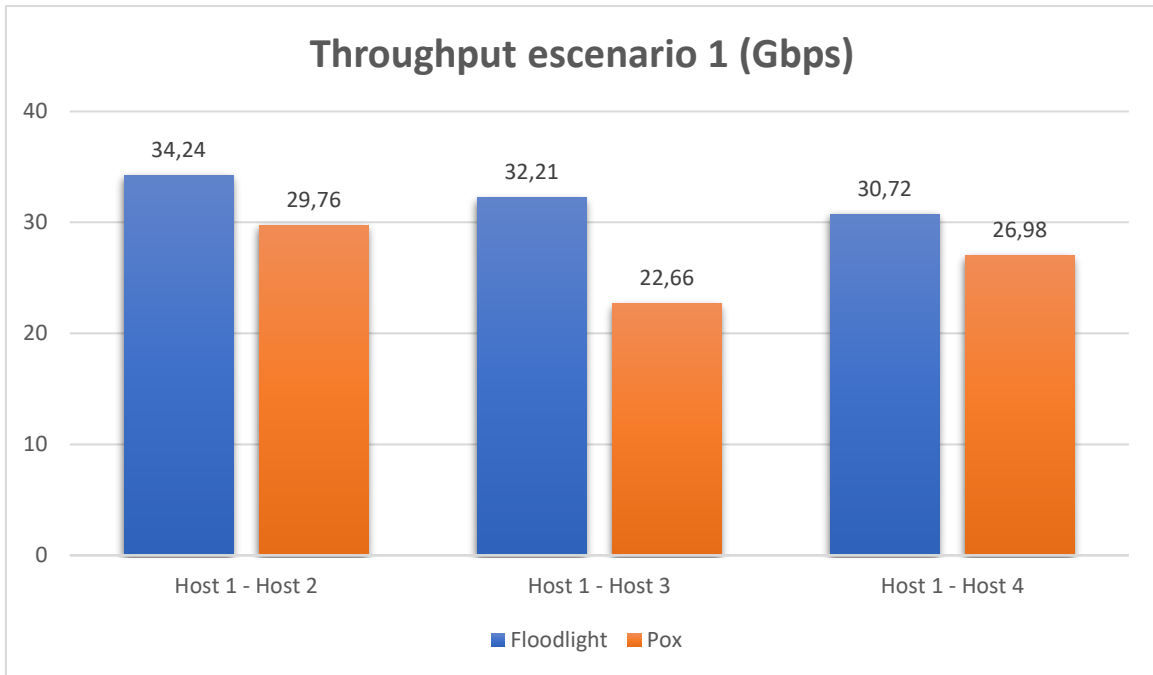


Figura 102 Throughput entre nodos de la red para el escenario 1

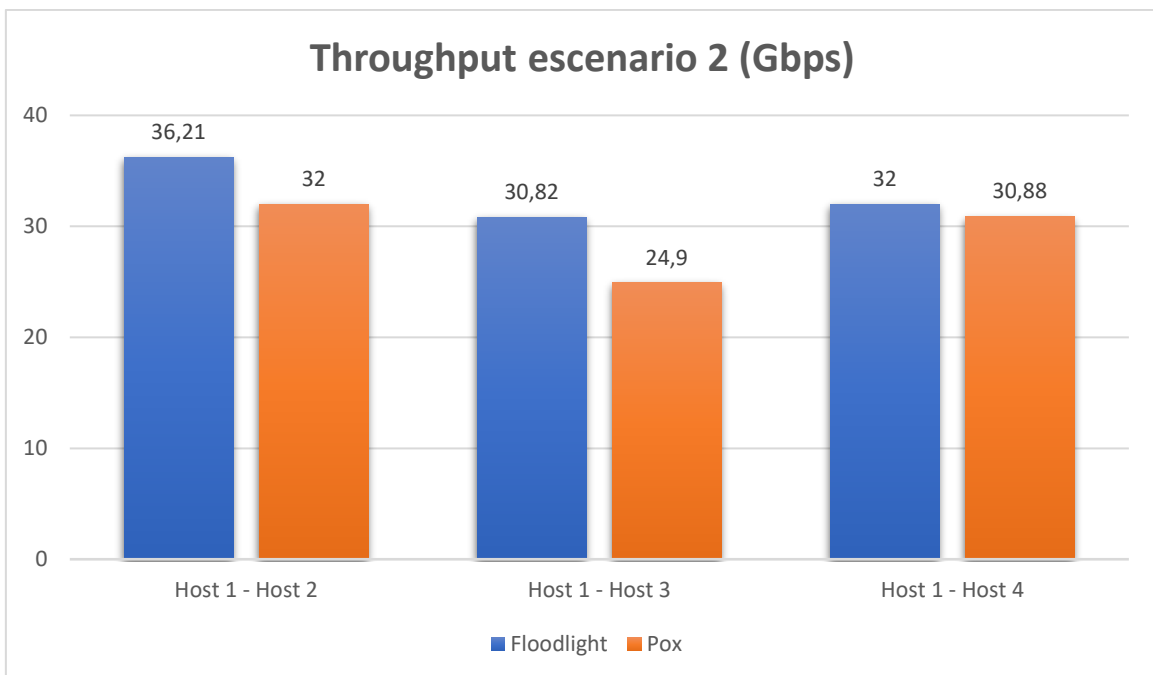


Figura 103 Throughput entre nodos de la red para el escenario 2

De igual manera que en las anteriores métricas, aunque en este caso no sea muy notorio, se mantiene el patrón de cómo el controlador Floodlight y su módulo de funcionamiento a partir de Java es mucho más rápido y eficiente que el módulo del

controlador Pox, el cual solamente funciona a base de python. En el escenario 1 Floodlight tiene valores de throughput bastante mejores que Pox, a diferencia del escenario 2 donde los dos controladores poseen valores relativamente similares.

Por último, aunque el controlador Floodlight siga mostrando mejores resultados que Pox, las métricas también muestran cómo cada transmisión hecha en la red SDN en ambos casos tiene valores muy altos de transferencia de datos sin ningún error aparente, por lo se estima que el diseño e implementación del Testbed SDN para posteriores prácticas y aplicaciones ha sido satisfactorio.

4.2 ANÁLISIS DE LOS AMBIENTES DE PRÁCTICA SDN

El primer ambiente de práctica SDN, desarrollado en la sección 3.4.1, además de presentar un punto de partida en el manejo de redes SDN y Mininet, permite entender de forma básica que OpenFlow funciona actualizando las entradas en la tabla de flujos del conmutador como lo indica la Figura 61.

En este caso, la diferencia sobre una red tradicional radica en que SDN no utiliza un protocolo de encaminamiento de paquetes o routing, más bien, SDN utiliza OpenFlow que define un estándar para que ciertas reglas de flujo (campos coincidentes de direcciones MAC, IP, Vlan, QoS o etiquetas MPLS) sean agregadas a la tabla de reenvío del conmutador, de la misma manera que se observa en la Figura 61. Además, en estas tablas se puede observar qué puerto será el encargado de transmitir las diferentes tramas y paquetes en ese conmutador.

De forma adicional, usando el software de captura de paquetes Wireshark, se logra verificar cómo el controlador actúa de cerebro en una red SDN, ya que como indica la Figura 104, un paquete ICMP antes de ser reenviado entre la direcciones IPv4 192.168.10.1 y 192.168.10.2, primero es procesado por el controlador, el cual controla el flujo de los dispositivo de renvío y direcciona al conmutador sobre qué acción tomar a partir de los mensajes *OFPT_PACKET_IN* y *OFPT_PACKET_OUT*.

