UNIVERSIDAD DISTRITRAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA



PROYECTO DE GRADO
MONOGRAFÍA

"ANÁLISIS DE LAS CAPACIDADES Y PRESTACIONES DE CALIDAD DE SERVICIO EN REDES DEFINIDAS POR SOFTWARE"

JONIER HERNANDO PORRAS DUQUE
DANIEL ORLANDO DUCUARA BELTRAN

DIRECTOR
GUSTAVO ADOLFO PUERTO LEGUIZAMON

INGENIERÍA ELECTRÓNICA
BOGOTÁ D.C, JUNIO DE 2017

TABLA DE CONTENIDO

CAPITULO 1. INTRODUCCIÓN 5
1.1 PLANTEAMIENTO DEL PROBLEMA 5
1.2 JUSTIFICACIÓN 7
1.3 OBJETIVOS 8
1.3.1 Objetivo General 8
1.3.2 Objetivos Específicos 8
1.4 ALCANCE Y LIMITACIONES 9
CAPÍTULO 2. MARCO DE REFERENCIA 10
2.1 REDES DEFINIDAS POR SOFTWARE (SDN) 10
2.1.1 Antecedentes 10
2.1.2 Concepto 10
2.1.3 Objetivo principal de su implementación 11
2.1.4 Arquitectura 11
2.1.5 Características 13
2.1.6 Beneficios de su implementación 13
2.1.7 Protocolo OpenFlow 14
2.2 ESTADO DEL ARTE DE LAS REDES DEFINIDAS POR SOFTWARE 15
2.3 CALIDAD DE SERVICIO (QOS) 17
2.3.1 Definición 17
2.3.2 Aplicaciones 17
2.3.3 Factores que afectan la calidad en el servicio 17
2.3.4 Clases de tráfico y tipo de servicio 17
2.3.5 Niveles 18
2.3.6 Mecanismos para establecer QoS 18
2.3.7 Encolamiento 19
2.4 BALANCEO DE CARGA (LOAD BALANCER) 21
2.4.1 Definición 21
2.4.2 Importancia 21
2.4.3 Utilidad en internet 21
2.4.4 Inconvenientes en su implementación 22
2.4.5 Camino más corto mediante el algoritmo de Dijkstra 22

2.4.6 Balance de cargas por destino y por paquete 23
CAPÍTULO 3. MININET Y CONTROLADORES SDN 25
3.1 MININET 25
3.1.1 Creación de una red 25
3.1.2 Herramienta ping 26
3.1.3 Ancho de banda 27
3.1.4 Topologías 27
3.1.5 Miniedit 31
3.1.6 Conexión con Wireshark 34
3.2 CONTROLADOR FLOODLIGHT 35
3.2.1 Ejecución del controlador 35
3.2.2 Conexión del controlador con Mininet 36
3.2.3 Verificación de conexión entre los hosts 36
3.2.4 Interfaz gráfica de la configuración de red a través de API JSON 37
3.2.5 Adición de reglas al controlador 37
3.2.6 Visualización de reglas instaladas 38
3.2.7 Eliminación de reglas del controlador 38
3.3 CONTROLADOR OPENDAYLIGHT 39
3.3.1 Ejecución del controlador 39
3.3.2 Conexión del controlador con Mininet 39
3.3.3 Verificación de comunicación entre los hosts 40
3.3.4 Interfaz gráfica de la configuración de red a través de API JSON 40
3.3.5 Adición de flujos al controlador 42
3.3.6 Visualización de flujos instalados 43
3.3.7 Eliminación de reglas del controlador 44
3.4 CONTROLADOR POX 45
3.4.1 Ejecución del controlador 45
3.4.2 Conexión del controlador con Mininet 46
3.4.3 Verificación de conexión entre los hosts 46
3.5 COMPARACIÓN ENTRE LOS CONTROLADORES 48
3.5.1 Solicitudes de ping 49
3.5.2 Ancho de banda 53
3.5.3 Conclusiones 55
CAPÍTULO 4. IMPLEMENTACIÓN DE ALGORITMO PARA BALANCE DE CARGA 56

4.1 USANDO CONTROLADOR OPENDAYLIGHT 56
4.1.1 Pruebas del camino más corto o de menor carga 57
4.1.2 Pruebas de balance de carga 66
4.2 USANDO CONTROLADOR FLOODLIGHT 75
4.2.1 Prueba del camino más corto o de menor carga 76
4.2.2 Prueba de balance de carga 85
CAPÍTULO 5. IMPLEMENTACIÓN DE POLÍTICAS DE CALIDAD DE SERVICIO (QOS) 90
5.1 PRIORIZACIÓN POR INTERFACES DE SALIDA 90
5.1.1 Ancho de banda sin interfaces restringidas 91
5.1.2 Ancho de banda con interfaces restringidas 92
5.2 PRIORIZACIÓN DE TRÁFICO MEDIANTE ENCOLAMIENTO 94
5.3 CALIDAD DE SERVICIO EN LA TRANSMISIÓN DE UN VIDEO 97
5.3.1 Implementando reglas QoS 97
5.3.2 Transmisión de video streaming 98
5.3.3 Análisis de la información transmitida y recibida 100
CAPÍTULO 6. COMPARACIÓN DE UNA RED TRADICIONAL Y UNA RED SDN MEDIANTE SIMULACIÓN 104
6.1 COMPORTAMIENTO DE UNA RED EN PACKET TRACER Y MININET 104
6.1.1 Medición de ancho de banda de la red 106
6.1.2 Solicitudes de ping 106
6.2 COMPORTAMIENTO DE UNA RED COMPUESTA POR SOLO SWITCHES EN PACKET TRACER Y MININET 110
CONCLUSIONES 113
REFERENCIAS 114
ANEXOS 117
ANEXO 1: Código de la red personalizada 1 117
ANEXO 2: Módulo de balance de carga para OpenDaylight 120
ANEXO 3: Módulo de balance de carga para Floodlight 128
ANEXO 4: Código de la red personalizada 2 138

CAPITULO 1. INTRODUCCIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

Con la aparición del internet se redefinió la manera en la que se comunicaban las personas. Su amplia implementación actual ha permitido el acceso a la información a millones de usuarios a nivel mundial, esto ha requerido que las redes de telecomunicaciones se estén actualizando constantemente para cubrir la demanda por parte de los usuarios que requieren servicios confiables y seguros. Sin embargo, esta demanda sigue creciendo día tras día, exigiendo tiempos de actualización en las redes cada vez más cortos para los proveedores de servicios de telecomunicaciones [1].

Los objetivos actuales de internet pasan por garantizar la demanda, la calidad del servicio y sobretodo gestionar mejor los recursos de la red y optimizar su rendimiento. Todo esto supone un desafío muy difícil de lograr para las arquitecturas de red IP tradicionales debido fundamentalmente a dos razones principales [2]:

- La primera es debida a lo que muchos conocen con el nombre de “Osificación de Internet”. Como consecuencia del amplio despliegue e inversión de internet realizado al principio y al ser una de las infraestructuras críticas de nuestra sociedad actual, internet se ha convertido en algo extremadamente difícil de evolucionar [2].

- La segunda se debe a que la arquitectura sobre la cual se trabaja está organizada de forma vertical, donde el plano de datos como el de control se encuentran mezclados dentro de cada uno de los dispositivos de red, reduciendo la flexibilidad, innovación y evolución de la infraestructura de red [2].

Las redes tradicionales son estáticas e inflexibles y al implementar en su gran mayoría dispositivos de hardware donde el control es distribuido y mezclado con el plano de datos, no permiten una evolución rápida hacia un proceso que contribuya a mejorar los servicios prestados [3].

Actualmente, una de las soluciones que permiten una mejor gestión sobre las redes de comunicación son las redes definidas por software o SDN por sus siglas en inglés.

Las redes definidas por software son un conjunto de técnicas de servicios de red que aumentan la flexibilidad y la utilización de los recursos, reduciendo así gastos generales en la red. Estas se caracterizan por su estructura centralizada, en las que el plano de control (software) toma decisiones automatizadas por medio del uso de algoritmos que deciden los caminos y los tratamientos de los paquetes de datos que se transmiten al plano de datos (hardware) [4].

Las SDN permiten que las redes cambien su comportamiento estático a uno dinámico, con el objetivo de aprovechar más los recursos de la red mediante el análisis del plano de control. Todo esto se lleva a cabo en tiempo real permitiendo así realizar una gestión directa y eficiente sobre los recursos de la red. Mediante esta nueva tecnología se busca optimizar considerablemente el uso de

los recursos de las redes de comunicaciones, así como su dependencia ante la intervención humana en la actualización del software y comportamiento en general.

Las redes SDN son consideradas mucho mejores que las redes tradicionales en el manejo de flujo de datos por razones que se mencionaron de manera general anteriormente, pero si se realizara un análisis más profundo entre estos dos estilos de redes, ¿Cuáles serían los aspectos referentes a calidad de servicio en los que sobresalen las redes SDN frente a las redes tradicionales?

1.2 JUSTIFICACIÓN

Las redes definidas por software, o SDN, representan el nuevo horizonte hacia la administración y gestión de redes de telecomunicaciones. Cada día se hace necesario el uso adecuado de los recursos existentes para proveer servicios de calidad a una demanda global en continuo crecimiento. Es por esto que la investigación en este campo se hace imprescindible en los retos que se presentan para los ingenieros encargados en esta área de la industria.

Son muchas las razones por las cuales hoy en día se desea realizar estudios y avances en esta nueva tecnología. En primer lugar, tiene un gran aporte socioeconómico al ofrecer una nueva forma de manejo de datos que permite solucionar el problema de los recursos físicos limitados para atender a una demanda en constante crecimiento. Los usuarios de las redes de comunicaciones, así como sus proveedores, requieren de nuevas técnicas que aseguren altas tasas de transferencia de datos, así como seguridad y mayor aprovechamiento de los recursos de la red. Los beneficiados con este nuevo modelo de redes, no solo serían las empresas líderes en servicios de telecomunicaciones y aquellas multinacionales que poseen grandes y sofisticadas redes dentro de su infraestructura, sino también todos aquellos que hacen uso de los diferentes servicios de comunicaciones, como redes telefónicas, internet mediante uso del WiFi o Ethernet, etc.

Además del aumento en la velocidad de transferencia de datos que proporciona las SDN, también se podrá dar una reducción en costos de operación y mantenimiento, debido a que la implementación de redes flexibles y escalables implica un aprovechamiento máximo de los recursos físicos existentes.

Esta tecnología al ser un campo que se encuentra aún en constante investigación, abre el camino a todas aquellas personas que se interesen en el tema a realizar estudios para aportar y ampliar el conocimiento ya existente y abrir paso hacia su implementación. Cada nuevo aporte permitiría el desarrollo de futuros estudios en lo que pueden ser partícipes ingenieros de diferentes disciplinas, como electrónica, sistemas, telecomunicaciones, entre otras.

1.3 OBJETIVOS

1.3.1 Objetivo General

- Realizar un estudio sobre las características de una red definida por software, así como las ventajas y desventajas que trae su utilización en términos de calidad de servicio.

1.3.2 Objetivos Específicos

- Reconocer las características de las redes definidas por software así como los parámetros de red que permiten establecer aspectos de calidad de servicio.

- Evaluar la configuración y desempeño de diferentes controladores SDN.

- Implementar algoritmos que mejoren las prestaciones de una red SDN en términos de ancho de banda y retardos.

- Comparar mediante procesos de simulación las prestaciones de una red tradicional y de una red SDN

1.4 ALCANCE Y LIMITACIONES

Se pretende desarrollar un estudio relacionado con el funcionamiento de la redes SDN frente a las redes convencionales. Para ello será necesario hacer uso de la información existente sobre el estado actual de las redes definidas por software tanto en desarrollos investigativos como hacia los diferentes procesos de implementación simulada y física que se ha venido dando en diferentes partes del mundo. Además, se realizará un análisis comparativo entre las redes convencionales y las redes definidas por software orientado al manejo de calidad de servicio, ancho de banda y retardos. Para este análisis se recurrirá a programas de simulación de redes y analizadores de tráfico.

CAPÍTULO 2. MARCO DE REFERENCIA

2.1 REDES DEFINIDAS POR SOFTWARE (SDN)

2.1.1 Antecedentes

La aparición de nuevos servicios de internet y la gran cantidad de información que hoy en día manejan los servidores, ha provocado que la infraestructura de red actual sea insuficiente para satisfacer toda la demanda. Este proceso de evolución de los servicios de comunicaciones, ha traído consigo grandes retos tanto a empresas como a desarrolladores de hardware y software de redes, que buscan explotar al máximo la infraestructura de red existente para permitir satisfacer todas estas necesidades [9].

Debido a que la arquitectura de red actual es ineficiente para suplir todas estas necesidades, se busca una nueva alternativa que permita solucionar estos inconvenientes sin tener que cambiar toda la infraestructura existente, y que reduzca los complejos protocolos que son incapaces de manejar la gran cantidad de datos que se manejan hoy en día [9]. Estas arquitecturas actuales carecen de flexibilidad y son demasiado complejas, donde su foco ha estado en minimizar el riesgo de indisponibilidad de los servicios, más que en la adaptabilidad, flexibilidad y agilidad. La gestión de las redes actuales está poco automatizada y para aplicar una política específica es necesario configurar manualmente múltiples dispositivos a través de comandos de línea [7].

Por todo lo anterior, las redes definidas por software (SDN) surgen como una solución a todos estos problemas que requieren un cambio de paradigma de cómo se comporta una red, haciéndolas más dinámicas e independientes, que permitan mejores tasas de velocidad, seguridad y mayor aprovechamiento de los recursos.

2.1.2 Concepto

Las redes definidas por software o SDN (*Software defined Networking*), se definen como una arquitectura de red dinámica, manejable, adaptable y de costo eficiente que la hace ideal para las altas demandas de ancho de banda. Esta arquitectura desacopla el control de la red y la funcionalidad de envío de información permitiendo que el control de esta pueda ser completamente programable, logrando que la infraestructura de red subyacente sea abstraída por las aplicaciones y servicios de red [9].

Las SDN surgen como una manera de simplificar las redes, llegando a la industria justo cuando la demanda en aumento, la gran diversidad de servicios y la complejidad cada vez mayor de las infraestructuras desafiaron la capacidad de los operadores de redes para activar y desactivar servicios rápidamente. Al abstraerse el control de los reenvíos, las SDN brindan a los operadores de redes un control centralizado más flexible y con un mejor tráfico de la red a través de la programación [10].

2.1.3 Objetivo principal de su implementación

Conseguir redes más sencillas, programables y flexibles, así como como crear redes más escalables y automatizables, tener un control centralizado, aumento en la seguridad y fiabilidad de la misma [9].

2.1.4 Arquitectura

Las redes definidas por software son una manera de abordar la creación de redes en la cual el control se desprende del hardware y se le da a una aplicación de software llamada controlador [11].

Los dispositivos de red tradicionales están conformados por dos partes o planos: el plano de datos y el plano de control. En el plano de datos es donde ocurre el tráfico e intercambio de paquetes, mientras que en el plano de control se toman las decisiones sobre qué hacer con los paquetes que se reciben y hacia dónde deben ir [12].

Así, las SDN proponen la desvinculación de los planos de control y de datos; de tal manera que el plano de control pueda ser implementado mediante software, mientras que la capa de datos (física) resida en servidores de propósito general, con hardware estándar [12].

Con este enfoque, el plano de control puede centralizarse en una sola instancia que permita el manejo de los elementos de red desde un solo punto. Adicionalmente, al tratarse de software, las operaciones de control pueden ser programadas para responder a diversos modos de uso, por ejemplo: reconfiguración automática para responder a condiciones de tráfico cambiantes, asignación de prioridades por tipo de tráfico, por aplicación o por usuario [12].

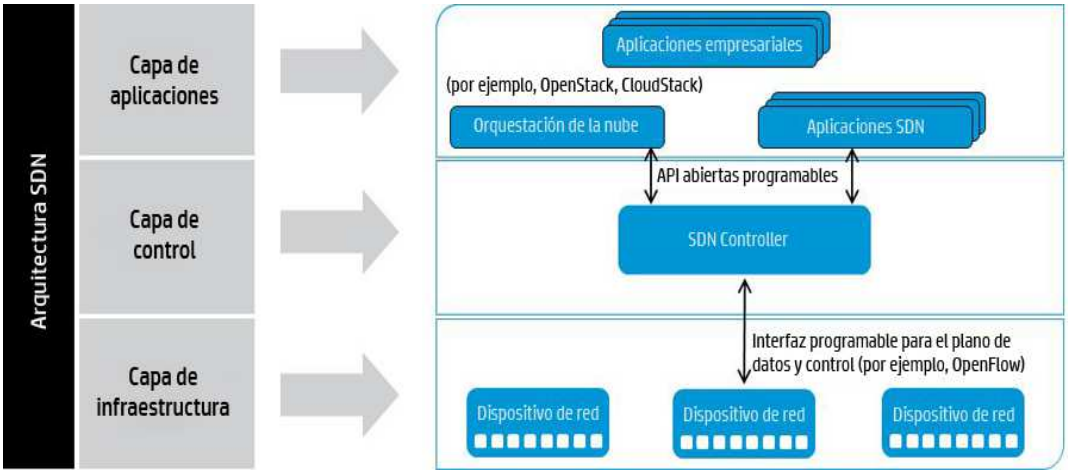


Figura 1. Arquitectura SDN [12].

2.1.4.1 Planos de control y de datos

- **Capa de datos:** Este plano permanece descentralizado y se encarga del reenvío de los paquetes tan rápido como sea posible. Permanece en el hardware de red y típicamente implementada a través de elementos especializados, conocidos como circuitos integrados de aplicación específica. Estos elementos de *hardware* deben operar con el protocolo Openflow para comunicarse con el plano de control e implementar las reglas de reenvío de tráfico [4].

- **Capa de control:** Contiene la inteligencia de la red de una forma centralizada, entiende la topología de la red y puede tomar decisiones sobre a dónde debe ir un flujo de tráfico específico (este plano siempre está centralizado). Permite modificar el comportamiento de la red en tiempo real así como desplegar nuevos servicios en poco tiempo (horas). El controlador es una especie de *middleware* que abstrae los componentes físicos subyacentes de la red como enrutadores, *firewalls* o balanceadores [4].

Al centralizar la inteligencia de la red, es posible verla de forma completa, lo que permite tomar mejores decisiones. Por otro lado, las políticas que se pueden implementar a través de este componente contemplan aspectos como control de accesos, control de tráfico, calidad de servicio, seguridad, etcétera [4].

En el siguiente diagrama se puede observar la arquitectura conceptual divida en los diferentes planos.

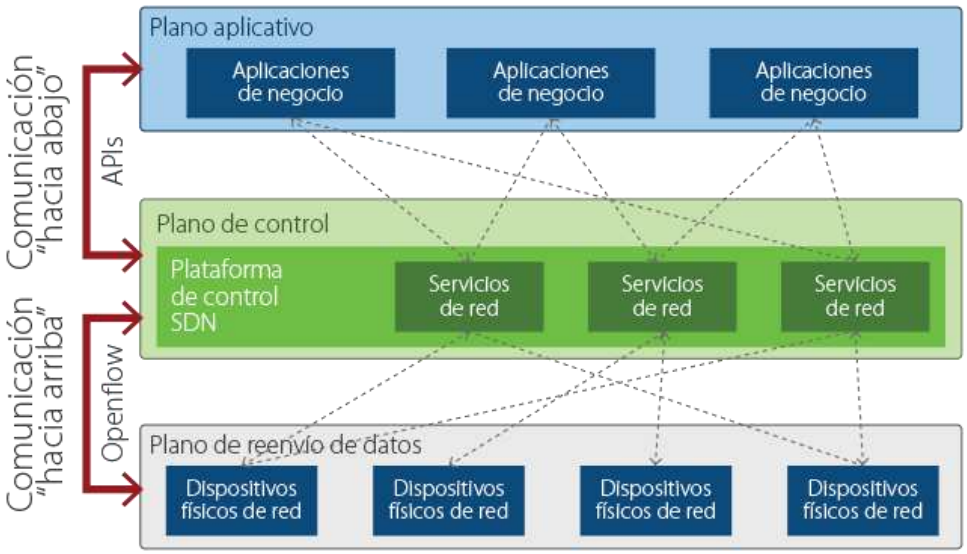


Figura 2. Capas de control y de datos [4].

2.1.4.2 Interfaces

De este modelo de planos se derivan dos tipos principales:

- "Hacia abajo" (*southbound*): Realizada principalmente a través del protocolo Openflow para comunicar al controlador con los equipos de red. Openflow permite el acceso y la manipulación directa del plano de reenvío datos de los equipos de red, ya sean físicos o virtuales [7].

- "Hacia arriba" (*northbound*): Para comunicar al controlador con las aplicaciones de negocio, se realiza principalmente a través de interfaces de programación de aplicaciones (application programming interface, API) [7].

2.1.5 Características

Las tres características principales de una Red Definida por Software son: [12]

- Separación de los planos de Control y de Datos
- Centralización del control
- Programabilidad

Otra característica muy importante de las SDN es el reconocimiento de la existencia de la capa aplicativa, en la cual se encuentran los servicios de red, las herramientas de orquestación y las aplicaciones de negocio que interactúan con el plano de control a través de API específicos [7].

2.1.6 Beneficios de su implementación

- **Simplicidad:** La red puede ser controlada y gestionada de forma centralizada como una sola entidad. Las tareas de gestión son automatizadas y aisladas de la compleja infraestructura física a través de interfaces fáciles de utilizar [7].

- **Agilidad:** Los nuevos servicios y aplicaciones pueden ser provistos en muy poco tiempo, además los administradores de TI obtienen la posibilidad de programar funcionalidades y servicios según lo requieran, eliminando así la dependencia de los fabricantes de hardware [7].

- **Control:** A través del plano de control se puede supervisar el flujo de información de la red, contemplando las aplicaciones como un elemento central. Permite lograr mejoras en la confiabilidad y seguridad de la red, centralizando la definición, configuración e implementación de políticas [7].

- **Mejoras en la experiencia del usuario final:** Al permitir que las aplicaciones exploten la información centralizada sobre el estado de la red y de la capacidad de adaptación del comportamiento de la misma, se pueden realizar ajustes en tiempo real para que los usuarios finales tengan un tiempo de respuesta de acuerdo a los niveles de servicio establecidos [7].

- **Reducción de costos operativos:** Finalmente, todo lo anterior redunda en el principal beneficio que se espera, que consiste en un menor costo operativo de gestión de las redes [7].

2.1.7 Protocolo OpenFlow

Actualmente, la especificación más popular para crear una red definida por software es un estándar abierto llamado OpenFlow. Este protocolo permite a los administradores de red controlar tablas de enrutamiento de forma remota [7].

El protocolo OpenFlow constituye la base de las redes abiertas definidas por software basadas en estándares. Este protocolo empezó a desarrollarse en 2007 y es el resultado de la colaboración de los sectores académico y empresarial. Fueron las universidades de Stanford y California en Berkeley quienes llevaron las riendas en primera instancia.

En la actualidad, la Open Networking Foundation (ONF) se encarga de la definición del estándar y es el grupo que más se asocia con el desarrollo de normas basadas en SDN [13].

2.2 ESTADO DEL ARTE DE LAS REDES DEFINIDAS POR SOFTWARE

Los intentos por optimizar el uso de las redes de telecomunicaciones surge a inicios de la década de los 90, cuando el uso de internet se popularizó y se hizo necesario crear técnicas que permitieran lograr una mejor gestión de los recursos existentes. Surge entonces la necesidad de crear protocolos de administración de los dispositivos de las redes, que con el tiempo llevarían al desarrollo de las SDN.

Diferentes tipos de técnicas han sido utilizadas desde entonces, cada una intentando dar respuesta a la necesidad de lograr una mejor gestión de los recursos. La historia de las SDN puede dividirse en tres etapas: Las redes activas, la separación de los planos de control y de datos; y el protocolo Openflow y Sistemas operativos de redes. Cada una de las etapas anteriores aporta al desarrollo posterior de lo que se definiría como redes definidas por software [1].

Actualmente se ha extendido la popularidad de las SDN para el diseño de redes de telecomunicaciones, pero pocos han sido los estudios en donde se analice la eficiencia de dichas redes en diferentes topologías. A la fecha se han realizado estudios de eficiencia de las arquitecturas SDN enfocándose en áreas específicas, sin realizar un análisis global del comportamiento de las redes. Por ejemplo, el estudio de Tootonchian se enfoca en la evaluación del comportamiento del plano de control; Rostos propone una herramienta para la evaluación de una arquitectura específica de SDN. Bianco realiza otro estudio en el que evalúa el desempeño del plano de datos. Finalmente, el estudio realizado por Gelberger, Yemini y Giladi analiza varias variables simultáneamente, y centran su atención en el impacto de la SDN en términos de rendimiento y latencia [5].

El primer caso de éxito a nivel mundial de una red SDN es CandIT-Media fundada en 2010. Se trata de una compañía especializada en centros de datos de contenidos multimedia con necesidades tremendamente dinámicas en cuanto a los flujos necesarios. Alcatel-Lucent ha desplegado la primera referencia SDN en CandIT-Media, integrándose en el controlador software existente, proporcionando una red configurable que optimiza en todo momento el acceso al almacenamiento [6].

En un evento realizado por Network World, se encontró que solo 10% de los 450 asistentes dijeron entender el concepto de SDN. Por otro lado, de los miembros del ONUG 56% está revisando el concepto de SDN, mientras que 28% ya está corriendo pruebas de concepto (*PoC*), y un 16% ha realizado ya implementaciones limitadas [7].

Empresas como HP, IBM y Cisco han visto una gran oportunidad en la investigación y desarrollo de las SDN como una tendencia actual en la industria de las telecomunicaciones.

La empresa Cisco es pionera en el desarrollo e implementación de redes definidas por software. Se centra en hacer que la red sea más programable y virtualizada, lo que requiere algo más que tecnología SDN. De esta forma, su estrategia comprende tres partes: SDN, virtualización de red y programación a través de API.

Cisco también ha estado proporcionando interfaces de programación abiertos para sistemas operativos (IOS, ISO-XR y NX-OS) facilitando la integración de aplicaciones escalables con la

infraestructura de red subyacente y el acceso a herramientas de gestión y orquestación de terceros fabricantes. Además está participando activamente en el desarrollo de OpenFlow y de la tecnología SDN**,** colaborando con grupos de la industria como ONF, IETF o ITU [6].

Cisco integra su completo desarrollo en SDN en la arquitectura Cisco ONE (Open Network Environment) la cual proporciona un completo marco de trabajo para la programación de red, la provisión automatizada y las interacciones basadas en las aplicaciones. En este sentido, Cisco ONE ofrece un controlador de software para las Universidades, Cisco One Platform Kit (onePK) para proporcionar las API que reclaman los clientes de grandes centros de datos, la solución Cisco Cloud Connected que proporciona a los proveedores la capacidad para optimizar y diferenciar sus servicios cloud, así como un host de Virtual Overlay Networks que amplía el soporte para OpenStack, múltiples hipervisores y funcionalidad de Gateway VXLAN para garantizar la consistencia a través tanto de redes físicas como virtuales [6].

HP, por su parte, es la única compañía que proporciona a las empresas una solución de red definida por software completa, que automatiza las tareas de configuración manual a través de hardware, software y aplicaciones, desde el centro de datos hasta el escritorio, a través de un único plano de control [6]. Por otra parte, HP fundó la Open Networking Foundation (ONF) que se dedica a la promoción y adopción de las SDN a través de desarrollo abierto de estándares basados en el protocolo Openflow, además del análisis continuo de los requerimientos en la implementación de redes, según las necesidades a nivel comercial [8].

Hoy en día existen varias tendencias que provocan cambios importantes en los requerimientos hacia la arquitectura de redes: el fuerte incremento en el uso de dispositivos móviles, la proliferación de la virtualización de servidores, la alta adopción de servicios en la nube, entre otros, implica grandes cambios en los patrones de tráfico que circulan en la red, nuevos requerimientos en los anchos de banda y diferentes niveles de calidad de servicio. En pocas palabras, se requiere que los usuarios se puedan conectar, asegurándoles una experiencia satisfactoria en todo momento y desde cualquier dispositivo.

2.3 CALIDAD DE SERVICIO (QOS)

2.3.1 Definición

Es la capacidad que posee una red de proveer diferentes niveles de servicio para asegurar distintos perfiles de tráfico [17].

Mediante el establecimiento de QoS en la red, se deben cumplir un conjunto de requisitos en el servicio que va enfocado en el transporte de flujos. Puede ser implementada en diferentes situaciones, tanto para gestionar la congestión o para evitarla. En general, mediante QoS se permite controlar características significativas de la transmisión de paquetes [17].

Estas características pueden especificarse en términos cuantitativos o estadísticos, tales como: ancho de banda, latencia, jitter, pérdida de paquetes en la red; asegurando un grado de fiabilidad preestablecido que cumpla con los requisitos de tráfico, en función del perfil y ancho de banda para un determinado flujo de datos [17].

2.3.2 Aplicaciones

La motivación para aplicar Calidad de Servicio en redes se resume en las siguientes necesidades:

- Priorizar ciertas aplicaciones en la red que requieren de un alto nivel de servicio VOIP.
- Maximizar el uso de la infraestructura de red, manteniendo un margen de flexibilidad, seguridad y crecimiento para servicios emergentes.
- Mejorar las prestaciones para servicios en tiempo real.
- Responder a los cambios en el perfil de tráfico establecido.
- Proporcionar mecanismos para priorizar tráfico.

2.3.3 Factores que afectan la calidad en el servicio

Son diversas las causas que pueden atentar contra el correcto funcionamiento de la red o que el usuario tenga una percepción negativa del servicio recibido. Estos factores están dados en su mayoría a que la voz debe viajar en un entorno diseñado para paquetes de datos, sufriendo cambios de paquetización, fragmentación, intercalado, codificación o decodificación a través de la red [17].

2.3.4 Clases de tráfico y tipo de servicio

El objetivo básico de las clases de tráfico (CT) y el tipo de servicio (ToS) es conseguir el ancho de banda y la latencia que se requiere para una aplicación determinada. Las clases de tráficos permiten al administrador de la red agrupar diferentes flujos de paquetes, teniendo cada uno requisitos de latencia y ancho de banda diferente [17].

El tipo de servicio es un campo en una cabecera IP, que permite que tenga lugar una clase de servicio determinada. Por otro lado, las clases de servicio (CoS) son un esquema de clasificación

para agrupar los tráficos que tienen requerimientos de tratamiento similares, para diferenciar los tipos de tráfico y por ende poder priorizarlos [17].

2.3.5 Niveles

Los niveles de Calidad de Servicio están referidos a las actuales capacidades de las conexiones de extremo a extremo, es decir, las capacidades que tiene una red determinada de realizar un servicio para un tráfico específico. Los servicios difieren en cuán estrictos pueden ser los niveles de QoS, es decir, que tiene que ser específico para un ancho de banda, jitter o pérdida de paquetes determinado [17].

- Nivel Best Effort: básicamente estos servicios no ofrecen ninguna garantía. Usualmente utiliza técnicas FIFO (First in First Out o Primero en Entrar Primero en Salir), que no tienen ninguna diferenciación entre los distintos flujos [17].

- Nivel para Servicios Diferenciados (Diffserv): se basa en la división del tráfico en diferentes clases y en la asignación de prioridades [17].

- Nivel Garantizado: está destinada para aplicaciones con requerimientos exigentes de tiempo real. Esta calidad asegura un ancho de banda, un límite en el retardo y ninguna pérdida en las colas [17].

2.3.6 Mecanismos para establecer QoS

Son diversos los mecanismos existentes que se implementan para garantizar una adecuada Calidad de Servicio, los cuales se muestran a continuación: [17]

2.3.6.1 Gestión de colas

Por la naturaleza que tiene la transmisión de aplicaciones multimedia a través de la red, propicia que la cantidad de tráfico no exceda la velocidad de la conexión haciendo varias colas para los diferentes servicios [17].

2.3.6.2 Clasificación de paquetes

Para manipular los tráficos y otorgarles QoS, se utilizan los procedimientos básicos de clasificación y asignación de prioridad [17].

2.3.6.3 Medición y flujo de formación de tráfico

En muchas ocasiones es necesario limitar la cantidad de tráfico de una aplicación a través de varias interfaces [17].

2.3.6.4 Gestión de colas de altas velocidades

Se basa en la manera que los protocolos operan, con el fin de no llegar a la congestión de la red [17].

2.3.6.5 Metodologías de Estimación de Calidad de Servicio Percibida

Es la calidad del servicio percibido por el usuario independientemente del estado de la red. Las medidas de calidad percibida pueden realizarse usando métodos objetivos o subjetivos [17].

2.3.7 Encolamiento

2.3.7.1 Encolamiento de prioridad (PQ)

Es una metodología a través de la cual se ofrece un tratamiento preferencial a los paquetes, que en el momento de ingresar a la interfaz, son identificados por prioridad. Cada paquete se asigna a una de las colas disponibles, que son tratadas en estricto orden de prioridad. Los paquetes se sirven de la cabecera de una cola, sólo, si todas las colas de prioridad mayor están vacías. PQ se ajusta a condiciones donde existe tráfico importante, pero puede causar la total falta de atención de colas de menor prioridad. Por ejemplo, se pueden colocar prioridades a las aplicaciones de tiempo real, como voz y video interactivo, y que se traten de forma prioritaria frente a otras aplicaciones que no operan en tiempo real [21].

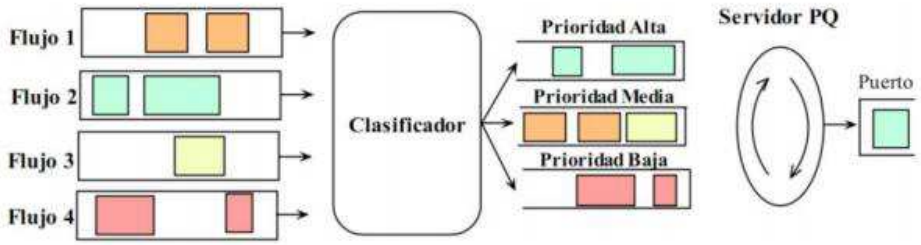


Figura 3. Encolamiento de prioridad [21].

2.3.7.2 Encolamiento Personalizado (CQ)

El encolamiento personalizado, o cola de prioridad, es un mecanismo establecido para priorizar el tráfico, evitando la inanición de las colas de menor prioridad, en donde se especifica el número de paquetes que deben atenderse por cada cola. Cabe resaltar que no asegura prioridad absoluta como en PQ. Es empleado para proporcionar ancho de banda a tráficos en particular, en un punto de posible congestión. Asegurando una porción de ancho de banda al tráfico que lo amerite, y permitiendo el uso de los recursos disponibles al resto del tráfico [21].

2.3.7.3 Encolamiento de baja latencia (LLQ)

Es el método de encolamiento recomendado para voz sobre IP (VoIP) y telefonía IP. Consta de colas de prioridad personalizadas basadas en clases de tráfico, junto con una cola de prioridad que tiene preferencia sobre el resto de colas. Debe configurarse ancho de banda límite reservado para la cola de prioridad. Esta cola da un máximo de retardo garantizado para los paquetes entrantes a la misma, que se calcula como el tamaño del MTU dividido por la velocidad del enlace [21].

2.4 BALANCEO DE CARGA (LOAD BALANCER)

2.4.1 Definición

El balanceo de carga permite utilizar rutas paralelas existentes entre nodos de ingreso y egreso para distribuir los flujos de información transmitidos en la red, lo que contribuye a la disminución de la congestión a través del ruteo y control de tráfico de acuerdo a los recursos existentes en los entornos backbone [18].

2.4.2 Importancia

Contribuye notablemente a la reducción de problemas de congestión de las redes y a utilizar de forma más apropiada los enlaces disponibles dentro del core. El hecho de combinar todas estas características puede llegar a generar nuevos modelos y estructuras que soporten la distribución adecuada y equilibrada de tráfico con garantías de calidad de servicio obteniendo de esta manera caminos más óptimos a destinos [18].

Desde el punto de vista de QoS, Load Balancing se postula con un gran papel hacia el futuro de las redes y comunicaciones, por su versatilidad y constantes mejoras para los diferentes sistemas de comunicación e información [20].

2.4.3 Utilidad en internet

El balanceo de carga en redes provee escalabilidad y fácil manejo a los servicios de TCP/IP, Web, Proxy, VPN (Redes Virtuales Privadas) y servicios de multimedia [20].

El balanceo de carga distribuye el tráfico IP a múltiples copias o instancias de servicios TCP/IP, cada uno corriendo en un host dentro de la granja (o cluster, si es una granja de servidores corriendo una aplicación web) de servidores. Se realizan particiones transparentes de las peticiones de los clientes a través de los hosts y se deja que los clientes accedan a la granja de servidores utilizando una o más direcciones IP virtuales. Desde el punto de vista del cliente, la granja parece ser un solo servidor que responde a sus peticiones. Si el tráfico se incrementa, el administrador de red simplemente conecta otro servidor a la granja [20].

En la figura 1, los servidores que se encuentran agrupados trabajan juntos para satisfacer el tráfico de la Internet. Cada servidor corre una copia de un servicio basado en IP, y el balanceo de carga distribuye la carga de trabajo de la red alrededor de ellos [20].

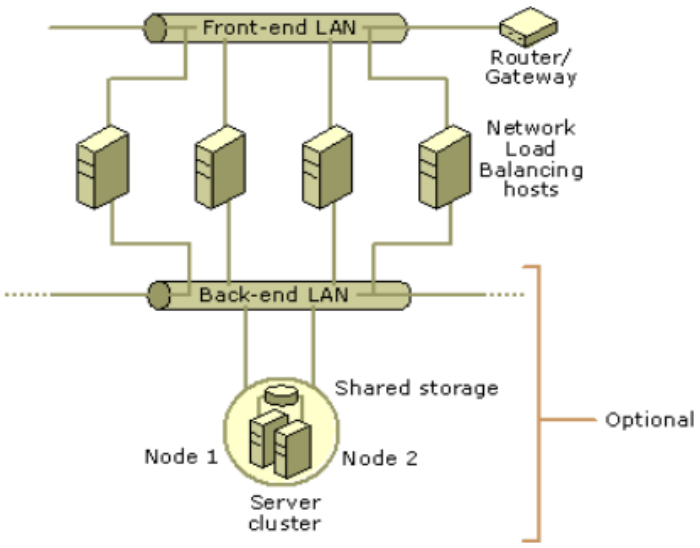


Figura 4. Escenario de red de balanceo de carga [20].

2.4.4 Inconvenientes en su implementación

La gran mayoría de los inconvenientes presentes en las redes actuales que impiden lograr un adecuado balanceo de carga tienen relación con el algoritmo de enrutamiento que utilizan, el cual fundamentalmente es el algoritmo del camino más corto (shortest path). Con este algoritmo, cada paquete que entra en la red busca el camino de menor número de saltos que le permita alcanzar el destino, que suele ser el mismo para todos los paquetes, así existan en la red otros caminos con mayor número de saltos y mucho más rápidos. Esto degrada el funcionamiento de la red en aspectos como: la congestión que se produce sobre el enlace de la ruta más corta; la sobre utilización de unos enlaces mientras otros son utilizados muy poco; y la disminución del throughput efectivo de la red [18].

2.4.5 Camino más corto mediante el algoritmo de Dijkstra

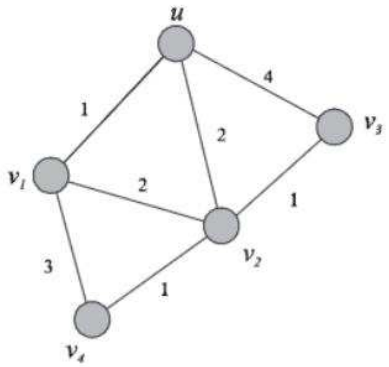


Figura 5. Escenario de red [19].

De acuerdo con la figura 1, cada nodo *v* del grafo *G(V,E)* tiene una etiqueta asociada *L(v)*, esta etiqueta indica la menor distancia conocida para ir desde un nodo fijado *u* hasta este nodo. Inicialmente, el valor de *L(v)* corresponde al peso *w(u,v)* de la arista que une los nodos *u* y *v*, en el caso de que esta arista exista, siendo $L(v) = \infty$; en caso contrario (desconocimiento de las distancias), el algoritmo funciona creando un conjunto de nodos $T \subset V$, para los cuales se ha obtenido hasta ese momento el camino más corto desde *u* hasta ellos. Al final del algoritmo, *L(v)* contiene el costo del camino más corto para ir desde *u* hasta *v* [19].

En cada iteración el algoritmo añade un nuevo nodo en la lista *T*. Esto se consigue escogiendo un nodo *v'* que todavía no pertenezca a *T* y que tenga una etiqueta *L(v´)* mínima. En otras palabras, se escoge el nodo *v'* más cercano a *u* y externo a la lista *T*. Una vez hecho esto, se actualizan las etiquetas de los nodos sobre los que incide *v'*, de manera que se hace un nuevo cálculo de las distancias de *u* a estos nodos y se añade este nodo *v'* a *T*. El proceso se repite hasta que todos los nodos del grafo se encuentren en la lista [19].

1. para todo $v \neq u$ $L(v) = w(u,v)$
2. $L(u) = 0$
3. $T = \{u\}$
4. mientras $T \neq V$
   Inicio
5. encuentra $v' \notin T$ de forma que $\forall\ v \notin T\ L(v') \leq L(v)$
6. $T = T \cup \{v'\}$
7. para todo $v \notin T$ de forma que v' es adyacente a v
   si $L(v) > L(v') + w(v',v)$
   entonces $L(v) = L(v') + w(v',v)$
   fin si
   fin para
   fin mientras

2.4.6 Balance de cargas por destino y por paquete

Puede configurarse el balance de cargas para que se realice por destino o por paquete.

El balance de cargas por destino significa que los paquetes se distribuyen según la dirección de destino. Dados dos trayectos a la misma red, todos los paquetes para el destino 1 en esa red se transmiten a través del primer trayecto; todos los paquetes para el destino 2 en esa red se transmiten a través del segundo trayecto y así sucesivamente. De esta manera se preserva el orden de paquetes, con un posible uso desigual de los enlaces. Si un host recibe la mayoría del tráfico, todos los paquetes usan un enlace, lo que deja el ancho de banda en los demás enlaces sin utilizar. Una gran cantidad de direcciones de destino lleva a más enlaces utilizados de forma equitativa. Para lograr más enlaces utilizados equitativamente se debe crear una entrada de caché de ruta para cada dirección de destino, en lugar de cada red de destino, como cuando sólo existe un único trayecto. Por consiguiente, el tráfico a diferentes hosts en la red de mismo destino puede usar diferentes trayectos. La desventaja de este enfoque es que los routers de núcleo/estructura básica que

transportan tráfico a miles de hosts de destino, donde los requisitos de memoria y procesamiento para mantener la caché pueden ser muy exigentes [20].

El balance de carga por paquete significa que el router envía un paquete para el destino 1 a través del primer trayecto, el segundo paquete para el destino 1 (mismo) a través del segundo trayecto y así sucesivamente. El balance de carga por paquete asegura el balance de carga entre todos los enlaces. No obstante, existe la posibilidad de que los paquetes puedan llegar sin un orden al destino a causa de la demora diferencial que existe dentro de la red [20].
Para el balance de cargas por paquete, el proceso de reenvío determina la interfaz saliente para cada paquete a través de una búsqueda en la tabla de rutas y la selección de la interfaz menos utilizada. De esta forma se garantiza la utilización equitativa de los enlaces, pero es una tarea que requiere una alta utilización del procesador e impacta en el desempeño en general del reenvío. Esta forma de balance de cargas por paquete no es la adecuada para las interfaces de velocidad más alta [20].

CAPÍTULO 3. MININET Y CONTROLADORES SDN

3.1 MININET

Mininet es una herramienta que permite emular redes definidas por software. Se caracteriza por poseer múltiples herramientas para personalización de redes, como topologías, anchos de banda, retardos, compatibilidad con varios controladores, entre muchas otras cosas. Las redes pueden ser generadas utilizando una interfaz gráfica a través de miniedit o mediante código Python.

La herramienta Mininet se utiliza en su versión más actual, es decir la 2.2.1. Cabe resaltar que este emulador de redes trabaja únicamente en el sistema operativo Linux, por lo que se utiliza la versión más actualizada de Ubuntu, la 16.04.

Nota: Se recomienda abrir este documento con Adobe Reader si se pretende copiar los comandos mostrados aquí, con el fin de evitar espacios y saltos de líneas que aparecen con otro visualizador.

3.1.1 Creación de una red

Teniendo previamente instalada la herramienta Mininet, podemos generar por defecto una red muy sencilla basada en 1 switch y 2 host:

sudo mn

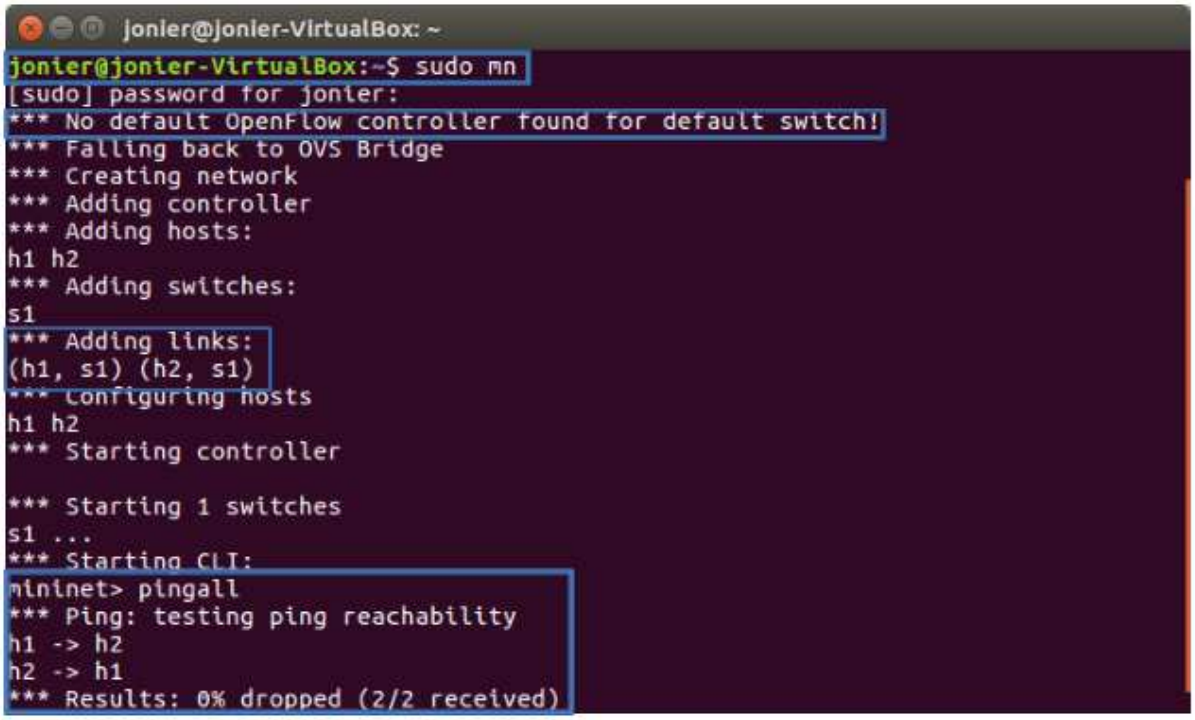


Figura 6. Red básica en Mininet utilizando la instrucción "sudo mn"

En la imagen anterior se puede observar los parámetros generales de la red creada, como la cantidad de hosts y switches de la red, y la respectiva conexión entre estos. También aparece el mensaje “No default Openflow controller found default switch”, que nos indica que Mininet no ha encontrado ningún controlador para conectarse, por lo que de forma predeterminada crea la red sin la existencia de un controlador externo que comande la red.

3.1.2 Herramienta ping

Para comprobar la conexión entre todos los dispositivos de la red, se utiliza la siguiente sentencia en Mininet:

pingall

En la figura 6 se puede observar que hay comunicación entre los 2 hosts. Si hay algún problema en la conexión aparecerá una ‘X’ en vez del host correspondiente.

