UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS

FACULTAD DE INGENIERIA ELECTRICA

DEPARTAMENTO DE ELECTRÓNICA Y TELECOMUNICACIONES

PLATAFORMA DE PRUEBAS PARA EVALUAR EL DESEMPEÑO DE LAS REDES DEFINIDAS POR SOFTWARE BASADAS EN EL PROTOCOLO OPENFLOW

Tesis presentada en opción al Título Académico de Máster en Telemática

Maestría en Telemática

Autor: Ing. Yanko Antonio Marín Muro

Tutor: Dr. C. Ing. Félix Álvarez Paliza

Santa Clara, Cuba, 2016

UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS

FACULTAD DE INGENIERIA ELECTRICA

DEPARTAMENTO DE ELECTRÓNICA Y TELECOMUNICACIONES

PLATAFORMA DE PRUEBAS PARA EVALUAR EL DESEMPEÑO DE LAS REDES DEFINIDAS POR SOFTWARE BASADAS EN EL PROTOCOLO OPENFLOW

Tesis presentada en opción al Título Académico de Máster en Telemática

Maestría en Telemática

Autor: Ing. Yanko Antonio Marín Muro
ETECSA, yanko.marin@etecsa.cu

Tutor: Dr. C. Ing. Félix Álvarez Paliza
FIE, UCLV, fapaliza@uclv.edu.cu

Santa Clara, Cuba, 2016

Hago constar que la presente Tesis en Opción al Título Académico de Máster en Telemática fue realizada en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de los estudios correspondientes a la 5ta edición de Maestría en Telemática.

Autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Autor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

Pensamiento:

"¿Para qué, sino para poner paz entre los hombres han de ser los adelantos de la ciencia?"

José Martí

Agradecimientos y Dedicatorias:

Agradecimientos:

Quiero agradecer sinceramente a aquellas personas que compartieron sus conocimientos conmigo para hacer posible la conclusión de esta tesis. Especialmente agradezco a mi tutor el Dr. C. Ing. Félix Álvarez Paliza por su asesoría siempre dispuesta. Gracias a la Dra. C. Ileana Moreno Campdesuñer por su asesoría metodológica y al Ing. Abel Alfonso López Carbonell por colaborar activamente en la implementación de la plataforma de evaluación presentada en esta memoria.

A mis compañeros de trabajo del departamento de control de la DTSS y los del grupo de asistencia técnica del CORE de HUAWEI de ETECSA.

A los directivos y cuadros de ETECSA de la DTSS por darme la oportunidad y confiar en mí para la realización de este estudio.

Gracias a todos ellos.

Dedicatorias:

A mis padres, por su gran ejemplo de superación y valioso apoyo en todo momento.

A mi esposa Yailene por ese optimismo que siempre me impulsó a seguir adelante y por los días y horas que hizo el papel de madre y padre.

A mis hijos LIANET y LEONARDO, con todo el deseo de que crezca en ellos el deseo de superarse y de ser cada día mejores

Gracias por la inspiración

A toda mi familia, amistades y aquellas personas que siempre me han dado su apoyo tanto en los buenos como en los difíciles momentos que la vida me ha puesto por delante. A todos mis compañeros de estudio y profesores que de una forma u otra me han ayudado y han contribuido a obtener esta meta.

RESUMEN

El nacimiento de las aplicaciones en tiempo real, la transmisión de video; la masificación de las redes sociales, la introducción de la computación en la nube y muchos otros servicios, han dado como resultado el crecimiento exponencial del tráfico que circula por la red. A pesar de ello se continúan utilizando las mismas tecnologías que hace cincuenta años y los avances en cuanto a nuevas formas de comunicación y tratamiento de la información son casi inexistentes. Se ha intentado cubrir cada necesidad con la creación de protocolos que toman mucho tiempo en ser estandarizados. Si bien se ha dado una solución temporal a todos los requerimientos que demanda el mercado, ésta no representa una solución global al verdadero problema: la carencia de métodos de comunicación más eficientes. En consecuencia se plantea la necesidad de cambiar la forma de comunicación de las redes, en la cual se le proporcione mayor inteligencia a la misma.

Es así que nace el concepto de las Redes Definidas por Software (SDN, por sus siglas en inglés), el cual está revolucionando el campo de las comunicaciones mediante el control de los dispositivos de la red desde un *software* exterior, con la ayuda del protocolo OpenFlow creado especialmente para este propósito. OpenFlow propone nuevas características que permiten la programación de red y la experimentación sin la necesidad de que los fabricantes de equipos de red expongan la estructura interna de sus productos; basta con que se adicione compatibilidad con el estándar en los equipos de cada proveedor.

SDN es una tecnología que ofrece un amplio panorama de investigación y se prevé que en el futuro sea la base de una nueva era de las comunicaciones. El presente trabajo se centra en la implementación de una plataforma de pruebas para evaluar el desempeño de las SDN basadas en el protocolo OpenFlow. Para ello fueron analizados los fundamentos teóricos de este tipo de redes, los controladores disponibles en la industria y los criterios para su selección. Se hace un análisis de las herramientas de simulación, evaluación, desarrollo y depuración de problemas existentes y al mismo tiempo son identificadas las métricas relevantes para las SDN. Posteriormente se hace una evaluación de 6 controladores SDN basados en el protocolo OpenFlow y se desarrolla una aplicación que permite detectar intrusos en la red, así como controlar el acceso de los usuarios a los diferentes recursos de la misma. Finalmente se comprueba el impacto en la latencia y la tasa de transferencia de datos en la red cuando se aplican políticas de seguridad. La plataforma de prueba y las herramientas desarrolladas en esta memoria, contribuyen a sistematizar la toma de decisiones relacionadas con la selección y análisis del desempeño de los controladores SDN.

Palabras Claves: SDN (Software Defined Network), OPENFLOW, MININET, OpenFlow, network virtualization, network operating systems, programmable networks, network hypervisor, programming languages, flow based networking, scalability.

ÍNDICE

INTRODUCCIÓN

Durante mucho tiempo, las tecnologías de redes han evolucionado a un ritmo más lento en comparación con otras tecnologías de comunicaciones, lo cual ha incrementado la osificación de la red [1], [2]. La *Internet* basada en el protocolo de Internet (IP, por sus siglas en inglés) ha tenido gran éxito, centrándose principalmente en los conceptos tradicionales de conmutación y enrutamiento. Sin embargo, la complejidad de las redes actuales se enfrenta a problemas importantes entre los que figuran, la calidad del servicio, seguridad, gestión de la movilidad y escalabilidad [3].

Equipos de red como los conmutadores y enrutadores han sido tradicionalmente desarrollados por los fabricantes. Cada proveedor diseña su propia lógica de control para operar su dispositivo de una manera exclusiva y cerrada. Esto frena el avance de las innovaciones en las tecnologías de redes y provoca un aumento de los costos de gestión y operación cada vez que se despliegan nuevos servicios, tecnologías y equipos. Este es un problema universal, que ha hecho que cada día las arquitecturas de redes sean cada vez más complejas y tengan una baja capacidad de escalabilidad. Se han propuesto durante todos estos años muchas soluciones para reemplazar la tecnología de red actual, pero nunca se han aplicado por ser extremadamente difíciles de probar.

La arquitectura de las redes actuales se compone de tres planos lógicos: plano de control, plano de datos y el plano de gestión. Hasta ahora, los equipos de redes se han desarrollado con los planos de control y de datos estrechamente acoplados. Esto aumenta significativamente la complejidad, el costo de administración y la gestión de la red. Debido a estas limitaciones, las comunidades científicas de redes y líderes industriales del mercado han colaborado con el fin de replantear el diseño de las redes tradicionales. De esta manera, surgió un nuevo paradigma de redes llamado “Redes Programables” con el objetivo de facilitar la evolución de la red. El principio fundamental de las redes programables es permitir que la red sea más flexible, adaptable y dinámica [2]. Para materializar este concepto, surgieron por los años 90, dos escuelas separadas de pensamientos: OpenSig, de la comunidad de las telecomunicaciones y Redes Activas, de la comunidad de las redes IP. Entre las principales razones por las que estos grupos no tuvieron éxito figuran los problemas de rendimiento, complejidad y seguridad introducidos en sus propuestas [4].

Después de las redes programables, surgieron otros proyectos con un nuevo enfoque relacionado con la separación de los planos de control y de datos apoyado principalmente en los esfuerzos de crear interfaces abiertas y estándar entre estos dos planos; destacándose la arquitectura para la separación de los elementos de transmisión y control (ForCES, por sus siglas en inglés). Otros proyectos que promueven la separación de planos de control y de datos son la plataforma de control de enrutamiento (RCP, por sus siglas en inglés), la arquitectura segura para las redes empresariales (SANE, por sus siglas en inglés) y Ethane [4].

En los últimos años, también ha surgido el paradigma de las Redes Definidas por *Software* (SDN, por sus siglas en inglés) como una propuesta para superar las limitaciones anteriormente mencionadas. Recientemente, las Redes Definidas por *Software* han ganado popularidad en la academia y la industria. SDN no es una propuesta revolucionaria, pero es una reformulación de investigaciones anteriores, con los empeños de separar el plano de control y el plano de datos, así como centralizar la inteligencia de la red; abriendo un camino para la innovación mediante la programación de aplicaciones que pueden controlar directamente el plano de datos de la red. El éxito y el rápido progreso de las SDN son debido al triunfo del protocolo OpenFlow, unido a la nueva visión de la necesidad de un sistema operativo de red.

La fundación de redes abiertas (ONF, por sus siglas en inglés) es un consorcio de la industria sin fines de lucro. Éste se encarga del desarrollo y la estandarización de las SDN y del protocolo OpenFlow, único estándar hasta la fecha con la misión de la comunicación entre el control y los dispositivos de la red.

El paradigma de las redes tradicionales está cambiando rápidamente con el advenimiento de las SDN; de hecho, empresas que procesan grandes cantidades de datos, tales como Facebook y Google [5], están migrando sus redes hacia enfoques definidos por software. Incluso los proveedores de servicios de *Internet* (ISP, por sus siglas en inglés), como Verizon y AT&T, están valorando la creación de redes definidas por *software* como una alternativa viable para su infraestructura.

El reciente interés comercial en las SDN ha llevado al desarrollo de más de 50 controladores SDN basados en el protocolo OpenFlow y se observa claramente una aceptación por parte de la comunidad científica de que este nuevo paradigma de red seguirá proporcionando nuevas oportunidades de investigación. Estos controladores fueron creados por disímiles universidades o proveedores, escritos en diferentes lenguajes de programación, poseen múltiples aplicaciones y utilizan variadas técnicas que les permiten mejorar su desempeño.

Entre tanta variedad de controladores siempre surgen las siguientes interrogantes: ¿cuál de los controladores debe ser seleccionado y utilizado?, ¿cuál es la plataforma de cómputo ideal para un controlador?, ¿Cómo evaluar el desempeño de un controlador?, entre otras preguntas. Sin dudas, la selección de un controlador es un proceso complejo y al ser una nueva clase de producto, existen pocas herramientas para su evaluación.

Estudios recientes [6]–[11] han investigado el desempeño de los planos de datos y de control. Los enfoques hasta ahora, se han centrado en identificar qué controlador funciona mejor que los demás y han utilizado herramientas muy específicas que analizan métricas puntuales. Ninguno de los trabajos ha comparado la latencia de una red SDN con una tradicional o el impacto en la latencia y la tasa de transferencia de datos en la red cuando se aplican políticas de seguridad. Tampoco se ha creado una plataforma dirigida a automatizar las pruebas de escalabilidad en Mininet a diferentes tipos de topologías; ni para la medición en cada prueba de la memoria ocupada en el hipervisor donde corre el controlador. La automatización de la evaluación de controladores SDN no ha sido abordada en otras investigaciones sobre el tema; además de la medición de la utilización de memoria y CPU en el hipervisor donde corre el controlador.

Entonces, para desarrollar adecuadamente experimentos con redes SDN, los investigadores necesitan una nueva generación de plataformas de pruebas que les permita emular y evaluar de forma experimental este tipo de redes.

De lo anterior se infiere la necesidad de crear una plataforma de pruebas para la investigación, enseñanza y evaluación del desempeño de las redes SDN basadas en el protocolo OpenFlow. Lo cual conduce al **problema científico** de esta investigación:

¿Cómo contribuir a sistematizar la toma de decisiones relacionadas con la selección y análisis del desempeño de los controladores SDN en diferentes tipos de redes y escenarios?

En aras de responder a esa interrogante el **objeto de estudio** de este trabajo se enmarca en las Redes SDN basadas en el protocolo OpenFlow; y su **campo de acción** en las plataformas de pruebas para redes SDN utilizando herramientas de código abierto.

La presente memoria pretende dar a conocer la importancia de las Redes Definidas por Software y las posibilidades del protocolo OpenFlow como tecnologías que se proyectan a ser la base del funcionamiento de Internet. En virtud de ello se ha definido el siguiente **objetivo general**: Implementar una plataforma de pruebas para evaluar el desempeño de las SDN basadas en el protocolo OpenFlow; que permita contribuir a sistematizar la toma de decisiones relacionadas con la selección y análisis del desempeño de los controladores en diferentes tipos de redes y escenarios.

Con vistas a alcanzar este objetivo general se deben completar una serie de objetivos específicos que se resumen en los siguientes puntos:

- Sistematizar los fundamentos teóricos sobre las SDN basadas en el protocolo OpenFlow.
- Determinar las métricas de desempeño relevantes en las SDN basadas en el protocolo OpenFlow y los criterios para la selección del controlador.

- Diseñar experimentos utilizando campos de prueba virtuales de código abierto que permitan analizar el desempeño de las SDN.
- Evaluar el desempeño de diferentes controladores SDN teniendo en cuenta algunas de las métricas estudiadas.

Las tareas de investigación que se realizaron para el cumplimiento de los objetivos son:

- Análisis de los fundamentos teóricos de las SDN y la necesidad de una nueva arquitectura de red de comunicaciones.
- Estudio de los controladores disponibles en la industria y los criterios para su selección.
- Análisis de las herramientas de simulación, evaluación, desarrollo y depuración de problemas.
- Identificación de las métricas relevantes para las SDN.
- Selección de los controladores, el emulador, la versión del protocolo OpenFlow y las herramientas de evaluación.
- Evaluación de controladores SDN basados en el protocolo OpenFlow.
- Desarrollo de una aplicación SDN que permita detectar intrusos en la red y controlar el acceso de los usuarios a los diferentes recursos de la misma.
- Comprobación del impacto en la latencia y la tasa de transferencia de datos en la red cuando se aplican políticas de seguridad.

El cumplimiento de estos objetivos contribuye a la evaluación de controladores SDN. Por lo que los principales aportes de este trabajo se concretan en:

- Una contribución teórica al sistematizar la información relativa a las SDN.
- Desde el punto de vista práctico se ofrece una plataforma de pruebas que permite el estudio y evaluación de las SDN.
- El aporte social radica en la contribución a la sistematización y la toma de decisiones relacionadas con la selección y análisis del desempeño de los controladores SDN.
- El aporte metodológico consiste en la implementación de una plataforma de pruebas que puede contribuir a la realización de futuras investigaciones.

La investigación presenta además como novedad:

- El desarrollo de varias aplicaciones en el *bash* de Linux dirigidas a automatizar las pruebas de escalabilidad en Mininet a diferentes tipos de topologías; así como a la medición en cada prueba de la memoria ocupada en el hipervisor donde corre el controlador.
- La automatización de la evaluación de controladores SDN; la cual no ha sido abordada en otras investigaciones sobre el tema; además de la medición de la utilización de memoria y CPU en el hipervisor donde corre el controlador.
- El desarrollo de una aplicación SDN que permite en tiempo real detectar intrusos en la red, así como aplicar políticas de acceso a la misma en todo su conjunto.
- La evaluación del impacto en la latencia y la tasa de trasferencia de datos en la red cuando se aplican políticas de seguridad.

La producción científica relacionada con esta investigación incluye la publicación del siguiente artículo:

- Colaboración de IMS y SDN basadas en el protocolo OpenFlow. Una arquitectura para mitigar problemas de seguridad en redes futuras. VII Simposio de Telecomunicaciones, INFORMATICA 2016.

La memoria escrita está estructurada en introducción, tres capítulos, conclusiones, recomendaciones y anexos. En el capítulo uno se presentan las principales características y potencialidades de las Redes Definidas por Software, así como sus paradigmas; necesarios para la comprensión y el correcto desarrollo de esta investigación.

El capítulo dos muestra los controladores SDN comerciales y de código abierto. También se identifican los elementos claves para escoger un controlador, con vistas a sistematizar la toma de decisiones relacionadas con la selección de este tipo de elemento de red. Además, son identificadas las métricas de desempeño que usualmente se utilizan para evaluar los mismos. Finalmente, son abordadas las plataformas de simulación, herramientas para la depuración, solución de problemas y de evaluación existentes.

En el capítulo tres se implementa la plataforma de pruebas que permite evaluar controladores, conmutadores y arquitecturas de cómputo. También son presentados los experimentos que permiten comparar el desempeño de una red tradicional vs SDN. Se realizan pruebas de escalabilidad que permiten conocer el número máximo de nodos que puede alcanzar una red SDN. Finalmente se desarrolla una aplicación SDN que permite detectar intrusos en la red así como otra aplicación que facilita la evaluación de controladores de forma automática.

En las conclusiones se realiza un análisis del cumplimiento de los objetivos específicos planteados en el trabajo y se hacen recomendaciones para futuras investigaciones. En los anexos aparecen fragmentos de los programas y *script* desarrollados.

CAPÍTULO 1. FUNDAMENTOS TEÓRICOS SOBRE LAS SDN BASADAS EN EL PROTOCOLO OPENFLOW

En este capítulo se presentan las principales características y potencialidades de las Redes Definidas por Software, así como sus paradigmas; necesarios para la comprensión y el correcto desarrollo de esta investigación. Primeramente se exponen las limitaciones de las tecnologías de redes convencionales que hacen necesario pensar en la necesidad de una nueva arquitectura de red. Posteriormente son analizadas las definiciones, beneficios de este tipo de redes. A la postre es estudiada la arquitectura SDN y sus abstracciones fundamentales donde son abordadas las funciones de cada capa.

1.1 Necesidad de una nueva arquitectura de red de comunicaciones

El incremento vertiginoso de los dispositivos móviles, las redes sociales, la virtualización, los servicios en la nube, el acceso ubicuo, el incremento creciente de la demanda de grandes velocidades de conexión, la necesidad de movilidad y la modificación del patrón de tráfico de internet por parte de los usuarios hacia los centros de datos, son algunas de las innovaciones que han impulsado a la industria de las tecnologías de la información y las comunicaciones (ICT,por sus siglas en inglés) a reexaminar las tecnologías de red tradicionales [2], [4], [12]–[20].

Cumplir con los requisitos actuales del mercado es prácticamente imposible con las arquitecturas de red tradicionales y la principal causa es que estas no fueron diseñadas para satisfacer las necesidades de los usuarios de hoy en día. Actualmente los diseñadores de redes afrontan limitaciones que son intrínsecas de los modelos de redes convencionales y estas son [12]:

- **Complejidad de la arquitectura de red actual**

La tecnología de red hasta la fecha ha consistido en gran medida en un conjunto de protocolos discretos diseñados para conectar *hosts* de forma fiable a través de distancias, velocidades de enlace, y topologías variadas.

Para satisfacer las necesidades técnicas y del negocio en las últimas décadas, los protocolos de red han ido evolucionando para ofrecer un mayor rendimiento, fiabilidad, conectividad, así como la más estricta seguridad. Estos protocolos han sido concebidos de manera aislada para resolver problemas puntuales, sin embargo, a dichas implementaciones le ha faltado una visión abstracta y global de la red. Esto ha dado lugar a una de las principales limitaciones de las redes de hoy en día: la complejidad. Por ejemplo, para agregar o eliminar cualquier dispositivo, los especialistas deben configurar y tener en cuenta múltiples *conmutadores*, *enrutadores*, *cortafuegoss*, portales Web de autenticación, la actualización de las listas de control de acceso, las redes de área local virtuales (VLAN, por sus siglas en inglés), políticas de calidad de servicio (QoS, por sus siglas en inglés), proveedores de los equipamientos, versiones de software y la topología de la red. Debido a esta complejidad, las redes actuales son relativamente estáticas, en su intento de reducir al mínimo el riesgo de interrupción del servicio. Existe un marcado contraste entre la naturaleza estática de las redes y el entorno de los servidores donde la virtualización ha traído consigo un aumento considerable de dispositivos que requieren conectividad en las redes así como un aumento de la movilidad terminal y personal.

Antes de la llegada de la virtualización, el tráfico en el centro de datos era norte-sur [cliente-servidor]. Ahora, cada vez hay más tráfico este-oeste, es decir, comunicación entre servidores, porque las aplicaciones están compartimentadas en muchas máquinas virtuales que deben comunicarse entre sí.

Las Máquinas virtuales migran para optimizar la carga de trabajo de los servidores haciendo que los flujos de datos cambien rápidamente. La migración de las máquinas virtuales trae muchos desafíos en las redes tradicionales que están diseñadas a partir de esquemas de direccionamiento segmentados. Además de la adopción de tecnologías de virtualización, muchas empresas hoy en día operan una red convergente IP de voz, datos y tráfico de video. Las redes existentes pueden ofrecer un nivel de calidad de servicio diferenciados para distintas aplicaciones, pero la provisión de esos recursos es manual. Al mismo tiempo se deben configurar los equipos de cada proveedor por separado, y ajustar parámetros como el ancho de banda y calidad de servicio en cada sesión o aplicación. Debido a su carácter estático, la red no puede adaptarse dinámicamente a los cambios de tráfico, aplicaciones y demandas de los usuarios.

- **Políticas inconsistentes**

Para implementar una política en toda la red, se tendría que configurar miles de dispositivos y mecanismos. Por ejemplo, cada vez que una nueva máquina virtual es creada, puede tardar horas, en algunos casos días, para que las listas de control de acceso (ACL, por sus siglas en inglés) sean reconfiguradas en toda la red. La complejidad de las redes de hoy en día hace que sea muy difícil aplicar un consistente conjunto de acceso, seguridad, calidad de servicio, y otras políticas que cada vez más usuarios móviles demandan, lo que deja a las redes vulnerables a violaciones de la seguridad, el incumplimiento de las regulaciones, y otras consecuencias negativas.

- **Incapacidad de escalabilidad**

Debido a que las demandas en los centros de datos crecen rápidamente, la red también crece. Sin embargo, la red se vuelve muy compleja con la adición de cientos o miles de dispositivos nuevos que deben ser configurados y gestionados. Cuando se diseña la red o un servicio, los enlaces se sobredimensionan a partir de patrones de tráficos conocidos para garantizar la escalabilidad de la red; sin embargo en los centros de datos virtuales de hoy en día los patrones de tráfico son dinámicos y por tanto impredecibles. Grandes operadores como Google, Yahoo!, Facebook continúan afrontando cada día desafíos relacionados con la escalabilidad. Estos proveedores de servicios emplean algoritmos de procesamiento paralelo de gran escala y conjuntos de datos a través de toda su infraestructura de red. Cuando un usuario realiza una búsqueda de información en Google el intercambio de información entre sus nodos puede alcanzar los peta byte (1000 Tera byte). Estas empresas necesitan implementar las llamadas redes híper escala que pueden proporcionar un alto rendimiento y conectividad entre los cientos de miles, o millones de servidores físicos con un bajo costo. Redes como las que fueron mencionadas anteriormente no se puede implementar manualmente. Otras funciones que aparentemente pudieran parecer muy sencillas como encaminar el tráfico de un cliente para brindarle un rendimiento personalizado o entrega bajo demanda son muy complejas de implementar en las redes actuales y sobre todo en la de los operadores debido a que se necesitan equipos especializados en el borde de la red.[12], [21]

- **La dependencia de proveedores**

Los operadores y las empresas constantemente tratan de desplegar nuevas prestaciones y servicios dando respuestas en muchas ocasiones a las demandas de los usuarios o estrategias para ser más competitivos. Sin embargo, la capacidad de respuesta de la empresa o del operador se ve limitada por los ciclos de producción de los vendedores que pueden superar los 3 años. La falta de interfaces abiertas en equipos limita la interconexión con otros proveedores al intentar adaptar la red a situaciones particulares. El ciclo de vida de los equipos trae consigo la constante renovación de dispositivos con el consiguiente costo económico para las organizaciones al adquirir nuevas tecnologías. La falta de correspondencia entre las necesidades del mercado y las capacidades de la red ha llevado a la industria de las ICT a un punto de inflexión. En respuesta, la industria ha creado el nuevo paradigma de red SDN para solucionar las limitaciones de las redes actuales y manejar con mayor eficiencia las siguientes tendencias:

- **El cambio de los patrones de tráfico:** Fenómenos como la virtualización y computación en la nube están cambiando radicalmente los patrones de tráfico dentro de los centros de datos. Antiguamente, el tráfico en el centro de datos era norte-sur [cliente-servidor]. Ahora, cada vez hay más tráfico este-oeste, es decir, comunicación entre servidores, porque las aplicaciones están compartimentadas en muchas máquinas virtuales que deben comunicarse entre sí [12], [22].
- **El consumo de las tecnologías de información (IT):** Los usuarios están empleando cada vez más dispositivos personales móviles, como Smartphone, tabletas y ordenadores portátiles para el acceso a la red. Las tecnologías de información se encuentran bajo presión para dar cabida a todos estos dispositivos personales de una manera adecuada y al mismo tiempo proteger los datos corporativos y la propiedad intelectual [12]. La tendencia BYOD (Bring Your Own Device) exige que a todo usuario se le permita conectarse a la red desde diferentes puntos de acceso y al mismo tiempo garantizando la seguridad y la protección de los datos.
- **El aumento de los servicios en la nube:** Las empresas han adoptado con entusiasmo los servicios de la nube, tanto públicos como privados, lo que resulta en un crecimiento sin precedentes de estos servicios. Siendo así que

las unidades de negocio empresariales ahora quieren mayor agilidad a la hora acceder a aplicaciones, infraestructura y otros recursos de computación [12].

- **Mayor demanda de ancho de banda "Big data":** Manejar la gran cantidad de datos de hoy en día requiere un procesamiento paralelo masivo en miles de servidores, todos de los cuales necesitan conexiones directas entre sí. El surgimiento de esta gran cantidad de información está impulsando una demanda constante de la capacidad de red adicional en los data centers. Los operadores de redes de data centers de híper escala se enfrentan a la desalentadora tarea de ampliar la red a un tamaño inimaginable [12].

1.2 Definición y beneficios de las SDN

Últimamente SDN se ha convertido en uno de los temas más populares en el ámbito de las ICT. Sin embargo, al ser un nuevo concepto, no se ha alcanzado aún un consenso sobre su definición exacta. De hecho, una gran variedad de diferentes definiciones [12], [23]–[30] han surgido en los últimos años, cada una de las cuales tiene sus propios méritos. En esta sección, se presentan en primer lugar una definición generalmente aceptada de las SDN, y luego se esbozan un conjunto de beneficios y desafíos de este tipo de redes, para finalmente presentar un modelo de referencia de las SDN.

La ONF (Open Networking Foundation) es un consorcio sin ánimo de lucro dedicada al desarrollo, estandarización y comercialización de las SDN. Esta organización es la que ha presentado la visión más aceptada de las SDN y está conformada por todas las grandes instituciones y empresas de las ICT.

El fundamento de la arquitectura SDN según la ONF (Open Networking Foundation) en [12], [29] es "*Las redes definidas por software se definen como una arquitectura de red dinámica, gestionable, adaptable, de costo eficiente. Lo cual la hace ideal para las altas demandas de ancho de banda y la naturaleza dinámica de las aplicaciones actuales. Esta arquitectura desacopla el control de la red y la funcionalidad de reenvío de información permitiendo que el control de la red pueda ser completamente programable logrando que las aplicaciones y servicios de red se abstraigan de la infraestructura de red subyacente*".

Las principales características de la arquitectura SDN son:

- Directamente Programable: El control de la red es directamente programable debido a que las funciones de reenvío fueron desacopladas [29].
- Ágil: Al tener un monitoreo constante sobre la red de telecomunicaciones, permite al administrador de red o a las aplicaciones hacer cambios dinámicos en los flujos de red según los requerimientos [29].
- Gestión centralizada: La inteligencia de la red está centralizada en un software llamado controlador SDN, que mantiene una visión global de la red, haciendo parecer una gran red de conmutadores como si fuese un único conmutador lógico [29].
- Configuración mediante la programación: SDN permite a los administradores de red configurar, administrar, asegurar y optimizar los recursos de red muy rápidamente a través de aplicaciones SDN dinámicas, que pueden ser escritas por ellos mismos, ya que los programas no dependen de software propietario [29].
- Basados en estándares abiertos y de proveedor neutral: Como se implementa a través de estándares abiertos, SDN simplifica el diseño de la red y la operación [29].

Se puede definir a SDN como una arquitectura de red con cuatro pilares fundamentales:

- Los planos de control y de datos se encuentran desacoplados. La Funcionalidad de control es eliminada de los dispositivos de red, los que se convertirán en simples elementos de reenvío de paquetes. Los reenvíos de paquetes se basan en flujos y no en el análisis del destino de cada datagrama. Un flujo puede ser definido por muchos criterios entre los que se encuentran: puerto donde se recibe el paquete, etiqueta VLAN, MAC origen o destino, protocolo, etc. Estos criterios actúan como filtro que permiten ejecutar un conjunto de acciones sobre ese tráfico. En el contexto SDN / OpenFlow, un flujo es una secuencia de paquetes entre un origen y un destino. Todos los paquetes de un flujo reciben políticas de servicios idénticas en los dispositivos de reenvío. La abstracción de flujo

permite unificar el comportamiento de los diferentes tipos de dispositivos de red, incluyendo enrutador, conmutador, cortafuegos, y middleboxes. La programación de flujo permite una flexibilidad sin precedentes, limitado sólo a las capacidades de las tablas en los dispositvos anteriormente mencionados.

- La lógica de control se mueve a una entidad externa, el llamado controlador SDN o sistema operativo de red (NOS, por sus siglas en inglés). El NOS es una plataforma de software que se ejecuta en tecnología de servidores y que proporciona los recursos esenciales y abstracciones para facilitar la programación de dispositivos de conmutación de paquetes basado en una vista de red abstracta lógicamente centralizada. Su propósito es, por tanto, similar a la de un sistema operativo tradicional.
- La red se puede programar a través de aplicaciones o programas que estén ejecutándose en la parte superior del NOS que interactúa con los dispositivos del plano de datos subyacentes. Esta es una característica fundamental de las SDN, considerado como su principal propuesta de valor.

Las SDN también permiten simplificar los dispositivos de red, pues ya estos no tienen que entender y procesar varios protocolos estándares, sino simplemente aceptar instrucciones de los controladores SDN. La arquitectura de SDN permite a los administradores de red tener un control central programable del tráfico en la red, sin necesidad de acceso físico a los dispositivos de hardware de la red.

En las redes actuales, la manera como se procesan los paquetes depende de la configuración manual de los nodos, mientras que en una SDN está condicionada por una interfaz de programación con un software que gobierna su comportamiento. La manera de procesar los paquetes no depende de una configuración estática en cada uno de los nodos, sino de los mensajes que envía un software a cada elemento de la red de manera dinámica. Ésa es la gran diferencia: la red ya no es determinista, sino dinámica [12], [22].

SDN sustituye el nivel de control del hardware de red por una capa de software abstraída mediante técnicas de virtualización, tratando de hacer la red más programable. Esto se concreta en distintas aproximaciones, como son: proporcionar un acceso al hardware basado en programación mediante protocolos como OpenFlow; arquitecturas construidas sobre un nivel de control basado en software; y crear redes virtuales por encima del hardware que dirijan el tráfico a través de redes físicas.

Hablar de SDN requiere hacer foco en varios conceptos ya mencionados. El primero es que las redes definidas por software permiten la separación entre el plano de control (el software) y el plano de datos (la maquinaria que encamina los paquetes) de una red, facilitando que ese control se gestione mediante un acceso basado en programación. De este modo, las compañías adquieren un control sobre la red independiente del proveedor. SDN simplifica también los propios dispositivos de red, puesto que ya no necesitan entender y procesar miles de protocolos, sino simplemente aceptar instrucciones de los controladores SDN.

El tercer concepto que hay que abordar es el de la virtualización. En concreto, la virtualización de las redes es una de las aplicaciones de SDN. Con SDN se virtualiza la red independizándola de la infraestructura física subyacente y creando redes lógicas con el objetivo de cumplir los requisitos de rendimiento, escalabilidad y agilidad necesarios en modelos de computación en la nube. Este nuevo paradigma permite la provisión programática de las redes virtuales, necesarias para asumir cargas de trabajo de forma dinámica, las migraciones de máquinas virtuales y cuestiones relacionadas con la escalabilidad de un centro de datos o incluso entre centros de datos geográficamente dispersos.

Según el IETF (Internet Engineering Task Force) y IRTF(Internet Research Task Force), *"las redes definidas por software es un nuevo paradigma relacionado con las redes programables y refiere que es la capacidad de que un software pueda programar y controlar el comportamiento de la red en todo su conjunto"* [28], [30].

Los beneficios de SDN son:

- Visión unificada de la estructura de la red: Con SDN se tiene una visión unificada de la red, simplificando la configuración, gestión y la provisión de la misma.
- Enrutamiento mejorado: Conocida la topología de la red a priori y la conectividad en los niveles 1 a 3, pueden establecerse políticas que mejoren el enrutamiento.

- Monitorización y alertas centralizadas: Permite tomar conciencia de la calidad del servicio prestado, latencia, pérdida de paquetes, ancho de banda por aplicación. Permite realizar estadísticas y establecer SLAs para disparar alertas basándose en toda la información de la red.
- Ingeniería de Tráfico (TE, por sus siglas en inglés) centralizada: Una ingeniería de tráfico centralizada proporciona una visión global, tanto en la oferta como en la demanda de recursos en la red. Gestionar a través de esta visión global los caminos de datos extremo a extremo, permite conseguir una alta utilización de los enlaces.
- Manejo más rápido de fallos: Los fallos, ya sean en un enlace o nodo, se identifican y gestionan más rápido con el control centralizado. Además, los sistemas convergen más rápidamente hacia el objetivo óptimo y el comportamiento es predecible.
- Entorno de prueba de alta fidelidad: La red dorsal (backbone) es emulada completamente mediante software. Es decir, puede recrearse fielmente en un entorno de pruebas. Esto no sólo ayuda en la comprobación y verificación, sino también en la gestión de posibles escenarios que se puedan dar.
- Actualizaciones sin impacto: El desacoplamiento del plano de control respecto al plano de datos, permite llevar a cabo actualizaciones de software sin pérdida de paquetes o degradación de la capacidad.
- Reducción de la complejidad a través de la automatización**:** La redes SDN ofrecen un marco de automatización y gestión de red flexible, lo que hace posible el desarrollo de herramientas que automatizan muchas tareas de administración que se realizan de forma manual hoy en día.
- Control centralizado de entornos de múltiples proveedores: El software de control SDN puede controlar cualquier dispositivo de red habilitado para OpenFlow de cualquier proveedor, incluyendo conmutadores, enrutadores y conmutadores virtuales. En lugar de tener que gestionar grupos de dispositivos de un solo vendedor[12].
- Mayor tasa de innovación: La adopción SDN acelera la innovación empresarial permitiendo a los operadores de red, literalmente programar y reprogramar la red en tiempo real para satisfacer las necesidades específicas de negocio y de los usuarios a medida que estas surgen[12].
- Aumento de la fiabilidad y seguridad de la red: SDN hace posible definir sentencias de configuración y de política de alto nivel, que luego son traducidas a la infraestructura a través de OpenFlow. Una Arquitectura SDN basada en OpenFlow elimina la necesidad de configurar individualmente los dispositivos de red cada vez que se realice un cambio en un punto final, servicio, o aplicación, lo que reduce la probabilidad de fallas en la red debido a configuraciones erróneas o políticas inconsistentes[12].
- Dado que los controladores SDN proporcionan una completa visibilidad y control sobre la red, pueden asegurarse de que el control de acceso, la ingeniería de tráfico, calidad de servicio, seguridad y otras políticas sean aplicadas consistentemente a través de las infraestructuras de redes cableadas e inalámbricas, incluyendo las empresas, los campus y los centros de datos.
- Mejor experiencia de usuario: Al centralizar el control de la red y hacer que el estado de la información esté disponible para las aplicaciones de alto nivel, una infraestructura SDN puede adaptarse mejor a las necesidades dinámicas del usuario[12].

1.3 Arquitectura global de las Redes Definidas por Software

En esta sección, se presenta una visión simplificada de la arquitectura de las SDN al mismo tiempo que se describen sus principales componentes. Según la ONF, SDN es una arquitectura emergente que desacopla las funciones de control de la red y el renvío de datos. Esto hace posible que el control de la red pueda ser programable y posibilita que las aplicaciones y servicios se abstraigan de la infraestructura de red subyacente. La mayor potencialidad de este nuevo concepto es que la red puede ser tratada como una entidad lógica o virtual. En las figuras 1.1 y 1.2 se ilustran la arquitectura lógica de las SDN.

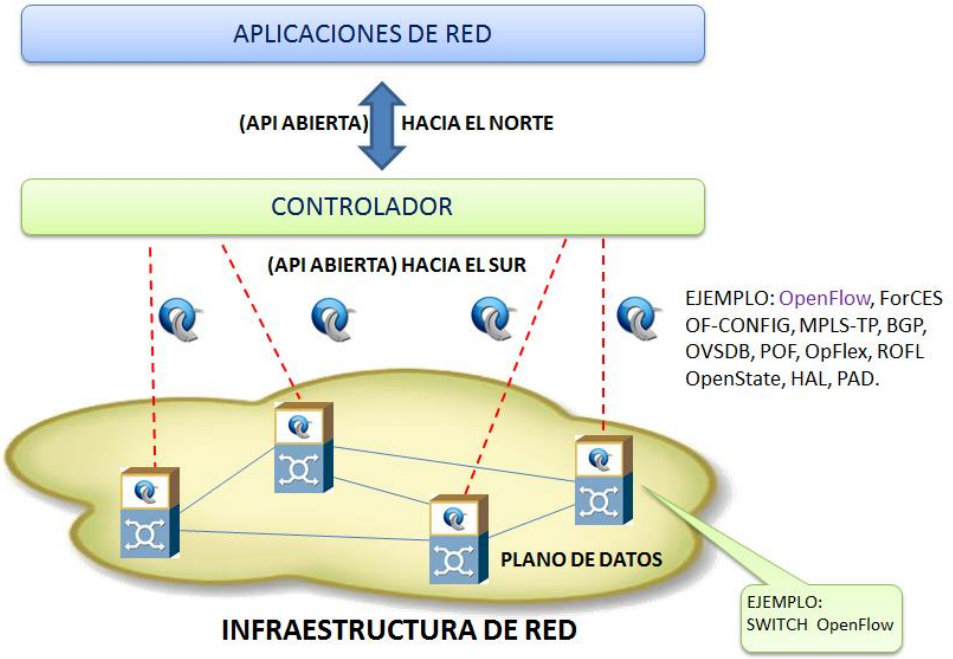


Figura 1.1. Visión simplificada de la arquitectura SDN.

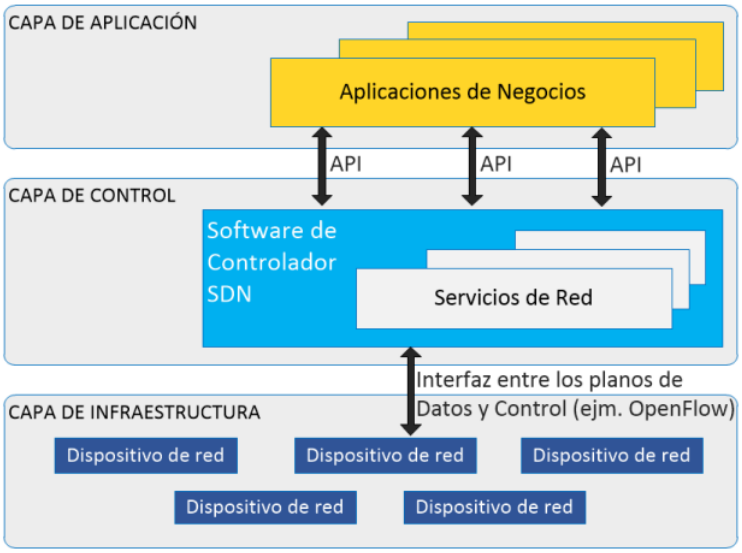


Figura 1.2. Arquitectura de las Redes Definidas por Software. Fuente: ONF, Open Networking Foundation.

En este tipo de arquitectura, los dispositivos de la capa de infraestructura se convierten en elementos especializados en el reenvío de paquetes que procesan los datagramas entrantes en función de un conjunto de reglas generadas por uno o varios controladores de la capa de control, de acuerdo a una lógica de programación predefinida en la capa de aplicación. El controlador usualmente se hospeda en un servidor remoto y se comunica mediante conexiones seguras con los dispositivos de la capa de infraestructura utilizando mandos estandarizados [2], [12].

SDN en su intento de solucionar las limitaciones de las infraestructuras de redes actuales en primer lugar rompe con la integración vertical de los modelos de red tradicionales al separar la lógica de control de la red de los *conmutadores y enrutadores* que encaminan y reenvían el tráfico [3]. En segundo lugar, con la separación de los planos de control y de datos, los *conmutadores* de la red se convierten en simples dispositivos de reenvío de paquetes y la lógica de control se implementa en un controlador lógicamente centralizado o un sistema operativo de red conocido como NOS; que permite simplificar la configuración, evolución y aplicación de políticas en la red. Es importante enfatizar que cuando se habla de

un modelo con control programático centralizado no se da por sentado que es físicamente centralizado. De hecho, garantizar un nivel adecuado de rendimiento, escalabilidad, y fiabilidad no pudieran implementarse con controles físicamente centralizados. Es por ello que los diseños de redes SDN actuales prevén planos de control físicamente distribuidos [26].

La arquitectura SDN está compuesta por 3 planos fundamentales:

- **Plano de infraestructura o de datos**: De manera similar a una red tradicional, el plano de infraestructura está compuesto por un conjunto de dispositivos, *conmutadores, enrutadores* interconectados entre sí y formando una gran red. La diferencia principal reside en el hecho de que los dispositivos físicos tradicionales son ahora elementos que se especializan en el reenvío de paquetes y no poseen un control o software embebido para tomar decisiones de forma autónoma. La inteligencia de la red ya no está en los dispositivos del plano de datos y pasa a un control lógicamente centralizado llamado NOS [3], [12].
- **Plano de control**: Es una entidad lógica centralizada que tiene la función de convertir las solicitudes de las aplicaciones SDN en ordenes hacia los dispositivos de las capas inferiores y así proporcionarles a las aplicaciones SDN una visión abstracta de la red. Es el responsable de tomar la decisión sobre como los paquetes deberán ser reenviados por uno o más elementos de red así como de la programación de los nodos de la red para implementar la decisión tomada. Tiene la función de mantener actualizada las tablas de reenvío de los dispositivos del plano datos o infraestructura basado en la topología de la red o solicitudes de servicios externos. Por medio de este plano, se crea una vista centralizada de la topología de la red de datos, la cual permite gestionar el flujo de datos en el plano de infraestructura por medio de protocolos estándares, también brinda una API para que desde la capa de aplicación se puedan programar flujos y retroalimentar a la aplicación, con información de la topología de red o el tráfico de paquetes.
- **Plano de aplicación**: Permite crear aplicaciones para automatizar tareas de configuración, provisión y despliegue de nuevos servicios en la red. Las aplicaciones de esta capa pueden considerarse el cerebro de la red debido a ellas implementan la lógica de control que se traduce en mandos que son enviados al plano de datos para dictar el comportamiento de los nodos especializados en el reenvío de paquetes. Las aplicaciones pueden definir la ruta a través de la cual los paquetes fluyen de un punto a otro de la red.

1.4 Arquitectura SDN y sus abstracciones fundamentales

Según se analizó anteriormente las SDN están compuesta por tres planos fundamentales. En la figura 1.4 (b) se muestran además las capas por las que están compuesta la arquitectura. Cada capa tiene sus propias funciones específicas y no necesariamente todas tienen que existir en implementaciones de redes SDN. Las capas hacia el sur (*southbound* API) y hacia el norte (*northbound* API)*, el* NOS*,* y la de aplicaciones de red siempre van a estar presentes en despliegues SDN.

A continuación se presenta siguiendo un enfoque de abajo hacia arriba los conceptos, definiciones y propiedades de cada capa basados en diferentes soluciones y tecnologías.

Capa I: Infraestructura

Una infraestructura SDN de manera similar a las redes tradicionales está compuesta por un conjunto de *conmutadores, enrutadores* interconectados entre sí y formando una red. Los nodos se interconectan utilizando diferentes medios de trasmisión tales como cables de cobre, fibras ópticas, acceso inalámbrico, etc. [19]. La principal diferencia reside en el hecho de que esos elementos ahora se especializan solamente en el reenvío de los paquetes que ingresan a cada uno de los dispositivos y no poseen un subsistema de control o software para tomar decisiones autónomamente. En la figura 1.4 (c) se puede observar como la inteligencia de la red es eliminada de los dispositivos de la capa de infraestructura hacia un sistema de control lógicamente centralizado. Más importante aún, estas redes se construyen conceptualmente sobre la base de interfaces estándares y abiertas (por ejemplo OpenFlow) para garantizar la configuración, interoperabilidad, compatibilidad en la comunicación entre los planos de datos y de control. En otras palabras, estas interfaces abiertas le permiten a los controladores programar dinámicamente los dispositivos especializados en el reenvío de datos; cuestión

que es difícil implementar en las redes tradicionales debido a la gran variedad de interfaces propietarias, cerradas y el control distribuido [6], [8]–[11], [35]–[39].