No.	Time	Source	Destination	Protocol	Length	Info
54	4.036115000	192.168.10.1	192.168.10.2	ICMP	100	Echo (ping) request id=0x092b,
55	4.036215000	10.0.0.10	10.0.0.10	OpenFlow	208	Type: OFPT_PACKET_IN
69	4.043280000	10.0.0.10	10.0.0.10	OpenFlow	206	Type: OFPT_PACKET_OUT
71	4.043378000	192.168.10.1	192.168.10.2	ICMP	100	Echo (ping) request id=0x092b,
72	4.043393000	192.168.10.2	192.168.10.1	ICMP	100	Echo (ping) reply id=0x092b,
73	4.043531000	10.0.0.10	10.0.0.10	OpenFlow	208	Type: OFPT_PACKET_IN
75	4.045319000	10.0.0.10	10.0.0.10	OpenFlow	206	Type: OFPT_PACKET_OUT
77	4.045370000	192.168.10.2	192.168.10.1	ICMP	100	Echo (ping) reply id=0x092b,
205	4.112321000	192.168.10.2	192.168.10.1	ICMP	100	Echo (ping) request id=0x092e,

Figura 104 Filtrado de paquetes de la dirección IPv4 192.168.10.2

Con respecto a redes tradicionales, en el caso de Firewalls, es necesario recordar que estos sistemas actúan como protectores de conexiones externas no confiables e inseguras a redes privadas o internas, tal y como lo indica la Figura 105. Por consiguiente, un firewall de red tradicional, lo que hace es filtrar y escanear el tráfico de entrada y salida de una red, a partir de una lista de reglas o políticas de seguridad.

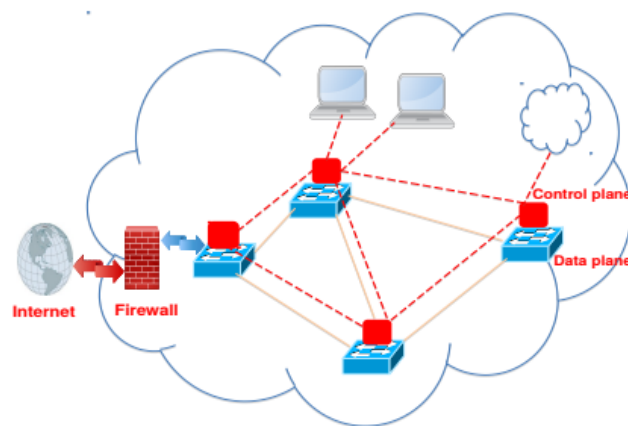


Figura 105 Firewall en una red tradicional

Fuente: (Hermant Dixit, y otros, 2018)

Por el contrario, en redes SDN, el Firewall puede ser representado como una aplicación que se ejecuta en el plano de control ubicado en el controlador, como se muestra en la Figura 106. Aquí, el controlador SDN usa diferentes reglas de tráfico y las convierte en reglas de flujo, las cuales se ejecutarán en cada una de las entradas de flujo de los conmutadores de la red, y actuarán como campos coincidentes de las tablas de envío de cada conmutador.

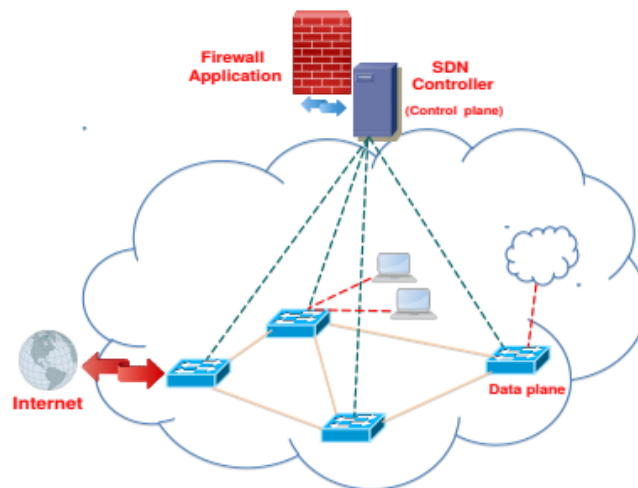


Figura 106 Firewall en una red SDN

Fuente: (Hermant Dixit, y otros, 2018)

Como resultado de todo este proceso, un Firewall SDN analizará el primer paquete entrante de cada flujo, donde lo comparará a los campos coincidentes, para luego agregar una regla de flujo en el resto de conmutadores o dispositivos de envío de la red. Esto se lo puede visualizar en la Figura 107, donde el paquete número 261 contiene un mensaje de request por parte del Host 4 con dirección IPv4 192.168.10.4, pero al ser procesado en el controlador, este es negado y nunca respondido de vuelta por el Host 1 con dirección IPv4 192.168.10.1

No.	Time	Source	Destination	Protocol	Length	Info
261	24.426031000	192.168.10.4	192.168.10.1	ICMP	100	Echo (ping) request id=0x09e0,
262	24.426167000	10.0.0.10	10.0.0.10	OpenFlow	208	Type: OFPT_PACKET_IN
270	24.476255000	10.0.0.10	10.0.0.10	OpenFlow	206	Type: OFPT_PACKET_OUT
272	24.476319000	192.168.10.4	192.168.10.1	ICMP	100	Echo (ping) request id=0x09e0
273	24.476320000	192.168.10.4	192.168.10.1	ICMP	100	Echo (ping) request id=0x09e0
274	24.476353000	192.168.10.4	192.168.10.1	ICMP	100	Echo (ping) request id=0x09e0
275	24.476354000	192.168.10.4	192.168.10.1	ICMP	100	Echo (ping) request id=0x09e0

Figura 107 Mensajes de request por parte del Host 4 hacia Host 1 sin respuesta

No obstante, cuando un flujo de paquetes entrantes coincide con una regla de flujo, como es el caso de la transmisión mostrada en la Figura 108, el host destino responderá a cada uno de los mensajes de request del host origen. Aquí, el Host 1 con dirección IPv4 192.168.10.1, envía mensajes de request que son procesados por el controlador paulatinamente en toda la conexión, y respondidos satisfactoriamente por el Host 2 con dirección IPv4 192.168.10.2 sin ningún contratiempo.

No.	Time	Source	Destination	Protocol	Length	Info
7753	453.835206000	192.168.10.1	192.168.10.2	ICMP	100	Echo (ping) request id=0x0a35,
7754	453.835338000	10.0.0.10	10.0.0.10	OpenFlow	208	Type: OFPT_PACKET_IN
7757	453.837554000	10.0.0.10	10.0.0.10	OpenFlow	206	Type: OFPT_PACKET_OUT
7759	453.837609000	192.168.10.1	192.168.10.2	ICMP	100	Echo (ping) request id=0x0a35,
7760	453.837618000	192.168.10.2	192.168.10.1	ICMP	100	Echo (ping) reply id=0x0a35,
7761	453.837709000	10.0.0.10	10.0.0.10	OpenFlow	208	Type: OFPT_PACKET_IN
7763	453.839123000	10.0.0.10	10.0.0.10	OpenFlow	206	Type: OFPT_PACKET_OUT
7765	453.839177000	192.168.10.2	192.168.10.1	ICMP	100	Echo (ping) reply id=0x0a35,
8343	466.846415000	192.168.10.2	192.168.10.1	ICMP	100	Echo (ping) request id=0x0a38,

Figura 108 Mensajes de request por parte del Host 1 hacia Host 2 con respuesta

Para finalizar, se debe enfatizar que el Testbed funciona como una herramienta base de estudio e investigación SDN, dónde el conjunto de aplicaciones que utilice el estudiante permitirá el posterior dominio de la tecnología, y aunque el material práctico presentado aquí es sólo la punta del iceberg de las redes definidas por software, se ha compilado de tal forma que futuros trabajos puedan usar esta plataforma para su beneficio y sea aprovechado al máximo.

CONCLUSIONES

Mediante la investigación bibliográfica llevada a cabo en el presente trabajo de grado, fue posible la comprensión del funcionamiento y operación de las Redes Definidas por Software (SDN), así como sus características distintivas con las redes tradicionales. También, gracias a documentación oficial, se logró definir los requerimientos necesarios para la instalación y uso de los controladores Pox y Floodlight, además del software de emulación para redes SDN Mininet.

En relación con el diseño e implementación del Testbed SDN, cabe señalar que la herramienta Mininet provee un ambiente de emulación controlado perfecto para estas redes, y que además permite la enseñanza e investigación de la tecnología SDN a partir de la creación de un diseño de red que no es más que la aplicación de un lenguaje de programación, en este caso Python, a partir de clases y métodos los cuales son expuestos en detalle por el software Mininet en su documentación oficial.