Para enviar solicitudes de ping entre dos hosts específicos, se utiliza la siguiente sentencia en Mininet:

h1 ping h2

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.546 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.032 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=0.032 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.036 ms
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 5997ms
rtt min/avg/max/mdev = 0.032/0.110/0.546/0.178 ms
```

Figura 7. Ping entre h1 y h2

De esta manera se envían paquetes de 64 bytes, con un tiempo de vida (TTL) de 64 por defecto, así mismo se puede observar el tiempo que demora en realizarse la comunicación, que suele ser proporcional al ancho de banda del enlace y a la carga de la red. Al finalizar la herramienta ping, Mininet genera unas estadísticas con la información de los paquetes transmitidos y recibidos, así como el porcentaje de paquetes perdidos.

min: Menor tiempo de un paquete en ser transmitido.
avg: Tiempo promedio de los paquetes transmitidos.
max: Mayor tiempo de un paquete en ser transmitido.

3.1.3 Ancho de banda

Mediante la herramienta iperf se pueden realizar mediciones de ancho de banda. Para medir el ancho de banda del enlace que conforma los hosts h1 y h2, se escribe la siguiente sentencia:

iperf h1 h2

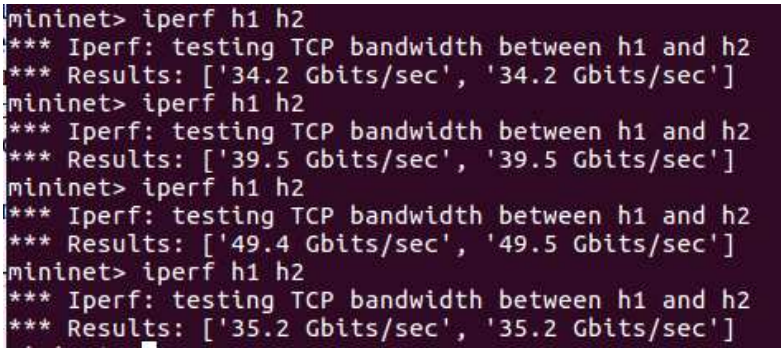


Figura 8. Ancho de banda entre h1 y h2

En la imagen anterior se pueden observar cuatro mediciones de ancho de banda, donde cada medición arroja un valor diferente, lo que indica que el ancho de banda en un enlace no siempre es el mismo. Sin embargo, esta variación no suele ser demasiado grande, mientras se mida bajo las mismas condiciones.

3.1.4 Topologías

Dentro de las funciones de Mininet está la posibilidad de crear topologías de red predeterminada como sencilla, lineal y en árbol. Si se desea crear una red personalizada es necesario hacerlo a través de código Python o miniedit.

3.1.4.1 Topología sencilla

Para crear topologías de red de forma predeterminada, Mininet posee instrucciones para facilitar su creación, por ejemplo para generar una red sencilla se utiliza la siguiente secuencia:

sudo mn --topo single,4

Esta instrucción crea una red con 1 switch y múltiples hosts (en este caso 4). Todos los hosts de la red se conectan al switch, como se puede observar en la siguiente imagen:

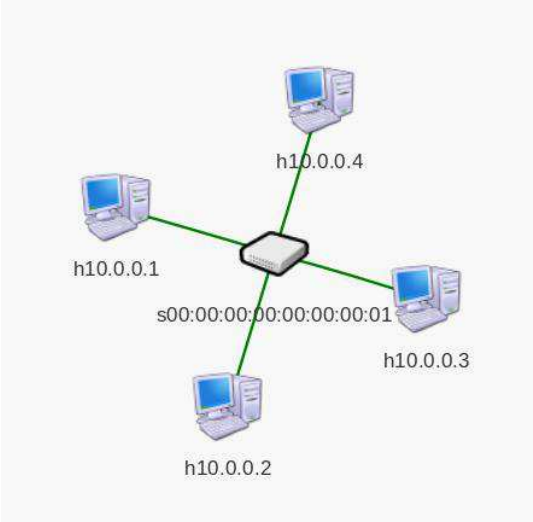


Figura 9. Topología sencilla

```
jonier@jonier-VirtualBox: ~
jonier@jonier-VirtualBox:~$ sudo mn --topo single,4
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** 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

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

Figura 10. Creación de una red sencilla en Mininet

Se recomienda realizar pingall cada vez que se crea una red, debido a que suele ocurrir muchos problemas con el controlador en la conexión del puerto de escucha, flujos creados, etc.

3.1.4.2 Topología lineal

Esta topología permite la creación de una red con el mismo número de hosts y switches.

sudo mn --topo linear,3

En este caso se crea una red con 3 hosts y 3 switches, cada host se conecta a un switch, y estos a su vez se enlazan en forma lineal o bus.

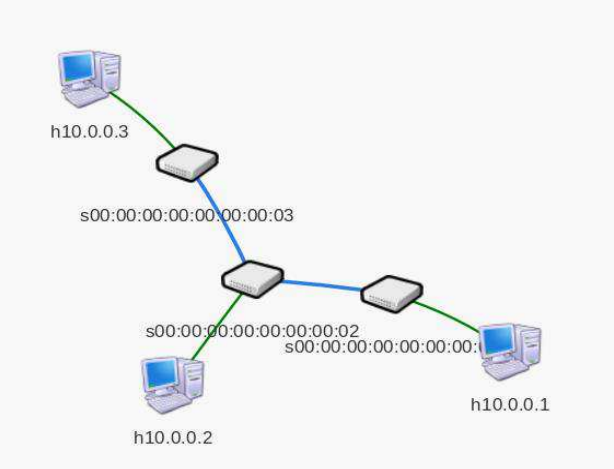


Figura 11. Topología Lineal

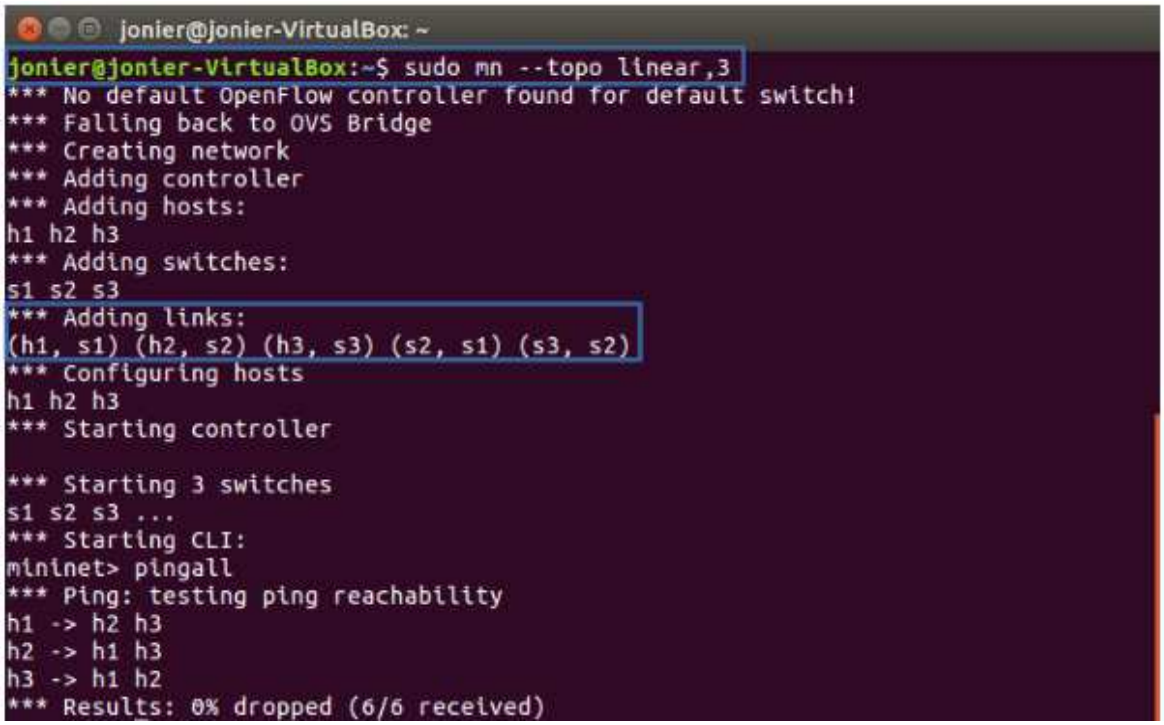


Figura 12. Creación de una red lineal en Mininet

3.1.4.3 Topología en árbol

Para crear una topología en árbol, se requiere introducir dos parámetros, la profundidad de la red, y el número de hosts que se conectan a cada switch en el fondo de la red.

sudo mn --topo tree,depth=2,fanout=3

Para este ejemplo, se crea una red con una profundidad de 2, es decir dos niveles, y 3 hosts en cada una de las ramas finales o switches del nivel más profundo.

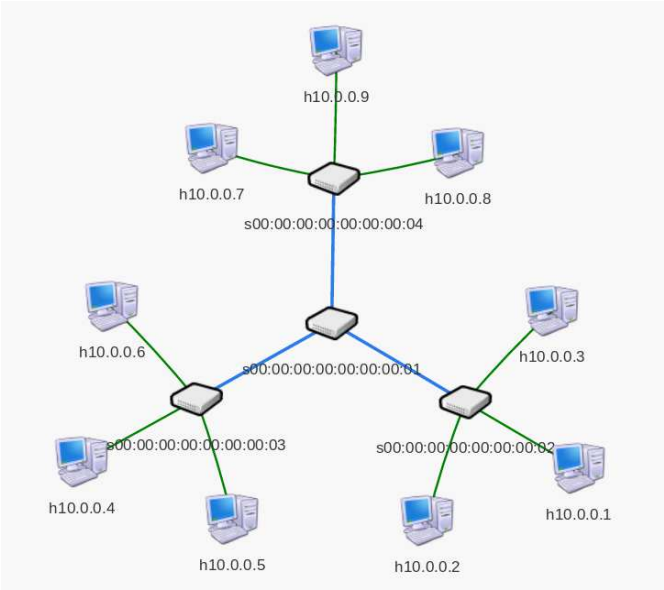


Figura 13. Topología árbol

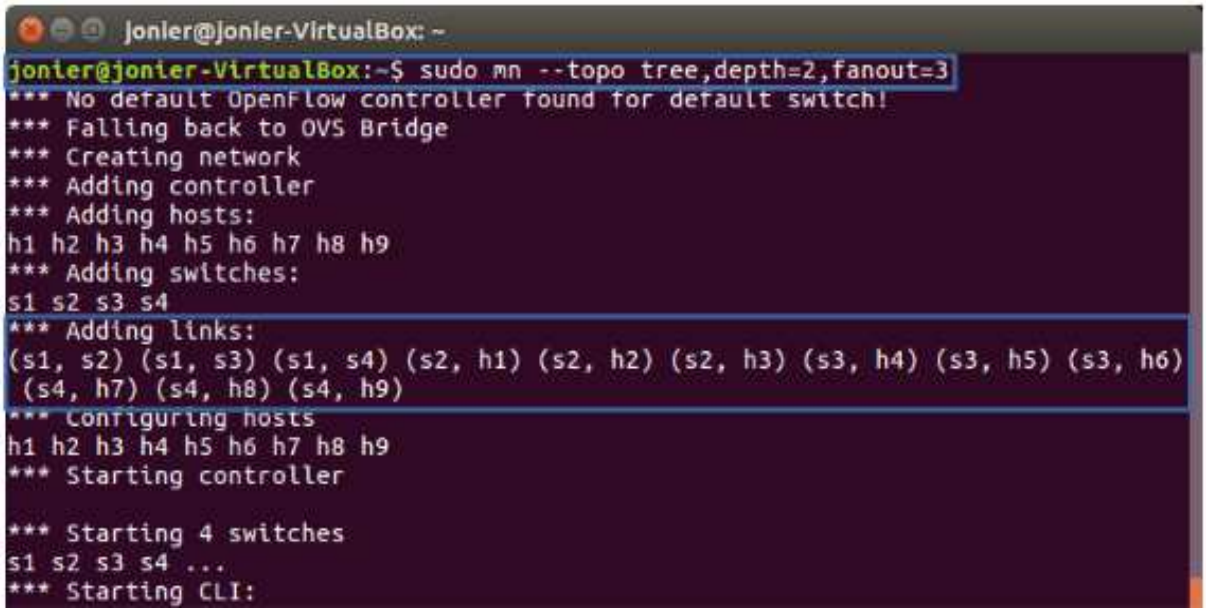


Figura 14. Creación de una red en topología árbol desde Mininet

3.1.5 Miniedit

Mininet posee una herramienta para crear redes personalizadas de forma gráfica, siendo una alternativa a los tediosos códigos que se requieren emplear para configurar una red.

sudo python /home/jonier/mininet/examples/miniedit.py

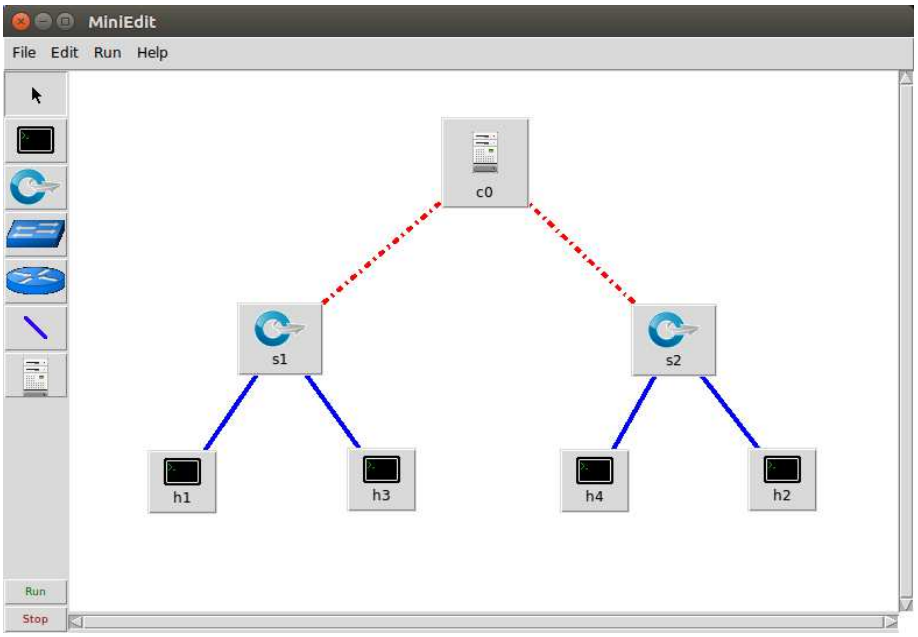


Figura 15. Interfaz gráfica de miniedit

En la imagen anterior aparece una red creada mediante la herramienta. El elemento c0 es el controlador, s1 y s2 son switches OpenFlow, h1, h2, h3 y h4 son hosts.

Es necesario especificar en las propiedades del controlador la dirección IP y el puerto de escucha, así como seleccionar la opción "Remote Controller" cuando se vaya a conectar Mininet con un controlador externo como Opendaylight o Floodlight.

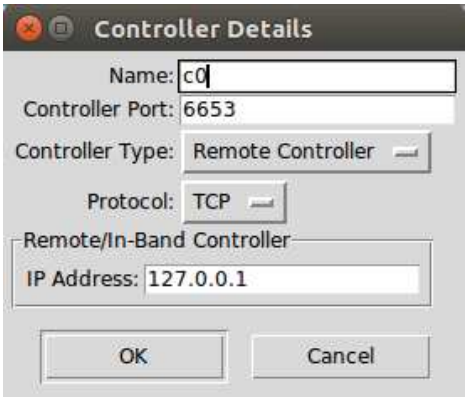


Figura 16. Propiedades del controlador

También es importante colocar en preferencias la dirección IP base que el controlador asignará a los hosts, así como el protocolo OpenFlow que se va a utilizar; esto se encuentra en la herramienta de opciones: Edit > Preferences.

Para poder ejecutar la red desde Mininet, primero se debe exportar el archivo como .py (código en Python), que se encuentra en la herramientas de opciones: File > Export Level 2 Script.
El archivo también se debe guardar como .mn para poderlo abrir con miniedit.

Teniendo el código de Python exportado, se utiliza la siguiente sentencia para iniciar la red desde Mininet:

sudo python /home/jonier/RedCreada.py



Figura 17. Verificación de conexión entre los hosts mediante pingall

Al comprobar la conexión entre los hosts, se puede observar que entre h1 y h3 hay comunicación así como entre h2 y h4, mientras que entre otros hosts no es posible, esto se debe a que la conexión entre los otros hosts requiere enviar paquetes a través del controlador, pero no existe un controlador externo conectado.

Para ejecutar redes creadas en código Python, también se puede utilizar la siguiente sentencia:

sudo mn --custom RedEnCodigo.py --topo mytopo

3.1.5.1 Conexión con un controlador externo

Para utilizar un controlador externo es necesario especificar la dirección IP y el puerto de escucha. Existen muchos controladores al día de hoy que son compatibles con la herramienta de emulación de Mininet, por ejemplo: Floodlight, Opendaylight, POX, NOX, Beacon, entre otros.

La sentencia que nos permite conectar Mininet con un controlador externo es la siguiente:

sudo mn --topo single,3 --controller=remote,ip=127.0.0.1,port=6653

Se puede especificar la versión del protocolo OpenFlow (1.3 en este caso) con esta sentencia:

sudo mn --topo single,3 --controller=remote,ip=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13

Este es un ejemplo para comunicarse con el controlador OpenDaylight o el controlador Floodlight, que por defecto utilizan el puerto 6653 y la dirección IP 127.0.0.1 para realizar la conexión. En el caso de usar otro controlador, se debe conocer el puerto de escucha y la dirección IP.

Antes de ejecutar la red en Mininet, se debe primero inicializar el controlador para que la red funcione adecuadamente (En la siguiente sección se explica cómo se ejecutan los controladores).

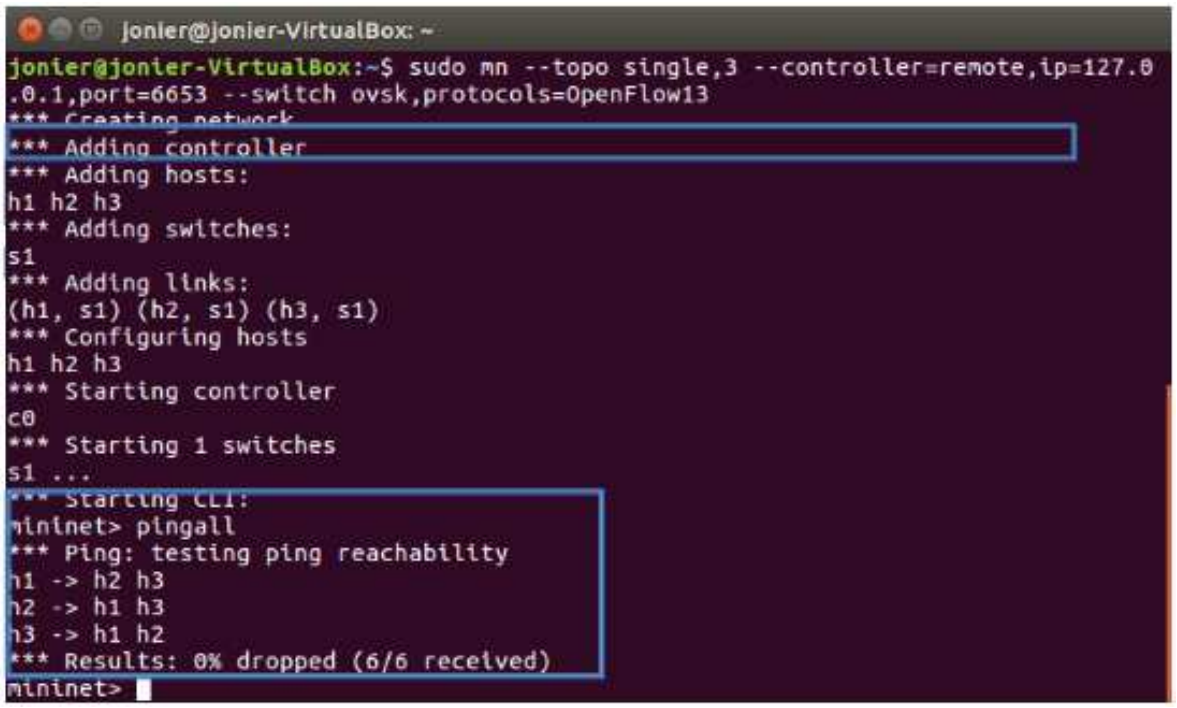


Figura 18. Creación de una red conectada a un controlador remoto

Se puede ver en la imagen anterior que el controlador es agregado sin problemas, y la conexión entre los hosts funciona correctamente; sin embargo, si existe algún problema en la conexión con el controlador, no habría comunicación entre los hosts. El mensaje típico de error al intentar agregar un controlador, es el siguiente: “Unable to contact the remote controller at 127.0.0.1:6653”.

Una herramienta alternativa a miniedit es el programa online VND por sus siglas en inglés Visual Network Description, con el cual es posible crear redes en un entorno gráfico y obtener el código en Python asociado a la estructura de red creada.

3.1.6 Conexión con Wireshark

Teniendo previamente instalado el analizador de protocolos Wireshark (versión actual 2.2.6), podemos ejecutarlo utilizando una terminal diferente, mediante la siguiente sentencia:

sudo wireshark

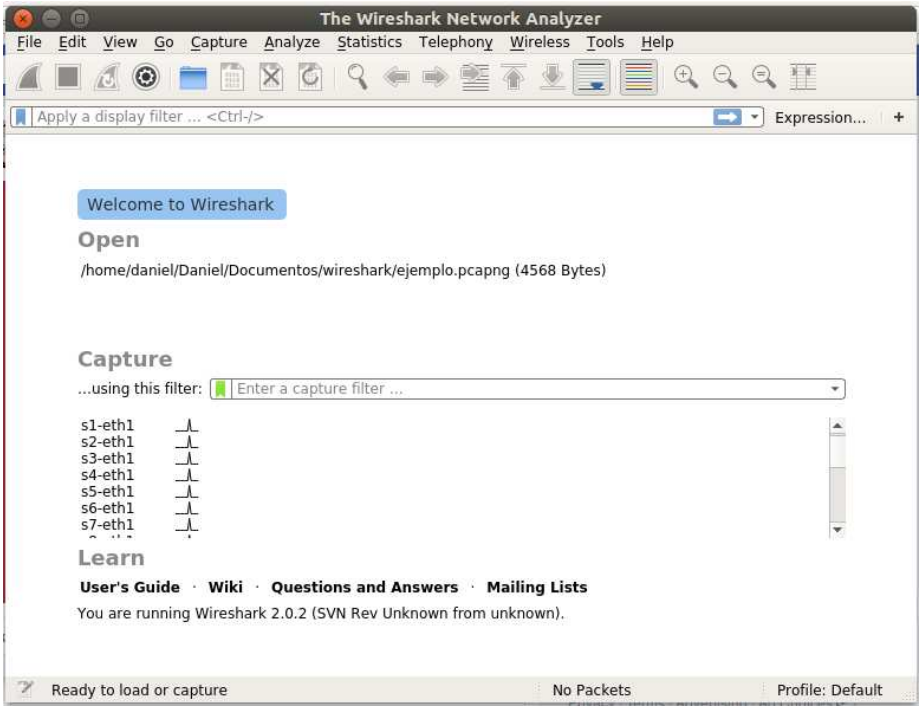


Figura 19. Interfaz gráfica de Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
80	31.287864647	127.0.0.1	127.0.0.1	TCP	66	53910 → 6653 [ACK] Seq=121 Ack=1894 Win=136 Len=0 TSval=476811 TSecr=476801
81	31.345146322	127.0.0.1	127.0.0.1	OpenFlow	189	Type: OFPT_PACKET_OUT
82	31.345171256	127.0.0.1	127.0.0.1	TCP	66	53910 → 6653 [ACK] Seq=121 Ack=2017 Win=136 Len=0 TSval=476826 TSecr=476826
83	31.345415707	127.0.0.1	127.0.0.1	OpenFlow	189	Type: OFPT_PACKET_OUT
84	31.345430374	127.0.0.1	127.0.0.1	TCP	66	53910 → 6653 [ACK] Seq=121 Ack=2140 Win=136 Len=0 TSval=476826 TSecr=476826
85	31.345627890	127.0.0.1	127.0.0.1	OpenFlow	189	Type: OFPT_PACKET_OUT
86	31.345637669	127.0.0.1	127.0.0.1	TCP	66	53910 → 6653 [ACK] Seq=121 Ack=2263 Win=136 Len=0 TSval=476826 TSecr=476826
87	33.348297709	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
88	33.348320687	127.0.0.1	127.0.0.1	TCP	66	53910 → 6653 [ACK] Seq=121 Ack=2271 Win=136 Len=0 TSval=477327 TSecr=477327
89	33.348783188	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
90	33.348809589	127.0.0.1	127.0.0.1	TCP	66	6653 → 53910 [ACK] Seq=2271 Ack=129 Win=1165 Len=0 TSval=477327 TSecr=477327
91	35.350087504	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
92	35.350566627	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
93	35.350669297	127.0.0.1	127.0.0.1	TCP	66	6653 → 53910 [ACK] Seq=2279 Ack=137 Win=1165 Len=0 TSval=477827 TSecr=477827
94	37.352449314	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
95	37.352924526	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
96	37.353027685	127.0.0.1	127.0.0.1	TCP	66	6653 → 53910 [ACK] Seq=2287 Ack=145 Win=1165 Len=0 TSval=478328 TSecr=478328
97	39.353913501	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
98	39.354375024	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
99	39.354495294	127.0.0.1	127.0.0.1	TCP	66	6653 → 53910 [ACK] Seq=2295 Ack=153 Win=1165 Len=0 TSval=478828 TSecr=478828
100	41.356218109	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
101	41.356682566	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
102	41.356782302	127.0.0.1	127.0.0.1	TCP	66	6653 → 53910 [ACK] Seq=2303 Ack=161 Win=1165 Len=0 TSval=479329 TSecr=479329
103	43.357004678	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
104	43.357401178	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY

Figura 20. Tráfico de mensajes OpenFlow en el controlador

3.2 CONTROLADOR FLOODLIGHT

Floodlight es un controlador de clase empresarial, disponible con licencia Apache para casi cualquier propósito. Es apoyado por una gran comunidad de desarrolladores, diseñado para trabajar con switches físicos y virtuales, routers y puntos de acceso compatibles con el protocolo OpenFlow. Es un sistema multiplataforma que funciona sobre la máquina virtual de Java [23].
Se utiliza la versión más actual en el momento de Floodlight, es decir, la versión 1.2.

3.2.1 Ejecución del controlador

Instalado el controlador previamente, se ingresa a la ruta donde se encuentra Floodlight, que por defecto es la siguiente:

cd floodlight

Para iniciar el controlador se utiliza la siguiente sentencia:

java -jar target/floodlight.jar

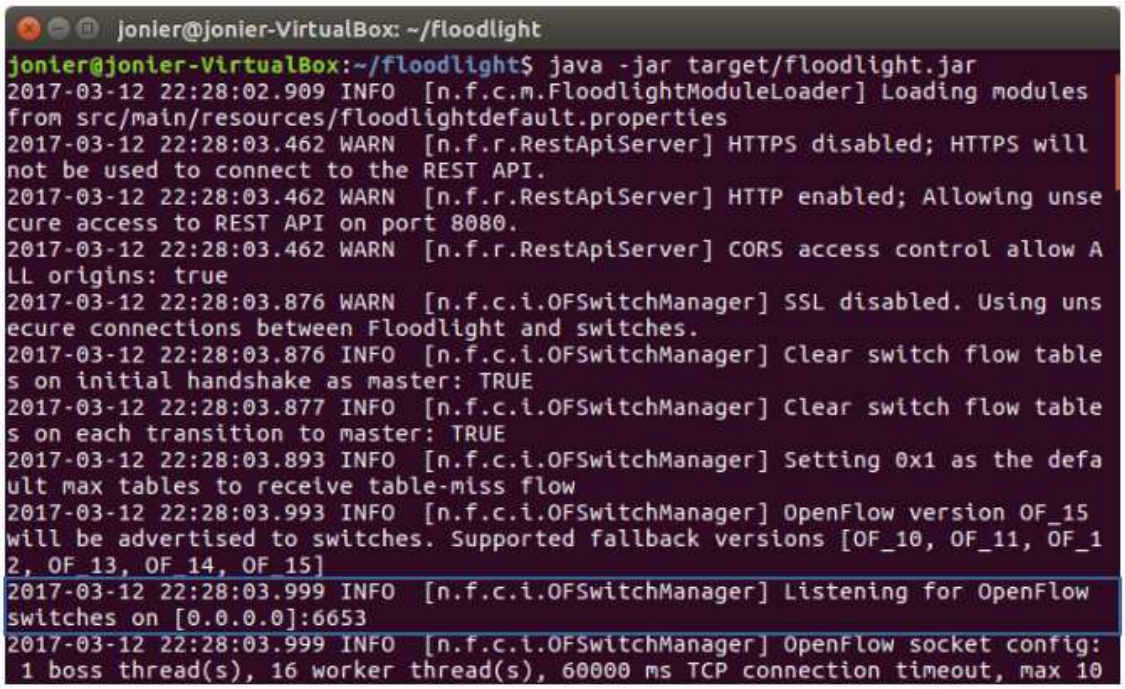


Figura 21. Ejecución del controlador Floodlight

En la imagen se puede observar que el puerto de escucha de Floodlight es el 6653, que es establecido por defecto.

Nota: En algunas ocasiones hay problemas con la ejecución del controlador, por lo que se recomienda escribir la sentencia *"ant"* antes de iniciar el controlador.

3.2.2 Conexión del controlador con Mininet

Para conectar el controlador con Mininet se debe especificar la dirección IP y el puerto de escucha del controlador. Utilizando otra terminal se escribe la siguiente sentencia:

sudo mn --topo single,5 --controller=remote,ip=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13



Figura 22. Creación de una red en Mininet y conectada al controlador Floodlight

En la imagen se observa que el controlador fue agregado de forma exitosa. Si hay algún problema en la conexión del controlador aparecerá un mensaje de error informando que la conexión no se pudo establecer. Cabe resaltar que se utiliza el protocolo OpenFlow 1.3 en dicha instrucción.

3.2.3 Verificación de conexión entre los hosts

pingall

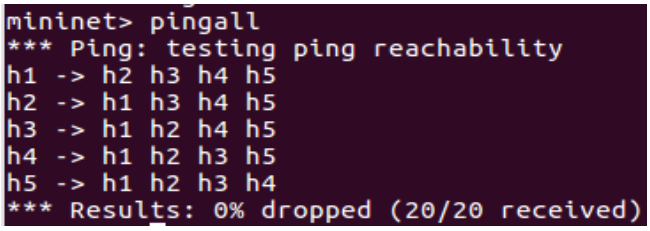


Figura 23. Verificación de conexión mediante Pingall

Como se puede observar en la imagen anterior, la conexión entre los hosts es correcta, lo que indica que el controlador Floodlight está trabajando adecuadamente con Mininet.

3.2.4 Interfaz gráfica de la configuración de red a través de API JSON

Desde el navegador de internet se ingresa a la siguiente dirección:

http://127.0.0.1:8080/ui/pages/topology.html

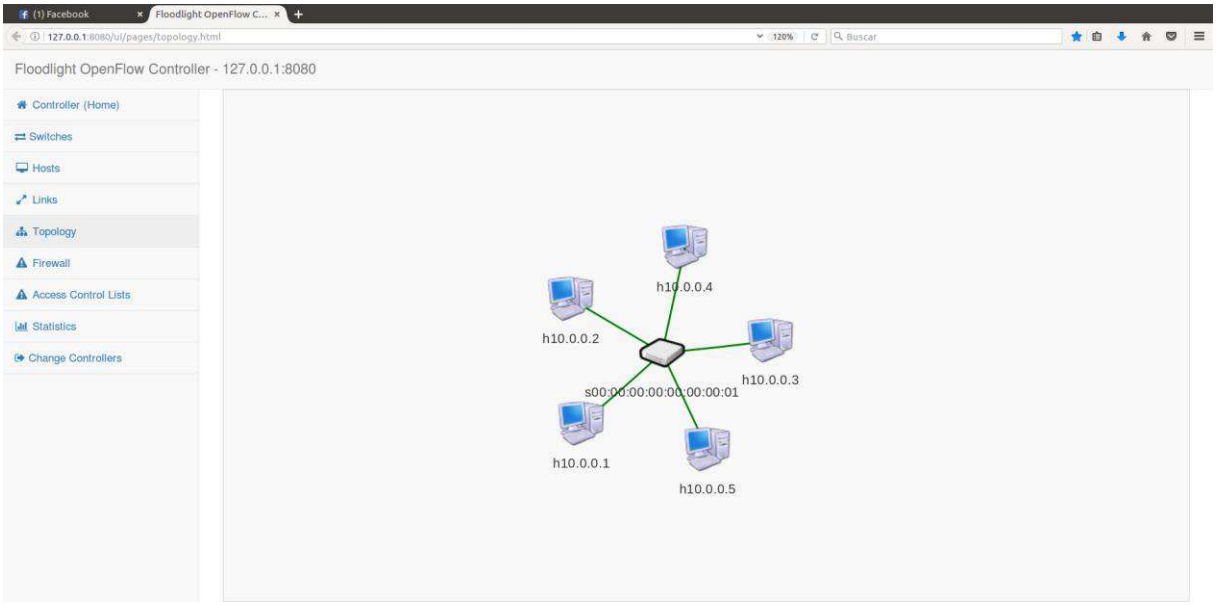


Figura 24. Interfaz gráfica a través de la API JSON del controlador Floodlight

En el menú de la Web API podemos ingresar en la opción “Topology” para visualizar la red de forma gráfica. También aparecen otras opciones útiles donde se puede encontrar las direcciones IP y MAC de los dispositivos de red, las reglas del controlador, la tabla de flujos, entre otras cosas.

3.2.5 Adición de reglas al controlador

Mediante la sentencia “curl” se adicionan, eliminan o editan las reglas del controlador. Es necesario especificar la acción de la regla, las direcciones fuente y destino de los hosts a los cuales se les va a realizar la acción.
En el siguiente ejemplo se impide la comunicación entre los hosts 1 y 2 utilizando la acción “deny”:

curl -X POST -d '{"src-ip":"10.0.0.1/32","dst-ip":"10.0.0.2/32","action":"deny"}' http://localhost:8080/wm/acl/rules/json

Para verificar que la regla fue instalada correctamente, se realiza ping entre los hosts y se comprueba que no es posible la comunicación entre los dos dispositivos:

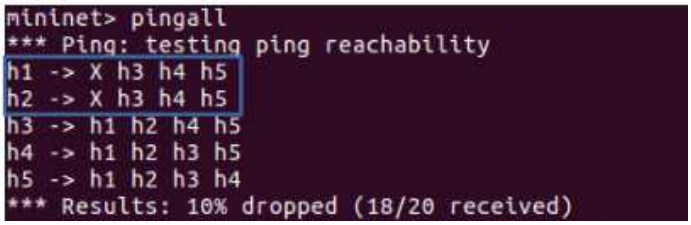


Figura 25. Verificación de conexión entre los hosts

3.2.6 Visualización de reglas instaladas

Utilizando la siguiente instrucción en consola se puede visualizar las reglas instaladas, donde se detalla la acción, el id, las direcciones IP de los hosts fuente y destino, etc.:

curl http://localhost:8080/wm/acl/rules/json | python -mjson.tool

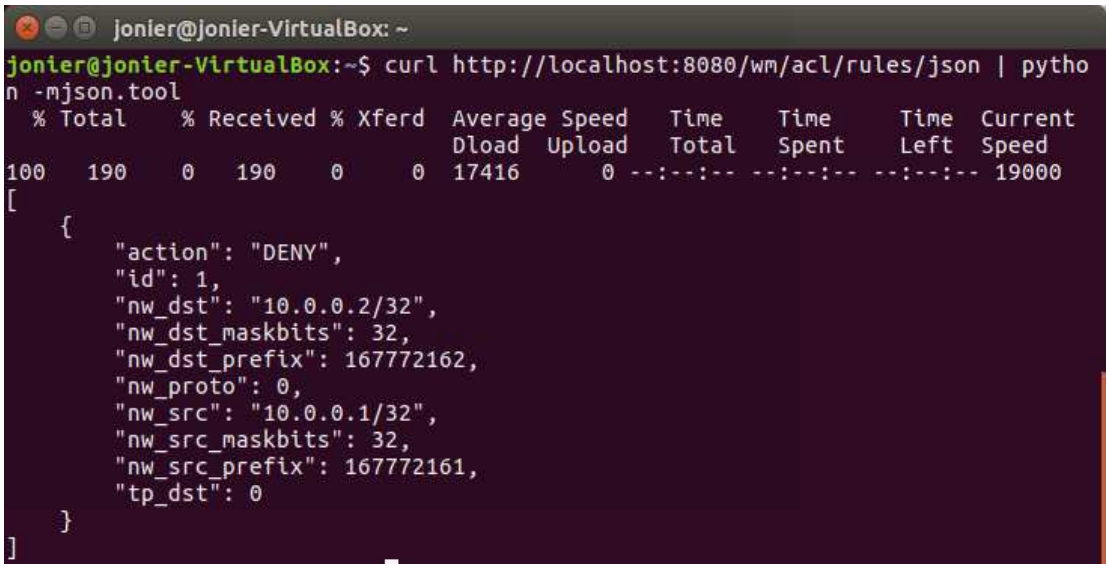


Figura 26. Visualización de reglas del controlador

3.2.7 Eliminación de reglas del controlador

Para eliminar una regla del controlador se escribe la siguiente sentencia (reemplazar *Number_id*):

curl -X DELETE -d '{"ruleid":"Number_id"}'
http://localhost:8080/wm/acl/rules/json

Para eliminar todas las reglas instaladas en el controlador se digita la siguiente sentencia:

curl http://localhost:8080/wm/acl/clear/json

3.3 CONTROLADOR OPENDAYLIGHT

OpenDaylight es un proyecto de código abierto que proporciona una plataforma flexible que puede ser desplegado en cualquier entorno que soporte Java. El controlador contiene APIs abiertas que facilitan su manejo. Tiene como objetivo principal acelerar y acrecentar la difusión de la innovación en el diseño e implementación de un estándar abierto y transparente de redes SDN [24]. Se utiliza la distribution-karaf-0.4.0-Beryllium de OpenDaylight.

3.3.1 Ejecución del controlador

Teniendo previamente instalado el controlador, se ingresa a la ruta donde se encuentra OpenDaylight, que por defecto es la siguiente:

cd distribution-karaf-0.4.0-Beryllium

Para iniciar el controlador se utiliza la siguiente sentencia:

sudo ./bin/karaf

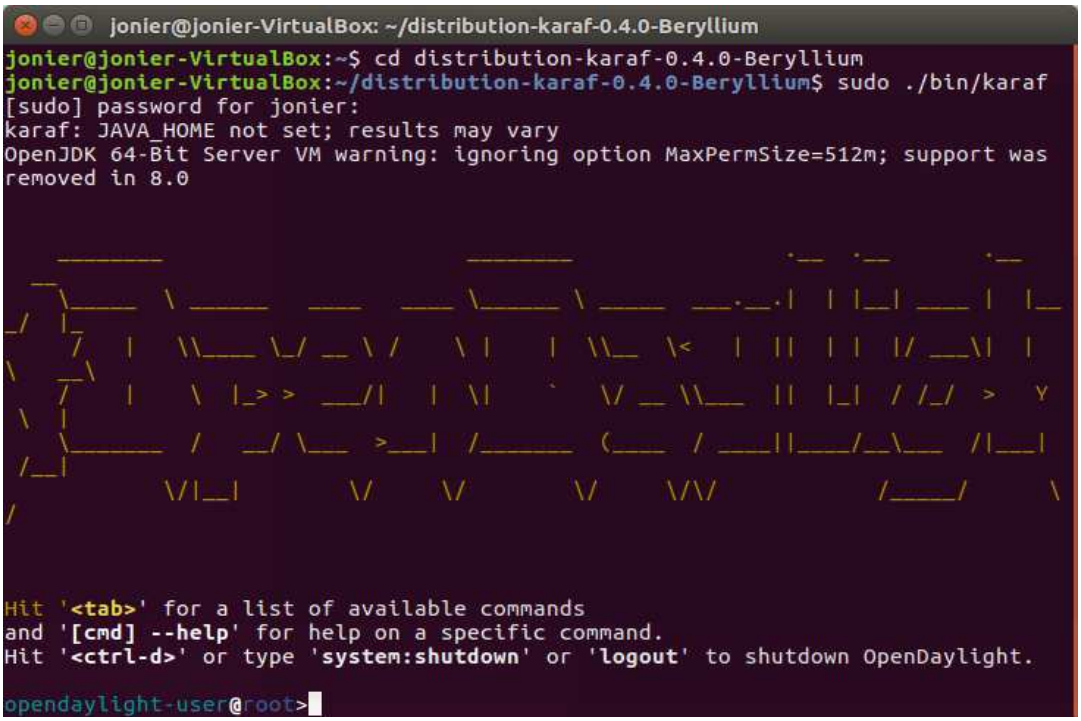


Figura 27. Ejecución del controlador OpenDaylight

3.3.2 Conexión del controlador con Mininet

Para conectar con Mininet se debe especificar la dirección IP y el puerto de escucha del controlador.

Por defecto la dirección IP y el puerto de escucha son los mismos que en el controlador Floodlight (127.0.0.1 y 6653), también funciona si se utiliza el puerto 6633.
Se crea una red sencilla y se conecta el controlador con Mininet (ver figura 28):

sudo mn --topo single,5 --controller=remote,ip=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13

Si existe algún problema con la conexión del controlador Opendaylight, se recomienda implementar las siguientes sentencias:

sudo ./bin/karaf clean

feature:install odl-l2switch-switch odl-mdsal-apidocs odl-dlux-all

3.3.3 Verificación de comunicación entre los hosts

Se realiza pingall para verificar la comunicación entre los hosts, de esa manera se puede comprobar que el controlador OpenDaylight se encuentra funcionando correctamente.

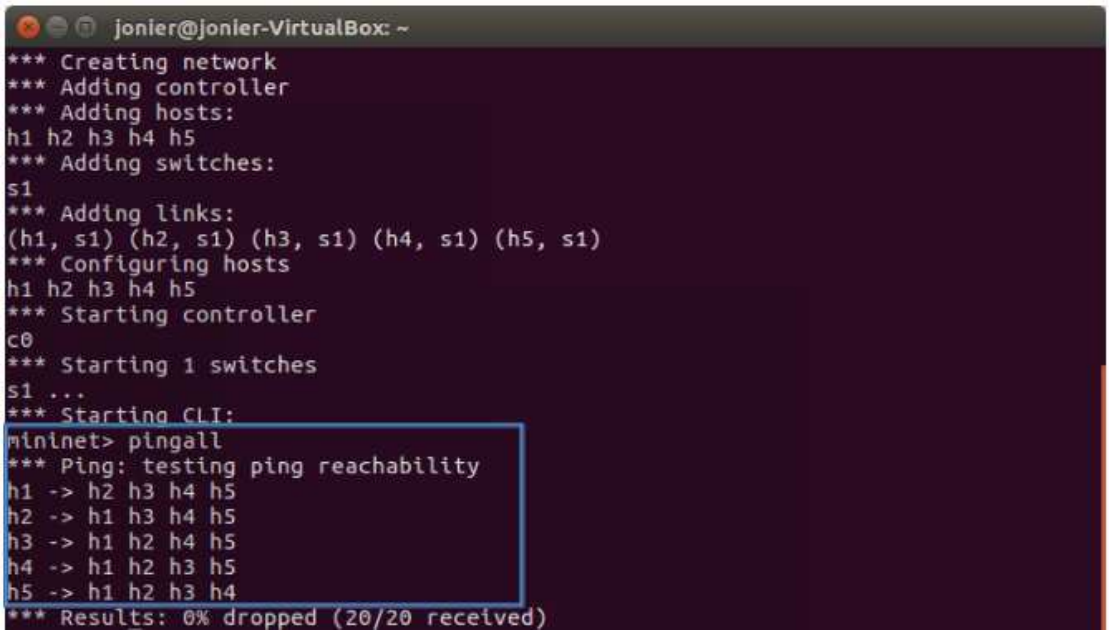


Figura 28. Verificación de la comunicación entre los hosts

3.3.4 Interfaz gráfica de la configuración de red a través de API JSON

Se ingresa a la siguiente dirección web:

http://127.0.0.1:8181/index.html#/topology

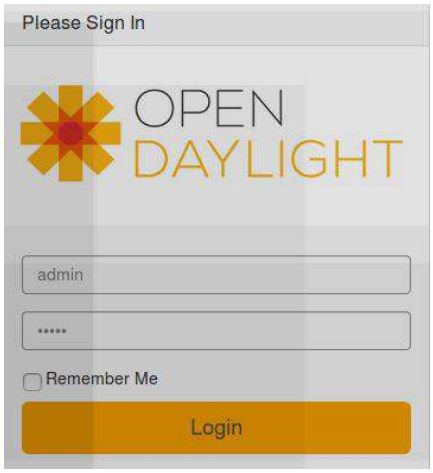


Figura 29. Inicio de sesión en la Web Api de OpenDaylight

Para entrar a la Web Api de OpenDaylight, se requiere ingresar usuario y contraseña, que por defecto para ambas es “admin”.

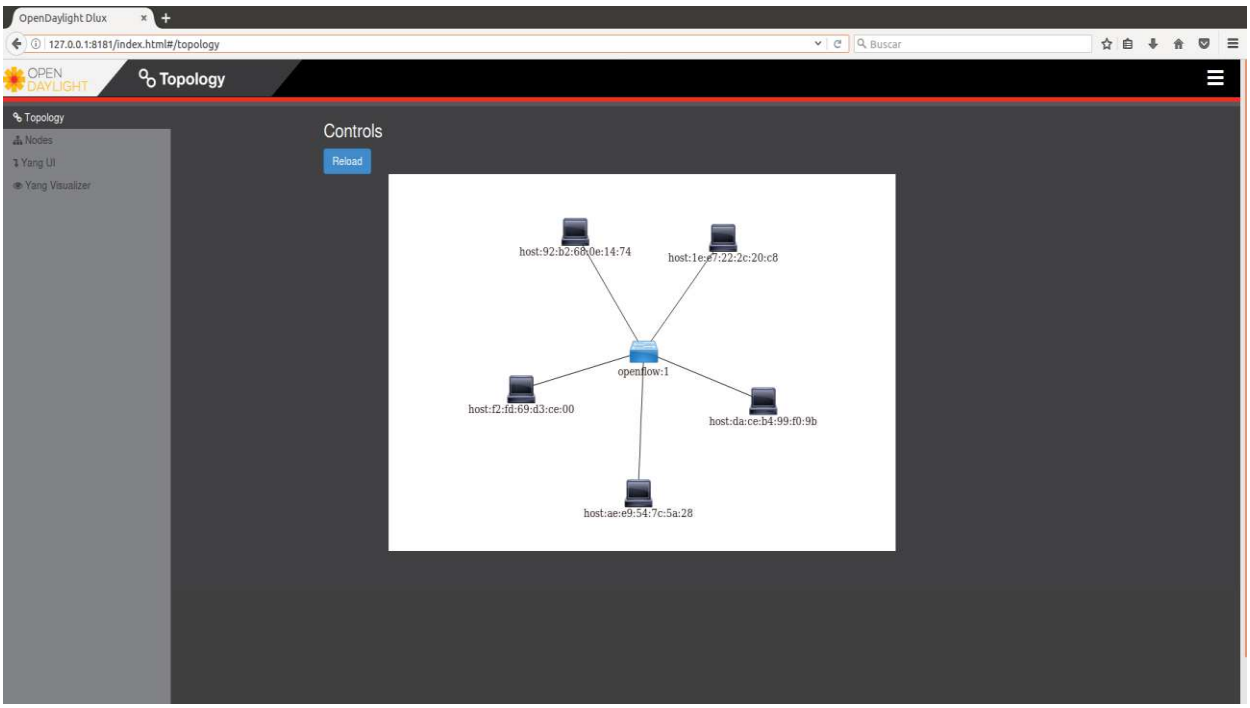


Figura 30. Interfaz gráfica para el controlador OpenDaylight a través de API JSON

En el menú de la Web API podemos ingresar en la opción “Topology” para visualizar la red de forma gráfica. También aparecen otras opciones donde se puede encontrar las direcciones IP y MAC de los dispositivos de red, las reglas del controlador, la tabla de flujos, entre otras cosas.