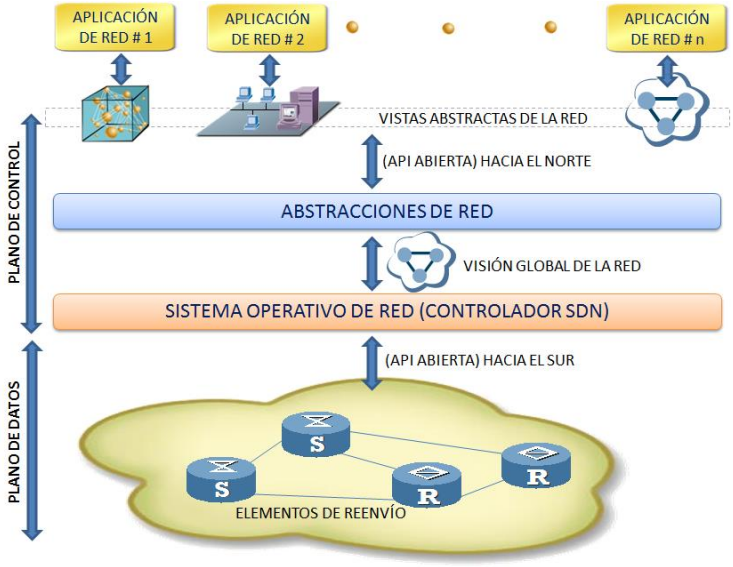


Figura 1.3. Arquitectura SDN y sus abstracciones fundamentales. Fuente: Autor.

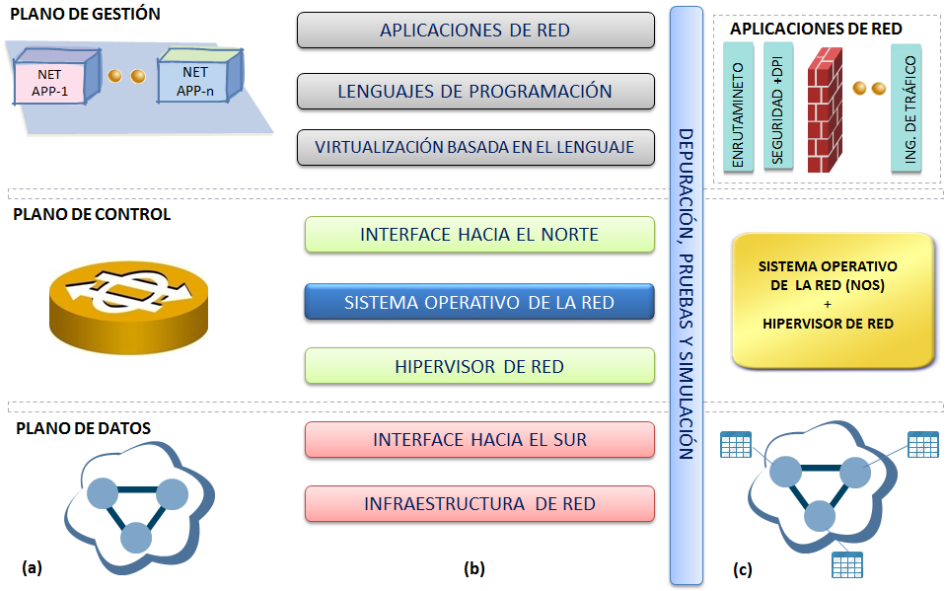


Figura 1.4. Las SDN en (a) Planos, (b) Capas, and (c) Arquitectura de diseño. Fuente: Autor

En una arquitectura SDN/OpenFlow existen dos elementos principales que son el controlador y los dispositivos de reenvío. Los dispositivos del plano de datos son elementos de hardware o software que se especializan en el reenvío de paquetes, mientras que el controlador es el cerebro de la red que se ejecuta en una plataforma de hardware.

Un dispositivo de la capa de infraestructura que soporte el protocolo OpenFlow posee una o varias tablas de flujo según la versión del protocolo OpenFlow. Cada registro o entrada de flujo de la tabla consta de tres partes fundamentales. La primera son los campos de coincidencia que es una regla para (encaminar, descartar, etc.) un tipo específico de tráfico después de analizar la coincidencia del tráfico de entrada con la regla en cuestión. La segunda son las acciones que serán aplicadas sobre el tráfico después de encontrada una coincidencia en la tabla de flujo entre las que se destacan: el reenvío de paquetes a puerto(s) a la velocidad de línea, descartar el tráfico o encapsularlo y enviarlo al controlador para su posterior análisis. La tercera es la recopilación de estadísticas de las tablas, los flujos, puertos y colas. Las estadísticas podrán ser utilizadas por las aplicaciones SDN para tomar decisiones en tiempo real en cuanto al comportamiento de la red.

Dentro de los dispositivos OpenFlow, un camino a través de una secuencia de tablas de flujo define como será manejado el tráfico. Cuando llega un nuevo paquete se inicia un proceso de búsqueda en la primera tabla de flujo y termina con una coincidencia en una de las tablas o es descartado o enviado al controlador en caso de no encontrar ninguna regla para procesar ese tipo de tráfico. Una regla de flujo puede ser definida a partir de la combinación de diferentes tipos de campos coincidentes. Usualmente se instala en estos dispositivos una regla predeterminada que envía los paquetes al controlador o procesamiento tradicional (no OpenFlow) en el conmutador cuando no se encuentra ninguna coincidencia. Todas las reglas son instaladas por una aplicación SDN a través de la API hacia el norte con el controlador que es quien finalmente le envía información (reglas y acciones) al elemento de la capa de infraestructura utilizando la interface hacia el sur.

Capa II: Interfaz hacia el sur (southbound interfaces)

Las interfaces hacia el sur (APIs hacia el sur) son los puentes de conexión entre los elementos de control los elementos de reenvío, y son por tanto, el instrumento fundamental para separar las funciones de control de las del plano de datos. Sin embargo, estas APIs están todavía fuertemente ligadas a los elementos de reenvío de la infraestructura física o virtual subyacente.

Por lo general la comercialización de un conmutador que es creado desde cero puede tardar alrededor de dos años. Los ciclos de actualización de software de estos dispositivos pueden también tardar hasta nueve meses [15]. El desarrollo de software de un nuevo producto puede tomar de seis meses hasta un año. Usualmente la inversión inicial en la creación de estos dispositivos es alta y arriesgada y las APIs hacia el sur son el componente central a la hora de diseñarlos. Es por ello que esta capa (interface hacia el sur) representa una de las principales barreras para la introducción y aceptación de cualquier nueva tecnología de red. En ese sentido desde el surgimiento del protocolo OpenFlow la industria acogió este nuevo paradigma de manera inmediata debido a que el estándar fomenta la interoperabilidad y el despliegue de dispositivos de red de múltiples proveedores. La interoperabilidad de equipos de diferentes proveedores utilizando el protocolo OpenFlow ha sido probada [3].

En el momento de escribir esta memoria el protocolo OpenFlow ha sido el estándar abierto hacia el sur más aceptado e implementado para las SDN [2]. El protocolo proporciona una especificación para implementar los dispositivos de reenvío así como el canal de comunicación entre los dispositivos de los planos de datos y control (por ejemplo entre conmutadores y controladores). OpenFlow ofrece tres fuentes de información para los sistemas operativos de red, en primer lugar el envío por parte de los dispositivos de reenvío de mensajes de eventos cuando cambia el estado de un enlace o puerto. En segundo lugar el envío de estadísticas de flujo al controlador. El tercero y más importante son los mensajes “PACKET_IN” enviados por los dispositivos de la capa de infraestructura hacia el controlador cuando estos elementos no saben que hacer con los paquetes entrantes. Estos canales de información son la vía principal mediante la cual el NOS recibe información de la red.

Uno de los protocolos más conocidos y utilizados al implementar la interfaz hacia el sur entre un controlador SDN y la infraestructura de red es el protocolo OpenFlow. Aunque este protocolo es el más visible dentro de la comunidad de SDN, existen otras APIs con el propósito de implementar la interfaz hacia el sur entre las que se destacan:

- NETCONF[35]: NETCONF y OpenFlow pueden proporcionar una vía de comunicación entre dispositivos y el controlador. NETCONF es un protocolo de configuración, mientras que OpenFlow opera sólo en las tablas de flujos que determinan como los paquetes serán encaminados en la red. Los conmutadores OpenFlow se configuran utilizando OF-CONFIG que a su vez utiliza NETCONF para comunicarse con dispositivos. NETCONF es un protocolo de gestión de red del IETF. Proporciona a un administrador o ingeniero de red de una forma segura configurar un cortafuego, conmutador, enrutador o cualquier otro dispositivo de red. Se basa en llamadas de procedimiento remoto (RPC) y fue diseñado para resolver los problemas que existen con los protocolos SNMP y la Interfaz de Comandos, referente a la configuración de dispositivos de red. La Fundación Open Networking (ONF) adoptó recientemente NETCONF y lo hizo obligatorio para la configuración de dispositivos compatibles con OpenFlow. La especificación, llamada OF-CONFIG, requiere que los dispositivos que la soportan deben implementar el protocolo NETCONF como el transporte. El protocolo NETCONF se adapta a cualquier arquitectura a través de un conjunto de contenidos opcionales, y los desarrolladores pueden crear capacidades adicionales para que dispositivos NETCONF puedan tener características propietarias.

- Perfil de Transporte MPLS (MPLS-TP) [36] es el perfil de transporte para la conmutación de etiquetas multiprotocolo. MPLS-TP está diseñado para ser utilizado como una tecnología de capa de red en redes de transporte. Las extensiones del protocolo MPLS están siendo diseñadas por el IETF basado en los requerimientos proporcionados por los proveedores de servicios. El protocolo será una aplicación de conmutación de paquetes orientado a la conexión (CO-PS) que ofrece una implementación MPLS dedicada a eliminar características que no son relevantes a las aplicaciones CO-PS y a agregar dispositivos que proporcionan soporte a funcionalidades críticas del transporte. La ONF propuso cambios a MPLS que incluyen el uso del plano de datos estándar de MPLS con un plano de control más simple basado en SDN y OpenFlow. Al contar con un plano de control simplificado que se desacopla del plano de datos, la ONF argumenta que es capaz de optimizar los servicios globalmente, al mismo tiempo que los hace más dinámicos. y crear nuevos servicios tales como ancho de banda en demanda al programar aplicaciones de red sobre el controlador SDN.
- El protocolo BGP [37]: BGP es un protocolo conocido de enrutamiento de Internet que algunos proveedores esperan utilizar en redes definidas por *software* de híbridas. Se define en RFC 1771, permite crear encaminamientos entre dominios sin bucles entre sistemas autónomos (AS). El controlador SDN utiliza BGP como protocolo del plano de control y maneja los dispositivos de la capa de infraestructura utilizando NETCONF.
- El Protocolo extensible de mensajería y presencia (XMPP): Es un protocolo abierto y extensible basado en XML, originalmente ideado para mensajería instantánea. Es usado además en una amplia gama de aplicaciones de mensajería de voz, video y presencia. XMMP ha surgido recientemente como un protocolo alternativo a OpenFlow para implementaciones de redes SDN hibridas.
- Protocolo de Gestión de Base de Datos Open vSwitch (OVSDB) es un protocolo de configuración de OpenFlow que está destinado a administrar las implementaciones Open vSwitch. Open vSwitch es un conmutador virtual que permite la automatización de la red y el soporte de interfaces y protocolos de administración estándar, tales como NetFlow. El protocolo también soporta la distribución a través de múltiples servidores físicos. En una implementación Open vSwitch, un grupo de control contiene administradores y controladores que utilizan el protocolo OVSDB para suministrar información de configuración al servidor de base de datos del conmutador. Los controladores usan OpenFlow para especificar los detalles de los flujos de paquetes en el conmutador. Cada controlador y gestor pueden manejar varios conmutadores, y cada conmutador puede recibir directivas de varios gestores y controladores.

Capa III: Hipervisores de red

La virtualización es ya una tecnología consolidada en la computación moderna. Los rápidos avances de la última década han hecho que la virtualización sea una nueva tendencia dominante. Reportes recientes revelan que el número de servidores virtuales han superado la cantidad de servidores físicos [3].

El hipervisor, también llamado monitor de máquina virtual (VMM), es el núcleo central de algunas de las tecnologías de virtualización de hardware más populares y eficaces. Inicialmente han sido desarrollados en el área de computación virtual para monitorear máquinas virtuales. Múltiples máquinas virtuales pueden funcionar en una plataforma de cómputo determinada. Con la virtualización, múltiples máquinas virtuales, cada una con su propio sistema operativo huésped, pueden correr en un (hardware físico) o plataforma de cómputo. Además del monitoreo de las máquinas virtuales, el hipervisor asigna recursos de la plataforma de computo física, por ejemplo, ciclos de ejecución en las unidades centrales de procesamiento (CPU), a las máquinas virtuales individuales. El hipervisor normalmente implementa una abstracción de la plataforma de cómputo física utilizando un conjunto de instrucciones estándar, para la interconexión con la misma [34].

Análogamente, las redes virtuales (ver figura 1.5) han surgido recientemente para permitir una flexibilidad en los nuevos servicios de red sin preocuparse de los detalles específicos de la red física subyacente. Además, a través de la virtualización, múltiples redes virtuales pueden operar con flexibilidad sobre la misma infraestructura de red física. En el contexto de las redes virtuales, un hipervisor supervisa las redes virtuales y asigna los recursos de red, como por ejemplo, la capacidad del enlace y buffer en nodos de conmutación, a las redes virtuales individuales [34].

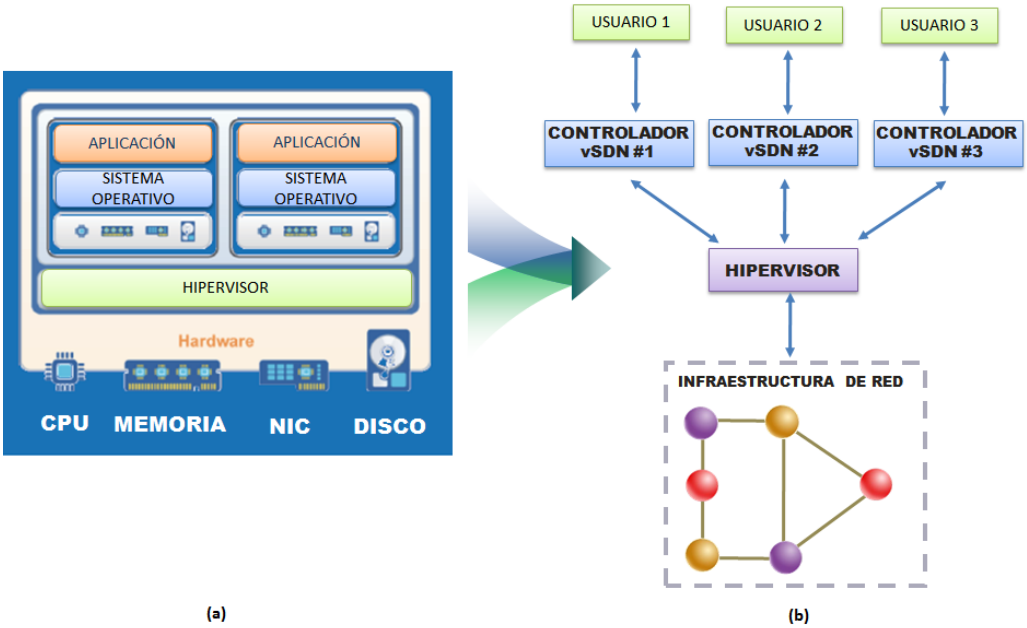


Figura 1.5. Hipervisores de red para la virtualización de las SDN. Fuente: Autor

Capa IV: Sistema operativo de red, "Controladores"

Los sistemas operativos tradicionales proporcionan abstracciones (por ejemplo, API) para acceder a los dispositivos de nivel inferior, gestionar los accesos concurrentes a los recursos subyacentes (por ejemplo, disco duro, adaptador de red, CPU, memoria), así como proporcionar mecanismos de protección y seguridad. Estas funcionalidades y recursos son claves para aumentar la productividad, así como facilitar la creación y desarrollo de nuevas aplicaciones. Su uso generalizado ha contribuido de manera significativa a la evolución de los diversos ecosistemas (por ejemplo, los lenguajes de programación) y el desarrollo de una gran variedad de aplicaciones.

Por el contrario, las redes han sido hasta ahora gestionadas y configuradas utilizando el nivel de abstracciones más bajo, mediante el empleo de instrucciones específicas de cada dispositivo así como sistemas operativos de red propietarios como por ejemplo el IOS de Cisco y el JunOS de Juniper. Este tipo de redes carecen de ofrecer una abstracción de las características de los dispositivos; y no proveen funcionalidades comunes. Es por ello que hoy en día los diseñadores de protocolos de enrutamiento tienen que hacer frente a complicados algoritmos distribuidos para solucionar los problemas de una red.

Se espera que SDN facilite la gestión y simplifique las soluciones de los problemas de red mediante la utilización de una lógica de control centralizada llamada sistema operativo de red. Al igual que con los sistemas operativos tradicionales, el valor crucial de un NOS, es proporcionar abstracciones, servicios esenciales, y APIs comunes para los desarrolladores. Funcionalidades genéricas tales como las de la obtener información del estado y topología de la red, la detección de dispositivos, así como la distribución de configuración a los dispositivos pueden ser servicios proporcionados por el NOS. Con los NOS, al definir las políticas de una red, un desarrollador ya no tendrá que preocuparse por los detalles de bajo nivel en los dispositivos del plano de datos; cuestión que acelerará la innovación y reducirá la complejidad de la creación de nuevos servicios y aplicaciones de red.

El NOS es un elemento crucial de la arquitectura SDN, ya que es la pieza fundamental que soporta la lógica de control (las aplicaciones SDN) quienes permiten generar una configuración de red en base a las políticas definidas por el operador de la misma. Al igual que en un sistema operativo tradicional, la plataforma de control abstrae los detalles de bajo nivel relacionados con la conexión y la interacción con dispositivos de reenvío.

Las funciones de un controlador:

Los controladores SDN deben brindar un grupo de servicios de red básicos que son considerados funcionalidades esenciales. Haciendo una analogía, estas funciones son como los servicios básicos de los sistemas operativos tales como: ejecución de programas, control de las operaciones de entrada y salida, comunicaciones, protección, etc. Estos servicios son utilizados por otros servicios a nivel de sistema operativo y aplicaciones de usuario. De manera similar, las funciones tales como la topología, estadísticas, notificaciones, y la gestión de dispositivos, junto con mecanismos de reenvío y selección de rutas más cortas así como la seguridad, son funcionalidades de control esenciales que las aplicaciones de red pueden utilizar en la construcción de su lógica. Por ejemplo, el gestor de notificaciones debe ser capaz de recibir, procesar y reenviar eventos (por ejemplo, notificaciones de alarma, alarmas de seguridad, cambios de estado) [38]. Los mecanismos de seguridad son otro ejemplo, debido a que ellos son los componentes críticos a la hora de proporcionar un aislamiento y seguridad entre los servicios y las aplicaciones.

En el nivel inferior de las plataformas de control, las APIs hacia el sur se pueden ver como una capa controladora de dispositivos. Esta capa proporciona una interfaz común para las capas superiores, permitiendo que la plataforma de control utilice diferentes APIs hacia el sur (por ejemplo, OpenFlow, OVSDB, y ForCES) u otros protocolos (por ejemplo, SNMP, BGP y NETCONF) que permitan gestionar dispositivos físicos o virtuales. Esto es esencial tanto para la compatibilidad con versiones anteriores así como la heterogeneidad, es decir, permitir varios protocolos y conexiones entre dispositivos. Por lo tanto, en el plano de datos, existe una mezcla de dispositivos físicos, virtuales (por ejemplo, vSwitch, vrouter) y una variedad de interfaces (por ejemplo, OpenFlow, OVSDB, OF-CONFIG, NETCONF, y SNMP) que pueden coexistir.

La mayoría de los controladores sólo son compatibles con OpenFlow. Aun así, algunos de ellos, como OpenDaylight, Onix, y HPVANSDN, ofrecen una gama más amplia de APIs hacia el sur y / o plugins de protocolos. OpenDayLight va un paso más allá al proporcionar una capa de abstracción de servicio (SLA) que permite que varias APIs y protocolos en dirección hacia el sur coexistan en la plataforma de control. Su arquitectura original fue diseñada para soportar al menos siete protocolos diferentes y *plugins*: OpenFlow, OVSDB, NETCONF, PCEP, SNMP, BGP, y LISP. Por lo tanto, OpenDayLight es una de las pocas plataformas de control que están concebidas para soportar una mayor integración de las tecnologías en una única plataforma de control.

En una red con controladores distribuidos, como se muestra en la figura 1.6, se necesitan APIs de comunicación entre los elementos de control. En la actualidad, cada controlador implementa su propia API hacia al este / oeste. Las funciones de estas interfaces incluyen importar / exportar los datos entre los controladores, algoritmos para garantizar la consistencia de datos, así como capacidades de supervisión / notificación (por ejemplo, comprobar si un controlador está en servicio o notificar la conmutación hacia otro controlador a un conjunto de dispositivos de la capa de infraestructura).

De manera similar a las interfaces norte y sur, las API hacia el este / oeste son componentes esenciales de los controladores distribuidos. Con el objetivo de identificar y proporcionar compatibilidad e interoperabilidad entre los diferentes controladores, es necesario disponer de interfaces hacia el este/oeste estándar. Por ejemplo SDNi [39], [40] define los requerimientos para coordinar la configuración de flujo e intercambio de información de accesibilidad a través de múltiples dominios. En esencia, estos protocolos se pueden utilizar en una forma orquestada e interoperable para crear plataformas de control distribuido escalables y fiables. La interoperabilidad se puede aprovechar para aumentar la diversidad del elemento de plataforma de control. De hecho, la diversidad aumenta la robustez del sistema mediante la reducción de la probabilidad de fallos comunes, tales como fallos de software [41].

Entre las propuestas que tratan de definir interfaces entre los controladores figuran las funciones de importación / exportación de datos Onix [42], la interface CE-CE de ForCES [43], los mecanismos de respaldo de ForCES para lograr alta disponibilidad; así como el almacenamiento de datos distribuidos. Una API este / oeste requiere mecanismos avanzados de distribución de datos, tales como el protocolo de cola de mensajes de avanzado (AMQP, advanced message queuing protocol) [44] utilizado por DISCO [45], técnicas para la creación de políticas distribuidas simultáneas y coherentes [46], bases de datos de transaccionales y DHTs [216] (utilizadas en Onix[42]), o algoritmos avanzados para la consistencia fuerte y tolerancia a fallos [47].

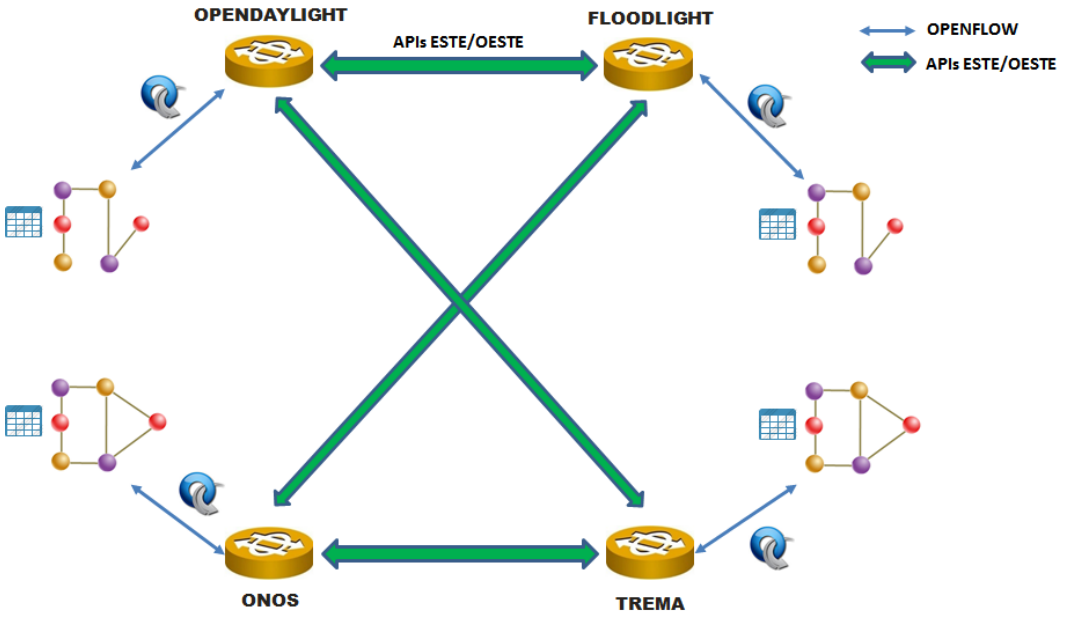


Figura 1.6. Controladores distribuidos: APIs Este/Oeste. Fuente: Autor.

Capa V: Las interfaces hacia el norte. (NBI, Northbound Interfaces)

La proliferación de Controladores SDN cada uno con APIs únicas ha creado crea una variedad de interfaces de programación que los proveedores de servicios de red, sistemas de orquestación y los proveedores de aplicaciones deben desarrollar para el fin de servir a diversos casos de uso SDN. Esto inhibe la adopción generalizada de SDN así como sus protocolos y tecnologías asociadas [48].

La interface hacia el norte desempeña un papel crucial en la promoción de la adopción de SDN ya que permite a los desarrolladores la libertad de desplegar sus aplicaciones para generar ingresos sin ser afectados y limitado por la complejidad de las redes subyacentes. Para ello, la NBI tiene que permitir que las aplicaciones puedan expresar sus necesidades y limitaciones en el lenguaje de programación específico de la aplicación, y el controlador SDN debe traducir esos requisitos en el lenguaje empleado en la capa de infraestructura para realizar la provisión de recursos y servicios para así satisfacer los requisitos de la aplicación [48], [49]. La ONF proporciona todos los principios y directrices para la construcción de este tipo de interface [3], [50].

A diferencia de las API hacia el sur, la interface NBI permite la comunicación entre los componentes de más alto nivel. Mientras que en las redes tradicionales se utilizan cortafuegos o determinadas configuraciones en los enrutadores para lograr un balance de carga en la red, SDN instala aplicaciones que emplean el controlador para realizar esas mismas funciones. La comunicación entre dicha aplicación y el controlador se logra utilizando una interface NBI.

Las interfaces de hacia el norte y el sur son dos abstracciones claves de las SDN. La interface hacia el sur ya tiene una propuesta ampliamente aceptada (OpenFlow) que tiene el objetivo de controlar el comportamiento de la red subyacente. El controlador SDN también proporciona una interfaz programable para que las aplicaciones puedan observar y cambiar el estado de red dinámicamente. Esta interface se llama API hacia el norte (NBI, North Bound Interface) [51] y actualmente sigue siendo un tema de discusión abierto. Para explotar todo el potencial de SDN es vital lograr una abstracción a nivel de aplicación SDN mediante el empleo de interfaces estándar. En este tipo de escenarios usualmente primero aparecen algunas aplicaciones prácticas y luego surgen los estándar. Sin embargo, el futuro de las SDN necesitan con urgencia las primeras propuestas de interfaces hacia el sur. Las discusiones sobre este tema ya han comenzado [52]–[54], y existe un consenso común en que las API hacia el norte son realmente importantes, pero es demasiado pronto para definir un estándar único en este momento.

Una red es un sistema distribuido complejo cuya estructura, función, recursos y comportamiento pueden cambiar dinámicamente. Como las estructura de las redes evolucionan las interfaces hacia el norte deben ser lo suficientemente

flexibles como para ajustarse a estos cambios. En este sentido, la interface hacia el norte RESTful[52]–[54] se ha convertido en una opción predominante en SDN, donde una API REST bien diseñada puede ser altamente extensible y fácil de mantener para la gestión de recursos distribuidos, como lo son las redes de datos [52]. Interfaces abiertas y estándar hacia el norte son cruciales para promover la portabilidad de las aplicaciones y la interoperabilidad entre las diferentes plataformas de control. Una API hacia el norte puede ser comparada con el estándar NOSIX [55] en los sistemas operativos, que representa una abstracción que garantiza una independencia en los lenguaje de programación así como el controlador.

Una API abierta en dirección hacia el norte, permite que el ecosistema de aplicaciones de red, que es la principal promesa de las SDN, según Rob Sherwood, arquitecto principal de “Big Switch Networks” y presidente del grupo de trabajo de arquitectura de la ONF: "Sin una API hacia el norte, todas las aplicaciones de red deben provenir directamente de los proveedores de equipos, lo que hace que sea más difícil la innovación en su red”. Piense detenidamente en una tienda de aplicaciones de red. No pudiera existir una tienda de este tipo sin una API en dirección norte, dice Sherwood. Los operadores de red también pueden modificar o personalizar rápidamente el control de la red a través una API en dirección norte [56].

Existen APIs que normalmente se consideran parte del propio controlador y son utilizadas por un programa de nivel superior para crear servicios de red, que su vez están expuestos "hacia el norte Norte". También hay API construidas por encima de estas API. Ambos escenarios son considerados interfaces hacia el norte [48].

Los controladores Floodlight, Trema, NOX, Onix, y OpenDayLight proponen y definen sus propias APIs hacia el norte. Sin embargo, cada uno de ellos tiene sus definiciones específicas. Los lenguajes de programación Frenetic, Nettle, NetCore, Procera, Pyretic, y NetKAT también abstraen los detalles interiores de la funciones del controlador y el plano de datos de los desarrolladores de aplicaciones [3]. Por otra parte, tal como se explicará, mas adelante, los lenguajes de programación pueden proporcionar una amplia gama de abstracciones y mecanismos tales como la composición de aplicaciones, transparente así como la tolerancia a fallos en el plano de datos [2], [16], [57].

El concepto de interface hacia el norte tradicional que fue abordado anteriormente es diseñado por expertos de redes y a partir de una visión global de la red. Las capacidades de la red son abstraídas y encapsuladas de abajo hacia arriba. Este tipo de diseño no se ocupa de las necesidades del lado del usuario pero expone todas las capacidades e información de la red. De esta manera, las aplicaciones pueden obtener la máxima capacidad de programación de toda la red a través de la NBI. Los ejemplos de funcionalidades básicas de la NBI incluyen: descubrimiento de enlaces y dispositivos, asignación de identificadores para interfaces de red, la configuración de reglas de reenvío, y la gestión de decenas de miles de información del estado de la red. Los desarrolladores de aplicaciones SDN que utilizan las interfaces hacia el norte creadas hasta el momento, necesitan están familiarizados con conceptos de redes y configuraciones complejas a pesar de existir una abstracción de la infraestructura de la red. Los desarrolladores de aplicaciones SDN quieren centrarse en la funcionalidad de la aplicación y en la interacción amigable con los usuarios de la misma. Se requiere entonces un servicio de red fácil de utilizar, automático como los servicios en la nube. En este contexto aparece un nuevo concepto llamado NBI intencionada (Intent NBI) que tiene el fin de simplificar la interacción entre la aplicación, los servicios de red y el controlador. Este nuevo enfoque (ver figura 1.7) es totalmente independiente de la tecnología de red utilizada, a la vez que proporciona términos y un vocabulario a las aplicaciones, de manera que los usuarios no vean VPN, MPLS, y cualquier otro protocolo de enrutamiento. Con una NBI intencionada se abstraen los objetos y las capacidades de la red, a la vez que se expresa la intención del usuario, sobre qué hacer, de forma declarativa, pero no cómo hacerlo. En este escenario el controlador SDN es una "caja negra" que asigna y gestiona recursos de red [58].

Finalmente, es probable que exista más de una interface NBI debido a que los requerimientos de las aplicaciones de red son diferentes. Las API para aplicaciones de seguridad es probable que sean diferentes de las aplicaciones financieras o de enrutamiento. Por tal motivo la ONF ha creado un grupo de trabajo llamado (NBI-WG, North Bound Interface Working Group) con el objetivo de definir y posteriormente estandarizar las APIs hacia el norte de las SDN.

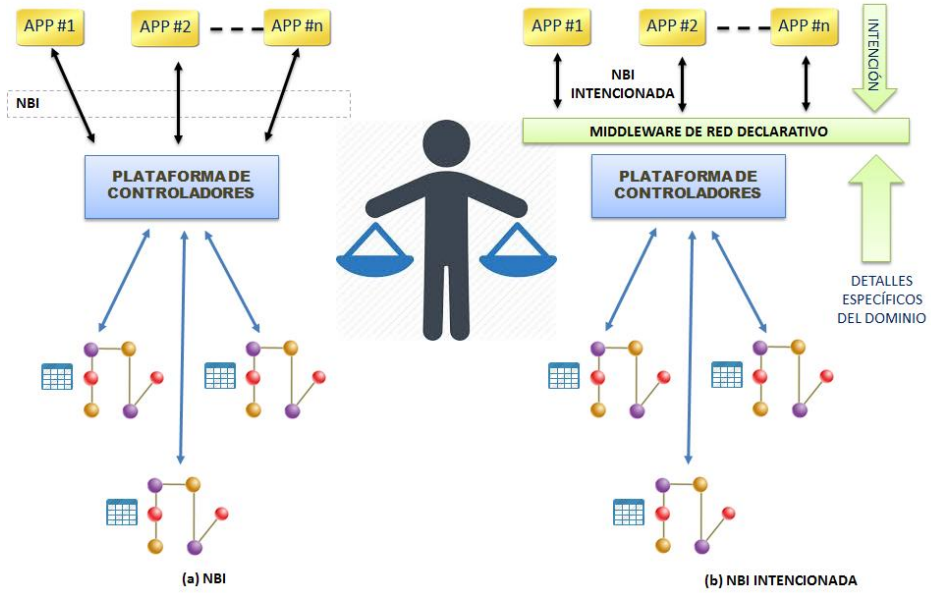


Figura 1.7. API EN DIRECCIÓN AL NORTE. NBI INTENCIONADA. Fuente: Autor.

Capa VI: La virtualización basada en el lenguaje

Dos características esenciales de las soluciones de virtualización son la capacidad de expresar modularidad así como permitir diferentes niveles de abstracción al mismo tiempo que garantiza propiedades deseadas, tales como la protección. Por ejemplo, las técnicas de virtualización pueden ofrecer diferentes puntos de vista de una única infraestructura física. Esto intrínsecamente simplifica la tarea de los desarrolladores de aplicaciones, ya que no tienen que pensar en las acciones que deben ser realizadas para instalar las reglas de flujos en los dispositivos de la capa de infraestructura y por lo tanto ver la red como si fuera un conmutador de gran dimensión. Ese tipo de abstracción simplifica significativamente el desarrollo y despliegue de aplicaciones de red complejas, como son los servicios relacionados con la seguridad. A modo de ejemplo, un "gran conmutador" virtual podría representar una combinación de varios dispositivos en la capa de infraestructura subyacente.

Pyretic [59]–[62] es un ejemplo interesante de un lenguaje de programación que ofrece un tipo de abstracción de alto nivel de la topología de la red. Este incorpora el concepto de abstracción mediante la introducción de objetos de red. Estos objetos consisten en una topología de red abstracta y los conjuntos de políticas que se le aplican a este. Los objetos de red ocultan simultáneamente información y ofrecen los servicios necesarios.

Otra forma de virtualización basada en el lenguaje es el corte estático de la red. En este la red es rebanada por un compilador, en base a las definiciones de capa de aplicación. La salida del compilador es un programa de control monolítico que ya ha segmentado las definiciones y los comandos de configuración para la red. En tales casos, no es necesario un que hipervisor que gestione de forma dinámica las rebanadas de la red. Un ejemplo de enfoque de corte estático es el "*Splendid isolation*" [63].

Otras soluciones, como libNetVirt[64], tratan de integrar tecnologías heterogéneas para la creación de rebanadas de red estáticas. LibNetVirt es una biblioteca diseñada para proporcionar de una manera flexible la creación y gestión de redes virtuales en diferentes entornos informáticos. Su idea principal es muy similar al proyecto Quantum de OpenStack [65].Mientras que Quantum está diseñado por OpenStack para entornos en la nube, Libvirt es una biblioteca de propósitos más generales que puede ser utilizada en diferentes ambientes. La biblioteca libNetVirt tiene dos capas: una interfaz de red genérica y controladores de dispositivo específicos de tecnología (por ejemplo, VPN, MPLS, OpenFlow).

Capa VII: Lenguajes de programación

Los lenguajes de programación han estado proliferando durante décadas. La academia y la industria han evolucionado a partir de lenguajes de máquina de bajo nivel para hardware específicos, tales como ensamblador para las arquitecturas

x86, a lenguajes de alto nivel tales como Java y Python. Los avances hacia códigos más portables y reutilizables han impulsado un cambio significativo en la industria de la informática.

Los dispositivos de red siempre se han considerado como cajas negras configurables hasta la aparición de las SDN. SDN permite que las redes puedan ser programadas de acuerdo con las necesidades de los usuarios; permitiendo además que la red pueda ser modificada fácilmente para adaptarse a demandas de transitorias [3], [54]. Con el fin de introducir nuevas características en la red a través de métodos de ingeniería de software y lograr plenamente el potencial de SDN, la comunidad de investigadores están tratando de estandarizar los lenguajes de programación para las SDN [66].

Del mismo modo, la programación en redes se está moviendo de lenguajes de máquina de bajo nivel, como por ejemplo OpenFlow ("ensamblador") hacia lenguajes de programación de alto nivel [16]. Los lenguajes de bajo nivel se encargan de gestionar el comportamiento de los dispositivos de la capa de infraestructura; cuestión que obliga a los desarrolladores a emplear mucho tiempo en los detalles de bajo nivel en lugar de la solución del problema. Los programas que manejan directamente el protocolo OpenFlow tienen que lidiar con cuestiones tales como el solapamiento y prioridades de las reglas, inconsistencia en el plano de datos entre otras. El uso de estos lenguajes de bajo nivel dificulta la reutilización de código, y la modularidad, cuestión que conduce a un proceso de desarrollo propenso a errores.

Las aplicaciones pueden ser programadas en tres niveles diferentes: Programación de bajo nivel, basadas en API, y lenguaje específico de dominio (DSL, Domain-Specific Language). La programación de bajo nivel usualmente se realiza a través de la interface entre la capa de infraestructura y el plano de control (CDPI, Control to Data-Path Interface). OpenFlow se destaca como la principal iniciativa CDPI, pero la programación no es muy fácil. En realidad, desde la perspectiva de los lenguajes de programación, se puede considerar como el lenguaje ensamblador de la red. Las aplicaciones basadas en API utilizan un software que radica en el controlador que es la interface entre la aplicación y el controlador, quien finalmente traduce las peticiones de la aplicación a mensajes CDPI que son enviados a la capa de infraestructura. Los lenguajes de programación DSL ofrecen, a través de notaciones y abstracciones, potencialidades centradas en resolver problemas de red de todos tipos. Los lenguajes de programación SDN tienen un compilador (o estructura) responsable de traducir los mensajes de alto nivel a la interface NBI del controlador. Algunos lenguajes son diseñados para manejar problemas específicos, mientras que otros ofrecen abstracciones más generales, y se pueden utilizar para crear cualquier aplicación de red [54].

Capa VIII: Aplicaciones de Redes

Las aplicaciones y servicios de red tradicionales de las SDN se construyen en la parte superior de la arquitectura SDN. A través de la capa de control, las aplicaciones SDN que son consideradas el “cerebro de la red” pueden tener en tiempo real una visión global de la misma a través de la interfaz NBI de los controladores. Utilizando esta información, las aplicaciones SDN pueden implementar mediante la programación estrategias para manipular las redes físicas subyacentes utilizando un lenguaje de alto nivel proporcionado por la capa de control. En este aspecto, SDN ofrece un modelo de "plataforma como servicio" para la interconexión de redes. A continuación, se describen varias aplicaciones SDN que pueden ser construidas:

Encaminamiento adaptativo: La conmutación de paquetes y el enrutamiento son las principales funciones de una red. Tradicionalmente, los diseños de conmutación y enrutamiento se basan en enfoques distribuidos. Sin embargo, dichos diseños distribuidos tienen muchas deficiencias relacionadas con la complejidad de las implementaciones, lenta convergencia y limita la capacidad de lograr un control adaptativo. Como solución alternativa SDN ofrece un control lógicamente centralizado que retroalimenta a las aplicaciones del estado global de la red. Este nuevo paradigma permite a las aplicaciones de forma adaptativa controlar el comportamiento de la red. Una de las aplicaciones más populares de SDN es la de balance de carga y diseño entre capas. Una aplicación que por ejemplo se encargue de encaminar el tráfico en una red; tendrá una lógica de control para definir las rutas a través de las cuales los paquetes van a ir de un punto “A” a otro “B”. Para lograr este objetivo, una aplicación de enrutamiento necesita, en base a la topología de la red, decidir sobre la ruta a utilizar y dar instrucciones al controlador para que instale las respectivas reglas en los dispositivos de la capa de infraestructura que intervienen en la comunicación entre los puntos A y B.

Balance de carga: El balance de carga es una técnica ampliamente utilizada para lograr un mejor uso de los recursos de la red. “*Cross-layer”* es una técnica muy popular para mejorar la integración e intercambiar información entre las entidades

en diferentes capas. Ejemplo de aplicaciones de este tipo en SDN son las que tienen como misión garantizar calidad de servicio (QoS).

Roaming ilimitado: En SDN, las redes de diferentes operadores con diferentes tecnologías podrían tener un plano de control unificado común. Esto permite la movilidad sin límites al realizar conmutaciones entre distintas tecnologías y soportes.

Mantenimiento de la red: Los errores de configuración son causas comunes de fallas en la red. Se ha reportado que más de 60% de las interrupciones de la red se debe a errores de configuración humanos. Las herramientas de red existentes actualmente que se ocupan del diagnóstico individual como ping, traceroute, tcpdump y NetFlow, no proporcionan una solución de mantenimiento de toda la red. A modo de comparación, la gestión centralizada, la automatizada; la aplicación de políticas, inherente a las redes SDN, ayudan a reducir los errores de configuración. Por otra parte, con una visión global y un control centralizado de la configuración, SDN ofrece oportunidades para diseñar un diagnostico avanzado de la red; con el objetivo de crear un mantenimiento automatizado. Un ejemplo de ello son los mecanismos de pronóstico en los que una aplicación SDN puede resolver directamente los fallos de red con un menor tiempo de convergencia de enrutamiento.

Seguridad de la red: La seguridad de la red es una parte notable de la seguridad cibernética y está ganando atención constantemente. Las prácticas tradicionales relacionadas con la seguridad de red se implementan desplegando cortafuegos, SBC, servidores proxy para proteger una red física.

Debido a la heterogeneidad de las aplicaciones en las redes, garantizar accesos exclusivos para las aplicaciones a la red, implica la aplicación de una política en toda la red y una tediosa configuración de cortafuegos, servidores proxy, y otros dispositivos. En este aspecto, SDN ofrece una plataforma conveniente para centralizar, combinar y controlar las políticas y configuraciones para asegurarse de que la implementación cumple con la protección requerida. De esta manera de una forma proactiva se evitan brechas de seguridad. Por otra parte, SDN proporciona mejores métodos para detectar y defenderse de ataques de forma reactiva. Debido a que SDN tiene la capacidad de recopilar estado de la red, se pueden analizar los patrones de tráfico en busca de amenazas de seguridad potenciales. Los ataques, como “ataques de ráfaga de baja velocidad” y distribuidos de denegación de servicio (DDoS, por sus siglas en inglés), se pueden detectar simplemente mediante el análisis de patrones de tráfico.

Las redes SDNs pueden ser desplegadas en cualquier entorno de red tradicional, desde las redes domésticas y empresariales hasta en a los centros de datos y puntos de acceso a Internet. Tal variedad de entornos ha dado lugar a una amplia gama de aplicaciones de red. Las aplicaciones de red existentes realizan funciones tradicionales como las vistas anteriormente, pero además también se exploran nuevos enfoques, tales como las de reducir el consumo de energía. Otros ejemplos incluyen la conmutación ante fallos y fiabilidad en el plano de datos, aplicaciones de calidad de servicio (QoS, por sus siglas en ingles) de extremo a extremo, la virtualización de redes, gestión de la movilidad en redes inalámbricas, entre muchas otras. Se espera que la variedad de las aplicaciones de red, combinado con la implementación real de casos de uso, sean una de las principales fuerzas para la promoción y adopción de SDN.

1.5 Estandarización de las SDN

La ONF es un consocio industrial sin ánimo de lucro que está liderando la evolución de SDN y estandarizando los elementos críticos de esta nueva arquitectura, tal y como el protocolo OpenFlow. El consorcio fue fundado en 2011 por Deutsche Telekom, Facebook, Google, Microsoft, Verizon y Yahoo!; contando en la actualidad con más de 140 compañías miembro.