En este sentido, aprovechando la facilidad de uso de Mininet, fue posible verificar la operatividad y funcionamiento del diseño SDN para los dos escenarios propuestos, tomando en consideración métricas de evaluación basadas en el rfc 2544, el cual trata sobre la realización de pruebas en redes de transporte y servicio. Esto permitió no sólo verificar el diseño del Testbed SDN en un ambiente controlado, sino que también se logró comparar el funcionamiento de los controladores Pox y Floodlight.

Al momento de la implementación del Testbed SDN en la facultad FICA, se consiguió importar satisfactoriamente la Máquina Virtual anteriormente puesta a prueba, con esto se logró evitar partir desde 0 al momento de instalar el Testbed y los dos controladores SDN dentro de los servidores Poxmox de la carrera. Esto permitió que posteriormente a la implementación del Testbed, este se pueda poner a prueba y se logre

conseguir una conexión exitosa a él a partir de cualquier máquina que esté conectada a la red Eduroam UTN.

Con base a los conocimientos teóricos adquiridos acerca de la tecnología SDN, se propuso tres ambientes de práctica los cuales permiten el análisis del comportamiento de una red tradicional versus una SDN; el manejo de entradas y tablas de flujo, la creación y aplicación de lista de control de acceso; y finalmente, el levantamiento y manejo a partir de reglas de un firewall.

De los datos obtenidos en la medición de latencia a partir de cada controlador, se determinó que Floodlight presenta menor latencia que Pox para los dos escenarios de prueba, donde en el primero, los valores de latencia media del controlador Floodlight llegan hasta 1.90ms en su peor caso, a diferencia de Pox, en el cual los valores de latencia se disparan, llegando a alcanzar incluso los 13.87ms, es decir, la latencia se incrementa en casi un 700% más que los presentados por Floodlight. De igual manera, para el segundo escenario, Pox llega a tener valores de 15.97ms, un 400% más que los obtenidos por Floodlight, los cuales rozan los 4ms.

Por otra parte, el análisis del jitter, permitió determinar un patrón cuando se realiza la comparación de resultados entre los controladores Floodlight y Pox, en donde el primero siempre presenta mejores valores que el segundo. En el caso de Pox se obtuvo valores en promedio de 15.72ms, una cantidad muy superior si se comparan con los obtenidos para Floodlight, los cuales apenas se acercaron a los 2.89ms.

El throughput no presentó ninguna sorpresa, y el patrón donde el controlador Floodlight es superior a Pox se mantiene. A simple vista, las cantidades resultantes de esta métrica resultan similares, pero debido a que estos valores se miden en Gbps, la diferencia de promedios se hizo más evidente. Aquí, al momento de medir la capacidad de enviar datos sin pérdida de paquetes en un tiempo de 15 segundos, el controlador

Floodlight presentó un promedio de 32.39 y 33.01 Gbps en el escenario 1 y 2 respectivamente, a diferencia de los valores un poco menores de los obtenidos a partir del controlador Pox los cuales son 26.46 y 29.26 Gbps para cada escenario. Todo esto demuestra que la red SDN diseñada y puesta a prueba para el Testbed presenta valores muy altos de transferencia de datos sin errores en cualquier controlador y escenario.

Finalmente, la ejecución de los ambientes de práctica permitió entender la diferencia al momento de reenviar paquetes por parte de SDN y una red tradicional, dónde SDN usa campos coincidentes en una entrada de flujo que se origina a partir de la capa de control en el controlador, a diferencia una red tradicional que usa un protocolo en encaminamiento de paquetes en cada conmutador. Además, otra característica que se logró comprender mediante estos ambientes de práctica es que, al momento de configurar reglas de seguridad, ya sea en ACLs o en un firewall, SDN implementará estas en el controlador, el cual las convertirá en reglas de flujo y repartirá estas instrucciones a cada conmutador, a diferencia de una red tradicional, en la cual estas reglas deben ser implementadas en cada dispositivo de reenvío de la red necesario para su funcionamiento.

RECOMENDACIONES

Aunque exista mucha documentación acerca del uso del software Mininet, además de los controladores Pox y Floodlight, el soporte oficial que cada uno posee en sus páginas es el más completo y con menos errores de los que se pueden encontrar. Para una persona que desea iniciarse en esta tecnología, se recomienda partir desde esta documentación, ya que cualquier información adicional que existe viene en su mayoría de observaciones y pruebas a partir de aplicaciones o indicaciones presentadas en estos documentos.

Al momento de trabajar con dos controladores SDN en una misma máquina, es necesario verificar la compatibilidad que estos poseen por parte de Python en su funcionamiento. Hay que comprender que una máquina con SO Linux en cualquiera de sus kernel viene con una versión de Python preinstalada, pero usualmente esa no se utiliza al momento de ejecutar un controlador u otro, por lo que, si se pretende usar dos en un mismo dispositivo, es necesario realizar pruebas para comprender qué controlador es necesario instalar primero y qué versión de Python usar para así no caer en errores de versión al momento de ejecución.

Cada controlador posee un módulo el cual contiene las reglas que se aplicarán en la red SDN. Para la modificación de este se recomienda poseer un nivel alto en los lenguajes de programación Python y Java ya que las clases y métodos usados en este son extensas y numerosas, por lo cual es fácil confundirse aun usando la documentación que estos presentan en sus páginas oficiales.

Para las pruebas de operabilidad y funcionamiento del Testbed se recomienda guiarse en el rfc 2544, el cuál posee diversos subtests a partir de diferentes métricas que permite verificar el estado de una red. Aunque en este caso sólo se usaron tres, guiarse por este rfc brinda una estrategia que proporcionan el entendimiento de cómo se encuentra

la red, y en el caso de una SDN, permite evaluar de primera mano la ejecución y estado de los controladores SDN en un ambiente controlado.

Como trabajo a futuro, se considera modificar el diseño de la red SDN adicionando parámetros en los enlaces y dispositivos para realizar pruebas de operabilidad con características de red aún más reales. También se aconseja la experimentación en el área de seguridad que provee cada controlador, tomando como punto de partida que Floodlight tiene una aplicación firewall desde su REST API, a diferencia de Pox que necesita la creación de un algoritmo para proveer estos mismos beneficios desde su módulo principal. Finalmente, se sugiere transportar el Testbed a un ambiente práctico para así lograr un entendimiento más profundo de esta nueva tecnología y sus aplicaciones.

Bibliografía

- IBM Cloud Education. (15 de Abril de 2021). *IBM*. Obtenido de <https://www.ibm.com/cloud/blog/software-defined-networking>
- Alcorn, J., & Melton, S. (2017). SDN data path confidence analysis. *2017 IEEE Conference on Dependable and Secure Computing* (págs. 209-216). IEEE.
- Azodolmolky, S. (2013). *Software Defined Networking with OpenFlow*. Birmingham: Packt Publishing.
- Contreras Higuera , M., & Granados Acuña , G. (15 de Noviembre de 2012). *Universidad Nacional Abierta y a Distancia*. Obtenido de https://www.researchgate.net/publication/317150558_Estimacion_del_Valor_Teorico_para_el_Throughput_en_Redex_LAN_Basadas_en_Tecnologia_Power_Line_Communications_Bajo_el_Estandar_Homeplug_10/fulltext/592811f4458515e3d465854a/Estimacion-del-Valor-Teorico-par
- Contributors, M. P. (2021). *Mininet Walkthrough*. Obtenido de <http://mininet.org/walkthrough/>
- Córdoba López, S. (2019). Estudio de redes SDN mediante Mininet y Miniedit. *Universidad Politécnica de Valencia*.
- Curl. (2022). *Curl*. Obtenido de <https://curl.se/docs/manpage.html#-x>
- Doxygen. (2021). *Mininet Python API Reference Manual*. Obtenido de <http://mininet.org/api/index.html>
- Fancy, C., & Pushpalatha, M. (2017). Performance evaluation of SDN controllers POX and floodlight in mininet emulation environment. *2017 International Conference on Intelligent Sustainable Systems (ICISS)*. Palladam: IEEE.
- Guerrero, D. (30 de Abril de 2019). *Repositorio Digital UTN*. Obtenido de Repositorio Digital UTN: <http://repositorio.utn.edu.ec/bitstream/123456789/9107/1/04%20RED%20221%20TRABAJO%20DE%20GRADO.pdf>
- Hermant Dixit, V., Doupé, A., Zhao, Z., Kyung, S., Shoshitaishvili, Y., & Joon, G. (2018). Challenges and Preparedness of SDN-based Firewalls. *Arizona State University*.
- Huang, D., Chowdary, A., & Pishadory, S. (2019). *Software-Defined Networking and Security*. Florida: CRC Press.
- IETF. (Marzo de 1999). *IETF*. Obtenido de <https://datatracker.ietf.org/doc/html/rfc2544>