3.3.5 Adición de flujos al controlador

Al igual que en Floodlight, se implementa la instrucción "*curl*" para agregar reglas al controlador.

- Host fuente 10.0.0.3, host destino 10.0.0.1

  curl --user "admin":"admin" -H "Accept: application/xml" -H "Content-type: application/xml" -X PUT http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1 -d '<?xml version="1.0" encoding="UTF-8" standalone="no"?><flow xmlns="urn:opendaylight:flow:inventory"><priority>32767</priority><flow-name>Flujo1</flow-name><match><in-port>3</in-port><ipv4-destination>10.0.0.1/32</ipv4-destination><ipv4-source>10.0.0.3/32</ipv4-source><ethernet-match><ethernet-type><type>2048</type></ethernet-type></ethernet-match></match><id>1</id><table_id>0</table_id><instructions><instruction><order>0</order><apply-actions><action><order>0</order><output-action><output-node-connector>1</output-node-connector></output-action></action></apply-actions></instruction></instructions></flow>'

- Host fuente 10.0.0.1, host destino 10.0.0.3

  curl --user "admin":"admin" -H "Accept: application/xml" -H "Content-type: application/xml" -X PUT http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/2 -d '<?xml version="1.0" encoding="UTF-8" standalone="no"?><flow xmlns="urn:opendaylight:flow:inventory"><priority>32767</priority><flow-name>Flujo 2</flow-name><match><in-port>1</in-port><ipv4-destination>10.0.0.3/32</ipv4-destination><ipv4-source>10.0.0.1/32</ipv4-source><ethernet-match><ethernet-type><type>2048</type></ethernet-type></ethernet-match></match><id>2</id><table_id>0</table_id><instructions><instruction><order>0</order><apply-actions><action><order>0</order><output-action><output-node-connector>3</output-node-connector></output-action></action></apply-actions></instruction></instructions></flow>'

La zona subrayada contiene la información del flujo que se está agregando. En este aparece el puerto de entrada y salida, la dirección fuente y destino, la prioridad, entre otros.

Alternativamente, se puede crear flujos a través de Open vSwitch (OVS) con la siguiente sentencia:

sudo ovs-ofctl add-flow s1 priority=500,in_port=1,actions=output:3

Si la red de Mininet se crea con un protocolo OpenFlow, agregar a la sentencia: -O OPENFLOW13

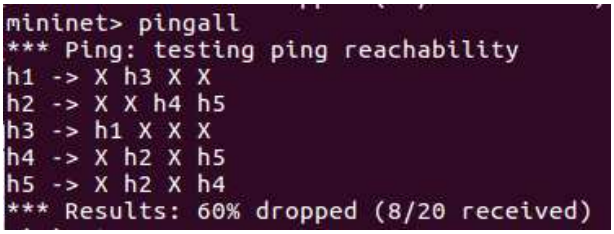


Figura 31. Pingall con los nuevos flujos instalados

Como se puede observar en la Figura 31, al agregar el flujo con el código alternativo de OVS, los hosts h1 y h3 únicamente tienen comunicación entre ellos dos debido a que se creó dos flujos que hacen que toda información proveniente del hosts 1 sea enviada por el puerto 3 que está conectado al host 3, y así mismo, toda información proveniente del host 3 sea enviada por el puerto 1, que se encuentra conectado al host 1.

3.3.6 Visualización de flujos instalados

En el navegador de internet se ingresa la siguiente dirección Web:

http://localhost:8181/restconf/config/opendaylight-inventory:
nodes/node/openflow:NumeroSwitch/table/0/flow/NumeroFlujo

Donde se debe reemplazar: *NumeroSwitch* y *NumeroFlujo.*


```
-<flow>
  <id>1</id>
 -<instructions>
  -<instruction>
    <order>0</order>
   -<apply-actions>
    -<action>
      <order>0</order>
     -<output-action>
       <output-node-connector>2</output-node-connector>
      </output-action>
    </action>
   </apply-actions>
  </instruction>
 </instructions>
 <priority>32767</priority>
-<match>
 -<ethernet-match>
  -<ethernet-type>
    <type>2048</type>
   </ethernet-type>
  </ethernet-match>
  <ipv4-source>10.0.0.3/32</ipv4-source>
  <ipv4-destination>10.0.0.1/32</ipv4-destination>
  <in-port>3</in-port>
 </match>
 <table_id>0</table_id>
 <flow-name>Load Balance 1</flow-name>
</flow>
```

```
-<flow>
  <id>2</id>
 -<instructions>
  -<instruction>
    <order>0</order>
   -<apply-actions>
    -<action>
      <order>0</order>
     -<output-action>
       <output-node-connector>3</output-node-connector>
      </output-action>
    </action>
   </apply-actions>
  </instruction>
 </instructions>
 <priority>32767</priority>
-<match>
 -<ethernet-match>
  -<ethernet-type>
    <type>2048</type>
   </ethernet-type>
  </ethernet-match>
  <ipv4-source>10.0.0.1/32</ipv4-source>
  <ipv4-destination>10.0.0.3/32</ipv4-destination>
  <in-port>2</in-port>
 </match>
 <table_id>0</table_id>
 <flow-name>Load Balance 2</flow-name>
</flow>
```


Figura 32. Visualización de flujos instalados en el controlador Opendaylight

```
daniel@daniel-Aspire-4560:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s3
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=312.351s, table=0, n_packets=2, n_bytes=196, priority=3276
7,ip,in_port=2,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:3
 cookie=0x0, duration=312.351s, table=0, n_packets=2, n_bytes=196, priority=3276
7,ip,in_port=3,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:2
 cookie=0x2b0000000000000b, duration=307.342s, table=0, n_packets=127, n_bytes=1
0795, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b0000000000000d, duration=301.511s, table=0, n_packets=6, n_bytes=364
, priority=2,in_port=2 actions=output:1,output:3
 cookie=0x2b0000000000000e, duration=301.511s, table=0, n_packets=8, n_bytes=560
, priority=2,in_port=1 actions=output:2,output:3
 cookie=0x2b0000000000000f, duration=301.511s, table=0, n_packets=6, n_bytes=364
, priority=2,in_port=3 actions=output:2,output:1,CONTROLLER:65535
 cookie=0x2b0000000000000b, duration=307.342s, table=0, n_packets=0, n_bytes=0,
priority=0 actions=drop
```

Figura 33. Flujos instalados en el controlador

Alternativamente, es posible visualizar los flujos instalados a través de OVS en una terminal aparte:

sudo ovs-ofctl dump-flows s1

```
jonier@jonier-VirtualBox:~$ sudo ovs-ofctl  dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=1007.823s, table=0, n_packets=49, n_bytes=2814, idle_age=5
79, priority=500,in_port=1 actions=output:3
 cookie=0x0, duration=670.358s, table=0, n_packets=33, n_bytes=1834, idle_age=57
1, priority=500,in_port=3 actions=output:1
 cookie=0x0, duration=1014.767s, table=0, n_packets=172, n_bytes=11884, idle_age
=563, priority=0 actions=CONTROLLER:65535
```

Figura 34. Flujos instalados en el controlador

3.3.7 Eliminación de reglas del controlador

Para eliminar una regla se utiliza la siguiente sentencia:

curl -X DELETE -H "Content-Type: application/json" -H "Accept: application/json" --user admin:admin http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:NumeroSwitch/table/0 /flow/NumeroFlujo

Donde se debe reemplazar: *NumeroSwitch* y *NumeroFlujo.*

Alternativamente, es posible eliminar los flujos instalados a través de OVS:

sudo ovs-ofctl add-flow s1 priority=32768,actions=drop

3.4 CONTROLADOR POX

POX es una plataforma de código abierto para aplicaciones de control de redes definidas por software (SDN) basadas en Python. Como otros controladores SDN de OpenFlow, permite un rápido desarrollo y prototipos. El controlador POX es un proyecto que surge del antiguo controlador NOX [25]. Se utiliza la versión 0.2.0 del controlador POX.

3.4.1 Ejecución del controlador

El controlador POX cuenta con un número reducido de módulos que cumplen con una función específica. Para poder realizar pruebas con este controlador, es necesario ejecutar alguno de estos módulos (dependiendo de la tarea que se desee realizar). Estos módulos se encuentran hechos en Python.

Teniendo el controlador instalado previamente, se ingresar a la ruta donde se encuentra instalado el controlador POX, que por defecto se puede acceder desde la terminal:

cd pox

Para iniciar el controlador se utiliza la siguiente sentencia:

./pox.py openflow.discovery forwarding.l3_learning log.level --DEBUG

```
daniel@daniel-Aspire-4560:~/pox$ ./pox.py openflow.discovery forwarding.l3_learning log.level --DEBUG
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.12/Nov 19 2016 06:48:10)
DEBUG:core:Platform is Linux-4.4.0-71-generic-x86_64-with-Ubuntu-16.04-xenial
DEBUG:forwarding.l3_learning:Up...
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

Figura 35. Ejecución del controlador POX

En la imagen se puede observar que el puerto de escucha de este controlador es el 6633, así como también la ejecución de dos módulos:

Forwarding.l3_learning: Este componente no define el comportamiento de un router, pero tampoco de un switch de nivel 2. POX usa una biblioteca de paquetes para examinar y elaborar solicitudes y respuestas ARP.

Openflow.discovery: Este componente envía mensajes LLDP de conmutadores OpenFlow para que pueda descubrir la topología de la red.

Una instrucción alternativa para ejecutar el controlador POX es la siguiente:

sudo ~/pox/pox.py forwarding.l3_learning

3.4.2 Conexión del controlador con Mininet

Para conectar con Mininet se debe especificar la dirección IP y el puerto de escucha del controlador. Por defecto la dirección IP es la dirección de local host 127.0.0.1 y el puerto de escucha el 6633.

sudo mn --topo=single,5 --controller=remote,ip=127.0.0.1,port=6633

La creación de la red se puede observar en la figura 36.

3.4.3 Verificación de conexión entre los hosts

Se utiliza la herramienta pingall para verificar la comunicación entre los dispositivos finales de la red (hosts):

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
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 36. Verificación de la comunicación entre los hosts

Como la comunicación fue correcta entre todos los hosts, indica que el controlador está comandando la red sin problemas.

Al realizar ping o pingall en la terminal donde se está ejecutado la red simulada, se generan mensajes ARP en la consola donde se está ejecutando el controlador, con el objetivo de determinar las direcciones de los diferentes hosts en la red. Estos mensajes ARP son enviados debido a la función que cumplen los dos módulos seleccionados al ejecutar el controlador (estos dos fueron explicados con anterioridad). A continuación se muestran los mensajes ARP enviados por el controlador al realizar pingall en la terminal de Mininet:

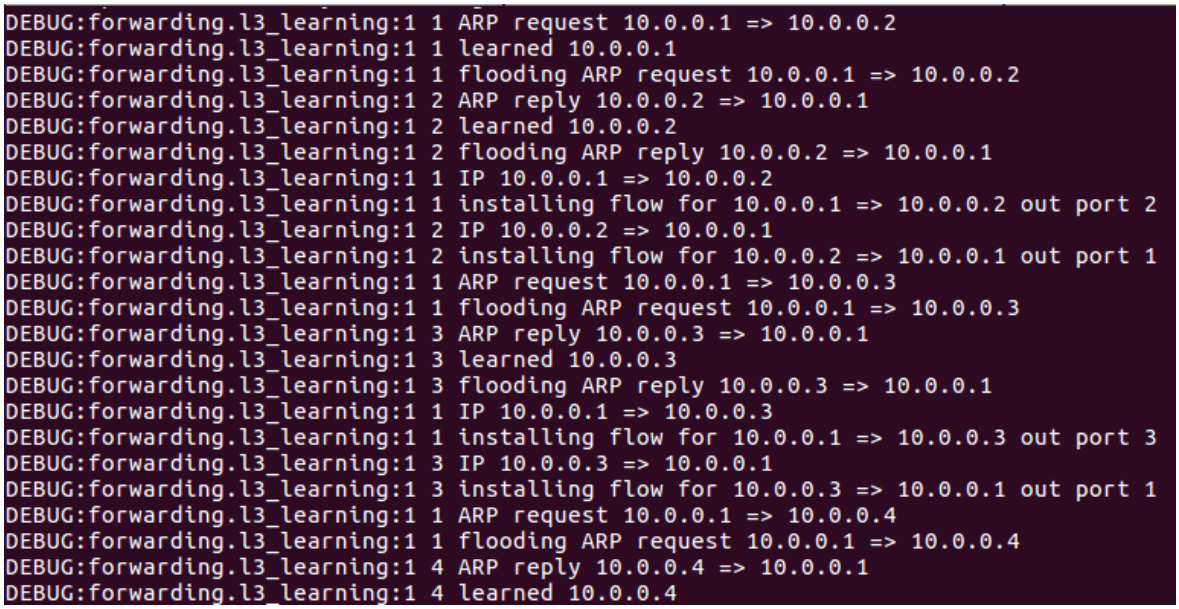


Figura 37. Solicitudes ARP y Aprendizaje de los elemento de la red

Finalmente, para el controlador POX no existe una Web Api hasta el momento como en los dos controladores anteriores.

3.5 COMPARACIÓN ENTRE LOS CONTROLADORES

Mediante solicitudes de ping y medición de ancho de banda, se realiza un análisis comparativo entre los tres controladores trabajados anteriormente. Para ello se utiliza una red tipo árbol creada con la siguiente sentencia en Mininet:

sudo mn --topo=tree,depth=3,fanout=3 --controller=remote,ip=127.0.0.1,port=6653

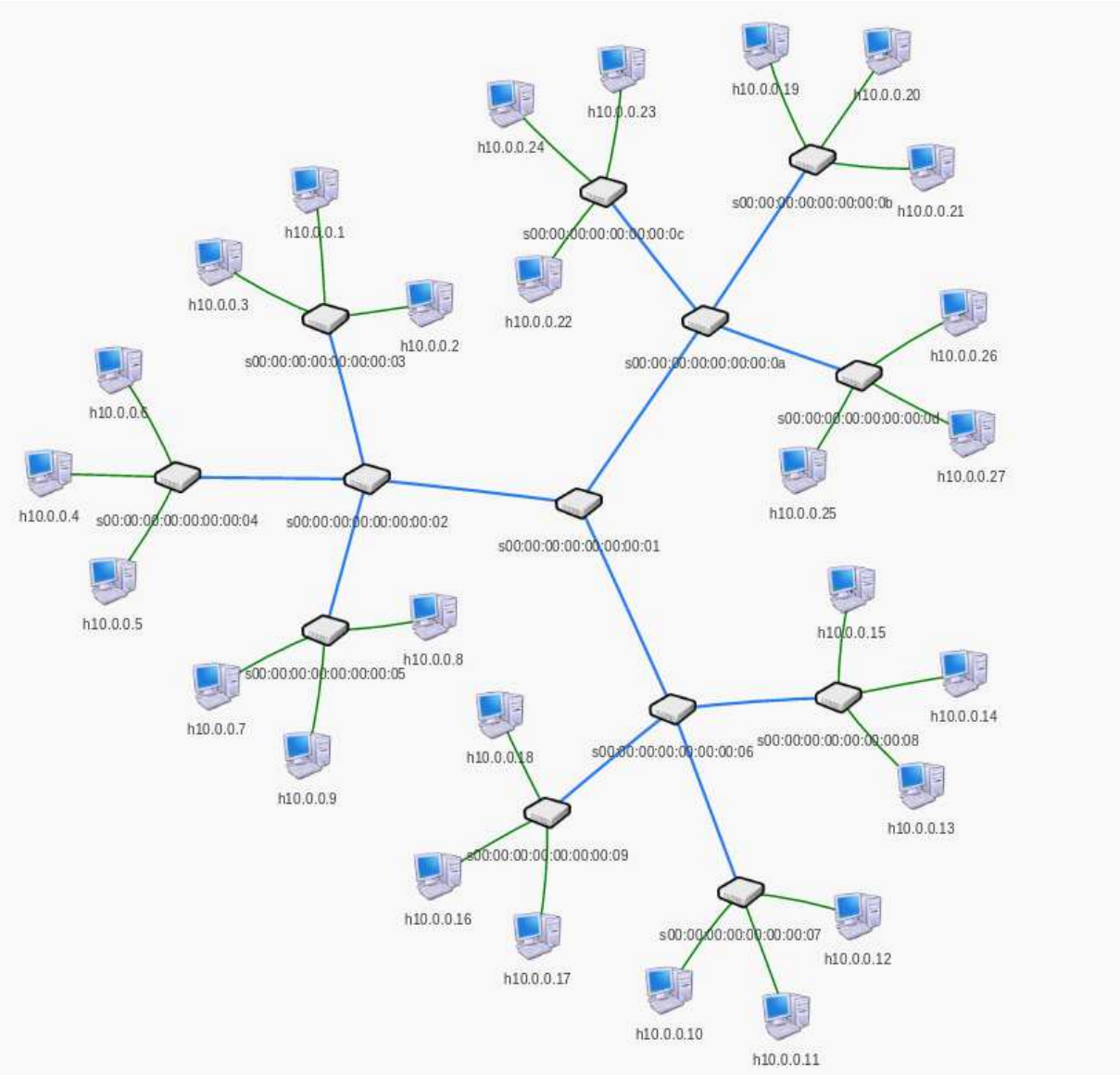


Figura 38. Visualización de la red de prueba para los diferentes controladores

3.5.1 Solicitudes de ping

Se envían 100 solitudes de ping desde h1 a h11 en cada uno de los controladores y se obtienen las estadísticas de tiempo.

3.5.1.1 Controlador Floodlight

```
--- 10.0.0.11 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99034ms
rtt min/avg/max/mdev = 0.037/0.164/9.583/0.947 ms
```

Figura 39. Estadísticas de solicitudes ping con controlador Floodlight

3.5.1.2 Controlador OpenDaylight

```
--- 10.0.0.11 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99061ms
rtt min/avg/max/mdev = 0.245/0.477/3.732/0.422 ms
```

Figura 40. Estadísticas de solicitudes ping con controlador OpenDaylight

3.5.1.3 Controlador POX

```
--- 10.0.0.11 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99015ms
rtt min/avg/max/mdev = 0.051/0.899/83.332/8.284 ms
```

Figura 41. Estadísticas de solicitudes ping con controlador POX

3.5.1.4 Análisis de resultados

A continuación se tabula y grafica las estadísticas de las solicitudes de ping obtenidas en el ítem anterior:

Controlador	Tiempo mínimo (ms)	Tiempo promedio (ms)	Tiempo máximo (ms)
Floodlight	0,037	0,164	9,583
Opendaylight	0,245	0,477	3,732
POX	0,051	0,899	83,332

Tabla 1. Estadísticas de solicitudes ping de los diferentes controladores

Cabe aclarar que entre menores sean los tiempos, mejor es la respuesta del controlador.

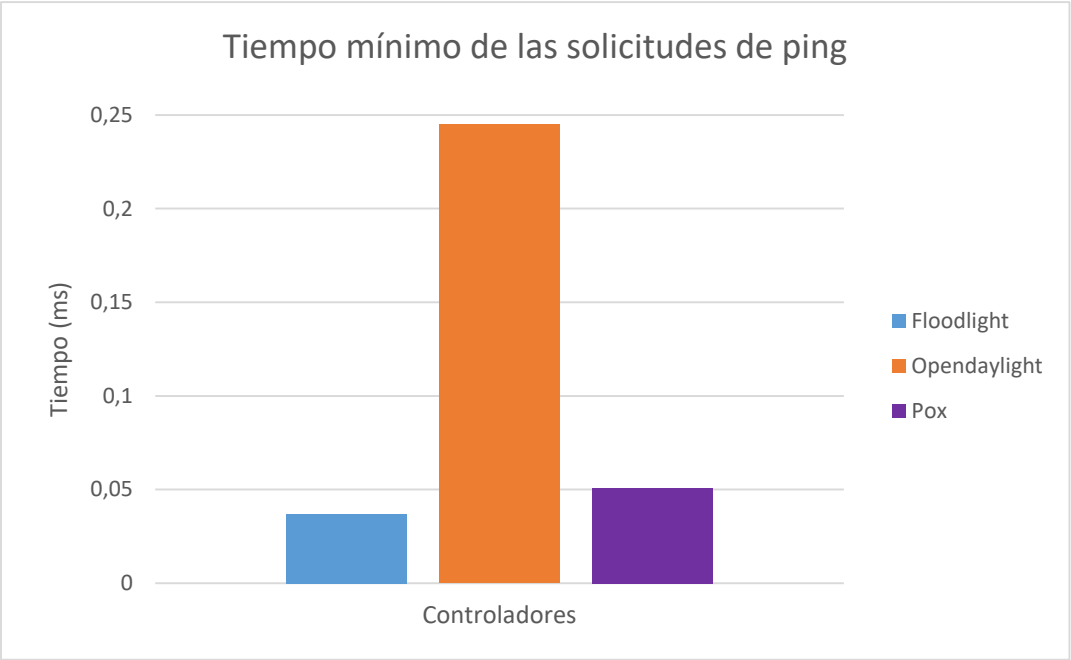


Figura 42. Gráfico de los tiempos mínimos de los controladores

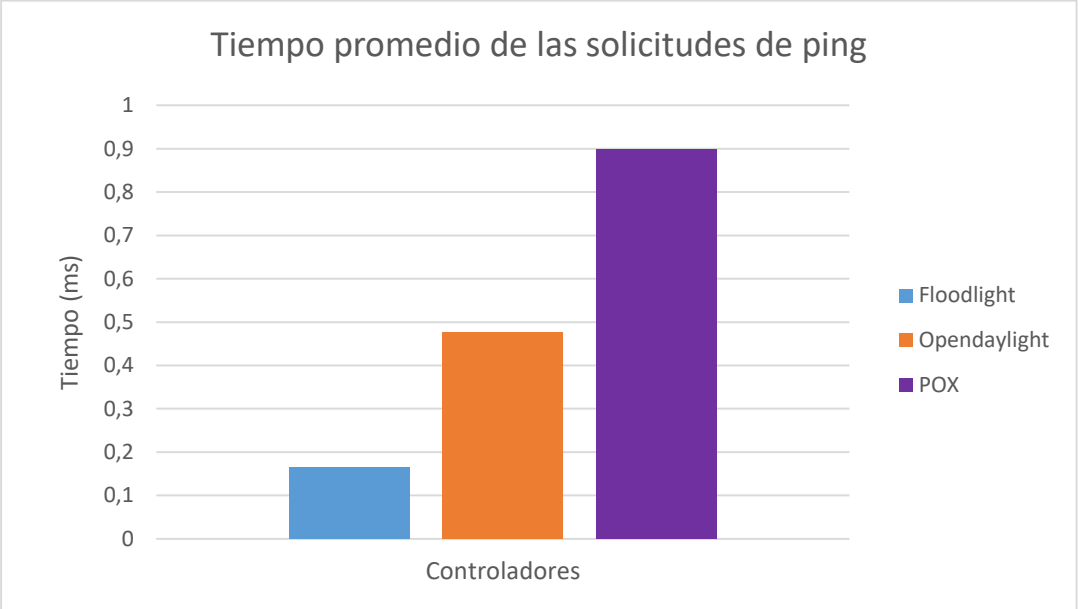


Figura 43. Gráfico de los tiempos promedios de los controladores

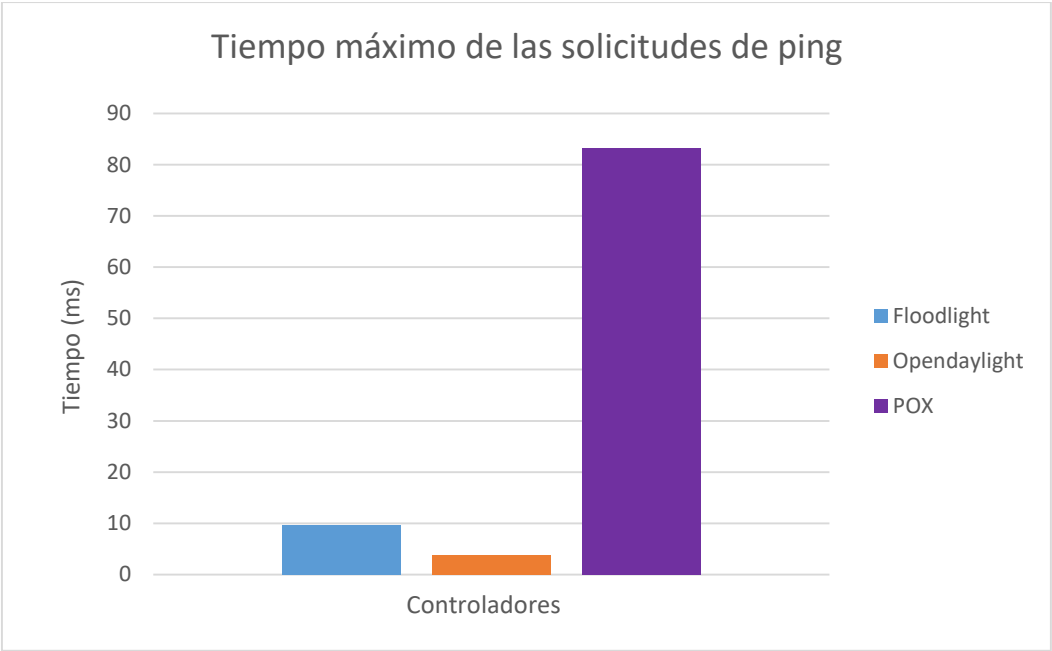


Figura 44. Gráfico de los tiempos máximos de los controladores

Como se puede observar, el controlador Floodlight obtuvo mejor rendimiento frente a los otros 2 controladores debido a que su tiempo promedio fue mucho menor. Así mismo, el controlador POX obtuvo la peor respuesta debido a que su tiempo promedio fue mayor, que se debe principalmente a que el tiempo del primer ping entre los hosts es demasiado alto.

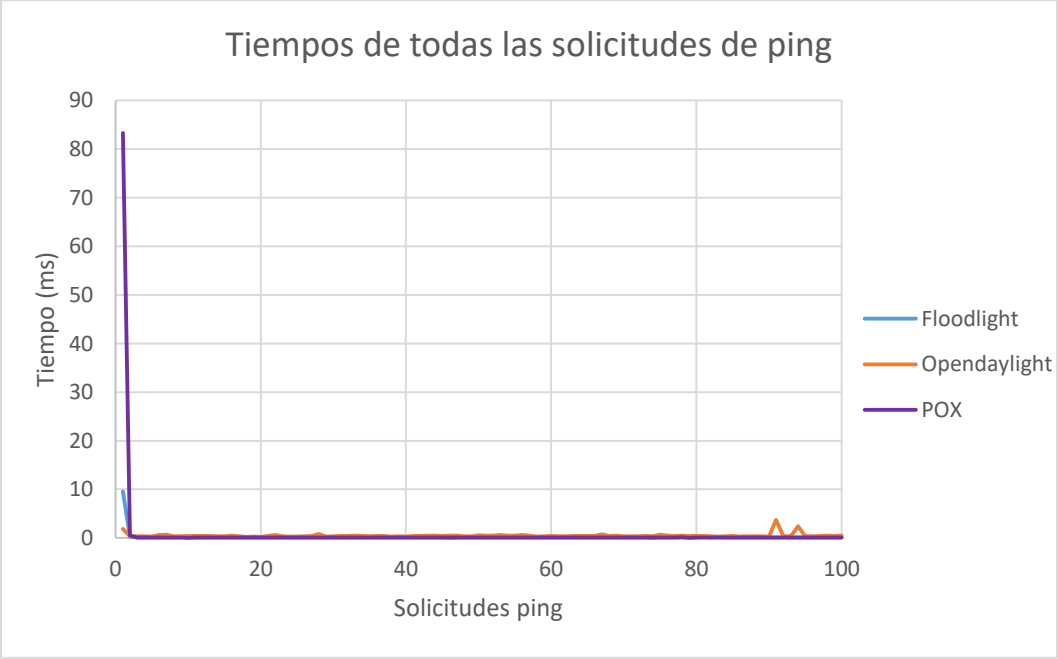


Figura 45. Tiempo de todos los ping transmitidos

Cómo se puede ver en el anterior gráfico, la primera solicitud enviada con el controlador POX tuvo un tiempo demasiado grande (83,3 ms), seguido del controlador Floodlight (9,58 ms) y luego el controlador OpenDaylight (1,89 ms). Por esta razón las estadísticas promedio en el controlador POX fueron inferiores frente a los otros dos controladores. Sin embargo, si se visualiza el comportamiento de las demás solicitudes de ping, sin tener en cuenta el primer ping, podremos ver que el comportamiento del controlador POX es mucho mejor de lo que aparenta de acuerdo a las estadísticas obtenidas.

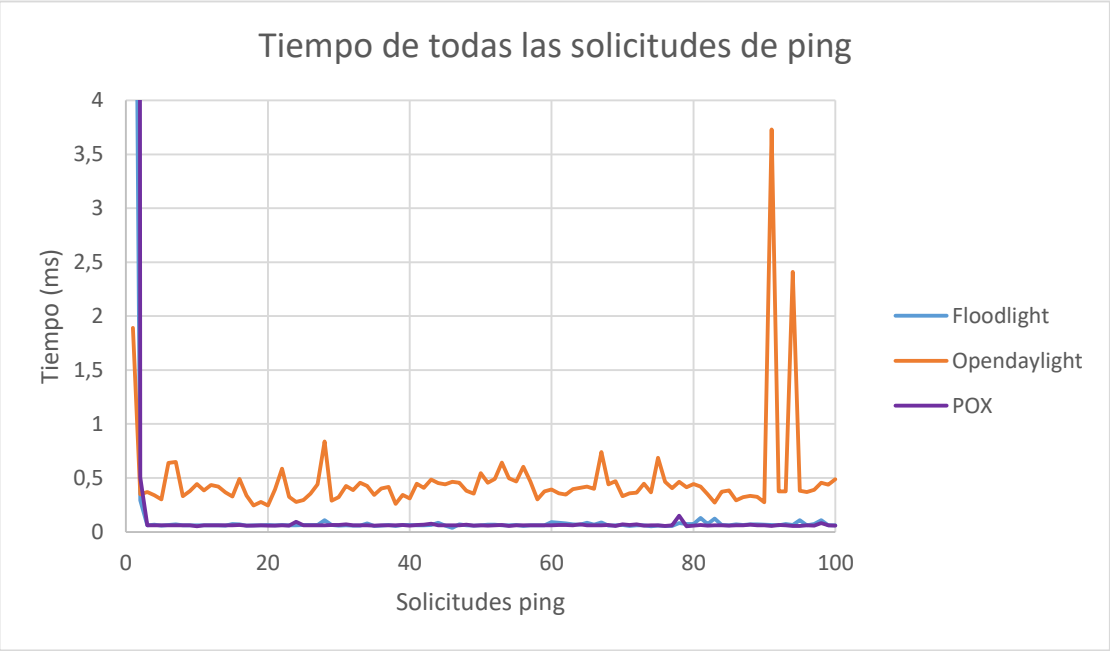


Figura 46. Tiempo de las Solicitudes de ping, sin tener en cuenta el primer ping

De esta manera se puede ver que la respuesta de los controladores POX y Floodlight es casi la misma sin contar la primera solicitud de ping, mientras que para el controlador OpenDaylight su comportamiento es muy inferior, por lo que si se realiza una prueba de muchas más solicitudes de Ping, la respueta del controlador POX será mucho mejor que la del OpenDaylight, incluyendo el tiempo de su primer ping.

Si se saca un promedio de los tiempos tomados en los ping enviados, sin tener en cuenta el primero transmitido, se obtienen los siguientes resultados:

Tiempo promedio de los 99 ping		
Floodlight	OpenDaylight	POX
0,069 ms	0,463 ms	0,067 ms

Tabla 2. Tiempo promedio de las 99 solicitudes de ping

Efectivamente la respuesta del controlador POX es muy buena, prácticamente igual que la del controlador Floodlight. Por este motivo se puede decir que el controlador POX tiene una respuesta en retardos buena cuando se envían muchos paquetes.

En conclusión, el controlador Floodlight fue el que mejor comportamiento tuvo en retardos al realizar las pruebas de solicitudes de ping. Por otro lado, el controlador OpenDaylight obtuvo la peor respuesta en retardos, y la razón principal se debe a que los otros dos controladores poseen módulos para encontrar caminos más cortos y de balance de carga, es por esa razón que el tiempo del primer ping es mayor en estos dos controladores que en el de OpenDaylight. En esta primera solicitud se realizan operaciones para encontrar las mejores rutas para los paquetes. Si se envían muy pocas solicitudes de ping, el controlador OpenDaylight tendrá la mejor respuesta, debido al poco tiempo que le toma realizar su primer ping.

3.5.2 Ancho de banda

Para las pruebas de ancho de banda se utiliza la misma red, así como también los mismos hosts de prueba.

3.5.2.1 Controlador Floodlight

```
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['31.1 Gbits/sec', '31.0 Gbits/sec']
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['31.8 Gbits/sec', '31.8 Gbits/sec']
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['31.8 Gbits/sec', '31.8 Gbits/sec']
```

Figura 47. Ancho de banda entre h1 y h11 utilizando el controlador Floodlight

3.5.2.2 Controlador OpenDaylight

```
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['316 Mbits/sec', '324 Mbits/sec']
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['303 Mbits/sec', '308 Mbits/sec']
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['315 Mbits/sec', '318 Mbits/sec']
```

Figura 48. Ancho de banda entre h1 y h11 utilizando el controlador OpenDaylight

3.5.2.3 Controlador POX