Otros organismos de estandarización (ATIS, Broadband Forum, ETSI, OIF, ITU-T, etc.) en especial el IETF, a través del grupo el grupo I2RS (Interface to the Routing System), también están trabajando en extender sus especificaciones para soportar los principios de SDN. Además de los organismos de estandarización, existen varias iniciativas de código abierto, como OpenStack, que están especificando varias herramientas SDN.

ETSI y ONF han mantenido una fuerte colaboración desde la creación de la NFV ISG en 2012. Además, en marzo de 2014, ETSI y ONF firmaron un acuerdo estratégico de colaboración para impulsar el desarrollo conjunto de las especificaciones de NFV, lo cual sin duda acelerará la adopción de tanto NFV como SDN, así como su desarrollo coordinado. Las dos organizaciones muestran así su compromiso en soportar los requerimientos de los operadores y un enfoque complementario en el desarrollo de estándares, metodologías y compartición del conocimiento. ETSI NFV ISG promoverá

las pruebas de concepto que empleen tanto NFV como SDN para demostrar los beneficios de emplear conjuntamente ambas tecnologías.

La ONF ha realizado un informe técnico que muestra cómo los operadores están combinando NFV y SDN para conseguir los objetivos comunes de ambas tecnologías para mejorar la agilidad de la red. Explica los retos que los operadores tienen que superar para implementar NFV y presenta casos de uso que demuestran cómo SDN habilitada por OpenFlow puede satisfacer las necesidades de una red más flexible, programable y abierta para soportar NFV. El informe demuestra cómo SDN puede acelerar los despliegues de NFV de forma escalable y elástica.

1.6 El protocolo OpenFlow

El protocolo OpenFlow constituye la base de las redes abiertas definidas por software. 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 versión más reciente es la 1.5, creada en diciembre de 2014. Aunque el estándar sigue evolucionando, la versión que ampliamente se ha desplegado e implementado es la 1.0.0.

El protocolo OpenFlow ha evolucionado durante el proceso de estandarización de la ONF, a partir de la versión 1.0, donde sólo existen 12 campos coincidentes y una sola tabla de flujo a la última versión que cuenta con varias tablas, más de 41 campos así como nuevas funciones [67].

La motivación para crear OpenFlow surgió de la necesidad de contar con una red experimental para la investigación que opere en paralelo con la infraestructura de red propia de un campus. El objetivo del protocolo es conseguir que los fabricantes incorporen una plataforma abierta en sus dispositivos, y que sea accesible desde una interfaz que permita configurar su comportamiento, pero que a la vez mantenga compatibilidad con las plataformas existentes (propias del fabricante). Se buscó además que la arquitectura esté disponible en entornos de alto rendimiento y en dispositivos de bajo costo.

En la primera versión del protocolo (1.0.0), solo existe una tabla de flujos con tres componentes en la entrada de flujo que son: Campos de cabecera, contadores y acciones [68]. Esta versión del protocolo tiene poca flexibilidad debido a la existencia de una sola tabla así como una limitada cantidad de campos de coincidencias. Los switch que implementen esta versión presentan problemas de explosión de las tablas de flujos motivado fundamentalmente porque el dispositivo al tener una sola tabla no puede ejecutar mas de una operación durante el proceso de envío de paquetes [69]. Como resultado, en la versión 1.1, OpenFlow introduce dos mejoras: múltiples tablas y grupos de tablas [68]. Al mismo tiempo se le cambia el nombre de los campos de cabecera a campos de coincidencia producido porque en realidad los campos de cabecera no coinciden con los del paquete. En esta versión también cambia el nombre de acciones por instrucciones. Al existir más de una tabla pueden ejecutarse múltiples acciones sobre un mismo paquete y existen menos entradas de flujos en tablas individuales. El empleo de múltiples tablas hace mas compleja la implementación del protocolo en los switch al procesar de manera flexible los paquetes y patrones de tablas de flujo [70]. Para solucionar los problemas anteriores el grupo de trabajo abstracciones de la capa de infraestructura de la ONF propuso el modelo NDN (Negotiable Dataplane Model) con el objetivo de ofrecer a los proveedores de dispositivos patrones de tipos de tablas.

Lo que propone OpenFlow es un nuevo enfoque para que los investigadores puedan experimentar con protocolos en las redes que se usan a diario. Permite a los investigadores experimentar con conmutadores de manera uniforme a la velocidad de la línea y con una densidad de puertos muy alta. Por otra parte, los fabricantes no tienen que exponer los procesos internos de sus conmutadores. La propuesta de OpenFlow es muy clara, permitir que los investigadores puedan evaluar sus ideas en un entorno de trabajo real y ser un componente muy útil para desarrollar plataformas de pruebas a gran escala.

El switch OpenFlow

La mayoría de los conmutadores Ethernet contienen tablas de flujos, el paradigma de OpenFlow es la posibilidad de modificar estas tablas de forma dinámica con el objetivo de implementar cortafuegos, NAT, QoS y recolectar estadísticas. Es posible experimentar con nuevos protocolos, nuevos modelos de seguridad, esquemas de direccionamiento, incluso

alternativas para IP lo que supone una mayor innovación. El plano de datos no se vería afectado ya que está aislado y se procesaría de la misma manera que se ha estado realizando hasta hoy día. El cambio reside en el plano de control que se implementaría mediante la utilización del protocolo OpenFlow.

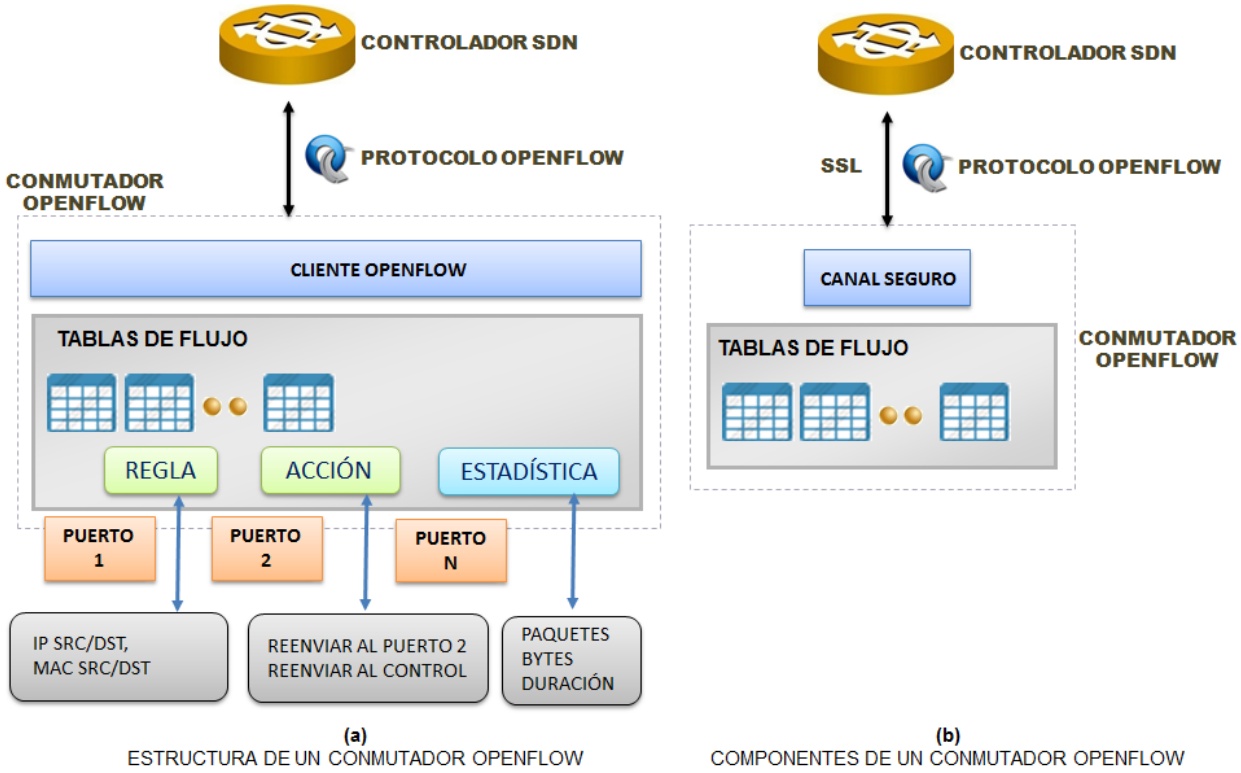


Figura 1.8. Estructura y componentes de un conmutador OpenFlow.

Los conmutadores OpenFlow pueden soportar diversas funcionalidades, aunque existen una serie de características comunes a todos ellos. La arquitectura de un conmutador OpenFlow consiste en al menos tres partes (ver figura 1.8 b)

- Tabla de flujos: Con una acción asociada a cada entrada de la tabla, indicando al switch como debe procesar ese flujo.
- Canal seguro que conecte el switch a un proceso de control remoto (controlador), que permita el envío de comandos y paquetes entre el conmutador y el controlador usando el protocolo OpenFlow.
- Controlador: Un controlador añade y elimina entradas en la tabla de flujos.

Los conmutadores tradicionales utilizan STP, SPB o TRILL para determinar cómo se reenvían los paquetes. OpenFlow traslada esta decisión de reenvío de los *conmutadores* a los controladores (típicamente un servidor o una estación de trabajo). Una aplicación de gestión se ejecutará en las interfaces del controlador que unen todos los *conmutadores* en la red, facilitando la configuración de caminos de reenvío que utilizarán todo el ancho de banda disponible. La especificación define el protocolo entre el controlador y los *conmutadores* y un conjunto de operaciones que se pueden realizar entre ellos Las instrucciones de reenvío se basan en el flujo, y son aplicadas a todos los paquetes comparten una serie de características comunes.

Un flujo en un conmutador puede estar definido por muchos criterios entre los que encontramos: puerto donde se recibe el paquete, etiqueta VLAN, MAC origen o destino, protocolo, etc. Si un paquete no encuentra ninguna coincidencia en las tablas debe ser enviado al Controlador SDN. Dicho controlador definirá un nuevo flujo para el paquete y enviará al conmutador una o más entradas para las tablas de flujo existente. Para la próxima ocasión los paquetes que coincidan con este nuevo flujo no tendrán que esperar la respuesta del Controlador, sino que serán encaminados a velocidad de línea por el conmutador OpenFlow.

La tabla de flujo (ver figura 1.9) sirve para realizar las búsquedas de paquetes de acuerdo a ciertas reglas definidas que determinan los criterios para el encaminamiento del tráfico. El canal seguro sirve para comunicarse con el controlador, el cual se encarga de configurar el conmutador a través del protocolo OpenFlow.

Una tabla de flujo contiene una serie de entradas de flujo. Cada entrada de la tabla de flujo contiene:

- Campos de encabezado o coincidencia (Match fields): Sirven para definir criterios de coincidencia (match) con los paquetes.
- Prioridad del flujo (Priority): En el caso de que haya más de un flujo con el mismo match tendrá prioridad el flujo con el número de prioridad más alto.
- Contadores (Counters): Contadores que se actualizan cada vez que se produce una correspondencia.
- Instrucciones (Instructions): Instrucciones a realizar cuando se produzca una correspondencia.
- Temporizador (Timeouts): tiempos antes de que un flujo expire. Son dos: idle_time_out y hard_time_out. El primero hace referencia al tiempo que un flujo lleva sin coincidencias de tráfico. El segundo indica el tiempo real desde que se instaló el flujo.
- Cookie: Valor que elige el controlador para filtrar las estadísticas, las modificaciones y el borrado de los flujos. No se usa: cuando se procesan los paquetes.

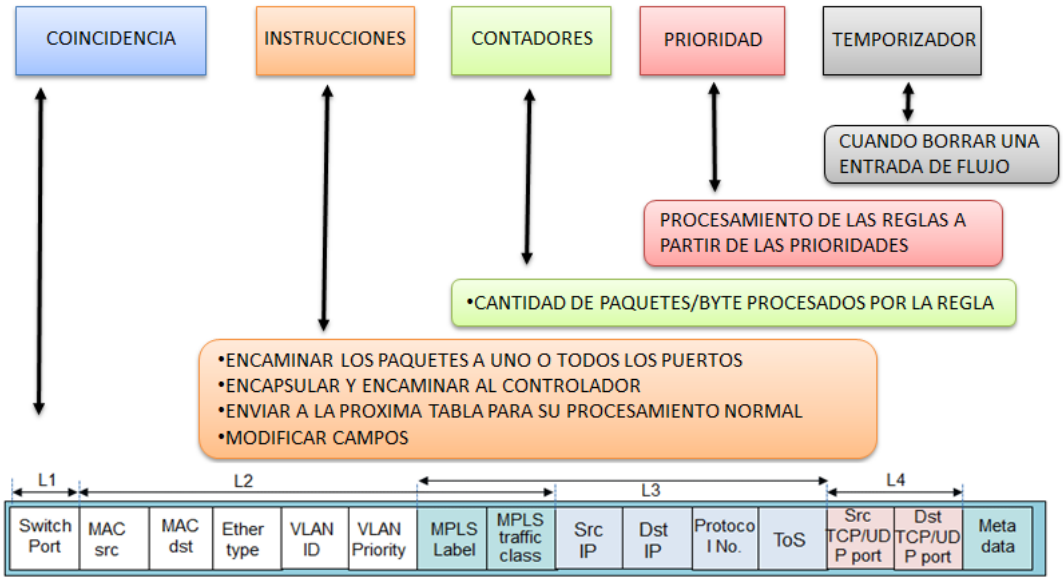


Figura 1.9. Anatomía de la tabla de flujo.

Mensajes del protocolo openflow

Los mensajes de OpenFlow pueden clasificarse en tres tipos: Controlador a conmutador, asíncronos y simétricos:

Controlador a conmutador: Iniciados por el Controlador para conocer el estado del conmutador:

- *Features:* A través de una sesión TLS (Transport Layer Security) el controlador solicita al conmutador las funcionalidades soportadas. El conmutador debe responder con un mensaje que especifique las capacidades que soporta.
- *Configuration:* El controlador puede establecer y consultar parámetros de configuración del conmutador. El conmutador sólo responde a las consultas del controlador.
- *Modify-State:* Estos mensajes son enviados por el controlador para administrar el estado de los conmutadores. Su

propósito primario es añadir, eliminar o modificar las tablas de flujo, y establecer las propiedades del puerto.

- *Read-State:* Estos mensajes son usados por el controlador para obtener estadísticas de las tablas de flujo de los conmutadores, puertos y entradas de flujo individuales.
- *Packet-out:* Son usados por el controlador para enviar paquetes de salida a través de un puerto específico del conmutador.
- *Barrier:* Enviado por el Controlador para informar que determinadas operaciones se han completadas.
- *Role-Request:* Estos mensajes de solicitud y respuesta son usados por el controlador para asegurarse que las dependencias se han cumplido o para recibir notificaciones de operaciones completadas.

Mensajes asíncronos: Los mensajes asíncronos son iniciados por el conmutador sin que el controlador los solicite. Los conmutadores envían estos mensajes al controlador para notificar la llegada de un paquete, un cambio de estado en el conmutador o un error. Los mensajes principales son:

- *Packet-in:* Se usa para todos los paquetes que no corresponden con ninguna entrada en la tabla de flujo, o para aquellos que tengan configurada como acción el envío al controlador.
- *Flow-Removed:* Cada vez que se añade una entrada a la tabla de flujo a través de un mensaje *Modify-State*, se especifica un tiempo de inactividad (idle timeout) después del cuál debe ser eliminada la entrada en la tabla, y un tiempo de vida (hard timeout) que elimine la entrada después de un tiempo determinado, independientemente de la actividad. En estos mensajes de modificación también se especifica si el conmutador debe o no enviar un mensaje Flow-Removed al controlador cada vez que expira la entrada en la tabla de flujo.
- *Port-status:* Estos mensajes son enviados por el conmutador cada vez que cambia el estado en la configuración de un puerto.
- *Error:* Son usados para notificar al controlador problemas en el conmutador.

Mensajes simétricos: Los mensajes simétricos son iniciados, bien por el conmutador o por el controlador, enviándose sin necesidad de solicitud previa. Existen tres tipos:

- *Hello:* Son usados por el conmutador y controlador durante el establecimiento de la conexión.
- *Echo:* Los mensajes de solicitud (echo request) requieren siempre una respuesta (echo reply). Pueden ser generados por el conmutador o controlador y son usados para obtener la latencia, ancho de banda o estado de una conexión.
- *Vendor:* Son usados para ofrecer funcionalidades adicionales dentro del posible espacio de mensajes OpenFlow. Está sujeto a futuras revisiones.
- *Experimenter:* Mensajes de experimentación para proveer funcionalidades adicionales de forma estándar.
- *Error: Mensajes de error notificados al otro extremo de la conexión. Normalmente, usado por el switch para indicar el fallo de una petición iniciada por el Controlador.*

La evolución de los campos de coincidencia es mostrada en la tabla 1.1; observándose como de una versión a la otra aumentan significativamente los mismos. A este fenómeno se le conoce en la comunidad científica como explosión de los campos de coincidencia. Debido a que constantemente se está cambiando la especificación de OpenFLow, los especialistas plantean que en el futuro el protocolo deberá soportar mecanismos flexibles para el análisis de los paquetes y la comparación con campos de coincidencia [71]. Otra característica significativa de la versión 1.2 es que el controlador puede cambiar de rol, lo que permite que puedan existir en la red múltiples controladores en una misma red. Los roles del controlador pueden ser Maestro, Esclavo o Igual; esto permite la conmutación en caso fallo de un elemento de control a otro con la consiguiente mejora de la disponibilidad de la infraestructura de la red [72].

Para brindar calidad de servicio en la redes, OpenFlow introduce una nueva funcionalidad en la versión 1.3 llamada tabla de contadores para extender la capacidad de calidad de servicio. También en esta versión son ampliadas las tablas de

flujo al introducirse el concepto de entrada no encontrada “table-miss”. Si un paquete no coincide con ninguna entrada de flujo en una tabla se procede entonces a descartarlo, pasarlo a otra tabla de flujo o enviarlo al controlador.

OpenFlow 1.4 amplía aún más la escalabilidad de la tabla de flujo al introducir la “sincronización de tablas". Con este procedimiento se pueden sincronizar de forma unidireccional o bidireccional las tablas de flujos. Si la sincronización de una tabla es bidireccional, entonces los cambios realizados por el controlador deben ser reflejados en la tabla de origen [73]. Por otra parte, OpenFlow 1.4 introduce una nueva característica llamada "Bundle". Esta funcionalidad permite realizar modificaciones por grupos y aplicarlas todas al mismo tiempo. Esta opción también es útil para preparar y validar mensajes OpenFlow aplicados a múltiples conmutadores.

La versión 1.5 de OpenFlow permite programar un horario para la ejecución de la funcionalidad “Bundle” anteriormente mencionada. Esta opción fortalece aún más la sincronización entre conmutadores. Otra caracterísitica importante es la aparición de la tabla de salida que posibilita realizar un procesamiento adicional que permite buscar coincidencias en base al puerto de salida. La tabla 1.1 muestra un resumen comparativo de las versiones del protocolo OpenFlow.

VERSIÓN	CAMPOS DE COINCIDENCIA	ESTADÍSTICAS
1.0	Puerto e entrada	Estadísticas por tabla
	Ethernet: src, dst, type, VLAN	Estadísticas por flujo
	IPv4: src, dst, proto, ToS	Estadísticas por puerto
	TCP/UDP: src port, dst port	Estadísticas por colas
1.1	Metadata, SCTP, VLAN tagging	Estadísticas por grupos
	MPLS: label, traffic class	Estadísiticas por acciones
1.2	OpenFlow Extensible Match (OXM)	
	IPv6: src, dst, flow label, ICMPv6	
1.3	PBB, IPv6 Extension Headers	Contadores por flujo
		Banda de contadores por flujo
1.4	--	Propiedades del puerto optico

Tabla 1.1. Comparación entre las versiones del protocolo OpenFLow.

1.7 Conclusiones del capítulo

Los desarrollos recientes en el ámbito de las TIC están haciendo que los usuarios demanden mayor ancho de banda y acceso a Internet, así como de una gestión más dinámica de sus servicios. SDN se considera como una solución prometedora para satisfacer estas demandas. El uso de la arquitectura SDN brinda ventajas significativas con respecto a las redes tradicionales como: facilidad de innovación y gestión, independencia de proveedores e incremento del rendimiento.

Las principales características y potencialidades de las Redes Definidas por Software básicamente se destacan por tres atributos: inteligencia de la red lógicamente centralizada, programación y separación de los planos de control y datos. En esta arquitectura sobresale una amplia utilización del protocolo OpenFlow como interface de comunicación entre la capa de control y la de infraestructura. Las redes SDNs implementadas con el protocolo OpenFlow, proporcionan un poderoso enfoque que garantiza una programación de la red, independencia de proveedores y la gestión de redes complejas con demandas dinámicas; al mismo tiempo que acelera la innovación en la red. Estas redes, introducen una nueva etapa en el desarrollo de las TIC, permitiendo mejorar significativamente la escalabilidad, fiabilidad, agilidad, automatización, seguridad, capacidad de gestión, dentro de la red mediante el empleo de controladores y aplicaciones SDN. Todo lo anterior justifica la necesidad de disponer de plataformas de pruebas que permitan evaluar el desempeño de estas nuevas arquitecturas y realizar estudios comparativos con las redes tradicionales. También será necesario abordar los controladores disponibles en la actualidad con soporte para el protocolo OpenFlow; además de disponer de herramientas que permitan la creación y evaluación de este tipo de redes.

CAPÍTULO 2. MÉTRICAS RELEVANTES PARA LAS SDN Y LA SELECCIÓN DEL CONTROLADOR

Debido al carácter aún novedoso de las SDN, se hace complejo el proceso de selección y evaluación de un controlador. En este capítulo se realiza una investigación de los controladores SDN comerciales y de código abierto existentes. Para el correcto desarrollo de esta investigación es realizada una búsqueda de los elementos claves para escoger un controlador con vistas a sistematizar la toma de decisiones relacionadas con la selección de este tipo de elemento de red. También son identificadas las métricas de desempeño que usualmente se utilizan para evaluar los mismos. Finalmente son abordadas las plataformas de simulación, herramientas para la depuración, solución de problemas y de evaluación existentes.

2.1 Introducción

El concepto de redes programables se ha planteado como una forma de facilitar la evolución de las redes de datos. Se ha explicado en el capítulo anterior que la característica distintiva en las redes programables, y en SDN en particular, es el desacoplo de las decisiones de control de las funciones de reenvío de paquetes en lo que se denominan planos de control y de datos. Esta separación proporciona abstracción de la red flexible y programable. Además de la abstracción de redes, la arquitectura SDN proporciona un conjunto de Interfaces de Programación de Aplicaciones que simplifican la implementación de servicios de red comunes; por ejemplo, el encaminamiento, la multidifusión, la gestión de la seguridad, el control de accesos, la gestión de ancho de banda, entre otras. A continuación se realiza un estudio de los principales controladores encontrados en esta investigación.

2.2 Controladores SDN disponibles y elementos para su selección

El plano de control es una parte esencial de la arquitectura SDN. Como se ha visto anteriormente un controlador básicamente añade y elimina entradas en la tabla de flujos en los dispositivos de la capa de infraestructura. Aunque existen numerosos controladores disponibles y que soportan diferentes protocolos hacia el sur, en este trabajo serán estudiados los que soportan el protocolo OpenFlow.

Existen diversos controladores y plataformas de control con arquitecturas y diseños variados. El primer controlador SDN con soporte para el protocolo OpenFlow fue el NOX y fue escrito en el lenguaje de programación C++ y estaba destinado a ser ejecutado en distribuciones de Linux [74].

En la evolución de las SDN los investigadores han identificado capacidades claves que debe tener un controlador. Los controladores se pueden clasificar en base a muchos aspectos. Desde un punto de vista de la arquitectura, una de las más relevantes son la centralizada o distribuida.

Un controlador centralizado es una entidad que gestiona todos los dispositivos de la capa de infraestructura. Naturalmente esto representa un solo punto de fallo y puede tener algunas limitaciones en cuanto a la escalabilidad. Un solo controlador puede no ser suficiente para manejar una red con un número elevado de nodos. Los controladores centralizados, NOX-MT [75], Maestro [32], Beacon [76], y Floodlight[77] han sido diseñados como sistemas altamente concurrentes, para lograr rendimientos elevados en redes de clase empresarial y centros de datos. Estos controladores se basan en diseños de múltiples subprocesos y emplean técnicas de paralelismo en arquitecturas con múltiples núcleos.

Al contrario de un diseño centralizado, un sistema operativo de red distribuido puede ser ampliado para satisfacer las necesidades de un entorno de red pequeño o de gran escala. Mientras que la primera alternativa puede ofrecer un alto rendimiento para centros de datos muy densos, la segunda puede ser más resistente a diferentes tipos de fallos lógicos y físicos [78]. Si bien SDN ofrece un punto de vista de un plano de control lógicamente centralizado, existe un amplio consenso que el plano de control debe ser distribuido espacialmente. Con el fin de proporcionar una alta disponibilidad, los controladores deben ser redundantes [79].

La mayoría de los controladores distribuidos ofrecen una consistencia débil, lo que significa que los datos se actualizan eventualmente. Esto implica que existe un período de tiempo en el que algunos nodos pueden leer valores viejos o nuevos para una misma propiedad. Los controladores deben poder formar clúster para garantizar la redundancia y escalabilidad de la red de forma tal que permita el incremento de nuevos dispositivos y la recuperación ante el fallo de un controlador.

Otra propiedad importante es la implementación de una variedad de interfaces hacia el sur lo que permite al controlador manipular y optimizar el comportamiento de los conmutadores y enrutadores a la hora de gestionar el tráfico.

El soporte de API extensibles garantiza que el controlador pueda ser utilizado en entornos variados para orquestar las comunicaciones a nivel de capa 2, 3, 4-7. Un controlador debe ser capaz de soportar los sistemas de orquestación de OpenStack así como los protocolos específicos del proveedor [80].

Además debe manejar un conjunto de APIs NBI, a menudo RESTful (Transferencia de estado representacional o *Representational State Transfer*, por sus siglas en inglés) que exponen los servicios del controlador a las aplicaciones de gestión. Esto facilita la interacción del controlador con estas aplicaciones.

La programabilidad de un controlador es la habilidad de redirigir el tráfico, aplicar filtros sofisticados a paquetes, en fin cambiar el comportamiento de los elementos de la capa de infraestructura de forma dinámica.

Estos elementos de red también deben garantizar un alto rendimiento al maximizar la cantidad de flujos por segundos y reducir al mínimo el tiempo de configuración de las entradas de flujo en los conmutadores y enrutadores. También al procesar el tráfico debe poder trabajar en los modos proactivos y reactivos y sin crear cuellos de botellas al hora de procesar las entradas de flujo.

Con la diversidad de casos de uso de hoy en día, así como la variedad de controladores, los investigadores, empresas, operadores se cuestionan constantemente la mejor manera de elegir o evaluar un controlador. Inmediatamente se realiza la pregunta científica: ¿Cómo elegir un controlador entre tanta diversidad?

La selección del controlador es vital en el diseño de una red SDN. A continuación se presentan las características que los autores de este trabajo, después de revisar la bibliografía, consideran que se deben tener en cuenta al evaluar un controlador SDN [20], [80], [81]. Estas son:

- Soporte del protocolo OpenFlow: Al elegir un controlador los operadores y administradores de redes necesitan conocer las características de las versiones del protocolo OpenFlow que el controlador puede manejar, así como las posibilidades que ofrece el proveedor para migrar a las nuevas versiones del protocolo.
- Elegir entre código abierto o propietario: Los primeros usuarios de SDN indican que este atributo sólo debe ser importante si se está considerando extender el controlador o realizar modificaciones específicas para una red. Ciertamente las soluciones de código abierto reducirán la probabilidad de dependencia de un proveedor, pero con un mayor enfoque en las APIs en dirección sur basadas en estándares como OpenFlow o ovsdb, y teniendo en cuenta las sensibilidades del mercado existentes en la prevención de obtención de información vital de una red. Se le debe prestar mayor atención a las funcionalidades y ajustes, así como otros atributos.

- Virtualización de red: Debido a los beneficios que ofrece la virtualización de red, un controlador SDN debe soportarla. Estas funcionalidades fueron explicadas anteriormente.
- El ecosistema de aplicaciones y la madurez de las API NBI. Se deben identificar los problemas del negocio o la red que se desean resolver para buscar si el controlador posee aplicaciones que implementen esa solución. Tan importante como entender si existen aplicaciones SDN para resolver un problema, se debe identificar si el controlador fue diseñado para las áreas de implementación que se necesitan. Los controladores diseñados para implementaciones WAN pueden que no funcionen bien en el entorno de los centros de datos o un controlador diseñado para funcionar en entorno empresarial pudiera tener poco desempeño en una implementación WAN.
- Rendimiento: Una de las principales funciones de un controlador SDN es la de gestionar las tablas de flujos de los dispositivos de la capa de infraestructura. Por ello, dos de los indicadores claves de rendimiento asociados con un controlador SDN son el tiempo de conformación de flujo y el número de flujos por segundo que puede establecer el controlador. Estas métricas de desempeño serán analizadas posteriormente.
- Funcionalidad de la red: Para lograr mayor flexibilidad en términos de cómo los flujos son enrutados, es importante que el controlador SDN pueda tomar decisiones de enrutamiento basado en múltiples campos de la cabecera del protocolo OpenFlow. También es importante que el controlador pueda definir los parámetros de QoS flujo por flujo. Otra importante funcionalidad en un controlador SDN es su capacidad para descubrir múltiples caminos desde el origen del flujo a su destino y para dividir el tráfico de un flujo dado a través de múltiples enlaces. Esta capacidad suprime la necesidad de STP (Spanning Tree Protocol por sus siglas en inglés) y aumenta el rendimiento y la escalabilidad de la red permitiendo, también, eliminar la necesidad de añadir a la complejidad de la red nuevos protocolos como TRILL (Transparent Interconnection of Lots of Links) o SPB (Shortest Path Bridging).
- Madurez, fiabilidad del controlador y su compatibilidad con equipos de red: Dado que el controlador será un elemento clave de la red, es importante examinar la madurez de la solución considerada y, en particular, prestar atención al número de despliegues de producción, las áreas de despliegue; así como la duración de los despliegues. Asimismo, se debe asegurar que el Controlador que se está evaluando ha sido probado (o mejor aún, certificado) con el equipo que va a utilizar. Incluso con protocolos estándar como OpenFlow, puede haber pequeñas pero importantes incompatibilidades en la implementación que pueden afectar la eficacia o fiabilidad de una solución.
- Integración con plataformas de orquestación: Casi todos los controladores comerciales y muchos controladores de código abierto proporcionan soporte OpenStack en el centro de datos. Si se tiene una implementación de centro de datos y ha elegido una plataforma específica de gestión de la nube o de orquestación (OpenStack, VMware vRealize, CloudStack u otros), consulte al proveedor o evalúe la distribución de código abierto para garantizar una integración perfecta en el entorno de orquestación.
- Escalabilidad: Una consideración fundamental con respecto a la escalabilidad de una red SDN es el número de conmutadores que un controlador SDN puede soportar. En la actualidad se debe esperar que los controladores soporten un mínimo de 100 conmutadores, pero en última instancia esto depende de las aplicaciones que soportan. Otro factor que limita la escalabilidad de una red SDN es la proliferación de entradas en la tabla de flujo, ya que sin algún tipo de optimización, se requiere de una entrada salto por alto para cada flujo. Al evaluar los controladores SDN, es necesario asegurarse que el controlador puede disminuir el impacto de sobrecarga de difusión de red, la cual limita la escalabilidad de la arquitectura de red implementada y reducir al mínimo la proliferación de las entradas de la tabla de flujo. Otro aspecto de la escalabilidad es la capacidad del controlador de SDN para crear una SDN que pueda abarcar múltiples sitios. Esta capacidad permite el movimiento de máquinas virtuales y el almacenamiento virtual entre sitios. Para maximizar el beneficio de esta capacidad, el controlador SDN debe permitir que las políticas de red

para el enrutamiento y reenvío se apliquen automáticamente para la migración de servidores y / o almacenamiento.

- Programación de red: Una de las características fundamentales de las SDN es la existencia de interfaces para la programación del controlador, lo que posibilita que este ofrezca varias funcionalidades. Ejemplos de programación que se deben buscar en un controlador SDN son la capacidad de redirigir el tráfico (por razones de seguridad se puede disponer que el tráfico entrante a un servidor pase a través de un corta fuegos, pero para no consumir los recursos del servidor de seguridad con tráfico limpio, se puede decidir que no pase por el cortafuegos el tráfico saliente del mismo servidor) y la posibilidad de aplicar filtros sofisticados a los paquetes (que pueden ser pensados como ACLs (Listas de Control de Acceso o Access Control List por sus siglas en inglés) dinámicas e inteligentes como combinaciones complejas de múltiples campos de cabecera de paquetes).
- Confiabilidad: Una de las técnicas que un controlador SDN puede utilizar para aumentar la fiabilidad de la red, es la capacidad de descubrir múltiples caminos desde el origen hasta el destino lo cual puede realizar si continuamente controla la topología de la red. Si el controlador SDN establece varias rutas entre el origen y el destino, la disponibilidad de la solución no se ve afectada por la interrupción de un solo enlace. Alternativamente, si el controlador SDN sólo establece una única ruta del origen al destino, cuando ocurra un fallo en un enlace, el controlador debe ser capaz de redirigir el tráfico rápidamente a un enlace activo.
- Seguridad de la red: Con el fin de proporcionar seguridad a la red, un controlador SDN debe ser capaz de soportar la autenticación y autorización. Debido a que un controlador SDN puede ser atacado, necesita poseer la capacidad de limitar las comunicaciones de control, y ser capaz de mitigar la amenaza cuando la red está experimentando una sospecha de ataque.
- Monitorización centralizada y visualización: Un controlador SDN tiene que ser capaz de utilizar los datos ofrecidos por el protocolo OpenFlow para identificar los problemas en la red y, automáticamente, cambiar la ruta que toma un flujo determinado. El controlador también debe poder definir los tipos de tráfico que se van a controlar. Ser capaz de visualizar una red tradicional siempre ha sido a la vez importante y difícil lo que se incrementa en un entorno en el que múltiples redes virtuales se ejecutan en la parte superior de una red física donde es necesario que el controlador SDN sea capaz de descubrir y presentar la visualización de los enlaces físicos de la red y de las múltiples redes virtuales que se ejecutan en la red física. El controlador también debe permitir ver los flujos, tanto de la perspectiva de la red física como de la virtual, y obtener información detallada sobre los mismos. También, debería ser posible monitorizar el controlador SDN utilizando protocolos y técnicas estándares de gestión tales como SNMP (Protocolo Simple de Gestión de Redes o Simple Network Management, por sus siglas en inglés). Adicionalmente el controlador SDN debe ofrecer soporte para una amplia gama de MIBs (Bases de Información de Gestión o Management Information Bases, por sus siglas en inglés) estándares y MIB privadas para poder controlar los elementos de la red virtual. Lo ideal sería que el controlador SDN ofrezca acceso a la información de red, por ejemplo a los dispositivos conectados, al estado del puerto, y al estado de enlace a través de una API REST.
- Fabricantes de controladores SDN: Dado el creciente interés en las SDN, numerosos fabricantes han entrado en el mercado y muchos más han anunciado la intención de hacerlo. Debido a la volatilidad del mercado SDN en general, y del mercado del controlador SDN, en particular, las organizaciones que deben evaluar controladores SDN para sus redes deben centrarse no sólo en los atributos técnicos del controlador, sino también en las características del vendedor. Entre estas se encuentra la competencia técnica y financiera del proveedor pues las organizaciones no pueden permitirse el lujo de tener sus desarrollos de red SDN perjudicadas por adquirir controladores de un proveedor que no puede mantenerse al día en el cambiante entorno SDN.

- Soporte de plataformas: Los controladores SDN corren sobre sistemas operativos siendo importante que el que se esté evaluando sea multiplataforma pues ello permite mayor flexibilidad e independencia al implantarlo. En muchas entidades es de gran interés que el controlador corra sobre plataformas de código abierto.
- Procesamiento: Al evaluar un controlador se debe conocer si el mismo soporta procesos múltiples o no, pues esto puede repercutir en la escalabilidad de los núcleos de la CPU. No tendría sentido que un controlador mono proceso se ejecute sobre un hardware que posee múltiples CPUs pues no se estaría usando adecuadamente el mismo o que un controlador que soporte procesos múltiples se esté ejecutando sobre un hardware de un solo CPU. Por otra parte, los controladores de procesos múltiples deben ser los utilizados en redes empresariales mientras que los mono proceso se pueden emplear en red pequeñas.

Según los requerimientos particulares de cada aplicación debe hacerse una selección cuidadosa de los mejores controladores y entornos de desarrollo, que satisfagan las necesidades de gestión de una red desde distintas perspectivas. A continuación son abordados los controladores más populares entre los investigadores.

El proyecto OpenDayLight (ODL)

A principios de 2013, fue creado un nuevo proyecto de código abierto de colaboración denominado OpenDayLight (ODL, por sus siglas en inglés). Si bien fue dirigido originalmente por IBM y Cisco, que fue posteriormente acogido por la Fundación Linux y contó con el apoyo y los recursos de toda la industria. La plataforma OpenDayLight proporciona una base común y un conjunto robusto de servicios para entornos SDN. La plataforma sigue un modelo de controlador que permite el uso de OpenFlow, así como protocolos hacia el sur alternativos. También posibilita la programación de aplicaciones SDN y Virtualización de Funciones de Red (Network Functions Virtualization o NFV) para redes de cualquier tamaño o escala. Este controlador SDN tiene una fuerte integración con la plataforma OpenStack.

ODL, fue el primer controlador de código abierto capaz de emplear protocolos no-OpenFlow, lo que hace posible integrar la plataforma en entornos de red heterogéneos de múltiples proveedores [80].

OpenDayLight, está escrito en Java e implementado dentro de su propia máquina virtual (JVM). Ha recibido el soporte de la "Linux Fundation" y de muchas empresas del sector como Cisco, HP, Red Hat, etc. En su hoja de ruta se encuentra dar soporte completo a tecnologías novedosas como NFV, VTN y SFC. El controlador contiene APIs abiertas que son utilizadas por las aplicaciones. OpenDayLight soporta el framework OSGi y REST bidireccional.

La arquitectura de OpenDayLight es totalmente modular. La interfaz de bajo nivel (SouthBound API) soporta múltiples protocolos (plugins de OpenDayLight) como OpenFlow 1.0, OpenFlow 1.3, NETCONF, OVSDB, etc. Los plugins se conectan dinámicamente a la SAL (Service Abstraction Layer) que funciona como una capa de abstracción para los módulos de redes superiores. Es decir, la SAL permite que OpenDayLight ejecute la tarea solicitada independientemente del protocolo usado para conectarse con los equipos físicos.

El framework OSGi se utiliza para las aplicaciones que se ejecutan en el mismo espacio de direcciones que el controlador mientras que REST API se utiliza para aplicaciones que no se ejecutan en el mismo espacio de direcciones que el controlador. La lógica de negocio y los algoritmos residen en las aplicaciones. Estas aplicaciones utilizan el controlador para reunir la inteligencia de red, ejecutar algoritmos para realizar análisis y, seguidamente, usar el controlador para crear nuevas reglas a través de la red.

La interfaz de bajo nivel es capaz de soportar múltiples protocolos (plugins separados) como OpenFlow 1.0, OpenFlow 1.3, BGP-LS, etc. Estos módulos están dinámicamente conectados a la Service Abstraction Layer (SAL). La SAL contiene servicios de dispositivos para los módulos de niveles más altos. La SAL determina cómo cumplir con el servicio solicitado, independientemente del protocolo utilizado entre el controlador y los dispositivos de red.

Su versión más reciente (Beryllium), incluye prestaciones de alta disponibilidad, creación de clústeres y seguridad, así como el mejoramiento o adición de nuevos protocolos.

Este proyecto ha contado con un amplio respaldo de la industria, enfocándose en crear un marco de trabajo abierto para construir aplicaciones SDN y NFV. No se limita a las innovaciones de OpenFlow, sino que está abierta a otros protocolos.

FLOODLIGHT

Floodlight es un controlador SDN de código abierto [82]. Es una contribución de Big Switch Networks a la comunidad. Cuenta con licencia Apache, está escrito en Java y soporta el protocolo OpenFlow como interfaz de comunicación (southbound interface) con los elementos de red. Es un controlador de clase empresarial, que está disponible con licencia Apache para casi cualquier propósito. Es apoyado por una gran comunidad de desarrolladores que incluyen ingenieros de Big Switch Networks. Floodlight está diseñado para trabajar con un gran número de conmutadores, enrutadores, conmutadores virtuales y puntos de acceso compatibles con el protocolo OpenFlow. Es un sistema multiplataforma ya que funciona sobre la máquina virtual de Java.

La arquitectura de Floodlight es modular, con componentes que incluyen las funcionalidades descritas en el modelo de controlador idealizado de la sección anterior. Floodlight adicionalmente proporciona un conjunto de aplicaciones (o módulos) en Java integradas en el controlador. La arquitectura de diseño de Floodlight es similar a la que se muestra en la figura 2.1 y utiliza Java Netty para manejar sockets múltiples hilos.

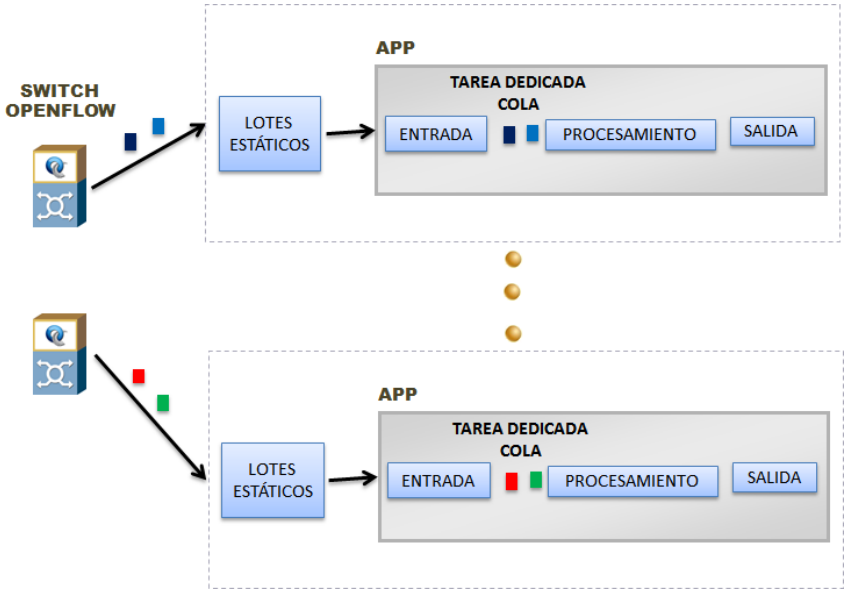


Figura 2.1. Distribución estática. Fuente: Autor basada en [83].

Este controlador utiliza un mecanismo para asegurar que los núcleos no estén subutilizados implementado con una técnica de distribución estática de los conmutadores utilizando un buffer de lectura de tamaño fijo y mediante la asignación de un conmutador a determinado subproceso. Después del procesamiento de los paquetes de entrada los de salida se procesan en grupo para reducir la sobrecarga resultante de realizar llamadas al sistema con cada paquete individual.

ON.Lab (ONOS)

El ON.Lab es una organización sin ánimo de lucro fundada por personas de la Universidad de Stanford y la Universidad de Berkeley. ON.Labs misión es "traer la apertura y la innovación a la Internet y de la nube por el bien público". ON.Lab cree que esta misión se logra mejor mediante la aplicación de los siguientes objetivos:

- Construir herramientas y plataformas que permiten y aceleran las SDN, que estarán disponibles a través de código abierto;
- Educar al público sobre los beneficios de la SDN;
- Proporcionar liderazgo de pensamiento para asegurar la continua innovación en torno a SDN para el beneficio del público.

Uno de los proyectos principales de la ON.Lab es ONOS (Sistema operativo de red) - un sistema operativo de código abierto SDN.

BEACON

Beacon es un controlador OpenFlow, multi-hilos y basado en Java con un diseño arquitectónico similar al de la Figura 2.1. Si n número de subprocesos están configurados para ejecutarse, el controlador genera n + 1 subprocesos, donde el subproceso adicional es responsable de escuchar las conexiones entrantes de los conmutadores y repartirlas entre los subprocesos de trabajo disponibles. Beacon utiliza un enfoque estático en el que se asigna un número fijo de conmutadores a un subproceso de trabajo. Los hilos de trabajo utilizan lotes de paquetes estáticos para servir las peticiones de los conmutadores conectados. Una vez que los paquetes se procesan y están listos para ser enviados, Beacon en su modo predeterminado realiza una escritura por bucle de selección de E / S para reducir la sobrecarga de llamadas de sistema de socket para cada mensaje de OpenFlow individual. Alternativamente, se puede habilitar el modo inmediato, en el que el controlador intenta escribir en el socket por cada mensaje OpenFlow saliente y espera que sea realizada la modificación en el conmutador para reducir la latencia por paquete.