- iPerf. (2022). Obtenido de <https://iperf.fr/>
- IRTF. (Enero de 2015). *Internet Research Task Force*. Obtenido de <https://datatracker.ietf.org/doc/html/rfc7426#ref-OF08>
- LiveAction. (2022). *LiveAction*. Obtenido de <https://www.liveaction.com/resources/tips-and-tricks/network-jitter/>
- Morrelae, P., & Anderson, J. (2015). *Software Defined Networking: Design and Deployment*. Boca Raton: CRC Press.
- Muhammad , R., Samineni, V., & Robertson, W. (2016). Physical and logical topology slicing through SDN. *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. Vancouver: IEEE.
- Nicolalde, W. (10 de Febrero de 2021). *Repositorio digital UTN*. Obtenido de Repositorio digital UTN: <http://repositorio.utn.edu.ec/bitstream/123456789/10907/2/04%20RED%20254%20TRABAJO%20GRADO.pdf>
- Noman, H. M., & Jasim, d. M. (2020). POX Controller and Open Flow Performance Evaluation in Software. *3rd International Conference on Sustainable Engineering Techniques (ICSET 2020)*. Babylon: IOP Publishing.
- ONF. (2012). *Software-Defined Networking: The New Norm for Networks*. Open Networking Foundation.
- ONF. (2013). *OpenFlow Switch Specification*.
- OvS. (2016). *Open vSwitch*. Obtenido de <https://docs.openvswitch.org/en/latest/>
- Paul Göransson, C. B. (2017). *Software Defined Networks: A Comprehensive Approach*. Cambridge: Morgan Kaufmann.
- POX. (s.f.). *POX Manual Current documentation*. Obtenido de <https://noxrepo.github.io/pox-doc/html/index.html>
- Ramanathan, S., Kannan, P., Mirkovic, J., & Sklower, K. (2017). Enabling SDN Experimentation in Network Testbeds. *Proceedings of ACM SDNNFV Security Workshop*. ISI.
- RAMIREZ, J. L. (2015). *Universidad Tecnológica de Pereira*. Obtenido de <https://docplayer.es/16995496-Guia-de-implementacion-y-uso-del-emulador-de-redes-mininet-jose-leonardo-henao-ramirez.html>
- Ribeiro, T. (25 de Octubre de 2018). *Project Floodlight*. Obtenido de <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview?homepageId=1343545>

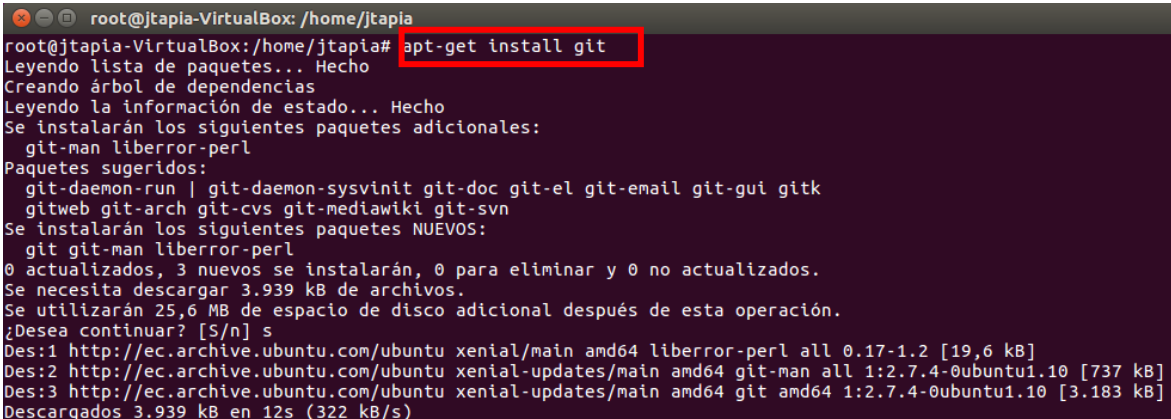
- Sadov, O., Grudin, V., Vlasov, D., & Titov, V. (2015). OpenFlow SDN testbed for Storage Area Network. *2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC)*. Moscow: IEEE.
- Shaghghi, A., Mohamed, A., & Rajkumar, B. (1 de Abril de 2018). *Cornell University*.
Obtenido de <https://arxiv.org/pdf/1804.00262.pdf>
- Stallings, W. (2016). *Foundations of Modern Networking*. Crawfordsville: Pearson Education.
- Tanenbaum, A. S. (2012). *Redes de Computadoras*. México: Pearson Educación.

6. ANEXOS

Anexo 1. Instalación del emulador Mininet

Los requisitos para una correcta instalación del emulador Mininet son mínimos, esto debido a que la mayoría de sus componentes vienen por defecto ya instalados en cualquier versión kernel de Linux (Ubuntu/Debian/Centos). Además, al momento de realizar la descarga del paquete Mininet, esta se encarga de buscar todos los requisitos adicionales que se podrían tener posteriormente según la versión de kernel utilizada por el usuario, por lo que para su instalación sólo es necesario seguir paso a paso la guía ya proporcionada en <http://mininet.org/download/> con la adición de ciertos pasos extras que se deben realizar para así evitar cualquier falla de compatibilidad de Mininet a futuro, los cuales se explican a continuación.

Para comenzar es necesario tener instalado el paquete Git, el cual permite el uso de GitHub en cualquier distribución de Linux. Este se lo instala mediante la consola o terminal de Linux a través del comando `apt-get install git` como indica la Figura 109.



```
root@jtapia-VirtualBox: /home/jtapia
root@jtapia-VirtualBox: /home/jtapia# apt-get install git
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  git-man liberror-perl
Paquetes sugeridos:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk
  gitweb git-arch git-cvs git-mediawiki git-svn
Se instalarán los siguientes paquetes NUEVOS:
  git git-man liberror-perl
0 actualizados, 3 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 3.939 kB de archivos.
Se utilizarán 25,6 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://ec.archive.ubuntu.com/ubuntu xenial/main amd64 liberror-perl all 0.17-1.2 [19,6 kB]
Des:2 http://ec.archive.ubuntu.com/ubuntu xenial-updates/main amd64 git-man all 1:2.7.4-0ubuntu1.10 [737 kB]
Des:3 http://ec.archive.ubuntu.com/ubuntu xenial-updates/main amd64 git amd64 1:2.7.4-0ubuntu1.10 [3.183 kB]
Descargados 3.939 kB en 12s (322 kB/s)
```

Figura 109 Instalación del paquete git.