```
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['30.8 Gbits/sec', '30.9 Gbits/sec']
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['31.6 Gbits/sec', '31.6 Gbits/sec']
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['31.4 Gbits/sec', '31.4 Gbits/sec']
```

Figura 49. Ancho de banda entre h1 y h11 utilizando el controlador POX

3.5.2.4 Análisis de resultados

A continuación se tabula y grafica el promedio del ancho de banda de las mediciones obtenidas en el ítem anterior:

Controlador	Ancho de banda promedio (Gbit/s)
Floodlight	31,57
OpenDaylight	0,31
POX	31,27

Tabla 3. Ancho promedio de los diferentes controladores

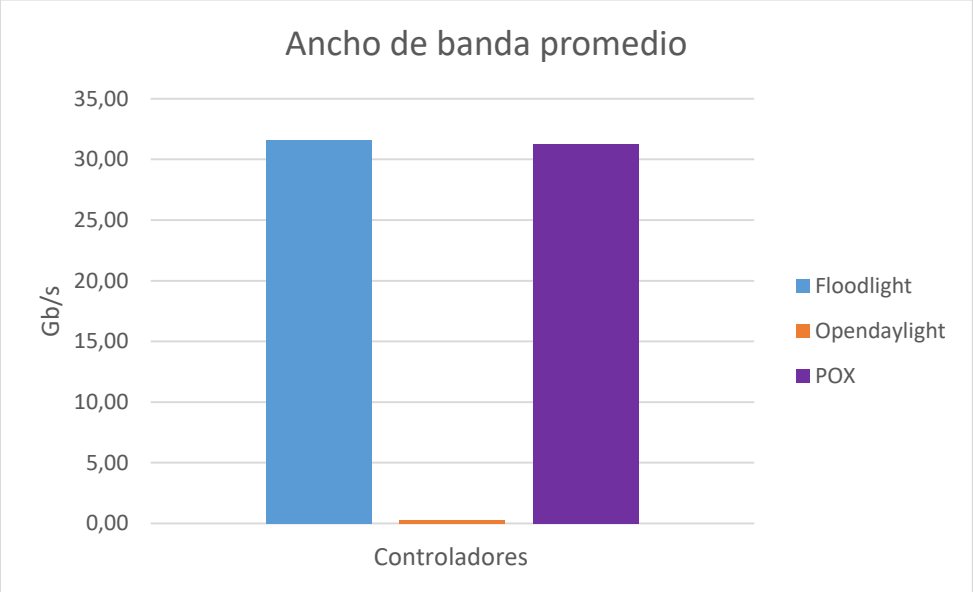


Figura 50. Gráfico de ancho de banda promedio de los diferentes controladores.

Como se puede observar en el gráfico, el ancho de banda del controlador OpenDaylight es muy pobre respecto al proporcionado por los otros dos controladores. Por otro lado el ancho de banda

del controlador Floodlight y del POX es prácticamente el mismo, solo por un par de Mb/s es mayor el de Floodlight.

3.5.3 Conclusiones

Para las pruebas siguientes de este trabajo, se eligen los controladores Floodlight y OpenDaylight, esto debido a que en la actualidad se utilizan mucho más y su soporte es mayor que el del controlador POX. Este último controlador tiene muy poco soporte en documentación y desarrollos, como por ejemplo no posee una REST API (Interfaz online).

Por otro lado, escogemos estos dos controladores debido a que el Floodlight obtuvo el mejor desempeño y el OpenDaylight el peor, de tal manera que mediante el uso de algoritmos se pueda mejorar el rendimiento de este último controlador a la par de Floodlight tanto en ancho de banda como en retardos.

Por último, se adiciona una tabla comparativa de algunos controladores SDN donde está incluido los tres controladores estudiados en este documento.

	Beacon	Floodlight	NOX	POX	ODL
Soporte Openflow	OF v1.0	OF v1.O,v1.3 OF v1.2,v1.4 (beta)	OF v1.0	OF v1.0	OF v1.0,v1.3
Lenguaje de desarrollo	Java	Java	C++	Python	Java
REST API	No	Si	No	No	Si
GUI	Web	Web	Python+,QT4	Python+, QT4,Web	Web
Plataformas soportadas	Linux,Mac OS,Windows y Android	Linux, Mac OS,Windows	Linux	Linux, MacOS,Windows	Linux, MacOS, Windows
Soporte OpenStack	No	Si	No	No	Si
Multiprocesos	Si	Si	Si	No	Si
Código abierto	Si	Si	Si	Si	
Tiempo en el mercado (aprox)	5 años	3 años	7 años	2 años	1 año
Documentación	Buena	Buena	Media	Pobre	Media

Figura 51. Comparación entre controladores SDN [26].

CAPÍTULO 4. IMPLEMENTACIÓN DE ALGORITMO PARA BALANCE DE CARGA

El balanceo de carga aplicado al manejo de las redes definidas por software no se encuentra muy alejado del funcionamiento en una red tradicional. Para el caso de las redes SDN, el controlador es quien se encarga de realizar el proceso de selección de la ruta más apropiada, a partir de protocolos de enrutamiento para el envío de paquetes de un punto a otro a través de una red, teniendo en cuenta la carga de los enlaces dentro de esta. Para la aplicación del balanceo de carga en esta sección se implementarán dos códigos en Python (uno con el controlador Floodlight y otro con el controlador OpenDaylight) basados en la obtención del camino más corto a través del algoritmo de Dijkstra.

4.1 USANDO CONTROLADOR OPENDAYLIGHT

Se crea una red personalizada por código mediante Python. La personalización de la red está basada principalmente en muchas opciones de caminos que permitan a los paquetes tomar diferentes rutas, y con ello observar la obtención del camino más corto y el balanceo de carga en la red.

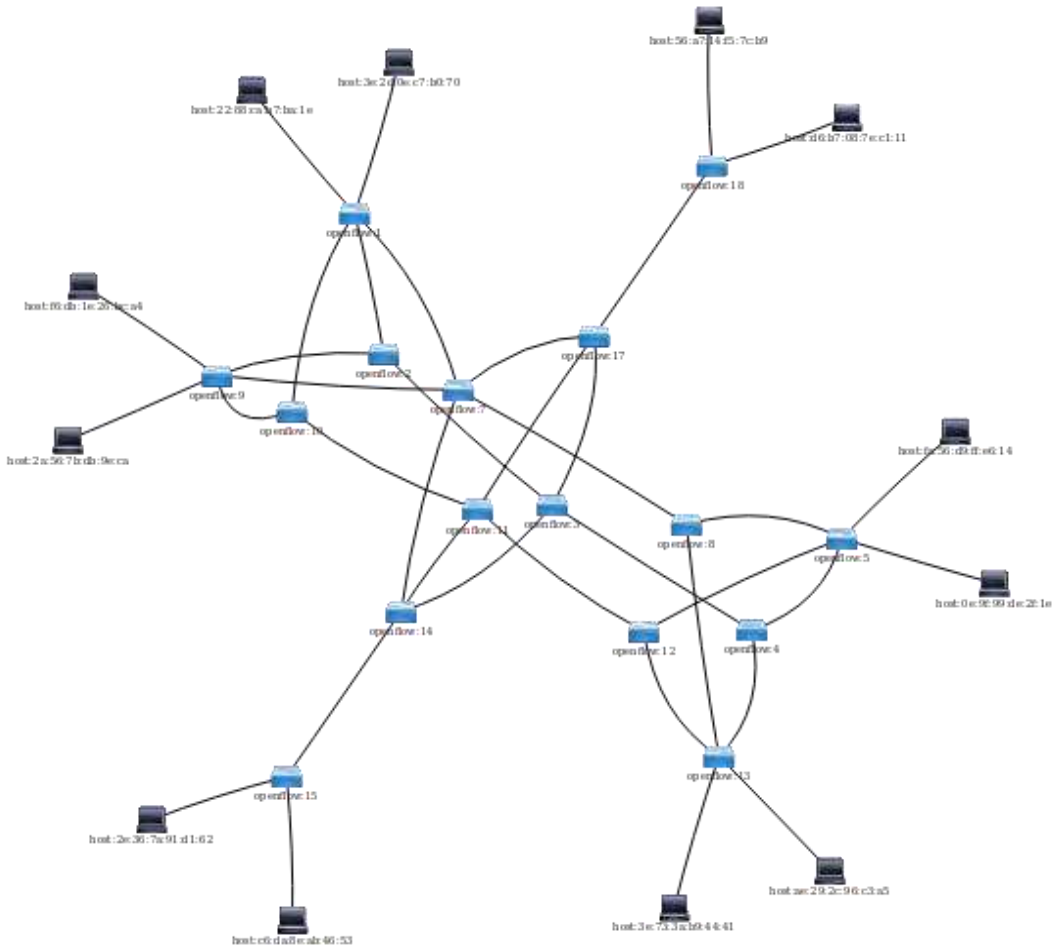


Figura 52. Visualización de la red personalizada

Nota: El código de la red personalizada, se encuentra en el **Anexo 1**.

Para ejecutar la red con Mininet se utiliza la siguiente sentencia:

sudo mn --custom redpropia.py --topo mytopo --controller=remote,ip=127.0.0.1,port=6653

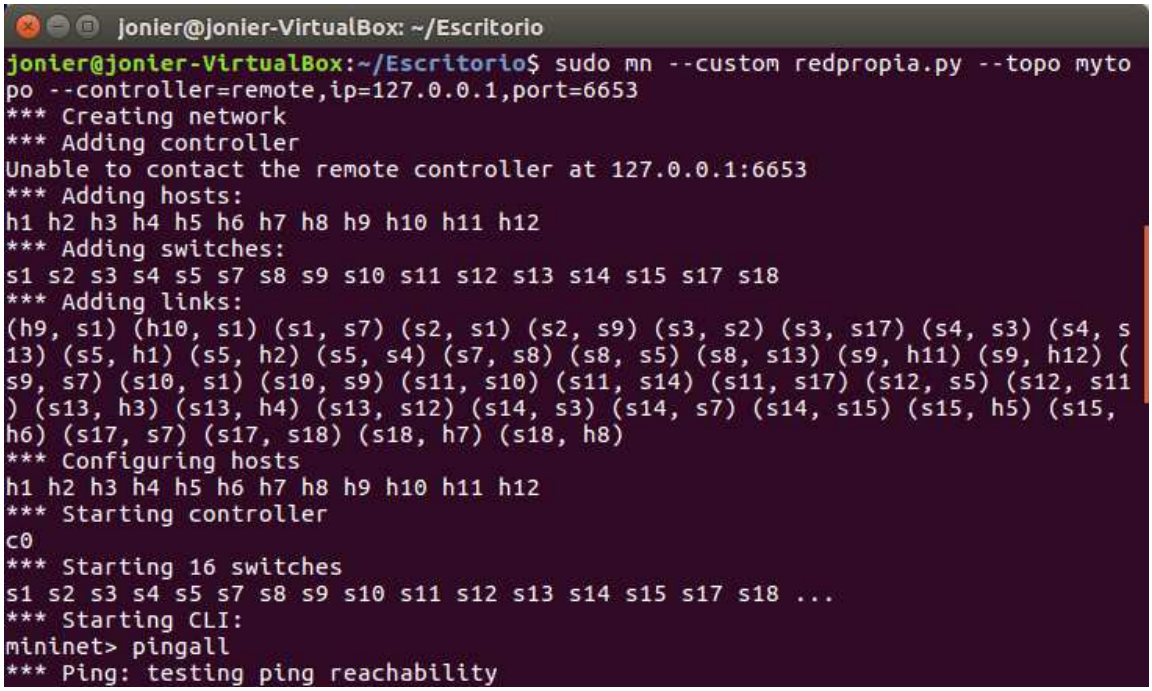


Figura 53. Configuración de red personalizada en Mininet

4.1.1 Pruebas del camino más corto o de menor carga

En este apartado se pretende observar como al aplicar el módulo de balance de carga se selecciona el camino más óptimo para el par de host utilizando el algoritmo de Dijkstra (camino más corto) y analizando la carga de los enlaces.

4.1.1.1 Ancho de banda sin usar el módulo de balance de carga

- *Ancho de banda entre h1 y h12*

Se realizan 3 mediciones de ancho de banda mediante el comando iperf:

iperf h1 h12

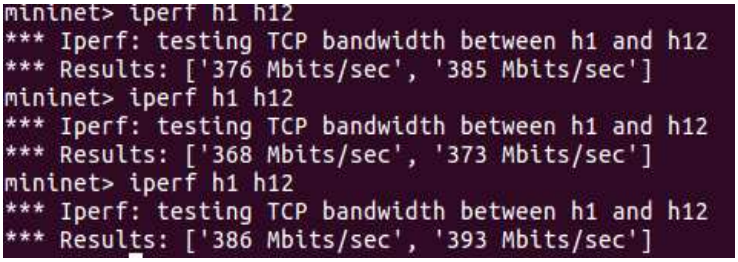


Figura 54. Ancho de banda entre h1 y h12

- *Ancho de banda entre h3 y h8*

Nuevamente se realizan 3 mediciones de ancho de banda utilizando el comando iperf:

iperf h3 h8

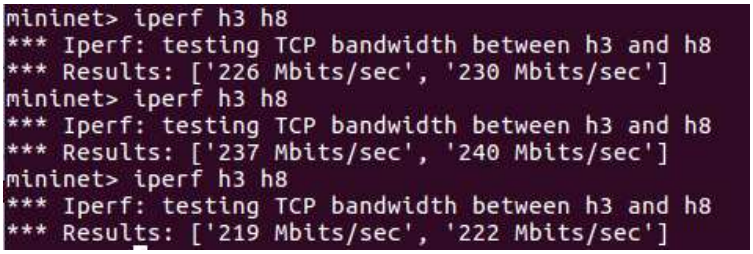


Figura 55. Ancho de banda entre h3 y h8

4.1.1.2 Ancho de banda utilizando el módulo de balance de carga

Nota: El módulo de balance de carga implementado fue creado por usuarios de **GITHUB**, y modificado por nosotros para acoplarlo a nuestras pruebas y versiones actuales de software, también se corrigieron algunos problemas presentes en el código. El código de balance de carga está hecho en Python y se encuentra en el **anexo 2**.

Para ejecutar el código se abre una nueva terminal y se escribe la siguiente secuencia:

sudo python LoadBalancer.py

- **Implementando módulo de balance de carga entre los hosts h1 y h12**

Al ejecutarse el módulo, se piden 3 datos:

Host 1 y host 2: Hosts entre los cuales se aplicará balance de carga.
Host 3: Host vecino al host 2.

Después de ingresar estos 3 datos aparecerá una información de las direcciones IP, MAC, puertos de los dispositivos, y la carga de los caminos en ese instante junto con el o los caminos más cortos, obtenido mediante la implementación del algoritmo de *Dijkstra*.

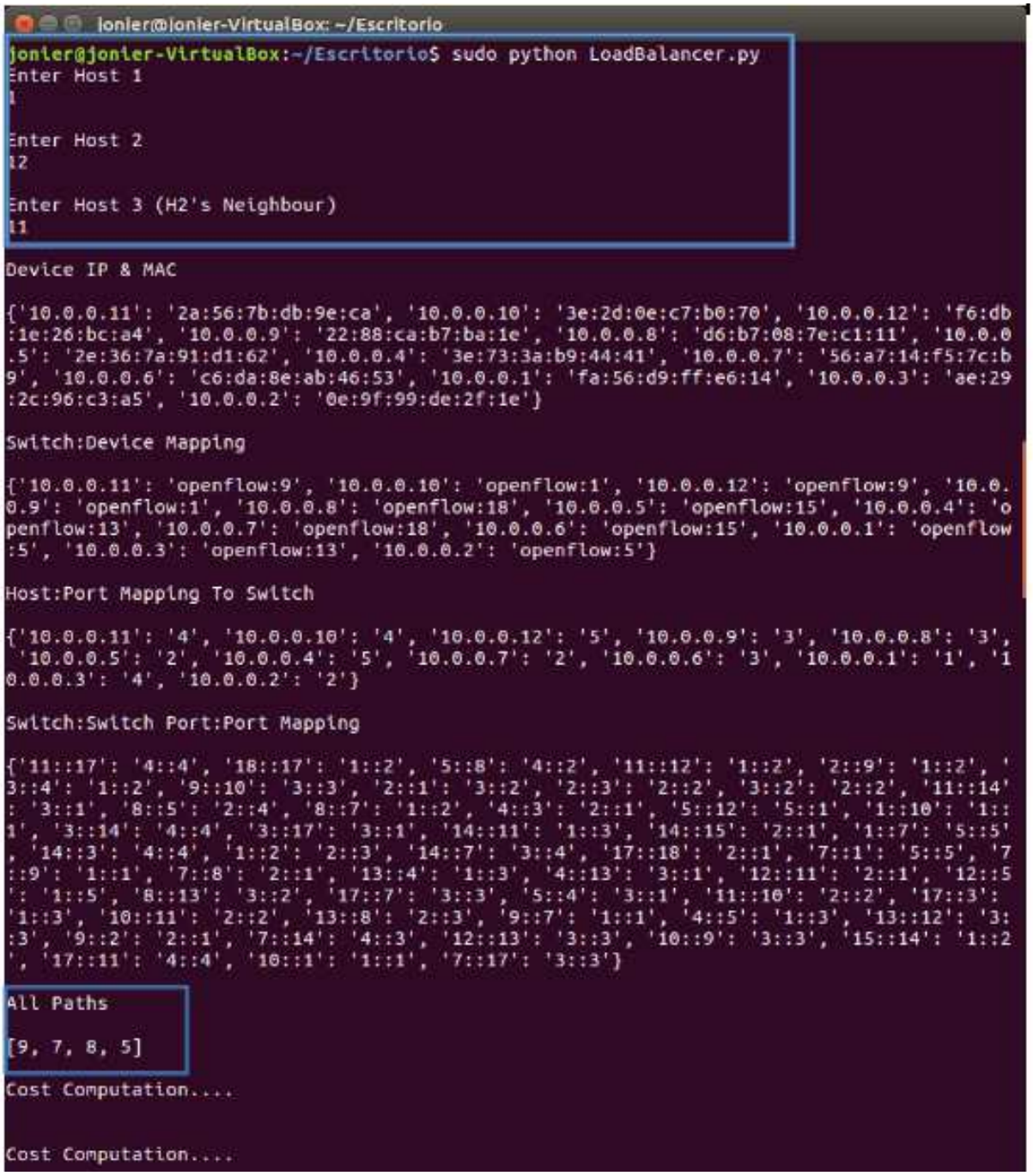


Figura 56. Información de la red al aplicar el módulo de balance de carga

En la imagen anterior aparece un apartado que dice “All Paths”, que hace referencia a los mejores caminos encontrados para el par de hosts, sin embargo aparece un solo camino debido a que entre todas las posibles rutas, hay un sólo camino más corto.

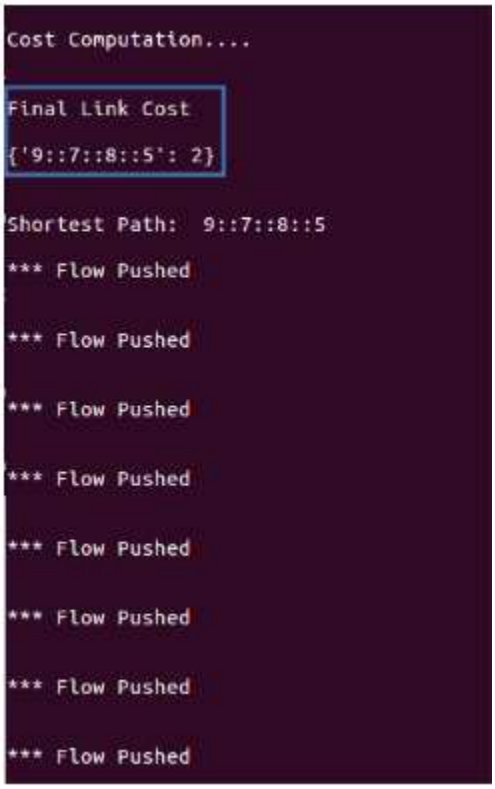


Figura 57. Instalación de flujos mediante el camino más corto encontrado

En la imagen anterior aparece el enlace final junto con su carga. Debido a que existe un camino más corto entre todos los caminos, el módulo de forma automática selecciona esta ruta sin analizar los demás enlaces que pueden tomar los paquetes entre estos puntos. Esto puede ser una desventaja en el momento en que el camino más corto seleccionado esté muy ocupado.

Nuevamente se realizan pruebas de ancho de banda entre este par de hosts para observar la mejora obtenida al implementar este módulo.

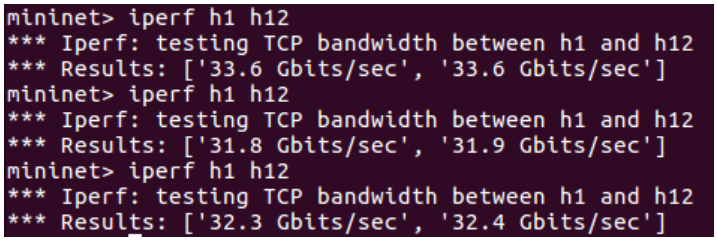


Figura 58. Ancho de banda entre h1 y h12

- **Implementando módulo de balance de carga entre los hosts h3 y h8**

Se realiza el mismo procedimiento llevado a cabo para los host h1 y h12.

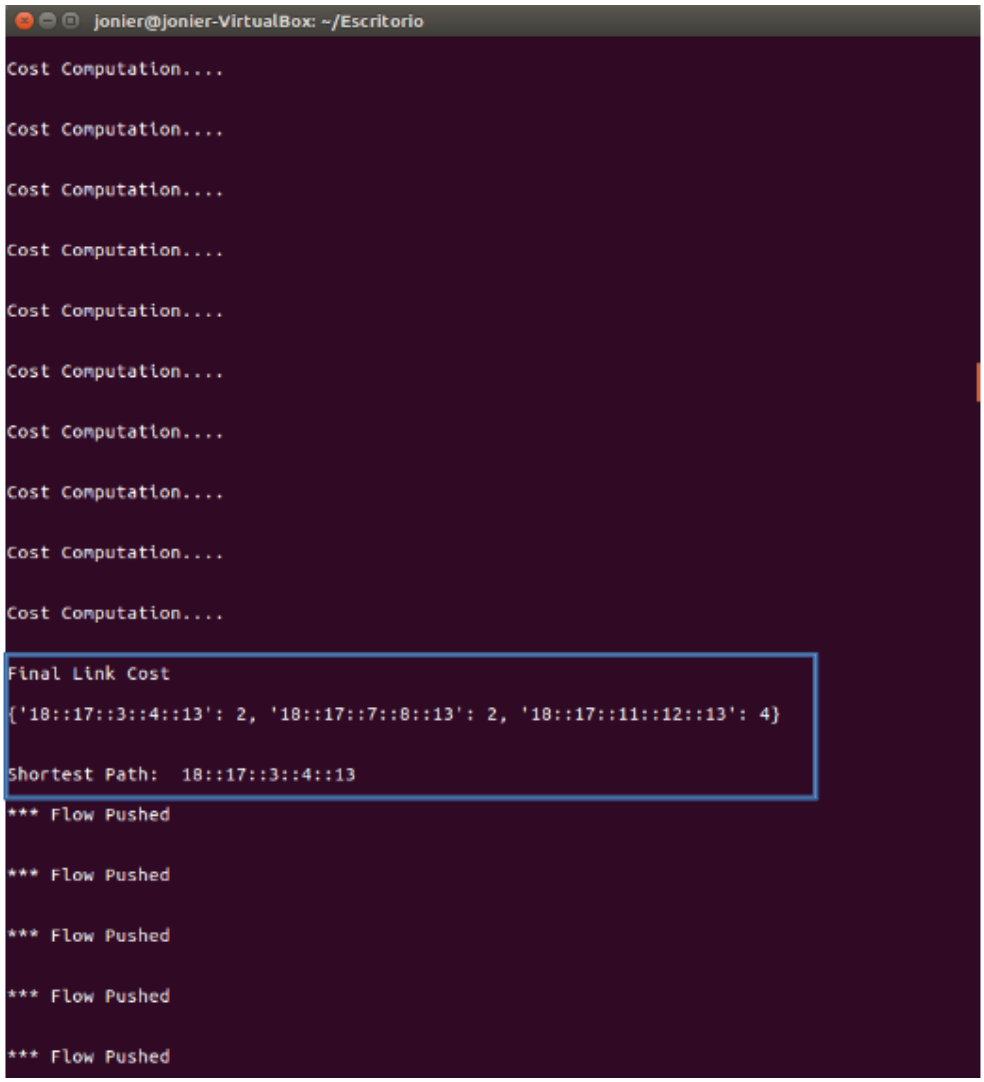


Figura 59. Camino más corto encontrado e instalación de los nuevos flujos

Nuevamente se realizan pruebas de ancho de banda entre este par de hosts para observar la mejora obtenida al implementar este módulo.

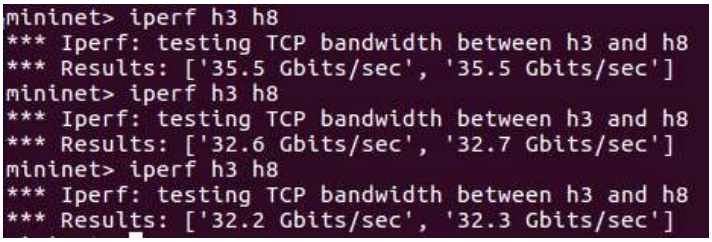


Figura 60. Ancho de banda entre h3 y h8

4.1.1.3 Comparación de los resultados obtenidos

- **Ancho de banda**

  - *Hosts h1 y h12*

A continuación se tabula y grafica los resultados obtenidos en las mediciones de ancho de banda para los hosts h1 y h12 cuando se aplica el módulo de balanceo de carga y cuando no.

Sin módulo de balance de carga	Con módulo de balance de carga
0,376 Gbit/s	33,6 Gbit/s
0,369 Gbit/s	31,9 Gbit/s
0,386 Gbit/s	32,3 Gbit/s

Tabla 4. Comparación del ancho de banda entre h1 y h12 con y sin balanceo de carga

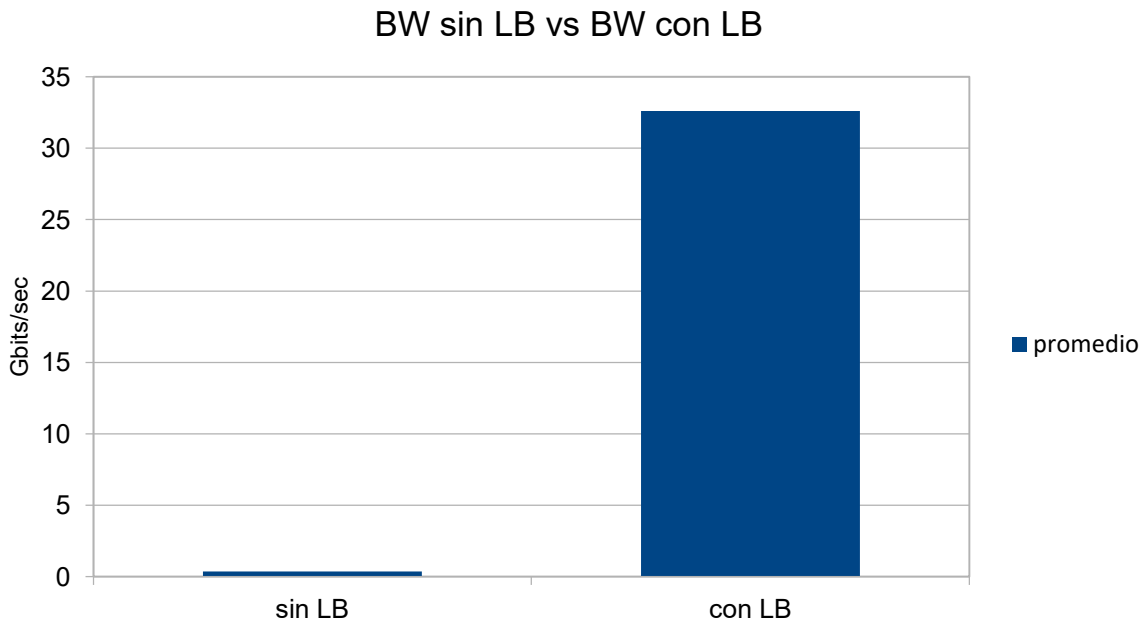


Figura 61. Promedio del ancho de banda entre h1 y h12 con y sin balanceo de carga

Los valores promedios de los resultados presentados son: 0,377 Gbit/s para la red sin el módulo de balanceo de carga y 32,6 Gbit/s al aplicar el módulo de balanceo de carga. Es decir que el ancho de banda obtuvo una mejora de 86,47 veces.

  - *Hosts h3 y h8*

A continuación se tabula y grafica los resultados obtenidos en las mediciones de ancho de banda para los hosts h3 y h8 cuando se aplica el módulo de balanceo de carga y cuando no.

Sin módulo de balance de carga	Con módulo de balance de carga
0,226 Gbit/s	33,5 Gbit/s
0,237 Gbit/s	32,6 Gbit/s
0,219 Gbit/s	32,2 Gbit/s

Tabla 5. Comparación del ancho de banda entre h3 y h8 con y sin balanceo de carga

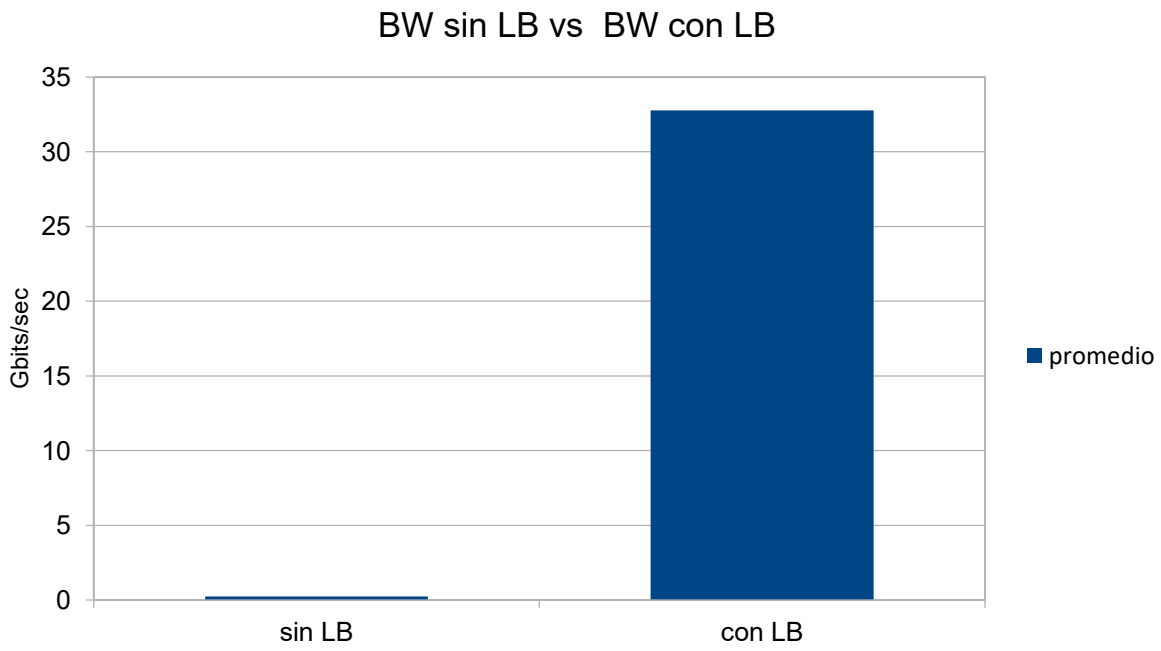


Figura 62. Promedio del ancho de banda entre h3 y h8 con y sin balanceo de carga

Los valores promedios de los resultados presentados son: 0,227 Gbit/s para la red sin utilizar el módulo de balanceo de carga y 32,766 Gbit/s para la red utilizando el módulo de balanceo de carga. Es decir que el ancho de banda obtuvo una mejora de 144,34 veces.

Nota: Al implementar el módulo de balance de carga, los flujos quedan instalados de forma permanente y esto impedirá el buen funcionamiento de las nuevas redes creadas utilizando el controlador OpenDaylight, por ello es necesario aplicar la siguiente instrucción para eliminar los flujos instalados:

> *curl -X DELETE -H "Content-Type: application/json" -H "Accept: application/json" --user admin:admin http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:NúmeroSwitch/table/0/flow/NúmeroFlujo*

Es necesario cambiar *NúmeroSwitch* por los números de los switches que hacen parte de la ruta más corta establecida por el balanceo de carga. Así mismo se debe cambiar *NúmeroFlujo* por 1 y 2, para borrar los flujos de ida y vuelta.

En general, los resultados obtenidos con la implementación del código para balanceo de carga en una red controlada con OpenDaylight, muestran un gran mejoramiento en el ancho de banda al seleccionar una ruta más corta o de menor carga para el par de hosts.
Este aumento de ancho de banda estuvo entre 80 y 150 veces más que el ancho de banda obtenido sin la implementación del módulo de balanceo de carga.

- **Solicitudes de ping**

Se envían 100 solicitudes ping y se obtienen las estadísticas de los tiempos que toman los paquetes en ser transmitidos.

➢ *Hosts h1 y h12*

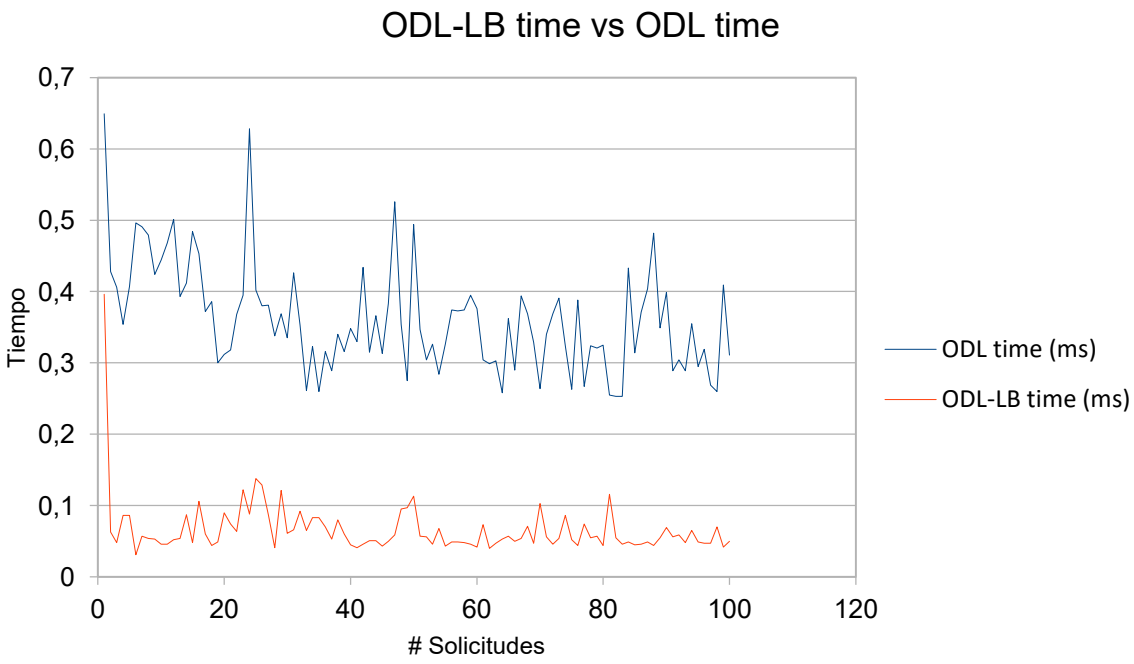


Figura 63. Tiempo que toma cada solicitud ping entre los hosts h1 y h12 cuando se utiliza balanceo de carga y cuando no

La gráfica azul corresponde al tiempo que demoran las solicitudes ping en llegar a su destino cuando no se utiliza el módulo de balance de carga, mientras que la gráfica roja se obtiene al aplicar el módulo de balance de carga. Cabe resaltar que entre menor sea el tiempo de las solicitudes, su respuesta es mejor.

Estadísticas de tiempo sin implementar el módulo de load balancer:

```
--- 10.0.0.12 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99057ms
rtt min/avg/max/mdev = 0.253/0.361/0.649/0.079 ms
```

Figura 64. Estadísticas de las solicitudes ping entre h1 y h12 sin balanceo de carga

Estadísticas de tiempo al implementar el módulo de load balancer:

```
--- 10.0.0.12 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99042ms
rtt min/avg/max/mdev = 0.031/0.065/0.396/0.041 ms
```

Figura 65. Estadísticas de las solicitudes ping entre h1 y h12 al aplicar balanceo de carga

	Sin balanceo de carga	Con balanceo de carga
Tiempo mínimo	0,253 ms	0,031 ms
Tiempo máximo	0,649 ms	0,369 ms
Tiempo promedio	0,361 ms	0,065 ms

Tabla 6. Estadísticas de tiempo de las solicitudes ping entre h1 y h12 al aplicar balanceo de carga y sin balanceo de carga

El tiempo promedio de las solicitudes de ping se redujo 5,55 veces cuando se utiliza el módulo de balance de carga para la selección del camino más óptimo, lo cual mejora considerablemente el comportamiento de la red.

- *Hosts h3 y h8*

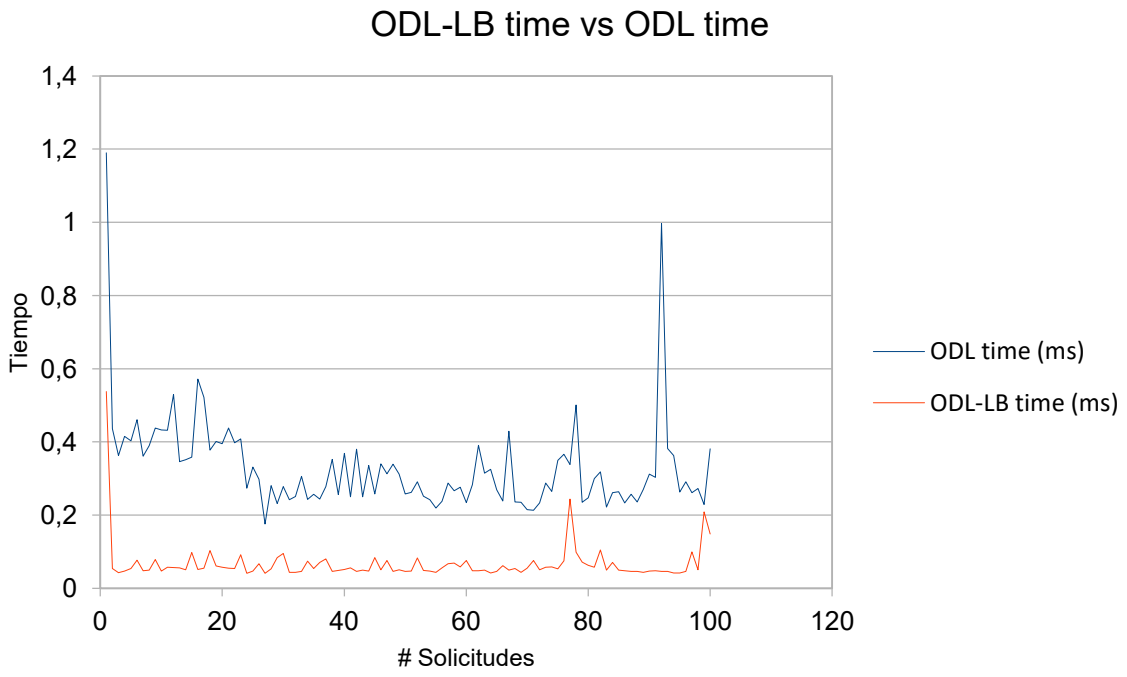


Figura 66. Tiempo que toma cada solicitud ping enviada entre los hosts h3 y h8 al utilizar el módulo de balanceo de carga y sin balanceo de carga

La gráfica azul corresponde al tiempo que demoran las solicitudes ping en llegar a su destino cuando no se utiliza el módulo de balance de carga, mientras que la gráfica roja se obtiene al aplicar el módulo de balanceo de carga.

Estadísticas de tiempo sin implementar el módulo de load balancer:



Figura 67. Estadísticas de las solicitudes ping entre h3 y h8 sin balanceo de carga

Estadísticas de tiempo al implementar el módulo de load balancer:

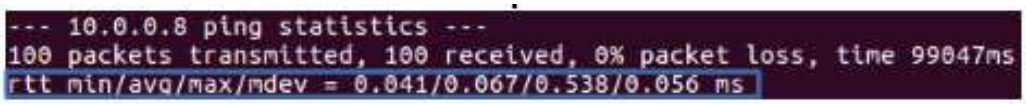


Figura 68. Estadísticas de las solicitudes ping entre h3 y h8 al aplicar balanceo de carga

	Sin balanceo de carga	Con balanceo de carga
Tiempo mínimo	0,176 ms	0,041 ms
Tiempo máximo	1,199 ms	0,538 ms
Tiempo promedio	0,330 ms	0,067 ms

Tabla 7. Comparación de los tiempos de las solicitudes ping entre h3 y h8 con balanceo de carga y sin balanceo de carga

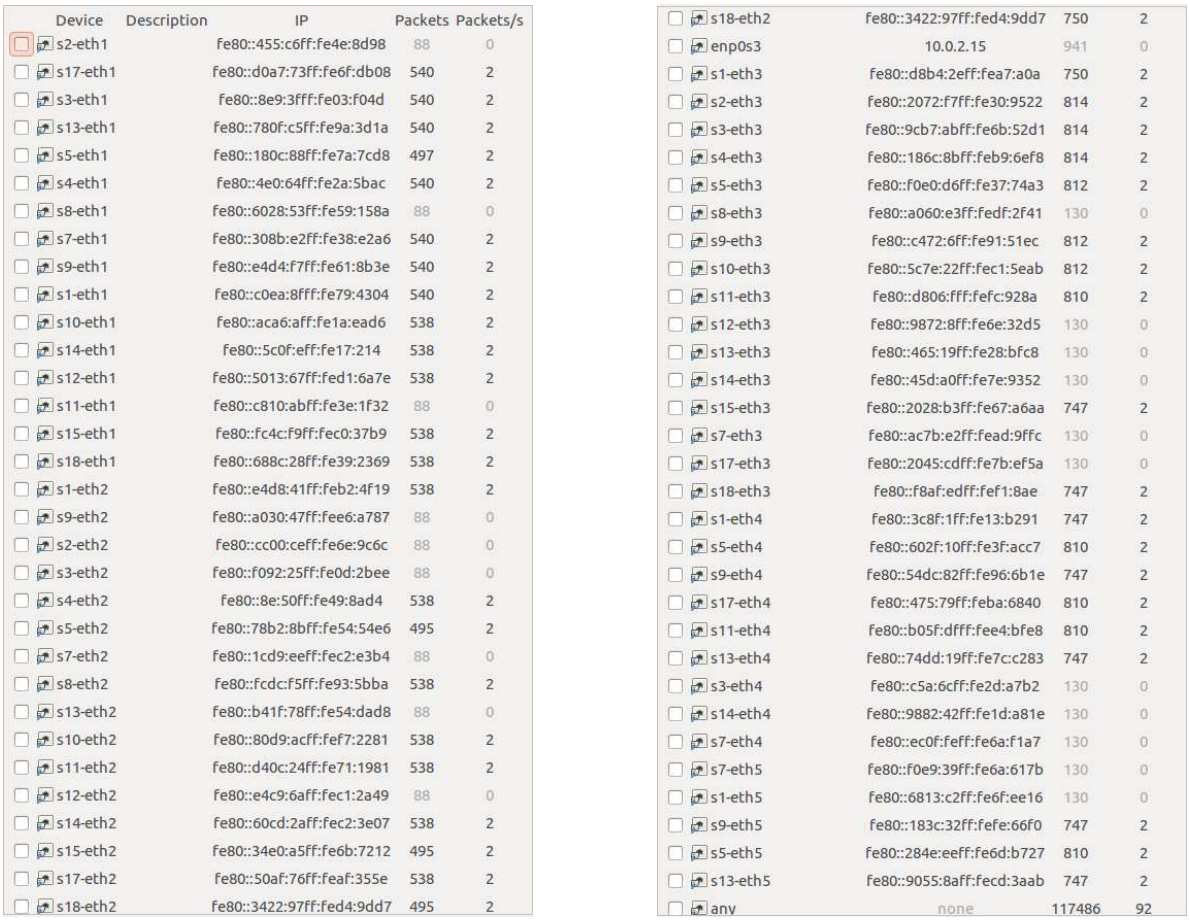
El tiempo promedio de las solicitudes de ping se redujo 4,93 veces cuando se utiliza el módulo de balance de carga para la selección del camino más óptimo, lo cual mejora considerablemente el comportamiento de la red.

En general, los resultados obtenidos con la implementación del código de balanceo de carga en una red controlada con OpenDaylight, muestran un mejoramiento significativo al reducir los tiempos de las solicitudes de ping entre dos extremos de la red al seleccionar un camino más corto o de menor carga.

4.1.2 Pruebas de balance de carga

En este apartado se pretende observar el comportamiento de la red al aplicar el módulo de balance de carga cuando se envían paquetes entre diferentes hosts al mismo tiempo. Con ello se pretende observar cómo se equilibra la carga de los diferentes enlaces de la red (balanceo de carga).

La Primera prueba consiste en revisar las interfaces de los switches mediante Wireshark (v 1.10.6) al realizar ping desde h4 a h8 sin utilizar el módulo de balance de carga:



Device	Description	IP	Packets	Packets/s
s2-eth1		fe80::455:c6ff:fe4e:8d98	88	0
s17-eth1		fe80::d0a7:73ff:fe6f:db08	540	2
s3-eth1		fe80::8e9:3fff:fe03:f04d	540	2
s13-eth1		fe80::780f:c5ff:fe9a:3d1a	540	2
s5-eth1		fe80::180c:88ff:fe7a:7cd8	497	2
s4-eth1		fe80::4e0:64ff:fe2a:5bac	540	2
s8-eth1		fe80::6028:53ff:fe59:158a	88	0
s7-eth1		fe80::308b:e2ff:fe38:e2a6	540	2
s9-eth1		fe80::e4d4:f7ff:fe61:8b3e	540	2
s1-eth1		fe80::c0ea:8fff:fe79:4304	540	2
s10-eth1		fe80::aca6:aff:fe1a:ead6	538	2
s14-eth1		fe80::5c0f:eff:fe17:214	538	2
s12-eth1		fe80::5013:67ff:fed1:6a7e	538	2
s11-eth1		fe80::c810:abff:fe3e:1f32	88	0
s15-eth1		fe80::fc4c:f9ff:fec0:37b9	538	2
s18-eth1		fe80::688c:28ff:fe39:2369	538	2
s1-eth2		fe80::e4d8:41ff:feb2:4f19	538	2
s9-eth2		fe80::a030:47ff:fee6:a787	88	0
s2-eth2		fe80::cc00:ceff:fe6e:9c6c	88	0
s3-eth2		fe80::f092:25ff:fe0d:2bee	88	0
s4-eth2		fe80::8e:50ff:fe49:8ad4	538	2
s5-eth2		fe80::78b2:8bff:fe54:54e6	495	2
s7-eth2		fe80::1cd9:eeff:fec2:e3b4	88	0
s8-eth2		fe80::fcdc:f5ff:fe93:5bba	538	2
s13-eth2		fe80::b41f:78ff:fe54:dad8	88	0
s10-eth2		fe80::80d9:acff:fef7:2281	538	2
s11-eth2		fe80::d40c:24ff:fe71:1981	538	2
s12-eth2		fe80::e4c9:6aff:fec1:2a49	88	0
s14-eth2		fe80::60cd:2aff:fec2:3e07	538	2
s15-eth2		fe80::34e0:a5ff:fe6b:7212	495	2
s17-eth2		fe80::50af:76ff:feaf:355e	538	2
s18-eth2		fe80::3422:97ff:fed4:9dd7	495	2

s18-eth2	fe80::3422:97ff:fed4:9dd7	750	2
enp0s3	10.0.2.15	941	0
s1-eth3	fe80::d8b4:2eff:fea7:a0a	750	2
s2-eth3	fe80::2072:f7ff:fe30:9522	814	2
s3-eth3	fe80::9cb7:abff:fe6b:52d1	814	2
s4-eth3	fe80::186c:8bff:feb9:6ef8	814	2
s5-eth3	fe80::f0e0:d6ff:fe37:74a3	812	2
s8-eth3	fe80::a060:e3ff:fedf:2f41	130	0
s9-eth3	fe80::c472:6ff:fe91:51ec	812	2
s10-eth3	fe80::5c7e:22ff:fec1:5eab	812	2
s11-eth3	fe80::d806:fff:fefc:928a	810	2
s12-eth3	fe80::9872:8ff:fe6e:32d5	130	0
s13-eth3	fe80::465:19ff:fe28:bfc8	130	0
s14-eth3	fe80::45d:a0ff:fe7e:9352	130	0
s15-eth3	fe80::2028:b3ff:fe67:a6aa	747	2
s7-eth3	fe80::ac7b:e2ff:fead:9ffc	130	0
s17-eth3	fe80::2045:cdff:fe7b:ef5a	130	0
s18-eth3	fe80::f8af:edff:fef1:8ae	747	2
s1-eth4	fe80::3c8f:1ff:fe13:b291	747	2
s5-eth4	fe80::602f:10ff:fe3f:acc7	810	2
s9-eth4	fe80::54dc:82ff:fe96:6b1e	747	2
s17-eth4	fe80::475:79ff:feba:6840	810	2
s11-eth4	fe80::b05f:dfff:fee4:bfe8	810	2
s13-eth4	fe80::74dd:19ff:fe7c:c283	747	2
s3-eth4	fe80::c5a:6cff:fe2d:a7b2	130	0
s14-eth4	fe80::9882:42ff:fe1d:a81e	130	0
s7-eth4	fe80::ec0f:feff:fe6a:f1a7	130	0
s7-eth5	fe80::f0e9:39ff:fe6a:617b	130	0
s1-eth5	fe80::6813:c2ff:fe6f:ee16	130	0
s9-eth5	fe80::183c:32ff:fefe:66f0	747	2
s5-eth5	fe80::284e:eeff:fe6d:b727	810	2
s13-eth5	fe80::9055:8aff:fecd:3aab	747	2
any	none	117486	92

Figura 69. Análisis de tráfico en las interfaces de los switches al enviar ping desde h4 a h8

Como se puede observar en la imagen, al realizar ping entre estos hosts se puede verificar que todos los switches de la red son utilizados, es decir, que todos se encuentran enviando paquetes (2 paquetes/s) por sus diferentes interfaces, lo que demuestra que el controlador no gestiona de manera adecuada las rutas por las cuales deben ir los paquetes para obtener un buen rendimiento en ancho de banda, retardos, y en general la distribución de la carga en los enlaces. Si el controlador enruta la información por los caminos más cortos o con menor costo mediante un algoritmo para balanceo de carga, mejoraría de forma significativa el comportamiento de la red. En el caso de realizar ping entre otros pares de hosts, se obtendrá el mismo resultado o comportamiento.

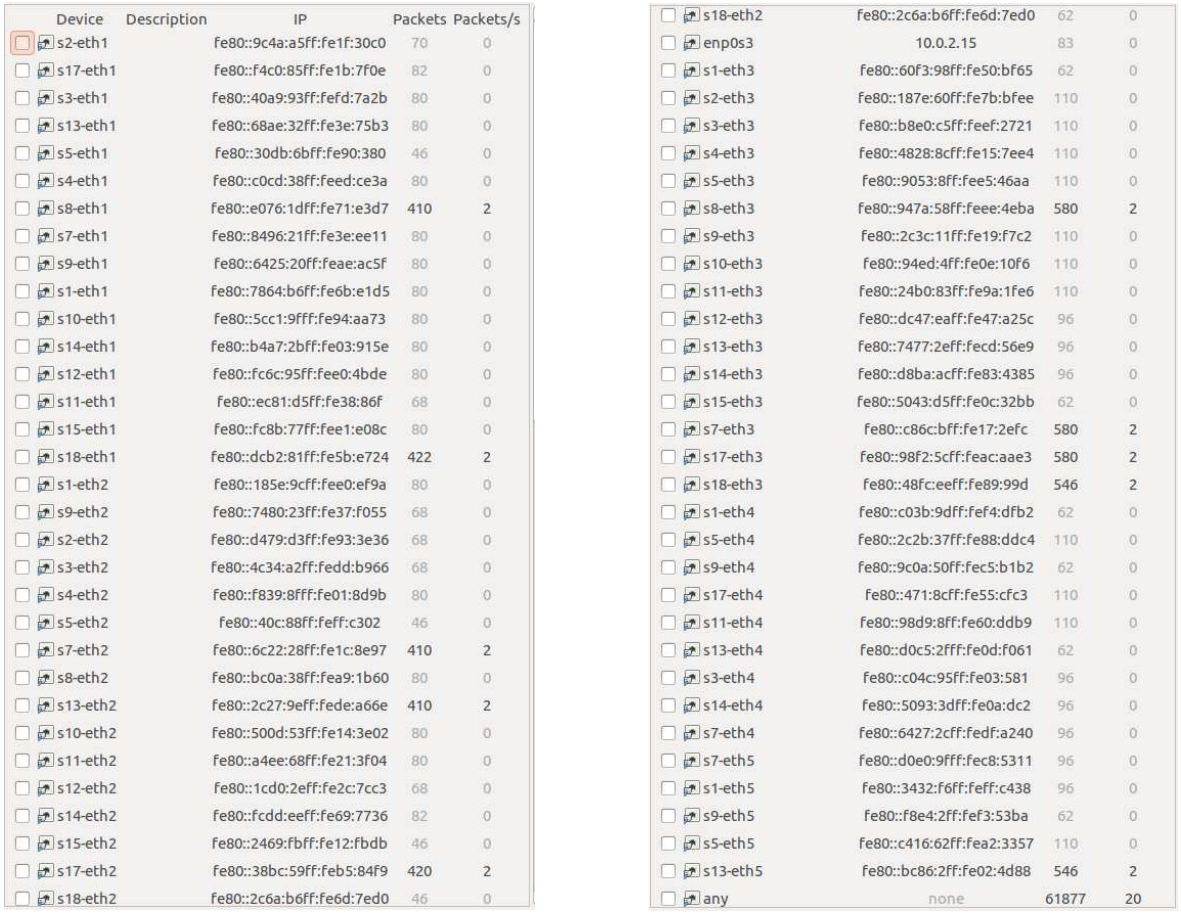
4.1.2.1 Aplicando balance de carga entre h4 y h8

```
Final Link Cost
{'18::17::3::4::13': 4, '18::17::7::8::13': 2, '18::17::11::12::13': 4}

