UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA:
INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a la obtención del título de:

Ingenieros de Sistemas

TEMA:

EVALUACIÓN DEL RENDIMIENTO DE UN PROTOTIPO SDN (SOFTWARE DEFINED NETWORKING) BAJO EL PROTOCOLO OPENFLOW UTILIZANDO HERRAMIENTAS OPEN SOURCE EN UN ENTORNO VIRTUALIZADO

AUTORES:

DIEGO JOSÉ LASSO GUAMÁN

JEFFERSON RICARDO PUCHAICELA CONDOY

TUTOR:

MANUEL RAFAEL JAYA DUCHE

Quito, febrero del 2021

CESIÓN DE DERECHOS DE AUTOR

Nosotros Diego José Lasso Guamán, Jefferson Ricardo Puchaicela Condoy, con documento de identificación N° 1754074746 y N° 1723027759, manifestamos nuestra voluntad y cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos los autores del trabajo de titulación intitulado EVALUACIÓN DEL RENDIMIENTO DE UN PROTOTIPO SDN (SOFTWARE DEFINED NETWORKING) BAJO EL PROTOCOLO OPENFLOW UTILIZANDO HERRAMIENTAS OPEN SOURCE EN UN ENTORNO VIRTUALIZADO, mismo que ha sido desarrollado para optar por el título de: INGENIEROS DE SISTEMAS, en la Universidad Politécnica Salesiana, quedando la Universidad facultad para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autores nos reservamos los derechos morales de la obra antes citada. En concordancia, suscribimos este documento en el momento que hacemos entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana

……………………………	…………………………….......
Diego José Lasso Guamán	Jefferson Ricardo Puchaicela Condoy
1754074746	1723027759

Quito, febrero del 2021

DECLARATORIA DE COAUTORÍA DEL TUTOR

Yo declaro que bajo mi dirección y asesoría fue desarrollado el Artículo Académico, con el tema: EVALUACIÓN DEL RENDIMIENTO DE UN PROTOTIPO SDN (SOFTWARE DEFINED NETWORKING) BAJO EL PROTOCOLO OPENFLOW UTILIZANDO HERRAMIENTAS OPEN SOURCE EN UN ENTORNO VIRTUALIZADO, realizado por Diego José Lasso Guamán y Jefferson Ricardo Puchaicela Condoy obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerado como trabajo final de titulación.

Quito, febrero 2021

Manuel Rafael Jaya Duche
C.I:1710631035

DEDICATORIA

Dedico este trabajo de titulación a mi familia por ser un soporte en todo momento.

A mis padres Antonio y Amparito, que son mi mayor motivación, inspiración y el pilar fundamental que me ha acompañado a lo largo de mi carrera universitaria a través de su amor, sacrificio, paciencia y apoyo incondicional. A mis hermanos Cris y Oscar quienes han sido mis guías y me han ayudado a llegar a donde hoy estoy a través de su apoyo y motivación. A mis abuelitos por la compañía y ayuda que cada día me brindaron.

Diego José Lasso Guamán

Dedico este documento a mis padres Judith y Jaime mis hermanos Mishell y Javier por apoyarme en toda mi formación académica su amor, paciencia, sacrificio y concejos me motivaron a cumplir mis objetivos y metas. A mis compañeros, amigos con quienes obtuve grandes experiencias y compartimos nuestros conocimientos.

Jefferson Ricardo Puchaicela Condoy

AGRADECIMIENTO

Agradecemos a la Universidad Politécnica Salesiana, institución que nos aportó conocimientos en nuestra formación personal y académica, a nuestro tutor Manuel Rafael Jaya Duche por habernos orientado y motivado a culminar nuestro trabajo, a nuestros amigos y compañeros por las experiencias y momentos agradables compartidos durante nuestra vida universitaria.

Diego Lasso y Jefferson Puchaicela

EVALUACIÓN DEL RENDIMIENTO DE UN PROTOTIPO SDN (SOFTWARE DEFINED NETWORKING) BAJO EL PROTOCOLO OPENFLOW UTILIZANDO HERRAMIENTAS OPEN SOURCE EN UN ENTORNO VIRTUALIZADO

PERFORMANCE EVALUATION OF AN SDN PROTOTYPE (SOFTWARE DEFINED NETWORKING) UNDER THE OPENFLOW PROTOCOL USING OPEN SOURCE TOOLS IN A VIRTUALIZED ENVIRONMENT

Diego José Lasso Guamán[1], Jefferson Ricardo Puchaicela Condoy[1], Manuel Rafael Jaya Duche[2]

Resumen

El presente artículo tiene como objetivo evaluar el desempeño de un prototipo de red definida por software (SDN), el estudio está enfocado a través del uso de herramientas de código abierto para la gestión de red con controladores SDN (OpenDayLight, ONOS y Floodlight) y para la emulación del entorno (Mininet). Además del uso de métricas para el análisis de rendimiento como latencia, conectividad, pérdida de paquetes, intercambio de mensajes, throughput, disponibilidad del controlador SDN y el desempeño en base a servicios de red como streaming, tasa de transferencia en ftp y servicio web. Los principales resultados muestran que el desempeño de la red está relacionado con capacidades del controlador SDN, además que cada uno destaca en diferentes parámetros de rendimiento siendo OpenDayLight el que mejor desempeñó en medidas de QoS, obteniendo un promedio de 0,936 ms con una tendencia a variar por debajo o encima de 0,72 ms y 1,97% en pérdida de paquetes.

Palabras Clave: Redes definidas por software, rendimiento de red, OpenFlow, servicios de red.

Abstract

The aim of this article is to evaluate the performance of a software-defined network prototype (SDN), the study is focused through the use of open-source tools for network management with SDN controllers (OpenDayLight, ONOS and Floodlight) and for environment emulation (Mininet). In addition to the use of metrics for performance analysis such as latency, connectivity, packet loss, message exchange, throughput, SDN controller availability and performance based on network services such as streaming, ftp transfer rate and web service. The main results show that network performance is related to SDN controller capabilities, and that each controller excels in different performance parameters, with OpenDayLight performing best in QoS measures, averaging 0.936 ms with a tendency to vary below or above 0.72 ms and 1.97% in packet loss.

Keywords: Software defined networks, network performance, OpenFlow, network services.

---

[1] Estudiantes de Ingeniería de Sistemas - Universidad Politécnica Salesiana, Egresado - UPS - Sede Quito. Email: dlassog1@est.ups.edu.ec, jpuchaicela@est.ups.edu.ec

[2] Magister en Redes de Información y Conectividad, Ingeniero en Electrónica y Telecomunicaciones, Profesor de Ingeniería de Sistemas - UPS - sede Quito, Email: mjaya@ups.edu.ec

1. Introducción

En la última década, el crecimiento de las redes a nivel global ha sido inevitable por el aumento continuo de usuarios, dispositivos, y nuevas aplicaciones [1]. Ocasionando que las redes tradicionales se compongan de un gran número de dispositivos de interconexión, y éstos, demanden una mayor necesidad de requerimientos como: ancho de banda, agilidad en la red a través de la introducción de nuevos servicios o dispositivos de forma rápida sin alterar el funcionamiento de red, mayor almacenamiento, recursos que la arquitectura de red tradicional no puede satisfacer eficazmente [2].

Esto deriva en problemas de capacidad, escalabilidad, dificultad en la gestión y control centralizado, problemas en el establecimiento de políticas de seguridad, además, de una alta dependencia en proveedores [3]. Para lo cual, se ha desarrollado la tecnología SDN (Redes Definidas por Software), siendo una nueva tecnología que se diseñó para enfrentar los retos actuales y futuros dentro de las redes de telecomunicaciones, especialmente para brindar una solución a los problemas de las redes tradicionales a través de una nueva visión a la administración de las redes de datos [3]. Por lo tanto, implementar SDN permite obtener redes flexibles, adaptables y expandibles donde se reducirán notoriamente los costos de implementación y operación [4]. SDN presenta un gran crecimiento y tendencia a nivel mundial [5], principalmente por todas las características y ventajas que muestran y la relevancia de su nuevo paradigma de red que va tomando mayor fuerza sobre las redes tradicionales. Se espera que a futuro SDN brinde una mayor eficiencia en las redes y que se acople correctamente junto a otros estándares como el 5G y el Internet de las cosas (IoT).