Con el paquete git adquirido, y como muestra la Figura 110, se procede a clonar desde GitHub los archivos necesarios para la instalación de *Mininet*. Esto se realiza

mediante el comando `sudo git clone git://github.com/mininet/mininet` el cual siempre va a descargar la versión activa más reciente de Mininet.

```
root@jtapia-VirtualBox:/home/jtapia# sudo git clone git://github.com/mininet/mininet
Clonar en «mininet»...
remote: Enumerating objects: 10165, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 10165 (delta 2), reused 7 (delta 2), pack-reused 10154
Receiving objects: 100% (10165/10165), 3.19 MiB | 309.00 KiB/s, done.
Resolving deltas: 100% (6784/6784), done.
Comprobando la conectividad... hecho.
root@jtapia-VirtualBox:/home/jtapia#
```

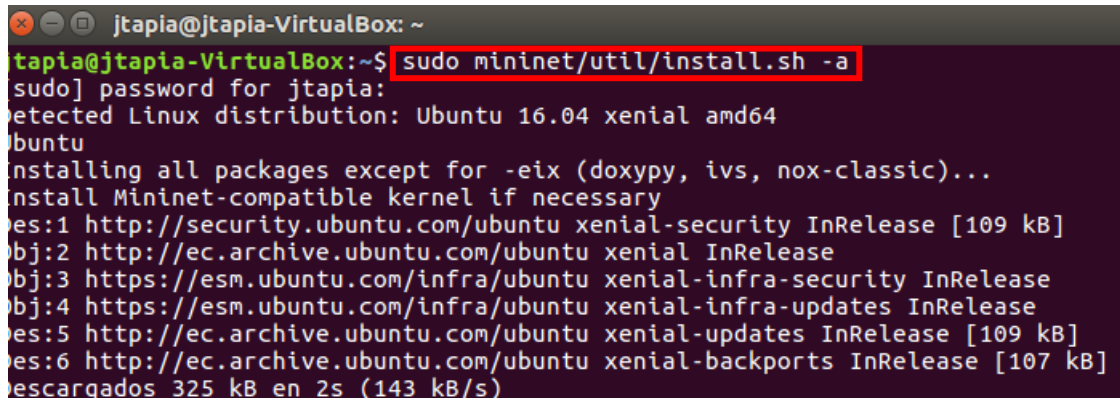
Figura 110 Descarga paquete Mininet desde GitHub

Ahora, para ejecutar la versión más reciente del emulador u otro de preferencia del usuario, hay que dirigirse a la carpeta de descarga de Mininet, ejecutar el comando `git tag`, y así se podrá visualizar las versiones existentes tal cual como indica la Figura 111. Finalmente, solo hay que seleccionar la versión a utilizar, en este caso la 2.2.2 a través del comando `sudo git checkout -b 2.2.2 2.2.2`.

```
root@jtapia-VirtualBox: /home/jtapia/mininet
root@jtapia-VirtualBox:~# cd /home/jtapia
root@jtapia-VirtualBox:/home/jtapia# ls -l
total 48
drwxr-xr-x  2 jtapia jtapia 4096 may 16 19:42 Descargas
drwxr-xr-x  2 jtapia jtapia 4096 may 16 19:42 Documentos
drwxr-xr-x  2 jtapia jtapia 4096 may 16 19:42 Escritorio
-rw-r--r--  1 jtapia jtapia 8980 may 16 19:30 examples.desktop
drwxr-xr-x  2 jtapia jtapia 4096 may 16 19:42 Imágenes
drwxr-xr-x 11 root   root   4096 may 16 20:26 mininet
drwxr-xr-x  2 jtapia jtapia 4096 may 16 19:42 Música
drwxr-xr-x  2 jtapia jtapia 4096 may 16 19:42 Plantillas
drwxr-xr-x  2 jtapia jtapia 4096 may 16 19:42 Público
drwxr-xr-x  2 jtapia jtapia 4096 may 16 19:42 Vídeos
root@jtapia-VirtualBox:/home/jtapia# cd mininet/
root@jtapia-VirtualBox:/home/jtapia/mininet# git tag
1.0.0
2.0.0
2.1.0
2.1.0p1
2.1.0p2
2.2.0
2.2.1
2.2.2
2.3.0
2.3.0b1
2.3.0b2
2.3.0d3
2.3.0d4
2.3.0d5
2.3.0d6
2.3.0rc1
2.3.0rc2
cs244-spring-2012-final
root@jtapia-VirtualBox:/home/jtapia/mininet# sudo git checkout -b 2.2.2 2.2.2
Switched to a new branch '2.2.2'
```

Figura 111 Seleccionando la versión de Mininet a ejecutar

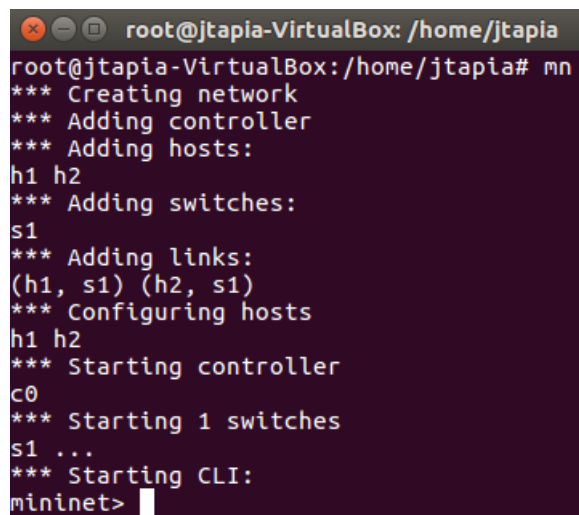
Con la versión ya seleccionada, se procede a instalar Mininet mediante el comando `sudo mininet/util/install.sh -a` como indica la Figura 112. La instalación detectará de forma automática la distribución de Linux activa y procederá a desempaquetar los archivos necesarios para posteriormente instalarlos.



```
jtapia@jtapia-VirtualBox: ~
jtapia@jtapia-VirtualBox:~$ sudo mininet/util/install.sh -a
[sudo] password for jtapia:
detected Linux distribution: Ubuntu 16.04 xenial amd64
Ubuntu
Installing all packages except for -eix (doxypy, ivs, nox-classic)...
Install Mininet-compatible kernel if necessary
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]
Get:2 http://ec.archive.ubuntu.com/ubuntu xenial InRelease
Get:3 https://esm.ubuntu.com/infra/ubuntu xenial-infra-security InRelease
Get:4 https://esm.ubuntu.com/infra/ubuntu xenial-infra-updates InRelease
Get:5 http://ec.archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:6 http://ec.archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
Fetched 325 kB in 2s (143 kB/s)
```

Figura 112 Instalación Mininet

Finalmente, se realiza una pequeña prueba mediante el comando `mn` el cual creará una topología SDN por defecto con un conmutador y dos hosts como indica la Figura 113.



```
root@jtapia-VirtualBox: /home/jtapia
root@jtapia-VirtualBox: /home/jtapia# mn
*** 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 113 Ejecución Mininet

Anexo 2. Instalación del controlador POX

Al momento de instalar el emulador Mininet, también el controlador POX se instala por defecto, por lo cual no es necesario realizar más pasos adicionales. Ahora, para verificar la existencia de este sólo hay que buscarlo en la carpeta personal que se encuentra en el directorio principal de la máquina Linux como indica la Figura 114.

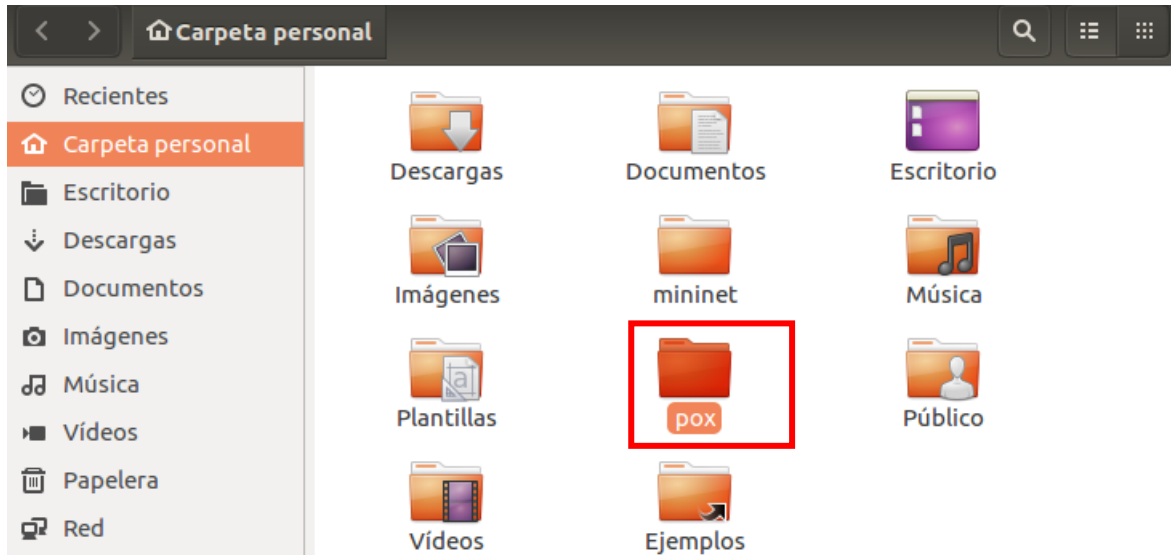


Figura 114 Verificación de carpeta Pox

Después de verificar el directorio POX, hay que dirigirse a él, donde a partir de aquí se procederá a inicializar el controlador en modo DEBUG como indica la Figura 115. Esto se realiza para comprobar cualquier error previo antes de su ejecución.