NOX

El controlador NOX [84] fue creado en 2008 y ha sido uno de los primeros controladores SDN de código abierto. Ha contado con contribuciones de la Universidad de Stanford, UC Berkeley e ICSI. NOX proporciona una API en C++ para OpenFlow (versión 1.0) y un modelo de programación asíncrono basado en eventos [85].

NOX es un controlador y un marco de trabajo para crear aplicaciones para SDN. Aunque sólo proporciona módulos para OpenFlow, puede extenderse a otros protocolos. Ha sido usado principalmente en entornos académicos y de investigación.

Los criterios de diseño e implementación de NOX se presentan en el siguiente artículo [74]:

- Componentes: Los componentes de la arquitectura NOX son los conmutadores OpenFlow, un servidor que ejecuta NOX (controlador) y una base de datos que almacena la vista global de la red.
- Granularidad: La arquitectura de NOX es granular y permite compensar aspectos de escalabilidad y flexibilidad.
- Abstracción de conmutadores: Se ha adoptado el modelo definido en el protocolo OpenFlow, representando el comportamiento de los conmutadores en tablas de flujo.
- Operación: Los paquetes que coincidan con entradas de las tablas de flujo actualizarán los contadores y realizarán las acciones establecidas. Los que no, serán enviados al controlador.
- Escalabilidad: Considera un manejo eficiente de escalas de tiempo y requerimientos de consistencia de datos para asegurar la escalabilidad.

NOX-MT

NOX-MT tiene un diseño de arquitectura multi-hilo similar a la Beacon. Utiliza las bibliotecas Boost-Asio para la red y la programación de E / S de bajo nivel. Como se muestra en la Figura 2.1, Utiliza un enfoque estático en el que se asigna un número fijo de conmutadores a un subproceso de trabajo. La técnica de procesamiento por lotes de

paquetes estático se utiliza para reducir las llamadas frecuentes al sistema de lectura. Los paquetes entrantes se procesan uno por uno y luego se procesan juntos en caso de alto tráfico de control antes de que sean enviados al conmutador **[83]**.

POX

En esencia, es una plataforma para el rápido desarrollo y creación de prototipos para el control de la red por software utilizando Python. Es uno de un número creciente de frameworks para SDN (incluyendo NOX, Floodlight, Opendaylight...) con el fin de prestar ayuda para programar un controlador OpenFlow.

POX es la implementación de NOX en Python que aporta las siguientes ventajas:

- Cuenta con una interfaz OpenFlow para Python.
- POX tiene componentes reusables para la selección de rutas, descubrimiento de topologías, etc.
- Soporta las mismas interfaces gráficas de usuario (GUI) y herramientas de visualización que NOX.

Las aplicaciones presentan un buen desempeño comparadas con aquellas escritas en NOX.

RYU

Ryu [86] es un entorno de trabajo basado en componentes para el desarrollo de aplicaciones SDN. Proporciona componentes de software y una API que, en conjunto, facilitan el desarrollo de aplicaciones para la gestión de una red SDN. Soporta varios protocolos para el manejo de dispositivos de red, tales como OpenFlow (versiones 1.0 a la 1.4), Netconf, OF-config, entre otros. Está, a su vez, fuertemente integrado con OpenStack. Está escrito íntegramente en Python, lo que puede constituir un factor a considerar si lo que se requiere es desarrollar aplicaciones de alto rendimiento que suelen ser mejor implementadas en lenguajes compilados. Ryu implementa también una interfaz REST para interactuar con las aplicaciones.

Todo el código está disponible libremente a través de la licencia Apache 2.0. Ryu está totalmente escrito en Python. Ryu significa «flow» en japonés. Ryu posee gran cantidad de documentación, ordenada, y fácil de comenzar a utilizar.

MAESTRO

Maestro es un controlador OpenFlow multi-hilos cuya arquitectura es idéntica al diseño que se muestra en la figura 2.2.

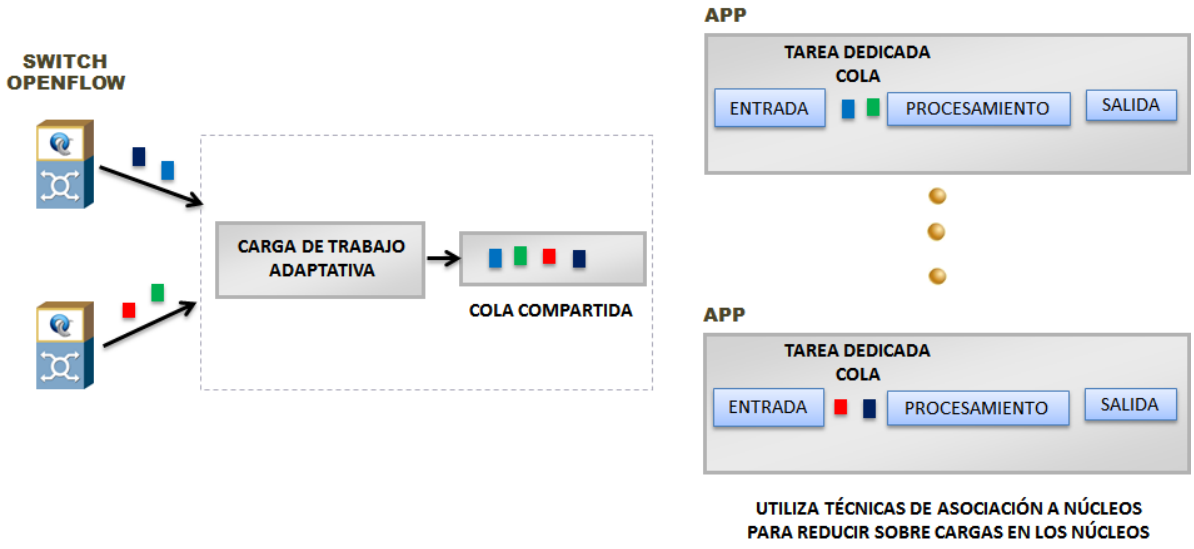


Figura 2.2. Cola adaptativa. Fuente: Autor basada en [83].

Los paquetes OpenFlow se reciben desde los sockets por el hilo principal y se colocan en una cola compartida. Se utiliza un sistema de procesamiento por lotes de paquetes, en el que el número de bytes a leer no es estático y depende de la carga de trabajo en ese momento. Maestro es el único controlador público disponible que utiliza el procesamiento de tareas por lotes con el objetivo de procesar múltiples solicitudes de flujo en una sola ejecución. Los paquetes hacia un mismo destino son agrupados y enviados utilizando una única llamada al socket del sistema.

2.3 Herramientas de simulación, desarrollo y depuración

En esta sección, se ofrece una visión general de las herramientas y entornos actualmente disponibles para desarrollar servicios y protocolos basados en SDN. Los software de simulación y emulación son de especial importancia para realizar prototipos y pruebas rápidas sin necesidad de dispositivos físicos costosos. Mininet [17] es el primer sistema que proporciona una manera rápida y fácil de realizar prototipos de redes así como evaluar protocolos y aplicaciones SDN. Una de las principales propiedades de Mininet es el uso de conmutadores OpenFlow basados en software en contenedores virtuales, proporcionando una semántica exacta a los conmutadores OpenFlow basados en hardware. Esto significa que los controladores o aplicaciones desarrolladas y probadas en el entorno emulado pueden (en teoría) desplegarse en una red habilitada para OpenFlow sin ninguna modificación. Los usuarios pueden emular fácilmente una red OpenFlow con cientos de nodos y docenas de conmutadores utilizando una sola computadora personal.

Las herramientas disponibles para este propósito puede dividirse en cuatro categorías: i) plataformas de simulación y emulación; ii) implementaciones de conmutadores mediante software; iii) soluciones de caja blanca; y iv) herramientas de depuración y solución de problemas [87].

Plataformas de simulación y desarrollo

EstiNet es una herramienta de software comercial que tiene las características de ser un simulador y un emulador al mismo tiempo con su propio reloj de simulación para controlar el tiempo de ejecución de la red simulada. Todos los programas de aplicación reales (incluyendo ping) que se ejecutan sobre él se activan por su reloj de simulación en lugar del reloj de tiempo real del sistema. Utiliza controladores OpenFlow y aplicaciones SDN reales, la pila de protocolos TCP/IP del kernel de Linux para generar resultados de las simulaciones con alta precisión [88], [89]. La herramienta utiliza los servidores de la empresa para ejecutar la simulación o los proyectos de emulación. Este servicio de nube, se conoce como simulación como un Servicio. EstiNet tiene buenas propiedades de simulación entre ellas son resultados precisos con una interfaz gráfica de usuario y animación de paquetes, junto con una buena presentación de las estadísticas de la simulación.

La capacidad de simular dispositivos OpenFlow también se han añadido al popular simulador ns-3 pero solo soporta la versión 0.89 del protocolo [88]. Se ha desarrollado un módulo externo basado en la biblioteca ofsoftswitch13 para este simulador que soporta la versión 1.3 de OpenFlow. Ns-3 es un simulador de eventos discretos, definidos en C++ o en scripts de Python, bajo licencia de software libre. Provee un ambiente de simulación para un rango amplio de protocolos de red y puede integrarse con una GUI denominada NetAnim. El proyecto ns-3 tiene como objetivo desarrollar un núcleo de simulación sólido, bien documentado, fácil de usar y depurar. Una de sus principales ventajas es que su infraestructura de software permite el desarrollo de modelos de simulación lo suficientemente objetivos como para simular una red real y posibilitar la interacción con sistemas reales. Los principales usuarios de este simulador lo emplean en la simulación de redes IP inalámbricas mediante modelos basados en los estándares IEEE 802.11 e IEEE 802.16, por las amplias prestaciones que ofrecen estos modelos, en comparación con los de otros simuladores.

Otro simulador es fs-sdn que posee un controlador SDN y un conmutador con soporte OpenFlow. Su principal objetivo es proporcionar una plataforma de simulación más realista y escalable en comparación con Mininet. Por último, STS es un simulador diseñado para permitir a los desarrolladores especificar y aplicar una variedad de casos de prueba, mientras que les permite examinar de forma interactiva el estado de la red. [3].

El OMNet ++ [90] es un simulador de eventos discretos que permite el desarrollo y prueba de modelos basados en SDN. Los proyectos orientados a SDN pueden ser integrados con OMNeT ++ utilizando componentes OpenFlow y un INET Framework.

OpenNet es un simulador/emulador de código abierto para las Redes de Área Local Inalámbricas Definidas por Software (Software Defined Wireless Local Area Network, SDWLAN por sus siglas en inglés) basado en Mininet y ns-3. Soporta todos los tipos de simulación que utilizan Mininet y ns-3, del primero toma el amplio soporte que posee para el trabajo con controladores y aplicaciones OpenFlow y del segundo utiliza los modelos inalámbrico/móvil. El empleo de estos dos simuladores tan potentes permite aprovechar las ventajas de cada uno y ponerlas en función de la simulación de una SDWLAN. OpenNet soporta hasta la versión 1.3 del protocolo OpenFlow.

Implementaciones de conmutadores mediante software

Un resumen no exhaustivo de los conmutadores que se utilizan para experimentos y el desarrollo de nuevos servicios en SDN son abordados a continuación.

Open vSwitch, es uno de los conmutadores de software más ampliamente implementados. Emplea una pila de protocolo OpenFlow que puede utilizarse tanto como un conmutador virtual en topologías de redes virtualizadas así como también en múltiples plataformas de hardware. El Open vSwitch es parte del kernel de Linux desde la versión 3.3 [87].

Ofsoftswitch13 se ejecuta en el espacio de usuario de Linux y también proporciona soporte para múltiples versiones de OpenFlow [91]. Es una implementación de un conmutador por software compatible con la versión 1.3 de OpenFlow construido sobre el conmutador OpenFlow/Stanford de referencia principalmente para la realización de experimentos. En la versión actual se encuentran disponibles los siguientes componentes: Ofdatapath (implementación del conmutador), Ofprotocol (canal seguro para la conexión del conmutador al controlador), Oflib (librería para convertir al formato 1.3) y la herramienta Dpctl.

El proyecto Indigo (IVS, Indigo Virtual Switch por sus siglas en inglés) es una implementación de código abierto OpenFlow que se puede ejecutar en una gama amplia de conmutadores físicos y que utiliza las características de hardware de los conmutadores Ethernet existentes para realizar conmutaciones a la velocidad de línea. El software se basa en la implementación OpenFlow de referencia de Stanford y actualmente admite todas las características requeridas en el estándar OpenFlow 1.0.

El PicOS desarrollado por Pica8, es un sistema operativo de red que permite la creación de redes flexibles y programables utilizando conmutadores implementados con cajas blancas basados en OpenFlow [92].

Pantou modifica un enrutador y punto de acceso inalámbrico comercial en un conmutador OpenFlow. El protocolo OpenFlow se implementa como una aplicación en la parte superior de la plataforma de OpenWRT[93].

Herramientas para la depuración y solución de problemas

Con STS se pueden simular los dispositivos de una red, permitiendo al mismo tiempo generar mediante programación diferentes pruebas y casos de uso [94]. Los usuarios pueden visualizar de forma interactiva el estado de la red, así como los cambios en tiempo real. Esta facilidad permite determinar automáticamente los eventos erróneos e identifican errores.

El Open vSwitch viene con un conjunto completo de herramientas para depurar el comportamiento de los conmutadores y de la red:

- ovs-vsctl: Se utiliza para configurar la base de datos del conmutador (conocida como ovs-db.).
- ovs-ofctl es una herramienta de línea de comandos para la gestión de los conmutadores OpenFlow.
- ovs -appctl: Se utiliza para consultar y controlar los procesos Open vSwitch.

- La herramienta de línea de comando Dpctl (DataPathController) permite enviar comandos básicos para insertar, eliminar, modificar, inspeccionar entradas en las tablas de flujo de un conmutador OpenFlow y se comunica con éste mediante puertos previamente configurados. Esta es una herramienta muy útil para gestionar directamente los conmutadores sin necesidad de un controlador y generalmente se utiliza para analizar en detalle el proceso de modificación de las entradas en las tablas de flujo. DPCTL envía mensajes OpenFlow a un conmutador previa orden a través de la línea de comandos. Útil para la visualización de las estadísticas de los puertos y de flujos de los conmutadores, además de insertar manualmente las entradas de flujo en caso de ser necesario. Es útil para la depuración debido a permite la visualización del estado de los flujos así como los contadores. En la tabla que aparece a continuación son presentados los principales comandos de la herramienta DPCTL que será utilizado en experimentos posteriormente.

IPERF es otra herramienta de código abierto que es utilizada para medir la velocidad, throughput y calidad de un enlace de red realizando pruebas de comprobación sobre el mismo. La herramienta utiliza los protocolos TCP y UDP para realizar las mediciones. Con TCP mide la velocidad y el throughput del enlace, y con UDP la pérdida de paquetes y el jitter. Una de las principales ventajas de IPERF es que puede medir los parámetros anteriormente mencionados entre dos host que se encuentren geográficamente separados [95]. Iperf fue desarrollado por el Distributed Applications Support Team (DAST) en el National Laboratory for Applied Network Research (NLANR) y está escrito en C++.

Permite al usuario ajustar varios parámetros que pueden ser usados para hacer pruebas, ajustar u optimizar una red. Puede funcionar como cliente o como servidor y puede medir el rendimiento entre los dos extremos de la comunicación, unidireccional o bidireccionalmente. Es software de código abierto y puede ejecutarse en varias plataformas incluyendo Linux, Unix y Windows [95].

OFTrace es una herramienta para la depuración y análisis del protocolo OpenFlow. A partir de un archivo con formato libpcap (resultado de ejecutar Tcpdump o Wireshark) produce estadísticas útiles sobre la sesión OpenFlow. Ejemplos de aplicaciones Oftrace pueden ser: ofstats, la cual graba el retardo de procesamiento del controlador, es decir, la diferencia de tiempo entre que un controlador recibe un mensaje OFPT_PACKET_IN y envía el correspondiente OFPT_PACKET_OUT u OFPT_FLOW_MOD para la modificación de las entradas de flujo, y ofdump, que lista los mensajes de control OpenFlow con marcas de tiempo.

OFlops [96] tiene como objetivo implementar mediciones básicas del protocolo OpenFlow que permita a los desarrolladores especificar y analizar las capacidades de los dispositivos OpenFlow. Esta herramienta comprueba el desempeño de un conmutador OpenFlow, tanto software como hardware, emulando un controlador conectado al conmutador. OFlops combina las mediciones en el plano de control con las mediciones en el plano de datos. Su filosofía de diseño es lograr poca interacción con un dispositivo OpenFlow sobre múltiples canales de datos para no introducir demoras adicionales de procesamiento.

La herramienta OFtest posee una colección de pruebas basada en Python que permiten evaluar las funcionalidades básicas de los conmutadores OpenFlow. Soporta las versiones de la 1.0 a la 1.3 del protocolo OpenFlow. Esta herramienta ejecuta un servidor OFtest que se conecta al conmutador y coordina los comandos OpenFlow para el monitoreo [97].

Wireshark es el analizador de protocolos más utilizado que hay actualmente, permite la captura y búsqueda interactiva del tráfico que atraviesa una red, actualmente se considera un estándar a nivel no solo educativo sino también industrial y comercial. Para analizar el tráfico de control de OpenFlow se debe ejecutar "$ sudo wireshark &" y posteriormente se debe el protocolo OpenFlow o mensajes y parámetros deseados. La funcionalidad que provee es similar a la de tcpdump de Linux, pero añade una interfaz gráfica y muchas opciones de organización y filtrado de información. Así, permite ver todo el tráfico que pasa a través de una red (usualmente una red Ethernet, aunque es compatible con algunas otras) estableciendo la configuración en modo promiscuo. También incluye una versión basada en texto llamada tshark. Permite la captura datos de una red en tiempo real y también a partir de

un archivo de captura salvado en disco. Se puede analizar la información capturada, a través de los detalles y sumarios por cada paquete.

Nice es otra herramienta que realiza una prueba automatizado con el objetivo de identificar posibles problemas y errores en el protocolo OpenFlow [87].

OFRewind [98] es otra herramienta que permite la depuración de eventos de red, tanto en el plano de control y de datos.

Herramientas para la evaluación

Para evaluar el desempeño de un controlador los profesores Rob Sherwood y Kok-Kiong crearon la herramienta, cbench, que posteriormente se integró en las distribuciones de MiniNet.

Cbench (Controller Benchmarker, por sus siglas en inglés) es la herramienta más utilizada para la evaluación de controladores OpenFlow [75], [99]. Es una herramienta de prueba de esfuerzo que opera en los modos latencia o rendimiento.

Cbench emula un número configurable de conmutadores OpenFlow los cuales se comunican con un controlador OpenFlow para medir diferentes aspectos del rendimiento y latencia del mismo. La prueba que realiza la herramienta se basa en el envío de mensajes de arribo de nuevos flujos (mensajes OFPT_PACKET_IN) por parte de los conmutadores al controlador y el monitoreo de la respuesta (mensajes OFPT_PACKET_OUT o OFPT_FLOW_MOD). En la siguiente figura se puede observar los mensajes intercambiados entre un conmutador y el controlador bajo evaluación.

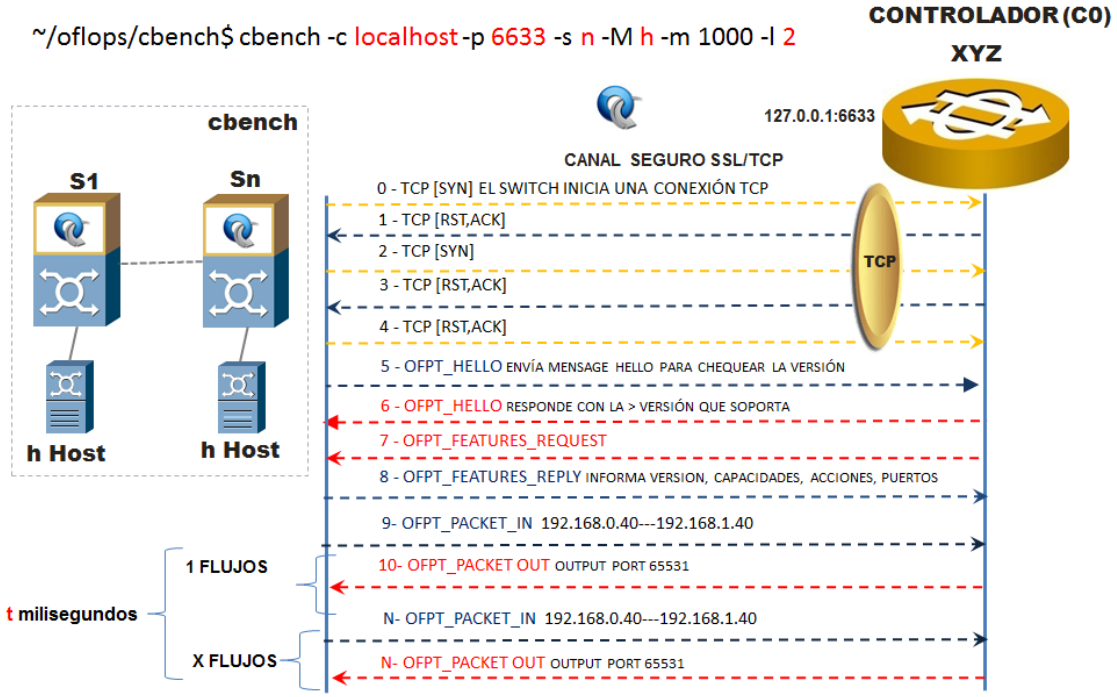


Figura 2.3. Intercambio de señalización entre cbench y el controlador. Fuente: Autor.

Modo latencia: En modo latencia cada conmutador realiza una petición de nuevo flujo (OFPT_PACKET_IN) y espera por la respuesta (OFPT_PACKET_OUT u OFPT_FLOW_MOD) antes de solicitar la próxima petición. Este modo mediría el tiempo de proceso del controlador en condiciones de bajo tráfico.

Modo throughput: En modo throughput, cada conmutador realiza una petición de nuevo flujo (OFPT_PACKET_IN) y mantiene tantas peticiones pendientes como sus buffers le permiten. Este modo es utilizado para evaluar la razón máxima de establecimiento de flujos que un controlador puede mantener.

Existen otros software para evaluar el desempeño de controladores que utilizan Cbench como librería principal, entre ellos se puede destacar KCbench, que soporta la versión 1.3 de OpenFlow pero que solo permite evaluar al controlador Mul, y WCbench, que actualmente solo permite evaluar al controlador OpenDayLight. Cbench es una de las herramientas más útiles que se integran a las distribuciones de Mininet y sus opciones se pueden observar mediante el comando $: cbench --help. Esta herramienta tiene la ventaja que permite definir la cantidad de iteraciones, el tiempo de duración de cada iteración y que muestra una estadística de las pruebas realizadas con los valores mínimos, máximos, promedio y la desviación estándar.

HCPROBE está escrita en Haskell, y permite crear con facilidad, escenarios para pruebas de control SDN. Es capaz de simular un gran número de conmutadores y host conectados a un controlador. Empleando Hcprobe se pueden analizar varios índices de operación del controlador de forma flexible. Con esta herramienta se pueden especificar patrones para generar mensajes OpenFlow (incluidos los malformados) y establecer perfiles de tráfico, entre otros. Sus características principales son:

- Generación de paquetes OpenFlow y de tablas de red;
- Implementación en un lenguaje de alto nivel, lo que hace que sea más fácil de extender;
- Existencia de una API para el diseño de pruebas personalizadas;

En la tabla 2.1 que se muestra a continuación se resumen las principales herramientas organizadas por categoría.

Tabla 2.1. Plataformas de simulación y herramientas de depuración.

CATEGORÍA	PROPÓSITO	SOFTWARE Y HERRAMIENTAS
Emulación y simulación	Simulación de topologías de red, así como una referencia para la simulación de eventos de red	Mininet, ns-3, OMNeT++
Switch y plataformas de software	Una plataforma de sotware para probar y validar el comportamiento de los switch así como el funcionamiento de la interface sur.	Open vSwitch, ofsoftswitch13, Indigo, Pica8 PicOS, Pantou
Depuración y solución de problemas	Conjunto de herramientas especializadas en la depuración del comportamiento de los swtich y controladores SDN	STS, Open vSwitch, NICE, OFTest, Anteater, VeriFlow, OFRewind, NDB, Wireshark

2.3.1 Descripción del emulador MiniNet

Mininet fue creado por un grupo de profesores en la Universidad de Stanford para ser utilizado como una herramienta para la investigación y enseñanza de tecnologías de redes. Es un software construido en lenguaje Python que a nivel de código está compuesto por un conjunto de librerías, algunas de sus clases están diseñadas para ofrecer funcionalidades de consola facilitando el uso del emulador con respecto a los usuarios. Al iniciar el emulador básicamente se invocan constructores que definen parámetros prederminados para la prueba. Además, Mininet puede personalizarse por medio del uso de Scripts en código Python, vinculado a las librerías del propio software, facilitando la creación de las redes a virtuales en base a líneas de código. Después de iniciar la emulación

Mininet entra en un estado captura de comandos, esto es posible debido a una clase construida para tal fin con funcionalidades como las que se verán en la sección

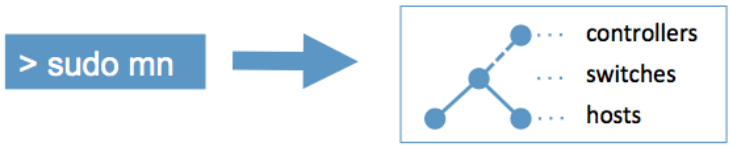


Figura 2.4. La forma más fácil de crear una red SDN.

Mininet es una plataforma que permite crear redes virtuales a gran escala de forma rápida y eficiente gracias al uso de una característica conocida como virtualización ligera (light-weight virtualization).

Esta plataforma es muy práctica para el estudio de SDN, debido a que permite implementar nodos con protocolo OpenFlow. Entre las características de Mininet se encuentran:

- Flexibilidad: Topologías y características nuevas se pueden configurar por software usando Lenguajes de programación y SO comunes.
- Aplicabilidad: Correctas implementaciones (prototipos) se pueden implementar en redes basadas en Hardware sin cambiar su código fuente.
- Interactividad: Administración y simulación ocurren en tiempo real.
- Escalabilidad: En un ordenador se pueden crear redes complejas y comprobar la escalabilidad de los sistemas.

El simulador de redes funciona sobre Linux y permite crear hosts, enlaces, controladores y *conmutadores* que entienden el protocolo *OpenFlow*. Iniciado desde una máquina virtual, permite crear una red SDN completa en segundos, además de permitir el acceso de un controlador SDN externo en el momento de la creación. Las topologías de redes se pueden hacer a medida, tan complejas como se desee, a través de scripts en el lenguaje de programación Python. Mininet permite acceder a cada host individualmente y ejecutar las aplicaciones instaladas en el sistema desde estos. De esta manera, se pueden monitorizar las conexiones de cada host mediante la herramienta Wireshark y también se pueden crear servidores webs dentro de cada host.

Todo esto hace que Mininet sea el simulador de redes SDN por excelencia. Dado que es un proyecto de código abierto, está en constante evolución, permitiendo además que cualquier usuario pueda modificar el código con total libertad. La estructura de la línea de comando de Mininet aparece en la figura 2.5.

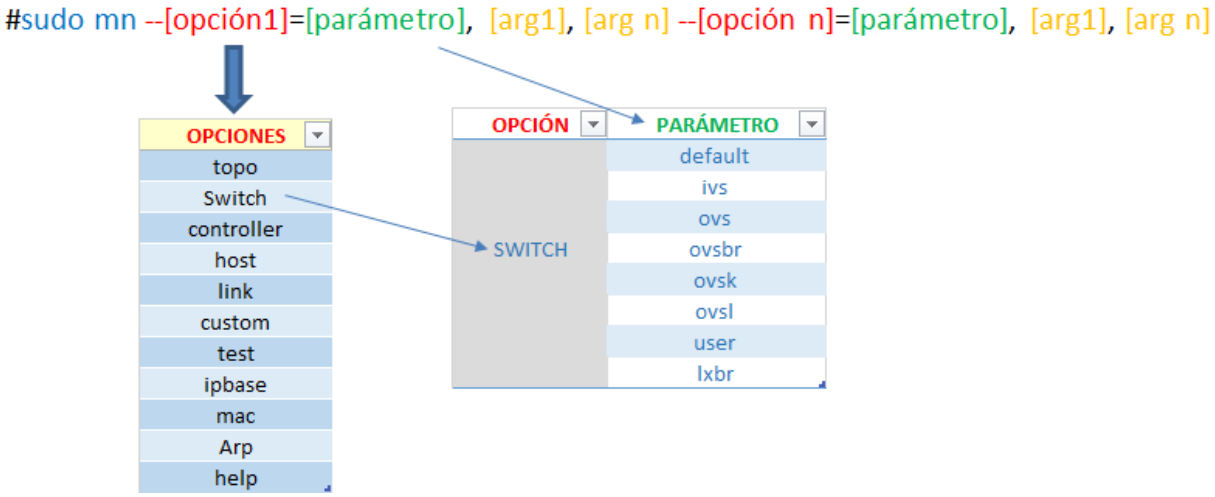


Figura 2.5. La estructura de la línea de comando de MININET. Fuente: Autor.

Mininet es un emulador de red que ejecuta una colección de dispositivos finales, conmutadores, enrutadores y enlaces en un solo núcleo de Linux. Se utiliza la virtualización ligera para hacer que un solo sistema parezca una red completa. Los programas que se ejecutan pueden enviar paquetes a través de lo que parece ser una interfaz de Ethernet real, con una velocidad de enlace y con retardo. Los paquetes se procesan simulando un conmutador real, un enrutador o middlebox, con una determinada cantidad de colas. Cuando dos programas, como iperf cliente y el servidor, se comunican a través MiniNet, el rendimiento medido debe coincidir con el de dos máquinas nativas más lentas. En resumen, los dispositivos virtuales de MiniNet, conmutadores, enlaces y los controladores se crean utilizando software en lugar de hardware, en su mayor parte el comportamiento es similar a los elementos de hardware.

El comando "sudo mn", ejecutado desde la terminal de Ubuntu inicia el emulador Mininet. Es posible personalizar la topología y el funcionamiento de la red por medio de opciones adicionales.

Las OPCIONES disponibles son:

--*help:* muestra en pantalla un listado de las posibles opciones que se pueden invocar con el comando sudo mn.

--*switch=[PARAMETRO]:* Permite invocar un tipo de conmutador:

- default: El valor predeterminado utiliza un conmutador del tipo Open vSwitch.
- ivs: IVSSwitch, conmutador OpenFlow que utiliza tecnología Indigo Virtual Switch, y requiere instalación previa.
- ovs: Open vSwitch, usa tecnología Open vSwitch compatible con OpenFlow.
- ovsbr: OVSBridge, usa un conmutador Ethernet implementado a partir de Open vSwitch, soporta STP, como argumento utiliza: stp=1 para activar el protocolo Spanning tree.
- ovsk: usa Open vSwitch en modo kernel para cada conmutador.
- ovsl: Open vSwitch que trabaja en el espacio del kernel tradicional, actualmente solo trabaja con el espacio de nombres principal Root.
- user: conmutador con implementación OpenFlow invocado desde el espacio de usuario, es decir, externo al emulador Mininet.
- lxbr: Linux Bridge, conmutador implementado en código abierto, como argumento usa: stp=1 para activar el protocolo Spanning tree.

--host=[PARAMETRO]: Limita el ancho de banda del procesador de un host virtual:

- cfs: Planificador de uso de recursos de procesamiento en Linux.
- rt: Planificador POSIX real-time (Interfaz de sistema operativo portable de tiempo real), este planificador ha sido deshabilitado por defecto en todos los kernel Linux, para esto se debe habilitar RT_GROUP-SCHED.

--*controller*=[PARAMETRO]: Permite invocar un tipo de controlador:

- default: Utiliza el controlador predeterminado del sistema y es compatible con el protocolo OpenFlow.
- none: deshabilita el uso de un controlador.
- nox: habilita el controlador NOX, requiere instalación previa.
- ovsc: Utiliza el controlador de prueba de Open vSwitch.

- remote: Permite la utilización de un controlador remoto compatible con el protocolo OpenFlow. En los argumentos es necesario indicarle la dirección IP y puerto.
- ryu: Utiliza el controlador Ryu, Es necesario que el controlador Ryu esté instalado en el sistema.

--*link*=[PARAMETRO]: Permite variar parámetros como ancho de banda y latencia de los enlaces:

- default: configura un enlace con ancho de banda, latencia y pérdida de paquetes predeterminado.
- tc: Personaliza las interfaces por medio de la utilidad control de tráfico permitiendo especificaciones de límites de ancho de banda, latencia, perdidas y máxima longitud de colas manejadas. Usa argumentos como:

  bw=[ANCHO_BANDA],delay=[TIEMPO],loss,[PORCENTAJE].

--*topo*=[PARAMETRO]: Permite cambiar el tamaño y el tipo de topología a emular:

linear: Genera una topología de k conmutadores en serie. Los k host se conectan a cada conmutador.

2.4 Métricas relevantes para las SDN

Desde que surgió la idea de un control lógicamente centralizado surgió la necesidad de evaluar el comportamiento de la red es este nuevo escenario. Existen muchas métricas definidas para las redes tradicionales que igualmente son válidas para las SDN, aunque también se han definido por la ONF métricas que son específicas para ese tipo de redes, relacionadas con los mensajes del protocolo OpenFlow, con el controlador OpenFlow o con los conmutadores OpenFlow. También el IETF ha definido algunos términos con la idea de unificar criterios para medir y evaluar el desempeño de controladores SDN sin importar los protocolos y servicios soportados. A continuación son mostradas las métricas más importantes para evaluar el desempeño, escalabilidad, confiabilidad y seguridad de controladores SDN independientemente de los protocolos “northbound y southbound” implementados en la red [100].

Métricas de desempeño para el controlador

Tiempo de descubrimiento de la topología de la red: Es el tiempo empleado por el controlador para descubrir los nodos de la topología de la red así como las conexiones entre los mismos. El descubrimiento de la topología de la red es un elemento clave para para la provisión y gestión global de la red [101].

Tiempo de procesamiento de mensajes asincrónicos: Es el tiempo empleado por el controlador en procesar mensajes asincrónicos. Para que una red SDN puede realizar una provisión dinámica es necesario medir la rapidez con la que el controlador responde a un evento desencadenado por la red. El evento pudiera ser la notificación por parte de los dispositivos de la capa de infraestructura del arribo de un nuevo flujo, enlace interrumpido, etc. [100].

Tasa de procesamiento de mensajes asincrónicos: Mide el número máximo de mensajes asincrónicos que un controlador puede procesar durante un tiempo determinado. Como SDN garantiza una flexibilidad y agilidad en la red es necesario medir el número de evento que un controlador puede manejar. En esta prueba todos los dispositivos de la capa de infraestructura envían al controlador todos los mensajes asincrónicos que puedan durante el tiempo de prueba. La unidad de medida es mensajes procesados por segundo[101]. Los mensajes asíncronos son iniciados por el conmutador sin que el controlador los solicite. Los conmutadores envían estos mensajes al controlador para notificar la llegada de un paquete (OFPT_PACKET_IN), un cambio de estado en el conmutador (PORT_STATUS) o un error. Esta métrica será utilizada para evaluar el desempeño de una selección de controladores SDN basados en el protocolo OpenFlow.

Tiempo de provisión de entradas de flujo en modo reactivo: Es el tiempo empleado por el controlador en configurar una entrada de flujo en modo reactivo. Esta métrica es importante para medir cuán rápido el controlador configura las entradas de flujos en todos los elementos de la capa de infraestructura involucrados en una conexión entre dos

elementos de la red. Para calcularlo es necesario medir la diferencia entre la instalación de la primera y última entrada de flujo [100].

Tiempo de aprovisionamiento de entradas de flujo en modo reactivo: Es el tiempo empleado por el controlador en configurar una entrada de flujo de forma proactiva .Esta métrica es importante para medir cuán rápido el controlador configura las entradas de flujos en todos los elementos de la capa de infraestructura involucrados en una conexión entre dos elementos de la red. Usualmente la recepción de la petición de instalación de la entrada de flujo en modo proactivo es a través de la interface northbound del controlador. Para calcularlo es necesario medir la diferencia entre la instalación de la primeria y última entrada de flujo. La unidad de medida es milisegundos [100].

Razón de aprovisionamiento de entradas de flujo en modo reactivo: Mide el número máximo de entradas de flujos independientes que un controlador puede de forma concurrente establecer entre un nodo de origen y otro de destino de forma reactiva y durante el tiempo de prueba. Esta prueba es muy importante y su principal objetivo es medir cuantos flujos extremo a extremo un controlador puede configurar en la capa de infraestructura. Esta métrica se expresa en caminos configurados por segundo [100].

Razón de aprovisionamiento de entradas de flujo en modo proactivo: Mide el número máximo de entradas de flujos independientes que un controlador puede de forma concurrente establecer entre un nodo de origen y otro de destino de forma proactiva y durante el tiempo de prueba. Esta prueba es muy importante y su principal objetivo es medir cuantos flujos extremo a extremo un controlador puede configurar en la capa de infraestructura. Esta métrica se expresa en caminos configurados por segundo.

Tiempo de detección de cambio en la topología de la red: Es el tiempo requerido por el controlador para detectar un cambio en la topología de la red. Con el objetivo de que el controlador pueda recuperar rápidamente y de forma automática el servicio en caso de fallo en algún elemento de red; es crítico medir cuán rápido el controlador puede los eventos de cambio de estado de la red. Para computar esta métrica se calcula el tiempo dese la llegada del evento de cambio de estado de la red hasta que el controlador inicia el proceso de redescubrimiento de la topología. Esta métrica se expresa en milisegundos [100].

Escalabilidad del controlador SDN

La escalabilidad es el número máximo de dispositivos y/o *conmutadores* que un controlador puede manejar sin que disminuya su rendimiento. Idealmente, el rendimiento de un controlador no debería verse afectado por el número de *conmutadores* conectados a él. Sin embargo, el aumento de los hilos de ejecución, la dinámica del protocolo TCP y las tareas que se ejecuten en el núcleo del controlador, son factores que pueden degradar su rendimiento si existe un gran número de *conmutadores* altamente activos [102].

Capacidad de sesiones de control: Mide el número máximo de sesiones de control que un controlador puede mantener. Esta métrica es importante para medir los requerimientos de recursos del sistema y ancho de banda. En esta prueba cada elemento de la capa de infraestructura establece una sesión de control con el controlador hasta que falle. El número de sesiones que se establecieron con éxito proporciona la capacidad de sesiones de control [100].

Descubrimiento del tamaño de la red: Mide el tamaño de la red (cantidad de nodos, enlaces, y host) que un controlador puede descubrir. Para una planificación óptima de la red es clave medir el tamaño máximo de la red que un controlador puede descubrir. En esta prueba se incrementa el número de dispositivos para determinar el número máximo de nodos que un controlador puede descubrir.

Capacidad de la tabla de renvío: El número máximo de entradas de flujo que un controlador puede gestionar en su tabla de renvío sin descartar o inundar entradas. En esta prueba se le envían al controlador nuevas entradas de flujo en los modos proactivo o reactivo hasta que se llene la tabla de renvío [100].

Seguridad de un controlador SDN

Manejo de excepciones: Permite determinar el efecto del manejo de paquetes con errores y notificaciones en las pruebas de desempeño. Para esta prueba es obligatorio haber realizado primero todas las pruebas de desempeño descritas anteriormente. Mediante un software se le envían al controlador paquetes con errores y se activa al mismo tiempo las pruebas de desempeño. La diferencia entre esta prueba y la de desempeño mostrará el efecto que introduce en manejo de excepciones en el desempeño del controlador [100].

Manejo de ataques de denegación de servicio: Esta prueba tiene el objetivo de determinar el efecto manejar ataques de denegación de servicio (DoS) en el desempeño y escalabilidad del controlador. Para esta comprobación es necesario primero realizar las pruebas de desempeño y escalabilidad definidas en las secciones anteriores. La diferencia entre esta prueba, las de desempeño y escalabilidad mostrará el efecto que introduce en manejo de ataques de denegación de servicio en el desempeño del controlador [100].

En [102] se plantea también la necesidad de computar la cantidad de mensajes TSL fallidos como métrica para comprobar la seguridad de un controlador.

Confiabilidad

Tiempo de conmutación en caso de fallo del controlador: Cuando el controlador se encuentra trabajando en modo redundancia. Es el tiempo que demora la conmutación del controlador activo para el de reserva en caso de fallo en el controlador activo. Esta prueba determina el impacto en la provisión de nuevos flujos cuando el controlador activo falla.

Tiempo para volver a encaminar el tráfico: Es el tiempo empleado por el controlador en volver a encaminar el tráfico en caso de algún fallo en las rutas establecidas previamente. Esta prueba determina la habilidad del controlar para redefinir las entradas de flujo en caso de fallo de red.

En esta prueba se asume que la topología de red soporta caminos redundantes entre un origen y un destino y que el controlador no tiene pre configurada la ruta redundante.

Métricas cuantitativas para el protocolo OpenFlow

Entre las métricas más importantes que pueden ser analizadas relacionadas con los mensajes OpenFlow se destacan:

Tiempo de respuesta del protocolo OpenFlow: Tiempo transcurrido entre la realización de una petición *OpenFlow* del *conmutador* al controlador o del controlador al *conmutador*, y el arribo de la respuesta. Los tiempos de respuesta y la variación en los tiempos de respuesta se pueden agrupar y categorizar por: tipo de mensaje, *conmutador OpenFlow*, controlador *OpenFlow* (en el caso de que hayan más de uno) y por dirección (si es del controlador al *conmutador* o del *conmutador* al controlador) [102].

Tiempo de activación del protocolo: Tiempo transcurrido entre una petición OpenFlow del conmutador al controlador, y el respectivo cambio en la entrada de flujo. Los tiempos de activación y la variación en los tiempos de activación pueden ser agrupados por tipo de mensaje o por conmutador OpenFlow [100].

Duración de una entrada: Tiempo que transcurre entre la instauración de una entrada en una tabla de flujo del conmutador y la expiración de la misma, lo que se puede obtener mediante los campos *idle_timeout* y *hard_timeout* de cada entrada. La duración de una entrada puede agruparse por: entrada de flujo, puerto OpenFlow, cola OpenFlow, entrada meter OpenFlow, etc. [102].

Métricas cuantitativas para un conmutador OpenFlow

Para cada conmutador OpenFlow se pueden extraer las siguientes métricas:

Disponibilidad: Porcentaje del tiempo que el *conmutador* está activo y disponible para comunicarse con el controlador OpenFlow, y porcentaje del tiempo que el *conmutador* está activo y disponible para realizar funciones del plano de datos[102].

Confiabilidad: Tiempo promedio entre las fallas de un *conmutador*, tiempo promedio de recuperación de dichas fallas (que incluye un factor asociado con los recursos humanos) y el tiempo promedio para reiniciar el *conmutador*.

Capacidad: Porcentaje de los recursos del *conmutador* que se pueden utilizar (por ejemplo en función de la cantidad máxima de paquetes que puede almacenar a la vez).

Escalabilidad: Cantidad de tablas de flujo que un conmutador OpenFlow puede soportar.

Exactitud: Porcentaje de paquetes conmutados exactamente como se especifica en las tablas de flujo del conmutador.

Seguridad: El número de mensajes TLS handshake fallidos.

2.5 Conclusiones del capítulo

En este capítulo fueron abordados los principales controladores SDN con soporte para el protocolo OpenFlow, sus arquitecturas; así como los elementos claves para seleccionar un sistema operativo de red. OpenFlow es una tecnología prometedora para la creación de redes programables con un enorme potencial para futuras investigaciones y despliegues. Los controladores SDN con soporte para el protocolo OpenFlow más populares entre los investigadores son los de código abierto; destacándose la utilización en artículos científicos: OpenDayLight (ODL), FLOODLIGHT, ON.Lab (ONOS), BEACON, NOX, POX, RYU, y MAESTRO.

Entre los software para la creación de redes SDN se destaca Mininet con alta popularidad dentro de la comunidad científica dado que es un proyecto de código abierto, que está en constante evolución, y permite además que cualquier usuario modifique el código con total libertad. Siendo esta última la herramienta más apropiada para el trabajo con las SDN.

Los principales elementos a tener en consideración para la selección de los controladores SDN son: soporte del protocolo OpenFlow, aplicaciones, virtualización de red, funcionalidad de la red, escalabilidad, procesamiento, rendimiento, programación de red, confiabilidad, seguridad de la red, monitorización centralizada y visualización, experiencia del fabricante y el soporte de plataformas.

La evaluación del desempeño de las SDN es un proceso complejo que depende de muchos factores como la topología, software de los conmutadores virtuales, el controlador, los protocolos hacia el sur, la carga de tráfico de la red, la plataforma de cómputo, etc. La estandarización de todos los aspectos del desempeño de las SDN aún constituye tema de estudio para la ONF y el IETF. Existen muchas métricas definidas para las redes SDN, pero en la bibliografía revisada las más populares son: la tasa de procesamiento de mensajes asincrónicos y la escalabilidad del controlador.