Sin embargo, el carácter aún novedoso de esta arquitectura exige profundizar en un estudio detallado acerca del desempeño y funcionalidad que presentan este nuevo tipo de redes [6].

La conceptualización de SDN donde menciona la separación entre el plano de datos y el plano de control, logrando como resultado que el plano de control funcione totalmente independiente a la vez que sea programable y centralizado. Muchos de los conmutadores tradicionales operan el plano de datos y el plano de control en el mismo dispositivo. Además, el plano de datos tiene como funcionalidad el reenvió de paquetes a diferencia del plano de control que se responsabiliza de controlar el flujo de tráfico, implementando directivas y reglas para toda la red [7].

SDN utiliza el protocolo OpenFlow el cual le permite controlar los flujos, eligiendo las rutas que sigan sus paquetes de un conmutador [8]. Por ende, se incluyó la arquitectura de tres capas de SDN que se presenta a continuación y se la observa en la Figura 1.

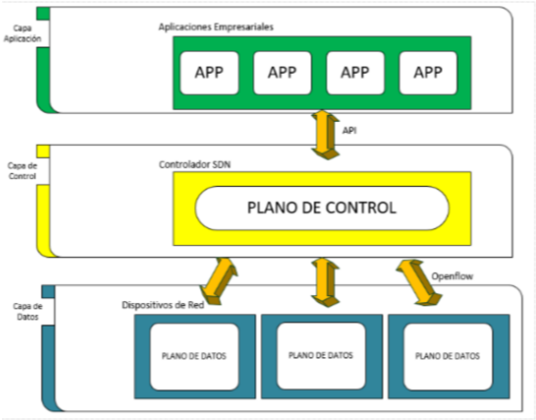


Figura 1. Estructura de tres capas SDN

- **Capa Aplicación:** SDN utiliza la API para la comunicación con diferentes aplicaciones y programas que contengan en esta capa. Estas aplicaciones puedes ser empresariales o de administración de red [9]
- **Capa de Control:** Recolecta toda la información relacionada con la red de los componentes hardware y presenta una vista global de la red a la capa aplicación [7].
- **Capa de Datos:** Contiene a los dispositivos SDN tanto físicos como

virtuales que realizan conmutación y renvió de paquetes, por ejemplo, switches, enrutadores, etc [10].

Las capas se comunican una hacia otra a través de 2 interfaces: Southbound o Interfaz hacia el sur que permite la conexión entre la capa de datos y el plano de control (Controlador SDN) y la Northbound o Interfaz hacia el norte que permite la conexión entre el plano de control hacia las aplicaciones [10], [11], [12].

En consecuencia, el presente documento plantea y evalúa el desempeño de un prototipo de red basado en SDN que tenga los parámetros de una topología personalizada bajo el protocolo OpenFlow con la utilización de servicios de red dentro de su arquitectura tales como Ftp, Web y Video Streaming. Además, se han considerado tres controladores de código abierto que son: OpenDayLight (ODL) [13], Open Network Operating System (ONOS) [14], Floodlight [15]. Estos controladores fueron elegidos en función de su popularidad y sus características como se muestra en la tabla 1, todo esto llevado a cabo dentro de un entorno virtual.

Tabla 1. Principales características de los controladores SDN seleccionados [16].

	ODL	Floodlight	ONOS
Lenguaje Programación	Java	Java	Java
Arquitectura	Distribuida	Centralizada	Distribuida
Protocolo de Southbound	OpenFlow 1.0 - 1.3	OpenFlow 1.0 - 1.3	OpenFlow 1.0 - 1.3
Protocolos de Northbound	REST, XMPP, NETCONF	REST JavaRPC, Quantum	REST, Neutron
Plataformas	Linux, Windows, MacOS	Linux, Windows, MacOS	Linux, Windows, MacOS
Licencia	EPL 1.0	Apache 2.0	Apache 2.0
Documentación	Buena	Buena	Buena
Modularidad	Alta	Regular	Alta
Interface de administración	Web UI, CLI	Web UI	Web UI, CLI

La organización de este documento muestra los siguientes ítems de la siguiente manera. El Ítem 2 proporciona los trabajos relacionados sobre el rendimiento entre los controladores. El Ítem 3 detalla los materiales y describe la metodología realizada para este documento. El Ítem 4 describe los resultados y discusión obtenidos a través de las métricas. El Ítem 5 da a conocer los trabajos futuros y el Ítem 6 concluye el documento.

2. Trabajos Relacionados

K. Arahunashi, S. Neethu, H. V. Ravish Aradhya [7] han realizado una comparación del rendimiento entre los controladores SDN Floodlight, OpenDayLight, Ryu enfocado en topologías por defecto que vienen dentro de Mininet y la relación de los controladores en términos de retardo y rendimiento. Los autores resumieron que al tabular la información el controlador Floodlight es adecuado para evitar menor retardo en comparación a Ryu y OpenDayLight.

Ahmed Hassan, Ahmed Alhassan, Fathia Izzeldean [17] evalúan los controladores Floodlight y OpenDayLight dentro del entorno de emulación Ofnet en términos de generación de flujo, latencia media de configuración y los mensajes de flujo entre controladores. Concluyendo que OpenDayLight maneja flujos estáticos en los paquetes de los mensajes logrando así superar en términos de latencia promedio a Floodlight.

C.Fancy, M.Pushpalatha [18] analizan las características de los controladores POX y Floodlight en términos de retardo y throughput enfocado en las topologías de estrella, lineal, árbol dentro del programa de emulación Mininet. Se concluye que el controlador Floodlight supera al controlador POX en términos de rendimiento especialmente en latencia puesto que llega a ser 31 veces más rápido que POX.

En [19] se realiza el esquema de la arquitectura SDN y se detalla el desempeño de una red definida por software en base a tres parámetros que son latencia, fluctuación y rendimiento, esto se realiza en el protocolo

OpenFlow y con elemento de reenvío genérico programable a través de un controlador concluyendo que las SDN representan un cambio drástico en la arquitectura de redes e implantando hacia las redes de siguiente generación.

L. Mamushiane, A. Lysko, S. Diamini realiza una comparación de los principales controladores SDN de código abierto Onos, Ryu, Floodlight, OpenDayLight en términos de rendimiento neto de controlador con la herramienta llamada Cbench, los autores recomiendan la adopción de OpenDayLight puesto que tiene más funciones en términos de soporte de proveedores de interfaces [20].

A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu y E. C. Popovici [21] realizaron pruebas de rendimiento en los controladores POX, Ryu, ODL y ONOS, considerando la topología árbol para su red y encontraron que ONOS logra un mejor rendimiento, con los anchos de banda que exceden 9 Gbps.

3. Materiales y Métodos

Para la elaboración del artículo, se plantea la metodología descriptiva [22], que detallará el prototipo de red definida por software a utilizar, sus materiales y su respectivo análisis en base a características medibles a través de métricas de red. El equipamiento utilizado es el siguiente:

- Máquina Física HP-8Gb de RAM
- Máquinas virtuales Ubuntu 16.04 de 3Gb RAM
- Controlador SDN OpenDayLight
- Controlador SDN Floodlight
- Controlador SDN ONOS
- Emulador Mininet.