```
juan@juan-VirtualBox:~/pox$ sudo ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.5.2/Jul 17 2020 14:04:10)
DEBUG:core:Platform is Linux-4.15.0-112-generic-x86_64-with-Ubuntu-16.04-xenial
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.5.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

Figura 115 Inicialización del controlador POX

Finalmente, se realiza la conexión del emulador Mininet con el controlador POX mediante una topología sencilla como muestra la Figura 116 a través del comando `sudo mn --topo single,3 --mac --switch ovsk --controller remote`. Si todo funciona correctamente el controlador se conectará con cero errores tal y como indica la Figura 117.

```
juan@juan-VirtualBox:~/pox$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
[sudo] password for juan:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> █
```

Figura 116 Ejecución de Mininet con POX

```
CINFO:openflow.of_01:[00-00-00-00-00-01 2] connected
*DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 2]
```

Figura 117 Controlador POX conectado correctamente

Anexo 3. Instalación del controlador Floodlight

Este controlador posee una máquina virtual preconfigurada y lista con Mininet, Open vSwitch, Java v1.7 y Floodlight v1.2 la cual se encuentra en la dirección <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/8650780/Floodlight+VM>.

En la MV, simplemente es necesario realizar una prueba de conexión entre el controlador y una topología SDN cualquiera para verificar que el controlador se encuentra

libre de errores. Para esto, primero se inicializa el controlador como indica la Figura 118 mediante el comando `java -jar target/Floodlight.jar`.

```
floodlight@floodlight:~$ cd floodlight/  
floodlight@floodlight:~/floodlight$ java -jar target/floodlight.jar
```

Figura 118 Inicialización de Floodlight

Ya inicializado Floodlight, se procede a ejecutar la topología tal y como indica la Figura 119. Después de que la topología se inicializa, se accede a la interfaz gráfica proporcionada por el controlador a partir de la dirección IP de este más su puerto correspondiente, el cual en este caso es el 8080 como se observa en la Figura 120. En la interfaz se verifica el *uptime* o tiempo activo de la red SDN y su estado de *true* lo que demuestra que la red está ejecutándose sin errores.

```
floodlight@floodlight:~/floodlight$ sudo mn --custom TopologiaTestbed.py --topo=  
mytopo --controller=remote,ip=10.0.2.15,port=6653 --link=tc  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2 h3 h4  
*** Adding switches:  
s1 s2 s3  
*** Adding links:  
(h1, s2) (h2, s2) (h3, s3) (h4, s3) (s1, s3) (s2, s1)  
*** Configuring hosts  
h1 h2 h3 h4  
*** Starting controller  
c0  
*** Starting 3 switches  
s1 s2 s3 ...  
*** Starting CLI:
```

Figura 119 Ejecución de la topología SDN con el controlador Floodlight

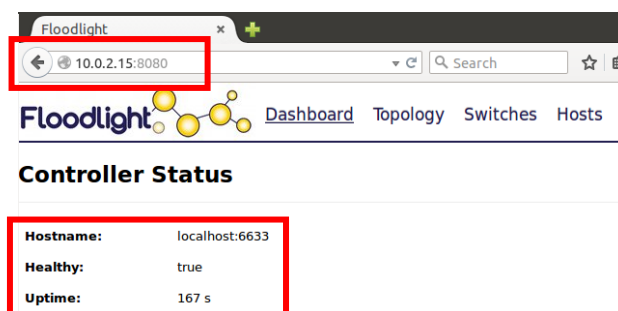


Figura 120 Interfaz gráfica Floodlight

Anexo 4. Creación de topología predefinida Sencilla (Single)

Esta topología predefinida o por defecto, consiste en un simple switch conectado a 2 hosts, donde el único parámetro modificable es el número de hosts conectados al switch. Su comando es `sudo mn -topo single,N` donde N es el número de hosts (Doxygen, 2021).

```
juan@juan-VirtualBox:~$ sudo mn --topo single,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

Figura 121 Ejemplo topología Sencilla

Como se observa en la Figura 121, el valor de N se ha reemplazado con 4, lo cual significa que Mininet creará una topología sencilla que consta de 1 switch (s1) al cual se conectan 4 hosts denominados h1, h2, h3 y h4. Y adicionalmente el comando inicializa la línea de comandos (CLI) de Mininet.

Para verificar la conectividad de la red desplegada se ejecuta el comando `pingall` en el CLI de mininet, tal como se muestra en la Figura 122, en donde se evidencia que cada host tiene respuesta de los otros con un resultado de 0% de paquetes perdidos.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Figura 122 Prueba de conectividad entre hosts

Anexo 5. Creación de topología predefinida Lineal (Linear)

A diferencia de la topología single, una linear permite la modificación en el número de conmutadores en la red, así como el número de hosts conectados a estos, el cual por defecto viene en una relación de 1 a 1, es decir cada host conectado a su respectivo switch y los switches interconectados entre ellos. Su comando por defecto es `sudo mn -topo linear,N`, donde N es el número de conmutadores deseados.

```
juan@juan-VirtualBox:~$ sudo mn --topo linear,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
```

Figura 123 Ejemplo topología linear

En la Figura 123 se ha creado una topología linear, donde el valor de N es igual a 4, lo que permite a Mininet crear la misma cantidad de conmutadores (s1, s2, s3 y s4) a los cuales estará conectado 1 host. Adicionalmente, y cómo se muestra en la Figura 124, se realizó una prueba de conectividad entre todos los hosts mediante el comando `pingall`.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Figura 124 Prueba de conectividad

Anexo 6. Creación de topología predefinida Árbol (Tree)

Finalmente, las topologías tipo árbol vienen dadas por el comando `sudo mn --topo tree,depth=N,fanout=M`, donde N es el número de niveles que van a existir y M el número de hosts que cada nivel tendrá.