CAPÍTULO 3. EVALUACIÓN DEL DESEMPEÑO DE LAS SDN

3.1 Estrategias para la implementación y metodología

Con el fin de evaluar las redes programables y poder llevar a cabo experimentos en la arquitectura SDN, es esencial tener un entorno para la realización de pruebas. El objetivo principal de estos experimentos es el de evaluar y comprender el comportamiento de las redes programables. Para lograr este objetivo, se utiliza una red real con el fin de emular este tipo de redes así como la presentación de diferentes casos de uso o demostraciones. Las diferencias entre una emulación y una red real son; mientras que en redes reales el costo y la cantidad de equipos es limitado, en el entorno emulado no existe tal problema, pero por el contrario pueden surgir otros inconvenientes como por ejemplo alteración de los resultados; si el dispositivo que ejecuta la emulación no tiene suficiente capacidad de procesamiento.

Como se mencionó al principio objetivo del presente trabajo es implementar una plataforma de pruebas para evaluar el desempeño de las SDN basadas en el protocolo OpenFlow.

La implementación de la plataforma requirió primero de la selección de tres escenarios (ver ANEXO 5) que tuvieran diferencias en cuanto a procesador, memoria instalada, ediciones de Windows, diferentes tipo de hipervisores. El sistema operativo huésped y la función de activación de vitualización se mantuvieron idénticos en los tres escenarios con el objetivo de no crear diferencia en el sistema operativo donde se ejecuta el controlador. Posteriomente se descarga de internet el sistema operativo Ubuntu 14.04 y se procede a la actualización del mismo desde el repositorio. Seguidamente se realiza la selección de los controladores, el emulador, la versión del protocolo, la selección e instalación del entorno de desarrollo integrado (IDE, por sus siglas en inglés) para la programación en Python.

La selección de los controladores

La selección de los controladores se realizó siguiendo los criterios expuestos en el capítulo dos y fundamentalmente se basaron en los siguientes criterios: soporte del protocolo OpenFlow, controladores de código abierto, el ecosistema de aplicaciones y la madurez de las API NBI, arquitectura, lenguajes de programación en los que fueron realizados, la compatibilidad con equipos de red, la existencia de interfaces para la programación del controlador y la capacidad de procesamiento. También fue importante el acceso a la documentación de cada controlador de forma tal que posibilitara la instalación, configuración, programación y evaluación. Finalmente los controladores seleccionados para la evaluación fueron: POX, RYU, OVS, ONOS, FLOODLIGHT y OPENDAYLIGHT. Las principales características de estos controladores aparecen en los anexos.

La selección del emulador

Mininet fue seleccionado primero por ser de código abierto y después porque fue creado pensando en SDN y permite crear redes con host, conmutadores, controladores y enlaces virtuales. El emulador posee muy buena documentación sobre todo la API de programación que fue en la que se desarrolló el primer caso de uso que es la comparación de la latencia en una red tradicional con una SDN.

La selección de la versión del protocolo OpenFlow

El protocolo OpenFlow constituye la base de las redes abiertas definidas por software basadas en estándares. OpenFlow 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 ONF se encarga de la definición del estándar.

El protocolo OpenFlow ha evolucionado durante el proceso de estandarización de la ONF, a partir de la versión 1.0, donde sólo existen 12 campos coincidentes y una sola tabla de flujo a la última versión que cuenta con varias tablas, más de 41 campos así como nuevas funciones [67]. En esta investigación fue seleccionada la versión 1.0 del protocolo porque es soportada por Mininet y los controladores elegidos.

La implementación de la plataforma de pruebas

Finalmente a la máquina virtual con el sistema operativo Ubuntu 14.04 se le instaló el emulador Mininet, el analizador de protocolos WIRESHARK con soporte para el protocolo OpenFlow; así como los controladores POX, RYU, OVS, ONOS, FLOODLIGHT y OPENDAYLIGHT. En todos los casos para la instalación y configuración de cada uno de los elementos se siguieron las guías y procedimientos encontrados en la documentación. El entorno de desarrollo integrado escogido para programar en Python fue el Pycharm porque es multiplataforma, proporciona análisis de código, depuración gráfica, integración con VCS / DVCS y soporte para el desarrollo web con Django, entre otras bondades.

Debido al carácter aún novedoso de las SDN, se hace complejo el proceso de investigación y evaluación de esta arquitectura, incluso mediante la simulación. Tomó mucho tiempo de esta investigación completar la puesta a punto de Mininet y todos los controladores anteriormente mencionados. Inicialmente cuando fueron analizadas las potencialidades de la herramienta de evaluación cbech, se pudo comprobar que era posible automatizar las pruebas y añadir otras métricas como la utilización de CPU, consumo de memoria programando en el bash de Linux e interactuando con cbench. Es por ello que adicionalmente se realiza:

- El desarrollo de varias aplicaciones dirigidas a realizar pruebas de escalabilidad en Mininet a diferentes tipos de topologías; además de, la evaluación de controladores SDN de forma automática.
- La implementación de una aplicación SDN que permite en tiempo real detectar intrusos en la red así como aplicar políticas de acceso a la red en todo su conjunto.
- Utilizando la aplicación de seguridad fue implementado un escenario que permite evaluar el impacto de esta en la latencia de la red y la tasa de transferencia de datos cuando aumenta el tamaño de la red SDN.

Como ya se ha analizado, la simulación es una de las técnicas más utilizadas en el mundo de la investigación y el desarrollo de las SDN debido, entre otras, al carácter programable de esta arquitectura. En la presente investigación se trabajó en función de responder las siguientes interrogantes científicas:

¿Cuánto será la latencia de una red SDN en comparación con una red tradicional?

¿Cuán grande puede ser una red SDN creada en el emulador Mininet?

¿Cuánto será el tiempo de compilación de toda la red cuando se utiliza el emulador Mininet?

¿Cuál es el conmutador con el que se obtiene el menor tiempo de compilación de la red?

¿Influye la topología en el tiempo de compilación de la red?

¿Cuál será la plataforma de cómputo en la que se obtienen los menores tiempos de compilación de la red?

¿Cómo implementar una aplicación SDN que permita detectar intrusos y controlar el acceso a los recursos de la red?

¿Cuánto será el impacto en la latencia y la tasa de transferencia de datos en la red cuando se aplican políticas de seguridad?

3.2 Caso de uso: SDN vs Red tradicional

Desde el surgimiento del modelo de conmutador OpenFlow ha sido necesaria una comparación en términos de retardo con los conmutadores tradicionales. En esta sección se evalúa el comportamiento de las SDN respecto a una red tradicional analizando la métrica de latencia.

¿Cuánto será la latencia de una red SDN en comparación con una red tradicional?

El objetivo principal de esta prueba es demostrar que las redes SDN poseen igual o menor latencia que las redes tradicionales.

La latencia es una métrica importante a considerar en las redes actuales, especialmente si se utiliza para transportar datos de aplicaciones que son sensibles al retardo o el jitter. Una comunicación de voz de buena calidad requiere que la red tenga un valor de latencia menor de 50 ms [103].

En esta sección se realizará un estudio sobre un escenario llamado “red tradicional” conformado por tres *conmutadores* que utilizan la tecnología LINUX BRIDGE más tres máquinas virtuales realizando las funciones de *host* h1,h2,h3. Todos los elementos anteriores se encuentran conectados en una topología lineal donde cada host se conecta a un conmutador y los conmutadores están conectados linealmente entre ellos.

El otro escenario llamado “red SDN” está conformado por tres conmutadores del tipo ovs-vswitch, el controlador de referencia de MiniNet y tres máquinas virtuales que realizan la función de host h1, h2, h3 igual que en el escenario anterior.

Los dos escenarios fueron implementados mediante programación en Python y utilizando el API de Mininet. En un mismo script se encuentran las dos topologías y el usuario elige por la línea de comandos cuál de las dos es el que desea probar.

Prueba de retardo en una red tradicional

Con el objetivo de implementar la red anteriormente mencionada será necesario crear en el emulador MININET una topología personalizada que tenga en cuenta la utilización de un conmutador del tipo LINUX BRIDGE. Por lo tanto uno de los primeros requisitos es instalar en Ubuntu 14.04 bridge-utils además de estar familiarizado con la API de Mininet.

Para instalar Linux bridge:

```
$ sudo apt-get install bridge-utils
```

Linux bridge ha sido una tecnología de conmutación concebida antes de OVS y ofrece un modelo sencillo para interactuar con la capa de reenvío virtual en el núcleo de Linux. Esta tecnología existe desde casi antes que los usuarios se dieran cuenta de que era posible conectar utilizando una máquina virtual de Linux dos o más puertos de una red física. En esta prueba será utilizado LINUX BRIDGE para comprobar el retardo de una red tradicional. Actualmente muchas de las infraestructuras de virtualización utilizan LINUX BRIDGE para posibilitar la comunicación entre máquinas virtuales.

En la figura 3.1 y 3.2 aparecen la topología evaluada y el código en Python para la implementación respectivamente. La medición de latencia fue realizada con el comando ping de Linux enviando 100 paquetes ICMP desde el host h1 hasta h3.

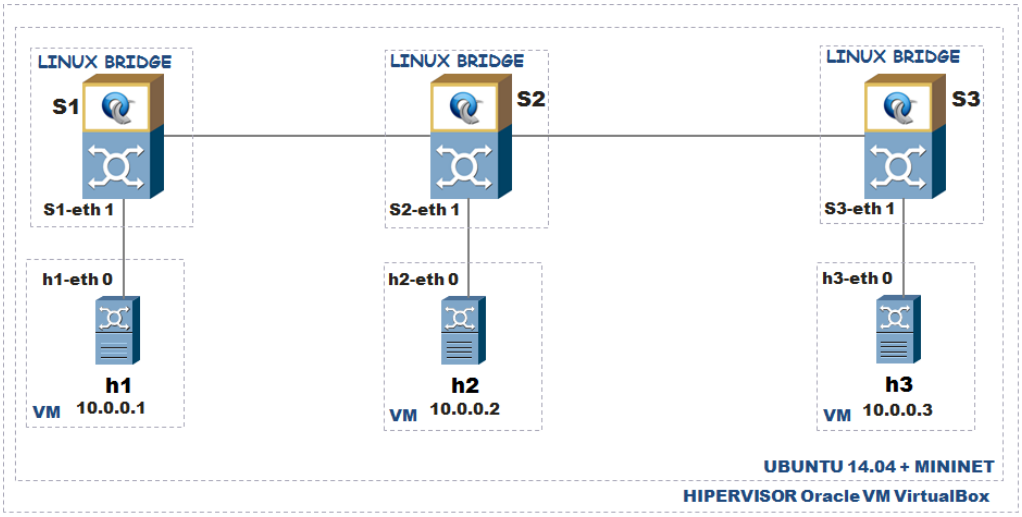


Figura 3.1. Evaluando el retardo entre los nodos h1 y h3 en una red tradicional. Fuente: Autor.

Ejecutando el siguiente mando es ejecutada la topología:

$ sudo mn --custom /home/ubuntu/mininet/custom/uclv_sdn_vs_tradicional.py --topo tradicional

Mininet>xterm h1

Mininet>xterm h3

Posteriormente desde el host “h1” es ejecutado el comando “ping –c 100 10.0.0.3” y se analizan los resultados.

Prueba de retardo en una red SDN

Con el objetivo de implementar la red anteriormente mencionada será necesario crear en el emulador MININET una topología personalizada que tenga en cuenta la utilización de un conmutador del tipo OVS y un controlador. Para simplificar la prueba fue utilizado el controlador de referencia REF que está incluido en la distribución de MiniNet. La medición de latencia fue realizada utilizado el comando ping de Linux enviando 100 paquetes ICMP desde el host h1 hasta h3.

Un conmutador virtual (OVS-VSWITCH) no es más que una versión de software de un conmutador físico. Este elemento proporciona un medio eficiente para que las VMs se comuniquen unas con otras en una red virtual. Open vSwitch extiende esta abstracción a través de hosts virtuales, permitiendo a las VMs en un host físico comunicarse transparentemente con VMs en otro host físico. OVS-VSWITCH será utilizado para comprobar el retardo de una red SDN utilizando el controlador de referencia de Mininet. En las figuras 3.3 y 3.4 aparecen la topología y el fragmento del código en Python empleado para evaluar el retardo.

Creando la topología utilizado el siguiente comando:

$ sudo mn --custom /home/ubuntu/mininet/custom/uclv_sdn_vs_tradicional.py –topo sdn

Posteriormente desde el host “h1” es ejecutado el comando “ping –c 100 10.0.0.3”

```python
from mininet.topo import Topo
from mininet.nodelib import LinuxBridge
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import Controller, RemoteController, OVSController

#PARA EJECUTAR LA TOPOLOGIA
#sudo mn --custom /home/ubuntu/mininet/custom/uclv_legacy.py --topo tradicional
#sudo mn --custom /home/ubuntu/mininet/custom/uclv_legacy.py --topo sdn


class UclvRedTradicional( Topo ):
    "Topologia de red tradicional."

    def __init__( self ):
        "Creando una topologia personalizada."

        # Inicializando la topologia
        Topo.__init__( self )


        # Creando host y switch
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        s1 = self.addSwitch( 's1', cls=LinuxBridge )
        s2 = self.addSwitch( 's2', cls=LinuxBridge )
        s3 = self.addSwitch( 's3', cls=LinuxBridge )

        # Adicionado los enlaces
        self.addLink( h1, s1 )
        self.addLink( s1, s2 )
        self.addLink( h2, s2 )
        self.addLink( s2, s3 )
        self.addLink( s3, h3 )
```

Figura 3.2. Fragmento de código Python utilizado para evaluar el retardo entre los nodos h1 y h3 en una red tradicional. Fuente: Autor.

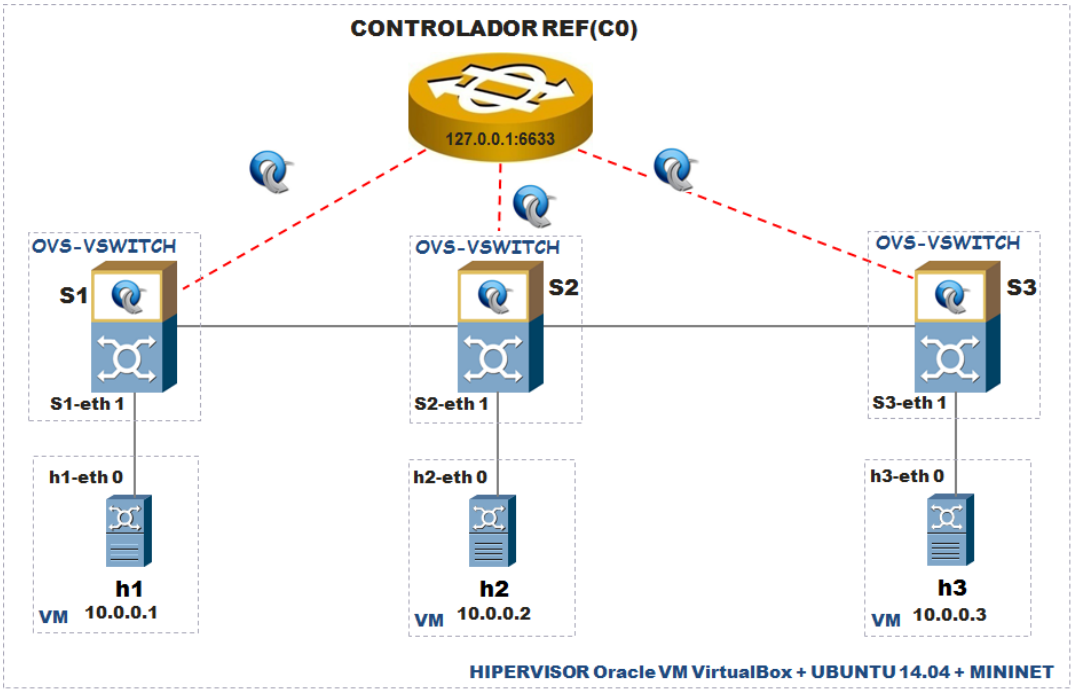


Figura 3.3. Evaluando el retardo entre los nodos h1 y h3 en una red SDN. Fuente: Autor.

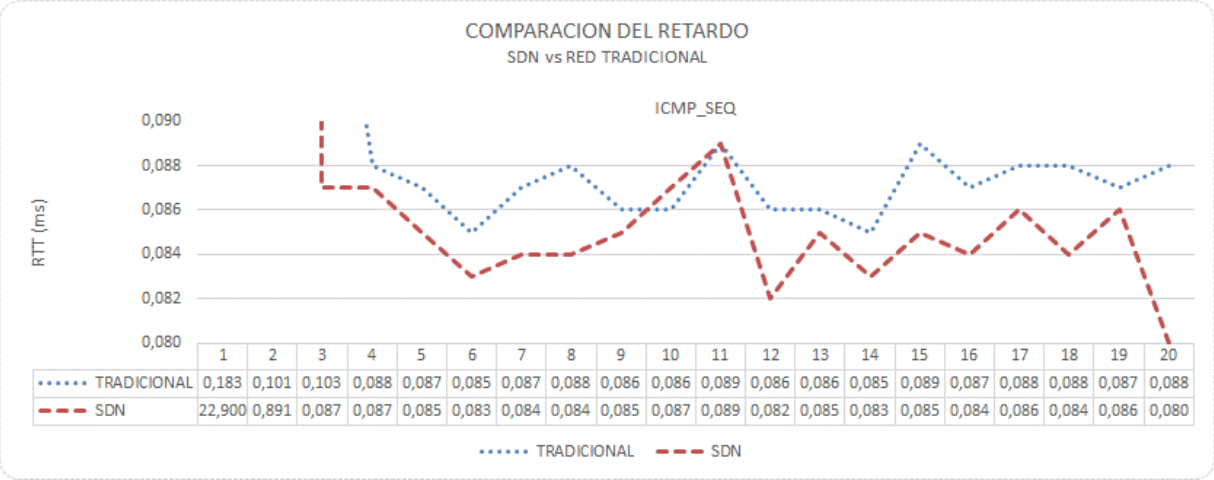
```
class UclvRedSDN( Topo ):
    "Topologia de red SDN."
    def __init__( self ):
        "Creando una topologia personalizada."
        # Inicializando la topologia
        Topo.__init__( self )
        # Creando host y switch
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        s1 = self.addSwitch( 's1', cls=OVSKernelSwitch )
  s2 = self.addSwitch( 's2', cls=OVSKernelSwitch )
  s3 = self.addSwitch( 's3', cls=OVSKernelSwitch )
        # Creando los enlaces entre los nodos
        self.addLink( h1, s1 )
        self.addLink( s1, s2 )
        self.addLink( h2, s2 )
        self.addLink( s2, s3 )
        self.addLink( s3, h3 )

topos = { 'tradicional': ( lambda:
UclvRedTradicional() ) , 'sdn': ( lambda: UclvRedSDN() )}
```

Figura 3.4. Fragmento de código Python utilizado para evaluar el retardo entre los nodos h1 y h3 en una red SDN. Fuente: Autor.

Análisis de los resultados

Procesando los datos de las simulaciones realizadas son obtenidos los siguientes resultados.

Como se muestra en la figura 3.5 cuando se envía el primer paquete ICMP ocurre una demora de 0.183 ms en la red tradicional y 22.90 ms en la SDN. Esta diferencia se debe a que en SDN cuando el paquete que envía el host h1 llega al conmutador este elemento de red no sabe cómo encaminarlo, lo encapsula en el protocolo OpenFlow y reenvía todo el contenido del paquete entrante al controlador. Entonces es quien se encarga de gestionar la instalación de las tablas de flujos en cada conmutador. Este proceso inicial de establecimiento de flujo en modo reactivo consume un tiempo que introduce una latencia en la red. Este mismo proceso ocurre en los conmutadores “S2” y “S3”. En el envío de los dos primeros paquetes ICMP la red tradicional obtuvo los mejores resultados, pero a partir del tercer paquete se observa claramente que la red SDN tiene un mejor desempeño. Esto se debe a que en una red SDN después de instalados los flujos los caminos para encaminar las tramas se almacenan en la memoria para lograr un mejor rendimiento y después no es necesario realizar en cada elemento de la red una inserción o actualización (función br_fdb_update de LINUX BRIDGE) en la base da datos del conmutador (MAC.src vs Puerto.src) después de procesar todas las tramas entrantes. En una red tradicional es necesario analizar todas las tramas entrantes en cada conmutador, como parte del proceso de aprendizaje de MAC así como de actualización de una base de datos interna en el conmutador que posee la dirección física del host y el puerto de origen.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TRADICIONAL	0,183	0,101	0,103	0,088	0,087	0,085	0,087	0,088	0,086	0,086	0,089	0,086	0,086	0,085	0,089	0,087	0,088	0,088	0,087	0,088
SDN	22,900	0,891	0,087	0,087	0,085	0,083	0,084	0,084	0,085	0,087	0,089	0,082	0,085	0,083	0,085	0,084	0,086	0,084	0,086	0,080

Figura 3.5. Comparación del retardo de las redes tradicionales con las SDN. Fuente: Autor.

Si es descartado el primer datagrama ICMP y se calcula el valor medio del retardo es posible comprobar que el RTT_tradicional=0,089 ms, RTT_sdn=0,127 ms. Cuando se descarta el primer y el segundo datagrama RTT_tradicional=0,088 ms, RTT_sdn=0,085 ms.

El aspecto más importante que marca la diferencia de retardo está relacionado con la propia arquitectura de vswitch. El vSwitch al trabajar con el concepto de caminos de datos guarda un grupo de reglas en una memoria cache y al encontrar el plano de renvío una coincidencia entre la trama entrante y la cache, en el propio kernel de Linux ocurre la conmutación a partir de la acción configurada en la regla y la operación es completada en menor tiempo. También este tipo de conmutación es implementada sin que viaje la trama a través de una interface física. Posteriormente fueron repetidos los experimentos anteriores pero en este caso fueron enviados 500 paquetes ICMP. Igualmente fueron comparado los retardos entre la red tradicional (RTT_TRAD) y la SDN (RTT_SDN). Igualmente la RED SDN obtuvo los mejores resultados cuando fue analizado el retardo entre los host h1 y h3.

Tabla 3.1. Comparación del retardo de las redes tradicionales con las SDN. Fuente: Autor.

ICMP_SEQ	RTT_TRAD (ms) PROMEDIO	RTT_SDN (ms) PROMEDIO	RTT_SDN-RTT_TRAD (ms)
2-500	0,104	0,101	0,003
3-500	0,104	0,099	0,005

En futuras investigaciones se puede utilizar el propio script creado en esta memoria para implementar redes más grandes en las que seguramente serán observadas marcadas diferencias en la latencia entre la red tradicional y una SDN.

3.3 Caso de uso: Pruebas de escalabilidad de SDN

Una prueba computarizada para medir el rendimiento de una tecnología es a lo que se le llama “Prueba de comprobación” o “benchmarking”. Entre las variables a analizar podrían incluirse la latencia, tasa de transferencia de datos, carga del CPU, memoria utilizada, etcétera. Realizar pruebas de comprobación o aceptación en la actualidad es algo que se realiza a diario por empresas antes de tomar la decisión de comprar o no una tecnología, implementar o realizar cambios en topologías de red. Antes de comprar cualquier equipo se someten a pruebas que permiten simular cargas de trabajo elevadas y así mediante aplicaciones de software observar el comportamiento de cada una de las variables estudiadas.

¿Cuán grande puede ser una red SDN creada en el emulador Mininet?

¿Cuánto será el tiempo de compilación de toda la red?

¿Cuál es el conmutador con el que se obtiene el menor tiempo de compilación de la red?

¿Influye la topología en el tiempo de compilación de la red?

En SDN para evaluar la capacidad de escalar grandes redes con numerosos hosts y conmutadores se realizan pruebas de escalabilidad (benchmarking). Para realizar estas pruebas se crearon varios script en el Shell bash de Linux para crear nodos y topologías de forma automática, concurrente e interactuando con el emulador MiniNet. La realización de los script estuvo motivada debido a que MiniNet permite simular una sola topología en un instante dado y al mismo tiempo entrega los datos en un formato difícil de procesar cuando se necesitan simular muchos escenarios. Otra cuestión importante es que usualmente Mininet cuando realiza la compilación de la topología no entrega datos referente al impacto en el sistema relacionados con el aumento de los hilos de ejecución y ocupación de memoria. Es por ello en el script realizado se encuesta en cada compilación la ocupación de la memoria para tener referencias a la hora de analizar posibles degradaciones en el rendimiento. El script creado entrega por pantalla el tiempo que tarda en crear y destruir la red, además del uso de memoria. En el Anexo 1 aparece un fragmento del script creado para realizar las pruebas de escalabilidad a la topología simple de Mininet.

Estudiar la demora total de la compilación de la red permite investigar el tiempo empleado por MiniNet en iniciar el controlador y cada uno de los nodos de la red, además del tiempo de establecimiento de la conexión de cada conmutador con el controlador. El intercambio de mensajes entre el conmutador y el controlador es mostrado en la figura 3.6.

Para la realización de esta prueba se han trazado los siguientes objetivos:

- Comprobar el número máximo de dispositivos y/o conmutadores que una red en SDN puede manejar con un controlador lógicamente centralizado sin que disminuya el rendimiento mediante la simulación de diferentes escenarios que incluyen la utilización de diferentes topologías, conmutadores, controladores.
- Determinar la máxima cantidad de nodos de una red SDN que se pueden simular utilizando MiniNet para las topologías simple, lineal y árbol.
- Determinar el modelo de Conmutador que presenta mejores resultados y el escenario con mejor rendimiento cuando se analizan las métricas de escalabilidad y el tiempo necesario en crear la red SDN y destruirla.
- Determinar el escenario con mejor rendimiento al escalar grandes redes SDN con topologías “simple, lineal y árbol” para realizar en ella el resto de las simulaciones.
- Explorar las limitaciones del emulador MiniNet cuando se incrementa el número de nodos en la red.

A la hora de elegir las computadoras en las que se virtualizan las redes SDN fueron seleccionadas dos que no tuvieran marcadas diferencias en cuanto al CPU, Caché, etc. Fue elegida una laptop con CPU superior a las computadoras de escritorio para analizar el impacto de una mayor capacidad de procesamiento. Las tres estaciones de trabajo poseen diferencias en cuanto al sistema operativo instalado, capacidad de procesamiento y memoria. Las características técnicas más importantes son presentadas en el anexo 4.

En la plataforma de virtualización de cada una de estas estaciones es creada una máquina virtual y se instala el emulador Mininet así como el analizador de protocolo Wireshark con el pluging para soportar el protocolo OpenFlow.

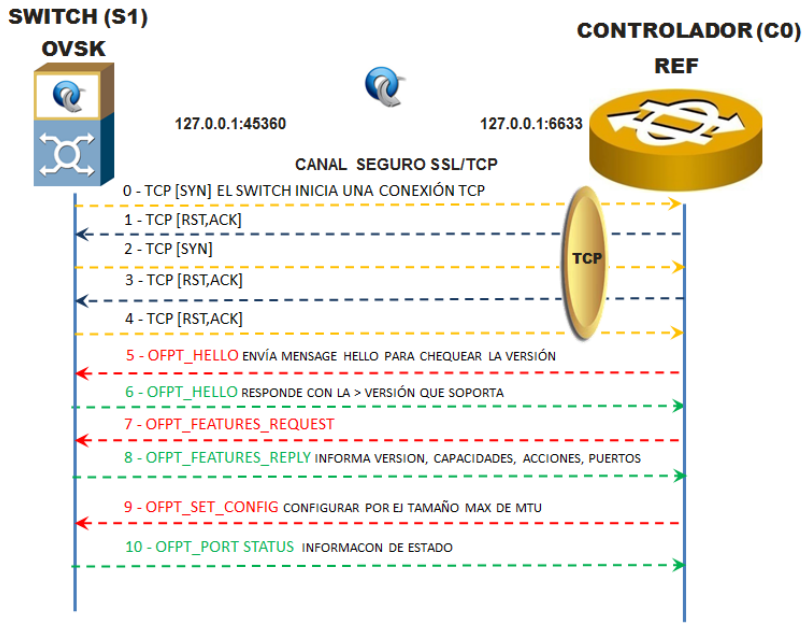


Figura 3.6. Establecimiento de la conexión entre el conmutador y el controlador. Fuente: Autor.

En todos los escenarios se evalúa el comportamiento de la red SDN cuando se utilizan los siguientes conmutadores conectados al controlador de referencia de MiniNet:

- ovsk u ovs: Es un conmutador del tipo Open vSwitch que reside en el espacio del Kernel. OpenvSwitch (OVS) es un sofware Open Source que actúa como un conmutador virtual multicapa. Su objetivo es implementar una plataforma que soporte los protocolos de red estándares y permitir que las funciones de encaminamiento puedan ser programadas. Permite utilizar OpenFlow y soporta las versiones 1.0, 1.1, 1.2 y 1.3.
- Ovsbr: Es un conmutador OVs en el modo “standalone/bridge”
- User: Es un conmutador OVs que se ejecuta en el espacio de usuario de Linux.

En los *script* creados fueron contemplados los conmutadores ovsk, ovsbr, user, de Mininet los que se conectaron según la topología configurada cada uno de forma individual al controlador OpenFlow de referencia que viene pre instalado en Mininet. En cada compilación de la prueba se va incrementando la cantidad de nodos siguiendo la regla de potencias de 2 hasta obtener a la máxima cantidad de nodos en el sistema sin que se deteriore el rendimiento.

A pesar de que Mininet es considerada como una gran herramienta y conveniente, realmente tiene algunas limitaciones debido a que no puede garantizar alta fidelidad y rendimiento cuando la carga de trabajo es muy alta. En estos experimentos se comprobará la máxima cantidad de nodos que el emulador puede manejar en cada escenario. Debido a que Mininet se ejecuta en un solo sistema, existe una limitación de recursos de procesamiento y memoria en el hardware que impone una limitación en el tamaño real de las redes que se pueden escalar.

Escalabilidad del controlador de referencia en una topología simple

La topología “Single” de Mininet que es mostrada en la figura 3.7 posee un controlador, un conmutador y n host. El script bash creado inicia la topología simple con n host, muestra la cantidad de memoria utilizada hasta este punto, destruye la red virtual y visualiza el tiempo empleado en todo el procedimiento.

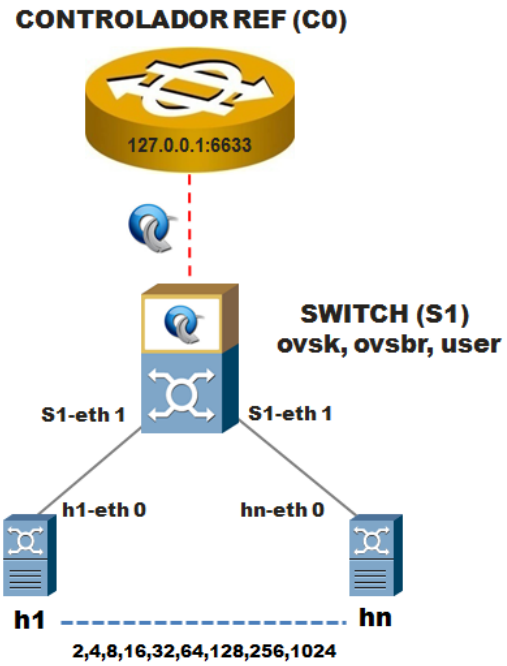


Figura 3.7. Prueba de escalabilidad SDN utilizando la topología SINGLE. Fuente: Autor.

En el Anexo 1 aparece un fragmento del script creado para la simulación automática en Mininet de la topología simple y los resultados que entrega respectivamente. En las tablas 3.3 y 3.4 se muestran los resultados de las pruebas realizadas en los escenarios A, B y C.

En la columna “nodos” fueron computados la cantidad de nodos (conmutador + host) de cada topología mientras que en el resto se registra en tiempo de inicio/fin según cada escenario/conmutador. En el experimento lo primero que se observa es que en el escenario B solamente se pueden crear redes SDN de hasta 257 nodos mientras que el A puede llegar hasta 1025. A partir de 1025 nodos en los escenarios “A, C” y 257 en el B la red se vuelve inestable. En la tabla los iconos de color verde representan los escenarios que terminaron primero a la hora de ejecutar una topología con una cantidad de nodos determina. El icono de color rojo representa el escenario en el que más demoro la simulación.

El conmutador con mejor desempeño hasta un total de 257 nodos fue el “USER” en el escenario A con tiempo total de 44,742 segundos mientras que el peor fue el “OSVBR” en el escenario B. Sin embargo la diferencia de tiempo entre el mejor escenario y el peor son solamente 6.31 segundos.

Para la creación de redes virtuales pequeñas, el tiempo inicio/fin es muy corto y este tiempo aumenta con el incremento del número de nodos virtuales. Por ejemplo, en MiniNet tomó aproximadamente 5,698 segundos para crear una topología de SINGLE con 65 nodos, mientras que con 1025 nodos, el tiempo empleado aumentó sustancialmente a un valor de110.55 segundos.

La comparación de resultados de los tres escenarios da una indicación de que la cantidad de tiempo necesario para crear una red virtual aumenta con el aumento del número de nodos virtuales en todos los casos.

En las pruebas realizadas anteriormente se corrió una simulación por cada escenario variando el número de nodos en potencias de dos. La prueba nos permitió conocer hasta donde se puede escalar una red SDN en MiniNet en diferentes escenarios. Cuando se corren varias veces una simulación para un mismo escenario existen pequeñas diferencias en cuando al tiempo de creación y destrucción de la red SDN lo que permitió comprobar que existen problemas de fidelidad de los resultados entregados por MiniNet cuando se analiza la variable tiempo. Con el objetivo de obtener resultados más precisos fue creado otro script bash que en cada escenario realiza una simulación con tantas repeticiones según especifica el usuario. El script bash permitió en este trabajo correr 2250

simulaciones de forma automática y obtener los siguientes resultados que pueden ser útiles para los investigadores.

Tabla 3.2. Tiempo(s) de inicio/fin de la prueba en los escenarios A, B y C topología simple.

NODOS	ESCENARIO A			ESCENARIO B			ESCENARIO C		
	ESC_A-ovsk t (seg)	ESC_A-ovsbr t (seg)	ESC_A-user t (seg)	ESC_B-ovsk t (seg)	ESC_B-ovsbr t (seg)	ESC_B-user t (seg)	ESC_C-ovsk t (seg)	ESC_C-ovsbr t (seg)	ESC_C-user t (seg)
3	0,378	0,461	0,417	0,316	0,289	0,473	0,451	0,453	0,477
5	0,928	1,401	0,573	0,653	0,925	1,113	0,544	1,015	0,932
9	1,203	0,902	1,235	1,512	1,252	1,256	1,434	1,364	0,717
17	1,715	1,798	1,999	1,544	1,961	2,035	2,025	1,342	2,079
33	3,094	3,018	2,957	3,302	3,486	3,165	2,59	3,156	2,886
65	5,698	6,98	5,377	5,722	6,199	6,251	5,86	5,691	6,014
129	10,648	11,214	10,822	11,212	12,102	11,636	11,182	11,754	10,718
257	21,431	23,323	21,362	24,5	24,839	23,446	22,735	22,798	22,026
513	46,692	48,939	46,335	0	0	0	49,157	51,027	48,333
1025	110,558	116,351	107,684	0	0	0	117,909	116,629	110,942
TOTAL 257 n	45,095	49,097	44,742	48,761	51,053	49,375	46,821	47,573	45,849
TOTAL	202,345	214,387	198,761				213,887	215,229	205,124

Tabla 3.3. Memoria ocupada durante de la prueba en los escenarios A, B y C topología simple.

NODOS	ESCENARIO A			ESCENARIO B			ESCENARIO C		
	ESC_A-ovsk Memoria Ocupada (MB)	ESC_A-ovsbr Memoria Ocupada (MB)	ESC_A-user Memoria Ocupada (MB)	ESC_B-ovsk Memoria Ocupada (MB)	ESC_B-ovsbr Memoria Ocupada (MB)	ESC_B-user Memoria Ocupada (MB)	ESC_C-ovsk Memoria Ocupada (MB)	ESC_C-ovsbr Memoria Ocupada (MB)	ESC_C-user Memoria Ocupada (MB)
3	649	713	710	570	582	588	1109	1048	1010
5	650	710	708	572	582	587	1110	1047	1009
9	653	712	710	574	584	590	1113	1048	1010
17	661	713	711	579	588	595	1117	1052	1014
33	672	723	720	590	598	604	1122	1060	1020
65	695	743	740	613	617	624	1141	1079	1039
129	740	783	781	655	659	664	1184	1119	1078
257	831	865	861	744	746	745	1270	1198	1157
513	1015	1037	1028	0	0	0	1446	1365	1321
1025	1385	1390	1372	0	0	0	1712	1688	1662

Posteriormente se realiza el procedimiento aplicado anteriormente con la topología lineal que es mostrada en la figura 3.8 que tiene un controlador, “n” conmutadores y “n” host. También es creado un script en el bash de LINUX que inicia la topología LINEAR con “n” host, muestra la cantidad de memoria utilizada hasta ese punto, destruye la red virtual y visualiza el tiempo empleado en todo el procedimiento. Los resultados fueron muy similares al de la topología simple escalando solamente hasta 512 nodos. En el escenario ESC_B a partir de 512 nodos la red se comporta de forma inestable.

Finalmente la topología “TREE” de MiniNet que es mostrada en la figura 3.9 también es escalada y posee un controlador “C0”, con una cantidad de conmutadores y host que dependen de la profundidad y las expansiones (fanout) configuradas.

En el script bash creado para la topología de árbol el usuario puede variar las expansiones y profundidad seleccionando un “fanout=2” y variando la profundidad hasta encontrar la máxima cantidad de host que puede escalar Mininet utilizando esta topología. En cada simulación se muestra la cantidad de memoria utilizada hasta ese punto, destruye la red virtual y visualiza el tiempo empleado en todo el procedimiento. En esta prueba solo se tuvieron en cuenta los escenarios “A y C”.

Tabla 3.4. Duración Media(s) de inicio/fin de la topología en los escenarios A, B Y C de 2 a 50 nodos con topología simple.

NODOS	ESCENARIO A			ESCENARIO B			ESCENARIO C		
	ESC_A-ovsk t (seg)	ESC_A-ovsbr t (seg)	ESC_A-user t (seg)	ESC_B-ovsk t (seg)	ESC_B-ovsbr t (seg)	ESC_B-user t (seg)	ESC_C-ovsk t (seg)	ESC_C-ovsbr t (seg)	ESC_C-user t (seg)
2	0,334	0,413	0,398	0,400	0,397	0,388	0,491	0,462	0,389
4	0,849	0,856	0,813	0,912	0,929	0,873	0,936	1,038	0,839
6	0,961	0,969	0,914	1,211	1,102	1,145	1,160	1,080	1,069
8	1,181	1,064	1,205	1,360	1,303	1,302	1,303	1,293	1,219
10	1,262	1,293	1,330	1,414	1,388	1,508	1,455	1,445	1,334
12	1,420	1,452	1,422	1,677	1,666	1,736	1,828	1,943	1,647
14	1,580	1,598	1,572	1,844	1,804	1,949	1,753	1,767	1,672
16	1,755	1,866	1,676	1,954	2,002	2,155	1,916	2,009	1,867
18	1,909	1,959	1,901	2,127	2,220	2,316	2,034	2,014	1,960
20	2,020	2,074	2,081	2,404	2,335	2,506	2,139	2,182	2,129
22	2,142	2,120	2,166	2,570	2,565	2,468	2,525	2,596	2,280
24	2,398	2,317	2,313	2,737	2,716	2,667	2,464	2,555	2,389
26	2,434	2,438	2,454	3,324	2,879	2,855	2,749	2,580	2,543
28	2,689	2,633	2,677	3,461	3,134	3,215	2,828	2,860	2,682
30	2,821	2,787	2,845	3,324	3,235	3,288	3,492	2,931	2,846
32	2,957	2,905	3,000	3,651	3,455	3,494	3,110	3,105	3,035
34	3,149	3,055	3,213	3,815	3,746	3,633	3,313	3,298	3,191
36	3,303	3,278	3,413	4,364	3,978	4,123	3,370	3,438	3,326
38	3,456	3,533	3,399	4,322	4,121	3,987	3,490	3,485	3,509
40	3,499	3,582	3,605	4,363	4,341	4,395	3,728	3,665	3,716
42	3,811	3,669	3,844	4,576	4,340	4,446	3,838	3,847	3,804
44	3,924	3,860	4,070	4,577	4,499	4,981	3,906	4,024	3,927
46	4,102	4,078	4,108	4,752	4,748	4,788	4,320	4,144	4,157
48	4,249	4,198	4,256	4,817	4,829	4,815	4,331	4,262	4,215
50	4,315	4,455	4,396	5,172	4,801	5,417	4,390	4,562	4,367
TOTAL	62,517	62,452	63,069	75,128	72,532	74,445	66,868	66,586	64,108

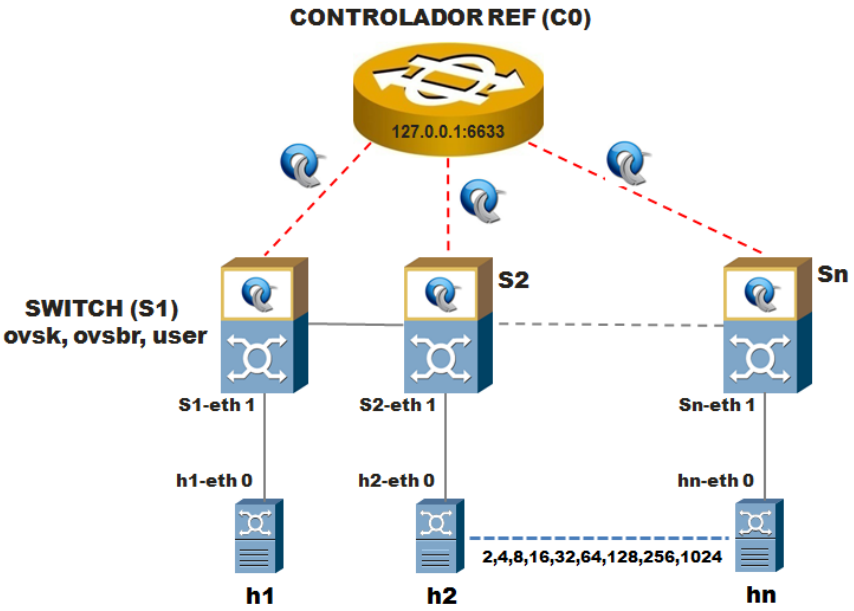


Figura 3.8. Prueba de escalabilidad SDN utilizando la topología LINEAR. Fuente: Autor.

Comparando los resultado de las topologías simple y lineal se obtiene una tabla que muestra los mejores (verde) y peores tiempo de compilación de la topología (rojo). Esta tabla puede ser útil a los investigadores que pretendan realizar simulaciones con menos de 50 nodos y diferentes tipos de topologías y escenarios. La topología que converge más rápido es la simple porque presenta un solo conmutador y "n" host. Al tener un solo conmutador la

cantidad de mensajes de control del protocolo OpenFlow es mínima. También se observa que en este tipo de topología solamente se pueden escalar hasta 511 nodos.

Tabla 3.5. Duración Media(s) de inicio/fin de la topología en los escenarios A, B Y C de 2 a 50 nodos con topología lineal.

NODOS	ESCENARIO A			ESCENARIO B			ESCENARIO C		
	ESC_A-ovsk t (seg)	ESC_A-ovsbr t (seg)	ESC_A-user t (seg)	ESC_B-ovsk t (seg)	ESC_B-ovsbr t (seg)	ESC_B-user t (seg)	ESC_C-ovsk t (seg)	ESC_C-ovsbr t (seg)	ESC_C-user t (seg)
2	0,384	0,377	✓ 0,257	0,388	0,392	0,476	✗ 0,556	0,466	0,373
4	0,728	✓ 0,714	0,746	✗ 0,958	0,841	0,943	0,939	0,915	0,872
6	1,112	1,095	✓ 1,007	1,163	1,170	1,184	✗ 1,377	1,345	1,089
8	1,303	✓ 1,265	1,279	1,393	1,347	1,412	✗ 1,538	1,515	1,303
10	1,483	✓ 1,459	1,466	1,608	1,582	1,598	✗ 1,848	1,742	1,501
12	1,743	1,692	✓ 1,640	1,904	1,836	1,866	✗ 1,975	1,938	1,716
14	1,855	✓ 1,841	1,933	2,113	2,098	2,157	2,191	✗ 2,307	2,126
16	2,163	2,188	2,159	2,234	2,276	✗ 2,427	2,410	2,386	✓ 2,110
18	✓ 2,346	2,348	2,414	✗ 2,707	2,372	✗ 2,710	2,551	2,597	2,348
20	2,651	✓ 2,465	2,554	2,777	2,752	✗ 2,930	2,845	2,806	2,560
22	✓ 2,737	2,825	2,792	2,934	3,163	3,151	3,125	✗ 3,203	2,756
24	3,045	✓ 2,966	3,128	3,484	3,291	✗ 3,505	3,275	3,366	3,406
26	✓ 3,127	3,295	3,306	✗ 3,739	3,419	3,700	3,525	3,492	3,141
28	3,334	✓ 3,300	3,602	3,693	3,736	✗ 4,014	3,643	3,833	3,306
30	3,685	3,743	3,860	4,101	4,043	✗ 4,383	3,979	4,096	✓ 3,557
32	3,810	✓ 3,778	4,096	4,309	4,353	✗ 4,575	4,154	4,206	3,831
34	4,082	4,119	4,347	4,587	4,482	✗ 4,913	4,459	4,353	✓ 4,052
36	✓ 4,201	4,227	4,577	4,455	4,696	5,204	4,723	✗ 5,341	4,293
38	4,552	4,484	4,838	5,284	4,878	✗ 5,452	4,913	4,921	✓ 4,441
40	4,887	4,725	5,071	5,351	5,458	✗ 5,763	5,242	5,087	✓ 4,561
42	4,973	✓ 4,734	5,393	5,492	5,597	✗ 6,029	5,427	5,284	4,917
44	5,200	✓ 5,029	5,616	6,146	5,683	✗ 6,411	5,690	5,568	5,193
46	5,440	5,477	5,911	5,935	6,201	✗ 6,596	5,880	5,965	✓ 5,316
48	5,559	5,763	6,128	6,278	6,234	✗ 6,959	5,800	6,095	✓ 5,536
50	5,668	✓ 5,554	6,461	6,782	6,489	✗ 7,164	6,324	6,194	5,794
TOTAL	80,067	✓ 79,462	84,581	89,812	88,386	✗ 95,520	88,389	89,020	80,097

Resultados:

Como se puede comprobar, los tiempos efectivamente aumentan de forma exponencial, según el número de nodos de la red, cuestión que también fue comprobada por el autor en [104]. Si es analizado el tiempo promedio por nodo ayuda a visualizar el hecho de que el tiempo de creación por cada nodo no es constante, y depende de que tan grande sea la red.