La red se desarrollará y ejecutará a través del programa Mininet y el funcionamiento del mismo será dirigido a través de los controladores OpenDayLight, Floodlight y ONOS puesto que en la investigación [4] se indican que los 3 controladores SDN son los más ocupados en la actualidad. Posteriormente se realiza su respectiva evaluación de rendimiento. El esquema del controlador – emulador (Mininet) es el siguiente:

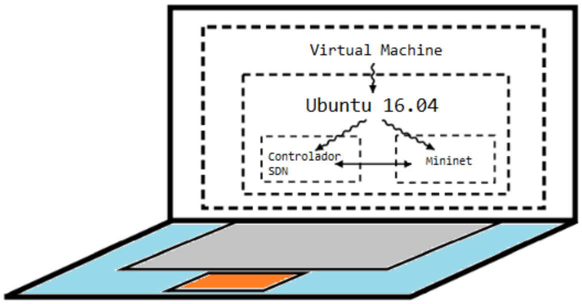


Figura 2. Esquema del entorno a utilizar

3.1. Escenario

Para el desarrollo de la arquitectura de red del prototipo se ha utilizado el modelo jerárquico de tres capas: núcleo, distribución y acceso. Pues, el tener la división de estas capas hace que cada capa cumpla con funciones específicas, simplificando el diseño de la red [23]. Dado que esta arquitectura se administra y expande con gran facilidad teniendo enlaces redundantes entre sus dispositivos. Donde se describe cada capa a continuación y se observa en la figura 3.

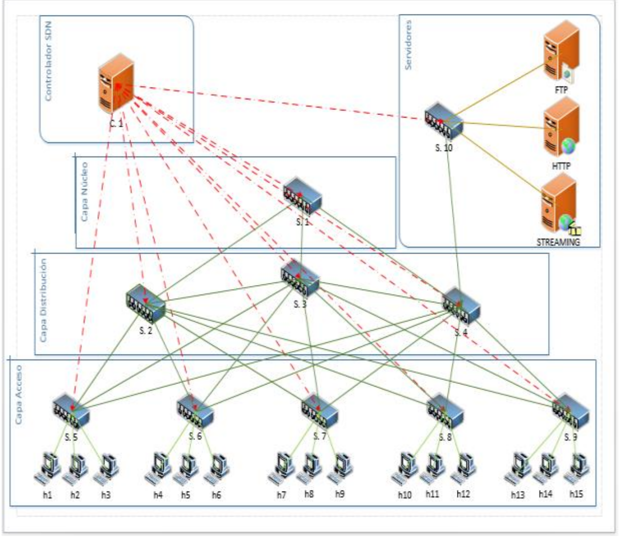


Figura 3. Topología de red utilizada

- **Capa de acceso:** Se encuentran los puntos finales y los usuarios que tienen acceso directo a la red.
- **Capa de distribución:** Agrega capas de acceso proporcionando conectividad a los servicios que se maneje.
- **Capa de núcleo:** Conocido como el backbone de alta velocidad de la red e interconecta a la capa de distribución, pues debe estar disponible y redundante para la red.

Con respecto a los servicios FTP, HTTP y Streaming conectados a la red del prototipo se deben levantar dentro del sistema operativo para que la virtualización basada en procesos que utiliza Mininet pueda heredar los servicios en los hosts emulados [24].

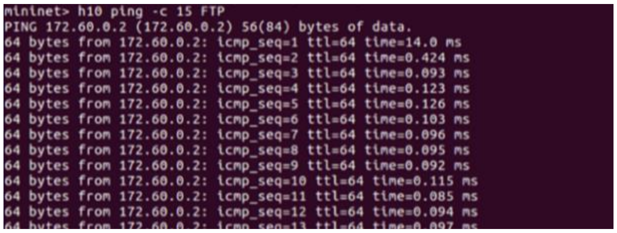
En el caso del servicio de Streaming se instala la aplicación VLC, misma que se utilizará para la emisión de videos con el protocolo de transporte en tiempo real.

3.2. Evaluación del Desempeño

Para la evaluación del prototipo de red indicado, se obtuvieron como referencia las métricas de: latencia, tasa de transferencia de datos, conectividad, pérdida de paquetes, intercambio de mensajes, throughput y disponibilidad de controlador. La evaluación se realizó ejecutando un controlador a la vez.

3.2.1. Latencia

Para medir la latencia se utilizó el comando ping, el cual permite crear solicitudes ICMP entre dos hosts, especificando el host de origen y el host de destino [7].



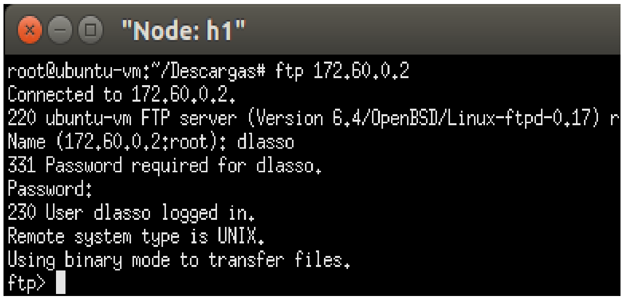
```
mininet> h10 ping -c 15 FTP
PING 172.60.0.2 (172.60.0.2) 56(84) bytes of data.
64 bytes from 172.60.0.2: icmp_seq=1 ttl=64 time=14.0 ms
64 bytes from 172.60.0.2: icmp_seq=2 ttl=64 time=0.424 ms
64 bytes from 172.60.0.2: icmp_seq=3 ttl=64 time=0.093 ms
64 bytes from 172.60.0.2: icmp_seq=4 ttl=64 time=0.123 ms
64 bytes from 172.60.0.2: icmp_seq=5 ttl=64 time=0.126 ms
64 bytes from 172.60.0.2: icmp_seq=6 ttl=64 time=0.103 ms
64 bytes from 172.60.0.2: icmp_seq=7 ttl=64 time=0.096 ms
64 bytes from 172.60.0.2: icmp_seq=8 ttl=64 time=0.095 ms
64 bytes from 172.60.0.2: icmp_seq=9 ttl=64 time=0.092 ms
64 bytes from 172.60.0.2: icmp_seq=10 ttl=64 time=0.115 ms
64 bytes from 172.60.0.2: icmp_seq=11 ttl=64 time=0.085 ms
64 bytes from 172.60.0.2: icmp_seq=12 ttl=64 time=0.094 ms
64 bytes from 172.60.0.2: icmp_seq=13 ttl=64 time=0.097 ms
```

Figura 4. Ejecución del comando ping

En el prototipo presentado se enviaron 50 solicitudes ping de un tamaño de 65507 bytes (tamaño máximo permitido por Ubuntu) y el valor por defecto de 64 bytes, esto a través del comando: ping <dir_origen> -c 50 -s 65507 <dir_origen>.

3.2.2. Tasa de transferencia de datos

Desde la terminal de Mininet se ejecuta el comando xterm <nombre_host> para abrir la línea de comandos de cada host emulado, una vez abierto la terminal se procede a realizar una conexión entre el servidor FTP y un host cliente dentro de la topología a través del comando FTP <dir_ip_servidor>.



```
root@ubuntu-vm:~/Descargas# ftp 172.60.0.2
Connected to 172.60.0.2.
220 ubuntu-vm FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) r
Name (172.60.0.2:root): dlasso
331 Password required for dlasso.
Password:
230 User dlasso logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> 
```

Figura 5. Inicio del servicio FTP

Una vez realizada la conexión FTP correctamente, se procede a enviar 3 archivos de diferente tamaño (1, 10,100 Megabytes) entre el servidor y el cliente.