Shortest Path:  18::17::7::8::13
```

Figura 70. Selección del camino más corto y de menor carga entre h4 y h8

A continuación se envían solicitudes de ping entre estos 2 hosts (Ver figura 72) y mediante el uso de Wireshark se analiza las interfaces de los switches que se encuentran transmitiendo paquetes, de tal forma que se pueda verificar que efectivamente las solicitudes de ping transmitidas sean enrutadas con el camino seleccionado por el módulo de balance de carga.



Device	Description	IP	Packets	Packets/s
s2-eth1		fe80::9c4a:a5ff:fe1f:30c0	70	0
s17-eth1		fe80::f4c0:85ff:fe1b:7f0e	82	0
s3-eth1		fe80::40a9:93ff:fefd:7a2b	80	0
s13-eth1		fe80::68ae:32ff:fe3e:75b3	80	0
s5-eth1		fe80::30db:6bff:fe90:380	46	0
s4-eth1		fe80::c0cd:38ff:feed:ce3a	80	0
s8-eth1		fe80::e076:1dff:fe71:e3d7	410	2
s7-eth1		fe80::8496:21ff:fe3e:ee11	80	0
s9-eth1		fe80::6425:20ff:feae:ac5f	80	0
s1-eth1		fe80::7864:b6ff:fe6b:e1d5	80	0
s10-eth1		fe80::5cc1:9fff:fe94:aa73	80	0
s14-eth1		fe80::b4a7:2bff:fe03:915e	80	0
s12-eth1		fe80::fc6c:95ff:fee0:4bde	80	0
s11-eth1		fe80::ec81:d5ff:fe38:86f	68	0
s15-eth1		fe80::fc8b:77ff:fee1:e08c	80	0
s18-eth1		fe80::dcb2:81ff:fe5b:e724	422	2
s1-eth2		fe80::185e:9cff:fee0:ef9a	80	0
s9-eth2		fe80::7480:23ff:fe37:f055	68	0
s2-eth2		fe80::d479:d3ff:fe93:3e36	68	0
s3-eth2		fe80::4c34:a2ff:fedd:b966	68	0
s4-eth2		fe80::f839:8fff:fe01:8d9b	80	0
s5-eth2		fe80::40c:88ff:feff:c302	46	0
s7-eth2		fe80::6c22:28ff:fe1c:8e97	410	2
s8-eth2		fe80::bc0a:38ff:fea9:1b60	80	0
s13-eth2		fe80::2c27:9eff:fede:a66e	410	2
s10-eth2		fe80::500d:53ff:fe14:3e02	80	0
s11-eth2		fe80::a4ee:68ff:fe21:3f04	80	0
s12-eth2		fe80::1cd0:2eff:fe2c:7cc3	68	0
s14-eth2		fe80::fcdd:eeff:fe69:7736	82	0
s15-eth2		fe80::2469:fbff:fe12:fbdb	46	0
s17-eth2		fe80::38bc:59ff:feb5:84f9	420	2
s18-eth2		fe80::2c6a:b6ff:fe6d:7ed0	46	0

s18-eth2	fe80::2c6a:b6ff:fe6d:7ed0	62	0
enp0s3	10.0.2.15	83	0
s1-eth3	fe80::60f3:98ff:fe50:bf65	62	0
s2-eth3	fe80::187e:60ff:fe7b:bfee	110	0
s3-eth3	fe80::b8e0:c5ff:feef:2721	110	0
s4-eth3	fe80::4828:8cff:fe15:7ee4	110	0
s5-eth3	fe80::9053:8ff:fee5:46aa	110	0
s8-eth3	fe80::947a:58ff:feee:4eba	580	2
s9-eth3	fe80::2c3c:11ff:fe19:f7c2	110	0
s10-eth3	fe80::94ed:4ff:fe0e:10f6	110	0
s11-eth3	fe80::24b0:83ff:fe9a:1fe6	110	0
s12-eth3	fe80::dc47:eaff:fe47:a25c	96	0
s13-eth3	fe80::7477:2eff:fecd:56e9	96	0
s14-eth3	fe80::d8ba:acff:fe83:4385	96	0
s15-eth3	fe80::5043:d5ff:fe0c:32bb	62	0
s7-eth3	fe80::c86c:bff:fe17:2efc	580	2
s17-eth3	fe80::98f2:5cff:feac:aae3	580	2
s18-eth3	fe80::48fc:eeff:fe89:99d	546	2
s1-eth4	fe80::c03b:9dff:fef4:dfb2	62	0
s5-eth4	fe80::2c2b:37ff:fe88:ddc4	110	0
s9-eth4	fe80::9c0a:50ff:fec5:b1b2	62	0
s17-eth4	fe80::471:8cff:fe55:cfc3	110	0
s11-eth4	fe80::98d9:8ff:fe60:ddb9	110	0
s13-eth4	fe80::d0c5:2fff:fe0d:f061	62	0
s3-eth4	fe80::c04c:95ff:fe03:581	96	0
s14-eth4	fe80::5093:3dff:fe0a:dc2	96	0
s7-eth4	fe80::6427:2cff:fedf:a240	96	0
s7-eth5	fe80::d0e0:9fff:fec8:5311	96	0
s1-eth5	fe80::3432:f6ff:feff:c438	96	0
s9-eth5	fe80::f8e4:2ff:fef3:53ba	62	0
s5-eth5	fe80::c416:62ff:fea2:3357	110	0
s13-eth5	fe80::bc86:2ff:fe02:4d88	546	2
any	none	61877	20

Figura 71. Análisis de tráfico en las interfaces de los switches al enviar ping desde h4 a h8 aplicando balance de carga entre los hosts

Por cada switch presente en el camino más corto deben aparecer dos interfaces, una de entrada y una de salida. Por lo anterior aparecen 10 interfaces que se encuentran transmitiendo paquetes (2 paquetes/s) asociados a las solicitudes de ping. Además, los switches que no están relacionados con el camino más corto se encuentran inactivos (0 paquetes/s).

Efectivamente los paquetes transmitidos se encaminan a través de los switches 18, 17, 7, 8 y 13. Los demás switches se encuentran inactivos.

4.1.2.2 Aplicando balance de carga entre h3 y h7

Antes de aplicar balance de carga entre los hosts, es necesario ocupar el enlace seleccionado para los anteriores hosts, de tal forma que el módulo de balance de carga analice la carga presente en la red, y evite elegir el mismo camino seleccionado por los anteriores hosts.

En Mininet se digita la siguiente sentencia para utilizar la terminal del host h1 y enviar ping desde dicho host:

xterm h1

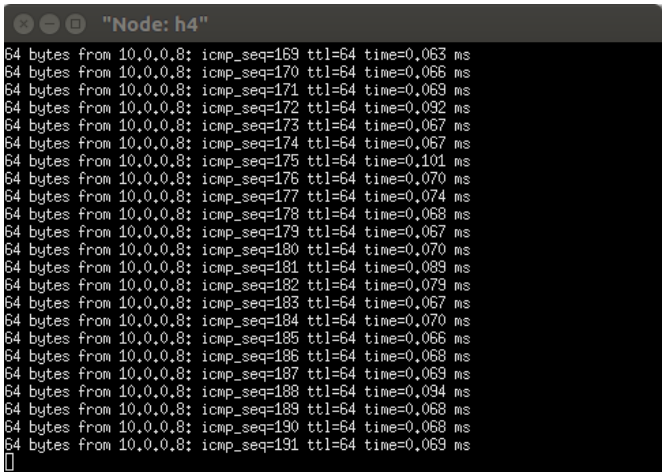


Figura 72. Solicitudes de ping enviadas desde h4 a h8

Luego se aplica el módulo de balance de carga entre el par de hosts y se obtienen los caminos más cortos junto con la carga de estos enlaces:

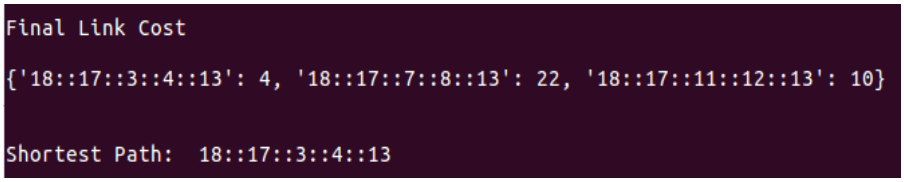
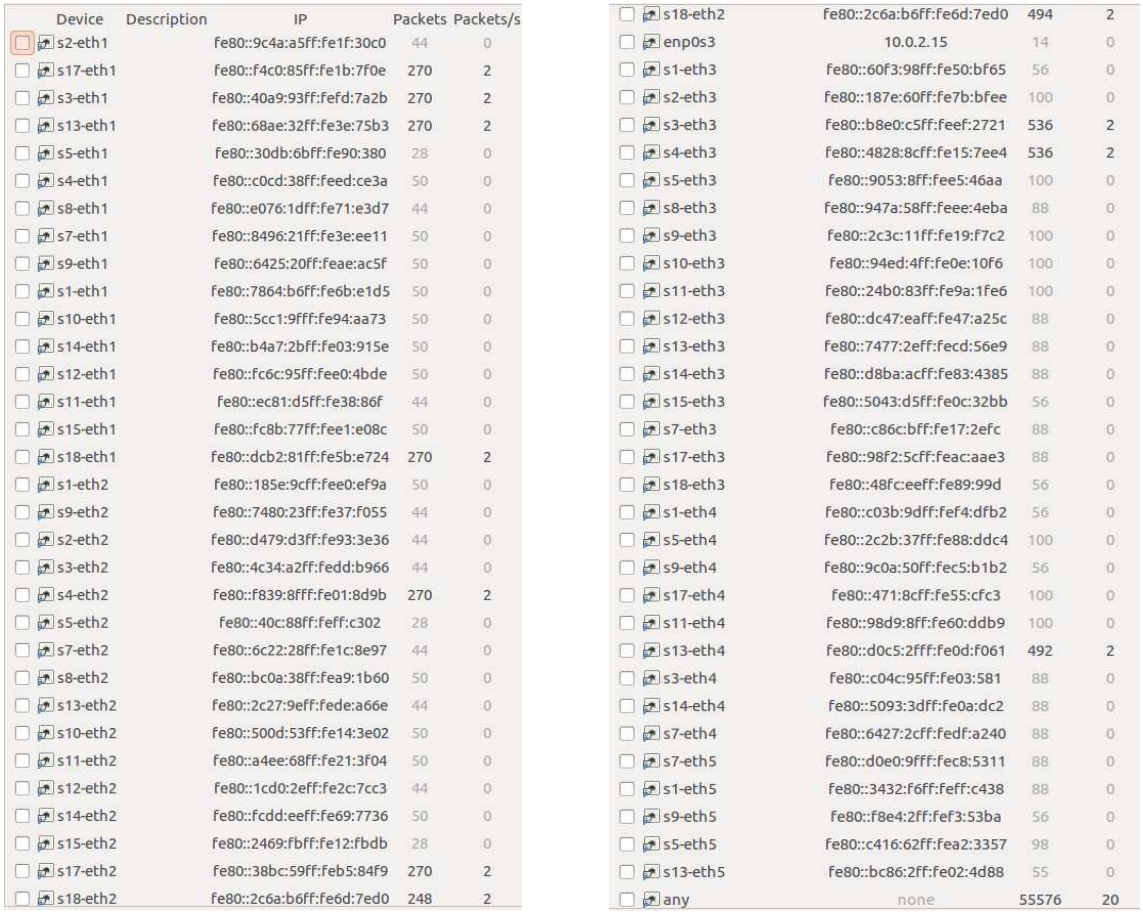


Figura 73. Selección del camino más corto y de menor carga entre h3 y h7

Cómo se puede ver en la imagen anterior, aparecen los mismos 3 caminos encontrados al aplicar el módulo entre h3 y h7, con la diferencia de que la carga entre los enlaces esta vez es diferente. La mayor carga de los enlaces se encuentra presente en el camino seleccionado por los hosts h3 y h7, lo que permite que el módulo elija un camino diferente a este y de esa manera se distribuya mejor la carga en los enlaces de la red.

Luego se realiza ping entre estos 2 hosts (h3 y h7) y se detienen las demás solicitudes ping iniciadas en las pruebas anteriores, de esta manera se pretende verificar que las interfaces de los switches que se encuentran transmitiendo los paquetes, sea la misma elegida por el módulo de balance de carga para los hosts h3 y h7.



Device	Description	IP	Packets	Packets/s
s2-eth1		fe80::9c4a:a5ff:fe1f:30c0	44	0
s17-eth1		fe80::f4c0:85ff:fe1b:7f0e	270	2
s3-eth1		fe80::40a9:93ff:fefd:7a2b	270	2
s13-eth1		fe80::68ae:32ff:fe3e:75b3	270	2
s5-eth1		fe80::30db:6bff:fe90:380	28	0
s4-eth1		fe80::c0cd:38ff:feed:ce3a	50	0
s8-eth1		fe80::e076:1dff:fe71:e3d7	44	0
s7-eth1		fe80::8496:21ff:fe3e:ee11	50	0
s9-eth1		fe80::6425:20ff:feae:ac5f	50	0
s1-eth1		fe80::7864:b6ff:fe6b:e1d5	50	0
s10-eth1		fe80::5cc1:9fff:fe94:aa73	50	0
s14-eth1		fe80::b4a7:2bff:fe03:915e	50	0
s12-eth1		fe80::fc6c:95ff:fee0:4bde	50	0
s11-eth1		fe80::ec81:d5ff:fe38:86f	44	0
s15-eth1		fe80::fc8b:77ff:fee1:e08c	50	0
s18-eth1		fe80::dcb2:81ff:fe5b:e724	270	2
s1-eth2		fe80::185e:9cff:fee0:ef9a	50	0
s9-eth2		fe80::7480:23ff:fe37:f055	44	0
s2-eth2		fe80::d479:d3ff:fe93:3e36	44	0
s3-eth2		fe80::4c34:a2ff:fedd:b966	44	0
s4-eth2		fe80::f839:8fff:fe01:8d9b	270	2
s5-eth2		fe80::40c:88ff:feff:c302	28	0
s7-eth2		fe80::6c22:28ff:fe1c:8e97	44	0
s8-eth2		fe80::bc0a:38ff:fea9:1b60	50	0
s13-eth2		fe80::2c27:9eff:fede:a66e	44	0
s10-eth2		fe80::500d:53ff:fe14:3e02	50	0
s11-eth2		fe80::a4ee:68ff:fe21:3f04	50	0
s12-eth2		fe80::1cd0:2eff:fe2c:7cc3	44	0
s14-eth2		fe80::fcdd:eeff:fe69:7736	50	0
s15-eth2		fe80::2469:fbff:fe12:fbdb	28	0
s17-eth2		fe80::38bc:59ff:feb5:84f9	270	2
s18-eth2		fe80::2c6a:b6ff:fe6d:7ed0	248	2

s18-eth2	fe80::2c6a:b6ff:fe6d:7ed0	494	2
enp0s3	10.0.2.15	14	0
s1-eth3	fe80::60f3:98ff:fe50:bf65	56	0
s2-eth3	fe80::187e:60ff:fe7b:bfee	100	0
s3-eth3	fe80::b8e0:c5ff:feef:2721	536	2
s4-eth3	fe80::4828:8cff:fe15:7ee4	536	2
s5-eth3	fe80::9053:8ff:fee5:46aa	100	0
s8-eth3	fe80::947a:58ff:feee:4eba	88	0
s9-eth3	fe80::2c3c:11ff:fe19:f7c2	100	0
s10-eth3	fe80::94ed:4ff:fe0e:10f6	100	0
s11-eth3	fe80::24b0:83ff:fe9a:1fe6	100	0
s12-eth3	fe80::dc47:eaff:fe47:a25c	88	0
s13-eth3	fe80::7477:2eff:fecd:56e9	88	0
s14-eth3	fe80::d8ba:acff:fe83:4385	88	0
s15-eth3	fe80::5043:d5ff:fe0c:32bb	56	0
s7-eth3	fe80::c86c:bff:fe17:2efc	88	0
s17-eth3	fe80::98f2:5cff:feac:aae3	88	0
s18-eth3	fe80::48fc:eeff:fe89:99d	56	0
s1-eth4	fe80::c03b:9dff:fef4:dfb2	56	0
s5-eth4	fe80::2c2b:37ff:fe88:ddc4	100	0
s9-eth4	fe80::9c0a:50ff:fec5:b1b2	56	0
s17-eth4	fe80::471:8cff:fe55:cfc3	100	0
s11-eth4	fe80::98d9:8ff:fe60:ddb9	100	0
s13-eth4	fe80::d0c5:2fff:fe0d:f061	492	2
s3-eth4	fe80::c04c:95ff:fe03:581	88	0
s14-eth4	fe80::5093:3dff:fe0a:dc2	88	0
s7-eth4	fe80::6427:2cff:fedf:a240	88	0
s7-eth5	fe80::d0e0:9fff:fec8:5311	88	0
s1-eth5	fe80::3432:f6ff:feff:c438	88	0
s9-eth5	fe80::f8e4:2ff:fef3:53ba	56	0
s5-eth5	fe80::c416:62ff:fea2:3357	98	0
s13-eth5	fe80::bc86:2ff:fe02:4d88	55	0
any	none	55576	20

Figura 74. Paquetes enviados a través de las interfaces de los switches

Los switches 18, 17, 3, 4 y 13 se encuentran activos, los demás dispositivos están inactivos.

4.1.2.3 Aplicando balance de carga entre h1 a h8

Primero se envía solicitudes de ping entre los hosts que anteriormente se les aplicaron balance de carga, ocupando de esta manera los caminos seleccionados anteriormente.

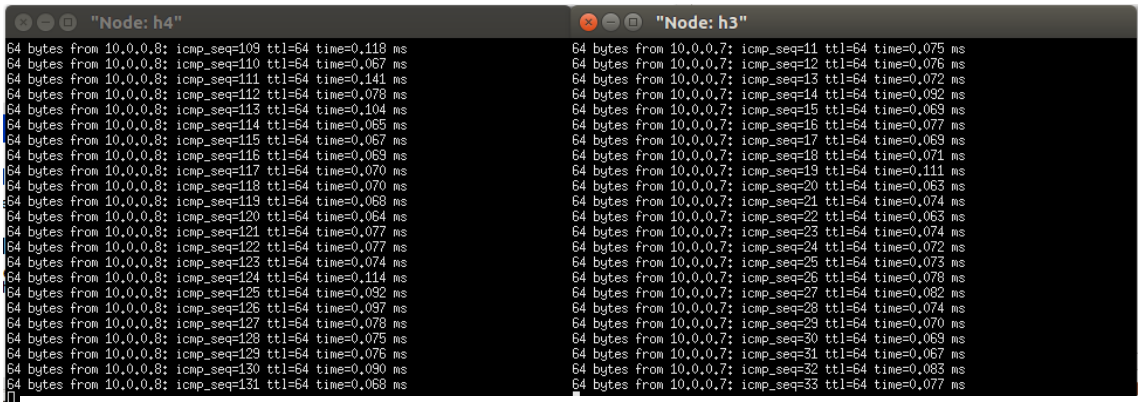


Figura 75. Solicitudes de ping enviadas desde h4 a h8 y h3 a h7

Luego se aplica balance de carga entre los hosts h1 y h8:

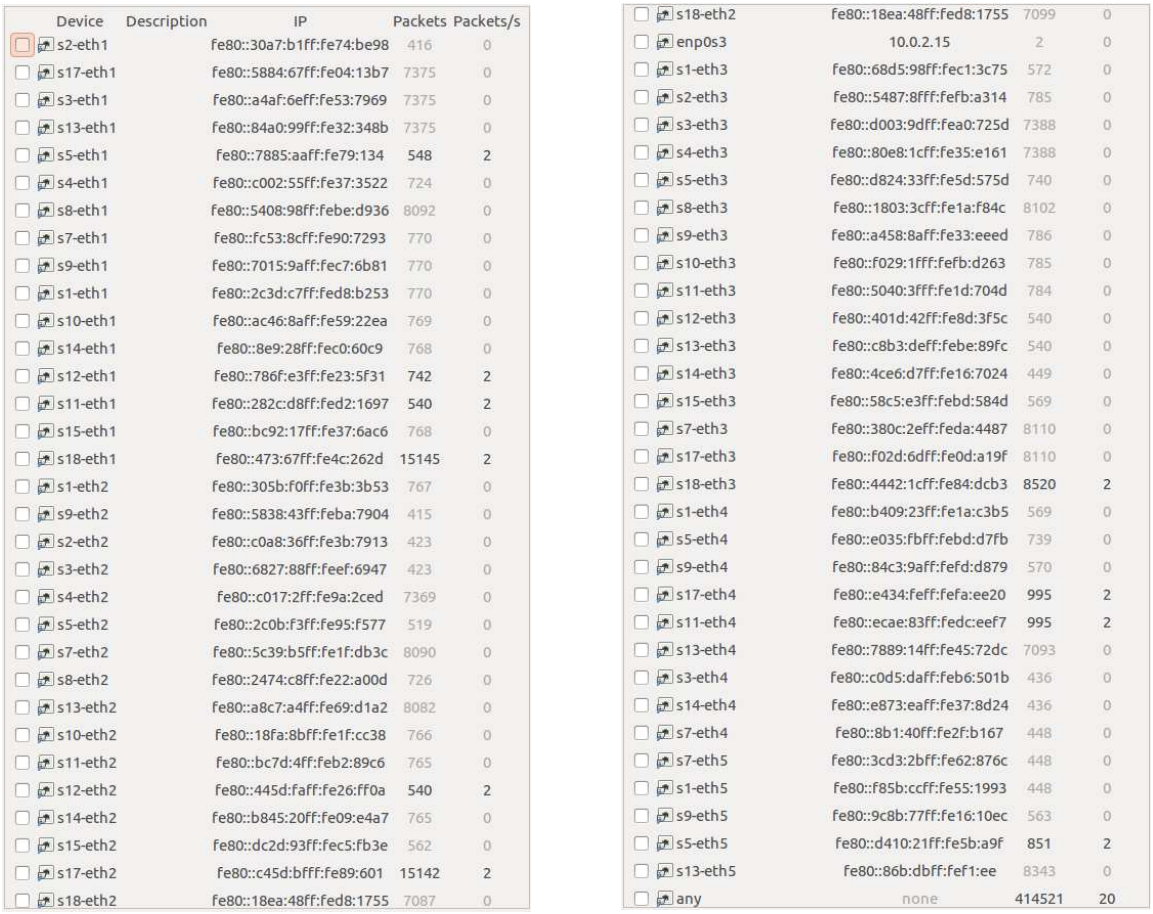
```
Final Link Cost
{'18::17::11::12::5': 16, '18::17::3::4::5': 24, '18::17::7::8::5': 20}

Shortest Path:  18::17::11::12::5
```

Figura 76. Selección del camino más corto y de menor carga entre h4 y h8

Como se puede observar en la imagen anterior, el módulo de balance de carga escogió el camino más apropiado, puesto que el camino 18:17:11:12:5 posee más switches diferentes a los seleccionados por los anteriores hosts, es decir, los caminos 18:17:3:4:13 y 18:17:7:8:13 están siendo más ocupados debido a las solicitudes ping.

Luego se detienen las solicitudes ping que se encuentran activas, y se hace ping de h4 a h8, para verificar el tráfico en las interfaces de los switches.



Device	Description	IP	Packets	Packets/s
s2-eth1		fe80::30a7:b1ff:fe74:be98	416	0
s17-eth1		fe80::5884:67ff:fe04:13b7	7375	0
s3-eth1		fe80::a4af:6eff:fe53:7969	7375	0
s13-eth1		fe80::84a0:99ff:fe32:348b	7375	0
s5-eth1		fe80::7885:aaff:fe79:134	548	2
s4-eth1		fe80::c002:55ff:fe37:3522	724	0
s8-eth1		fe80::5408:98ff:febe:d936	8092	0
s7-eth1		fe80::fc53:8cff:fe90:7293	770	0
s9-eth1		fe80::7015:9aff:fec7:6b81	770	0
s1-eth1		fe80::2c3d:c7ff:fed8:b253	770	0
s10-eth1		fe80::ac46:8aff:fe59:22ea	769	0
s14-eth1		fe80::8e9:28ff:fec0:60c9	768	0
s12-eth1		fe80::786f:e3ff:fe23:5f31	742	2
s11-eth1		fe80::282c:d8ff:fed2:1697	540	2
s15-eth1		fe80::bc92:17ff:fe37:6ac6	768	0
s18-eth1		fe80::473:67ff:fe4c:262d	15145	2
s1-eth2		fe80::305b:f0ff:fe3b:3b53	767	0
s9-eth2		fe80::5838:43ff:feba:7904	415	0
s2-eth2		fe80::c0a8:36ff:fe3b:7913	423	0
s3-eth2		fe80::6827:88ff:feef:6947	423	0
s4-eth2		fe80::c017:2ff:fe9a:2ced	7369	0
s5-eth2		fe80::2c0b:f3ff:fe95:f577	519	0
s7-eth2		fe80::5c39:b5ff:fe1f:db3c	8090	0
s8-eth2		fe80::2474:c8ff:fe22:a00d	726	0
s13-eth2		fe80::a8c7:a4ff:fe69:d1a2	8082	0
s10-eth2		fe80::18fa:8bff:fe1f:cc38	766	0
s11-eth2		fe80::bc7d:4ff:feb2:89c6	765	0
s12-eth2		fe80::445d:faff:fe26:ff0a	540	2
s14-eth2		fe80::b845:20ff:fe09:e4a7	765	0
s15-eth2		fe80::dc2d:93ff:fec5:fb3e	562	0
s17-eth2		fe80::c45d:bfff:fe89:601	15142	2
s18-eth2		fe80::18ea:48ff:fed8:1755	7087	0

s18-eth2	fe80::18ea:48ff:fed8:1755	7099	0
enp0s3	10.0.2.15	2	0
s1-eth3	fe80::68d5:98ff:fec1:3c75	572	0
s2-eth3	fe80::5487:8fff:fefb:a314	785	0
s3-eth3	fe80::d003:9dff:fea0:725d	7388	0
s4-eth3	fe80::80e8:1cff:fe35:e161	7388	0
s5-eth3	fe80::d824:33ff:fe5d:575d	740	0
s8-eth3	fe80::1803:3cff:fe1a:f84c	8102	0
s9-eth3	fe80::a458:8aff:fe33:eeed	786	0
s10-eth3	fe80::f029:1fff:fefb:d263	785	0
s11-eth3	fe80::5040:3fff:fe1d:704d	784	0
s12-eth3	fe80::401d:42ff:fe8d:3f5c	540	0
s13-eth3	fe80::c8b3:deff:febe:89fc	540	0
s14-eth3	fe80::4ce6:d7ff:fe16:7024	449	0
s15-eth3	fe80::58c5:e3ff:febd:584d	569	0
s7-eth3	fe80::380c:2eff:feda:4487	8110	0
s17-eth3	fe80::f02d:6dff:fe0d:a19f	8110	0
s18-eth3	fe80::4442:1cff:fe84:dcb3	8520	2
s1-eth4	fe80::b409:23ff:fe1a:c3b5	569	0
s5-eth4	fe80::e035:fbff:febd:d7fb	739	0
s9-eth4	fe80::84c3:9aff:fefd:d879	570	0
s17-eth4	fe80::e434:feff:fefa:ee20	995	2
s11-eth4	fe80::ecae:83ff:fedc:eef7	995	2
s13-eth4	fe80::7889:14ff:fe45:72dc	7093	0
s3-eth4	fe80::c0d5:daff:feb6:501b	436	0
s14-eth4	fe80::e873:eaff:fe37:8d24	436	0
s7-eth4	fe80::8b1:40ff:fe2f:b167	448	0
s7-eth5	fe80::3cd3:2bff:fe62:876c	448	0
s1-eth5	fe80::f85b:ccff:fe55:1993	448	0
s9-eth5	fe80::9c8b:77ff:fe16:10ec	563	0
s5-eth5	fe80::d410:21ff:fe5b:a9f	851	2
s13-eth5	fe80::86b:dbff:fef1:ee	8343	0
any	none	414521	20

Figura 77. Paquetes enviados a través de las interfaces de los switches

Efectivamente los paquetes se enrutan por el camino escogido por el módulo: 18:17:11:12:5

En algunas ocasiones el módulo de balanceo de carga no escoge la mejor opción para algunos caminos, esto es debido a que la carga de algunos enlaces es muy cercana, Por ello es mejor enviar solicitudes de ping de 64000 Bytes para diferenciar mejor los caminos ocupados de los libres.

ping DirIPHost -s 64000

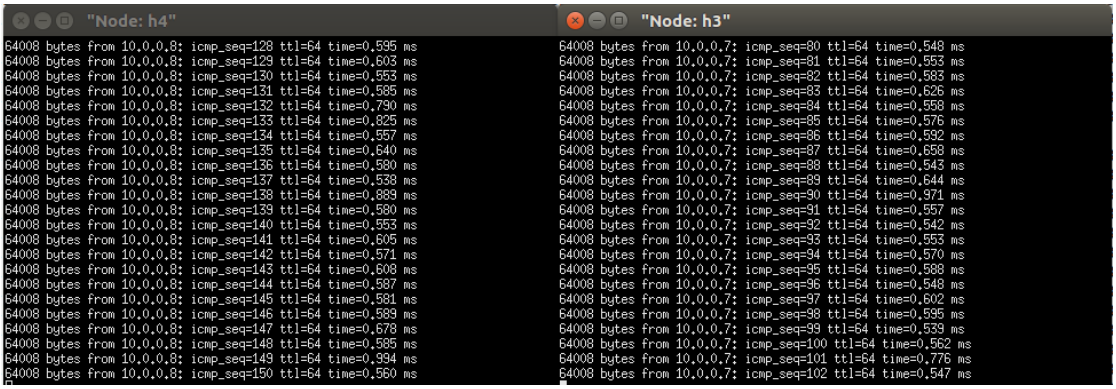


Figura 78. Solicitudes de ping de 64000 Bytes enviadas desde h4 a h8 y de h3 a h7

Se aplica nuevamente el módulo de balance de carga y se obtienen los siguientes resultados:

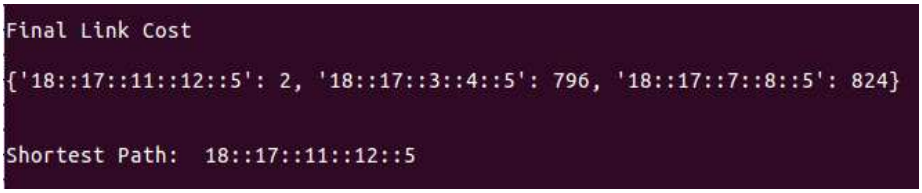


Figura 79. Selección del camino más corto y de menor carga entre h4 y h8

En la imagen anterior se puede observar que la carga de los enlaces difiere bastante, de esta manera no habrá ningún problema en la selección del camino más óptimo para el par de hosts.

A continuación se dibuja en la red los caminos seleccionados anteriormente mediante el módulo para cada par de hosts:

El camino elegido para los hosts h4 y h8 se señaliza con color negro, el camino para los hosts h3 y h7 aparece señalado con color rojo, y finalmente para el enlace entre los hosts h1 y h8 se utilizan flechas moradas (ver figura 80).

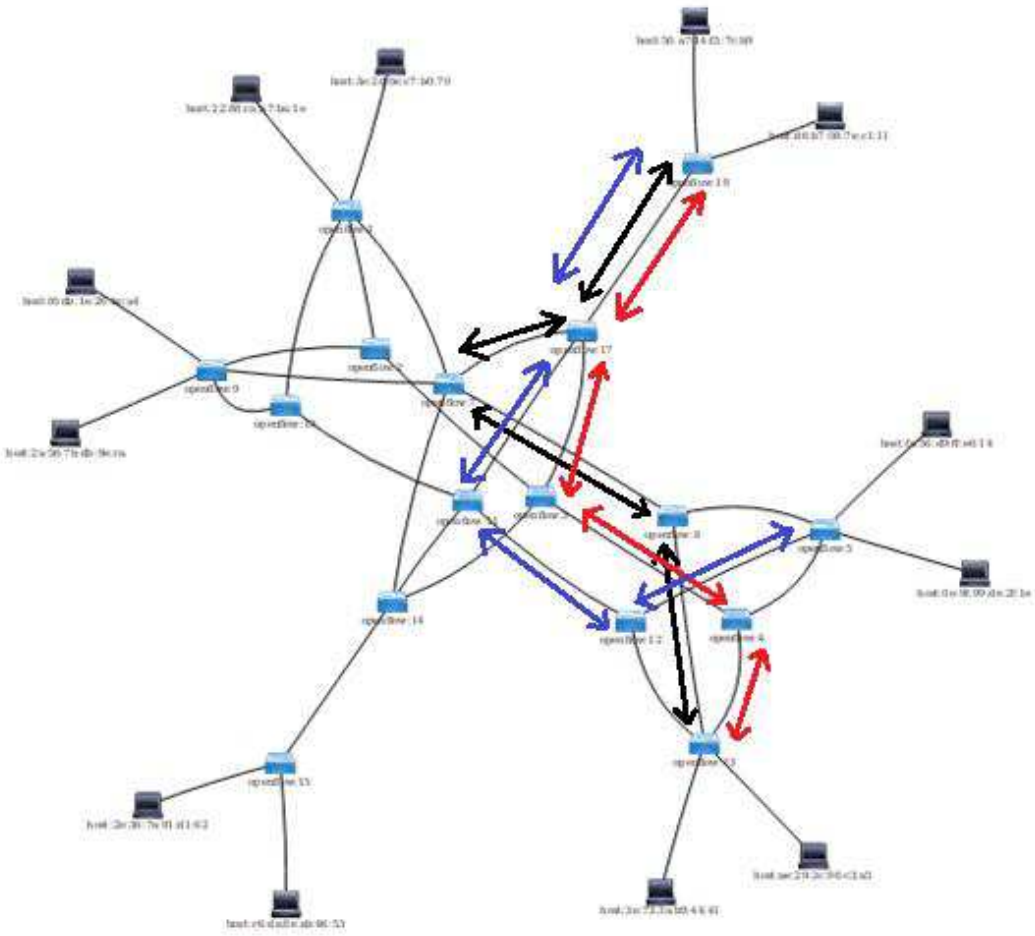


Figura 80. Caminos seleccionados para los pares de hosts.

4.1.2.4 Estadísticas de solicitudes ping

- **Sin aplicar balance de carga**

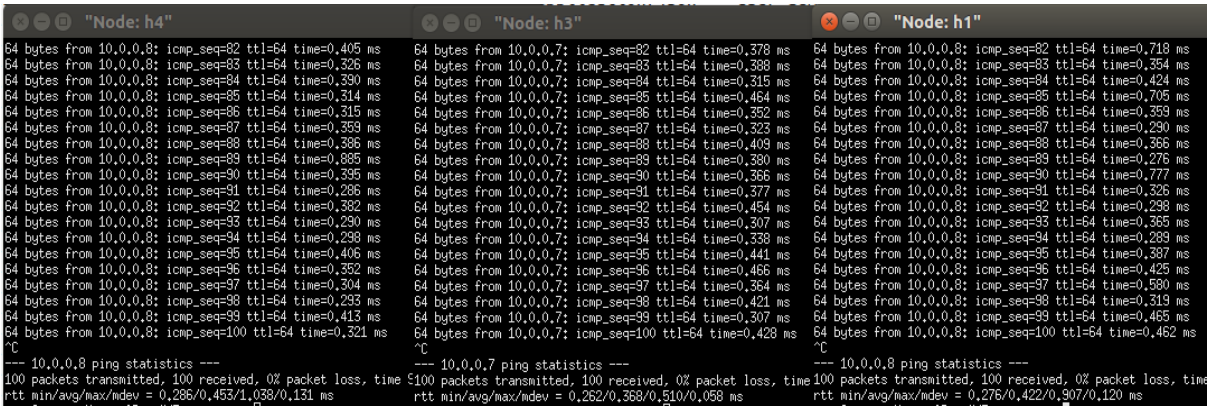


Figura 81. Estadísticas de tiempo de solicitudes ping sin balance de carga

Par de hosts	Tiempo mínimo (ms)	Tiempo promedio (ms)	Tiempo máximo (ms)
Ping h4 a h8	0,286	0,453	1,038
Ping h3 a h7	0,262	0,368	0,510
Ping h1 a h8	0,276	0,422	0,907

Tabla 8. Estadísticas de tiempo de las solicitudes de ping sin balance de carga

- **Aplicando módulo balance de carga**

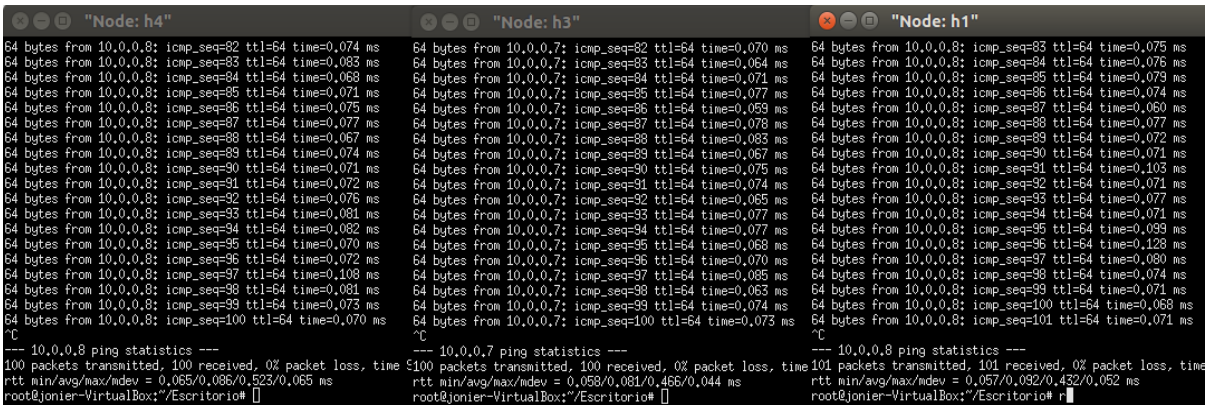


Figura 82. Estadísticas de tiempo de solicitudes de ping con balanceo de carga

Par de hosts	Tiempo mínimo (ms)	Tiempo promedio (ms)	Tiempo máximo (ms)
Ping h4 a h8	0,065	0,086	0,523
Ping h3 a h7	0,058	0,081	0,466
Ping h1 a h8	0,057	0,092	0,432

Tabla 9. Estadísticas de solicitudes de ping con balanceo de carga

4.1.2.5 Conclusiones

Como se puede observar en las tablas 8 y 9, los resultados obtenidos al realizar las solicitudes de ping entre varios hosts al mismo tiempo, cuando se aplica balance de carga y cuando no, muestran unos retardos de tiempo satisfactorios a la hora de equilibrar la carga en la red. El tiempo promedio se redujo 5 veces aproximadamente en todas las parejas de hosts, mejorando considerablemente el comportamiento de la red.

Gracias al balanceo de carga en una red se puede mejorar la velocidad de transmisión de los paquetes entre diferentes puntos de la red, especialmente en redes grandes donde existen muchos caminos por los cuales se pueden enrutar los paquetes.

Cabe resaltar que normalmente los módulos para balancear la carga en una red no siempre trabajan de la forma más adecuada, y esto se debe a que no siempre se elige los enlaces más óptimos, pues estos algoritmos se basan en la obtención de caminos más cortos, por lo que puede elegir el camino con menos saltos pero con una carga excesiva. Esto se puede ver en la figura 56 donde solo se analiza el camino más corto sin estudiar su carga y la de los demás enlaces.

4.2 USANDO CONTROLADOR FLOODLIGHT

Se utiliza el mismo código de Python de la red implementada para el controlador OpenDaylight para realizar las pruebas de balance de carga

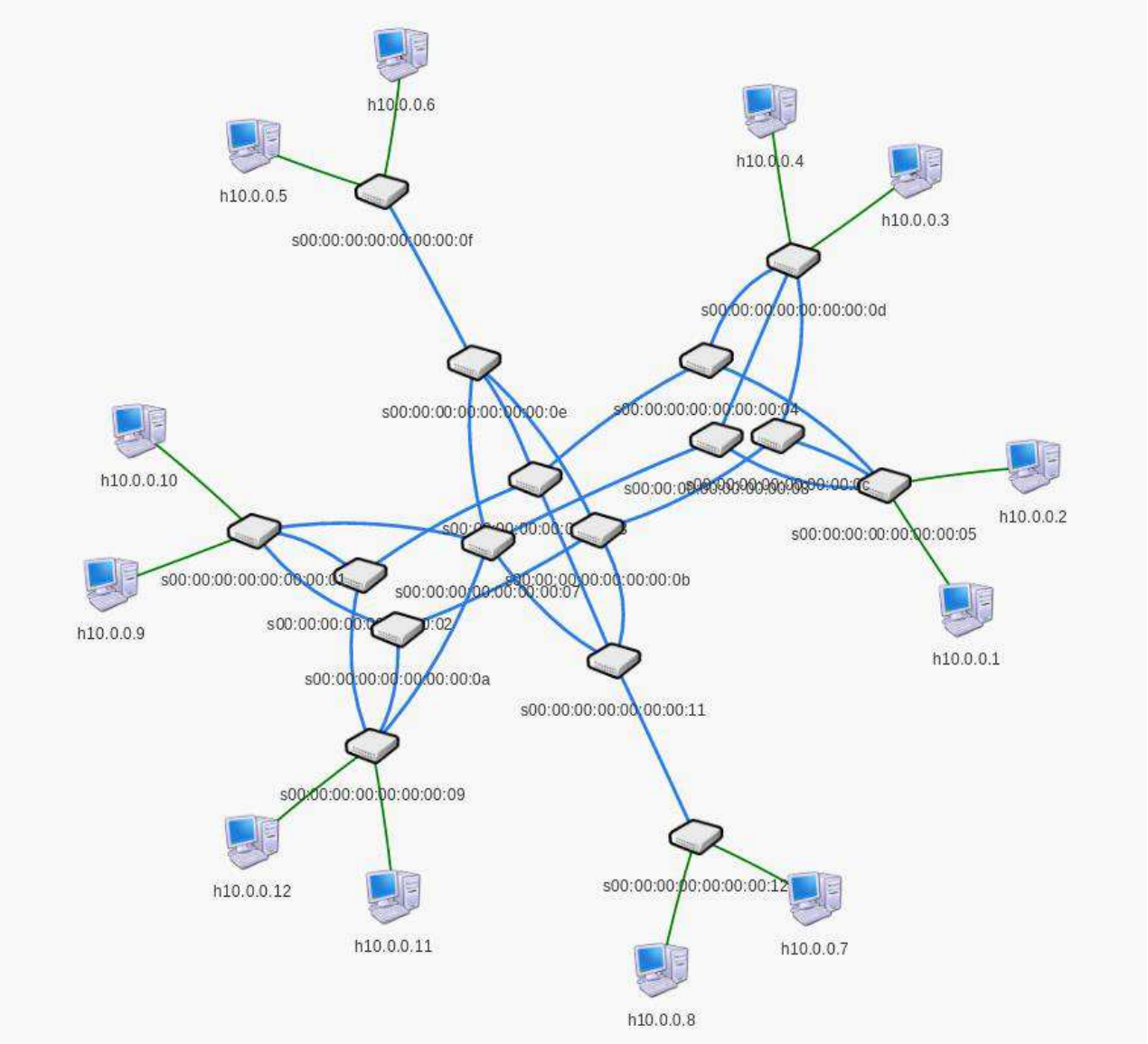


Figura 83. Visualización de la red personalizada con la Api Web de Floodlight

Cabe aclarar que los números de los switches aparecen en sistema hexadecimal.

Nota: las pruebas y sentencias utilizadas en esta sección son idénticas a las que se usaron para el controlador OpenDaylight, por ello se omiten muchos pasos y solo se muestran los resultados de las pruebas.

4.2.1 Prueba del camino más corto o de menor carga

Nota: El módulo de balance de carga implementado fue creado por usuarios de **GITHUB**, con algunas modificaciones hechas por nosotros para acoplarlo a las versiones actuales de software y se soluciona algunos problemas presentados al ejecutar el código.
Este código de balance de carga no es el mismo usado para el controlador OpenDaylight, sin embargo es muy similar. En el **anexo 3** se puede encontrar dicho código**.**

4.2.1.1 Ancho de banda sin usar el módulo de balance de carga

- Ancho de banda entre h1 y h12

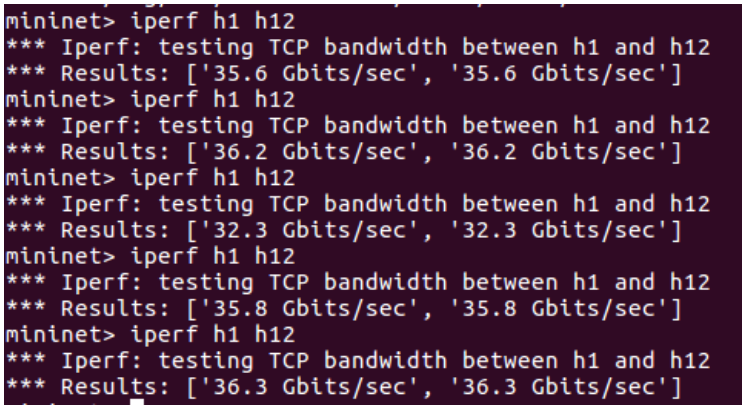


Figura 84. Ancho de banda entre h1 y h12

- Ancho de banda entre h5 y h10

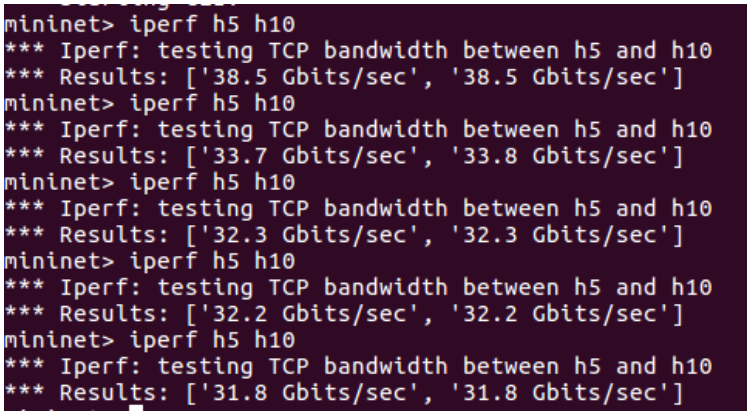


Figura 85. Ancho de banda entre h5 y h10

4.2.1.2 Ancho de banda utilizando el módulo de balance de carga.

Para ejecutar el código se abre una nueva terminal y se escribe la siguiente sentencia:

sudo python LoadBalancerF.py

- **Aplicando balance de carga entre los hosts h1 y h12**

Al ejecutarse el módulo, se piden 3 datos:

Host 1 y Host 2: Hosts entre los que se aplicará balance de carga
Host 3: Host vecino al host h2.

Después de ingresar estos 3 datos aparece la información de las direcciones IP, MAC, puertos de los dispositivos, la carga de los enlaces en ese instante junto con el o los camino más cortos, obtenido mediante la implementación del algoritmo de *Dijkstra*.

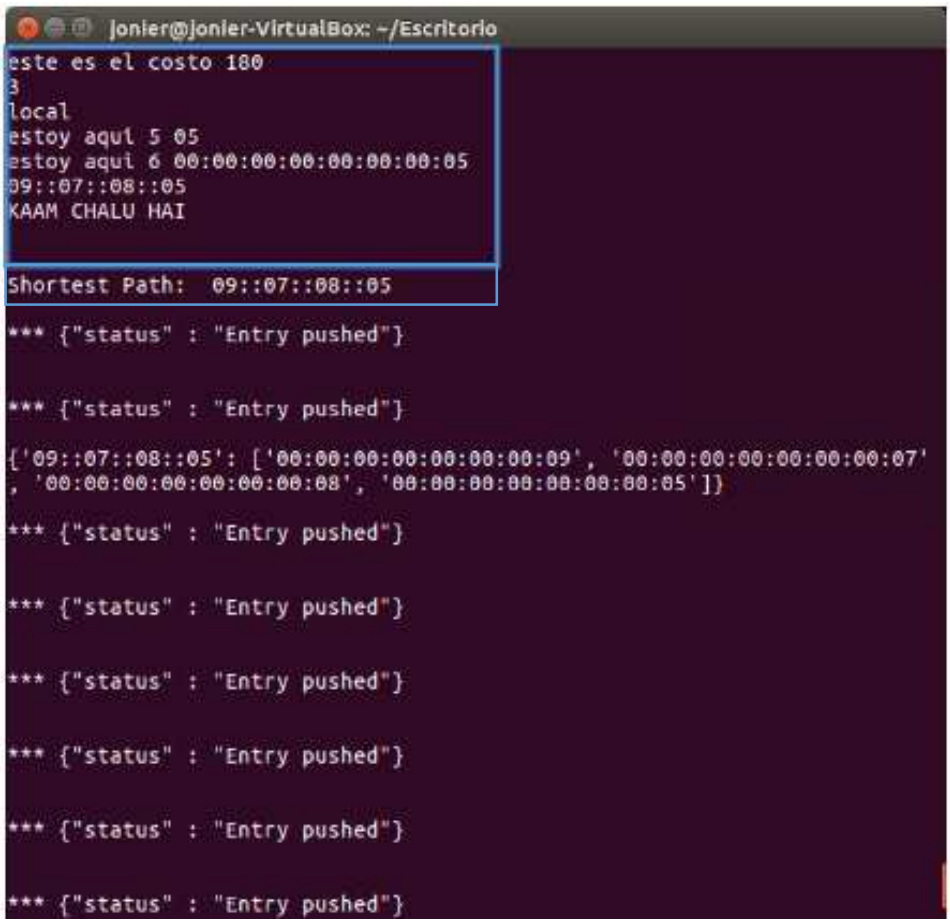


Figura 86. Información de la carga del o los caminos más cortos e instalación de flujos.



Figura 87. Camino más corto seleccionado con su respectiva carga.

- Ancho de banda entre h1 y h12

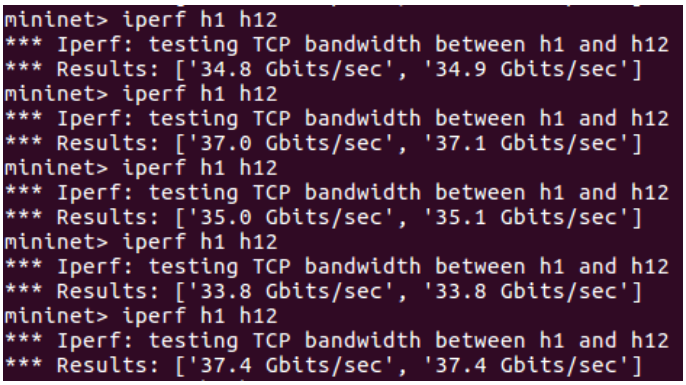


Figura 88. Ancho de banda entre h1 y h12

- **Aplicando balance de carga entre los hosts h5 y h10**

```
este es el costo 198
3
4
estoy aqui 5 0f
estoy aqui 6 00:00:00:00:00:00:00:0f
01::07::0e::0f
KAAM CHALU HAI


Shortest Path:  01::07::0e::0f

*** {"status" : "Entry pushed"}


*** {"status" : "Entry pushed"}

{'01::07::0e::0f': ['00:00:00:00:00:00:00:01', '00:00:00:00:00:00:00:07', '00:00
:00:00:00:00:00:0e', '00:00:00:00:00:00:00:0f']}
```

Figura 89. Información de la carga del o los caminos más cortos e instalación de flujos

```
Paths (SRC TO DST)

{'01::07::0e::0f': ['00:00:00:00:00:00:00:01', '00:00:00:00:00:00:00:07', '00:00
:00:00:00:00:00:0e', '00:00:00:00:00:00:00:0f']}

Final Link Cost (First To Second Switch)

{'01::07::0e::0f': 198}
```

Figura 90. Camino más corto seleccionado con su respectiva carga.