```
juan@juan-VirtualBox:~$ sudo mn --topo tree,depth=2,fanout=2
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
```

Figura 125 Ejemplo topología Árbol

Como muestra en la Figura 125, se reemplazaron los valores de N y M con 2, lo cual significa que Mininet creará una topología de 2 niveles con 2 hosts cada uno. De igual manera se probó la conectividad entre todos los hosts como indica la Figura 126.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Figura 126 Prueba de conectividad

Anexo 7. Creación de topologías en Miniedit

A diferencia de las topologías por defecto, Miniedit permite a partir de una interfaz gráfica, desarrollar una red casi sin ninguna limitante de conmutadores, hosts y enlaces, donde el único obstáculo viene a ser la capacidad operativa del ordenador donde se realiza la emulación. Para abrir la interfaz de Miniedit sólo es necesario ejecutar en la consola Linux el siguiente comando: `cd ./mininet/examples/Miniedit.py`.

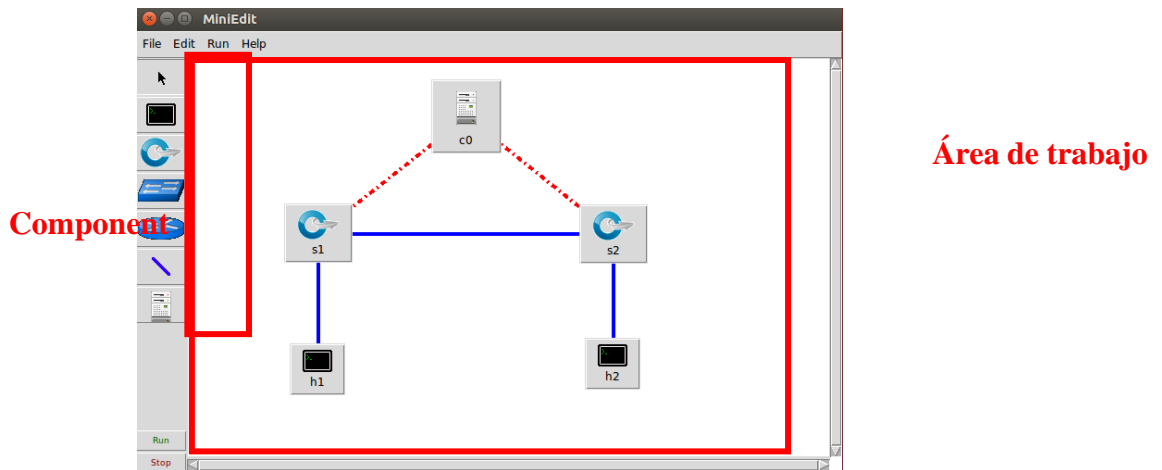


Figura 127 Interfaz Miniedit

La Figura 127 muestra la interfaz de Miniedit, la cual es sencilla e intuitiva, ya que sólo es necesario arrastrar los diferentes componentes a utilizar, los cuales ya se encuentran diferenciados gráficamente en el menú de la parte izquierda de la interfaz. Posteriormente a ubicarlos en el área de trabajo, lo único que se debe realizar es conectarlos mediante un link o enlace. Adicionalmente, cada dispositivo tendrá parámetros modificables como se indica en la Figura 128, donde un link puede ser configurado en su ancho de banda, retardo, pérdida de paquetes, entre otros.

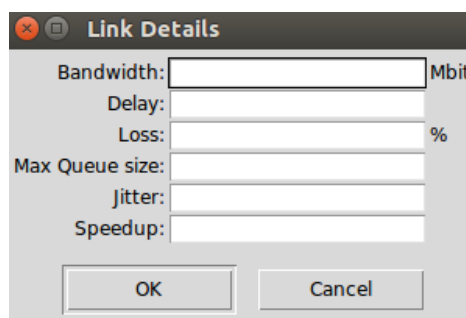


Figura 128 Parámetros modificables del enlace o link

Cuando se ejecute la topología se abrirá automáticamente la CLI de Mininet, y como muestra la Figura 129, se verificará la creación de la red a partir de la topología diseñada en Miniedit. Adicionalmente, y cómo se realizó en las topologías anteriores,

mediante el comando `pingall` se realiza una prueba de conectividad tal como se indica en la Figura 130.

```
Build network based on our topology.
Getting Hosts and Switches.
Getting controller selection:ref
<class 'mininet.node.Host'>
<class 'mininet.node.Host'>
Getting Links.
*** Configuring hosts
h1 h2
**** Starting 1 controllers
c0
**** Starting 2 switches
s2 s1
No NetFlow targets specified.
No sFlow targets specified.
```

Figura 129 Topología en Mininet a partir de Miniedit

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

Figura 130 Prueba de conectividad

Anexo 8. Creación de topología Personalizada en Mininet

La forma más completa para trabajar en redes SDN con Mininet es mediante topologías personalizadas. Estas, permiten desarrollar topologías de redes SDN a partir de un archivo ejecutable Python el cual posee la ventaja de ser totalmente modificable en cualquier momento. Este archivo se lo genera desde cero y tiene la capacidad de ser muy simple, o complejo, dependiendo del nivel de entendimiento en programación y características de Mininet y Python que posea la persona a cargo de su desarrollo.

Para su creación se usará una mezcla de funciones, clases y métodos en un archivo `.py` (archivo Python), el cual puede ser creado a partir de la línea de comandos Linux

mediante cualquier editor de texto. Un ejemplo de topología creada desde cero está compuesto de la siguiente manera:

La parte inicial del código consiste en llamar todas las clases necesarias según el tipo de topología que se ha planteado generar, en este caso, y como se muestra en la Figura 131, sólo se llamará a la clase “MyTopo”, la cual se refiere a la representación de la topología en el archivo Python que se está trabajando.

```
from mininet.topo import Topo

class MyTopo(Topo):
    def __init__(self):
        Topo.__init__(self)
```

Figura 131 Inicio topología básica personalizada

Después, y como se muestra en la Figura 132, se procede a crear una red la cual consiste en tres conmutadores y dos hosts. Las líneas de código adicionales en cada enlace determinan que su valor de ancho de banda en la red será tomado del campo *bw* especificado en cada línea, y más no de un valor aleatorio que puede asignar la red al momento de su ejecución.

```
rightHost1 = self.addHost('h4')
rightHost2 = self.addHost('h5')
leftSwitch = self.addSwitch('s1')
middleSwitch = self.addSwitch('s2')
rightSwitch = self.addSwitch('s3')

self.addLink(leftHost1, leftSwitch, bw=5)
self.addLink(leftHost2, leftSwitch, bw=5)
self.addLink(middleHost, middleSwitch, bw=5)
self.addLink(rightHost1, rightSwitch, bw=5)
self.addLink(rightHost2, rightSwitch, bw=15)
self.addLink(leftSwitch, middleSwitch, bw=15)
self.addLink(middleSwitch, rightSwitch, bw=15)
```

Figura 132 Creación de Switches, hosts y links

Finalmente, se usa la función lambda para la ejecución del archivo de forma total debido a que esta permite al código su aplicación inmediata, a diferencia de otras funciones que ejecutan el archivo progresivamente a manera de compilación línea por línea.

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

Figura 133 Ejecución de la función

Para ejecutar el archivo `.py` creado, hay que dirigirse al directorio y ejecutar el comando: `sudo mn -custom prueba1.py -topo=mytopo -mac -switch ovsk -controller=remote,ip=10.0.2.15,port=6653 -link=tc`. Este comando de ejecución usualmente es el mismo para cualquier topología en Mininet, donde puede variar en campos como el nombre de archivo Python (`prueba1.py`) o la dirección IP del controlador SDN (IP = 10.0.2.15) como muestra la Figura 134.

```
floodlight@floodlight:~/floodlight$ sudo mn --custom prueba1.py --topo=mytopo --mac --switch ovsk --controller=remote,ip=10.0.2.15,port=6653 --link=tc
*** Creating network
*** Adding controller
Unable to contact the remote controller at 10.0.2.15:6653
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3
*** Adding links:
(5.00Mbit) (5.00Mbit) (h1, s1) (5.00Mbit) (5.00Mbit) (h2, s1) (5.00Mbit) (5.00Mbit) (h3, s2) (5.00Mbit) (5.00Mbit) (h4, s3) (15.00Mbit) (15.00Mbit) (h5, s3) (15.00Mbit) (15.00Mbit) (s1, s2) (15.00Mbit) (15.00Mbit) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ..(5.00Mbit) (5.00Mbit) (15.00Mbit) (5.00Mbit) (15.00Mbit) (15.00Mbit) (5.00Mbit) (15.00Mbit) (15.00Mbit)
*** Starting CLI:
```

Figura 134 Topología personalizada a partir de Python

Inmediatamente después de la ejecución del archivo Python, se visualiza los diferentes componentes creados de la red SDN a partir del código generado. También permite verificar la asignación de valores en cada parámetro configurado, por ejemplo, el ancho de banda que fue asignado de forma específica a cada enlace. Finalmente, y de la

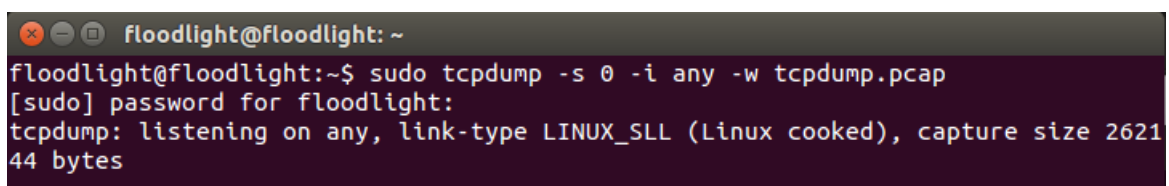
misma manera que en las topologías descritas anteriormente, se realiza una prueba básica de conectividad entre hosts usando el comando *pingall*. La Figura 135 evidencia que cada host tiene respuesta de los otros con un resultado de 0% de paquetes perdidos.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

Figura 135 Prueba de conectividad topología personalizada

Anexo 9. Captura de paquetes de forma remota en el Testbed SDN

Para capturar paquetes del Testbed SDN de forma remota sólo es necesario tener instalado un cliente SFTP o FTP en la máquina local donde se busca analizar el tráfico de la red. De esta manera, como indica la Figura 136, mientras se trabaja de forma remota en el Testbed SDN, se ejecutará el comando `sudo tcpdump -s 0 -i any -w tcpdump.pcap`, el cual permite capturar todos los paquetes de la red SDN en un archivo llamado *tcpdump.pcap* en la máquina dónde se encuentra el Testbed.