3.2.3. Conectividad y Pérdida de paquetes

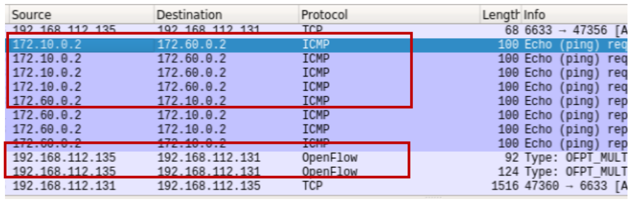
Para determinar la cantidad de paquetes perdidos se utiliza la herramienta pingall que viene por defecto dentro de Mininet y permite realizar una verificación de conexión entre todos los nodos existentes en la red a través de un ping [25]. Este parámetro se mide principalmente cuando inicia la red, es decir, cuando se instalan los flujos de red dentro de cada switch y cuando la conexión ya es establecida correctamente.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 FTP HTTP STREAM
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 FTP HTTP STREAM
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 FTP HTTP STREAM
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 FTP X X
h5 -> X X h3 X X X X X X X X X X X X X X
h6 -> X X X X X X X X X X X X X X X X X
h7 -> X X h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 h15 FTP HTTP STREAM
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15 FTP HTTP STREAM
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15 FTP HTTP STREAM
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15 FTP HTTP STREAM
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15 FTP HTTP STREAM
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15 FTP HTTP STREAM
h13 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15 FTP HTTP STREAM
h14 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15 FTP HTTP STREAM
h15 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 FTP HTTP STREAM
FTP -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 HTTP STREAM
HTTP -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 FTP STREAM
STREAM -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 FTP HTTP
*** Results: 12% dropped (269/306 received)
```

Figura 6. Ejecución del comando pingall

3.2.4. Intercambio de mensajes

Se utiliza la herramienta de análisis wireshark, para el filtro de paquetes, especialmente para aquellos de tipo OpenFlow para la conectividad controlador – red y los paquetes ICMP para determinar la conectividad entre host – host.

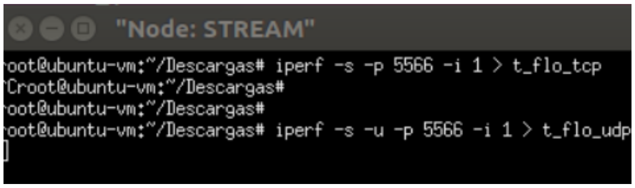


Source	Destination	Protocol	Length	Info
192.168.112.135	192.168.112.131	TCP	68	6633 → 47356 [A
172.10.0.2	172.60.0.2	ICMP	100	Echo (ping) req
172.10.0.2	172.60.0.2	ICMP	100	Echo (ping) req
172.10.0.2	172.60.0.2	ICMP	100	Echo (ping) req
172.10.0.2	172.60.0.2	ICMP	100	Echo (ping) req
172.60.0.2	172.10.0.2	ICMP	100	Echo (ping) rep
172.60.0.2	172.10.0.2	ICMP	100	Echo (ping) rep
172.60.0.2	172.10.0.2	ICMP	100	Echo (ping) rep
172.60.0.2	172.10.0.2	ICMP	100	Echo (ping) rep
192.168.112.135	192.168.112.131	OpenFlow	92	Type: OFPT_MULT
192.168.112.135	192.168.112.131	OpenFlow	124	Type: OFPT_MULT
192.168.112.131	192.168.112.135	TCP	1516	47360 → 6633 [A

Figura 7. Visualización de protocolos ICMP y OpenFlow

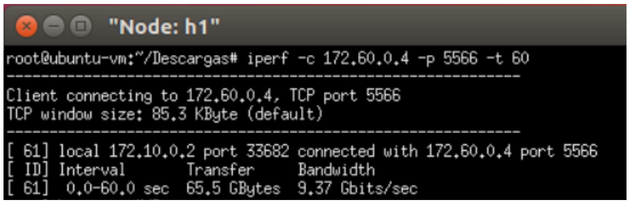
3.2.5. Throughput

Para determinar el throughput se utiliza la herramienta iperf permitiendo determinar la cantidad de flujo tanto para TCP como UDP, esto realiza a través del comando iperf -s -p 5566 -i para el servidor y el comando iperf –c <ip_servidor> -p 5566 -t para el cliente en donde -s significa que se habilita el modo servidor, -i el intervalo de tiempo en segundos de medición en este caso 1 segundo, -p para especificar bajo que puerto se realizara la conexión, -c para habilitar el modo cliente y -t para determinar el tiempo de duración de la prueba.



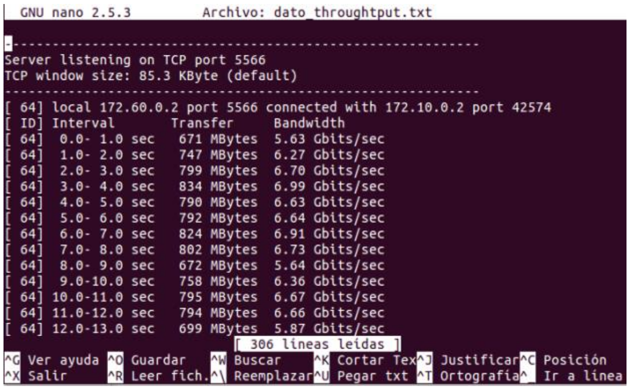
```
"Node: STREAM"
oot@ubuntu-vm:~/Descargas# iperf -s -p 5566 -i 1 > t_flo_tcp
Croot@ubuntu-vm:~/Descargas#
oot@ubuntu-vm:~/Descargas#
oot@ubuntu-vm:~/Descargas# iperf -s -u -p 5566 -i 1 > t_flo_udp
]
```

Figura 8. Ejecución del comando Iperf en la terminal del servidor



```
"Node: h1"
root@ubuntu-vm:~/Descargas# iperf -c 172.60.0.4 -p 5566 -t 60
------------------------------------------------------------
Client connecting to 172.60.0.4, TCP port 5566
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 61] local 172.10.0.2 port 33682 connected with 172.60.0.4 port 5566
[ ID] Interval       Transfer     Bandwidth
[ 61]  0.0-60.0 sec  65.5 GBytes  9.37 Gbits/sec
```

Figura 9. Ejecución del comando Iperf en la terminal del cliente



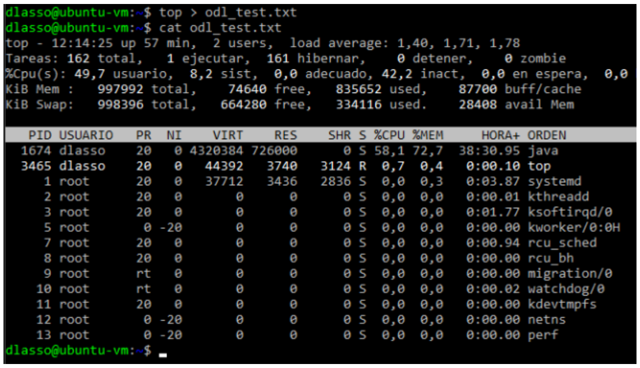
```
  GNU nano 2.5.3         Archivo: dato_throughtput.txt

------------------------------------------------------------
Server listening on TCP port 5566
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 64] local 172.60.0.2 port 5566 connected with 172.10.0.2 port 42574
[ ID] Interval       Transfer     Bandwidth
[ 64]  0.0- 1.0 sec   671 MBytes  5.63 Gbits/sec
[ 64]  1.0- 2.0 sec   747 MBytes  6.27 Gbits/sec
[ 64]  2.0- 3.0 sec   799 MBytes  6.70 Gbits/sec
[ 64]  3.0- 4.0 sec   834 MBytes  6.99 Gbits/sec
[ 64]  4.0- 5.0 sec   790 MBytes  6.63 Gbits/sec
[ 64]  5.0- 6.0 sec   792 MBytes  6.64 Gbits/sec
[ 64]  6.0- 7.0 sec   824 MBytes  6.91 Gbits/sec
[ 64]  7.0- 8.0 sec   802 MBytes  6.73 Gbits/sec
[ 64]  8.0- 9.0 sec   672 MBytes  5.64 Gbits/sec
[ 64]  9.0-10.0 sec   758 MBytes  6.36 Gbits/sec
[ 64] 10.0-11.0 sec   795 MBytes  6.67 Gbits/sec
[ 64] 11.0-12.0 sec   794 MBytes  6.66 Gbits/sec
[ 64] 12.0-13.0 sec   699 MBytes  5.87 Gbits/sec
                                  [ 306 líneas leídas ]
^G Ver ayuda ^O Guardar    ^W Buscar     ^K Cortar Tex^J Justificar^C Posición
^X Salir     ^R Leer fich.^\ Reemplazar^U Pegar txt ^T Ortografía^_ Ir a línea
```

Figura 10. Datos de Throughput almacenado en el archivo dato_throughput.txt

A través del uso de tuberías, el lenguaje awk y Shell de Linux, el comando explicado en el párrafo anterior se puede guardar dentro de un archivo para su obtención de datos relevantes a través del comando: cat ‘dato_throughput’ | grep sec | head -15 | tr - " " | awk '{print $4, $8}' > datos.txt

3.2.6. Disponibilidad de controlador

Se utiliza la herramienta top para ver un resumen cada 3 segundos del estado del sistema es decir los procesos que se ejecutan [26], en este caso permitió observar el tamaño de los procesos que ocupa cada controlador mientras la red está activa y ejecutándose.