En la tabla 3.5 está reflejado el resultado de las simulaciones de cada escenario y se observa claramente como el script creado permite identificar rápidamente el escenario con mejores resultado (ESC_A) y que aquí aparecen representado por flecha hacia arriba de color verde. El peor escenario fue el (ESC_B) y se corresponde con el equipo de menos recursos de cómputo.

Desde el punto de vista de la escalabilidad se pudo comprobar que en las topologías simple lineal se pudieron escalar hasta 1024 nodos por el contrario de la de árbol que solo pudo llegar a 512. Existe una estrecha relación de dependencia entre la capacidad de la plataforma de cómputo, la topología de la red y el máximo tamaño de la misma. Lo anterior quedó demostrado en los resultados de las pruebas de escalabilidad a la topología de árbol en la que se escala solamente hasta 512 nodos.

Respecto al uso de memoria, se observa que esta se mantiene constante para redes pequeñas, y comienza a aumentar linealmente a medida que la red crece. Es preciso recalcar el hecho de que las diferencias en los valores se deben a los diferentes equipos en los cuales se realizan las pruebas así como a la

capacidad de memoria instalada. Esto es importante destacarlo también para enfatizar la dependencia de los recursos de los que dispone Mininet en la emulación. Mientras más recursos, más grande son las redes que puede escalar.

Por otro lado, con las pruebas realizadas se demuestra la versatilidad e interactividad de Mininet. Las redes simuladas pueden lograr interacción con redes reales, pudiéndose conectar a Internet. Además, los hosts emulados son capaces de realizar acciones de hosts reales, como montar un servidor web entre otros. Además, los conmutadores emulados son capaces de usar nuevos protocolos de ruteo, los cuales se pueden diseñar e implementar fácilmente.

Sin embargo, Mininet también presenta desventajas. Con las pruebas de comprobación realizadas queda claro que la capacidad de Mininet de emular redes a gran escala es limitada por los recursos disponibles en la maquina en donde corre. En este sentido, los tiempos dependen de la capacidad de procesamiento de la computadora, y estos aumentan exponencialmente al aumentar el número de nodos, mientras que el uso de memoria aumenta de forma lineal. Esto limita claramente la escalabilidad que se puede alcanzar con Mininet. Es importante destacar además de que el hecho de que todos los nodos compartan los mismos recursos de la máquina es también una desventaja.

Las SDN presentan una innovación en el área de las redes de computadores, y para su estudio el uso de la plataforma Mininet resulta de gran ayuda. El hecho de que Mininet realice emulaciones en lugar de simulaciones ofrece una gran diferencia respecto a otras herramientas. La emulación permite usar los prototipos y códigos usados en redes reales, además de extrapolar los resultados logrados; lo que es una ventaja frente a una red simulada.

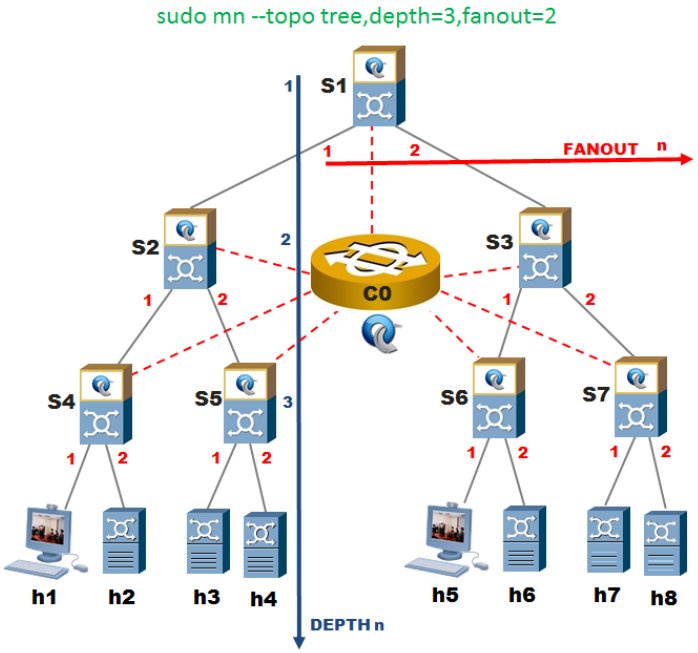


Figura 3.9. Topología árbol en Mininet. Fuente: Autor.

Tabla 3.6. Tiempo(s) de inicio/fin de la prueba en los escenarios A y C topología de árbol.

NODOS	ESCENARIO A			ESCENARIO C		
	ESC_A-ovsk t (seg)	ESC_A-ovsbr t (seg)	ESC_A-user t (seg)	ESC_C-ovsk t (seg)	ESC_C-ovsbr t (seg)	ESC_C-user t (seg)
3 (1 switch + 2 host)	✓ 0,533	0,711	0,576	✗ 0,836	0,617	0,725
7 (3 switch + 4 host)	1,393	1,501	1,404	✓ 1,268	✗ 1,537	1,521
15 (7 switch + 8 host)	✓ 2,370	✗ 3,252	2,649	2,400	2,896	2,695
31 (15 switch + 16 host)	✓ 3,812	5,585	5,407	4,437	5,965	✗ 7,604
63 (31 switch + 32 host)	8,301	11,315	11,745	✓ 8,121	11,094	✗ 11,793
127 (63 switch + 4 host)	✓ 16,484	21,764	✗ 27,581	17,123	25,501	22,588
255 (127 switch + 128 host)	33,207	43,534	✗ 83,643	✓ 32,942	55,369	48,507
511 (255 switch + 256 host)	✓ 73,632	82,455	✗ 287,827	75,256	210,463	
TOTAL	✓ 139,733	170,115	✗ 420,833	142,383	313,442	

Tabla 3.7. Mejores y peores tiempos de compilación de la red en diferentes escenarios y topologías.

	TOPOLOGIA SIMPLE. MEJORES Y PEORES TIEMPO DE COMPILACION									TOPOLOGIA LINEAL. MEJORES Y PEORES TIEMPO DE COMPILACION								
	ESCENARIO A			ESCENARIO B			ESCENARIO C			ESCENARIO A			ESCENARIO B			ESCENARIO C		
NODOS	ovsk t (seg)	ovsbr t (seg)	user t (seg)	ovsk t (seg)	ovsbr t (seg)	user t (seg)	ovsk t (seg)	ovsbr t (seg)	user t (seg)	ovsk t (seg)	ovsbr t (seg)	user t (seg)	ovsk t (seg)	ovsbr t (seg)	user t (seg)	ovsk t (seg)	ovsbr t (seg)	user t (seg)
2												✓				✗		
4								✗			✓							
6			✓													✗		
8		✓														✗		
10	✓															✗		
12	✓															✗		
14			✓														✗	
16			✓												✗			
18			✓										✗		✗			
20	✓														✗			
22		✓															✗	
24			✓												✗			
26	✓												✗					
28		✓													✗			
30		✓													✗			
32		✓													✗			
34		✓													✗			
36		✓															✗	
38			✓												✗			
40	✓														✗			
42		✓													✗			
44		✓													✗			
46		✓													✗			
48		✓													✗			
50	✓														✗			
TOTAL		✓													✗			

3.4 Aplicación SDN para detectar intrusos en la red

Según fue abordado en las secciones anteriores a través de la capa de control, las aplicaciones SDN, que son consideradas el “cerebro de la red” pueden tener en tiempo real una visión global de la misma a través de la interfaz NBI de los controladores. Utilizando esta información, las aplicaciones SDN pueden implementar mediante la programación estrategias para manipular las redes físicas subyacentes utilizando un lenguaje de alto nivel proporcionado por la capa de control.

¿Cómo implementar una aplicación SDN que permita detectar intrusos y controlar el acceso a los recursos de la red?

¿Cuánto será el impacto en la latencia y la tasa de trasferencia de datos en la red cuando se aplican políticas de seguridad?

Este estudio se centra en el desarrollo de una aplicación de seguridad que se ejecuta sobre un controlador SDN basado en OpenFlow con el objetivo de demostrar que la mayoría de las funcionalidades de un cortafuego pueden ser implementadas por software sin la necesidad de un hardware dedicado. Fue elegido el controlador POX entre muchos controladores por ser de código abierto, poseer buena documentación y estar escrito en Python que es un lenguaje de programación muy popular. Para crear la topología de red SDN, se ha utilizado Virtual Box y Mininet. En este estudio, se muestra el detalle de implementación de la aplicación cortafuego, así como el resultado de la experimentación.

Para responder a la primera interrogante científica y con el objetivo de implementar la aplicación de seguridad fueron manejados dos enfoques. El primero consisten en prestablecer las reglas en las tablas de flujo de forma proactiva; mientras que la segunda sería manejar los paquetes directamente de forma reactiva a medida que entren al conmutador. Fue elegido manejar los paquetes entrantes en modo reactivo debido a la flexibilidad que ofrece para la gestión de la red y la detección de eventos en tiempo real. Este método posee una desventaja y es que al llegar muchos paquetes al controlador ocupan gran parte de sus recursos. Es mucho más eficiente bloquear paquetes innecesarios en la capa de infraestructura utilizando las ventajas del método de instalación de reglas de forma proactiva; pero si se envía de todo el tráfico ARP al controlador se puede detectar en tiempo real un intruso en la red.

Si se desea detectar intrusos en una red primeramente se debe partir de una base de datos de usuarios autorizados así como recopilar aspectos importantes de la red tales como la topología, configuración de los puertos en los conmutadores, direcciones IP/MAC de los host autorizados. En este experimento se crea con Mininet una red con tres conmutadores con topología lineal, los que a su vez tienen conectado un host. En dicha red se desea que solamente los usuarios autorizados (host h1 y h2) puedan acceder a la misma desde sus respectivos conmutadores, dirección MAC de la interfaz de red, IP especificados en el diccionario “usuarios” que no es mas que una lista de control de acceso. Sin un usuario que no esté en esta lista, como es el caso del host “h3”, trata de acceder a la red; se le debe denegar la comunicación y generar una alarma que permita detectar intrusos en la red. También, si los usuarios autorizados tratan de acceder desde un conmutador, puerto, MAC, IP, no previamente autorizadas en la tabla se les deberá denegar el acceso a la red.

La aplicación para poder implementar políticas en la red primero debe ser capaz de realizar las funciones de un conmutador tradicional de capa dos en las que es necesario implementar un mecanismo en el que constantemente los conmutadores estén aprendiendo las direcciones MAC de las interfaces así como el puerto por donde se observa el tráfico. En este proceso es vital la implementación del protocolo ARP que es la fuente principal para el aprendizaje de las direcciones MAC y el puerto del conmutador. Como dentro de los requerimientos de la aplicación se encuentra que los usuarios solo pueden acceder desde un conmutador/puerto/MAC/IP previamente configurados en la lista de control de acceso; es necesario implementar además la funcionalidad de capa tres en los conmutadores para restringir el acceso de un host en caso de que se detecte un dirección IP no autorizada. Se hace necesario en cada paquete las direcciones IP de origen y destino para aplicar las reglas según correspondan. La aplicación SDN que implementa esa funcionalidad se encuentran en los Anexos y fue desarrollada sobre la API de POX que a su vez está escrito en Python. Todos los módulos de la aplicación se encuentran debidamente comentados lo que facilita la futura realización de aplicaciones SDN.

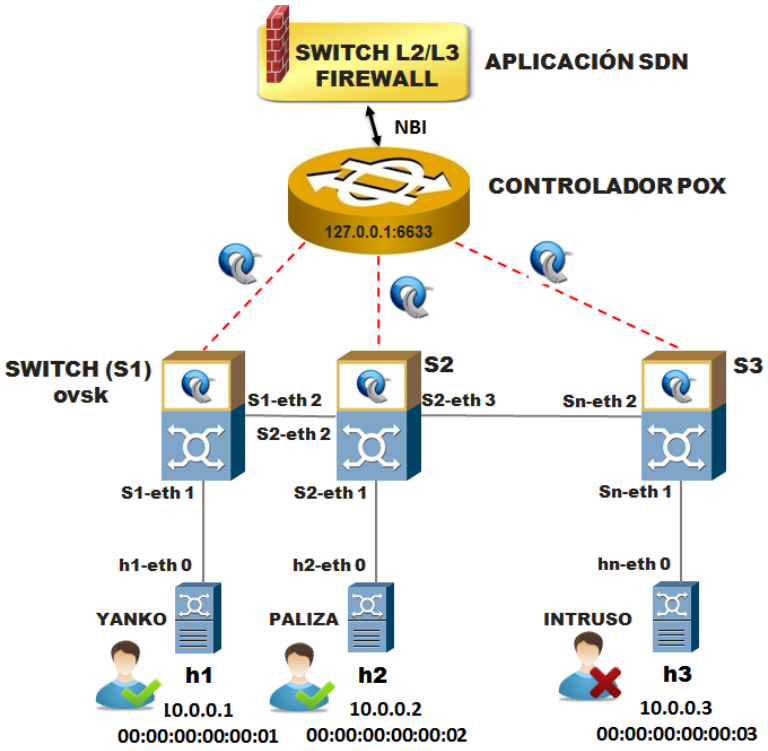


Figura 3.10. Aplicación SDN que detecta intrusos en la red. Fuente: Autor.

```
1  # DECLARANDO LOS USUARIOS AUTORIZADOS
2  opcion_seguridad_arp=False;
3  opcion_seguridad_ip=False;
4  opcion_seguridad_registro=False;
5  usuario = ({'nombre':'YANKO',
6              'ip':'10.0.0.1',
7              'switch':1, 'puerto':1,
8              'mac':EthAddr("00:00:00:00:00:01")},
9             {'nombre':'PALIZA',
10             'ip':'10.0.0.2',
11             'switch':2, 'puerto':1,
12             'mac':EthAddr("00:00:00:00:00:02")}
13            );
14
```

Figura 3.11. Lista de control de acceso, usuarios autorizados en la red. Fuente: Autor.

Para crear la topología de este experimento en Mininet se debe ejecutar el siguiente mando:

sudo mn --topo linear,3 --controller remote –mac

Posteriormente para invocar la aplicación creada los hacemos con el siguiente mando:

./pox.py forwarding.uclv_L2_L3_SWITCH_ACL_TRAZA

Con el desarrollo anterior se comprueba que la aplicación realiza las funciones requeridas lo que permite demostrar la factibilidad de las redes programables. En la figura se puede observar como desde Mininet se realiza un PING desde el host h3 al h1 en el que todos los paquetes han sido descartados. Al mismo tiempo la aplicación SDN genera una alarma el día 6 de noviembre 09:27:20 del 2016 indicando que se ha detectado un intruso en la red por el puerto 1 del conmutador 3 con dirección IP=10.0.0.3, MAC=00:00:00:00:00:03 tratando de acceder a la IP 10.0.0.1 que corresponde al host h1.

```
*** Starting CLI:
mininet> h3 ping h1 -c 1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet>
```

```
*****************************************************
UNIVERSIDAD CENTRAL MARTA ABREU DE LAS VILLAS (UCLV)
*****************************************************
APLICACION SDN: SWITCH L2/L3 , ACL, TRAZA
Autores: Ing.Yanko Antonio Marin Muro
         Dr.C. Ing. Felix Alvarez Paliza
*****************************************************
INFO:forwarding.uclv_L2_L3_SWITCH_ACL_TRAZA:Swtich en modo Reactivo
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
[HORA-Sun Nov  6 09:27:02 2016]-SWITCH-3 DETECTADO, CREANDO UNA NUEVA TABLA SWITCH/MAC/IP/PUERTO
[HORA-Sun Nov  6 09:27:02 2016]-SWITCH-2 DETECTADO, CREANDO UNA NUEVA TABLA SWITCH/MAC/IP/PUERTO
[HORA-Sun Nov  6 09:27:02 2016]-SWITCH-1 DETECTADO, CREANDO UNA NUEVA TABLA SWITCH/MAC/IP/PUERTO
[HORA-Sun Nov  6 09:27:20 2016]-SWITCH-3 DETECTADO PROTOCOLO ARP POR EL PUERTO-1
-->[Sun Nov  6 09:27:20 2016]: seguridad_arp: Intruso detectado en la red en switch:3,Puerto:1:
                                              IP=10.0.0.3,
                                              MAC:00:00:00:00:00:03.
                                              Tratando de acceder a:10.0.0.1
```

Figura 3.12. PING desde el host h3 al h1 es detectado como un evento de seguridad. Fuente: Autor.

Finalmente y para responder a una de las interrogantes de este experimento, se realiza una evaluación del impacto en la latencia y la tasa de trasferencia de datos en la red cuando se aplican políticas de seguridad. Para ello se crea una red SDN lineal con topología similar a la de la figura 3.10 y se configuran las listas de acceso de la aplicación SDN de seguridad desarrollada para que el usuario autorizado se conecte desde los host (h2, h3..hn). Posteriormente se va ampliando el tamaño de la red según el valor de n y luego se realiza una prueba de conectividad en la que se computan la latencia mínima, máxima, promedio y la desviación estándar. También se computa la pérdida de paquetes pero no se muestra en forma de gráfica porque en todos los casos fue cero. Fue utilizada la herramienta de código abierto IPERF, para medir la tasa de transferencia de datos entre los nodos h1 y hn con el objetivo de estudiar también el impacto en la tasa de transferencia de datos entre dos punto de una red en crecimiento y con políticas de seguridad.

La realización de este experimento fue realizado con el siguiente procedimiento.

sudo ./pox.py forwarding.uclv_L2_L3_SWITCH_ACL_TRAZA

sudo mn --topo linear,n --mac --switch ovsk --controller remote

h1 ping hn -c 3 & iperf

Analizando el resultado del experimento y que es mostrado en la figura 3.13, se pudo comprobar que la aplicación de seguridad cuando es implementada en el modo reactivo genera una latencia en la red que depende del tamaño de la misma. Por otra parte se comprueba que la aplicación de políticas de seguridad en el modo reactivo tiene un impacto negativo en la tasa de transferencia entre los nodos de la red. La tasa de transferencia disminuye en la medida que la red crece.

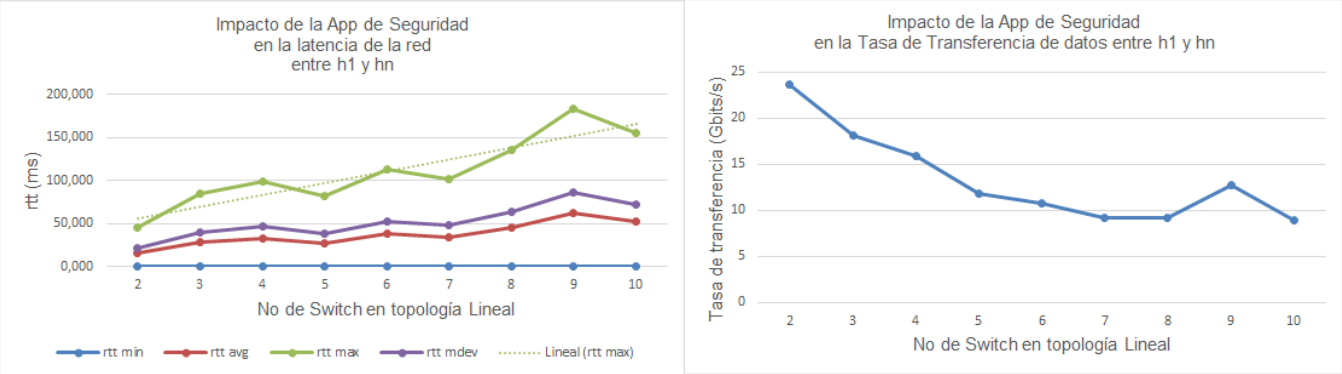


Figura 3.13. Impacto de la aplicación de seguridad en la latencia y el ancho de banda de dos nodos de la red. Fuente: Autor.

La popularidad de SDN en el mundo de TI hoy se demuestra por los numerosos casos de uso que proliferan en esta área. Aunque existen cortafuegos excepcionales en el mercado, es bastante costoso para las empresas instalar numerosos cortafuegos en toda la red para garantizar una alta seguridad. Si se fuera a realizar esta misma implementación en una red tradicional sería necesario configurar cada conmutador por separado con la consiguiente posibilidad de generar problemas en toda la red por errores humanos. El mismo problema se presentaría si se quisieran eliminar o crear nuevos usuarios en las listas de control de acceso. En SDN con la separación del control de los elementos de la capa de infraestructura se logra una flexibilidad en la red sin precedentes que crea enormes potencialidades en el campo de la seguridad de las redes.

El diseño de la aplicación solo examina de forma reactiva los campos de encabezado de los paquetes para determinar la acción a realizar. Futuras investigaciones pueden mejorar aún más esta lógica incorporando otras capacidades de las SDN mediante la observación de todo el flujo de la red con el fin de bloquear eficazmente los ataques a la red en la capa de infraestructura sin la necesidad de realizar una inspección profunda de paquetes.

3.5 Evaluación de controladores

Actualmente existen más de 50 controladores OpenFlow creados por diferentes grupos de investigación de universidades o proveedores, escritos en diferentes lenguajes de programación, y utilizan diferentes técnicas multi hilo que les permiten una mayor escalabilidad. Por lo tanto, es necesario crear herramientas y procedimientos que permitan evaluar el throughput y la latencia de los controladores. Como fue explicado al inicio de este capítulo los controladores seleccionados para esta prueba son: POX, RYU, OVS, ONOS, FLOODLIGHT y OPENDAYLIGHT. A continuación son realizadas las siguientes interrogantes científicas:

¿Cuál de los controladores seleccionados posee menor latencia y el mayor throughput?

La últimas evaluaciones de los controladores SDN / OpenFlow, [11], [105], fueron abordados una buena cantidad de controladores pero no fueron incluidos el controlador OpenDayLight, el de referencia o dejaron de analizarse otros controladores populares entre la comunidad de investigadores. Otra cuestión importante que motivo la realización de esta investigación es que se han realizado actualizaciones de una buena parte de los controladores por lo que es necesario avaluar nuevamente el desempeño de los mismos.

ESCENARIO DE PRUEBAS

El escenario para las pruebas de throughput y latencia fue implementado con una computadora de escritorio llamada escenario “A” con un CPU Intel ® Core ™ i3-3220 CPU @ 3.30 GHz, con una memoria RAM de 8 GB. El sistema operativo Windows 10 Pro (64 BITS). Fue utilizado el hipervisor ORACLE Virtual Box Versión (5.0.14 r105127) para instalar una máquina virtual con sistema operativo huésped Ubuntu 14.04 x86 (3.13.0-27-generic.). A la máquina virtual le fueron asignados 2 CPU y 4 GB de memoria RAM. El controlador, y el software para la

evaluación del controlador se encuentran en una misma máquina virtual para descartar cualquier variación en el rendimiento introducido por las interfaces de red del sistema.

Para realizar las pruebas de escalabilidad variando la capacidad de procesamiento del sistema fue escogida una laptop “escenario C” con un Intel(R) Core(TM) i5-2540M CPU @ 2.60 GHz 2.60 GHz, con una memoria RAM de 4 GB. El sistema operativo Windows 7, 32 BITS. Fue utilizado el hipervisor ORACLE Virtual Box Versión (5.0.14 r105127) para instalar una máquina virtual con sistema operativo huésped Ubuntu 14.04 x86 (3.13.0-27-generic.). A la máquina virtual le fueron asignados [1 a 4] CPU y 2 GB de memoria RAM. El controlador, y el software para la evaluación del controlador se encuentran en una misma máquina virtual para descartar cualquier variación en el rendimiento introducido por las interfaces de red del sistema.

METODOLOGÍA PARA LA EVALUACIÓN

La metodología de prueba incluye mediciones de throughput, latencia y escalabilidad.

Para evaluar los controladores se utilizó un una herramienta muy conocida por la comunidad de investigadores llamada CBENCH [112] y que actualmente se encuentra incluida dentro de la herramienta para la evaluación de conmutadores OpenFlow llamada OFLOPS. Esta herramienta mide las métricas de latencia y throughput en un escenario previamente configurado y con un único controlador. Sin embargo, no estábamos satisfechos con las métricas contempladas por cbench, así que desarrollamos una aplicación para evaluar controladores SDN/OpenFlow que utiliza los resultados de cbench e incorpora otras métricas.

Nuestro escenario de prueba utiliza CBENCH pero puede evaluar múltiples controladores de forma automática, es posible la configuración de pruebas de escalabilidad en las que se incrementan la cantidad de conmutadores, de host así como la cantidad de pruebas por escenarios más el tiempo de duración de la prueba. Con nuestra herramienta es posible obtener la siguiente estadística:

- Latencia, throughput (CBECH) de múltiples controladores.
- Características de la computadora donde se ejecuta la prueba: Memoria del BIOS, Modelo del CPU, Cantidad de procesadores lógicos, memoria instalada, arquitectura del sistema operativo, Cantidad de hilos por core y cantidad de core por socket. Frecuencia del CPU, Hipervisor, Memoria cache (L1, L2, L3).
- Ocupación de memoria,
- Ocupación del CPU,
- Tiempos de respuestas RTT del servidor realizando una prueba de conectividad a la dirección IP 127.0.0.1.

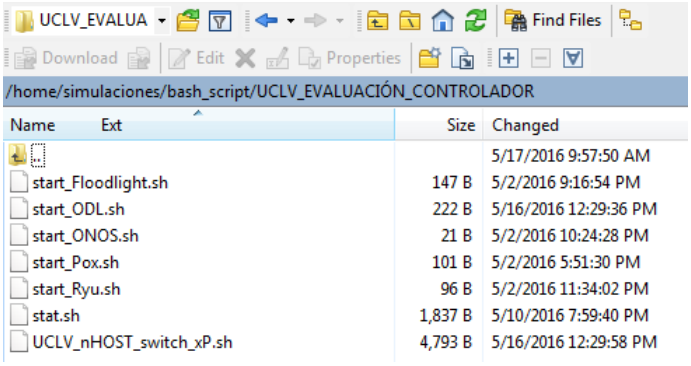


Figura 1 Estructura de la script desarrollados. Fuente: Autor.

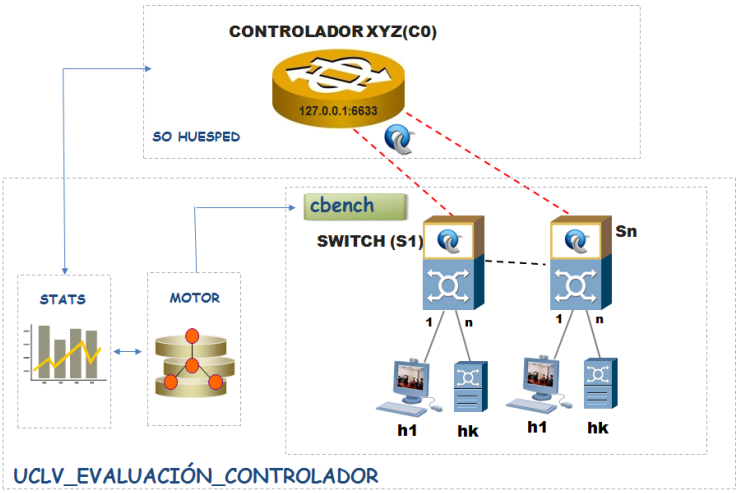


Figura 3.45. Arquitectura de la herramienta desarrollada. Fuente: Autor.

```
1  ------------------------------------------------
2  |PRUEBA DE THROUGHPUT DE CONTROLADOR:ODL
3  |CONTROLADOR: IP=localhost, PUERTO=6633
4  |Cantidad switch:1, Cantidad de host:100000
5  ------------------------------------------------
6  05:59:50.878 1   switches: flows/sec:  26845   total = 2.682224 per ms
7  06:00:00.992 1   switches: flows/sec:  191934   total = 19.171694 per ms
8  06:00:11.097 1   switches: flows/sec:  148090   total = 14.802638 per ms
9  06:00:21.206 1   switches: flows/sec:  122382   total = 12.227163 per ms
10 06:00:31.320 1   switches: flows/sec:  166154   total = 16.593472 per ms
11 06:00:41.450 1   switches: flows/sec:  160422   total = 15.996098 per ms
12 06:00:51.556 1   switches: flows/sec:  133854   total = 13.377511 per ms
13 06:01:01.667 1   switches: flows/sec:  127175   total = 12.703597 per ms
14 06:01:11.782 1   switches: flows/sec:  158205   total = 15.796966 per ms
15 06:01:21.889 1   switches: flows/sec:  145081   total = 14.500389 per ms
16 06:01:31.993 1   switches: flows/sec:  155052   total = 15.499918 per ms
17 RESULT: 1 switches 10 tests min/max/avg/stdev = 12227.16/19171.69/15066.94/1940.56 responses/s

47 ---------------------------------
48 Memoria ocupada:1785152
49 CPU:average: 1.62 0.65
50 Ping:rtt min/avg/max/mdev = 0.017/0.019/0.022/0.004 ms
51 us sy id=307982 66 33
52 ---------------------------------
```

Figura 3.46. Ejemplo de los resultados que entrega el script creado para la evaluación de controladores. Fuente: Autor.

El script principal de la prueba es “UCLV_nHOST_switch_xP.sh” está realizado en el bash Shell del Linux. A continuación se muestran las partes más importantes del código empleado:

```
1  #!/bin/bash
2  #Limpiar la pantalla
3  clear
4  echo "------------------------------------------------------------"
5  echo "|TITULO:PRUEBA DE LATENCIA y THROUGHPUT DE CONTROLADOR SDN|"
6  echo "|Autor: Ing. Yanko Antonio Marin Muro                      |"
7  echo "|Tutor: Dr. C. Ing. Felix Alvarez Paliza                   |"
8  echo "|email:yanko.marin@etecsa.cu ; yanko.antonio@gmail.com     |"
9  echo "|      fapaliza@uclv.edu.cu                                |"
10 echo "------------------------------------------------------------"
11
12 #DATOS PARA LA SIMULACION ELEGIDOS POR EL USUARIO
13 read -p "Seleccione el controlador (POX,NOX,FloodLight, ONOS, RYU, Trema, REF, ODL: " controlador
14 read -p "Modo de cbench (latencia ("L") o THROUGHPUT(T): " modo
15 read -p "Cantidad de host por switch: " chost
16 read -p "Numero de saltos (P1,P2,P10): " saltos
17 read -p "Cantidad de max de switch : " cswitch
18 read -p "IP DEL CONTORLADOR:" cip
19 read -p "PUERTO DEL CONTORLADOR:" puerto
20 read -p "DURACION DE LA PRUEBA (ms):" duracion
21 read -p "NUMERO DE PRUEBAS:" npruebas
22
23 # REUTILIZACION DE SOCKETS EN ESTADO TIME_WAIT
24 echo 1 | sudo tee /proc/sys/net/ipv4/tcp_tw_reuse;
25
26 #INICIANDO LA CAPTURA DE ESTADISTICAS DE MOMORIA, CPU, ETC
27 sudo sh stat.sh $cip $puerto $cswitch $chost $duracion $npruebas $controlador $modo &
```

Figura 3.47. Fragmento del script desarrollado "UCLV_nHOST_switch_xP.sh". Fuente: Autor.

Como se puede observar en el código del script principal "UCLV_nHOST_switch_xP.sh" el usuario puede manualmente ingresar los datos necesarios para la simulación que son el nombre del controlador, dirección IP, puerto, cantidad del saltos que controla como se van incrementando la cantidad de conmutadores en el sistema; que pueden ser incremento secuencial, potencias de dos o de diez.

Posteriormente en la línea 27 es invocado el script de estadísticas"stat.sh" y se le pasa como parámetros las variables que contienen los datos de la simulación.

Se dipone de la función "escuchar_puerto" que se encarga de verificar si existe algún controlador escuchando por el puerto definido por el usuario para detener la simulación.

La función "probar_controlador_latencia" es la encargada de realizar las pruebas de latencia utilizando la herramienta CBENCH, mientras que "probar_controlador_throughput" prueba la cantidad máxima de flujos que un controlador puede procesar. Esta última función es la que logra saturar el controlador.

El script estadística"stat.sh" es el encargado durante de la simulación recopilar todos los datos del comportamiento de CPU, ocupación de memoria del servidor, tiempo de respuesta donde se encuentra hospedado el controlador SDN.

Los datos que entrega la simulación, el usuario de la aplicación puede elegir si desea obtener el detalle de toda la simulación o un resumen para procesamiento estadístico.

A la herramienta se le pueden incorporar futuras métricas modificando el fichero de estadísticas o el fichero UCLV_nHOST_switch_xP.sh.

Comprendiendo el funcionamiento de CBENCH

En este tópico nos centramos en evaluar los dos parámetros principales de un controlador OpenFlow que son rendimiento (throughput) y la latencia. Por ejemplo, examinemos la funcionalidad del controlador POX. Para esto utilizaremos el componente l2_learning, que pertenece a los ejemplos suministrados con POX, en el directorio pox/forwarding. Dicho componente realiza la función clásica de un conmutador. Ejecutamos el controlador:

$sudo ./pox.py forwarding.l2_learning

Modo latencia de cbench:

Antes de comenzar las pruebas de rendimiento del controlador son estudiados en el analizador de protocolo wireshark los mensajes intercambiados entre el controlador y un conmutador con un host generados virtualmente por cbench. Utilizando el siguiente comando son configuradas dos pruebas con una duración de 1000 ms. Como la primera prueba cbench la descarta entonces realmente se realizará una sola prueba entre el conmutador y el controlador. Por cada host configurado el conmutador comienza a enviar mensajes OFPT_PACKET_IN al controlador y no envía el próximo hasta no haber recibido el mensaje OFPT_PACKET_OUT. Al recibir el mensaje OFPT_PACKET_OUT calcula las estadísticas. El siguiente comando realiza una prueba en modo latencia de cbench.

~/oflops/cbench$ cbench -c localhost -p 6633 -s 1 -M 1 -m 1000 -l 2

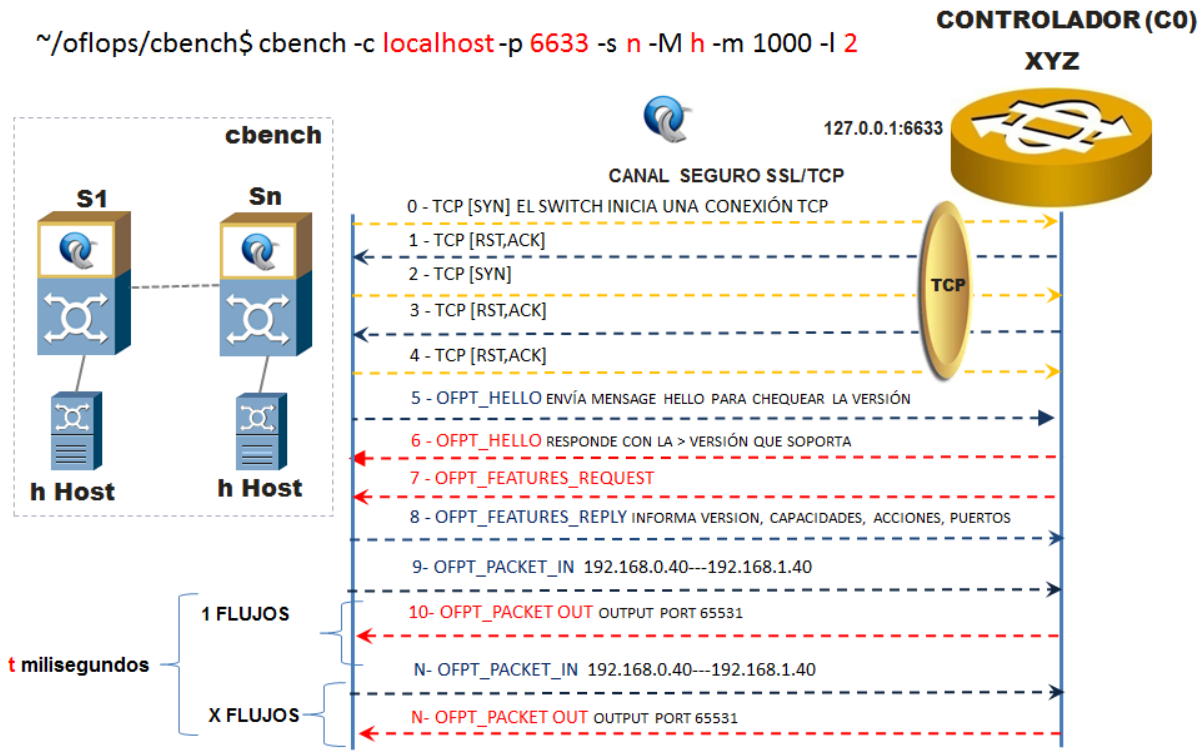


Figura 3.48. Intercambio de señalización entre cbench y el controlador. Fuente: Autor.

En las siguientes figuras se pueden observar la conexión del conmutador al controlador POX, el resultado de la prueba de cbench y el valor en cero de la primera prueba que indica que fue descartada.

```
ubuntu@sdnhubvm:~/pox$ cd /home/ubuntu/pox && ./pox.py log.level --DEBUG forward
ing.tutorial_l2_hub
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.6/Jun 22 2015 18:00:18)
DEBUG:core:Platform is Linux-3.13.0-27-generic-i686-with-Ubuntu-14.04-trusty
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:openflow.of_01:[00-00-00-00-00-01 1] closed
```

```
ubuntu@sdnhubvm:~/oflops/cbench$ sudo ./cbench -c localhost -p 6633 -s 1 -M 1 -m 1000 -l 2
cbench: controller benchmarking tool
   running in mode 'latency'
   connecting to controller at localhost:6633
   faking 1 switches offset 1 :: 2 tests each; 1000 ms per test
   with 1 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 0 ms delay after features_reply
   ignoring first 1 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
22:00:19.614 1   switches: flows/sec:  0   total = 0.000000 per ms
22:00:20.715 1   switches: flows/sec:  860   total = 0.859996 per ms
RESULT: 1 switches 1 tests min/max/avg/stdev = 860.00/860.00/860.00/0.00 responses/s
```

Filter: openflow.reason==0 Expression... Clear Apply Save Openflow 1.0

No.	Time	Source	Destination	Protocol	Length	Info
10	0.011545000	80:00:00:00:00:01	Broadcast	OpenFlow	1442	Type: OFPT_PACKET_IN
12	0.012654000	192.168.0.40	192.168.1.40	OpenFlow	148	Type: OFPT_PACKET_IN
19	1.100645000	192.168.0.40	192.168.1.40	OpenFlow	148	Type: OFPT_PACKET_IN

File: "/home/simulaciones/Trazas/cb... Packets: 1745 · Displayed: 863 (49.5%) · Dropped: 0 (0.0%) · Load time: 0:00.098

Figura 3.49. cbench conectado al controlador POX. Fuente: Autor.

3.5.1 Pruebas de desempeño y escalabilidad

El desempeño de un controlador SDN / OpenFlow se define por dos características fundamentales: throughput y la latencia. El objetivo de esta prueba es obtener el máximo throughput (número flujos / segundos) y el tiempo de respuesta (latencia) de cada controlador. Para ello censamos el comportamiento de cada controlador cuando se incrementa la capacidad de la red (incremento de conmutadores y host) o cuando se incrementa la capacidad de procesamiento. Se llevarán a cabo dos tipo de pruebas en los controladores; una será para calcular el throughput del establecimiento de flujos mientras que la otra calculará la latencia de establecimiento de flujo. Estos dos parámetros son los que nos permitirán evaluar los controladores. Para ello emplearemos la herramienta desarrollada en este trabajo.

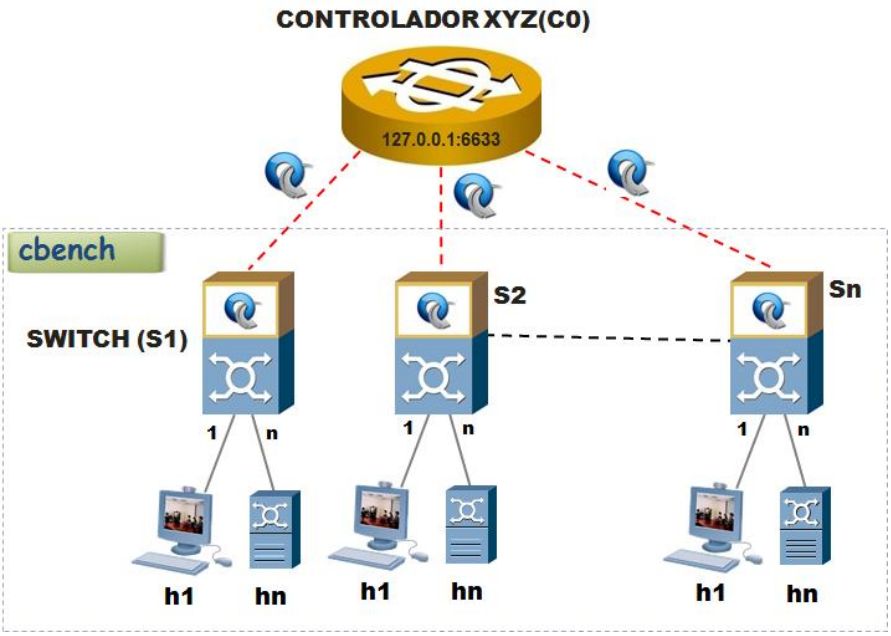


Figura 3.50. cbench conectado al controlador POX. Fuente: Autor.

Serán analizadas la correlación de la latencia y el throughput de los controladores cuando:

- Latencia: El número de conmutadores conectados al controlador (1, 2 4, 8, 16, 64, 256, 512, 1024) con una cantidad fija de host (100)
- Throughput: El número de conmutadores conectados al controlador (1, 2 4, 8, 16, 64, 256, 512, 1024) con una cantidad fija de host (100K)
- Throughput: El número de conmutadores conectados al controlador (1, 2 4, 8, 16, 64, 256, 512, 1024) con una cantidad fija de host (100K) pero se va incrementando la capacidad de procesamiento (1, 2, 3, 4 CPU).

La latencia se mide con un conmutador, que envía solicitudes (PACKET_IN), y espera por la respuesta del controlador (PACKET_OUT) antes de enviar la siguiente solicitud. En cada ejecución de cbench son realizados 10 corridas de 10 segundos de duración con el objetivo de obtener suficientes muestras y para que el reporte de valor problema que entre la aplicación sea más fidedigno.

La herramienta de evaluación creada y el controlador se han instalado en una misma computadora para que la limitación de velocidad de la interface de red no sea un obstáculo.

Durante la prueba se ejecutaron al mismo tiempo la herramienta de comprobación y cada controlador de forma individual. En la prueba de avaluación del controlador fueron conectados diferentes cantidades de conmutadores virtuales donde cada uno posee 100 ó 100K (host) con una única MAC en dependencia de la prueba realizada.

En cada simulación se escoge un controlador y se va incrementando la cantidad de conmutadores (1, 2,4, 8,16,32,128, 256, 512, 1024) para investigar el impacto en el rendimiento del controlador cuando se incrementa el número de conmutadores en una red SDN. En cada prueba se realizan 10 ciclos con una duración cada uno de 10 segundos y fue registrado en todos los casos los valores mínimos, máximo, promedio, así como la desviación estándar. El primer ciclo es descartado para no tener en cuenta una posible demora inicial del controlador.

Para realizar las pruebas de latencia y throughput fueron creados varios script en el bash Shell de Linux que permiten realizar pruebas de comprobación a controladores SDN en entornos similares al expuesto en esta investigación. El software creado inicia automáticamente el controlador bajo prueba y luego comienza a realizar de forma automática las pruebas de comprobación utilizando la herramienta cbench. Cada resultado de cada prueba es colectado, procesado y guardado en un formato fácil de procesar.