```
floodlight@floodlight: ~
floodlight@floodlight:~$ sudo tcpdump -s 0 -i any -w tcpdump.pcap
[sudo] password for floodlight:
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
```

Figura 136 Ejecución de *tcpdump* para captura de paquetes de forma remota

Posteriormente, para analizar los paquetes capturados, se procede a detener el proceso mediante *Ctrl + C* como indica la Figura 137, con lo que permitirá que el archivo *tcpdum.pcap* se guarde en el directorio */home* del Testbed SDN.


```
floodlight@floodlight: ~
floodlight@floodlight:~$ sudo tcpdump -s 0 -i any -w tcpdump.pcap
[sudo] password for floodlight:
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture size 2621
44 bytes
^C4398 packets captured
5446 packets received by filter
0 packets dropped by kernel
floodlight@floodlight:~$
```

Figura 137 Detención de captura de paquetes

Finalmente, mediante un cliente SFTP, en este caso WinSCP, se realiza una conexión al Testbed SDN mediante las credenciales indicadas en la Figura 138. A partir de esto, se procederá a copiar el archivo en la máquina local y abrir con el analizador de paquetes Wireshark como indica la Figura 139.

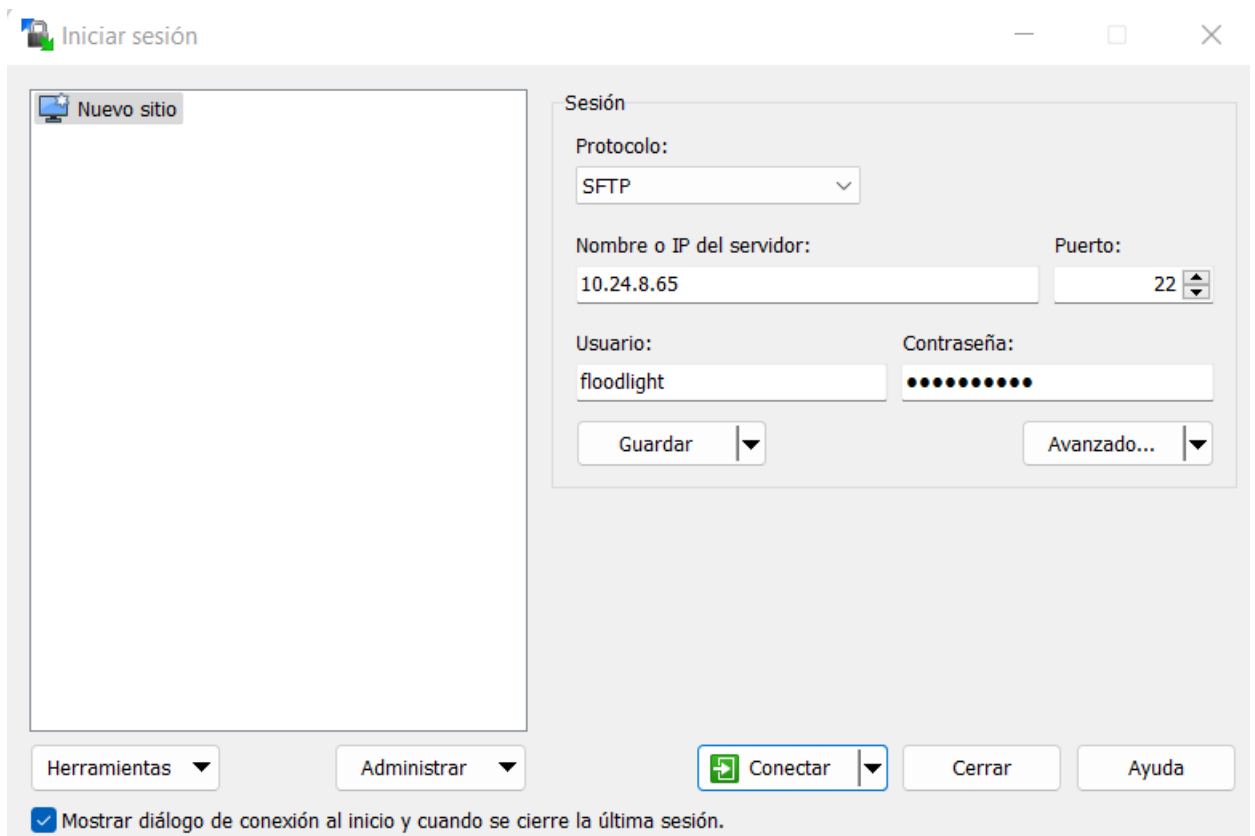


Figura 138 Conexión SFTP para copia de archivos de forma remota

No.	Time	Source	Destination	Protocol	Length	Info
2119	143.047118	10.0.0.10	10.0.0.10	OpenFlow	76	Type: OFPT_HELLO
2121	143.047147	10.0.0.10	10.0.0.10	OpenFlow	76	Type: OFPT_HELLO
2123	143.047163	10.0.0.10	10.0.0.10	OpenFlow	76	Type: OFPT_HELLO
2144	143.123256	10.0.0.10	10.0.0.10	OpenFlow	76	Type: OFPT_FEATURES_REQUEST
2146	143.123489	10.0.0.10	10.0.0.10	OpenFlow	100	Type: OFPT_FEATURES_REPLY
2147	143.123940	10.0.0.10	10.0.0.10	OpenFlow	76	Type: OFPT_FEATURES_REQUEST
2149	143.124082	10.0.0.10	10.0.0.10	OpenFlow	100	Type: OFPT_FEATURES_REPLY
2151	143.139010	10.0.0.10	10.0.0.10	OpenFlow	76	Type: OFPT_FEATURES_REQUEST
2153	143.139172	10.0.0.10	10.0.0.10	OpenFlow	100	Type: OFPT_FEATURES_REPLY
2164	143.202109	10.0.0.10	10.0.0.10	OpenFlow	84	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
2165	143.202178	10.0.0.10	10.0.0.10	OpenFlow	340	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
2167	143.203963	10.0.0.10	10.0.0.10	OpenFlow	84	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
2168	143.204002	10.0.0.10	10.0.0.10	OpenFlow	276	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
2170	143.218911	10.0.0.10	10.0.0.10	OpenFlow	84	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
2171	143.218970	10.0.0.10	10.0.0.10	OpenFlow	340	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
2173	143.234726	10.0.0.10	10.0.0.10	OpenFlow	96	Type: OFPT_GET_CONFIG_REQUEST
2174	143.234769	10.0.0.10	10.0.0.10	OpenFlow	76	Type: OFPT_BARRIER_REPLY
2176	143.234790	10.0.0.10	10.0.0.10	OpenFlow	96	Type: OFPT_GET_CONFIG_REPLY

> Frame 2119: 76 bytes on wire (608 bits), 76 bytes captured (608 bits)
 > Linux cooked capture v1
 > Internet Protocol Version 4, Src: 10.0.0.10, Dst: 10.0.0.10
 > Transmission Control Protocol, Src Port: 35755, Dst Port: 6653, Seq: 1, Ack: 1, Len: 8
 > OpenFlow 1.3

Figura 139 Paquete capturados de forma remota en Wireshark