```
dlasso@ubuntu-vm:~$ top > odl_test.txt
dlasso@ubuntu-vm:~$ cat odl_test.txt
top - 12:14:25 up 57 min,  2 users,  load average: 1,40, 1,71, 1,78
Tareas: 162 total,   1 ejecutar,  161 hibernar,    0 detener,    0 zombie
%Cpu(s): 49,7 usuario,  8,2 sist,  0,0 adecuado, 42,2 inact,  0,0 en espera,  0,0
KiB Mem :   997992 total,    74640 free,   835652 used,    87700 buff/cache
KiB Swap:   998396 total,   664280 free,   334116 used.    28408 avail Mem

  PID USUARIO   PR  NI    VIRT    RES    SHR S %CPU %MEM     HORA+ ORDEN
 1674 dlasso    20   0 4320384 726000      0 S 58,1 72,7  38:30.95 java
 3465 dlasso    20   0   44392   3740   3124 R  0,7  0,4   0:00.10 top
    1 root      20   0   37712   3436   2836 S  0,0  0,3   0:03.87 systemd
    2 root      20   0       0      0      0 S  0,0  0,0   0:00.01 kthreadd
    3 root      20   0       0      0      0 S  0,0  0,0   0:01.77 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S  0,0  0,0   0:00.94 rcu_sched
    8 root      20   0       0      0      0 S  0,0  0,0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S  0,0  0,0   0:00.00 migration/0
   10 root      rt   0       0      0      0 S  0,0  0,0   0:00.02 watchdog/0
   11 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kdevtmpfs
   12 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 netns
   13 root       0 -20       0      0      0 S  0,0  0,0   0:00.00 perf
dlasso@ubuntu-vm:~$ _
```

Figura 11. Visualización de disponibilidad del controlador

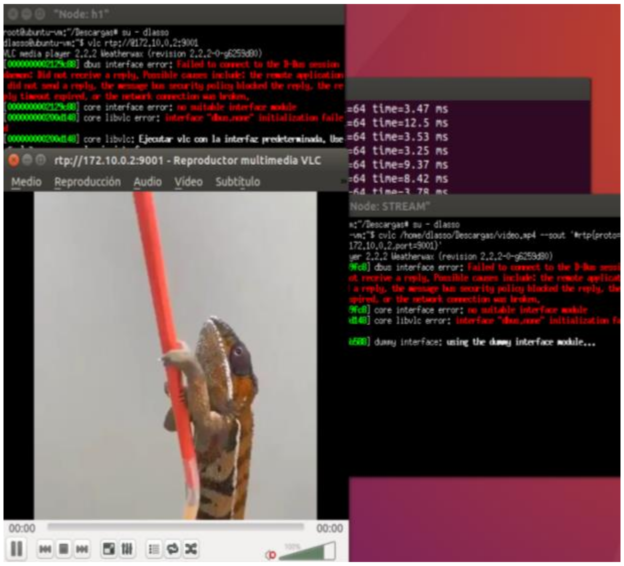
3.2.7. Streaming

Para la transmisión de streaming se utilizó el programa VLC y el xterm proporcionado por Mininet, dentro de los clientes se utiliza el comando:

- su -<usuario>: para cambiar de root a usuario normal y poder ejecutar vlc con la ip de la topología
- vlc rtp://@ip_destino:9001 para indicar la dirección ip que transmite el stream

Para el servidor stream se abre la terminal y se ejecuta los siguientes comandos:

- su -<usuario>: para cambiar de root a usuario normal y poder ejecutar vlc con la ip de la topología
- cvlc /.../nombre_archivo.mp4 --sout '#rtp{proto=udp,mux=ts,dst=ip_destino,port=9001}' en donde se especifica el protocolo a ocupar, la dirección de emisión y el puerto de transmisión.



```
root@ubuntu-vm:~/Descargas# su - dlasso
dlasso@ubuntu-vm:~$ vlc rtp://@172.10.0.2:9001
VLC media player 2.2.2 Weatherwax (revision 2.2.2-0-g6259d80)
[00000000002129c88] dbus interface error: Failed to connect to the D-Bus session
daemon: Did not receive a reply. Possible causes include: the remote application
 did not send a reply, the message bus security policy blocked the reply, the re
ply timeout expired, or the network connection was broken.
[00000000002129c88] core interface error: no suitable interface module
[0000000000200d148] core libvlc error: interface "dbus,none" initialization faile
d
[0000000000200d148] core libvlc: Ejecutar vlc con la interfaz predeterminada. Use
```

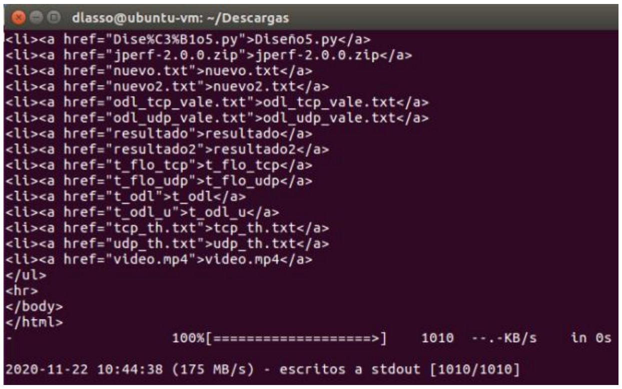
```
=64 time=3.47 ms
=64 time=12.5 ms
=64 time=3.53 ms
=64 time=3.25 ms
=64 time=9.37 ms
=64 time=8.42 ms
```

```
m:~/Descargas# su - dlasso
-vm:~$ cvlc /home/dlasso/Descargas/video.mp4 --sout '#rtp{proto=
172.10.0.2,port=9001}'
yer 2.2.2 Weatherwax (revision 2.2.2-0-g6259d80)
9fc8] dbus interface error: Failed to connect to the D-Bus sessi
ot receive a reply. Possible causes include: the remote applicat
 a reply, the message bus security policy blocked the reply, the
xpired, or the network connection was broken.
9fc8] core interface error: no suitable interface module
d148] core libvlc error: interface "dbus,none" initialization fa

6580] dummy interface: using the dummy interface module...
```

Figura 12. Inicio de servicio Streaming

3.2.8. Servicio Web

Para el servicio HTTP se optó por utilizar el módulo SimpleHTTPServer que viene por defecto dentro de la librería de Python de Ubuntu y a su vez permite la ejecución de un servicio de http simple para realizar pruebas a través de dirección local. Y a través de wireshark se analizaron los paquetes con tiempos de respuesta de HTTP.

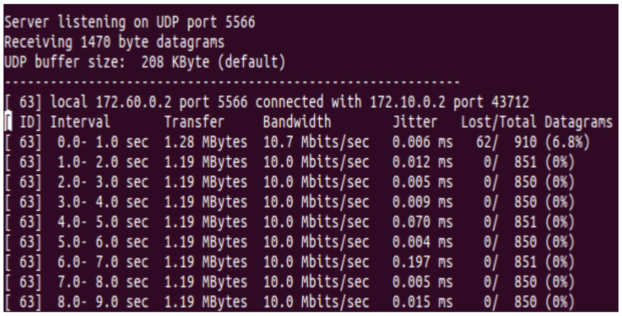


```
<li><a href="Dise%C3%B1o5.py">Diseño5.py</a>
<li><a href="jperf-2.0.0.zip">jperf-2.0.0.zip</a>
<li><a href="nuevo.txt">nuevo.txt</a>
<li><a href="nuevo2.txt">nuevo2.txt</a>
<li><a href="odl_tcp_vale.txt">odl_tcp_vale.txt</a>
<li><a href="odl_udp_vale.txt">odl_udp_vale.txt</a>
<li><a href="resultado">resultado</a>
<li><a href="resultado2">resultado2</a>
<li><a href="t_flo_tcp">t_flo_tcp</a>
<li><a href="t_flo_udp">t_flo_udp</a>
<li><a href="t_odl">t_odl</a>
<li><a href="t_odl_u">t_odl_u</a>
<li><a href="tcp_th.txt">tcp_th.txt</a>
<li><a href="udp_th.txt">udp_th.txt</a>
<li><a href="video.mp4">video.mp4</a>
</ul>
<hr>
</body>
</html>
-                     100%[===================>]    1010  --.-KB/s    in 0s

2020-11-22 10:44:38 (175 MB/s) - escritos a stdout [1010/1010]
```

Figura 13. Inicio de servicio Web

3.2.9. Jitter

Para el jitter (variación en la latencia) igualmente se utilizó la herramienta iperf ejecutando una prueba con UDP con el comando iperf -c <ip_servidor> -u <tráfico UDP>.