Para la prueba de latencia la configuración de la herramienta desarrollada fue la siguiente:

- controlador="XXXXXX"
- modo="L": Para indicar que es el modo latencia de cbench
- chost=100000: Especificando la cantidad de host por conmutador
- saltos="P2": Indicar que los conmutadores se incrementen siguiendo el esquema de potencia de dos.
- cswitch=2048: La cantidad máxima de conmutadores a la que puede llegar la prueba.
- cip="localhost": Dirección IP del controlador
- puerto=6633: Para especificar el puerto por donde escucha el controlador
- duración=100: duración de la prueba 10 segundos.
- npruebas=11: Realizar 10 pruebas en cada escenario. La primera prueba es descartada.

El procedimiento para ejecutar la herramienta desarrollada es:

```
root@uclvsdn:/home/simulaciones/bash_script/cbench# sudo sh cbench_nHOST_switch_xP.sh
```

En los anexos se detallan los pasos seguidos durante la instalación y configuración de todos los controladores.

Controlador POX

En los anexos se detallan los pasos seguidos durante la instalación y configuración de POX en una máquina virtual con Ubuntu 14.04. El script que iniciará el controlador es:

```
$ cd /home/ubuntu/pox && ./pox.py forwarding.l2_learning
```

Posteriormente es necesario iniciar el script creado para la prueba de comprobación del controlador y luego configurar los parámetros según fue mostrado anteriormente:

Controlador RYU

Para realizar la prueba primero se debe instalar el controlador RYU

```
apt-get install python-pip

pip install ryu

sudo apt-get install python-setuptools python-bobo

git clone git://github.com/osrg/ryu.git

cd ryu

sudo python ./setup.py install
```

Posteriormente es necesario iniciar el controlador mediante el siguiente procedimiento:

```
cd /home/ubuntu/ryu/bin && sudo ./ryu-manager ryu/ryu/app/tutorial_l2_switch.py

cd /home/ubuntu/ryu && ./bin/ryu-manager --verbose ryu/app/simple_switch.py
```

Posteriormente es necesario iniciar el script creado para la prueba de comprobación del controlador y luego configurar los parámetros según fue mostrado anteriormente

A continuación ejecutar la prueba utilizando el script:

```
root@uclvsdn:/home/simulaciones/bash_script/cbench# sudo sh cbench_nHOST_switch_xP.sh
```

Controlador OVS

Para realizar la prueba será utilizado el controlador de referencia de MiniNet.

Posteriormente se creará una topología simple en MiniNet que posee dos host y un conmutador. El controlador debe ser iniciado mediante el siguiente procedimiento:

```
$ sudo mn
```

Posteriormente es necesario iniciar el script creado para la prueba de comprobación del controlador y luego configurar los parámetros según fue mostrado anteriormente:

A continuación ejecutar la prueba utilizando el script:

```
root@uclvsdn:/home/simulaciones/bash_script/cbench# sudo sh cbench_nHOST_switch_xP.sh
```

Controlador FLOODLIGHT

Para realizar la prueba primero se debe instalar el controlador FLOODLIGHT según es mostrado en los anexos y luego es necesario iniciar el controlador mediante el siguiente procedimiento:

En la carpeta /home/ubuntu/floodlight/src/main/java/net/floodlightcontroller/learningswitch en el fichero LearningSwitch.java modificamos la siguiente línea de código:

protected static final boolean LEARNING_SWITCH_REVERSE_FLOW = false;

Luego ejecutar floodlight:

cd /home/ubuntu/floodlight && sudo java -jar target/floodlight.jar -cf src/main/resources/floodlightdefault.properties

Posteriormente es necesario iniciar el script creado para la prueba de comprobación del controlador y luego configurar los parámetros según fue mostrado anteriormente.

A continuación ejecutar la prueba mediante el script:

root@uclvsdn:/home/simulaciones/bash_script/cbench# sudo sh cbench_nHOST_switch_xP.sh

Controlador OPENDAYLIGHT (ODL)

Para realizar la prueba primero se debe instalar el controlador OpenDayLight siguiendo el siguiente procedimiento:

sudo tar -xvzf distribution-karaf-0.4.0-Beryllium.tar

sudo ./bin/karaf

feature:list //Listar todas las aplicaciones disponibles

feature:list -i //Listar las instaladas

feature:install odl-openflowplugin-flow-services-ui

feature:install odl-openflowplugin-drop-test

dropallpacketsrpc on

Posteriormente es necesario iniciar el script creado para la prueba de comprobación del controlador y luego configurar los parámetros según fue mostrado anteriormente.

A continuación ejecutar la prueba mediante el script:

root@uclvsdn:/home/simulaciones/bash_script/cbench# sudo sh cbench_nHOST_switch_xP.sh

Controlador ONOS

En los anexos se detallan los pasos seguidos durante la instalación y configuración de ONOS en una máquina virtual con Ubuntu 14.04. El script que iniciará el controlador es:

ubuntu@sdnlab:~/onos$ source ./tools/dev/bash_profile

ubuntu@sdnlab:~/onos$ echo $KARAF_ROOT

ubuntu@sdnlab:~/onos$ mvn clean install -nsu -DskipIT –DskipTests

Posteriormente es necesario iniciar el script creado para la prueba de comprobación del controlador y luego configurar los parámetros según fue mostrado anteriormente.

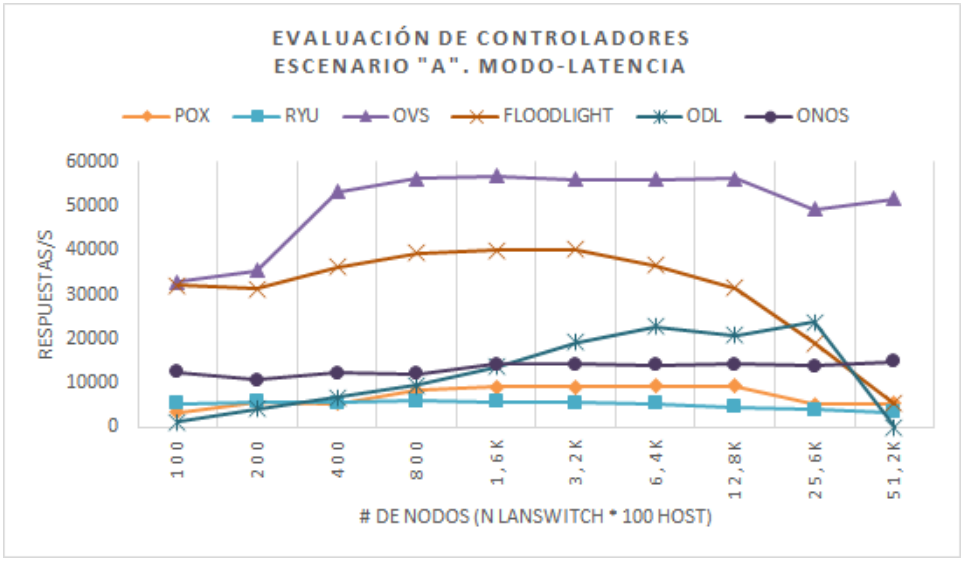
A continuación ejecutar la prueba utilizando el script:

root@uclvsdn:/home/simulaciones/bash_script/cbench# sudo sh cbench_nHOST_switch_xP.sh

Los resultados de las pruebas fueron los siguientes:

En la prueba de latencia fueron obtenidos los siguientes resultados:

NODOS	LATENCIA					
	POX	RYU	OVS	FLOODLIGHT	ODL	ONOS
	Resp/s	Resp/s	Resp/s	Resp/s	Resp/s	Resp/s
100	3474	5469	32944	32000	1255	12461
200	5534	5825	35397	31303	4292	10722
400	5505	5689	53307	36242	6738	12331
800	8436	5950	56131	39338	9545	12075
1,6K	9121	5826	56867	40080	13798	14381
3,2K	9228	5588	56007	40155	19200	14248
6,4K	9420	5484	56002	36554	22890	14073
12,8K	9362	4690	56114	31582	20857	14262
25,6K	5294	3973	49321	19140	23752	13957
51,2K	5462	3473	51623	5424	0	14948



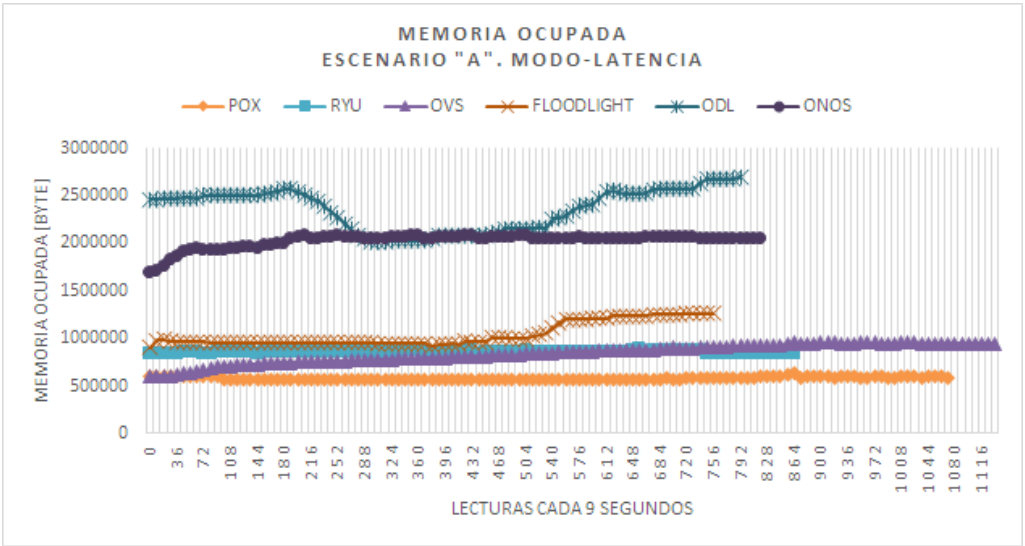


Figura 3.52. Evaluación de los controladores en el escenario "A". Modo latencia. Fuente: Autor.

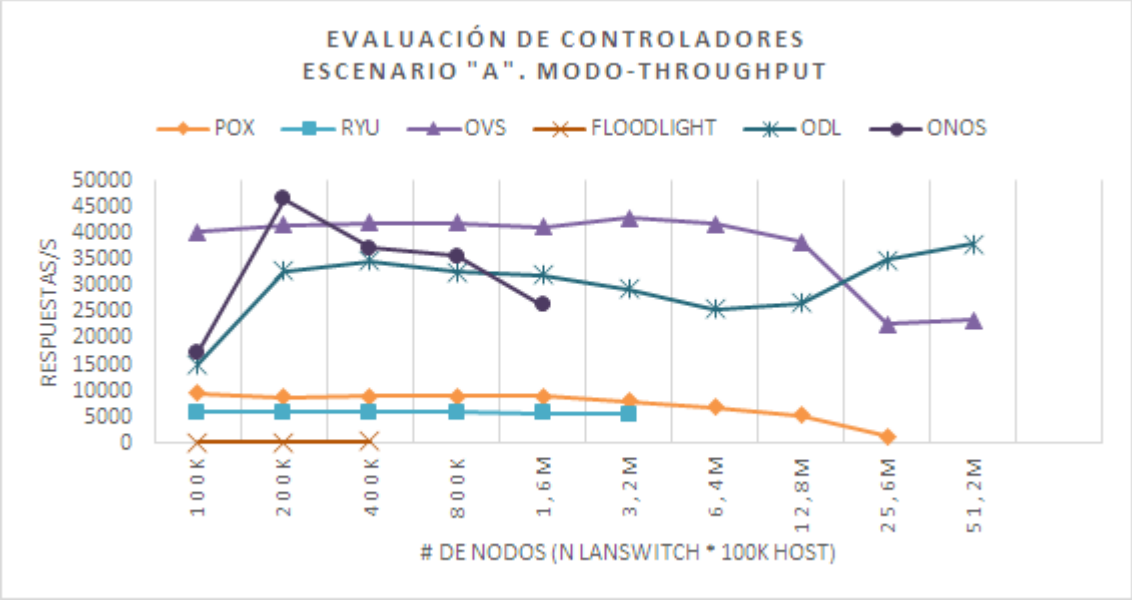
- El controlador OVS en el modo latencia obtuvo los mejores resultados con 56867 respuestas/s. El controlador en la medida que se va incrementando la cantidad de conmutadores va incrementado su

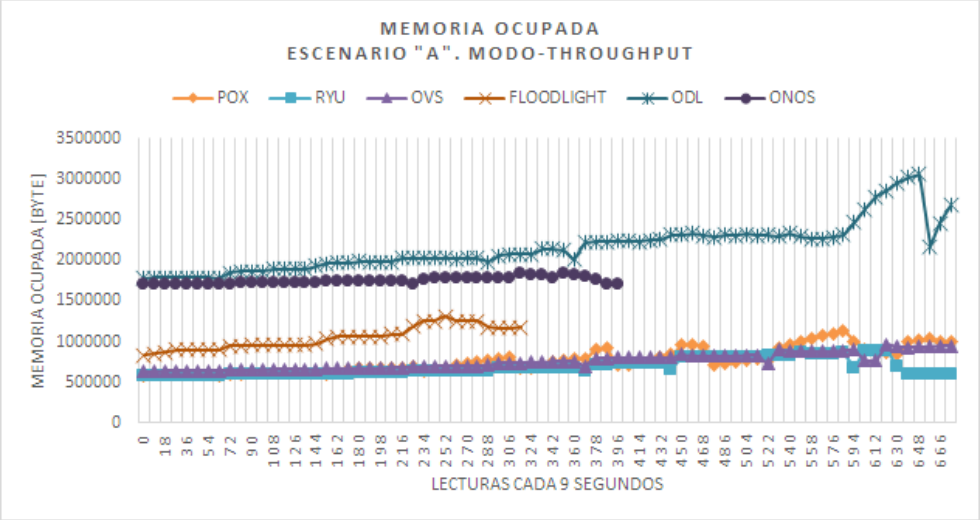
capacidad de respuesta y a partir de los 800 nodos entra en una etapa de meseta. Cuando el sistema llega a 25,6K nodos el controlador comienza a empeorar su desempeño.

- El controlador FLOODLIGHT fue el segundo con mejor latencia alcanzando un valor de 40155 respuestas/s a los 3,2K de nodos. A partir de 12,8K nodos se degrada considerablemente el desempeño del controlador.
- El controlador OPENDAYLIGHT fue el tercero con mejor desempeño 23752 de respuestas/segundo a los 25,6K nodos. En cuanto al escalabilidad presentó problemas cuando el tamaño de la red llegó a 51,2K nodos.
- El controlador ONOS siendo el cuarto lugar, mostró una gran estalabilidad y alcanzó su valor máximo de 14948 respuestas/segundo cuando la red tenía 51,2K nodos. En cuanto a la utilización de la memoria no se observan grandes variaciones en la ocupación independientemente del tamaño de la red.
- Finalmente RYU y POX mostraron pobres desempeños comparados en el resto de los controladores.
- En cuanto a la utilización de la memoria y CPU sobresale en ambos casos el controlador OPENDAYLIGHT seguido de ONOS y FLOODLIGHT observándose una considerable diferencia entre los dos primeros y el resto.

En la prueba de throughput fueron obtenidos los siguientes resultados:

NODOS	THROUGHPUT					
	POX Resp/s	RYU Resp/s	OVS Resp/s	FLOODLIGHT Resp/s	ODL Resp/s	ONOS Resp/s
100K	9569	5933	✓ 40298	✗ 254	15067	17321
200K	8828	5995	41585	✗ 226	32859	✓ 46541
400K	9088	5993	✓ 41963	✗ 433	34753	37312
800K	9102	✗ 6014	✓ 41966		32595	35612
1,6M	8973	✗ 5826	✓ 41234		32077	26352
3,2M	8092	✗ 5628	✓ 42989		29473	
6,4M	✗ 6871		✓ 41807		25660	
12,8M	✗ 5423		✓ 38284		26817	
25,6M	✗ 1420		22696		✓ 34999	
51,2M			✗ 23476		✓ 37993	





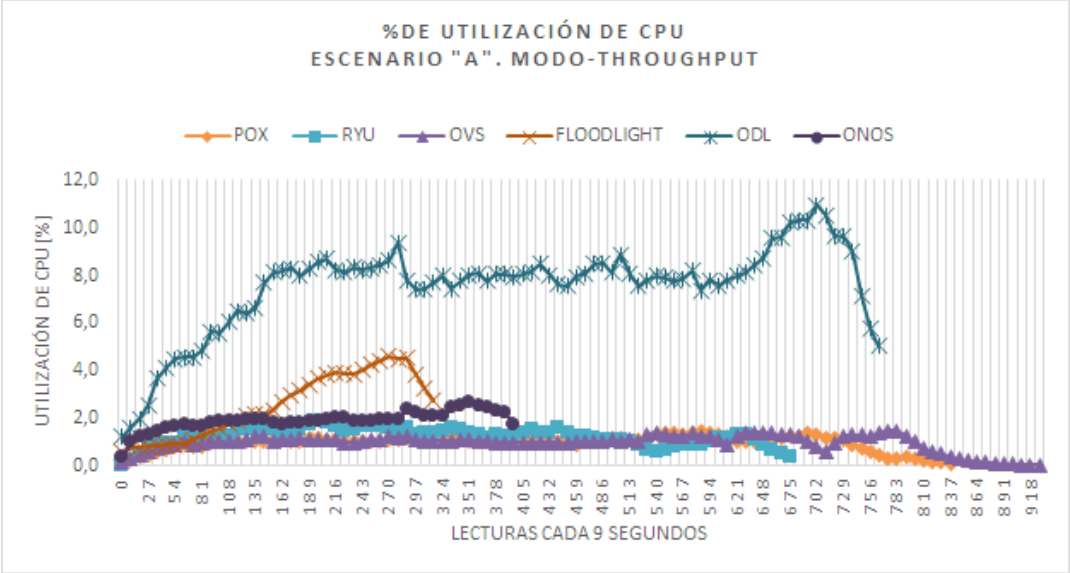


Figura 3.53. Evaluación de los controladores en el escenario "A". Modo throughput. Fuente: Autor.

- Los únicos controladores que pudieron escalar hasta 51,2M de nodos en el escenario creado fueron el OVS y el ODL. Después aparecen los controladores POX, RYU, ONOS, y FLOODLIGHT.
- El controlador con mejor desempeño en la prueba de throughput fue el ONOS alcanzando 46541 respuestas/seg con una red de 200K nodos. Posteriormente aparecen los controladores OVS y el OPENDAYLIGHT. En modo throughput, cada conmutador realiza una petición de nuevo flujo (OFPT_PACKET_IN) y mantiene tantas peticiones pendientes como sus buffers le permiten y por tanto el controlador es sometido a un tráfico muy cercano a la realidad.
- El controlador FLOODLIGHT que en la prueba de latencia había obtenido el segundo mejor resultado, aquí demuestra que no posee suficientes capacidades para manejar grandes cargas de tráfico real por lo que obtiene el peor resultado. Después de consultar con la comunidad de desarrolladores de FLOODLIGHT, Ryan Izard uno de sus miembros plantea que este controlador posee un solo hilo de ejecución en el momento que los mensajes packet_in llegan al controlador. Este hilo de ejecución procesa el mensaje con el módulo "learning_switch" quien los analiza, pone en cola y envía a la librería "Netty" donde finalmente si son procesados en un entorno multi-hilos.

- El desempeño de RYU y POX no dependen de la cantidad de conmutadores mostrando los valores más bajos. Esto se debe mayormente a que estos dos controladores no son multi-hilos y están escrito en Python que es un lenguaje interpretado de alto nivel.

En los experimentos de escalabilidad realizados en 3.4 utilizando Mininet, se demostró que el desempeño de la red tiene relación de dependencia con las características de la plataforma donde corre el controlador. Bajo el principio anterior fue formulada la siguiente interrogante científica:

¿Cómo incrementar la cantidad de respuestas/seg de un controlador SDN?

Para responder a la pregunta fue utilizado el campo de prueba anterior instalando en el escenario ESC_C la herramienta desarrollada y el controlador OPENDAYLIGHT sobre el sistema operativo huésped Ubuntu 14.04 y la plataforma de virtualización ORACLE Virtual Box Versión 5.0.14.

A la máquina virtual le fueron asignados 1, 2, 3, 4 CPU para investigar el impacto en el rendimiento del controlador cuando se incrementa la capacidad de procesamiento donde corre el sistema operativo de la red. En la red creada se incrementa la cantidad de conmutadores en potencias de dos (1, 2, 4, etc.) hasta obtener el máximo tamaño de la red que se puede escalar así como investigar el comportamiento del throughput cuando se incrementa la capacidad de procesamiento. Los resultados del experimento son mostrados en la figura 3.54 donde se aprecia claramente como incrementando la capacidad de procesamiento en el sistema operativo de la red, aumenta la capacidad de procesamiento de tráfico del controlador. Cuando se varía la capacidad de procesamiento de 1 a 4 CPU el factor de multiplicación de la capacidad de tráfico muestra valores entre 3 y 5.5 en redes de variados tamaños. El máximo factor de multiplicación de la capacidad de tráfico se logra cuando la red posee 800K nodos obteniéndose un valor de 5.5. Como promedio analizando redes con cantidad de nodos que van desde 100K hasta 12,8 millones se obtuvo un factor de multiplicación promedio de 3.8. En cuanto a la escalabilidad el valor máximo se obtiene con un CPU llegando a procesar un total de 25.6 millones de nodos pero con un throughput mínimo. Posteriormente cuando los CPU varían de 2 a 4 la escalabilidad alcanza un valor máximo de 12.8 millones de nodos.

NODOS	OPENDAYLIGHT		
	THROUGHPUT [Respuestas/segundo]		
	CPU x CORE x Thread 1x1x1	CPU x CORE x Thread 4x4x4	FACTOR MULTIPLICADOR DE TRÁFICO
100K	5451	17647	3,2
200K	10008	30574	3,1
400K	8548	35213	4,1
800K	7436	40956	5,5
1,6M	7680	31493	4,1
3,2M	7523	27544	3,7
6,4M	7518	24638	3,3
12,8M	7172	25313	3,5

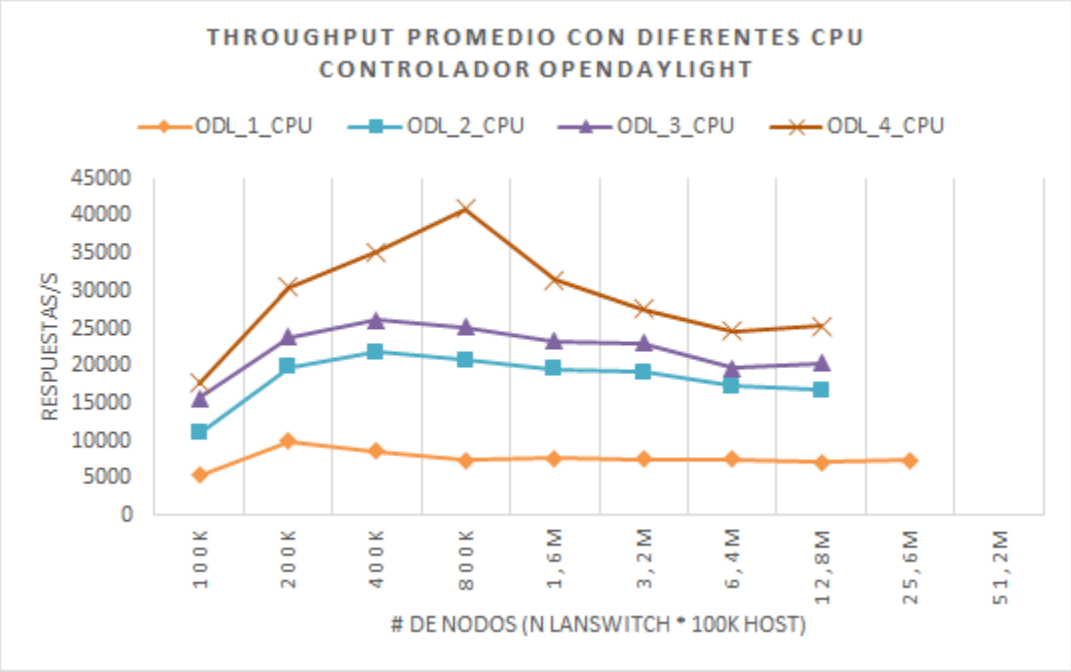


Figura 3.54. Escalabilidad de CPU de ODL en el escenario "A". Modo throughput. Fuente: Autor.

3.6 Conclusiones del capítulo

Actualmente existen más de 50 controladores OpenFlow creados por diferentes grupos de investigación de universidades o proveedores, escritos en diferentes lenguajes de programación, y utilizan técnicas diferentes multi-hilo. Por lo tanto, es necesario crear herramientas y procedimientos que permitan evaluar el desempeño y la latencia de los controladores.

Desde el surgimiento del modelo de conmutador OpenFlow ha sido necesaria una comparación en términos de retardo con los conmutadores tradicionales. En este trabajo fue avaluado el comportamiento de las SDN respecto a una red tradicional analizando la métrica de latencia. Los dos escenarios fueron implementados mediante programación en Python y utilizando el API de MiniNet. En el envío de los dos primeros paquetes ICMP la red tradicional obtuvo mayor retardo, pero a partir del tercer paquete se observa claramente que la red SDN tiene un menor retardo al trabajar en modo proactivo. Esto se debe a que en una red SDN después de instalados los flujos los caminos para encaminar las tramas se almacenan en la memoria cache para lograr un mejor rendimiento. Si es descartado el primer datagrama ICMP y se calcula el valor medio del retardo es posible comprobar que el RTT_tradicional=0,089 ms, RTT_sdn=0,127 ms. Cuando se descarta el primer y el segundo datagrama RTT_tradicional=0,088 ms, RTT_sdn=0,085 ms.

Para evaluar la capacidad de escalar grandes redes con numerosos hosts y conmutadores se realizaron pruebas de escalabilidad en las que se crearon varios script en el Shell bash de Linux para generar nodos y topologías de forma automática, concurrente e interactuando con el emulador MiniNet. Se pudo comprobar, que los tiempos de compilación aumentan de forma exponencial, según el número de nodos de la red, cuestión que también fue comprobada por el autor en [143]. El script creado permite identificar muy rápidamente el mejor (ESC_A) y peor (ESC_B) escenario. Desde el punto de vista de la escalabilidad se pudo comprobar que en las topologías simple y lineal pudieron alcanzar hasta 1024 nodos por el contrario de la de árbol que solo pudo llegar a 512. Sin embargo, Mininet también presenta desventajas. Con las pruebas de comprobación realizadas queda claro que la capacidad de Mininet de emular redes a gran escala es limitada por los recursos disponibles en la maquina en donde corre.

Las aplicaciones SDN, que son consideradas el "cerebro de la red" pueden tener en tiempo real una visión global de la misma a través de la interfaz NBI de los controladores. Utilizando esta información, las aplicaciones SDN pueden implementar mediante la programación estrategias para manipular las redes físicas subyacentes utilizando un lenguaje de alto nivel proporcionado por la capa de control. En este trabajo se implementó una aplicación SDN que permite detectar intrusos y controlar el acceso a los recursos de la red. La popularidad de SDN en el mundo de TI hoy se demuestra por los numerosos casos de uso que proliferan en esta área. Aunque existen cortafuegos excepcionales en el mercado, es bastante costoso para las empresas instalar numerosos cortafuegos en toda la red para garantizar una alta seguridad. Si se fuera a realizar esta misma implementación en una red tradicional sería necesario configurar cada conmutador por separado con la consiguiente posibilidad de generar problemas en toda la red por errores de configuración. El mismo problema se presentaría si se quisieran eliminar o crear nuevos usuarios en las listas de control de acceso. En SDN con la separación del control de los elementos de la capa de infraestructura se logra una flexibilidad en la red sin precedentes que crea enormes potencialidades en el campo de la seguridad de las redes.

Fue desarrollada una aplicación para evaluar controladores SDN/OpenFlow que utiliza los resultados de cbench e incorpora otras métricas. Con esta aplicación son evaluados los controladores POX, RYU, OVS, ONOS, FLOODLIGHT y OPENDAYLIGHT en cuanto a la latencia y cantidad de respuestas por segundos. El controlador OVS en el modo latencia obtuvo los mejores resultados con 56867 respuestas/s. Le siguieron los controladores FLOODLIGHT, OPENDAYLIGHT, ONOS, y finalmente RYU y POX mostrando los peores desempeños. El controlador OVS tiene mejores resultados porque como es un controlador de prueba de entorno académico realiza muy pocas funciones comparadas con un controlador profesional además de estar realizado en lenguaje “C” que es compilado y posee mejor rendimiento para aplicaciones de tiempo real. Las herramientas de comprobación desarrolladas en esta memoria son útiles para probar el rendimiento del controlador en el bucle entrada/salida. Para realizar una prueba justa fue necesario estudiar cada controlador con profundidad y encontrar las aplicaciones SDN equivalentes.

El controlador con mejor desempeño en la prueba de throughput fue el ONOS alcanzando 46541 respuestas/seg con una red de 200K nodos. Posteriormente aparecen los controladores OVS y el OPENDAYLIGHT. El controlador FLOODLIGHT que en la prueba de latencia había obtenido el segundo mejor resultado, aquí demuestra que no posee suficientes capacidades para manejar grandes cargas de tráfico real por lo que obtiene el peor resultado.

Finalmente para incrementar la cantidad de respuestas/seg de un controlador SDN se preparó un campo de pruebas que permite incrementar la capacidad del procesamiento del controlador ODL en el escenario ESC_C. Los resultados del experimento muestran que al incrementar la capacidad de procesamiento en el sistema operativo de la red, aumenta la capacidad de procesamiento de tráfico del controlador. Cuando se varía la capacidad de procesamiento de 1 a 4 CPU el factor de multiplicación de la capacidad de tráfico muestra valores entre 3 y 5.5 en redes de variados tamaños. Si esta variación de la capacidad de procesamiento se realiza de forma dinámica la red puede ajustarse elásticamente a las variaciones de tráfico sin la necesidad de incrementar elementos de hardware que usualmente son costosos.

Los resultados del experimento son mostrados en la figura 3.54 done se aprecia claramente como incrementando la capacidad de procesamiento en el sistema operativo de la red, aumenta la capacidad de procesamiento de tráfico del controlador.

Según las pruebas realizadas, los controladores OPENDAYLIGHT y ONOS son los de mejor desempeño; motivo por el cual proveedores importantes como CISCO, HUAWEI, Avaya, Brocade, HP, IBM y Ericsson los han seleccionado como la base de los controladores de código abierto que ellos ofertan.

CONCLUSIONES

Las SDN basadas en el protocolo OpenFlow se han distinguido en los últimos años como una arquitectura con grandes potencialidades de sustituir las redes actuales lo que sugiere la necesidad de profundizar en su estudio y evaluación.

A partir del cumplimiento del sistema de objetivos se arriba a las siguientes conclusiones:

1. El presente proyecto contribuye a la sistematización de los conocimientos sobre las SDN basadas en el protocolo OpenFlow. Fueron abordadas las principales características y potencialidades de las Redes Definidas por Software, así como sus paradigmas; necesarios para la comprensión y el correcto desarrollo de futuras investigaciones.
2. La estandarización de todos los aspectos del desempeño de las SDN aún constituye tema de estudio para la ONF y el IETF. Existen muchas métricas definidas para las redes SDN, pero en este trabajo fueron evaluadas: la tasa de procesamiento de mensajes asincrónicos, la latencia y la escalabilidad del controlador.
3. Los principales elementos a tener en consideración para la selección de controladores SDN son: soporte del protocolo OpenFlow, aplicaciones, virtualización de red, funcionalidad de la red, escalabilidad, procesamiento, rendimiento, programación de red, confiabilidad, seguridad de la red, monitorización centralizada y visualización, experiencia del fabricante y el soporte de plataformas.
4. Al culminar esta memoria se dispone de una plataforma de pruebas experimental que posibilita el estudio y evaluación de diferentes tipos de controladores, conmutadores, plataformas de cómputo y redes SDN reales; además del desarrollo de aplicaciones.
5. La implementación de una plataforma de pruebas y el diseño de experimentos sobre la misma permitió comprobar:
   - ✓ Las SDN pueden reducir la latencia de la red; cuestión demostrada en esta investigación al comparar este tipo de redes con las tradicionales. Cuando se descarta el primer y el segundo datagrama en la red tradicional se obtiene una latencia de 0,088 ms y en la SDN un valor de 0,085 ms.
   - ✓ Las pruebas de escalabilidad realizadas con las herramientas desarrolladas permitieron evaluar la capacidad de escalar grandes redes SDN con numerosos hosts y conmutadores. En ellas también se pudo comprobar que el escenario “ESC_A” y específicamente los conmutadores ovsk y ovsbr poseen los menores tiempos de compilación de la red; y la máxima cantidad de nodos alcanzados con 1025.
   - ✓ Existe una estrecha relación de dependencia entre la capacidad de la plataforma de cómputo, la topología de la red y el máximo tamaño de la misma. Lo anterior quedó demostrado en los resultados de las pruebas de escalabilidad a la topología de árbol en la que se escala solamente hasta 512 nodos.
   - ✓ Con las pruebas de comprobación realizadas se demuestra que la capacidad de Mininet de emular redes a gran escala es limitada por los recursos disponibles en la plataforma de cómputo.
   - ✓ El desarrollo de una aplicación SDN por parte de los autores de esta memoria utilizando la API del controlador POX permitió detectar intrusos y controlar el acceso a los recursos de una red. Se pudo comprobar que la separación del control de los elementos de la capa de infraestructura logra una flexibilidad en la red sin precedentes que crea enormes potencialidades en el campo de la seguridad

de las redes.

- ✓ Las aplicaciones de seguridad cuando son implementadas en el modo reactivo generan una latencia en la red que se incrementa con el tamaño de la misma. La aplicación de políticas de seguridad en el modo reactivo, tiene un impacto negativo en la tasa de transferencia entre los nodos de la red. La tasa de transferencia disminuye en la medida que la red crece.
- ✓ En la evaluación de los controladores POX, RYU, OVS, ONOS, FLOODLIGHT y OPENDAYLIGHT se pudo comprobar que el OVS en el modo latencia obtuvo el mejor resultado con 56867 respuestas/seg. Le siguieron los controladores FLOODLIGHT, OPENDAYLIGHT, ONOS, y finalmente RYU y POX mostrando los peores desempeños.
- ✓ El controlador con mejor desempeño en la prueba de throughput fue el ONOS alcanzando 46541 respuestas/seg con una red de 200K nodos. El controlador FLOODLIGHT que en la prueba de latencia había obtenido el segundo mejor resultado, aquí demuestra que no posee suficientes capacidades para manejar grandes cargas de tráfico real por lo que obtiene el peor resultado.
- ✓ La evaluación de controladores SDN requiere de un estudio profundo de cada sistema operativo de red, así como de las aplicaciones de red equivalentes para realizar una comparación justa.
- ✓ Cuando se varía la capacidad de procesamiento en la plataforma de virtualización de 1 a 4 CPU el factor de multiplicación de la capacidad de tráfico en el controlador muestra valores entre 3 y 5.5 en redes de variados tamaños. Si el incremento de la capacidad de procesamiento se realiza de forma dinámica en plataformas de virtualización tales como NFV; la red puede ajustarse elásticamente a las variaciones de tráfico sin la necesidad de incrementar elementos de hardware que usualmente son costosos. La asignación de recursos bajo demanda es uno de los principales aportes de NFV con enorme potencial para hospedar el sistema operativo de la red, multiplicar la escalabilidad, disminuir la latencia, garantizar fiabilidad y contribuir al ahorro de energía.

6. La plataforma de prueba y las herramientas desarrolladas en esta memoria, contribuyen a sistematizar la toma de decisiones relacionadas con la selección y análisis del desempeño de los controladores SDN. Los controladores ONOS y OPENDAYLIGHT finalmente fueron los que mostraron mejor desempeño; coincidiendo este resultado con la visión de importantes proveedores como CISCO, HUAWEI, Avaya, Brocade, HP, IBM y Ericsson quienes los han seleccionado como la base de los controladores de código abierto que ellos ofertan.

Después de implementada la plataforma de pruebas, varias herramientas para el estudio, evaluación de las SDN y comprobada su factibilidad, se considera resuelto el problema de la investigación y cumplido el objetivo perseguido en este trabajo científico.

RECOMENDACIONES

Se considera que las siguientes recomendaciones pueden ser de utilidad para enriquecer el estudio realizado y los resultados obtenidos:

- En futuras investigaciones se recomienda comprobar que la diferencia en cuanto a la latencia entra las redes SDN y las tradicionales son más acentuadas en la medida que la red sea más grande.
- Realizar la evaluación de otros controladores utilizando las mismas herramientas desarrolladas en la presente investigación y comparar los resultados.
- Implementar las VNF asociadas a los controladores evaluados en esta memoria en la infraestructura de funciones de red virtualizadas (NFVI) en laboratorio experimental DPI ETECSA [107]. Realizar la evaluación de los controladores SDN utilizando las mismas herramientas desarrolladas en la presente investigación y comparar los resultados. Realizar también pruebas de asignación dinámica de CPU y memoria para medir la latencia, el throughput y el máximo tamaño de las redes que se pueden escalar.
- Futuras investigaciones pueden mejorar aún más la lógica implementada en la aplicación SDN de seguridad creada en esta memoria, incorporando otras capacidades de las SDN mediante la observación de todo el flujo de la red con el fin de bloquear eficazmente los ataques a la red en la capa de infraestructura sin la necesidad de realizar una inspección profunda de paquetes.

REFERENCIAS BIBLIOGRAFICAS

[1] Y. Li y M. Chen, «Software-Defined Network Function Virtualization: A Survey», *IEEE Access*, vol. 3, pp. 2542-2553, 2015.

[2] Y. Jarraya, T. Madi, y M. Debbabi, «A Survey and a Layered Taxonomy of Software-Defined Networking», *IEEE Commun. Surv. Tutor.*, vol. 16, n.º 4, pp. 1955-1980, Fourthquarter 2014.

[3] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, y S. Uhlig, «Software-Defined Networking: A Comprehensive Survey», *Proc. IEEE*, vol. 103, n.º 1, pp. 14-76, ene. 2015.

[4] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, y T. Turletti, «A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks», *IEEE Commun. Surv. Tutor.*, vol. 16, n.º 3, pp. 1617-1634, Third 2014.

[5] S. Higginbotham, «Google launches Andromeda, a software defined network underlying its cloud», 02-abr-2014. [En línea]. Disponible en: https://gigaom.com/. [Accedido: 12-nov-2016].

[6] Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, y A. Rindos, «SDSecurity: A Software Defined Security experimental framework», en *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015, pp. 1871-1876.

[7] T. Kim, T. Koo, y E. Paik, «SDN and NFV benchmarking for performance and reliability», en *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, 2015, pp. 600-603.

[8] P. Isaia y L. Guan, «Performance benchmarking of SDN experimental platforms», en *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 116-120.

[9] A. Kondel y A. Ganpati, «Evaluating System Performance for handling scalability challenge in SDN», en *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, 2015, pp. 594-597.

[10] M. Jarschel, F. Lehrieder, Z. Magyari, y R. Pries, «A Flexible OpenFlow-Controller Benchmark», en *2012 European Workshop on Software Defined Networking (EWSDN)*, 2012, pp. 48-53.

[11] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, y R. Smeliansky, «Advanced study of SDN/OpenFlow controllers», en *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, 2013, p. 1.

[12] Open Networking Foundation, «Software-Defined Networking: The New Norm for Networks», 2012. [En línea]. Disponible en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf. [Accedido: 23-oct-2015].

[13] Z. Hu, M. Wang, X. Yan, Y. Yin, y Z. Luo, «A comprehensive security architecture for SDN», en *2015 18th International Conference on Intelligence in Next Generation Networks (ICIN)*, 2015, pp. 30-37.

[14] ONF, «2016 Predictions. | ONF BLOG». [En línea]. Disponible en: https://www.opennetworking.org/. [Accedido: 29-oct-2015].

[15] J. R. de Almeida Amazonas, G. Santos-Boada, y J. Solé-Pareta, «A critical review of OpenFlow/SDN-based networks», en *2014 16th International Conference on Transparent Optical Networks (ICTON)*, 2014, pp. 1-5.

[16] K. Govindarajan, K. C. Meng, y H. Ong, «A literature review on Software-Defined Networking (SDN) research topics, challenges and solutions», en *2013 Fifth International Conference on Advanced Computing (ICoAC)*, 2013, pp. 293-299.

[17] B. Lantz, B. Heller, y N. McKeown, «A network in a laptop: rapid prototyping for software-defined networks», pp. 1–6, 2010.

[18] F. Hu, Q. Hao, y K. Bao, «A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation», *IEEE Commun. Surv. Tutor.*, vol. 16, n.º 4, pp. 2181-2206, Fourthquarter 2014.

[19] W. Xia, Y. Wen, C. H. Foh, D. Niyato, y H. Xie, «A Survey on Software-Defined Networking», *IEEE Commun. Surv. Tutor.*, vol. 17, n.º 1, pp. 27-51, Firstquarter 2015.

[20] A. G. Centeno, C. M. R. Vergel, C. A. Calderón, y F. C. C. Bondarenko, «Controladores SDN, elementos para su selección y evaluación.», *Rev. Telem Tica*, vol. 13, n.º 3, pp. 10–20, 2014.

[21] Carlos Fernández y Jose L. Muñoz, «Openflow 1.3 with RYU controller». [En línea]. Disponible en: http://upcommons.upc.edu/. [Accedido: 20-ene-2016].

[22] «SDN, la evolución que necesitaba la red | Datacenter Dynamics». [En línea]. Disponible en: http://www.datacenterdynamics.es/focus/archive/2013/03/sdn-la-evoluci%C3%B3n-que-necesitaba-la-red. [Accedido: 26-ene-2016].

[23] CISCO, «Software-Defined Networking: Why We Like It and How We Are Building On It». [En línea]. Disponible en: http://www.cisco.com/. [Accedido: 26-ene-2016].

[24] ONF, «ONF TR-502 SDN architecture». jun-2014.

[25] ONF, «OpenFlow - Open Networking Foundation». [En línea]. Disponible en: https://www.opennetworking.org/sdn-resources/openflow. [Accedido: 29-oct-2015].

[26] ONF, «SDN Architecture overview». [En línea]. Disponible en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf. [Accedido: 29-oct-2015].

[27] Fabian Schneider, «SDN Standardization in ONF:ONF Structure, OpenFlow Spec & ONF's SDN Architecture». 2013.

[28] M. Boucadair y C. Jacquenet, «Software-Defined Networking: A Perspective from within a Service Provider Environment». [En línea]. Disponible en: https://tools.ietf.org/html/rfc7149. [Accedido: 29-oct-2015].

[29] «Software-Defined Networking (SDN) Definition - Open Networking Foundation». [En línea]. Disponible en: https://www.opennetworking.org/sdn-resources/sdn-definition. [Accedido: 29-oct-2015].

[30] S. Denazis, E. Haleplidis, J. H. Salim, O. Koufopavlou, D. Meyer, y K. Pentikousis, «Software-Defined Networking (SDN): Layers and Architecture Terminology», 2015.

[31] S. Scott-Hayward, S. Natarajan, y S. Sezer, «A Survey of Security in Software Defined Networks», *IEEE Commun. Surv. Tutor.*, vol. PP, n.º 99, pp. 1-1, 2015.

[32] A. Lara, A. Kolasani, y B. Ramamurthy, «Network Innovation using OpenFlow: A Survey», *IEEE Commun. Surv. Tutor.*, vol. 16, n.º 1, pp. 493-512, First 2014.

[33] I. Ahmad, S. Namal, M. Ylianttila, y A. Gurtov, «Security in Software Defined Networks: A Survey», *IEEE Commun. Surv. Tutor.*, vol. PP, n.º 99, pp. 1-1, 2015.

[34] A. Blenk, A. Basta, M. Reisslein, y W. Kellerer, «Survey on Network Virtualization Hypervisors for Software Defined Networking», *IEEE Commun. Surv. Tutor.*, vol. PP, n.º 99, pp. 1-1, 2015.

[35] R. Enns, M. Bjorklund, J. Schoenwaelder, y A. Bierman, «Network Configuration Protocol (NETCONF)», RFC Editor, RFC6241, jun. 2011.

[36] Open Networking Foundation, «OpenFlow-enabled Transport SDN», 27-may-2014. [En línea]. Disponible en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-of-enabled-transport-sdn.pdf. [Accedido: 27-jun-2016].

[37] Y. Rekhter y T. Li, «A Border Gateway Protocol 4 (BGP-4)». [En línea]. Disponible en: https://tools.ietf.org/html/rfc1771. [Accedido: 27-jun-2016].

[38] ONF, «OpenFlow Switch Specification», oct-2013. [En línea]. Disponible en: https://www.opennetworking.org/images/stories/downloads/sdnresources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf. [Accedido: 12-oct-2016].

[39] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, y R. Sidi, «A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains.» .

[40] W. Stallings, «Software-defined networks and openflow», *Internet Protoc. J.*, vol. 16, n.º 1, pp. 2–14, 2013.

[41] M. Garcia, A. Bessani, I. Gashi, N. Neves, y R. Obelheiro, «Analysis of operating system diversity for intrusion tolerance», *Softw. Pract. Exp.*, vol. 44, n.º 6, pp. 735–770, 2014.