```
mininet> iperf h5 h10
*** Iperf: testing TCP bandwidth between h5 and h10
*** Results: ['35.2 Gbits/sec', '35.2 Gbits/sec']
mininet> iperf h5 h10
*** Iperf: testing TCP bandwidth between h5 and h10
*** Results: ['36.9 Gbits/sec', '37.0 Gbits/sec']
mininet> iperf h5 h10
*** Iperf: testing TCP bandwidth between h5 and h10
*** Results: ['34.9 Gbits/sec', '34.9 Gbits/sec']
mininet> iperf h5 h10
*** Iperf: testing TCP bandwidth between h5 and h10
*** Results: ['39.0 Gbits/sec', '39.1 Gbits/sec']
mininet> iperf h5 h10
*** Iperf: testing TCP bandwidth between h5 and h10
*** Results: ['36.4 Gbits/sec', '36.5 Gbits/sec']
```

Figura 91. Ancho de banda entre h5 y h10

4.2.1.3 Comparación de los resultados obtenidos

- **Ancho de banda**
  - *Entre los hosts h1 y h12*

Sin módulo de balance de carga	Con módulo de balance de carga
35,6 Gbit/s	34,9 Gbit/s
36,2 Gbit/s	37,1 Gbit/s
32,3 Gbit/s	35 Gbit/s
35,8 Gbit/s	33,8 Gbit/s
36,3 Gbit/s	37,4 Gbit/s

Tabla 10. Comparación del ancho de banda entre h1 y h12 con y sin balanceo de carga

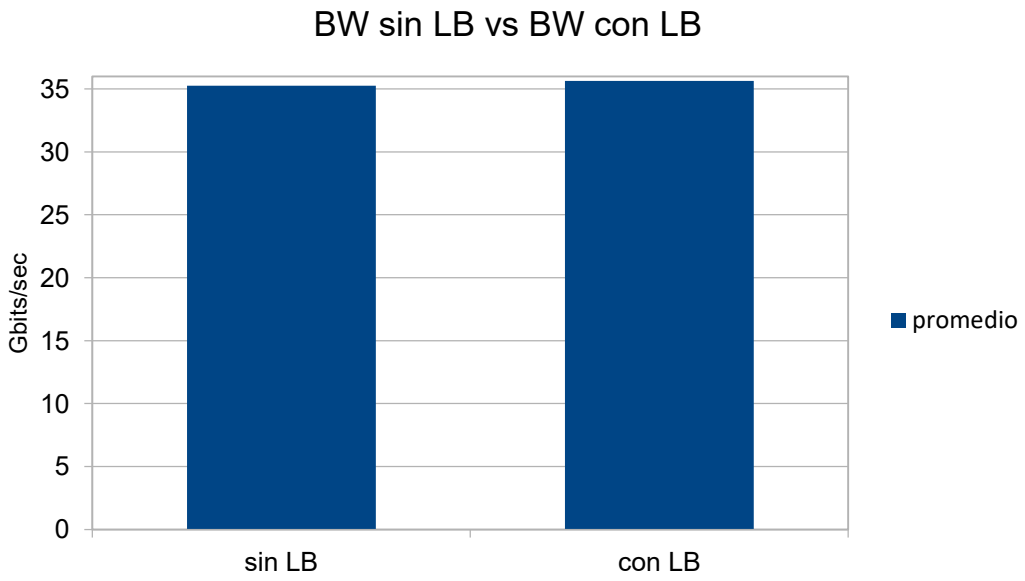


Figura 92. Promedio del ancho de banda entre h1 y h12 con y sin balanceo de carga

Los valores promedio de los resultados presentados en la tabla son: 35,24 Gbit/s para la red sin el módulo de balanceo de carga y 35,64 Gbit/s al aplicar el módulo de balanceo de carga.

- *Entre los hosts h5 y h10*

Sin módulo de balance de carga	Con módulo de balance de carga
38,5 Gbit/s	35.2 Gbit/s
33,8 Gbit/s	37 Gbit/s
32,3 Gbit/s	34,9 Gbit/s
32,2 Gbit/s	39,1 Gbit/s
31,8 Gbit/s	36,5 Gbit/s

Tabla 11. Comparación del ancho de banda entre h5 y h10 con y sin balanceo de carga

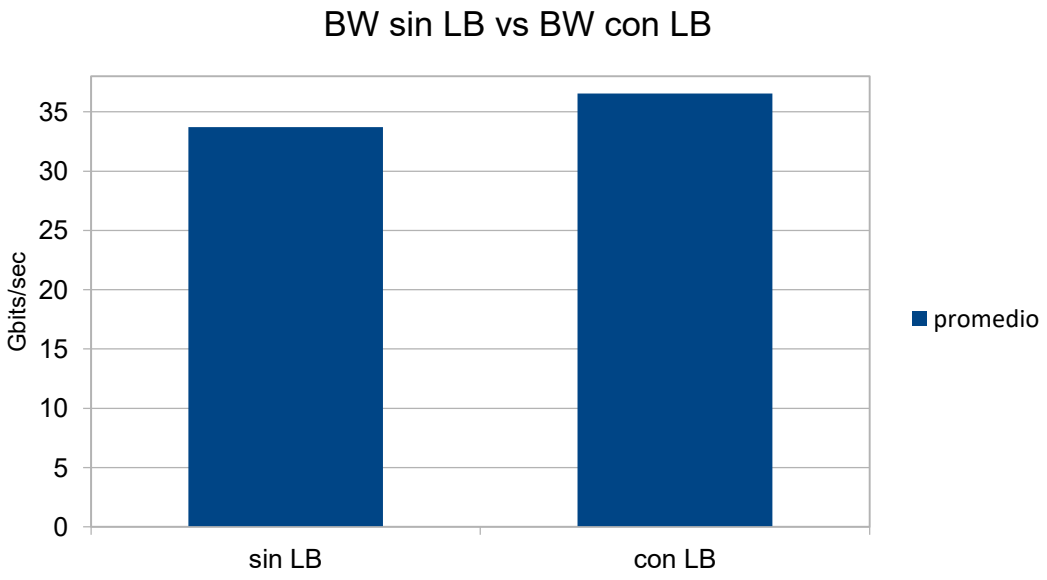


Figura 93. Promedio del ancho de banda entre h5 y h10 con y sin balanceo de carga

Los valores promedio de los resultados presentados en la tabla, son: 33,72 Gbit/s para la red sin el módulo de balanceo de carga y 36,54 Gbit/s para la red con el módulo de balanceo de carga.

Aunque el valor promedio del ancho de banda aumenta de un caso respecto al otro, las mejoras son muy diminutas a diferencia de los resultados obtenidos con el controlador OpenDaylight. La razón de esto, radica en que el controlador Floodlight cuenta con un gran número de módulos por defecto, que carga de manera automática cada vez que se ejecuta el controlador, y entre estos módulos se encuentra el de balance de carga.
Este código de balance de carga es realizado en Python siendo una alternativa al implementado por Floodlight, que se encuentra en java.

- **Solicitudes ping**
  - *Ping entre h5 y h10*

Estadísticas de tiempo sin implementar el módulo de load balancer:

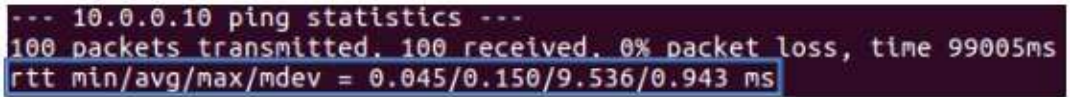


Figura 94. Estadísticas de las solicitudes ping entre h5 y h10 sin balanceo de carga

Estadísticas de tiempo al implementar el módulo de load balancer:



Figura 95. Estadísticas de las solicitudes ping entre h5 y h10 al aplicar balanceo de carga

	Sin balanceo de carga	Con balanceo de carga
Tiempo mínimo	0,045 ms	0,04 ms
Tiempo máximo	9,536 ms	0,520 ms
Tiempo promedio	0,150 ms	0,052 ms

Tabla 12. Estadísticas de tiempo entre h5 y h10 con balanceo de carga y sin balanceo de carga

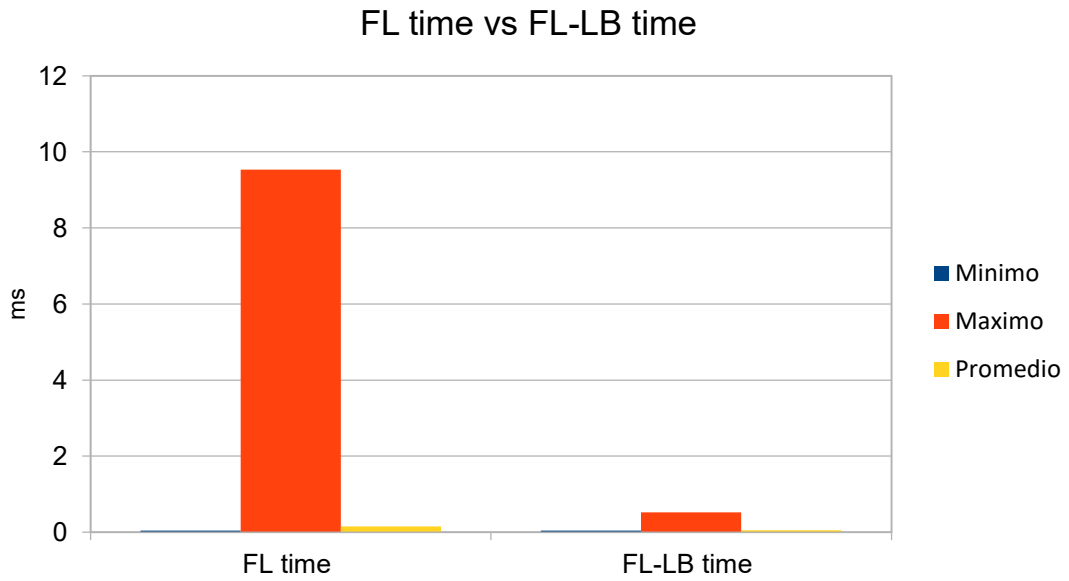


Figura 96. Estadísticas de tiempo entre h5 y h10 con balanceo de carga y sin balanceo de carga

El tiempo promedio para las 100 solicitudes ping es 2,88 veces menor cuando se utiliza el módulo de load balancer (entre menor sea el tiempo mejor es la respuesta).

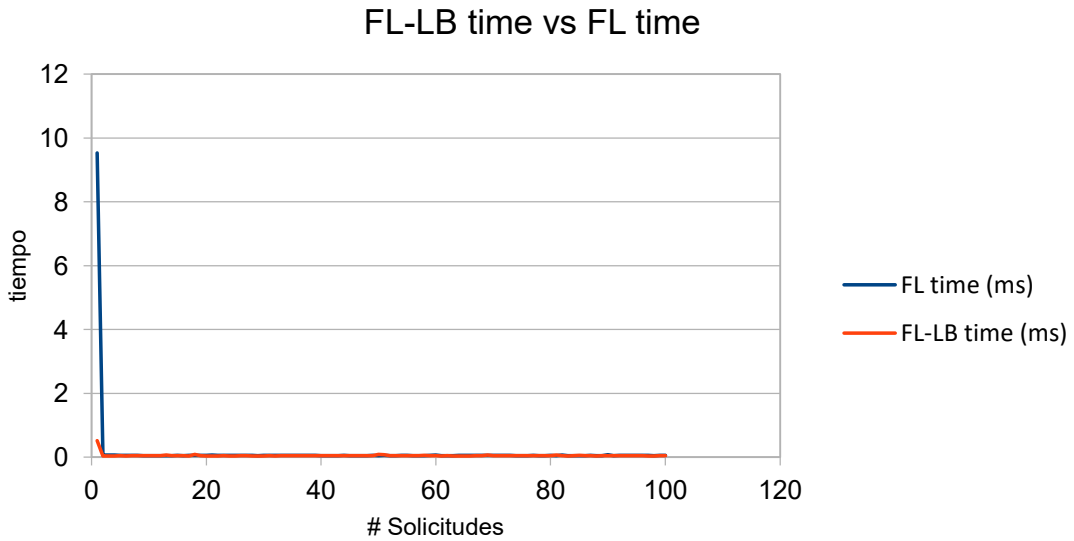


Figura 97. Tiempo de cada solicitud ping entre h5 y h10 con balanceo y sin balanceo de carga

La gráfica anterior de forma más detallada el comportamiento de todas las solicitudes ping, En color azul aparece la gráfica de las solicitudes ping sin utilizar el módulo de balance de carga, y la gráfica naranja al implementar balanceo de carga.

En la gráfica azul se observa que la primera solicitud ping tiene una respuesta en tiempo muy alto, de 9,536 ms, mientras que en la gráfica naranja se obtiene un valor de 0,520 ms para el primer ping, lo cual demuestra que el módulo de balance de carga implementado mejora el comportamiento de la red.

- *Ping entre h1 y h12*

Se realizan 100 solicitudes ping de h1 a h12 y se obtienen los siguientes resultados:

Estadísticas de tiempo sin implementar el módulo de load balancer:



Figura 98. Estadísticas de las ping entre h1 y h12 con balanceo de carga

Estadísticas de tiempo al implementar el módulo de load balancer:

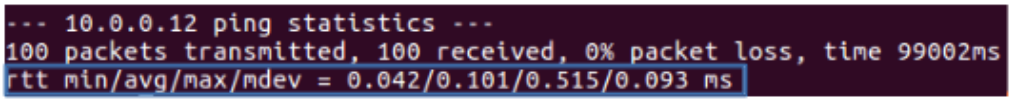


Figura 99. Estadísticas de las solicitudes ping entre h1 y h12 sin balanceo de carga

A continuación se gráfica y tabula los datos obtenidos de las estadísticas de tiempo de las solicitudes de ping realizadas entre el par de hosts:

	Sin balanceo de carga	Con balanceo de carga
Tiempo mínimo	0,039 ms	0,042 ms
Tiempo máximo	14,532 ms	0,515 ms
Tiempo promedio	0,221 ms	0,101 ms

Tabla 13. Comparación de los tiempos de las solicitudes ping entre h5 y h10 con balanceo de carga y sin balanceo de carga

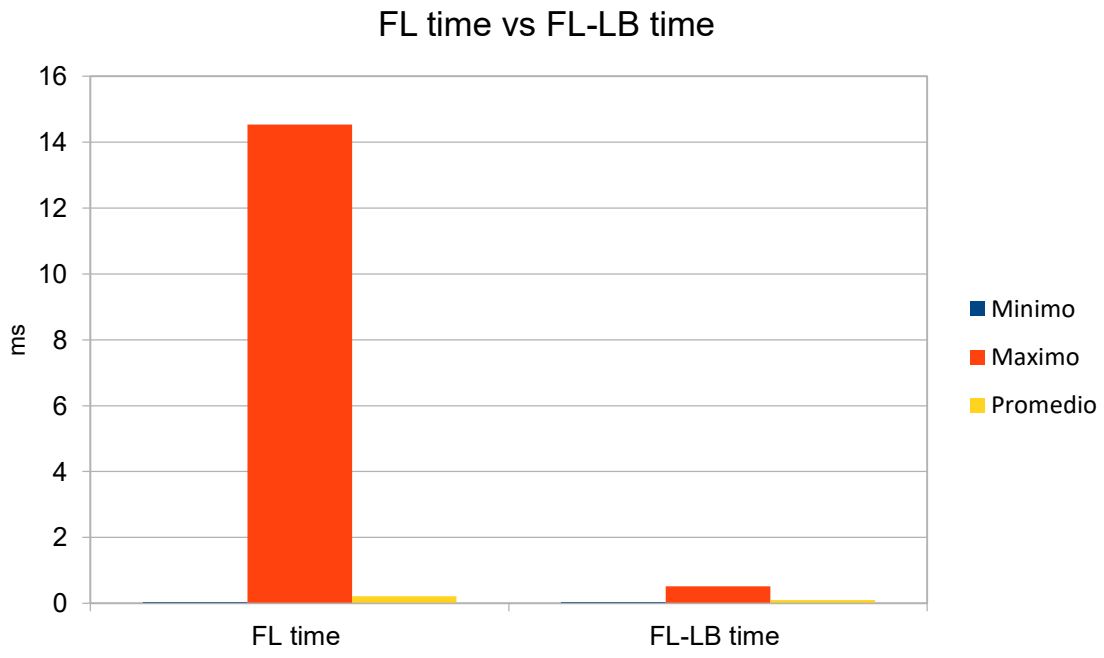


Figura 100. Gráfica de las solicitudes ping entre h1 y h12 con y sin balanceo de carga

El tiempo promedio para las 100 solicitudes ping es 2,19 veces menor cuando se utiliza el módulo de load balancer (entre menor sea el tiempo mejor es la respuesta).

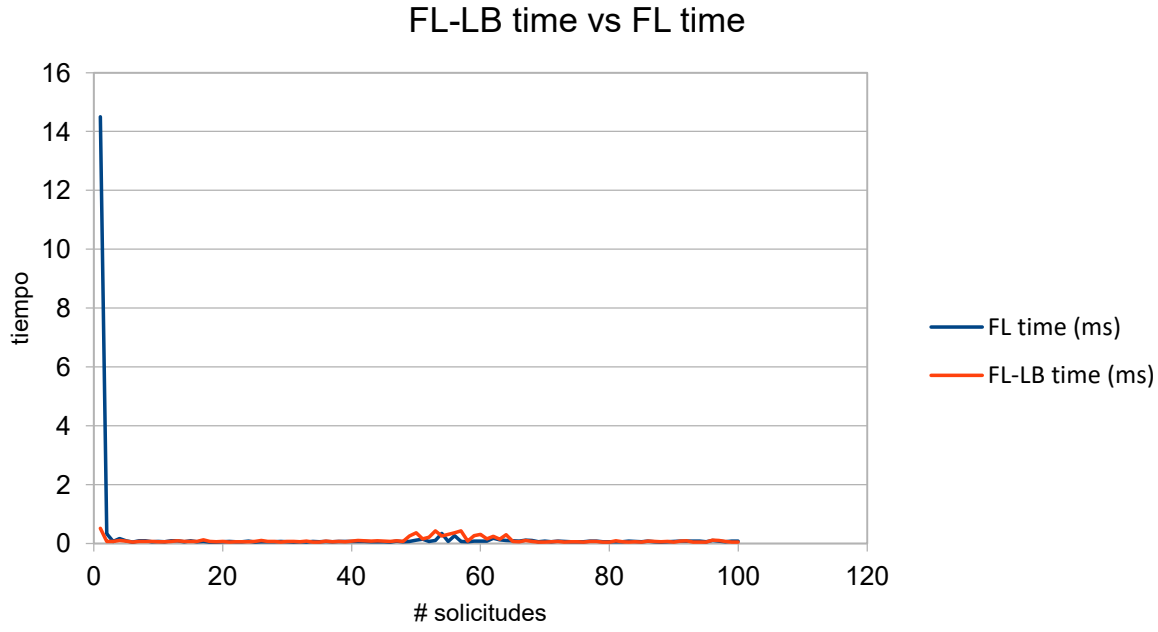


Figura 101. Tiempo que toma cada solicitud ping entre los hosts h1 y h12 cuando se utiliza balanceo de carga y cuando no

La gráfica de color azul corresponde a las solicitudes ping sin utilizar el módulo de balance de carga, y la gráfica naranja al implementarlo.

En la gráfica azul se observa que la primera solicitud ping tiene una respuesta en tiempo muy alto de 14,5 ms, mientras que en la gráfica naranja se obtiene un valor de 0,515 ms para el primer ping, lo cual demuestra que el módulo de balance de carga implementado mejora el comportamiento de la red.

En general, la implementación del código para balanceo de carga en una red controlada con Floodlight, muestra un mejoramiento significativo al reducir de forma drástica el tiempo de la primera solicitud ping donde el par de dispositivos finales se comunican por primera vez. Esto conlleva a que el tiempo promedio de las solicitudes ping disminuya considerablemente.

4.2.2 Prueba de balance de carga

A diferencia de OpenDaylight, Floodlight no envía las solicitudes ping por toda la red cuando se intentan comunicar dos hosts, por el contrario, este controlador determina el camino más apropiado para establecer la conexión y genera un flujo dentro de sus tablas. En la mayoría de los casos, el camino seleccionado suele ser el más corto y generalmente uno solamente. La siguiente prueba es similar a la realizada en OpenDaylight donde se observa que caminos define el controlador para la comunicación entre dos hosts cuando se utiliza el código de balanceo de carga y cuando no.

- **Caminos seleccionados entre los host h3 y h8**

Inicialmente se realiza la comunicación entre los hosts h3 y h8, los caminos obtenidos al establecer la conexión entre estos dos hosts sin el código de balanceo de carga y con el balanceo de carga es el siguiente:

Camino entre h3 y h8 sin load balancer
0d::0c::0b::11::12
0d::08::07::11::12

Camino entre h3 y h8 con load balancer
0d::0c::0b::11::12

Cuando se establece la conexión entre los dos hosts sin la implementación del código de balanceo de carga, los módulos por defecto con los que trabaja el controlador asignan dos caminos para la comunicación, uno de ida y otro de vuelta. Por el contrario, el código de balanceo de carga solamente asigna un camino para la comunicación. La asignación de dos caminos a la comunicación resulta ventajoso si consideramos esto como una manera para evitar saturar los enlaces y evitar aumentar la carga de ese camino. Por otro lado, resulta poco ventajoso si consideramos que otros hosts intentarán comunicarse entre ellos y esto implica una distribución y asignación ineficiente de los recursos disponibles en la red.

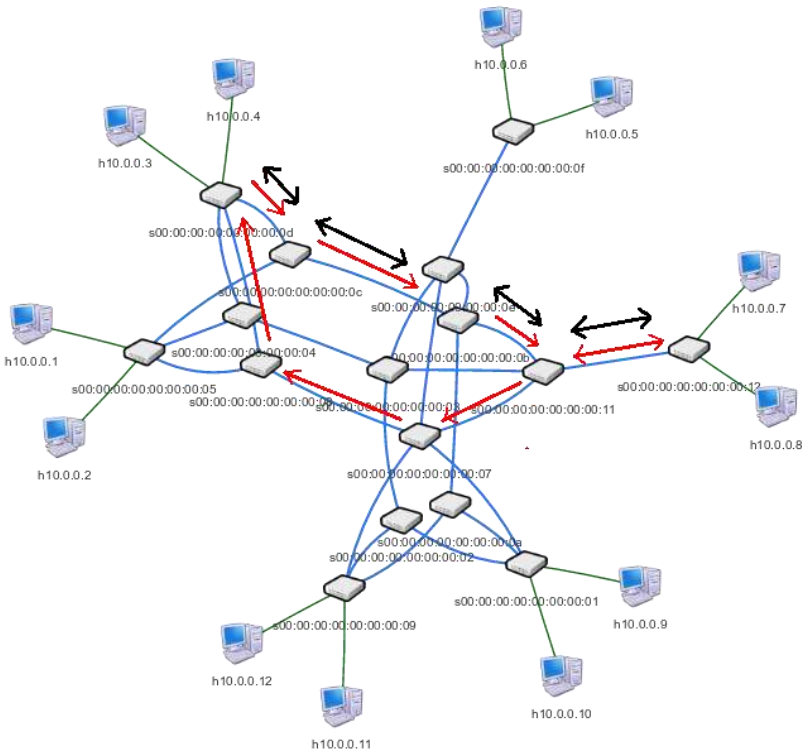


Figura 102. Caminos seleccionados entre los hosts h3 y h8

En la imagen anterior se muestra el camino asignado para el par de hosts.
Los caminos de color negro son los seleccionados con el código de balanceo de carga y los caminos en rojo los tomados sin el código de balanceo de carga, es decir, con los módulos propios del controlador.

- **Caminos seleccionados entre los host h4 y h7**

A continuación se deja al host h1 enviado solicitudes ping al host h8 mientras se intenta establecer conexión entre los host h4 y h7, con el fin de ocupar ese enlace. En ambos casos se asigna un solo camino y diferente a los asignados anteriormente.

Camino entre h4 y h7 sin load balancer
0d::04::03::11::12

Camino entre h4 y h7 con load balancer
0d::08::07::11::12

En la siguiente imagen se muestran los caminos tomados para cada uno de los casos. El color negro indica el camino tomado con el código de balanceo de carga y el camino en rojo el tomado sin el código de balanceo de carga, es decir, con los módulos propios del controlador.

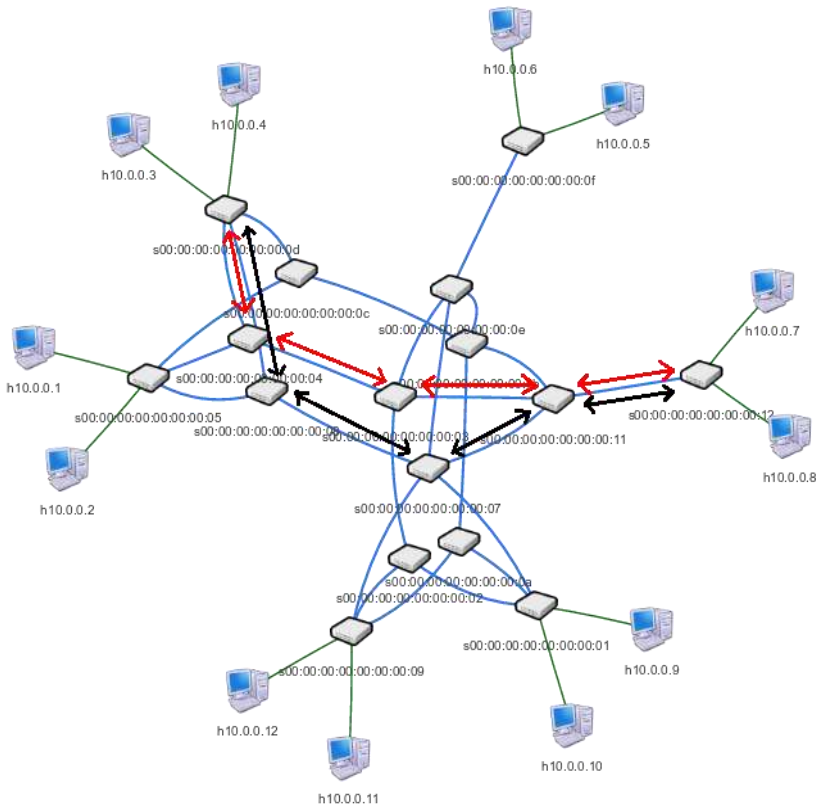


Figura 103. Caminos seleccionados entre los hosts h4 y h7

- **Caminos seleccionados entre los host h12 y h6**

De la misma manera, se deja al host h1 enviado solicitudes ping al host h8 y el host h4 enviando solicitudes ping al host h7 con el fin de ocupar dichos enlaces.

Camino entre h12 y h6 sin load balancer
0f::0e::07::09

Camino entre h12 y h6 con load balancer
0f::0e::07::09

En este caso, los caminos asignados para la comunicación entre h12 y h6 en las dos situaciones es el mismo. El controlador asigna exactamente el mismo camino que el código de balanceo de carga al analizar la carga que presenta la red hasta este momento.

En la siguiente imagen se muestran los caminos tomados para cada uno de los casos. El color negro indica el camino tomado con el código de balanceo de carga y el camino en rojo el tomado sin el código de balanceo de carga, es decir, con los módulos propios del controlador.

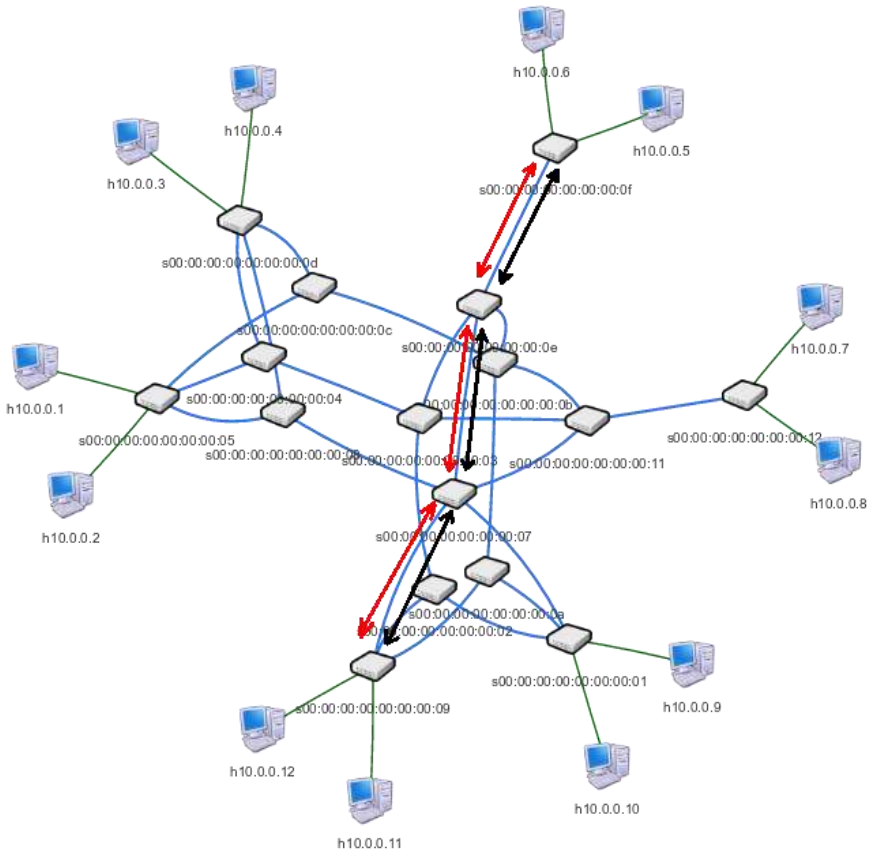


Figura 104. Caminos seleccionados entre los hosts h12 y h6

- **Caminos seleccionados entre los host h1 y h5**

Camino entre h1 y h5 sin Load Balancer
0f::0e::0b::0c::05
0f::0e::03::04::05

Camino entre h1 y h5 con Load Balancer
0f::0e::03::04::05

Finalmente, para la conexión entre h1 y h5 el controlador asigna dos caminos para la comunicación mientras que con el código de balanceo de carga se asigna uno solo.

En la siguiente imagen se muestran los caminos tomados para cada uno de los casos. El color negro indica el camino tomado con el código de balanceo de carga y los caminos en rojo los tomados sin el código de balanceo de carga, es decir, con los módulos propios del controlador.

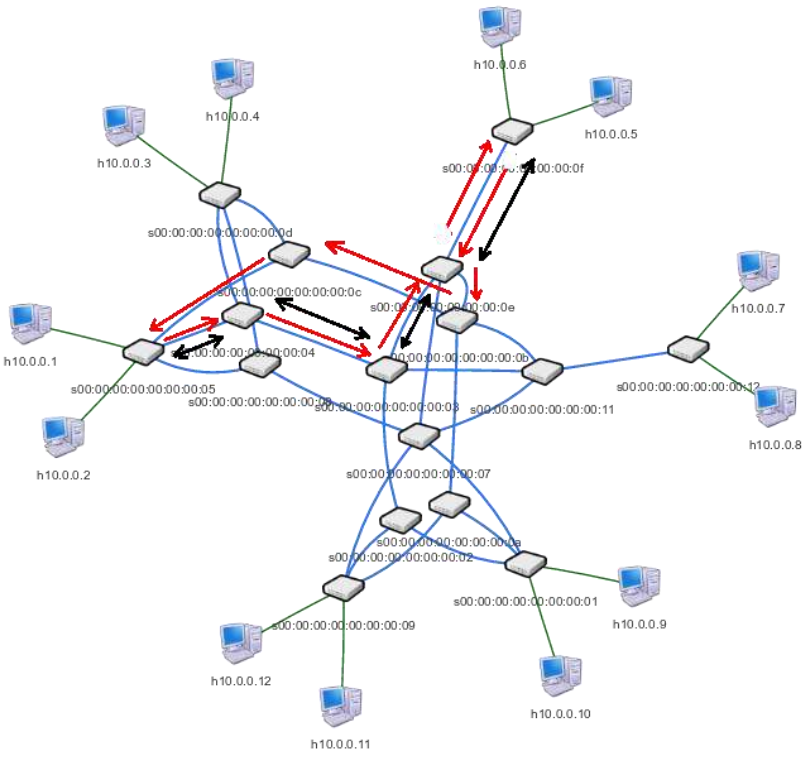


Figura 105. Caminos seleccionados entre los hosts h1 y h5

El comportamiento general del controlador muestra una muy buena asignación y distribución de los recursos de la red, casi comparable con el funcionamiento del código de balanceo de carga, el cual realiza bastante bien la asignación de caminos analizando la carga actual de los enlaces de la red. En este sentido, el controlador Floodlight cuenta con un gran compendio de módulos que permiten una buena comunicación entre hosts en la red.

CAPÍTULO 5. IMPLEMENTACIÓN DE POLÍTICAS DE CALIDAD DE SERVICIO (QOS)

Hay diferentes alternativas para establecer QoS en las redes definidas por software. Por ejemplo se podría establecer mediante el programa Visual Network Description (VND) que consiste en una interfaz gráfica para construir redes SDN de manera sencilla, otra forma es utilizando módulos para cada controlador que cumplen estas funciones. Una alternativa muy viable es a través de Open vSwitch (OVS) que es un software de código libre que se comporta como switch virtual. OVS ofrece un modelo centralizado más simple de administrar las redes permitiendo implementar VLANs, túneles GRE, VXLANs, políticas básicas de QoS, IPsec, LACP entre otras.

OVS funciona independientemente del controlador que se esté usando, por esta razón se opta por usar esta opción, ya que si se utiliza otra de las alternativas se requiere implementar módulos y sentencias distintas para controlador, así como también los códigos son mucho más sencillos de implementar en OVS que en cada controlador por aparte.

5.1 PRIORIZACIÓN POR INTERFACES DE SALIDA

Para garantizar calidad de servicio en las interfaces de los switches, se restringe la velocidad de cada una de ellas, de tal forma que existan unos enlaces con más prioridad que otros.

Primero se crea una red simple compuesta por 1 switch y 8 hosts.

sudo mn --topo single,8 --controller=remote,ip=127.0.0.1,port=6653

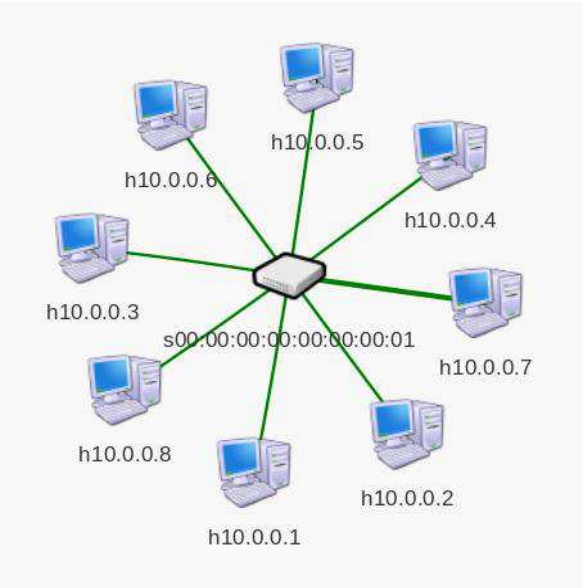


Figura 106. Red sencilla visualizada en la Web API de Floodlight

5.1.1 Ancho de banda sin interfaces restringidas

Primero se toman mediciones del ancho de banda de las interfaces de salida que se pretenden restringir, de tal manera que se pueda observar en el siguiente ítem que la velocidad de las interfaces de salida efectivamente se restringe al aplicar políticas de calidad de servicio.

- *Ancho de banda entre h2 y h5*

```
mininet> iperf h2 h5
*** Iperf: testing TCP bandwidth between h2 and h5
*** Results: ['36.2 Gbits/sec', '36.3 Gbits/sec']
mininet> iperf h2 h5
*** Iperf: testing TCP bandwidth between h2 and h5
*** Results: ['32.8 Gbits/sec', '32.8 Gbits/sec']
mininet> iperf h2 h5
*** Iperf: testing TCP bandwidth between h2 and h5
*** Results: ['42.3 Gbits/sec', '42.3 Gbits/sec']
```

Figura 107. Ancho de banda entre h2 y h5

- *Ancho de banda entre h3 y h7*

```
mininet> iperf h3 h7
*** Iperf: testing TCP bandwidth between h3 and h7
*** Results: ['30.0 Gbits/sec', '30.1 Gbits/sec']
mininet> iperf h3 h7
*** Iperf: testing TCP bandwidth between h3 and h7
*** Results: ['29.1 Gbits/sec', '29.1 Gbits/sec']
mininet> iperf h3 h7
*** Iperf: testing TCP bandwidth between h3 and h7
*** Results: ['33.8 Gbits/sec', '33.8 Gbits/sec']
```

Figura 108. Ancho de banda entre h3 y h7

- *Ancho de banda entre h1 y h4*

```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['37.3 Gbits/sec', '37.3 Gbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['36.2 Gbits/sec', '36.2 Gbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['35.4 Gbits/sec', '35.5 Gbits/sec']
```

Figura 109. Ancho de banda entre h1 y h4

5.1.2 Ancho de banda con interfaces restringidas

Después de iniciar el controlador y la red en Mininet, se abre una nueva una consola y se ingresan las reglas QoS que permiten restringir las velocidades de las interfaces del switch mediante el uso de OVS:

> *sudo ovs-vsctl set port s1-eth5 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=500000000*

Donde se debe especificar el switch, la interfaz, y el ancho de banda máximo que puede alcanzar la interfaz a su salida. Para este caso el switch: s1, la interfaz: eth5, y un BW= 500 Mbits/s.

Se realiza el mismo proceso para la interfaz Ethernet 7, estableciendo un ancho de banda de 4 Gb/s:

> *sudo ovs-vsctl set port s1-eth7 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=4000000000*

De igual manera para la interfaz Ethernet 4, se establece un ancho de banda de 10 Mb/s:

> *sudo ovs-vsctl set port s1-eth4 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=10000000*

Una instrucción alternativa para limitar el ancho de banda de una interfaz es la siguiente:

> *sudo ovs-vsctl set interface s1-eth1 ingress_policing_rate=10000*

Esta instrucción se diferencia de la anterior en que no se establece un ancho de banda máximo, sino un valor definido. Así mismo, para esta instrucción se debe ingresar la velocidad en Mbit y no en Bit como con las otras sentencias.

Las mediciones de ancho de banda se llevan a cabo entre diferentes hosts, no importa si se realiza entre un host u otro, puesto que el ancho de banda es restringido a la salida de la interfaz del switch.

- *Ancho de banda entre h2 y h5*

```
mininet> iperf h2 h5
*** Iperf: testing TCP bandwidth between h2 and h5
*** Results: ['442 Mbits/sec', '460 Mbits/sec']
mininet> iperf h2 h5
*** Iperf: testing TCP bandwidth between h2 and h5
*** Results: ['439 Mbits/sec', '442 Mbits/sec']
mininet> iperf h2 h5
*** Iperf: testing TCP bandwidth between h2 and h5
*** Results: ['439 Mbits/sec', '442 Mbits/sec']
```

Figura 110. Ancho de banda entre h2 y h5

- *Ancho de banda entre h3 y h7*

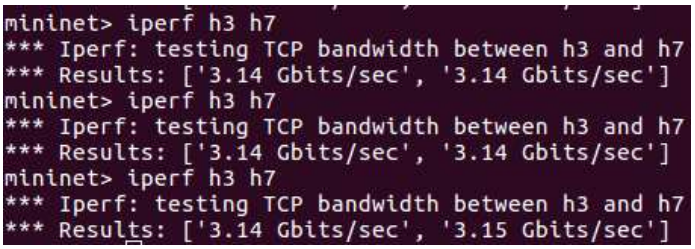


Figura 111. Ancho de banda entre h3 y h7

- *Ancho de banda entre h1 y h4*

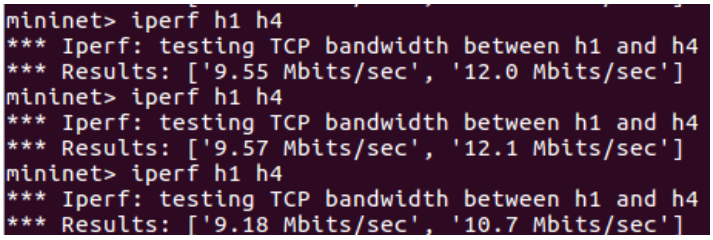


Figura 112. Ancho de banda entre h1 y h4

Para observar las reglas QoS establecidas, se utiliza el siguiente comando de OVS:

sudo ovs-vsctl list qos

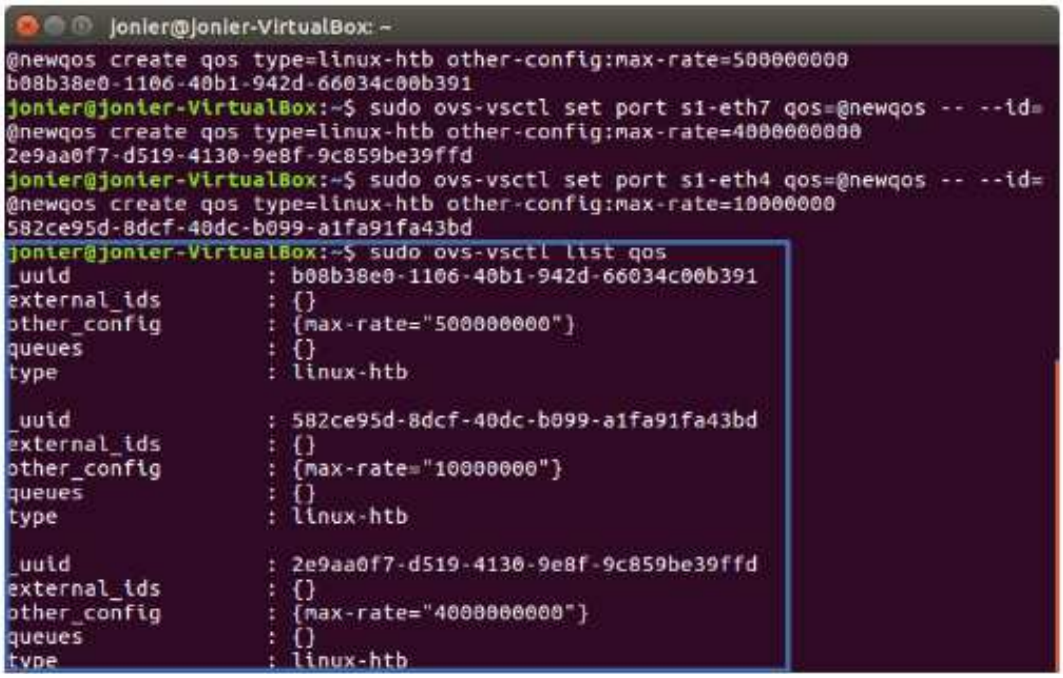


Figura 113. Lista de reglas QoS

En este sencillo ejemplo se puede observar cómo se le da prioridad a diferentes interfaces. En el caso de la interfaz 4 se le asigna una muy baja prioridad de tan solo 10 Mbit/s de ancho de banda como máximo, a la interfaz 5 se le establece una prioridad baja de 500 Mbit/s como máximo, y a la interfaz 7 se le establece una prioridad media limitando su velocidad a 4 Gbit/s. Las interfaces restantes tendrán una prioridad alta que ronda entre 30 y 40 Gbit/s que es el limitante del computador donde se realizan las pruebas.

A continuación se puede observar el ancho de banda de las demás interfaces.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['34.7 Gbits/sec', '34.7 Gbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['35.6 Gbits/sec', '35.6 Gbits/sec']
mininet> iperf h1 h6
*** Iperf: testing TCP bandwidth between h1 and h6
*** Results: ['38.9 Gbits/sec', '39.0 Gbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['36.9 Gbits/sec', '36.9 Gbits/sec']
```

Figura 114. Ancho de banda entre h1 y los hosts que no relaciona ninguna restricción

Cabe aclarar que los límites de ancho de banda son configurados a las salidas de las interfaces, más no a sus entradas, es decir que la limitante de ancho de banda solo se cumple si se envían paquetes desde una interfaz X a la interfaz que se le aplicó QoS, de forma contraria el parámetro de QoS no aplica, y la velocidad de transmisión será la máxima permitida por Mininet.

Para eliminar todas las reglas establecidas de QoS, se debe finalizar la red de Mininet y escribir la siguiente sentencia:

sudo ovs-vsctl --all destroy qos

El manejo y asignación del ancho de banda dentro de una red es un aspecto muy importante. Gestionar los recursos de la red permite dar prioridad minina o máxima a ciertos enlaces y de esta forma no malgastar el ancho de banda en actividades que no son de alta prioridad.

5.2 PRIORIZACIÓN DE TRÁFICO MEDIANTE ENCOLAMIENTO

Se puede priorizar el tráfico de paquetes proveniente de diferentes interfaces o puertos utilizando colas a través de OVS.

Se implementa la misma prueba del ítem anterior para dar prioridad a las interfaces, con la diferencia de que ya no se priorizan las salidas de las interfaces sino las entradas, es decir, los paquetes enviados desde diferentes hosts (diferentes interfaces de entradas) hacia un host (única interfaz de salida), tendrán una prioridad distinta mediante la restricción del ancho de banda.

Para implementar encolamiento primero se crea la regla QoS de la que harán parte las colas, para así proceder a la creación de las diferentes colas que se pretenden implementar, como se muestra a continuación:

sudo ovs-vsctl set port s1-eth1 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=5000000000 queues=1=@q1,2=@q2,3=@q3 -- --id=@q1 create queue other-config:max-rate=500000000 -- --id=@q2 create queue other-config:max-rate=4000000000 -- --id=@q3 create queue other-config:max-rate=10000000

En este ejemplo se establece una regla QoS compuesta por 3 colas y un ancho de banda limitado de 5 Gb/s, por lo que si no relacionan los puertos con las colas, el ancho de banda predeterminado estará limitado por esta velocidad.

Para relacionar las interfaces o puertos de entrada del switch con las diferentes colas, se utiliza la siguiente sentencia de OVS:

sudo ovs-ofctl add-flow s1 in_port=5,actions=set_queue:1,normal

De donde se puede observar que el puerto de entrada 5 del switch S1 se relaciona con la cola número 1. Toda información enviada desde la interfaz 5 (conectada a h5) a la interfaz 1 (conectada a h1) del switch, tendrá una limitante en ancho de banda de 500 Mbps.

Para la interfaz 7 se utiliza la cola número 2, de tal manera que todos los paquetes provenientes de este puerto hacia la interfaz 1, estarán limitados por una velocidad de enlace de 4 Gbps.

sudo ovs-ofctl add-flow s1 in_port=7,actions=set_queue:2,normal

Por último, el puerto de entrada 4 del switch se relaciona con la cola número 3, limitando a 10 Mb/s los paquetes provenientes desde esta interfaz cuando son enviados a la interfaz 1.

sudo ovs-ofctl add-flow s1 in_port=4,actions=set_queue:3,normal

A continuación se realizan mediciones de ancho de banda entre los diferentes hosts relacionados a las interfaces limitadas por colas:

- *Ancho de banda entre h5 y h1*

```
mininet> iperf h5 h1
*** Iperf: testing TCP bandwidth between h5 and h1
*** Results: ['461 Mbits/sec', '466 Mbits/sec']
mininet> iperf h5 h1
*** Iperf: testing TCP bandwidth between h5 and h1
*** Results: ['460 Mbits/sec', '478 Mbits/sec']
mininet> iperf h5 h1
*** Iperf: testing TCP bandwidth between h5 and h1
*** Results: ['463 Mbits/sec', '481 Mbits/sec']
```

Figura 115. Ancho de banda entre h5 y h1

- *Ancho de banda entre h7 y h1*

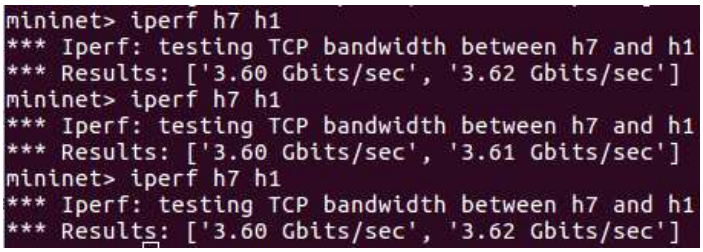


Figura 116. Ancho de banda entre h7 y h1

- *Ancho de banda entre h4 y h1*

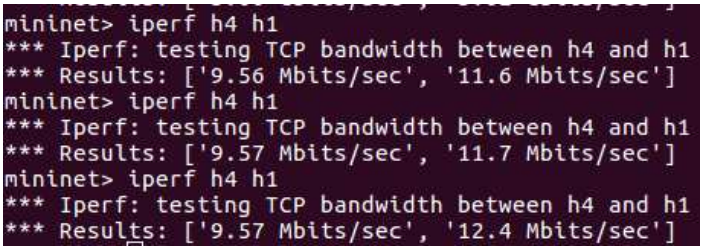


Figura 117. Ancho de banda entre h4 y h1

Para observar la lista de colas establecidas, se utiliza el siguiente comando:

sudo ovs-vsctl list queue

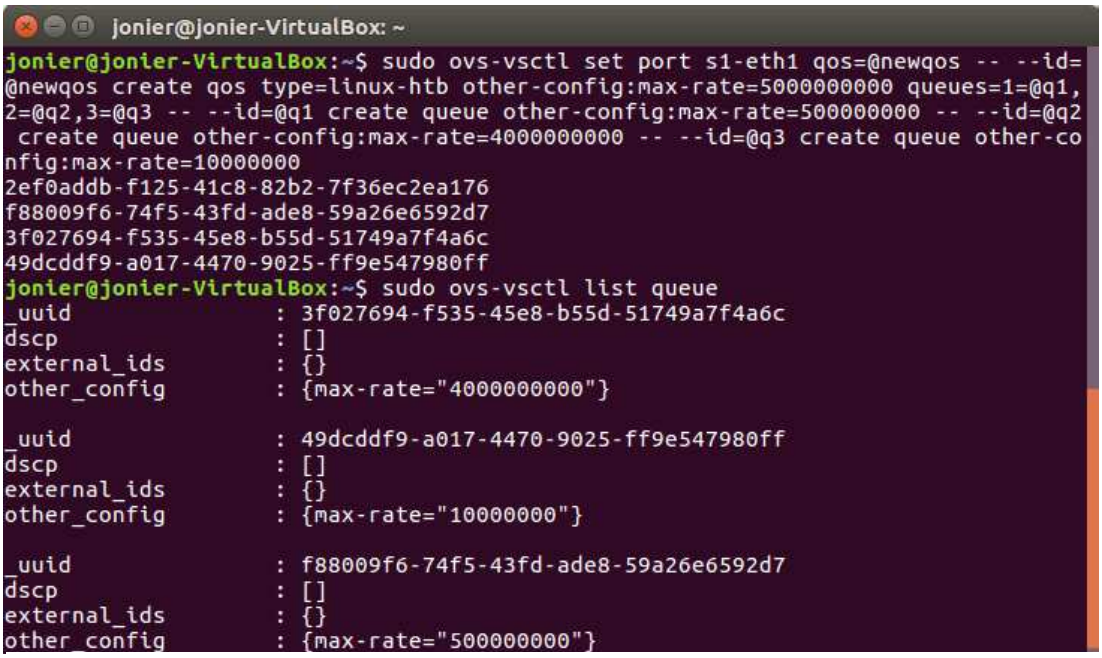


Figura 118. Lista de colas (queues)

Como se mencionó anteriormente, las interfaces a las cuales no se les relacionó una cola, tendrán un ancho máximo por defecto de 5 hasta Gb/s establecida por la regla QoS.

Las pruebas de ancho de banda de las interfaces del switch que no se les relaciona alguna de las colas, se muestra a continuación:

```
mininet> iperf h2 h1
*** Iperf: testing TCP bandwidth between h2 and h1
*** Results: ['4.12 Gbits/sec', '4.13 Gbits/sec']
mininet> iperf h3 h1
*** Iperf: testing TCP bandwidth between h3 and h1
*** Results: ['4.11 Gbits/sec', '4.11 Gbits/sec']
mininet> iperf h6 h1
*** Iperf: testing TCP bandwidth between h6 and h1
*** Results: ['4.13 Gbits/sec', '4.14 Gbits/sec']
mininet> iperf h8 h1
*** Iperf: testing TCP bandwidth between h8 and h1
*** Results: ['4.12 Gbits/sec', '4.12 Gbits/sec']
```

Figura 119. Ancho de banda entre los demás hosts

Finalmente, para eliminar todas las reglas de encolamiento establecidas, se debe primero finalizar la red de Mininet, luego eliminar las reglas QoS y a continuación eliminar las colas.

sudo ovs-vsctl --all destroy queue

Nota: Si surgen problemas a la hora de crear las redes en Mininet debido a problemas con OVS, se debe ingresar la siguiente sentencia para reiniciarlo:

sudo /etc/init.d/openvswitch-switch restart

5.3 CALIDAD DE SERVICIO EN LA TRANSMISIÓN DE UN VIDEO

Esta sección incorpora el uso de Mininet y del controlador Floodlight con el reproductor multimedia VLC para las pruebas de transmisión de material audio visual. VLC emite el video implementando el protocolo UDP, desde un host a otro sobre una red simulada.