```
Server listening on UDP port 5566
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 63] local 172.60.0.2 port 5566 connected with 172.10.0.2 port 43712
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 63]  0.0- 1.0 sec  1.28 MBytes  10.7 Mbits/sec   0.006 ms   62/  910 (6.8%)
[ 63]  1.0- 2.0 sec  1.19 MBytes  10.0 Mbits/sec   0.012 ms    0/  851 (0%)
[ 63]  2.0- 3.0 sec  1.19 MBytes  10.0 Mbits/sec   0.005 ms    0/  850 (0%)
[ 63]  3.0- 4.0 sec  1.19 MBytes  10.0 Mbits/sec   0.009 ms    0/  850 (0%)
[ 63]  4.0- 5.0 sec  1.19 MBytes  10.0 Mbits/sec   0.070 ms    0/  851 (0%)
[ 63]  5.0- 6.0 sec  1.19 MBytes  10.0 Mbits/sec   0.004 ms    0/  850 (0%)
[ 63]  6.0- 7.0 sec  1.19 MBytes  10.0 Mbits/sec   0.197 ms    0/  851 (0%)
[ 63]  7.0- 8.0 sec  1.19 MBytes  10.0 Mbits/sec   0.005 ms    0/  850 (0%)
[ 63]  8.0- 9.0 sec  1.19 MBytes  10.0 Mbits/sec   0.015 ms    0/  850 (0%)
```

Figura 14. Jitter obtenido a través del tráfico UDP.

4. Resultados y Discusión

En base a la topología de red indicada y las configuraciones mencionadas, se procede a realizar la evaluación del prototipo de red, obteniendo así los siguientes resultados.

La tabla 2 y tabla 3 proporciona valores de tiempo de un ping entre los distintos puntos de la red teniendo en cuenta dos tamaños de bytes un máximo y un mínimo tanto para el protocolo IPv4 e IPv6.

.

Tabla 2: Datos de latencia de topología en protocolo IPv4

Controlador	Tamaño [bytes]	T.Min [ms]	T.Max [ms]	AVG [ms]
ODL	65507	3,622	52,056	14,629
	56	0,433	10,254	0,936
Floodlight	65507	0,307	245,428	16,838
	56	0,084	21,805	1,573
ONOS	65507	0,601	252,484	20,870
	56	0,083	39,998	1,366

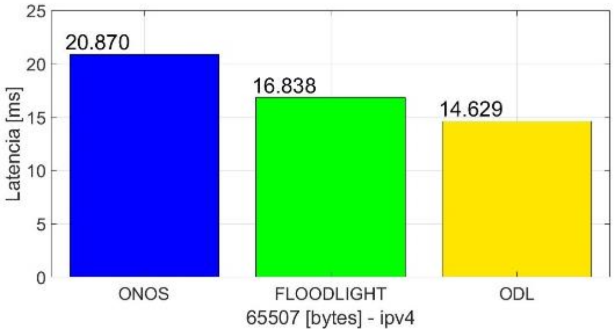


Figura 15. Comparación de latencia en controladores SDN bajo direccionamiento ipv4

Teniendo como mejor resultado de latencia en IPv4 el controlador ODL proporcionando a la topología un 29,90% de tiempo menor en comparación al controlador ONOS y 13,12% al controlador Floodlight en paquetes de 65507 bytes. Además de un 31,48% menor a ONOS y 40,5% menor a Floodlight en paquetes de 56 bytes. Además ODL presento una desviación estándar de 0,72 milisegundos.

Tabla 3: Datos latencia de topología en protocolo IPv6

Controlador	Tamaño [bytes]	T.Min [ms]	T. Max [ms]	AVG [ms]
ODL ipv6	65507	3,940	35,636	12,549
	56	0,462	1,952	0,805
Floodlight ipv6	65507	0,496	569,346	20,637
	56	0,104	30,352	2,180
ONOS ipv6	65507	0,746	6,935	13,921
	56	0,092	73,887	1,610

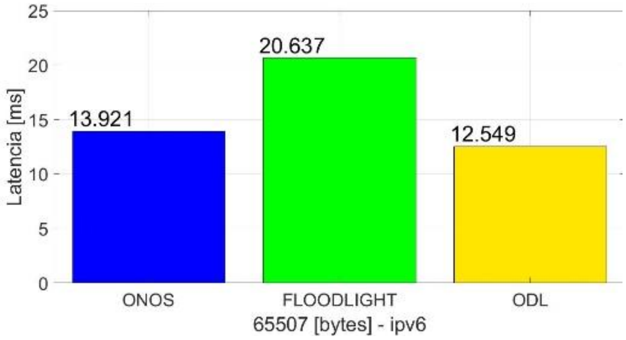


Figura 16. Comparación de latencia en controladores SDN bajo direccionamiento ipv6

Dentro del protocolo IPv6 el controlador ODL se mantiene como el más rápido en términos de latencia siendo 39% menor que Floodlight y 10% menor que ONOS en paquetes de 65507 bytes, además de 63% menor a ONOS y 50% menor a Floodlight en paquetes de 56 bytes.

Para el caso de la transferencia de datos en el servidor FTP se obtiene la información de la tabla 3, donde (V) es la velocidad de transmisión y (T) es el tiempo de envío de archivo, expresado en [MB/s] y [seg] respectivamente.

Tabla 4: Tiempo y velocidad de envío de archivos

	ODL		Floodlight		ONOS	
	V	T	V	T	V	T
Archivo 1	5,823	0,17	9,312	0,11	29,128	0,03
Archivo 2	6,982	1,37	48,871	0,15	23,378	0,43
Archivo 3	6,305	15,07	60,251	0,79	29,468	3,39

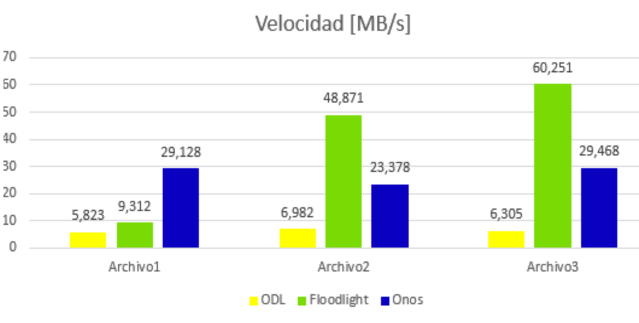


Figura 17. Comparación de velocidad en controladores SDN

Se obtuvo un desempeño notable del controlador Floodlight permitiendo mayores velocidades de transmisión de archivos grandes con un promedio 39,478 MB/s equivalente a un 83,86% más que ODL y 30,79% que ONOS, sin embargo, ONOS logró una mayor velocidad en cuanto a archivos ligeros.

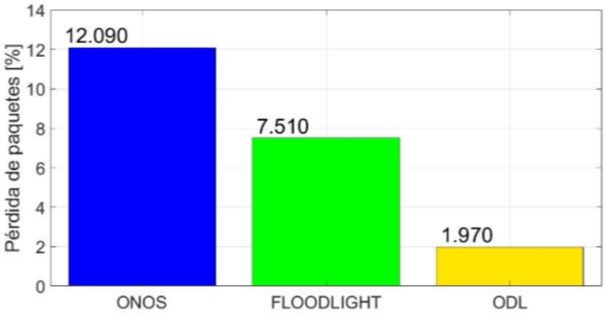


Figura 18. Comparación en (%) de pérdida de paquetes en controladores SDN

Referente a la pérdida de paquetes el controlador ODL fue el que menor pérdida presentó durante la instalación de flujo de datos con un 1,97%, seguido del controlador Floodlight con un 7,51% y ONOS con 12,09% de un total de 306 paquetes enviados, como se indica en la tabla 5.

Tabla 5: Conectividad y pérdida de paquetes

Controlador	1era conexión (%)	Conexión establecida (%)
ODL	6	0
Floodlight	23	1
ONOS	37	0

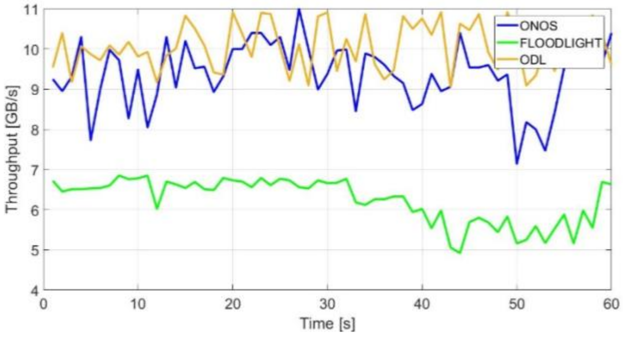


Figura 19. Comparación de throughput en controladores SDN para tráfico TCP

En el throughput con TCP el controlador ONOS presentó un promedio de 9,39 Gbps con una desviación estándar de 0,79 Gbps, el controlador Floodlight presento un promedio de 6,24 Gbps con una desviación de 0,95 Gbps y ODL presento 10,08 Gbps con desviación estándar de 0,59 Gbps. El mejor desempeño se realizó a través del controlador ODL con un aumento de 62% a comparación de Floodlight y 7,35% más que ONOS.

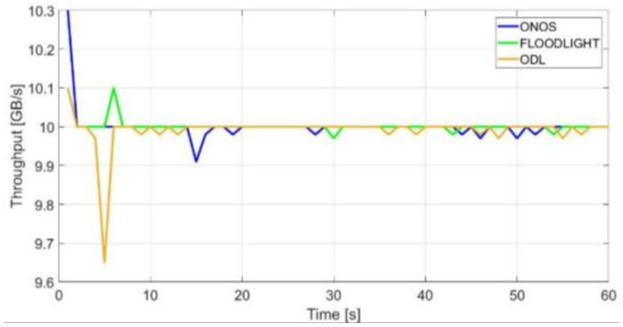


Figura 20. Comparación de throughput en controladores SDN para tráfico UDP