5.3.1 Implementando reglas QoS

Utilizando la red anterior (1 switch y 8 Hosts), se pondrá a prueba las reglas de calidad de servicio y encolamiento para la restricción del ancho de banda asignado a los enlaces, que será visualizado mediante contenido multimedia a través de streaming.

Para las pruebas a realizar se define un máximo de ancho de banda para toda la red de 100 Mbits/s mediante la regla QoS. Así mismo se establecen reglas de encolamiento en dirección al host h1 de 5 Mbits/s, 10 Mbits, 15 Mbits/s y 20 Mbits/s. La sentencia que permite realizar este proceso de definición de reglas es la siguiente:

sudo ovs-vsctl set port s1-eth1 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=100000000 queues=1=@q1,2=@q2,3=@q3,4=@q4 -- --id=@q1 create queue other-config:max-rate=5000000 -- --id=@q2 create queue other-config:max-rate=10000000 -- --id=@q3 create queue other-config:max-rate=15000000 -- --id=@q4 create queue other-config:max-rate=20000000

Para asignar las reglas de encolamiento a cada uno de los hosts, se hace uso de la siguiente sentencia donde se define el puerto del switch al cual está conectado el host y el número de la regla: 1 para 5 Mbits/s, 2 para 10 Mbits, 3 para 15 Mbits/s y 4 para 20 Mbits/s:

sudo ovs-ofctl add-flow s1 in_port=2,actions=set_queue:1,normal

5.3.2 Transmisión de video streaming

El envío de la información multimedia se realiza a través del protocolo UDP (User Datagram Protocol) con lo cual no hay confirmación de entrega o recepción de la carga útil del mensaje que permita la retransmisión de la información perdida, ni control de flujo que evite que el receptor se sature al poseer un ancho de banda inferior al óptimo para la transmisión.

Para ocupar todos los posibles caminos hacia el host receptor h1, se realiza ping desde h2, h3, h4, h5, h6 y h8, mientras que h7 envía un video FULL HD con un tamaño de 52.1 MB hacia h1. A este host transmisor se le asocian las diferentes reglas de encolamiento mencionadas anteriormente, estableciendo de esa forma diferente calidad de servicio en cada prueba.

A continuación se visualiza la calidad del video al utilizar las diferentes colas, es decir, relacionando diferentes anchos de bandas:

- *Implementando la primera regla de encolamiento (5 Mb/s)*

La siguiente sentencia permite relacionar el host transmisor h7 (asociado al Puerto 7) con la cola número 1:

sudo ovs-ofctl add-flow s1 in_port=7,actions=set_queue:1,normal



Figura 120. Calidad del video al utilizar un ancho de banda de 5 Mbit/s en el transmisor

- *Implementando la segunda regla de encolamiento (10 Mb/s)*

Se relaciona el host transmisor h7 con la cola número 2:

sudo ovs-ofctl add-flow s1 in_port=7,actions=set_queue:2,normal



Figura 121. Calidad del video al utilizar un ancho de banda de 10 Mbit/s en el transmisor

- *Implementando la tercera regla de encolamiento (15 Mb/s)*

Se relaciona el host transmisor h7 con la cola número 3:

sudo ovs-ofctl add-flow s1 in_port=7,actions=set_queue:3,normal



Figura 122. Calidad del video al utilizar un ancho de banda de 15 Mbit/s en el transmisor

- *Implementando la cuarta regla de encolamiento (20 Mb/s)*

Se relaciona el host transmisor h7 con la cola número 4:

sudo ovs-ofctl add-flow s1 in_port=7,actions=set_queue:4,normal

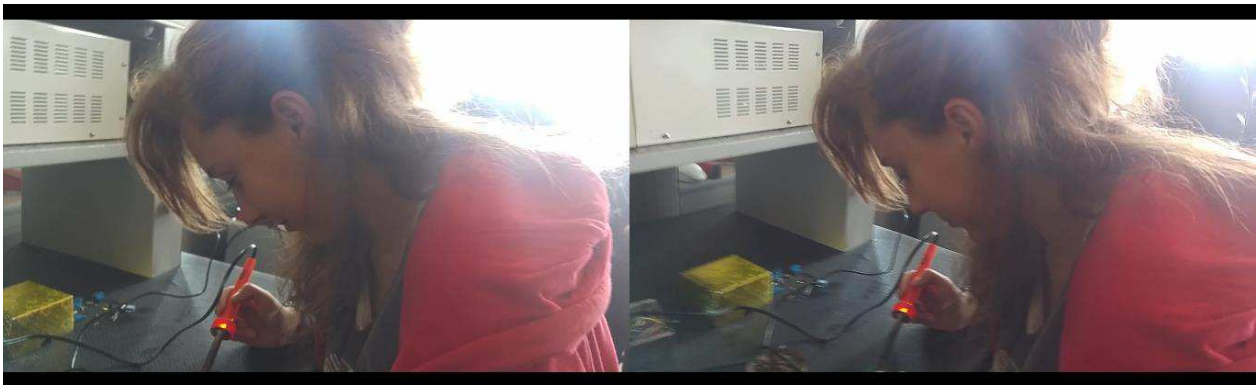


Figura 123. Calidad del video al utilizar un ancho de banda de 20 Mbits/s en el transmisor

5.3.3 Análisis de la información transmitida y recibida

Para poder apreciar las pérdidas que se presentan en la transmisión del video streaming se utiliza el analizador de paquetes Wireshark con un filtro que únicamente permita observar los mensajes UDP. Se mide la tasa de transmisión en B/s y en Paquetes/s en las interfaces del switch donde se encuentra conectado el host emisor (h7) y el host receptor (h1).

La información recibida en B/s cuando el ancho de banda es de 10 Mb/s, 15 Mb/s, 20 Mb/s y 5 Mb/s respectivamente:

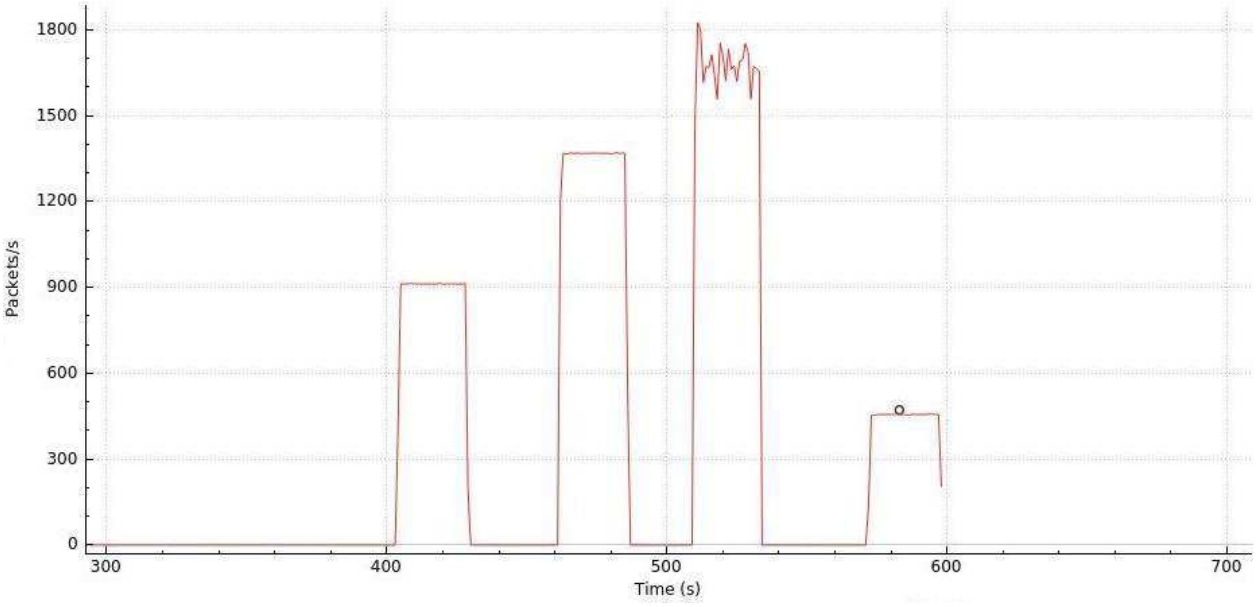


Figura 124. Tasa de transmisión en Paquetes/s en la interfaz s1-eth1

La información enviada en B/s cuando el ancho de banda es de 10 Mb/s, 15 Mb/s, 20 Mb/s y 5 Mb/s respectivamente:

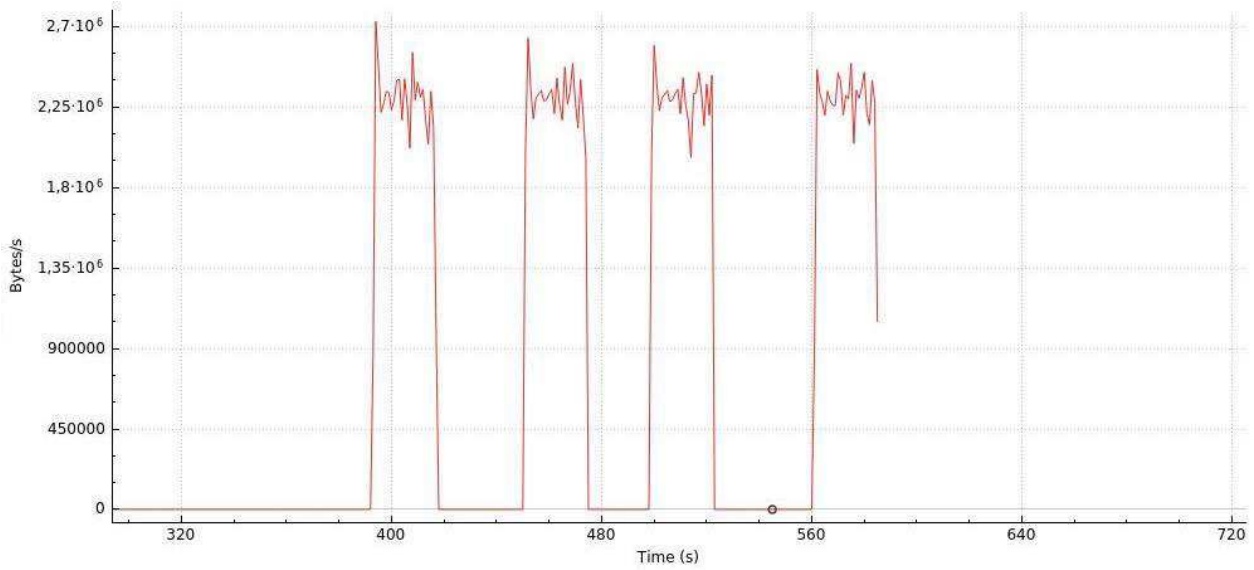


Figura 125. Tasa de transmisión en B/s en la interfaz s1-eth7

La información enviada en Paquetes/s cuando el ancho de banda es de 10 Mb/s, 15 Mb/s, 20 Mb/s y 5 Mb/s respectivamente:

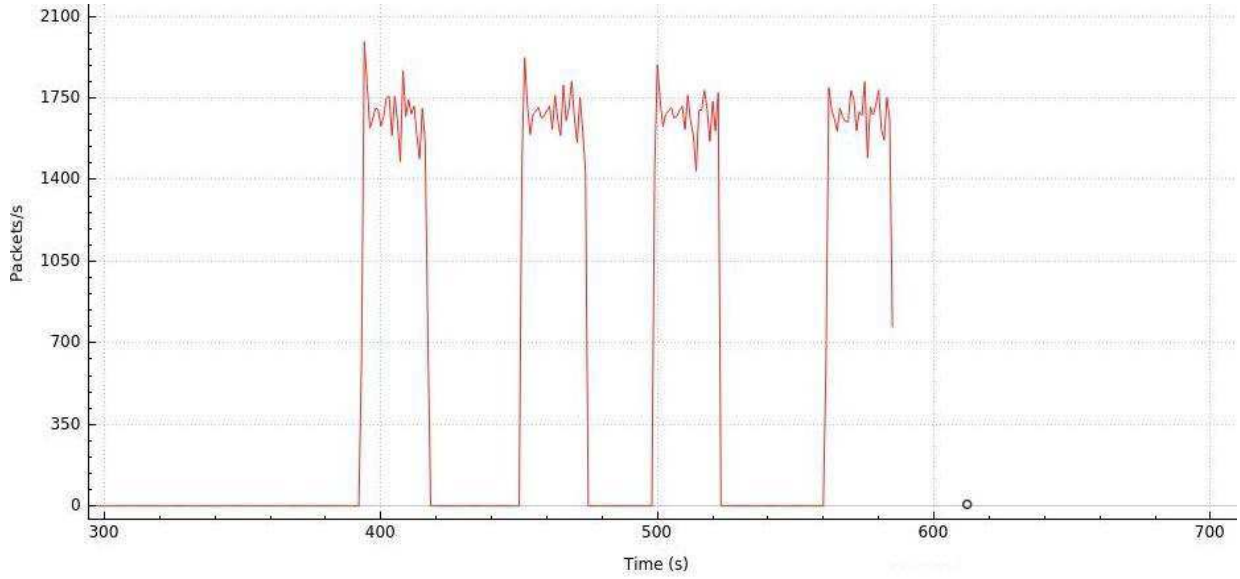


Figura 126. Tasa de transmisión en paquetes/s en la interfaz s1-eth7

La información recibida en Paquetes/s cuando el ancho de banda es de 10, 15, 20 y 5 Mb/s respectivamente:

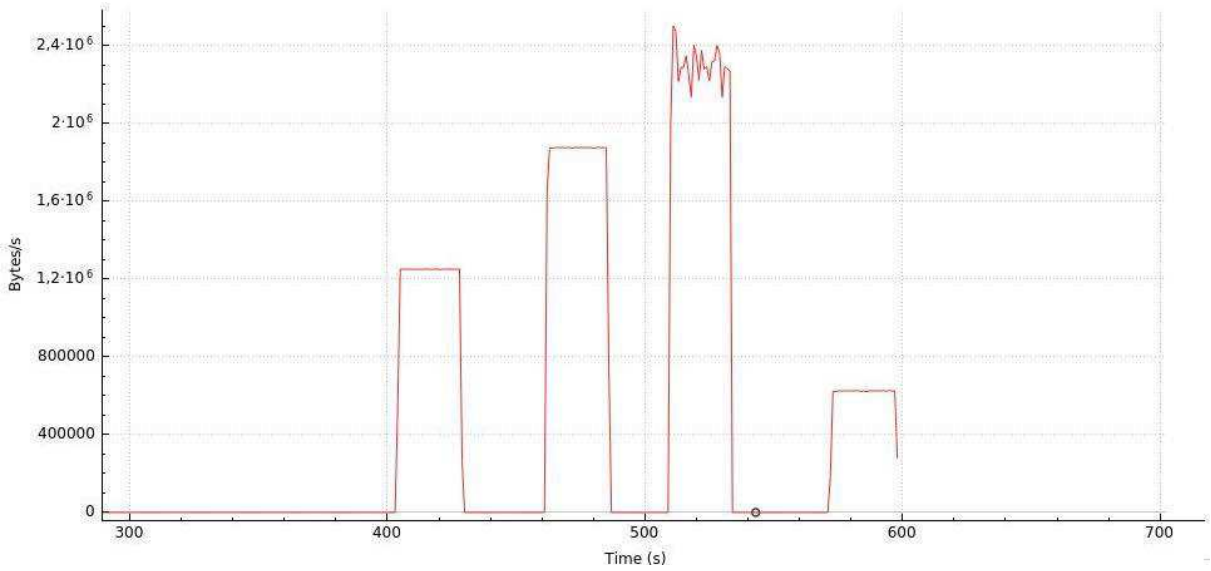


Figura 127. Tasa de transmisión en paquetes/s en la interfaz s1-eth1

La tasa de transmisión con la que se envía el video streaming es de aproximadamente 2.25 MB/s (no confundir con Mb/s) o 1680 Paquetes/s.

La tasa de transmisión en el host receptor se encuentra alrededor de 0.65 MB/s para un ancho de banda de 5 Mb/s, para 10 Mb/s la tasa transmisión es de aproximadamente 1.25 MB/s, para 15 Mb/s la tasa transmisión es de aproximadamente 1.9 MB/s, finalmente para un ancho de banda de 20 Mb/s la tasa de transmisión es de aproximadamente 2.35 MB/s. Estos resultados tienen mucho sentido al convertir Mb en MB.

Los resultados obtenidos en la interfaz que se comunica con el host h7 con respecto a los obtenidos en el host h1, muestran una reducción significativa en la información recibida al variar el ancho de banda.

- Para un ancho de banda de 5 Mbit/s la reducción fue de un 71.11 %, es decir se recibió un 28.89 % de la información.

- Para un ancho de banda de 10 Mbit/s la reducción fue de un 44.44 %, es decir se recibió un 55.56 % de la información.

- Para un ancho de banda de 15 Mbit/s la reducción fue de un 15.56 %, es decir se recibió un 84.44 % de la información.

- Para un ancho de banda de 20 Mbit/s no hay pérdidas en teoría, debido a que el canal de 2.35 MB/s es superior al requerido de 2.25 MB/s

El tamaño del video es de 52.1 MB que equivale a 416.8 Mb, con una duración de 24 segundos. Si el ancho de banda es de 5 Mb/s solo podrá transmitir 120 Mb en dicho tiempo, para 10 Mb/s de ancho de banda solo podrá transmitir 240 Mb, para 15 de ancho de banda solo podrá trasmitir 360 Mb y para 20 Mb/s de ancho de banda podrá transmitir 480 Mb. Esto porcentualmente equivale a 28,77 %, 57,5 %, 86,33 % y 100 % de la información transmitida respectivamente. Los valores porcentuales se acercan a los resultados presentados por Wireshark.

CAPÍTULO 6. COMPARACIÓN DE UNA RED TRADICIONAL Y UNA RED SDN MEDIANTE SIMULACIÓN

Se implementa una red personalizada compuesta por diferentes dispositivos de red, y varios caminos, esto con el objetivo de generar una carga importante dentro del comportamiento de la red. En total está compuesta por 14 hosts y 17 switches.

Nota: El código Python de esta red personalizada se encuentra en el **anexo 4**.

6.1 COMPORTAMIENTO DE UNA RED EN PACKET TRACER Y MININET

El protocolo de red para enrutamiento elegido para esta prueba es el OSPF, ya que cuenta con el algoritmo de Dijkstra enlace-estado para calcular la ruta idónea entre dos nodos cualesquiera dentro de una red. De igual manera, los módulos de load balancer implementados anteriormente, cuentan en su mayoría con la lógica de este algoritmo para encontrar el camino más corto.

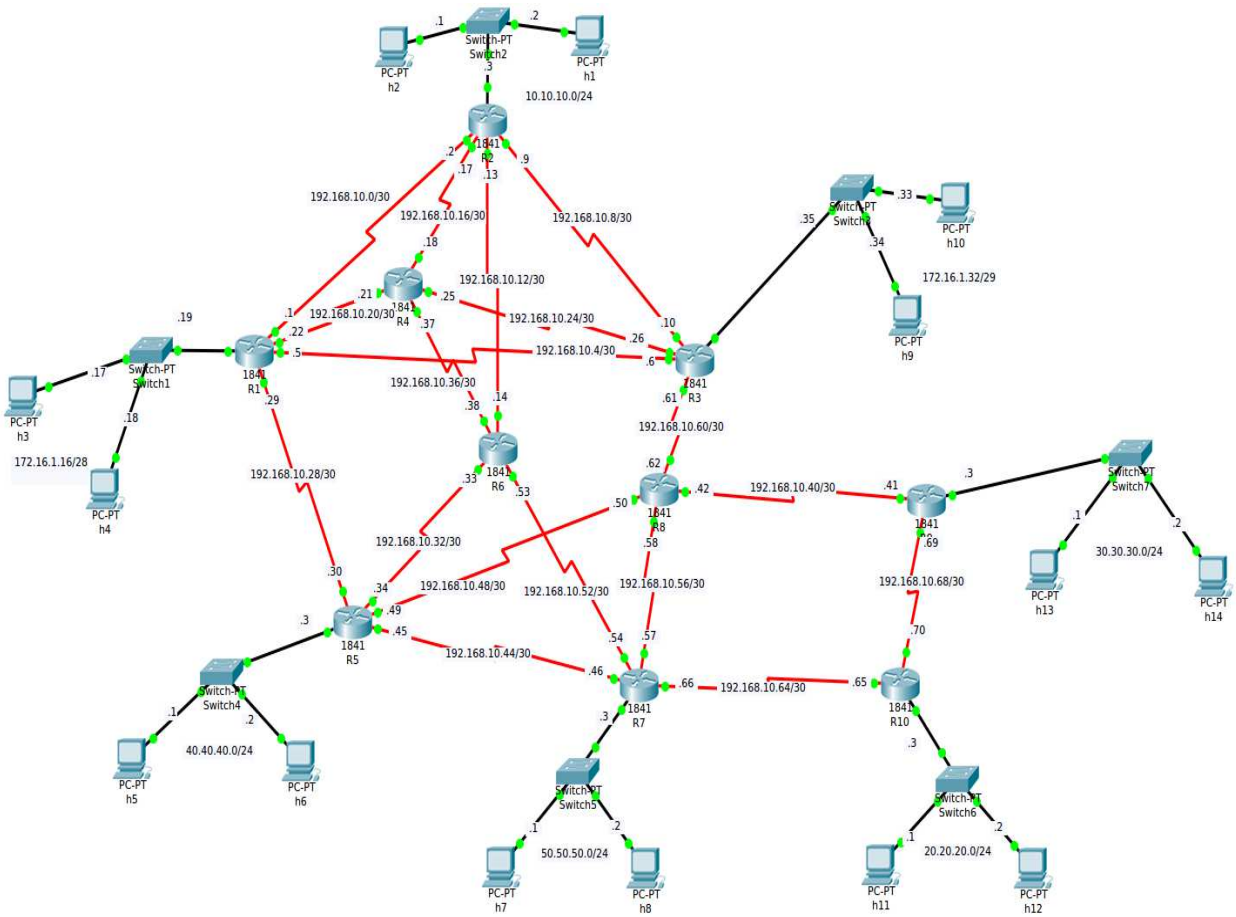


Figura 128. Red personalizada en Packet Tracer

En la configuración de la red de Packet Tracer se define el clock rate de los routers en 2 Mbit/s. Por esta razón, es necesario adecuar el ancho de banda de la red en Mininet para que cumpla con este mismo parámetro de velocidad, implementado la siguiente instrucción:

> *sudo mn --custom redpropia2.py --topo mytopo --controller=remote,ip=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13 --link tc,bw=2*

Donde se hace el llamado a la red personalizada “redpropia2.py”, con una configuración de ancho de banda de 2 Mbits/s para todos los enlaces mediante el comando “link tc, bw” en Mininet.

Para la construcción de la red en Mininet los routers son reemplazados por switches OpenFlow, puesto que en las redes definidas por software el enrutamiento lo realiza el controlador, es decir que no es necesario la implementación de routers. Para las pruebas en esta sección se utiliza el controlador Floodlight.

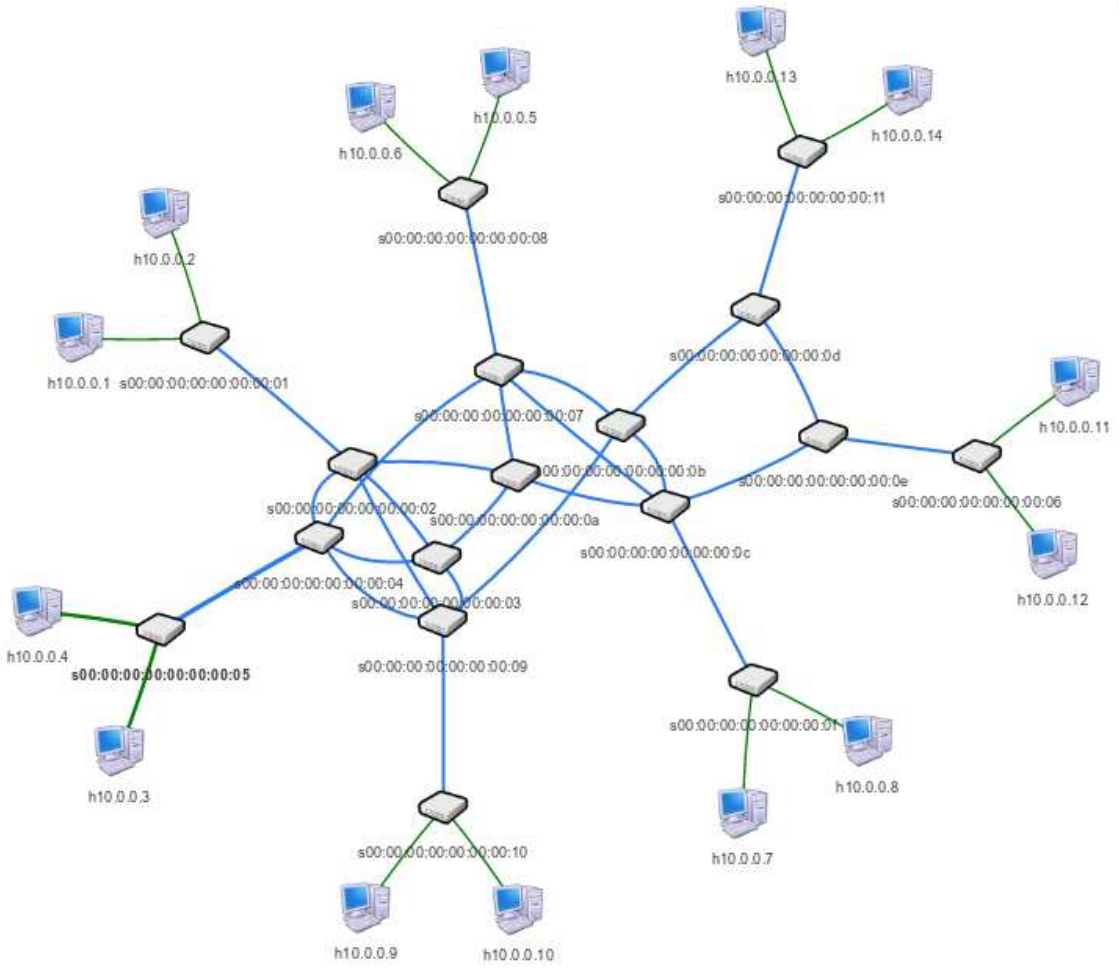


Figura 129. Red personalizada utilizando Floodlight

6.1.1 Medición de ancho de banda de la red

- *Ancho de banda entre h4 y h14 en Mininet*

```
mininet> iperf h4 h14
*** Iperf: testing TCP bandwidth between h4 and h14
*** Results: ['1.92 Mbits/sec', '2.20 Mbits/sec']
```

Figura 130. Ancho de banda entre h4 y h14

- *Ancho de banda entre h1 y h11 en Mininet*

```
mininet> iperf h1 h11
*** Iperf: testing TCP bandwidth between h1 and h11
*** Results: ['1.91 Mbits/sec', '2.11 Mbits/sec']
```

Figura 131. Ancho de banda entre h1 y h11

En Packet Tracer mediante la instrucción show running-config, se puede observar que las interfaces del router seleccionado tiene un clock rate de 2 Mbits/s

```
interface Serial0/0/0
 ip address 192.168.10.22 255.255.255.252
!
interface Serial0/0/1
 ip address 192.168.10.29 255.255.255.252
 clock rate 2000000
!
interface Serial0/1/0
 ip address 192.168.10.1 255.255.255.252
 clock rate 2000000
!
interface Serial0/1/1
 ip address 192.168.10.5 255.255.255.252
 clock rate 2000000
```

Figura 132. Clock rate para routers en Packet Tracer

6.1.2 Solicitudes de ping

- *Solicitudes de ping entre h4 y h14*

La primera prueba se realiza entre h4 con IP 10.0.0.4/8 y h14 con IP 10.0.0.14/8 en Mininet, que corresponde a la red en Packet Tracer a los hosts h4 con IP 172.16.1.18/28 y h14 con IP 30.30.30.2/24 respectivamente.

La prueba a realizar consta de la transmisión de 100 solicitudes ping desde h4 a h14:

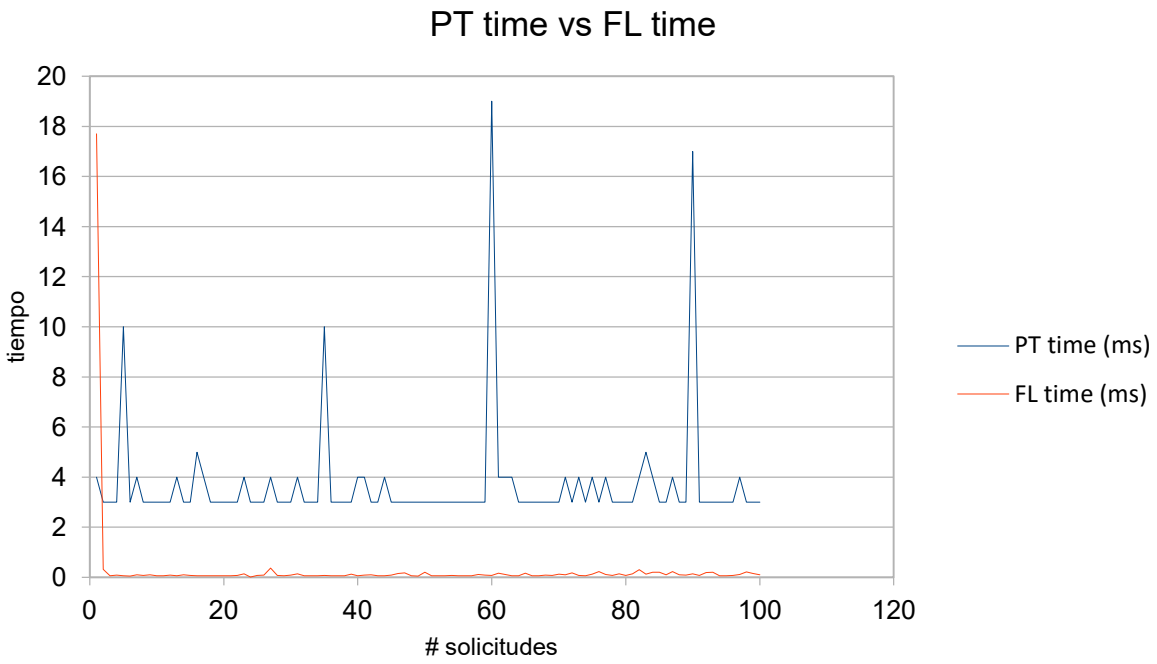


Figura 133. Tiempo que toman las solicitudes de h4 a h14 en Packet Tracer y Mininet

La gráfica azul corresponde a los tiempos obtenidos de las solicitudes ping realizadas en Packet Tracer y la roja a los de Mininet.

Nota: Para el caso de la simulación en Packet Tracer, la primera solicitud no es exitosa y aparece el mensaje de error "Request timed out" debido a que el tiempo de espera es agotado, por lo cual no se incluye en la gráfica.

Las estadísticas de tiempo de las solicitudes ping en ambos simuladores son las siguientes:

	Mininet	Packet Tracer
Tiempo mínimo	0,057 ms	3 ms
Tiempo máximo	17.784 ms	17 ms
Tiempo promedio	0,285 ms	4 ms

Tabla 14. Estadísticas de tiempo de las solicitudes ping entre h4 y h14 en ambos simuladores

Como se puede observar en la tabla anterior, el tiempo promedio de la red en Packet Tracer es de aproximadamente 14 veces el valor de tiempo de la red en Mininet. El tiempo minino es mucho más bajo en Mininet que en Packet Tracer, y el tiempo máximo es similar para ambos simuladores, esto se debe a que el primer ping que se realiza en Floodlight se hace reconocimiento de la red.

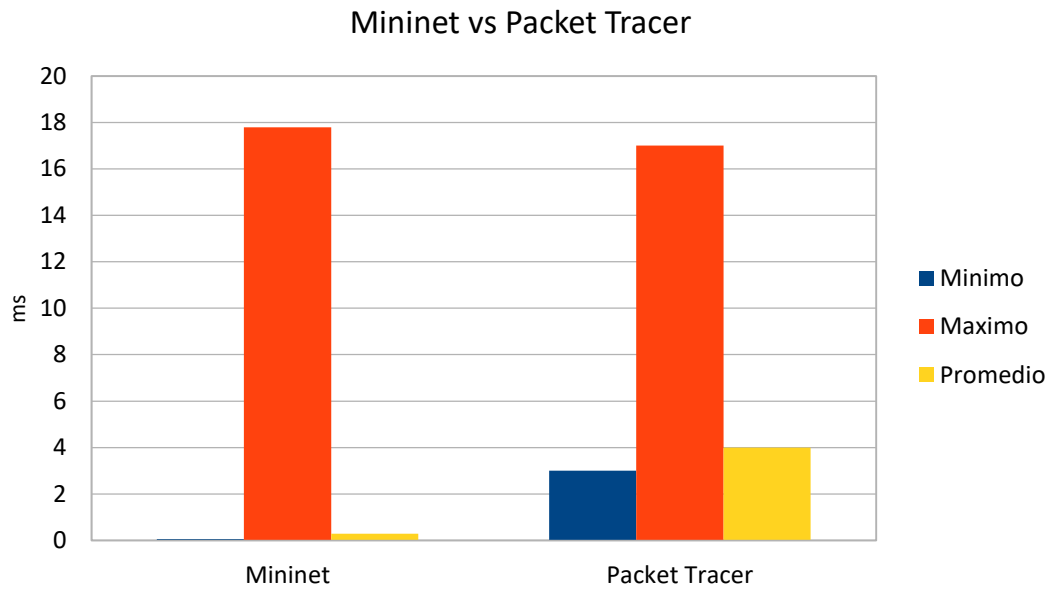


Figura 134. Estadísticas de tiempo de las solicitudes ping entre h4 y h14 en Mininet y Packet Tracer

- *Solicitudes de ping entre h1 y h11*

La siguiente prueba se realiza entre h1 con IP 10.0.0.1/8 y h11 con IP 10.0.0.11/8 en Mininet, que corresponde en Packet Tracer al host h1 con IP 10.10.10.2/24 y al host h11 con IP 20.20.20.1/24 respectivamente. Se envían 100 solicitudes ping desde h1 a h11.

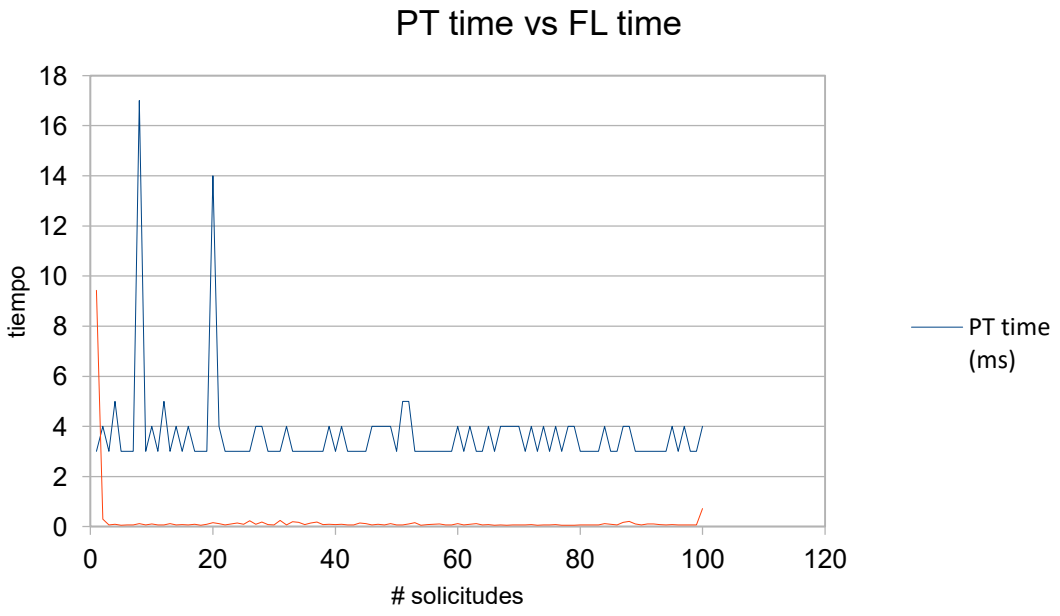


Figura 135. Tiempo que toman las solicitudes de h1 a h11 en Packet Tracer y Mininet

La gráfica azul corresponde a los resultados obtenidos en Packet Tracer y la roja a los de Mininet.

Nota: Para el caso de la simulación en Packet Tracer la primera solicitud no es exitosa y aparece un mensaje de respuesta “Request timed out” debido a que el tiempo de espera se agota y por lo tanto no se incluye.

Las estadísticas de tiempo de las solicitudes de ping entre h1 y h11 en ambos simuladores, son las siguientes:

	Mininet	Packet Tracer
Tiempo mínimo	0,06 ms	3 ms
Tiempo máximo	9.433 ms	17 ms
Tiempo promedio	0,182 ms	4 ms

Tabla 9. Estadísticas de tiempo de las solicitudes ping entre h1 y h11 en Mininet y Packet Tracer

En este caso también se puede observar en la tabla como el tiempo promedio en Packet Tracer es aproximadamente 22 veces mayor que el tiempo promedio en Mininet. Por otro lado, el tiempo mínimo es mucho más bajo en la red de Mininet que en Packet Tracer, y el tiempo máximo de ambos son equiparables, esto se debe a que el controlador en su primer ping hace reconocimiento de la red en busca de mejores rutas para los paquetes, lo que permite mejorar el comportamiento de los demás paquetes.

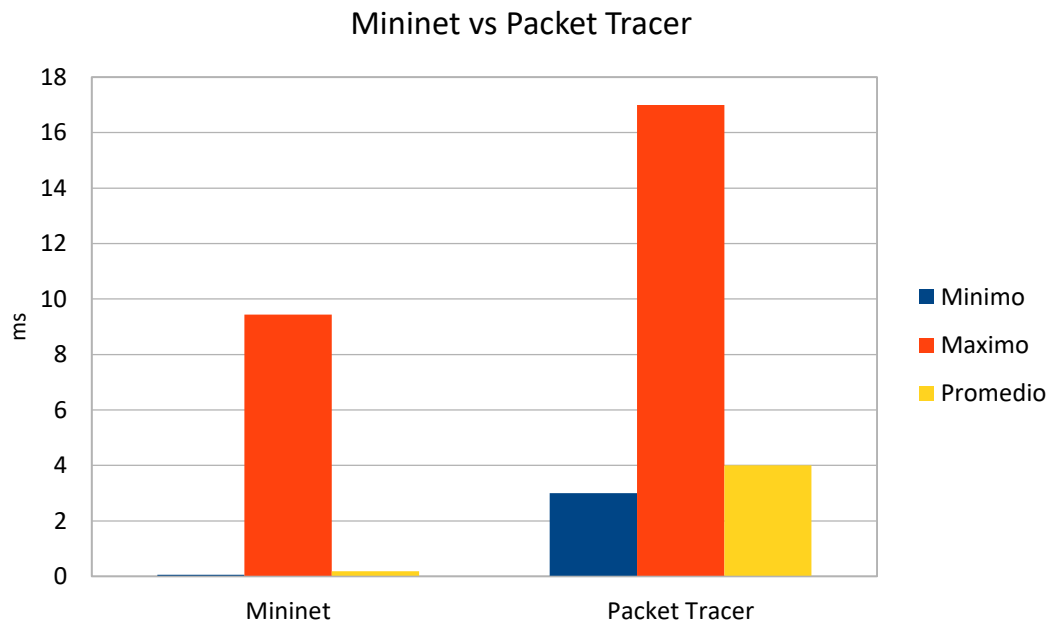


Figura 136. Estadísticas de tiempo de las solicitudes ping entre h1 y h11 en ambos simuladores

6.2 COMPORTAMIENTO DE UNA RED COMPUESTA POR SOLO SWITCHES EN PACKET TRACER Y MININET

Se construye una red compuesta por solo switches en Packet Tracer. Con esto se espera observar el comportamiento de una red tradicional sin enrutadores y verificar que un controlador SDN realiza el enrutamiento perfectamente sin necesidad de routers.

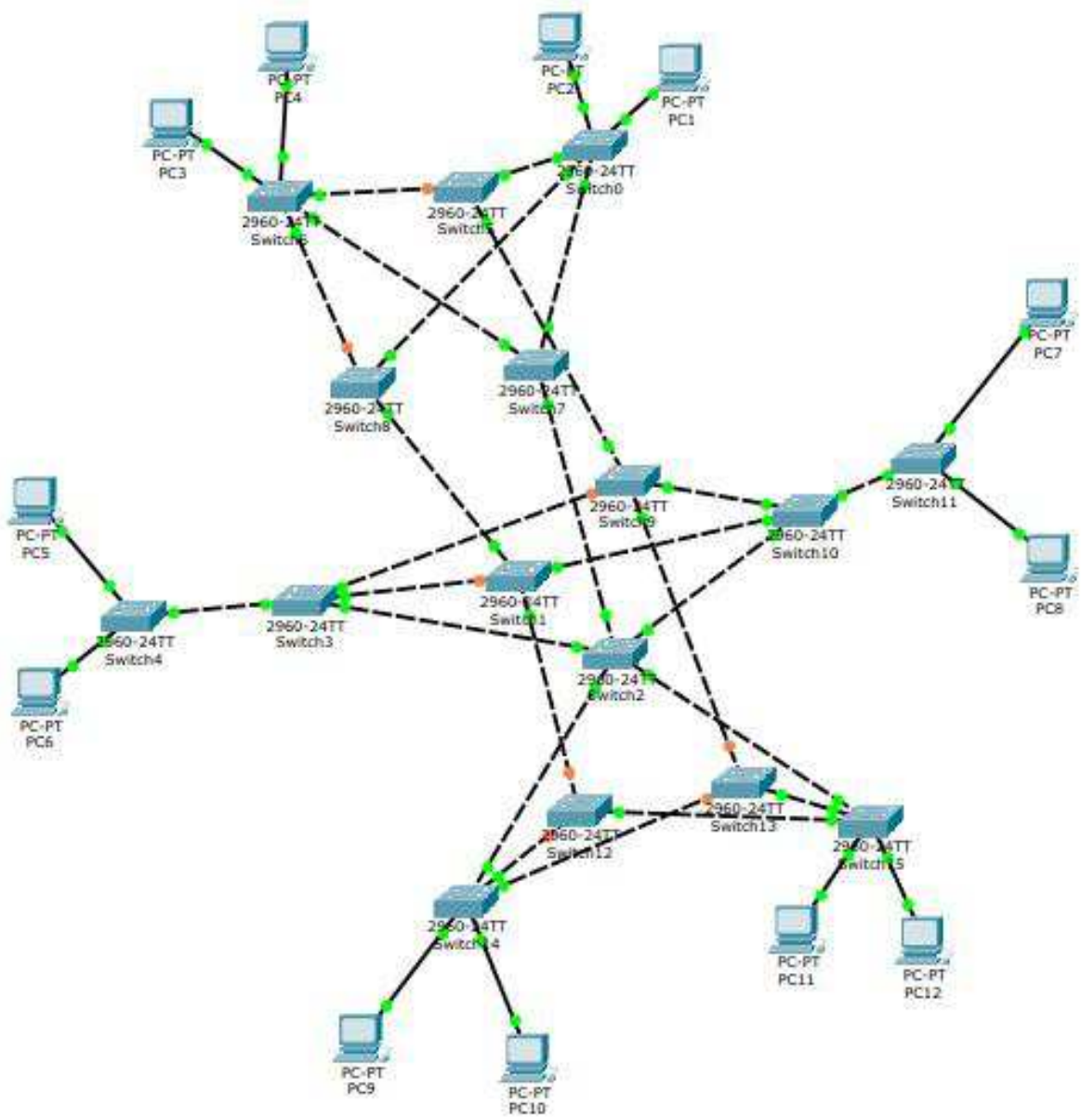


Figura 137. Red compuesta por solo switches en Packet Tracer

Se envían 4 solicitudes de ping de PC7 a PC8, y de PC7 a PC1, y se obtienen los siguientes resultados:

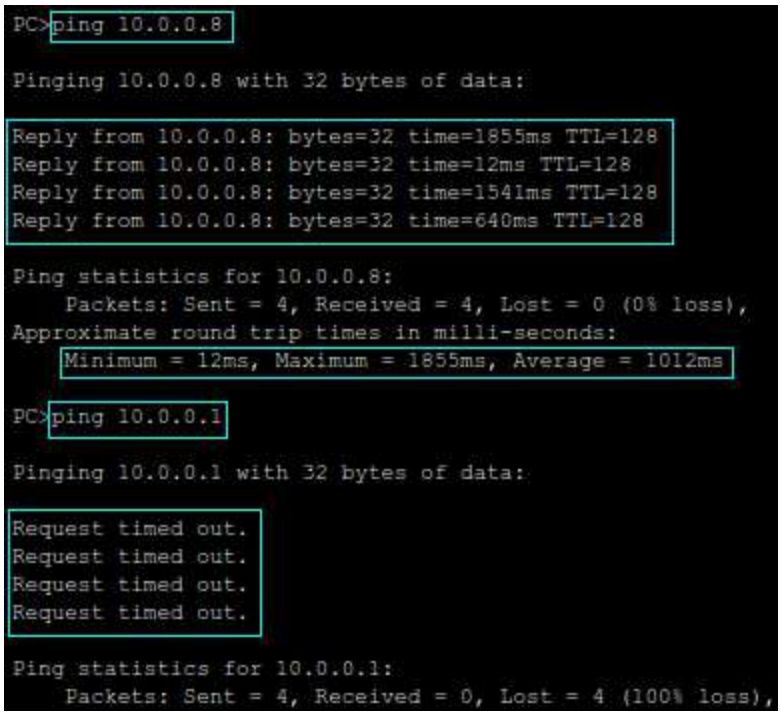


Figura 138. Tiempo de las solicitudes ping de PC7 a PC8 y PC7 a PC1

En la imagen anterior, se puede observar que los tiempos de cada solicitud ICMP son muy altos comparados con los tiempos de esta misma red bajo la presencia de un controlador SDN. Además hay que considerar que PC7 y PC8 se encuentran conectados al mismo switch y aun así el tiempo es bastante alto. De la misma manera, se envían solicitudes ping entre PC7 y PC1, con la diferencia de que estas no logran llegar a su destino debido a que exceden el tiempo de espera.

La respuesta obtenida de esta misma red simulada en Mininet y utilizando el controlador Floodlight, arroja los siguientes resultados:

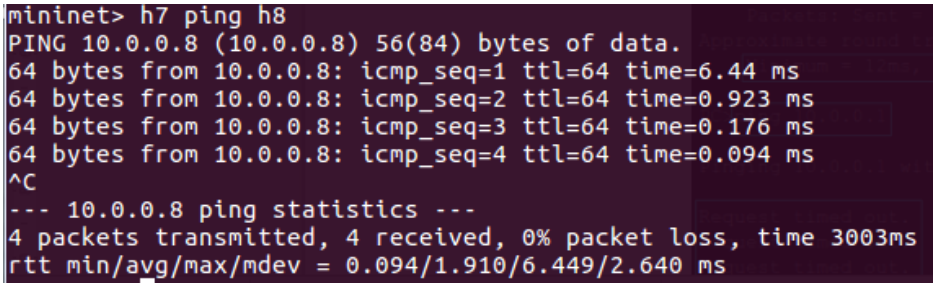


Figura 139. Tiempo de solicitudes ping entre h7 y h8

A continuación se tabula y grafica los tiempos de las solicitudes de ping entre h7 y h8:

Solicitudes de ping	Packet Tracer	Mininet
Primer ping	1855 ms	6,44 ms
Segundo ping	12 ms	0,923 ms
Tercer ping	1541 ms	0,176 ms
Cuarto ping	640 ms	0,094 ms

Tabla 10. Comparación de tiempos que toman las solicitudes ping en Packet Tracer y Mininet

Como se puede observar en la tabla, el tiempo de las solicitudes ping en Packet Tracer puede llegar a ser aproximadamente 288 veces el valor de tiempo en Mininet. Incluso, puede ocurrir que no se establezca conexión entre dos hosts en esta red de switches en Packet Tracer. Estos Resultados demuestran lo eficiente que es un controlador SDN enrutando los paquetes en la red.