En el throughput con UDP los tres controladores se mantuvieron en un promedio de 10 Gbps y una desviación estándar de 0,041 Gbps para ONOS, 0,019 Gbps para Floodlight y 0,047 Gbps para ODL.

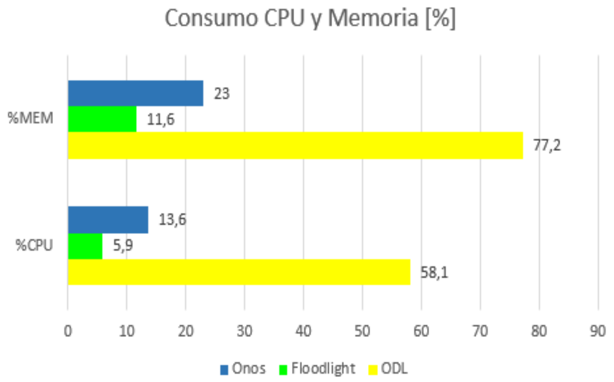


Figura 21. Comparación en (%) de disponibilidad en controladores SDN

Referente a la disponibilidad del controlador enfocado en la red para la topología presentada el controlador que mejor tuvo desempeño fue el Floodlight ocupando un 5,9% de la CPU total y un 11,6% de la memoria total.

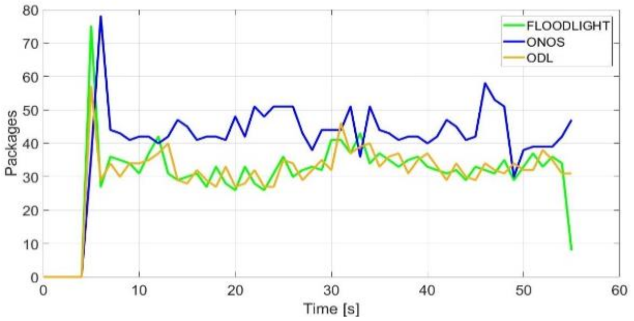


Figura 22. Comparación de controladores SDN con el uso del servicio streaming

Para el servicio de streaming el controlador que mejor se desempeñó dentro de la topología fue el controlador ONOS obteniendo una media de 40,43 paquetes por segundo dentro de la transmisión con una tendencia a variar por debajo o encima de 14,40 paquetes, incrementando un 33% en comparación de paquetes con ODL y Floodlight.

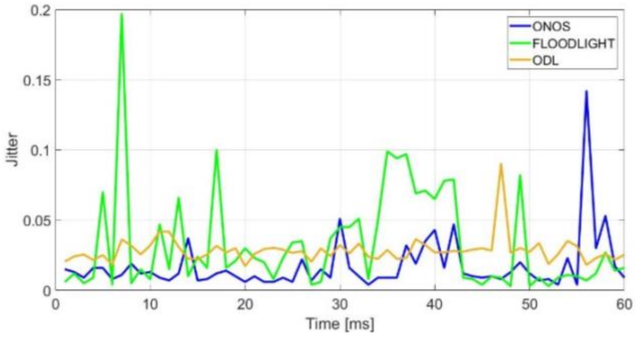


Figura 23. Comparación del jitter en controladores SDN

El jitter del controlador ONOS presento una diferencia de 0,01545 milisegundos con respecto a Floodlight y 0,017 milisegundos con respecto a ODL y con una desviación estándar de 0,020 milisegundos para ONOS, 0,036 milisegundos para Floodlight y 0,010 milisegundos para ODL.

Tabla 6: Tiempo de respuesta en el Servicio Web

Controlador	Tiempo de Respuesta [ms]
OpenDayLight	8,345
Floodlight	10,230
ONOS	11,586

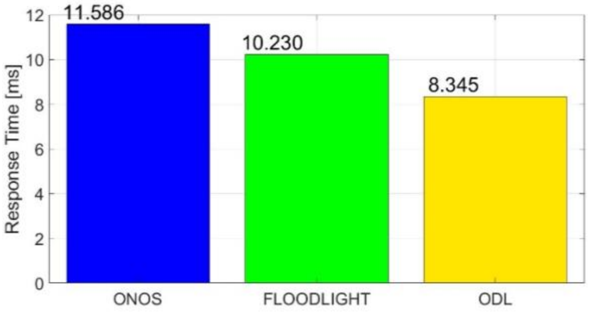


Figura 24. Comparación en (%) de tiempo de respuesta en controladores SDN

Para el servicio web se enfocó en los tiempos de respuesta de cada controlador a través de la herramienta wireshark obteniendo así que el controlador ODL presentó un mejor tiempo de respuesta con 18,43% menor a Floodlight y 27,97% menor a ONOS.

5. Trabajos futuros

En líneas futuras de investigación se puede realizar la evaluación del rendimiento de una red SDN enfocada en un escenario IoT (Internet de las cosas) en tecnologías como 6LoWPAN, con el uso de controladores SDN y la herramienta Mininet-IoT para determinar los beneficios que producen ambos entornos tecnológicos.

6. Conclusiones

El desempeño de una red SDN tiene una relación con los recursos computacionales como el CPU y la memoria que tiene asignado el controlador de red, esto se puede observar en la figura N° 19 el controlador OpenDayLight generó un mayor consumo de recursos de la máquina del controlador consumiendo un promedio de 87,86% el CPU y un 67,27% más de memoria que los otros controladores, ocasionando que presente mejores resultados en los parámetros principales de QoS con 0,936 milisegundos en términos de latencia con tamaños de paquetes predeterminados de 64 bytes y una pérdida inicial de paquetes de 1,97% como se muestra en las tablas 2, 3 y 5.

La tecnología SDN va tomando mayor fuerza y crecimiento a través de los años en las redes de comunicaciones de datos por ende es necesario saber que controlador conlleva un mejor rendimiento para las diferentes topologías de red, en la presente investigación con el prototipo de red definida por software se observa que cada controlador a pesar de que se basan en la misma arquitectura y funcionamiento destacan en diferentes parámetros de rendimiento propuestos, sin embargo, en línea general el controlador OpenDayLight presenta un mejor desempeño dentro de la red por lo tanto puede ser utilizado dentro de topologías de redes similares al prototipo indicado en este documento.

Servicios tales como FTP, WEB y STREAMING son unos de los tantos servicios para poder determinar el comportamiento en el rendimiento de redes SDN pues, el prototipo utilizó un modelo jerárquico de tres capas para tener un modelo de red escalable donde se le atribuyó estos servicios. Además, una de las ventajas que tiene el uso de un modelo jerárquico de tres capas es la relación costo/beneficio y que al combinarlo con SDN hace que se dé acogida al uso de infraestructuras de software de libre distribución para quienes lo apliquen en sus compañías.

En las pruebas realizadas con direccionamiento IPv4 e IPv6 y el uso de los tres controladores SDN ejecutados uno a la vez, se manifestó que, aunque IPv6 tiene un encabezado más grande a diferencia de IPv4 que tiene menos campos que procesar, sus porcentajes en las métricas del prototipo en los distintos controladores hace conocer el rendimiento de éstos protocolos, donde en ipv6 su latencia fue de 12.549 [ms], mientras que en ipv4 su latencia fue de 14,629 [ms] para enviar paquetes con un tamaño de 65507 bytes, resultado a que IPv6 combinado con SDN tengan un 14,22% menor a ipv4 y sea considerado para futuras implementaciones.