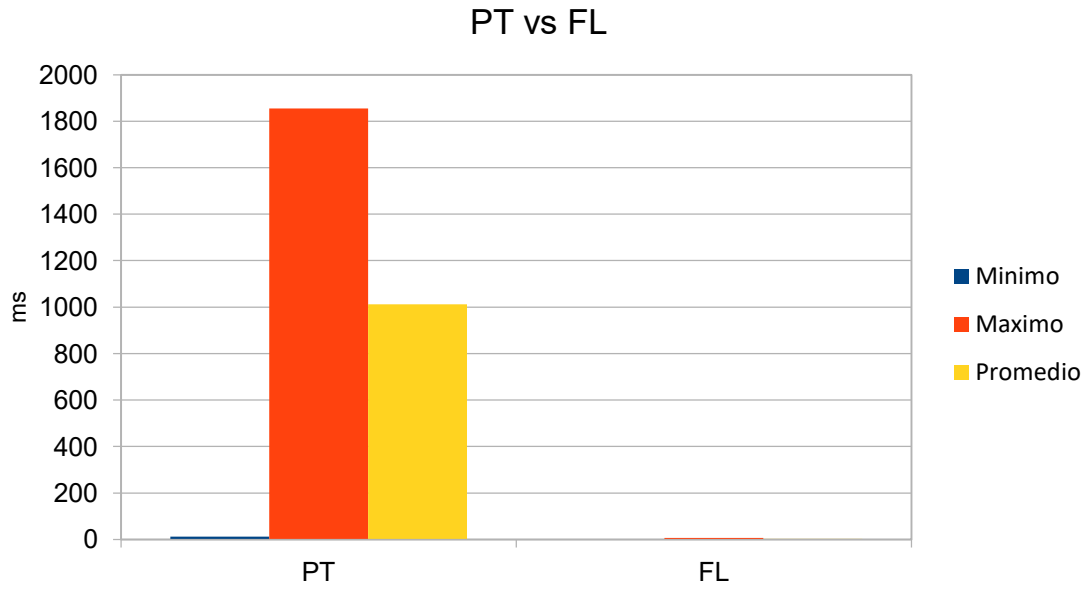


Figura 140. Estadísticas de tiempo de las solicitudes ping en Packet Tracer y Mininet

La gráfica refleja el resultado de los tiempos para las cuatro solicitudes ping realizadas entre h7 y h8 (PC7 y PC8). Para el caso de las solicitudes ping entre h7 y h1 no se realiza una comparativa gráfica debido a que no fue posible establecer la conexión entre estos hosts en Packet Tracer.

Como era de esperarse, la comunicación entre dos puntos de la red es pésima debido a que no hay un proceso de enrutamiento que permita encontrar un camino. De esta manera se puede concluir que las redes definidas por software no requieren de la implementación de routers para el enrutamiento, pues el controlador lo hace perfectamente, incluso mucho mejor. Por otro lado, las redes tradicionales necesitan implementar estos dispositivos de red (routers) para el enrutamiento, de lo contrario la comunicación será pésima.

CONCLUSIONES

Las redes definidas por software o redes programables, cuentan con una gran gama de controladores para el manejo de tráfico en la red, donde cada uno de ellos difiere entre sí por su complejidad y soporte dado por sus desarrolladores. Por ejemplo, el controlador POX cuenta con un buen rendimiento en el manejo de tráfico asociado a retardos y anchos de banda, pero presenta problemas como el elevado tiempo de respuesta en la primera solicitud ping en la comunicación entre dos hosts, posee poca documentación frente a los otros controladores, y la falta de una REST API. Por otro lado, el controlador OpenDaylight tiene un rendimiento más bajo a la hora de manejar tiempos de respuesta y asignación de anchos de banda, y tiene un excesivo consumo de memoria RAM, sin embargo cuenta con buen soporte por parte de sus desarrolladores y posee una Web API. Finalmente, el controlador Floodlight cuenta con un muy buen rendimiento en el manejo de tiempos de respuesta y ancho de banda al poseer un gran número de módulos que realizan acciones específicas que conjuntamente mejoran su eficiencia. Además, cuenta con buena documentación y facilidad de manejo por medio de instrucciones sencillas en la terminal o a través de su Web API.

Una diferencia abismal entre la redes tradicionales donde el control de la red es descentralizado y llevado a cabo por los routers como dispositivos de encaminamiento, y las redes definidas por software donde el control es centralizado y ejecutado por un controlador programable, es la posibilidad de generar enrutamiento en redes compuestas por un gran número de switches sin la implementación de routers. Por otro lado, se pudo observar en el desarrollo comparativo entre redes tradicionales y redes SDN, que los tiempos de las solicitudes ping eran más estables y más bajos en las SDN, así como sus anchos de banda eran superiores, o al menos a nivel de simulación.

La implementación de balance de carga permitió mejorar el comportamiento de una red definida por software en términos de calidad de servicio, mejorando el ancho de banda, disminuyendo tiempos de retardo y distribuyendo de forma óptima la carga de los enlaces. Mediante esta implementación se logró establecer calidad en el servicio, que para el caso del controlador OpenDaylight hubo mejoras de alrededor de 100 veces más en el ancho de banda al encontrar un camino más óptimo para los paquetes. Por otro lado, los tiempos de retardo en la transmisión de paquetes se obtuvo una mejora al reducir 5 veces en promedio estos tiempos. En general, los resultados en este ítem fueron bastante satisfactorio.

Mediante el establecimiento de calidad de servicio entre dos puntos de la red, sea a través de priorización de interfaces o de tráfico, se pudo observar cómo afecta la asignación de diferentes prioridades a un enlace a la hora de transmitir información. Para el caso de la transmisión de contenido multimedia mediante streaming, se observó claramente que al asignar prioridades distintas a un puerto del switch por el cual se transmitía un video, la calidad de la imagen cambiaba dependiendo de la prioridad asignada, y que se traducía en variaciones del ancho de banda.

Finalmente, se espera en trabajos futuros realizar pruebas de implementación física, mediante el uso de switches OpenFlow, que permita estudiar las prestaciones de una red SDN frente a una red tradicional, de tal manera que se pueda realizar un análisis de las ventajas y desventajas de implementar este nuevo paradigma de gestionar las redes. Con un estudio de este tipo se podrá llegar a la conclusión de cuándo es viable utilizar una red definida por software o una red tradicional.

REFERENCIAS

[1.] Feamster, N., Rexford, J., & Zegura, E. (s.f.) (2015). The Road to SDN: An Intellectual History of Programmable Networks. Obtenido de https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf

[2.] Planas, A.L. (Enero 2016) CONFIGURACIÓN DE UN ENTORNO DE EMULACIÓN QUE PERMITA EL DISEÑO, DESARROLLO Y EVALUACIÓN DE SOFTWARE- DEFINED NETWORKS CON CALIDAD DE SERVICIO. Available at: https://upcommons.upc.edu/bitstream/handle/2117/81460/104342.pdf?sequence=1&isAllowed=y (Accessed: 5 September 2016).

[3.] DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA CARRERA DE INGENIERÍA ELECTRÓNICA, REDES DE LA INFORMACIÓN Y COMUNICACIÓN DE DATOS TESIS PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN (2015) Available at: http://repositorio.espe.edu.ec/bitstream/21000/9848/1/T-ESPE-048694.pdf (Accessed: 5 September 2016).

[4.] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Turner, J. (14 de Marzo de 2008). OpenFlow: Enabling Innovation in Campus Networks. Obtenido de http://archive.openflow.org/documents/openflow-wp-latest.pdf

[5.] «IEEE Xplore Abstract - Performance Analysis of Software-Defined Networking (SDN)». [En línea]. Disponible en: http://ieeexplore.ieee.org/xpl/login.jsp?reload=true&tp=&arnumber=6730793&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6730793. [Accedido: 25-oct-2015].

[6.] «SDN marca el futuro del networking», *NetworkWorld*. [En línea]. Disponible en: http://www.networkworld.es/sdn/sdn-marca-el-futuro-del-networking. [Accedido: 25-oct-2015].

[7.] «Redes complejas hechas fácilmente, ¿será posible? | Magazcitum». [En línea]. Disponible en: http://www.magazcitum.com.mx/?p=2353#.Vi0SU_nMh3J. [Accedido: 25-oct-2015].

[8.] Open Networking Foundation, «ONF Overview - Open Networking Foundation,» [En línea]. Available: https://www.opennetworking.org/about/onf-overview. [Último acceso: 13 octubre 2015].

[9.] «UNIVERSIDAD TÉCNICA DE AMBATO FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL». [En línea]. Disponible en: http:// repo.uta.edu.ec/bitstream/123456789/10587/1/Tesis_982ec.pdf. [Accedido: 25-oct-2015].

[10.] « ¿Pueden las redes definidas por software (SDN) mejorar la rentabilidad del operador? ». [En línea]. Disponible en: https://www.cisco.com/web/ES/assets/pdf/networking_sdn_enhance_operator_monetization_wp.pdf. [Accedido: 25-oct-2015].

[11.] «¿Qué es Redes definidas por software (SDN)? - Definición en WhatIs.com». [En línea]. Disponible en: http://searchdatacenter.techtarget.com/es/definicion/Redes-definidas-por-software-SDN. [Accedido: 25-oct-2015]

[12.] «Diálogo TI». [En línea]. Disponible en: http://dialogoti.intel.com/es/documento/redes-definidas-por-software-sdn-y-funciones-de-red-virtualizadas-nfv. [Accedido: 25-oct-2015].

[13.] «SDN – Redes definidas por Software ». [En línea]. Disponible en: https://articulosit.files.wordpress.com/2013/10/sdn.pdf. [Accedido: 25-oct-2015].

[14.] Antonio, M., Monge, S., Javier, L., Villalba, G., Lucila, A., Orozco, S. and Trabajo De Fin De Máster En (2015) 'ARQUITECTURA PARA LA MEJORA DE LA CALIDAD DE SERVICIOS MULTIMEDIA EN REDES DEFINIDAS POR SOFTWARE'.

[15.] María Del Carmen Ternero, D. (2010) *Calidad de servicio (QoS) en redes*. Available at: http://www.dte.us.es/personal/mcromero/masredes/docs/SMARD.0910.qos.pdf (Accessed: 26 October 2016).

[16.] *Calidad de servicio* (2007) in *Wikipedia*. Available at: https://es.wikipedia.org/wiki/Calidad_de_servicio (Accessed: 26 October 2016).

[17.] *Calidad de servicio - EcuRed*. (2017). *Ecured.cu*. Retrieved 27 May 2017, from https://www.ecured.cu/Calidad_de_servicio

[18.] Castañeda, L., Sarmiento, D., & Parra, O. (2017). *BALANCEO DE CARGA EN REDES DE CONMUTACIÓN DE ETIQUETAS CON RSLB Y TLSB*. *Revistas.udistrital.edu.co*. Retrieved 27 May 2017, from http://revistas.udistrital.edu.co/ojs/index.php/REDES/article/view/6378/7895

[19.] *Citar un sitio web - Cite This For Me*. (2017). *Upcommons.upc.edu*. Retrieved 27 May 2017, from http://upcommons.upc.edu/bitstream/handle/2099.1/3762/54602-1.pdf?sequence=1&isAllowed=y http://www.cisco.com/cisco/web/support/LA/7/73/73030_46.pdf

[20.] Castro Jiménez, K., & Ferrer Gómez, r. (2004). *Load Balancing: Balanceo de carga. concepto, estado del arte y aplicabilidad en linux y windows*. Universidad Tecnológica de Bolívar Facultad de ingenierías programa de Ingeniería de Sistemas. http://biblioteca.unitecnologica.edu.co/notas/tesis/0026223.pdf

[21.] Cabrera Sanmartín, L. (2012). *Calidad de servicio IPV6*. Ingeniería Electrónica en la facultad de ingenierías de la Universidad Tecnológica de Bolívar. http://biblioteca.unitecnologica.edu.co/notas/tesis/0063146.pdf

[22.] *Home - Project Floodlight - OpenFlow news and projects*. (2017). *Project Floodlight*. Retrieved 27 May 2017, from http://www.projectfloodlight.org

[23.] *The OpenDaylight Platform | OpenDaylight*. (2017). *Opendaylight.org*. Retrieved 27 May 2017, from http://www.opendaylight.org

[24.] *What is POX? - Definition from WhatIs.com*. (2017). *SearchSDN*. Retrieved 27 May 2017, from http://searchsdn.techtarget.com/definition/POX

[25.] Cano Moreno, J. (2015). *Estudio de las redes definidas por software y escenarios virtuales de red orientados al aprendizaje*. Universidad Politécnica de Madrid Escuela Técnica Superior de Ingenieros de Telecomunicación. http://www.dit.upm.es/~posgrado/doc/TFM/TFMs2014-2015/TFM_Javier_Cano_Moreno_2015.pdf

ANEXOS

Los códigos que aparecen a continuación están realizados en Python, por lo que deben ser guardados con la extensión .py.
Los códigos de Load Balancer no fueron realizados por nosotros, sino por usuarios de **GITHUB**, sin embargo si hicimos algunas modificaciones en los códigos para poder utilizarlos.

Si se pretende utilizar los códigos que aparecen en este documento, lo más seguro es que existan problemas a la hora de copiar y pegar el código debido a que al obtener el texto desde archivos PDF, no se realiza con la estructura original del documento, en este caso de un Word (no se conservan los espacios). Para dar solución a ello se sube un archivo en la nube donde se encuentran los códigos en .doc. El enlace para descargar el documento es el siguiente (abrirlo con Word):

https://drive.google.com/file/d/0B7ttaaQn1GEYcUJZOUJ2R2VkLTg/view

ANEXO 1: Código de la red personalizada 1

```
#!/usr/bin/python

from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch
from mininet.topo import Topo

class fatTreeTopo(Topo):
   "Fat Tree Topology"
   def __init__(self):
      "Create Fat tree Topology"
      Topo.__init__(self)

      #Add hosts

         h3 = self.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
         h2 = self.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
         h5 = self.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)
         h6 = self.addHost('h6', cls=Host, ip='10.0.0.6', defaultRoute=None)
         h1 = self.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
         h4 = self.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
         h7 = self.addHost('h7', cls=Host, ip='10.0.0.7', defaultRoute=None)
         h8 = self.addHost('h8', cls=Host, ip='10.0.0.8', defaultRoute=None)
         h9 = self.addHost('h9', cls=Host, ip='10.0.0.9', defaultRoute=None)
         h10 = self.addHost('h10', cls=Host, ip='10.0.0.10', defaultRoute=None)
         h11 = self.addHost('h11', cls=Host, ip='10.0.0.11', defaultRoute=None)
         h12 = self.addHost('h12', cls=Host, ip='10.0.0.12', defaultRoute=None)
```

```
#Add switches
    s2 = self.addSwitch('s2', cls=OVSKernelSwitch)
    s11 = self.addSwitch('s11', cls=OVSKernelSwitch)
    s18 = self.addSwitch('s18', cls=OVSKernelSwitch)
    s17 = self.addSwitch('s17', cls=OVSKernelSwitch)
    s14 = self.addSwitch('s14', cls=OVSKernelSwitch)
    s7 = self.addSwitch('s7', cls=OVSKernelSwitch)
    s15 = self.addSwitch('s15', cls=OVSKernelSwitch)
    s13 = self.addSwitch('s13', cls=OVSKernelSwitch)
    s10 = self.addSwitch('s10', cls=OVSKernelSwitch)
    s1 = self.addSwitch('s1', cls=OVSKernelSwitch)
    s4 = self.addSwitch('s4', cls=OVSKernelSwitch)
    s8 = self.addSwitch('s8', cls=OVSKernelSwitch)
    s9 = self.addSwitch('s9', cls=OVSKernelSwitch)
    s5 = self.addSwitch('s5', cls=OVSKernelSwitch)
    s12 = self.addSwitch('s12', cls=OVSKernelSwitch)
    s3 = self.addSwitch('s3', cls=OVSKernelSwitch)

#Add links
    self.addLink(s9, s7)
    self.addLink(s2, s9)
    self.addLink(s10, s1)
    self.addLink(s5, h1)
    self.addLink(s5, h2)
    self.addLink(s5, s4)
    self.addLink(s4, s3)
    self.addLink(s3, s2)
    self.addLink(s2, s1)
    self.addLink(s7, s8)
    self.addLink(s8, s5)
    self.addLink(s12, s5)
    self.addLink(s12, s11)
    self.addLink(s11, s10)
    self.addLink(s10, s9)
    self.addLink(s4, s13)
    self.addLink(s8, s13)
    self.addLink(s13, s12)
    self.addLink(s13, h3)
    self.addLink(s13, h4)
    self.addLink(s11, s14)
    self.addLink(s14, s15)
    self.addLink(s15, h5)
    self.addLink(s15, h6)
    self.addLink(s3, s17)
    self.addLink(s17, s18)
    self.addLink(s18, h7)
    self.addLink(s18, h8)
```

```
        self.addLink(s17, s7)
        self.addLink(s11, s17)
        self.addLink(s14, s7)
        self.addLink(s14, s3)
        self.addLink(h9, s1)
        self.addLink(h10, s1)
        self.addLink(s9, h11)
        self.addLink(s9, h12)
        self.addLink(s1, s7)

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

ANEXO 2: Módulo de balance de carga para OpenDaylight

El código presentado en esta sección sirve para crear solamente 2 flujos, que se encargan de direccionar los paquetes cuando se envían desde un host hx a un host hy y de hy a hx. Por lo anterior, el código solo sirve para realizar balance de carga entre dos hosts, si se requiere implementar entre más de un par de hosts es necesario reemplazar el número del flujo que aparece en color rojo y subrayado en el código por otros flujos. Por ejemplo, los flujos 1 y 2 son los programados en el código, y para poder aplicar el módulo a otro par de hosts en la misma red sin eliminar los flujos creados anteriormente, se debe reemplazar el 1 por un 3 y el 2 por un 4; si se requiere realizar balance de carga agregando otro par de hosts, se reemplaza el 1 por un 5 y el 2 por un 6. Se recomienda crear diferentes archivos Python con los flujos ya creados (LoadBalancer, LoadBalacer2, LoadBalancer3, etc).
Por último, no olvidar eliminar los flujos generados mediante este módulo con la sentencia que aparece en la guía del controlador OpenDaylight, en el apartado: eliminación de flujos que se encuentra en capítulo 3, esto con el fin de no tener problemas en la conexión de los hosts a la hora de crear una nueva red. Se recomienda crear un archivo con la eliminación de todos los flujos de los switches para hacerlo de forma automática y no uno por uno, puesto que son muchos.

```
#!/usr/bin/env python

import requests
from requests.auth import HTTPBasicAuth
import json
import unicodedata
from subprocess import Popen, PIPE
import time
import networkx as nx
from sys import exit

# Method To Get REST Data In JSON Format
def getResponse(url,choice):

        response = requests.get(url, auth=HTTPBasicAuth('admin', 'admin'))

        if(response.ok):
                jData = json.loads(response.content)
                if(choice=="topology"):
                        topologyInformation(jData)
                elif(choice=="statistics"):
                        getStats(jData)
        else:
                response.raise_for_status()

def topologyInformation(data):
        global switch
```

```
        global deviceMAC
        global deviceIP
        global hostPorts
        global linkPorts
        global G
        global cost

        for i in data["network-topology"]["topology"]:
                for j in i["node"]:
                        # Device MAC and IP

                        if "host-tracker-service:addresses" in j:
                                for k in j["host-tracker-service:addresses"]:
                                        ip = k["ip"].encode('ascii','ignore')
                                        mac = k["mac"].encode('ascii','ignore')
                                        deviceMAC[ip] = mac
                                        deviceIP[mac] = ip

                        # Device Switch Connection and Port

                        if "host-tracker-service:attachment-points" in j:

                                for k in j["host-tracker-service:attachment-points"]:
                                        mac = k["corresponding-tp"].encode('ascii','ignore')
                                        mac = mac.split(":",1)[1]
                                        ip = deviceIP[mac]
                                        temp = k["tp-id"].encode('ascii','ignore')
                                        switchID = temp.split(":")
                                        port = switchID[2]
                                        hostPorts[ip] = port
                                        switchID = switchID[0] + ":" + switchID[1]
                                        switch[ip] = switchID

        # Link Port Mapping
        for i in data["network-topology"]["topology"]:
                for j in i["link"]:
                        if "host" not in j['link-id']:
                                src = j["link-id"].encode('ascii','ignore').split(":")
                                srcPort = src[2]
                                dst = j["destination"]["dest-tp"].encode('ascii','ignore').split(":")
                                dstPort = dst[2]
                                srcToDst = src[1] + "::" + dst[1]
                                linkPorts[srcToDst] = srcPort + "::" + dstPort
                                G.add_edge((int)(src[1]),(int)(dst[1]))

def getStats(data):
        print "\nCost Computation....\n"
```

```
	global cost
	txRate = 0
	for i in data["node-connector"]:
		tx = int(i["opendaylight-port-statistics:flow-capable-node-connector-
statistics"]["packets"]["transmitted"])
		rx = int(i["opendaylight-port-statistics:flow-capable-node-connector-
statistics"]["packets"]["received"])
		txRate = tx + rx
		#print txRate

	time.sleep(2)

	response = requests.get(stats, auth=HTTPBasicAuth('admin', 'admin'))
	tempJSON = ""
	if(response.ok):
		tempJSON = json.loads(response.content)

	for i in tempJSON["node-connector"]:
		tx = int(i["opendaylight-port-statistics:flow-capable-node-connector-
statistics"]["packets"]["transmitted"])
		rx = int(i["opendaylight-port-statistics:flow-capable-node-connector-
statistics"]["packets"]["received"])
		cost = cost + tx + rx - txRate

	#cost = cost + txRate
	#print cost

def systemCommand(cmd):
	terminalProcess = Popen(cmd, stdout=PIPE, stderr=PIPE, shell=True)
	terminalOutput, stderr = terminalProcess.communicate()
	print "\n*** Flow Pushed\n"

def pushFlowRules(bestPath):

	bestPath = bestPath.split("::")

	for currentNode in range(0, len(bestPath)-1):
		if (currentNode==0):
			inport = hostPorts[h2]
			srcNode = bestPath[currentNode]
			dstNode = bestPath[currentNode+1]
			outport = linkPorts[srcNode + "::" + dstNode]
			outport = outport[0]
		else:
			prevNode = bestPath[currentNode-1]
			#print prevNode
			srcNode = bestPath[currentNode]
```

```
                    #print srcNode
                    dstNode = bestPath[currentNode+1]
                    inport = linkPorts[prevNode + "::" + srcNode]
                    inport = inport.split("::")[1]
                    outport = linkPorts[srcNode + "::" + dstNode]
                    outport = outport.split("::")[0]

                xmlSrcToDst = '\'<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"no\"?><flow
xmlns=\"urn:opendaylight:flow:inventory\"><priority>32767</priority><flow-name>Load
Balance 1</flow-name><match><in-port>' + str(inport) +'</in-port><ipv4-destination>'+ h1
+'/32</ipv4-destination><ipv4-source>'+ h2 +'/32</ipv4-source><ethernet-match><ethernet-
type><type>2048</type></ethernet-type></ethernet-
match></match><id>1</id><table_id>0</table_id><instructions><instruction><order>0</order
><apply-actions><action><order>0</order><output-action><output-node-connector>' +
str(outport) +'</output-node-connector></output-action></action></apply-
actions></instruction></instructions></flow>\"

                xmlDstToSrc = '\'<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"no\"?><flow
xmlns=\"urn:opendaylight:flow:inventory\"><priority>32767</priority><flow-name>Load
Balance 2</flow-name><match><in-port>' + str(outport) +'</in-port><ipv4-destination>'+ h2
+'/32</ipv4-destination><ipv4-source>'+ h1 +'/32</ipv4-source><ethernet-match><ethernet-
type><type>2048</type></ethernet-type></ethernet-
match></match><id>2</id><table_id>0</table_id><instructions><instruction><order>0</order
><apply-actions><action><order>0</order><output-action><output-node-connector>' +
str(inport) +'</output-node-connector></output-action></action></apply-
actions></instruction></instructions></flow>\"

                flowURL = "http://127.0.0.1:8181/restconf/config/opendaylight-
inventory:nodes/node/openflow:"+ bestPath[currentNode] +"/table/0/flow/1"

                command = 'curl --user "admin":"admin" -H "Accept: application/xml" -H
"Content-type: application/xml" -X PUT ' + flowURL + ' -d ' + xmlSrcToDst

                systemCommand(command)

                flowURL = "http://127.0.0.1:8181/restconf/config/opendaylight-
inventory:nodes/node/openflow:"+ bestPath[currentNode] +"/table/0/flow/2"

                command = 'curl --user "admin":"admin" -H "Accept: application/xml" -H
"Content-type: application/xml" -X PUT ' + flowURL + ' -d ' + xmlDstToSrc

                systemCommand(command)

        srcNode = bestPath[-1]
        prevNode = bestPath[-2]
```

```
        inport = linkPorts[prevNode + "::" + srcNode]
        inport = inport.split("::")[1]
        outport = hostPorts[h1]

        xmlSrcToDst = '\'<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?><flow
xmlns=\"urn:opendaylight:flow:inventory\"><priority>32767</priority><flow-name>Load
Balance 1</flow-name><match><in-port>' + str(inport) +'</in-port><ipv4-destination>'+ h1
+'/32</ipv4-destination><ipv4-source>'+ h2 +'/32</ipv4-source><ethernet-match><ethernet-
type><type>2048</type></ethernet-type></ethernet-
match></match><id>1</id><table_id>0</table_id><instructions><instruction><order>0</order
><apply-actions><action><order>0</order><output-action><output-node-connector>' +
str(outport) +'</output-node-connector></output-action></action></apply-
actions></instruction></instructions></flow>\"

        xmlDstToSrc = '\'<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?><flow
xmlns=\"urn:opendaylight:flow:inventory\"><priority>32767</priority><flow-name>Load
Balance 2</flow-name><match><in-port>' + str(outport) +'</in-port><ipv4-destination>' + h2
+'/32</ipv4-destination><ipv4-source>'+ h1 +'/32</ipv4-source><ethernet-match><ethernet-
type><type>2048</type></ethernet-type></ethernet-
match></match><id>2</id><table_id>0</table_id><instructions><instruction><order>0</order
><apply-actions><action><order>0</order><output-action><output-node-connector>' +
str(inport) +'</output-node-connector></output-action></action></apply-
actions></instruction></instructions></flow>\"

        flowURL = "http://127.0.0.1:8181/restconf/config/opendaylight-
inventory:nodes/node/openflow:"+ bestPath[-1] +"/table/0/flow/1"

        command = 'curl --user \"admin\":\"admin\" -H \"Accept: application/xml\" -H \"Content-
type: application/xml\" -X PUT ' + flowURL + ' -d ' + xmlSrcToDst

        systemCommand(command)

        flowURL = "http://127.0.0.1:8181/restconf/config/opendaylight-
inventory:nodes/node/openflow:"+ bestPath[-1] +"/table/0/flow/2"

        command = 'curl --user "admin":"admin" -H "Accept: application/xml" -H "Content-type:
application/xml" -X PUT ' + flowURL + ' -d ' + xmlDstToSrc

        systemCommand(command)

# Main

# Stores H1 and H2 from user
global h1,h2,h3

h1 = ""
h2 = ""
```

```
print "Enter Host 1"
h1 = int(input())
print "\nEnter Host 2"
h2 = int(input())
print "\nEnter Host 3 (H2's Neighbour)"
h3 = int(input())

h1 = "10.0.0." + str(h1)
h2 = "10.0.0." + str(h2)
h3 = "10.0.0." + str(h3)

flag = True

while flag:

        #Creating Graph
        G = nx.Graph()

        # Stores Info About H3 And H4's Switch
        switch = {}

        # MAC of Hosts i.e. IP:MAC
        deviceMAC = {}

        # IP of Hosts i.e. MAC:IP
        deviceIP = {}

        # Stores Switch Links To H3 and H4's Switch
        switchLinks = {}

        # Stores Host Switch Ports
        hostPorts = {}

        # Stores Switch To Switch Path
        path = {}
        # Stores Link Ports
        linkPorts = {}
        # Stores Final Link Rates
        finalLinkTX = {}
        # Store Port Key For Finding Link Rates
        portKey = ""
        # Statistics
        global stats
        stats = ""
        # Stores Link Cost
        global cost
```

```
        cost = 0

        try:
                # Device Info (Switch To Which The Device Is Connected & The MAC Address
Of Each Device)
                topology = "http://127.0.0.1:8181/restconf/operational/network-topology:network-
topology"
                getResponse(topology,"topology")

                # Print Device:MAC Info
                print "\nDevice IP & MAC\n"
                print deviceMAC

                # Print Switch:Device Mapping
                print "\nSwitch:Device Mapping\n"
                print switch

                # Print Host:Port Mapping
                print "\nHost:Port Mapping To Switch\n"
                print hostPorts

                # Print Switch:Switch Port:Port Mapping
                print "\nSwitch:Switch Port:Port Mapping\n"
                print linkPorts

                # Paths
                print "\nAll Paths\n"
                #for path in nx.all_simple_paths(G, source=2, target=1):
                        #print(path)
                for path in nx.all_shortest_paths(G, source=int(switch[h2].split(":",1)[1]),
target=int(switch[h1].split(":",1)[1]), weight=None):
                        print path

                # Cost Computation
                tmp = ""
                for currentPath in nx.all_shortest_paths(G, source=int(switch[h2].split(":",1)[1]),
target=int(switch[h1].split(":",1)[1]), weight=None):
                        for node in range(0,len(currentPath)-1):
                                tmp = tmp + str(currentPath[node]) + "::"
                                key = str(currentPath[node])+ "::" + str(currentPath[node+1])
                                port = linkPorts[key]
                                port = port.split(":",1)[0]
                                port = int(port)
                                stats = "http://localhost:8181/restconf/operational/opendaylight-
inventory:nodes/node/openflow:"+str(currentPath[node])+"/node-
connector/openflow:"+str(currentPath[node])+":"+str(port)
                                getResponse(stats,"statistics")
```

```
                tmp = tmp + str(currentPath[len(currentPath)-1])
                tmp = tmp.strip("::")
                finalLinkTX[tmp] = cost
                cost = 0
                tmp = ""

        print "\nFinal Link Cost\n"
        print finalLinkTX

        shortestPath = min(finalLinkTX, key=finalLinkTX.get)
        print "\n\nShortest Path: ",shortestPath

        pushFlowRules(shortestPath)
        time.sleep(60)
except KeyboardInterrupt:
        break
        exit
```

ANEXO 3: Módulo de balance de carga para Floodlight

```
#!/usr/bin/env python

import requests
#from request.packages.urllib3.exceptions import HTTPError as BaseHTTPError
import json
import unicodedata
from subprocess import Popen, PIPE
import time
import networkx as nx
from sys import exit

# Method To Get REST Data In JSON Format
def getResponse(url,choice):

        print url
        response = requests.get(url)

        if(response.ok):
                jData = json.loads(response.content)
                print jData
                print " \ "
                if(choice=="deviceInfo"):
                        deviceInformation(jData)
                elif(choice=="findSwitchLinks"):
                        findSwitchLinks(jData,switch[h2])
                elif(choice=="linkTX"):
                        #print portKey
                        linkTX(jData,portKey)

        else:
                response.raise_for_status()

# Parses JSON Data To Find Switch Connected To H4
def deviceInformation(jdata):
        global switch
        global deviceMAC
        global hostPorts
        switchDPID = ""
        data=jdata['devices']
        for i in data:
                if(i['ipv4']):
                        ip = i['ipv4'][0].encode('ascii','ignore')
                        mac = i['mac'][0].encode('ascii','ignore')
                        deviceMAC[ip] = mac
```

```
                        for j in i['attachmentPoint']:
                                for key in j:
                                        temp = key.encode('ascii','ignore')
                                        if(temp=="switch"):#antes switchDPID
                                                switchDPID = j[key].encode('ascii','ignore')
                                                switch[ip] = switchDPID
                                        elif(temp=="port"):
                                                portNumber = j[key]
                                                switchShort = switchDPID.split(":")[7]
                                                hostPorts[ip+ "::" + switchShort] = str(portNumber)

# Finding Switch Links Of Common Switch Of H3, H4

def findSwitchLinks(data,s):
        global switchLinks
        global linkPorts
        global G

        links=[]
        for i in data:
                src = i['src-switch'].encode('ascii','ignore')
                dst = i['dst-switch'].encode('ascii','ignore')

                srcPort = str(i['src-port'])
                dstPort = str(i['dst-port'])

                srcTemp = src.split(":")[7]
                dstTemp = dst.split(":")[7]

                G.add_edge(int(srcTemp,16), int(dstTemp,16))

                tempSrcToDst = srcTemp + "::" + dstTemp
                tempDstToSrc = dstTemp + "::" + srcTemp

                portSrcToDst = str(srcPort) + "::" + str(dstPort)
                portDstToSrc = str(dstPort) + "::" + str(srcPort)

                linkPorts[tempSrcToDst] = portSrcToDst
                linkPorts[tempDstToSrc] = portDstToSrc

                if (src==s):
                        links.append(dst)
                elif (dst==s):
                        links.append(src)
                else:
                        continue
```

```
        switchID = s.split(":")[7]
        switchLinks[switchID]=links

# Finds The Path To A Switch

def findSwitchRoute():
        pathKey = ""
        nodeList = []
        src = int(switch[h2].split(":",7)[7],16)
        dst = int(switch[h1].split(":",7)[7],16)
        print "switch fuente "+ str(src)
        print "switch destino "+ str(dst)
        for currentPath in nx.all_shortest_paths(G, source=src, target=dst, weight=None):
                for node in currentPath:

                        tmp = ""
                        if node < 17:
                                pathKey = pathKey + "0" + str(hex(node)).split("x",1)[1] + "::"
                                tmp = "00:00:00:00:00:00:00:0" + str(hex(node)).split("x",1)[1]
                        else:
                                pathKey = pathKey + str(hex(node)).split("x",1)[1] + "::"
                                tmp = "00:00:00:00:00:00:00:" + str(hex(node)).split("x",1)[1]
                        nodeList.append(tmp)

                pathKey=pathKey.strip("::")
                path[pathKey] = nodeList
                pathKey = ""
                nodeList = []

        print "Este es path "
        print path

# Computes Link TX

def linkTX(data,key):
        global cost
        print data
        port = linkPorts[key]
        print port
        port = port.split("::")[0]
        print port
        for i in data:
                print i['port']
                if i['port']==port:
                        cost = cost + (int)(i['bits-per-second-tx'])
                        print "este es el costo " + str(cost)
```

```
# Method To Compute Link Cost

def getLinkCost():
	global portKey
	global cost

	for key in path:
		start = switch[h2]
		print "estoy aqui 1 " + str(start)
		src = switch[h2]
		print "estoy aqui 2 " + str(src)
		srcShortID = src.split(":")[7]
		print "estoy aqui 3 " + str(srcShortID)
		mid = path[key][1].split(":")[7]
		print "estoy aqui 4 " + str(mid)
		print "path[key][1] es "
		print  path[key][1]
		print  len(key.split("::"))
		#longitud= len(key.split("::"))
		for link in path[key]:
			temp = link.split(":")[7]
			print temp

			if srcShortID==temp:
				continue
			else:
				portKey = srcShortID + "::" + temp
				stats = "http://localhost:8080/wm/statistics/bandwidth/" + src +
"/all/json"
				getResponse(stats,"linkTX")
				 # l-----> antes era 0
				srcShortID = temp
				print "estoy aqui 5 " + str(srcShortID)
				src = link
				print "estoy aqui 6 " + str(src)

			if len(key.split("::")) == 3:
				portKey = start.split(":")[7] + "::" + mid + "::" +
switch[h1].split(":")[7]
				print portKey
			elif len(key.split("::")) == 4:
				portKey = start.split(":")[7] + "::" + mid + "::" +
path[key][2].split(":")[7] + "::" + switch[h1].split(":")[7]
				print portKey
			elif len(key.split("::")) == 5:
```

```
                                portKey = start.split(":")[7] + "::" + mid + "::" +
path[key][2].split(":")[7] + "::" + path[key][3].split(":")[7] +"::" + switch[h1].split(":")[7]
                                print portKey
                        elif len(key.split("::")) == 6:
                                portKey = start.split(":")[7] + "::" + mid + "::" +
path[key][2].split(":")[7] + "::" + path[key][3].split(":")[7] +"::" + path[key][4].split(":")[7] + "::"
+ switch[h1].split(":")[7]
                                print portKey
                        elif len(key.split("::")) == 7:
                                portKey = start.split(":")[7] + "::" + mid + "::" +
path[key][2].split(":")[7] + "::" + path[key][3].split(":")[7] +"::" + path[key][4].split(":")[7] + "::"
+ path[key][5].split(":")[7] + "::" + switch[h1].split(":")[7]
                                print portKey
                        elif len(key.split("::")) == 8:
                                portKey = start.split(":")[7] + "::" + mid + "::" +
path[key][2].split(":")[7] + "::" + path[key][3].split(":")[7] +"::" + path[key][4].split(":")[7] + "::"
+ path[key][5].split(":")[7] + "::" + path[key][6].split(":")[7] + "::" + switch[h1].split(":")[7]
                                print portKey
                        elif len(key.split("::")) == 9:
                                portKey = start.split(":")[7] + "::" + mid + "::" +
path[key][2].split(":")[7] + "::" + path[key][3].split(":")[7] +"::" + path[key][4].split(":")[7] + "::"
+ path[key][5].split(":")[7] + "::" + path[key][6].split(":")[7] + "::" + path[key][7].split(":")[7] +
"::" + switch[h1].split(":")[7]
                                print portKey

                finalLinkTX[portKey] = cost
                cost = 0
                portKey = ""

def systemCommand(cmd):
        terminalProcess = Popen(cmd, stdout=PIPE, stderr=PIPE, shell=True)
        terminalOutput, stderr = terminalProcess.communicate()
        print "\n***", terminalOutput, "\n"

def flowRule(currentNode, flowCount, inPort, outPort, staticFlowURL):
        flow = {
                'switch':"00:00:00:00:00:00:00:" + currentNode,
           "name":"flow" + str(flowCount),
           "cookie":"0",
           "priority":"32768",
           "in_port":inPort,
                "eth_type": "0x0800",
                "ipv4_src": h2,
                "ipv4_dst": h1,
                "eth_src": deviceMAC[h2],
                "eth_dst": deviceMAC[h1],
           "active":"true",
```

```
        "actions":"output=" + outPort
        }

        jsonData = json.dumps(flow)

        cmd = "curl -X POST -d \'" + jsonData + "\' " + staticFlowURL

        systemCommand(cmd)

        flowCount = flowCount + 1

        flow = {
                'switch':"00:00:00:00:00:00:00:" + currentNode,
        "name":"flow" + str(flowCount),
        "cookie":"0",
        "priority":"32768",
        "in_port":outPort,
                "eth_type": "0x0800",
                "ipv4_src": h1,
                "ipv4_dst": h2,
                "eth_src": deviceMAC[h1],
                "eth_dst": deviceMAC[h2],
        "active":"true",
        "actions":"output=" + inPort
        }

        jsonData = json.dumps(flow)

        cmd = "curl -X POST -d \'" + jsonData + "\' " + staticFlowURL

        systemCommand(cmd)

def addFlow():
        print "KAAM CHALU HAI"

        # Deleting Flow
        #cmd = "curl -X DELETE -d '{"00:00:00:00:00:00:00:0d": {"flows":[1]}}'
http://127.0.0.1:8080/wm/core/switch/all/flow/json"
        #systemCommand(cmd)

        eliminar="http://localhost:8080/wm/staticflowentrypusher/clear/all/json"

        #cmd = "curl -X DELETE -d \'{\"name\":\"flow2\"}\'
http://127.0.0.1:8080/wm/core/switch/all/flow/json"
        #systemCommand(cmd)

        flowCount = 1
```

```
        staticFlowURL = "http://127.0.0.1:8080/wm/staticflowpusher/json"

        shortestPath = min(finalLinkTX, key=finalLinkTX.get)
        print "\n\nShortest Path: ",shortestPath


        currentNode = shortestPath.split("::",2)[0]
        nextNode = shortestPath.split("::")[1]

        # Port Computation

        port = linkPorts[currentNode+"::"+nextNode]
        outPort = port.split("::")[0]
        inPort = hostPorts[h2+"::"+switch[h2].split(":")[7]]

        flowRule(currentNode,flowCount,inPort,outPort,staticFlowURL)

        flowCount = flowCount + 2

        print path
        bestPath = path[shortestPath]
        previousNode = currentNode

        for currentNode in range(0,len(bestPath)):
                if previousNode == bestPath[currentNode].split(":")[7]:
                        continue
                else:
                        port = linkPorts[bestPath[currentNode].split(":")[7]+"::"+previousNode]
                        inPort = port.split("::")[0]
                        outPort = ""
                        if(currentNode+1<len(bestPath) and
bestPath[currentNode]==bestPath[currentNode+1]):
                                currentNode = currentNode + 1
                                continue
                        elif(currentNode+1<len(bestPath)):
                                port =
linkPorts[bestPath[currentNode].split(":")[7]+"::"+bestPath[currentNode+1].split(":")[7]]
                                outPort = port.split("::")[0]
                        elif(bestPath[currentNode]==bestPath[-1]):
                                outPort = str(hostPorts[h1+"::"+switch[h1].split(":")[7]])


        flowRule(bestPath[currentNode].split(":")[7],flowCount,str(inPort),str(outPort),staticFlow
URL)
                        flowCount = flowCount + 2
                        previousNode = bestPath[currentNode].split(":")[7]
```

```
# Method To Perform Load Balancing
def loadbalance():
        linkURL = "http://localhost:8080/wm/topology/links/json"
        getResponse(linkURL,"findSwitchLinks")

        findSwitchRoute()
        getLinkCost()
        addFlow()

# Main

# Stores H1 and H2 from user
global h1,h2,h3

h1 = ""
h2 = ""

print "Enter Host 1"
h1 = int(input())
print "\nEnter Host 2"
h2 = int(input())
print "\nEnter Host 3 (H2's Neighbour)"
h3 = int(input())

h1 = "10.0.0." + str(h1)
h2 = "10.0.0." + str(h2)
h3 = "10.0.0." + str(h3)


while True:

        # Stores Info About H3 And H4's Switch
        switch = {}

        # Mac of H3 And H4
        deviceMAC = {}

        # Stores Host Switch Ports
        hostPorts = {}

        # Stores Switch To Switch Path
        path = {}

        # Switch Links

        switchLinks = {}
```

```
        # Stores Link Ports
        linkPorts = {}

        # Stores Final Link Rates
        finalLinkTX = {}

        # Store Port Key For Finding Link Rates
        portKey = ""

        # Stores Link Cost
        cost = 0
        # Graph
        G = nx.Graph()

        try:

                # Enables Statistics Like B/W, etc
                enableStats = "http://localhost:8080/wm/statistics/config/enable/json"
                requests.put(enableStats)

                # Device Info (Switch To Which The Device Is Connected & The MAC Address
Of Each Device)
                deviceInfo = "http://localhost:8080/wm/device/"
                getResponse(deviceInfo,"deviceInfo")

                # Load Balancing

                loadbalance()

                # ---------- PRINT ----------

                print "\n\n############ RESULT ############\n\n"

                # Print Switch To Which H4 is Connected
                print "Switch H4: ",switch[h3], "\tSwitchH3: ", switch[h2]

                print "\n\nSwitch H1: ", switch[h1]

                # IP & MAC
                print "\nIP & MAC\n\n", deviceMAC

                # Host Switch Ports
                print "\nHost::Switch Ports\n\n", hostPorts

                # Link Ports
                print "\nLink Ports (SRC::DST - SRC PORT::DST PORT)\n\n", linkPorts
```

```
        # Alternate Paths
        print "\nPaths (SRC TO DST)\n\n",path

        # Final Link Cost
        print "\nFinal Link Cost (First To Second Switch)\n\n",finalLinkTX

        print "\n\n########################################\n\n"

        time.sleep(60)

except KeyboardInterrupt:
        break
        exit()
```

ANEXO 4: Código de la red personalizada 2

```
#!/usr/bin/python

from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch
from mininet.topo import Topo

class fatTreeTopo(Topo):

   "Fat Tree Topology"

   def __init__(self):
      "Create Fat tree Topology"
      Topo.__init__(self)

         #Add hosts
         h3 = self.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
         h2 = self.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
         h5 = self.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)
         h1 = self.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
         h4 = self.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
         h6 = self.addHost('h6', cls=Host, ip='10.0.0.6', defaultRoute=None)
         h7 = self.addHost('h7', cls=Host, ip='10.0.0.7', defaultRoute=None)
         h8 = self.addHost('h8', cls=Host, ip='10.0.0.8', defaultRoute=None)
         h9 = self.addHost('h9', cls=Host, ip='10.0.0.9', defaultRoute=None)
         h10 = self.addHost('h10', cls=Host, ip='10.0.0.10', defaultRoute=None)
         h11 = self.addHost('h11', cls=Host, ip='10.0.0.11', defaultRoute=None)
         h12 = self.addHost('h12', cls=Host, ip='10.0.0.12', defaultRoute=None)
         h13 = self.addHost('h13', cls=Host, ip='10.0.0.13', defaultRoute=None)
         h14 = self.addHost('h14', cls=Host, ip='10.0.0.14', defaultRoute=None)

         #Add switches
         s1 = self.addSwitch( 's1',cls=OVSKernelSwitch)
         s2 = self.addSwitch( 's2',cls=OVSKernelSwitch)
         s3 = self.addSwitch( 's3',cls=OVSKernelSwitch)
         s4 = self.addSwitch( 's4',cls=OVSKernelSwitch)
         s5 = self.addSwitch( 's5',cls=OVSKernelSwitch)
         s6 = self.addSwitch( 's6',cls=OVSKernelSwitch)
         s7 = self.addSwitch( 's7',cls=OVSKernelSwitch)
         s8 = self.addSwitch( 's8',cls=OVSKernelSwitch)
         s9 = self.addSwitch( 's9',cls=OVSKernelSwitch)
         s10 = self.addSwitch( 's10',cls=OVSKernelSwitch)
         s11 = self.addSwitch( 's11',cls=OVSKernelSwitch)
         s12 = self.addSwitch( 's12',cls=OVSKernelSwitch)
         s13 = self.addSwitch( 's13',cls=OVSKernelSwitch)
```

```
        s14 = self.addSwitch( 's14',cls=OVSKernelSwitch)
        s15 = self.addSwitch( 's15',cls=OVSKernelSwitch)
        s16 = self.addSwitch( 's16',cls=OVSKernelSwitch)
        s17 = self.addSwitch( 's17',cls=OVSKernelSwitch)

        #Add links
        self.addLink(s4, s3)
        self.addLink(s3, s2)
        self.addLink(s4, s2)
        self.addLink(s2, s1)
        self.addLink(s1, h1)
        self.addLink(s1, h2)
        self.addLink(s5, s4)
        self.addLink(s5, h3)
        self.addLink(s5, h4)
        self.addLink(s4, s7)
        self.addLink(s7, s8)
        self.addLink(h5, s8)
        self.addLink(h6, s8)
        self.addLink(s3, s10)
        self.addLink(s7, s10)
        self.addLink(s10, s2)
        self.addLink(s3, s9)
        self.addLink(s4, s9)
        self.addLink(s2, s9)
        self.addLink(s9, s11)
        self.addLink(s7, s11)
        self.addLink(s10, s12)
        self.addLink(s11, s12)
        self.addLink(s7, s12)
        self.addLink(s11, s13)
        self.addLink(s12, s14)
        self.addLink(s13, s14)
        self.addLink(s12, s15)
        self.addLink(s15, h7)
        self.addLink(s15, h8)
        self.addLink(s9, s16)
        self.addLink(s16, h9)
        self.addLink(s16, h10)
        self.addLink(s13, s17)
        self.addLink(s14, s6)
        self.addLink(s6, h11)
        self.addLink(s6, h12)
        self.addLink(s17, h13)
        self.addLink(s17, h14)

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