Los resultados conseguidos dentro de esta investigación permitieron mostrar el potencial de las redes definidas por software especialmente con una interacción de servicios de red es decir los diferentes tiempos y velocidades que los controladores de red emplean para el envío y recepción de archivos con FTP, cantidad de paquetes enviados en la reproducción de video STREAMING y tiempos de respuesta en servicio WEB además de la importancia que tienen las herramientas de código abierto (Open Source) con este nuevo paradigma de redes que a futuro crearán una nueva generación de redes modernas.

7. Referencias

[1] N. Y. Serrato Losada, M. A. Barrera Perez y H. Vacca Gonzales, «OPENFLOW EN LAS REDES DEFINIDAS POR SOFTWARE (SDN),» *Congreso Internacional de Electrónica, Control y Telecomunicaciones, Tecnología para el Progreso y la Reconciliación (2018),* 2018.

[2] J. C. Chico, D. Mejía y I. Bernal, «Implementación de un Prototipo de una Red Definida por Software (SDN) Empleando una Solución Basada en Hardware,» *XXV Jornadas en Ingeniería Eléctrica y Electrónica*, pp. 356-365, 2014.

[3] C. L. Dueñas Santos, H. Cruz-Enriquez y Y. A. Marín Muro, «SDN Network vs. Traditional Network,» *8VA CONFERENCIA CIENTÍFICA INTERNACIONAL DE LA UNIVERSIDAD DE HOLGUÍN,* pp. 1-7, 2017.

[4] J. E. Salinas Santiago, C. A. Sánchez Venegas, J. C. Herrera Velásquez y C. Santiago, «SDN Implementación de redes de cómputo virtuales,» *LACCEI International Multi-Conference for Engineering, Education, and Technology*, pp. 1-8, 2019.

[5] M. J. O. Brito, Artist, *Características de las Redes Definidas por Software (SDN) para su Implementación en el Ecuador.* [Art]. Universidad Católica de Santiago de Guayaquil, 2018.

[6] L. Gámez Picó, C. Anías Calderón y S. Ballester Macías, «Evaluación de desempeño y configuraciones de las SDN mediante la simulación,» *Tono Revista Técnica de la Empresa de Telecomunicaciones de Cuba S.A.*, pp. 28-32, 2016.

[7] A. K. Arahunashi, S. Neethu y H. V. R. Aradhya, «Performance Analysis of Various SDN Controllers In Mininet Emulator,» *International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT-2019*, pp. 752-756, 2019.

[8] A. K. B. R. Adrian Lara, «Network Innovation using OpenFlow: A Survey,» *IEEE*, vol. 16, nº 1, pp. 493-512, 2014.

[9] C. B. T. C. Paul Goransson, Software Defined Networks, 2nd Edition ed., 2016.

[10] M. P. Doan Hoang, «On software-defined networking and the design of SDN controllers,» pp. 1-3, 2015.

[11] S. Azodolmolky, Software Defined Networking with OpenFlow, BIRMINGHAM - MUMBAI: Packt Publishing Ltd., 2013.

[12] P. Goransson y C. Black, Software Defined Networks : A Comprehensive Approach, Waltham: Elsevier Science & Technology, 2014.

[13] OpenDayLight Project, «OpenDaylight: A Linux Foundation Collaborative Project.,» [En línea]. Available: www.opendaylight.org/.

[14] Open Networking Foundation ONF, «ONOS,» 2017. [En línea]. Available: www.opennetworking.org/onos/.

[15] Project Floodlight, «Floodlight Controller,» 2014. [En línea]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview.

[16] L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du y M. Guizani, «SDN Controllers: Benchmarking & Performance Evaluation,» *arXiv preprint arXiv:1902.04491*, 2019.

[17] A. H. M. Hassan, A. M. Alhassan y F. Izzeldean, «Performance Evaluation of SDN Controllers in Ofnet Emulation Environment,» *International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pp. 1-6, 2019.

[18] F. C y P. M, «Performance Evaluation of SDN controllers POX and Floodlight in Mininet Emulation Environment,» *Proceedings of the International Conference on Intelligent Sustainable Systems (ICISS)*, pp. 696-699, 2017.

[19] A. Garg, V. Saini, M. Imran y M. Qadeer, «Performance Analysis of Software Defined Networks,» *International Conference on Computational Intelligence and Communication Networks* , vol. 9no, pp. 58-61, 2017.

[20] L. Mamushiane, A. Lysko y S. Dlamini, «A comparative evaluation of the performance of popular SDN controllers,» *2018 Wireless Days (WD)*, pp. 54-59, 2018.

[21] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu y E. C. Popovici, «A comparison between several Software Defined Networking controllers,» *IEEE*, pp. 223-226, 2015.

[22] R. Hernández Sampieri, C. Fernández Collado y M. d. P. Baptista Lucio, Metodología de la Investigación, México D.F.: Mexicana, 2014.

[23] CISCO, «Solution Design Guide,» de *Campus LAN and Wireless LAN*, 2020, pp. 1-76.

[24] Mininet Team, «Mininet: An Instant Virtual Network on your Laptop (orother PC).,» [En línea]. Available: http://mininet.org/overview/.

[25] J. L. H. Ramirez, Artist, *Guía De Implementación Y Uso Del Emulador De Redes Mininet.* [Art]. UNIVERSIDAD TECNOLOGICA DE PEREIRA, 2015.

[26] M. P. Esteso, «Funcionamiento del comando TOP en Linux,» Agosto 2014. [En línea]. Available: https://geekytheory.com/funcionamiento-del-comando-top-en-linux. [Último acceso: 30 Noviembre 2020].