[42] T. Koponen *et al.*, «Onix: A Distributed Control Platform for Large-scale Production Networks.», en *OSDI*, 2010, vol. 10, pp. 1–6.

[43] E. Haleplidis *et al.*, «Network Programmability With ForCES», *IEEE Commun. Surv. Tutor.*, vol. 17, n.º 3, pp. 1423-1440, thirdquarter 2015.

[44] M. Garcia, A. Bessani, I. Gashi, N. Neves, y R. Obelheiro, «OS diversity for intrusion tolerance: Myth or reality?», en *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, 2011, pp. 383-394.

[45] K. Phemius, M. Bouet, y J. Leguay, «Disco: Distributed multi-domain sdn controllers», en *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–4.

[46] M. Canini, P. Kuznetsov, D. Levin, y S. Schmid, «A distributed and robust SDN control plane for transactional network updates», en *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 190-198.

[47] F. Botelho, A. Bessani, F. M. V. Ramos, y P. Ferreira, «On the Design of Practical Fault-Tolerant SDN Controllers», en *2014 Third European Workshop on Software Defined Networks*, 2014, pp. 73-78.

[48] Sarwar Raza(HP) y David Lenrow (Plexxi), «Northbound Interfaces». 10-jun-2013.

[49] M. Pham y D. B. Hoang, «SDN applications - The intent-based Northbound Interface realisation for extended applications», 2016, pp. 372-377.

[50] ONF - Open Networking Foundation., «TR-535 ONF SDN Evolution». [En línea]. Disponible en: https://www.opennetworking.org/sdn-resources/technical-library. [Accedido: 14-oct-2016].

[51] L. Li, W. Chou, W. Zhou, y M. Luo, «Design Patterns and Extensibility of REST API for Networking Applications», *IEEE Trans. Netw. Serv. Manag.*, vol. 13, n.º 1, pp. 154-167, mar. 2016.

[52] «The SDN Gold Rush To The Northbound API», *SDxCentral*, 16-nov-2012. [En línea]. Disponible en: https://www.sdxcentral.com/articles/contributed/the-sdn-gold-rush-to-the-northbound-api/2012/11/. [Accedido: 13-oct-2016].

[53] S. Rivera, Z. Fei, y J. Griffioen, «RAPTOR: A REST API translaTOR for OpenFlow controllers», en *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 328-333.

[54] C. Trois, M. D. D. D. Fabro, L. C. E. de Bona, y M. Martinello, «A Survey on SDN Programming Languages: Towards a Taxonomy», *IEEE Commun. Surv. Tutor.*, vol. PP, n.º 99, pp. 1-1, 2016.

[55] M. Yu, A. Wundsam, y M. Raju, «NOSIX: A lightweight portability layer for the SDN OS», *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, n.º 2, pp. 28–35, 2014.

[56] «A primer on northbound APIs: Their role in a software-defined network», *SearchSDN*. [En línea]. Disponible en: http://searchsdn.techtarget.com/feature/A-primer-on-northbound-APIs-Their-role-in-a-software-defined-network. [Accedido: 14-oct-2016].

[57] S. T. Ali, V. Sivaraman, A. Radford, y S. Jha, «A Survey of Securing Networks Using Software Defined Networking», *IEEE Trans. Reliab.*, vol. 64, n.º 3, pp. 1086-1097, sep. 2015.

[58] «SDN-Intent NBI for Software Defined Networking». [En línea]. Disponible en: http://developer.huawei.com/en/ict/Products/SDN/Components/Forum/en-IntentNBIforSoftwareDefinedNetworking. [Accedido: 14-oct-2016].

[59] «The Frenetic Project». [En línea]. Disponible en: http://frenetic-lang.org/pyretic/. [Accedido: 14-oct-2016].

[60] O. Salman, I. H. Elhajj, A. Kayssi, y A. Chehab, «SDN controllers: A comparative study», en *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, 2016, pp. 1-6.

[61] V. Gupta, «Performance Analysis Of Python Based OpenFlow Controllers».

[62] P. Tanwar, R. Gohil, y M. Tanwar, «Quick Survey of Benefits from Control Plane and Data Plane Separation in Software-Defined Networking», *BVICAMs Int. J. Inf. Technol.*, vol. 8, n.º 1, 2016.

[63] D. Turull, M. Hidell, y P. Sjödin, «Evaluating OpenFlow in libnetvirt», en *The 8th Swedish National Computer Networking Workshop 2012 (SNCNW 2012)*, 2012.

[64] D. Turull, M. Hidell, y P. Sjödin, «libNetVirt: The network virtualization library», en *2012 IEEE International Conference on Communications (ICC)*, 2012, pp. 5543-5547.

[65] R. Khondoker, A. Zaalouk, R. Marx, y K. Bayarou, «Feature-based comparison and selection of Software Defined Networking (SDN) controllers», en *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 2014, pp. 1-7.

[66] F. A. Lopes, M. Santos, R. Fidalgo, y S. Fernandes, «A Software Engineering Perspective on SDN Programmability», *IEEE Commun. Surv. Tutor.*, vol. 18, n.º 2, pp. 1255–1272, 2016.

[67] Open Networking Foundation, «OpenFlow Switch Specification Version 1.5.0», 19-dic-2014. [En línea]. Disponible en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[68] Open Networking Foundation, «OpenFlow Switch Specification 1.0.0», 31-dic-2009. [En línea]. Disponible en: https://www.opennetworking.org/ images/stories/ downloads/sdn-resources/onf-specifications/openflow/ openflow-spec-v1.0.0.pdf.

[69] Ivan Pepelnjak, «Flow Table Explosion with OpenFlow 1.0 (And Why We Need OpenFlow 1.3)», *ipspace*, 21-oct-2013. .

[70] C. Ching-Hao y Y.-D. Lin, «OpenFlow Version Roadmap», 2015.

[71] Open Networking Foundation, «OpenFlow Table Type Patterns». 15-ago-2014.

[72] Open Networking Foundation, «OpenFlow Switch Specification 1.2». 05-dic-2011.

[73] Open Networking Foundation, «OpenFlow Switch Specifcation 1.4.0». 2013.

[74] N. Gude *et al.*, «NOX: towards an operating system for networks», *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, n.º 3, pp. 105–110, 2008.

[75] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, y R. Sherwood, «On Controller Performance in Software-Defined Networks», en *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.

[76] D. Erickson, «The beacon openflow controller», en *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 13–18.

[77] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, y R. Smeliansky, «Advanced study of SDN/OpenFlow controllers», en *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, 2013, p. 1.

[78] L. Schiff, S. Schmid, y P. Kuznetsov, «In-Band Synchronization for Distributed SDN Control Planes», *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, n.º 1, pp. 37–43, 2016.

[79] Y. Zhou *et al.*, «A Load Balancing Strategy of SDN Controller Based on Distributed Decision», en *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2014, pp. 851-856.

[80] «2015 SDN Controllers Report», *SDxCentral*. [En línea]. Disponible en: https://www.sdxcentral.com/reports/sdn-controllers-report-2015/. [Accedido: 23-oct-2015].

[81] M. Ashton y others, «Ten Things to Look for in an SDN Controller», *White Pap. Available Online Www Necam ComDocs*, 2013.

[82] «Floodlight OpenFlow Controller -», *Project Floodlight*. [En línea]. Disponible en: http://www.projectfloodlight.org/floodlight/. [Accedido: 14-may-2016].

[83] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, y S. A. Mehdi, «An architectural evaluation of SDN controllers», en *2013 IEEE International Conference on Communications (ICC)*, 2013, pp. 3504-3508.

[84] Noxrepo, «NOXRepo», 2016. .

[85] T. N. D. y K. Gray, *SDN: Software Defined Networks*, 1st ed. O'Reilly Media, Inc., 2013.

[86] «Ryu SDN Framework», 2016. [En línea]. Disponible en: http://osrg.github.io/ryu/. [Accedido: 14-may-2016].

[87] T. Bakhshi, «State of the Art and Recent Research Advances in Software Defined Networking».

[88] S.-Y. Wang, C.-L. Chou, y C.-M. Yang, «EstiNet openflow network simulator and emulator», *IEEE Commun. Mag.*, vol. 51, n.º 9, pp. 110-117, sep. 2013.

[89] S.-Y. Wang, «Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet», en *2014 IEEE Symposium on Computers and Communication (ISCC)*, 2014, pp. 1-6.

[90] «OMNeT++ Discrete Event Simulator - Home». [En línea]. Disponible en: https://omnetpp.org/. [Accedido: 03-nov-2016].

[91] «ofsoftswitch13 - CPqD», *GitHub*. [En línea]. Disponible en: https://github.com/CPqD/ofsoftswitch13. [Accedido: 04-nov-2016].

[92] «Pica8». [En línea]. Disponible en: http://www.pica8.com/. [Accedido: 04-nov-2016].

[93] «OpenFlow Wiki - OpenFlow Wiki». [En línea]. Disponible en: http://archive.openflow.org/wk/index.php/Main_Page. [Accedido: 04-nov-2016].

[94] «Sdn troubleshooting simulator». [En línea]. Disponible en: http://ucb-sts.github.io/sts/. [Accedido: 04-nov-2016].

[95] N. K. Jha, N. Agarwal, y P. Singh, «Realization of congestion in software defined networks», en *2015 International Conference on Computing, Communication Automation (ICCCA)*, 2015, pp. 535-539.

[96] C. Rotsos, G. Antichi, M. Bruyere, P. Owezarski, y A. W. Moore, «OFLOPS-Turbo: Testing the next-generation OpenFlow switch», en *2015 IEEE International Conference on Communications (ICC)*, 2015, pp. 5571-5576.

[97] S. Zeng, P. Zheng, y Y. Zhang, «Design of Test Case for OpenFlow Protocol Conformance Test Based on OFTest», en *2016 International Symposium on Computer, Consumer and Control (IS3C)*, 2016, pp. 465-470.

[98] A. Wundsam, D. Levin, S. Seetharaman, A. Feldmann, y others, «OFRewind: Enabling Record and Replay Troubleshooting for Networks.», en *USENIX Annual Technical Conference*, 2011.

[99] Z. K. Khattak, M. Awais, y A. Iqbal, «Performance evaluation of OpenDaylight SDN controller», en *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2014, pp. 671-676.

[100] A. Basil, B. Vengainathan, V. Manral, M. Tassinari, y S. Banks, «Terminology for Benchmarking SDN Controller Performance», 2016. [En línea]. Disponible en: https://tools.ietf.org/html/draft-ietf-bmwg-sdn-controller-benchmark-term-01. [Accedido: 18-may-2016].

[101] A. Basil, B. Vengainathan, V. Manral, M. Tassinari, y S. Banks, «Benchmarking Methodology for SDN Controller Performance», 2016. [En línea]. Disponible en: https://tools.ietf.org/html/draft-ietf-bmwg-sdn-controller-benchmark-meth-01. [Accedido: 18-may-2016].

[102] «ONF TR-507. Migration Tools and Metrics», 2014. [En línea]. Disponible en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/migration-tools-and-metrics.pdf. [Accedido: 18-may-2016].

[103] K. Phemius y M. Bouet, «Monitoring latency with OpenFlow», en *2013 9th International Conference on Network and Service Management (CNSM)*, 2013, pp. 122-125.

[104] R. L. S. de Oliveira, A. A. Shinoda, C. M. Schweitzer, y L. Rodrigues Prete, «Using Mininet for emulation and prototyping Software-Defined Networks», en *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014, pp. 1-6.

[105] Z. K. Khattak, M. Awais, y A. Iqbal, «Performance evaluation of OpenDaylight SDN controller», en *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2014, pp. 671-676.

[106] «Oflops - OpenFlow Wiki». [En línea]. Disponible en: http://archive.openflow.org/wk/index.php/Oflops. [Accedido: 30-abr-2016].

[107] Abel Alfonso López Carbonell y Hector Cruz Enriquez, «PLATAFORMA DE GESTIÓN NFV IaaS PARA EL OPERADOR DE TELECOMUNICACIONES ETECSA», UCLV, 2016.

[108] «Software-Defined Networking Products - Open SDN Network», *Avaya*, 18-abr-2016. [En línea]. Disponible en: http://www.avaya.com/usa/products/software-defined-networking/. [Accedido: 13-may-2016].

[109] «Big Switch Networks, Inc.», *Big Switch Networks, Inc.* [En línea]. Disponible en: http://bigswitch.com/. [Accedido: 13-may-2016].

[110] Mark Fabbi, «Brite-Box and Software-Defined Networking Are Driving Innovation and Data Center Network Savings», 06-nov-2015. [En línea]. Disponible en: https://www.gartner.com/. [Accedido: 13-may-2016].

[111] wlumabas, «SDN and OpenDaylight», *Brocade*. [En línea]. Disponible en: http://www.brocade.com/en.html. [Accedido: 13-may-2016].

[112] «Huawei Announces One SDN Controller For Campus, Data Center, WAN & IoT - Packet Pushers -», *Packet Pushers*, 01-sep-2016. [En línea]. Disponible en: http://packetpushers.net/huawei-announces-one-sdn-controller-campus-data-center-wan-iot/. [Accedido: 28-oct-2016].

[113] C. Corporation, «Ciena to Showcase Blue Planet Software at SDN & OpenFlow World Congress», *Ciena*. [En línea]. Disponible en: http://www.ciena.com/about/newsroom/press-releases/Ciena-to-Showcase-Blue-Planet-Software-at-SDN-OpenFlow-World-Congress.html. [Accedido: 13-may-2016].

[114] Cisco, «Cisco Open SDN Controller 1.2 Data Sheet - Cisco», 09-oct-2015. [En línea]. Disponible en: http://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/open-sdn-controller/datasheet-c78-733458.html. [Accedido: 13-may-2016].

[115] Coriant, «Coriant Transcend™ SDN Solution | Software Defined Networking | Coriant», 03-abr-2016. [En línea]. Disponible en: https://www.coriant.com/solutions/technologies/sdn.asp. [Accedido: 13-may-2016].

[116] Ciena, «Ciena Completes Acquisition of Cyan», *Ciena*. [En línea]. Disponible en: http://www.ciena.com/about/newsroom/press-releases/Ciena-Cyan-Acquisition-Now-Closed.html. [Accedido: 13-may-2016].

[117] Ericsson, «Software-Defined Networking (SDN)», *Ericsson.com*, 24-jul-2015. [En línea]. Disponible en: http://www.ericsson.com/spotlight/cloud/network-modernization/sdn. [Accedido: 13-may-2016].

ANEXOS

ANEXO 1: FRAGMENTO DEL CÓDIGO FUENTE DE LA APLICACIÓN DE ESCALABILIDAD DE LA TOPOLOGIA SIMPLE

```
#!/bin/bash
#Limpiar la pantalla
clear
echo "------------------------------------------------------------"
echo "|TITULO:PRUEBA DE ESCALABILIDAD DE SDN EN MININET          |"
echo "|Autor: Ing. Yanko Antonio Marin Muro                      |"
echo "|Tutor: Dr. C. Ing. Felix Alvarez Paliza                   |"
echo "|email:yanko.marin@etecsa.cu ; yanko.antonio@gmail.com     |"
echo "|      fapaliza@uclv.edu.cu                                |"
echo "------------------------------------------------------------"

echo "------------------------------------------------------------"
echo "|DATOS GENERALES                                           |"
echo "------------------------------------------------------------"
echo "KERNEL"
echo $SHELL
sudo lsb_release -idc
echo "------------------------------------------------------------"
echo "|PROCESADOR Y MEMORIA INSTALADA                            |"
echo "------------------------------------------------------------"
sudo lshw -short | grep "processor\|memory"

sudo lscpu



#Si MiniNet se bloquea por alguna razón, inicializar las variables:
echo "Inicializando MiniNet..."
sudo mn -c > /dev/null 2>&1     #para que no salga por pantalla.


#DATOS PARA LA SIMULACION
read -p "Modo resumen (s/n): " resumen
read -p "Topologia single o linear): " topologia
read -p "Cantidad de host de la topologia : " chost
read -p "Simulaciones por topologia: " simxtopo
read -p "Tiempo entre simulaciones (s): " esperar
read -p "Comenzar en: " inicio
read -p "Numero de saltos (1,2,3..) o (p2 para potencias de 2): " saltos
read -p "Tipo de Lanswitch (ovsk,ovsbr,user): " lanswitch
read -p "Controlador (ref,remote,): " controlador
hoy=$(date "+%Y-%m-%d %H:%M")

```

```
43
44 if [ $resumen = 's' ]
45 then
46 zdump EST
47         echo "-------------------------------------------------------------------------------------------------------"
48         echo "|TOPOLOGIA $topologia,OpenFlow10, switch=$lanswitch, Controlador=$controlador                        |"
49         echo "|TOPOLOGIA: sudo mn --topo $topologia,$chost --mac --switch $lanswitch --controller $controlador   |"
50         echo "|RESULTADO:TIEMPO DE INICIO/FIN DE LA TOPOLOGIA PARA CADA VALOR DE n                                |"
51         echo "|RESULTADO:CPU,MEMORIA TOTAL,OCUPADA. Mem: total-used-free-shared-buffers-cached                    |"
52         echo "|(Modo Resumen para estadistica):                                                                   |"
53         echo "-------------------------------------------------------------------------------------------------------"
54         sudo mn -c > /dev/null 2>&1     #Inicializando MiniNet
55         p=$inicio
56         while [ $p -lt `expr $chost \+ 1` ]
57         do
58                 c=0
59                 while [ $c -lt $simxtopo ]
60                         do
61                                 simulacion=$(echo "h1 bash /home/simulaciones/bash_script/host_script/host_script 10.0.0.2 |
62                                         grep ^Resultado | awk '{print \"Resultado: \" \$0}' \nexit" | \
63                                         sudo mn --topo $topologia,$p --switch $lanswitch
64                                         --controller $controlador 2>&1 | \grep "Resultado\|completed")
65                                 if [ $topologia = "single" ]
66                                 then
67                                         cantidad_nodos=`expr $p \+ 1`
68                                         cantidad_Switch=1
69                                         cantidad_host=`expr $p`
70                                 else
71                                         cantidad_nodos=`expr $p \* 2`
72                                         cantidad_Switch=`expr $p`
73                                         cantidad_host=`expr $p`
74                                 fi
75                                 simulacion_final=$(echo $simulacion | sed -e 's/\r//g')
76                                 HORA=$(date "+%Y-%m-%d %H:%M:%S")
77                                 echo "$HORA;Topologia:$topologia-;$p;switch:;$lanswitch;controlador:;
78                                         $controlador;cantidad_nodos= ;$cantidad_nodos;
79                                         cantidad_Switch= ;$cantidad_Switch;cantidad_host= ;$cantidad_host;$simulacion_final"
80                                 echo "$HORA;Topologia:;$topologia-$p;switch:;$lanswitch;controlador:;$controlador;
81                                 cantidad_nodos= ;$cantidad_nodos;cantidad_Switch= ;$cantidad_Switch;
82                                 cantidad_host= ;$cantidad_host;$simulacion_final" >> /home/simulaciones/resultados/$topologia.sdn
83
84                                 c=`expr $c \+ 1`
85                 done

86                   if [ $saltos = "p2" ]
87                   then
88                           p=`expr $p \* 2`
89                   else
90                           p=`expr $p \+ $saltos`
91                   fi
92
93          done
94
95 else
96
97
98
99 topologia=single
100 lanswitch=ovsk
101 zdump EST
```

```
102	echo "-------------------------------------------------------------------------------------------------------"
103	echo "|TOPOLOGIA $topologia,OpenFlow10, switch=$lanswitch, Controlador=$controlador                         |"
104	echo "|TOPOLOGIA: sudo mn --topo $topologia,$chost --mac --switch $lanswitch --controller $controlador     |"
105	echo "|RESULTADO:TIEMPO DE INICIO/FIN DE LA TOPOLOGIA PARA CADA VALOR DE n                                  |"
106	echo "|RESULTADO:CPU,MEMORIA TOTAL,OCUPADA. Mem: total-used-free-shared-buffers-cached                     |"
107	echo "|(Modo Resumen para estadistica):                                                                     |"
108	echo "-------------------------------------------------------------------------------------------------------"
109	sudo mn -c > /dev/null 2>&1     #Inicializando MiniNet
110	topologia=single
111	lanswitch=ovsk
112	p=$inicio
113	while [ $p -lt `expr $chost \+ 1` ]
114	do
115		c=0
116		while [ $c -lt $simxtopo ]
117			do
118				simulacion=$(echo "h1 bash /home/simulaciones/bash_script/host_script/
119					host_script 10.0.0.2 | grep ^Resultado | awk '{print
120					\"Resultado: \" \$0}' \nexit" | \sudo mn --topo $topologia,
121					$p --switch $lanswitch --controller $controlador 2>&1 |
122					\grep "Resultado\|completed")
123				if [ $topologia = "single" ]
124				then
125					cantidad_nodos=`expr $p \+ 1`
126					cantidad_Switch=1
127					cantidad_host=`expr $p`
128				else
129					cantidad_nodos=`expr $p \* 2`
130					cantidad_Switch=`expr $p`
131					cantidad_host=`expr $p`
132				fi
133				simulacion_final=$(echo $simulacion | sed -e 's/\r//g')
134				HORA=$(date "+%Y-%m-%d %H:%M:%S")
135				echo "$HORA;Topologia:$topologia-;$p;switch:;$lanswitch;controlador:;$controlador;
136				cantidad_nodos= ;$cantidad_nodos;cantidad_Switch= ;$cantidad_Switch;
137				cantidad_host= ;$cantidad_host;$simulacion_final"
138				echo "$HORA;Topologia:;$topologia-$p;switch:;$lanswitch;controlador:;$controlador;
139				cantidad_nodos= ;$cantidad_nodos;cantidad_Switch= ;$cantidad_Switch;
140				cantidad_host= ;$cantidad_host;$simulacion_final" >>
141				/home/simulaciones/resultados/$topologia.sdn
142
143				c=`expr $c \+ 1`
144		done
```

```
145                 if [ $saltos = "p2" ]
146                 then
147                         p=`expr $p \* 2`
148                 else
149                         p=`expr $p \+ $saltos`
150                 fi
151
152         done
153
154
155 topologia=single
156 lanswitch=ovsbr
157 zdump EST
158         echo "-------------------------------------------------------------------------------------------------------"
159         echo "|TOPOLOGIA $topologia,OpenFlow10, switch=$lanswitch, Controlador=$controlador                      |"
160         echo "|TOPOLOGIA: sudo mn --topo $topologia,$chost --mac --switch $lanswitch --controller $controlador  |"
161         echo "|RESULTADO:TIEMPO DE INICIO/FIN DE LA TOPOLOGIA PARA CADA VALOR DE n                              |"
162         echo "|RESULTADO:CPU,MEMORIA TOTAL,OCUPADA. Mem: total-used-free-shared-buffers-cached                 |"
163         echo "|(Modo Resumen para estadistica):                                                                 |"
164         echo "-------------------------------------------------------------------------------------------------------"
165         sudo mn -c > /dev/null 2>&1     #Inicializando MiniNet
166         p=$inicio
167         while [ $p -lt `expr $chost \+ 1` ]
168         do
169                 c=0
170                 while [ $c -lt $simxtopo ]
171                         do
172                                 simulacion=$(echo "h1 bash /home/simulaciones/bash_script/host_script/
173                                         host_script 10.0.0.2 | grep ^Resultado | awk '{print
174                                         \"Resultado: \" \$0}' \nexit" | \sudo mn --topo $topologia,$p
175                                         --switch $lanswitch --controller $controlador 2>&1 | \grep "Resultado\|completed")
176                                 if [ $topologia = "single" ]
177                                 then
178                                         cantidad_nodos=`expr $p \+ 1`
179                                         cantidad_Switch=1
180                                         cantidad_host=`expr $p`
181                                 else
182                                         cantidad_nodos=`expr $p \* 2`
183                                         cantidad_Switch=`expr $p`
184                                         cantidad_host=`expr $p`
185                                 fi
186                                 simulacion_final=$(echo $simulacion | sed -e 's/\r//g')
187                                 HORA=$(date "+%Y-%m-%d %H:%M:%S")
```

ANEXO 2: INSTALACION DE CBENCH Y HCPROBE EN UBUNTU 14.04

Instalando las dependencias:

$ sudo apt-get install autoconf automake libtool libsnmp-dev libpcap-dev libconfig8-dev

Descargando el código fuente de openflow:

$ git clone git://gitosis.stanford.edu/openflow.git

Entrando en la versión 1.0.0

$ cd openflow; git checkout -b mybranch origin/release/1.0.0

Descargando el código fuente de Oflops:

$ git clone git://gitosis.stanford.edu/oflops.git

Compilando el código fuente:

$ cd oflops ; sh ./boot.sh ; ./configure --with-openflow-src-dir=; make

En este caso:

cd oflops ; sh ./boot.sh ; ./configure --with-openflow-src-dir=/home/ubuntu/openflow; make

$ cd oflops ; sudo make install

Para ejecutar la aplicación:

cbench -c localhost -p 6633 -m 10000 -l 10 -s 16 -M 1000 -t

HCPROBE:

$ sudo apt-get install cabal-install

$ cdhcprobe/ARCCN-hcprobe-817d200/src

ANEXO 3: PROCEDIMIENTOS PARA EJECUTAR CONTROLADORES

CONTROLADOR RYU (OpenFlow 1.0 a 1.3):

$ sudo mn --topo single,3 --mac --controller remote --switch ovsk

sudo ovs-vsctl set bridge s1 protocols=OpenFlow13

sudo wireshark &

$ cd /home/ubuntu/ryu && ./bin/ryu-manager --verbose ryu/app/simple_switch_13.py

$cd /home/ubuntu/ryu/bin &&sudo ./ryu-manager ryu/app/tutorial_l2_switch.py

mininet> h1 ping h3

CONTROLADOR POX (OpenFlow 1.0):

$ sudo mn --topo single,3 --mac --controller remote --switch ovsk

$ sudo ovs-vsctl set bridge s1 protocols=OpenFlow10

$ cd /home/ubuntu/pox && ./pox.py log.level --DEBUG samples.pretty_log forwarding.l2_learning

$ cd /home/ubuntu/pox && ./pox.py log.level --DEBUG forwarding.tutorial_l2_hub

$ cd /home/ubuntu/pox &&./pox.py log level --DEBUG samples.pretty_log forwarding.l2_learning

mininet> h1 ping h2

CONTROLADOR ONOS:

$ cd onos

$ source ./tools/dev/bash_profile

$ echo $KARAF_ROOT

$ mvn clean install -nsu -DskipIT -DskipTests

karaf clean

onos> feature:list -i

sudo mn --topo linear,2 --mac --switch ovsk,protocols=OpenFlow13 --controller remote --arp

mininet> h1 ping h2

onos> hosts

onos> feature:uninstall onos-app-fwd

CONTROLADOR NOX:

$:./nox_core –i ptcp:<IP del controlador>:<puerto donde escucha el controlador><directorio donde se encuentra la aplicación> -v

CONTROLADOR ODL:

.<directorio donde se encuentra la aplicación>$:./run.sh

CONTROLADOR TREMA:

$ ./trema run <aplicación>.rb

CONTROLADOR BEACON:

//Descargar el software.

Git: git://gitosis.stanford.edu/beacon.git

Antes de ejecutar la aplicación modificamos el fichero “beacon/beacon.properties” y especificamos en el parámetro “threadCount” la cantidad de núcleos (core) en el sistema.

Si se van a realizar pruebas de rendimiento o escalabilidad es recomendable cambiar el parámetro “controller.immediate=true” en este mismo fichero. En el fichero “beacon.ini” configurar el parámetro “XX:+AggressiveOpts -Xmx3000M”

$: sudo ./beacon –configuration <aplicación>

$: sudo ./beacon –configuration configurationSwitch

//Si tenemos 2 Core con 4 hilos se puede ejecutar en la tarea 0-3.

$: sudo taskset -c 0-3 /beacon -configuration configurationSwitch

CONTROLADOR FLOODLIGHT:

//Instalación

sudo apt-get install build-essential default-jdk ant python-dev eclipse

git clone git://github.com/floodlight/floodlight.git

cd floodlight

sed -i -e "s/floodlight.modules = .*$/floodlight.modules = net.floodlightcontroller.learningswitch.

learningSwitch,net.floodlightcontroller.counter.NullCounterStore,net.floodlightcontroller.perfmon.NullPktInProcessingTime/" src/main/resources/floodlightdefault.properties

sed -i -e "s/^net.*,.*$//" src/main/resources/floodlightdefault.properties

sudo ant

$ java -jar target/floodlight.jar -cf src/main/resources/floodlightdefault.properties

$:java –jar target/floodlight.jar (levantar los módulos por defecto)

//Interfaz web de usuario

ANEXO 4: CÓDIGO FUENTE DE LA APLICACIÓN SDN PARA DETECTAR INTRUSOS EN LA RED

```
# SWITCH QUE APRENDE DIRECCIONES MAC (L2) & DIRECCIONES IP (L3)
# APLICACIONES:
#   SWITCH L2/L3 CON SOPORTE
#   ASOCIACION DE SWITCH/MAC/PUERTO/IP
#   ALARMA EN CASO DE INTRUSO
#   TRAZAS DE FLUJO DE TRAFICO CON NIVELES DE DETALLES QUE PERMITE ENTENDER EL FUNCIONAMIENTO DE SDN
#
"""
Autores: ING. YANKO ANTONIO MARIN MURO
         DR. C. ING. FELIX ALVAREZ PALIZA

EJECUTARLO:./pox.py forwarding.uclv_L2_L3_SWITCH_ACL_TRAZA


"""

import ...

log = core.getLogger()
# Nuestra Tabla de SWITCH MAC/PUERTO
TablaArp = {};
# Nuestra Tabla de SWITCH IP/PUERTO
TablaIP = {};
# HABILITAR O DESHABILITAR
traza = False;

# DECLARANDO LOS USUARIOS AUTORIZADOS
opcion_seguridad_arp=False;
opcion_seguridad_ip=False;
opcion_seguridad_registro=False;
usuario = ({'nombre':'YANKO','ip':'10.0.0.1','switch':1, 'puerto':1,'mac':EthAddr("00:00:00:00:00:01")},
           {'nombre':'PALIZA','ip':'10.0.0.2','switch':2, 'puerto':1,'mac':EthAddr("00:00:00:00:00:02")});

# CALCULANDO LA MAXIMA CANTIDAD DE USUARIOS DECLARADOS
max_usuarios=len(usuario)

sw_puertos = ({'switch':1,'puerto':1,'tipo':'acceso'},
              {'switch':1,'puerto':2,'tipo':'tronco'},
              {'switch':2,'puerto':1,'tipo':'acceso'},
              {'switch':2,'puerto':2,'tipo':'tronco'},
              {'switch':2,'puerto':3,'tipo':'tronco'},
              {'switch':3,'puerto':1,'tipo':'acceso'},
              {'switch':3,'puerto':2,'tipo':'tronco'},
              {'switch':3,'puerto':3,'tipo':'tronco'},
              {'switch':4,'puerto':1,'tipo':'acceso'},
              {'switch':4,'puerto':2,'tipo':'tronco'},
              {'switch':4,'puerto':3,'tipo':'tronco'});

# CALCULANDO LA MAXIMA CANTIDAD DE PUERTOS
max_sw_puertos=len(sw_puertos)


def _handle_ConnectionUp (event):
  """
  Modo proactivo:Cuando el switch se conecte al controlador instalar un fluio
  y de forma automatica que envie las tramas por todos los puertos
  """
  msg = of.ofp_flow_mod()
  msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
  event.connection.send(msg)

def _handle_PacketIn (event):
  """
  Modo reactivo:Esperamos primero que llegue la trama al controlador y luego
  procesar ARP, IP, etc.
  """
  dpid = event.connection.dpid  # OBTENIENDO EL ID DEL SWITCH
  packet = event.parsed         # ASIGNANDO EL EVENTO A LA VARIABLE PACKET
  inport = event.port           #PUERTO DEL SWITCH POR DONDE ENTRA EL PAQUETE
```

```
    # DETECTANDO SI ES UN NUEVO SWITCH
    if dpid not in TablaArp:
        TablaArp[dpid] = {}       # NUEVO SWITCH -- CREAR TABLA VACIA MAC/PUERTO
        TablaIP [dpid] = {}       # NUEVO SWITCH -- CREAR TABLA VACIA IP/PUERTO
        if traza: print "[HORA-%s]-SWITCH-%i DETECTADO, CREANDO UNA NUEVA " \
                        "TABLA SWITCH/MAC/IP/PUERTO" % (time.asctime(time.localtime(time.time())), dpid)

    #DETECTANDO EL PROTOCOLO ARP 0x0806 (2054)
    if packet.type == 0x0806:
        if traza: print "[HORA-%s]-SWITCH-%i DETECTADO PROTOCOLO ARP " \
                        "POR EL PUERTO-%i" % (time.asctime(time.localtime(time.time())), dpid,inport)
        procesamiento_arp(event)

    #DETECTANDO EL PROTOCOLO IP 0x0800 (2048)
    if packet.type == 0x0800:
        if traza: print "[HORA-%s]-SWITCH-%i DETECTADO PROTOCOLO " \
                        "IP POR EL PUERTO-%i" % (time.asctime(time.localtime(time.time())), dpid,inport)
        procesamiento_ip(event)


def procesamiento_arp(event):
    # PROCESAMIENTO DEL PROTOCOLO ARP
    # COMPROBANDO SI ESTA HABILITADA LA SEGURIDAD
    if opcion_seguridad_arp:
      if seguridad_arp (event):
        #ASIGNANDO EL ID DEL SWITCH, SRCMAC, DSTMAC, SRCIP, DSTIP
        dpid = event.connection.dpid     #ID DEL SWITCH
        inport = event.port              #PUERTO DEL SWITCH POR DONDE ENTRA EL PAQUETE
        packet = event.parsed            #PAQUETE COMPLETO
        srcmac = packet.src              #MAC DE ORIGEN
        dstmac = packet.dst              #MAC DE DESTINO
        protocolo = packet.type          #NUMERO DEL PROTOCOLO
        srcip = packet.next.protosrc     #IP DE ORIGEN
        dstip = packet.next.protodst     #IP DE DESTINO

        # LLENANDO LA TABLA SWITCH/MAC/PUERTO
        TablaArp [dpid][srcmac] = inport
        # LLENANDO LA TABLA SWITCH/IP/PUERTO
        TablaIP [dpid][srcip] = inport

        # PREPARACION DE LA INSTALACION DE LA REGLA EN EL SWITCH
        packet_in = event.ofp                # MENSAJE ofp_packet_in QUE LLEGO AL CONTROLADOR
        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        msg.in_port = packet_in.in_port

        # DETECTANDO SI ES UN BROADCAST ARP
        if packet.dst.is_multicast:
            # APRENDO LA MAC. LLENAR LA TABLA DE SWITCH/MAC/PUERTO
            TablaArp [dpid][srcmac] = inport
            # APRENDO LA SWITCH/IP/PUERTO
            TablaIP [dpid][srcip] = inport
            # ENVIANDO  ORDEN AL SWITCH PARA REALIZAR BROADCAST
            action = of.ofp_action_output(port = of.OFPP_FLOOD)
            msg.actions.append(action)
            event.connection.send(msg)
            if traza: print "[HORA-%s][OF_Packet_IN]    [SRC_MAC=%s APRENDO]-->[DST_MAC=%s], " \
                            "[ARP, SRC_IP=%s, DSP_IP=%s]-->[Pto-%i]-[S-%i]--->[BROADCAST ARP]" \
                            "--->[Pto-TODOS-%i]" % (time.asctime(time.localtime(time.time())),
                                                    srcmac,dstmac,srcip,dstip,inport, dpid, inport )

        else:
        # SI SE ESPECIFICA MAC_DST Y LA DIRECCION MAC NO ESTA EN LA TABLA --->
        # APRENDER Y DIFUNDIR EL MENSAJE POR TODOS LOS PUERTOS
            if (packet.dst not in TablaArp[dpid]):
                # APRENDO LA MAC.LLENO LA TABLA DE MAC/PUERTO
                TablaArp [dpid][srcmac] = inport
                # APRENDO LA IP/PUERTO
                TablaIP [dpid][srcip] = inport
                # Add an action to send to the specified port
                action = of.ofp_action_output(port = of.OFPP_FLOOD)
                msg.actions.append(action)
```

```
                # ENVIANDO EL MENSAJE AL SWITCH
                event.connection.send(msg)
                if traza: print "[HORA-%s][OF_Packet_IN]    [SRC_MAC=%s APRENDO]-->[DST_MAC=%s NO ENCONTRADA]," \
                                " [ARP, SRC_IP=%s, DSP_IP=%s]-->[Pto-%i]-[S-%i]--->[BROADCAST ARP]" \
                                "--->[Pto-TODOS-%i]" % (time.asctime(time.localtime(time.time())),
                                                        srcmac,dstmac,srcip,dstip,inport, dpid, inport )
            else:
                # SI LA DIRECCION MAC ESTA EN LA TABLA --->ENVIAR EL MENSAJE AL PUERTO ESPECIFICO
                dstport = TablaArp[dpid][packet.dst]
                msg = of.ofp_flow_mod()
                msg.match = of.ofp_match.from_packet(packet, event.port)
                msg.idle_timeout = 10
                msg.hard_timeout = 30
                msg.actions.append(of.ofp_action_output(port = dstport))
                msg.data = event.ofp # 6a
                event.connection.send(msg)
                if traza: print "[HORA-%s][OF_Packet_IN]    [SRC_MAC=%s]-->[DST_MAC=%s ENCONTRADA], " \
                                "[ARP, SRC_IP=%s, DSP_IP=%s]-->[Pto-%i]-[S-%i]--->" \
                                "[ENVIAR ARP]--->[Pto-%i]" % (time.asctime(time.localtime(time.time())),
                                                              srcmac,dstmac,srcip,dstip,inport, dpid, dstport )

    return True

def procesamiento_ip(event):
    dpid = event.connection.dpid    #ID DEL SWITCH
    inport = event.port             #PUERTO DEL SWITCH POR DONDE ENTRA EL PAQUETE
    packet = event.parsed           #PAQUETE COMPLETO
    srcmac = packet.src             #MAC DE ORIGEN
    dstmac = packet.dst             #MAC DE DESTINO
    protocolo = packet.type         #NUMERO DEL PROTOCOLO
    srcip = packet.next.srcip       #IP DE ORIGEN
    dstip = packet.next.dstip       #IP DE DESTINO

    if (dstip in TablaIP[dpid]):
      # PREPARACION DE LA INSTALACION DE LA REGLA EN EL SWITCH
      packet_in = event.ofp               # MENSAJE ofp_packet_in QUE LLEGO AL CONTROLADOR

      msg = of.ofp_packet_out()
      msg.buffer_id = event.ofp.buffer_id
      msg.in_port = packet_in.in_port
      # SI LA DIRECCION IP ESTA EN LA TABLA --->ENVIAR EL MENSAJE AL PUERTO ESPECIFICO
      dstport = TablaIP[dpid][dstip]
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet, event.port)
      msg.idle_timeout = 10
      msg.hard_timeout = 30
      msg.actions.append(of.ofp_action_output(port = dstport))
      msg.data = event.ofp
      event.connection.send(msg)
    return True

def seguridad_arp(event):

      #ASIGNANDO EL ID DEL SWITCH, SRCMAC, DSTMAC, SRCIP, DSTIP
      dpid = event.connection.dpid    #ID DEL SWITCH
      inport = event.port             #PUERTO DEL SWITCH POR DONDE ENTRA EL PAQUETE
      packet = event.parsed           #PAQUETE COMPLETO
      srcmac = packet.src             #MAC DE ORIGEN
      dstmac = packet.dst             #MAC DE DESTINO
      protocolo = packet.type         #NUMERO DEL PROTOCOLO
      srcip = packet.next.protosrc    #IP DE ORIGEN
      dstip = packet.next.protodst    #IP DE DESTINO

      # COMPROBANDO EL SWITCH/PUERTO/MAC/IP DEL USUARIO
      count= 0
      if puerto_de_aceso(dpid, inport):
          while count < max_usuarios:
              if usuario[count]['ip']== srcip and usuario[count]['switch']== dpid \
                      and usuario[count]['puerto']== inport and usuario[count]['mac']== srcmac:
                  if traza: print "-->[%s]: seguridad_arp: Usuario autorizado: %s, en switch:%i,Puerto:%i: " \
                                  "IP=%s,MAC:%s. Tratando de acceder a:%s" % \
                                  (time.asctime(time.localtime(time.time())),
                                   usuario[count]['nombre'],dpid, inport,srcip,srcmac,dstip)
```

```
                return True
                break
            else:
                count = count + 1
        if count == max_usuarios:
            if traza: print "-->[%s]: seguridad_arp: Intruso detectado en la red en switch:%i,Puerto:%i: " \
                            "IP=%s,MAC:%s. Tratando de acceder a:%s" % \
                            (time.asctime(time.localtime(time.time())),dpid,
                             inport,srcip,srcmac,dstip)
            return False

    else:
        return True


def puerto_de_aceso(switch, puerto):
    # RETORNA VERDADERO SI UN PUERTO DE UN SWITCH ES DE ACCESO
    count= 0
    while count < max_sw_puertos:
        if sw_puertos[count]['switch']== switch and sw_puertos[count]['puerto']== puerto \
                and sw_puertos[count]['tipo'] == 'acceso':
            return True
        else:
            if count == max_sw_puertos:
                return False
            count=count + 1
```

```
def launch (reactivo = True):
  if reactivo:
    core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
    print "*****************************************************"
    print "UNIVERSIDAD CENTRAL MARTA ABREU DE LAS VILLAS (UCLV)"
    print "*****************************************************"
    print "APLICACION SDN: SWITCH L2/L3 , ACL, TRAZA"
    print "Autores: Ing.Yanko Antonio Marin Muro"
    print "         Dr.C. Ing. Felix Alvarez Paliza"
    print "*****************************************************"
    log.info("Swtich en modo Reactivo")
  else:
    core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
    log.info("Switch en modo proactivo.")
```

ANEXO 5: ESCENARIOS DE TRABAJO

CARACTERÍSTICAS	ESCENARIO A	ESCENARIO B	ESCENARIO C
Procesador	Intel ® Core ™ i3-3220 CPU @ 3.30 GHz.	Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz	Intel(R) Core(TM) i5-2540M CPU @ 2.60 GHz 2.60 GHz
Memoria instalada:	8 GB	4 GB	4 GB
Virtualización Activada	Si	Si	Si
Edición de Windows:	Windows 10 Pro, 64 BITS	Windows 7, 32 BITS	Windows 7, 32 BITS
Hipervisor	ORACLE Virtual Box Versión 5.0.14 r105127	ORACLE Virtual Box Versión 4.3.28	ORACLE Virtual Box Versión 5.0.14 r105127
SO Huésped	Ubuntu 14.04. Versión del Kernel LINUX: 3.13.0-27-generic.	Ubuntu 14.04. Versión del Kernel LINUX: 3.13.0-27-generic.	Ubuntu 14.04. Versión del Kernel LINUX: 3.13.0-27-generic.
Número de CPU asignados a la VM	2 CPU	2 CPU	5 CPU
Memoria asignada a VM (MB)	4 GB	1 GB	2 GB

ANEXO 6: CLASIFICACIÓN DE CONTROLADORES

NOMBRE	ARQUITECTURA	CONSISTENCIA	FALLO	CODIGO ABIERTO	COMERCIAL	ACTIVO	LENGUAJE	ODL (BASADO)	NBI	DOCUMENTACIÓN	PROTOCOLOS SOUTHBOUND	NB API	SOPORTA OPENSATCK
Avaya[108]	Centralizada	no			si	si		si	si		OpenFlow, OVSDB, OF-CONFIG, NETCONF	si	si
Beacon[76]	Centralizada Multi-hilos	no	no			no	Java			3	OpenFlow 1.0	ad-hoc Api	no
Big Switch[109], [110]					si	si			si		OpenFlow		
Brocade[111]	Distribuida	si	si	si	si	si	Java	si			OpenFlow, OVSDB, BGP, NETCONF		
Agile HUAWEI BASADO EN ONOS [112]	Distribuida Multi-hilos	si	si	si	si	si	Java	no	si	5	OpenFlow, OVSDB, BGP, NETCONF	si	si
Ciena [113]					si	si		si					
Cisco [114]	Distribuida				si	si	Java	si	si	5	OpenFlow, NETCONF, RESTCONF, MP-BGP EVPN	REST	
ConteXtream (HP)					si	si		si					
Coriant [115]					si	si		si					
Cyan (acquired by Ciena)[116]					si	si		ALIADO					
DISCO	Distribuida		si				Java				OF1.1	REST	
Ericsson[117]					si	si	Java	si			OpenFlow, OVSDB, BGP, NETCONF, PCEP, BGP-LS		
Extreme					si	si		si					
Fleet	Distribuida	no	no								OF1.0	ad-hoc	
Floodlight	Centralizada Multi-hilos	no	no	si	no	si	Java	no		5	OF1.0 OF1.3 OF1.4	RESTful API	no
HP					si	si		ALIADO					
HP VAN SDN	Distribuida	Débil	si				Java				OF1.0	RESTful API	
Huawei	Distribuida Multi-hilos				si	si		si	si	5			
HyperFlow	Distribuida	Débil	si				c++				OF1.0		
IBM					si	si		si					
Inocybe					si	si		si					

Juniper					si	si			si				
Kandoo	Jerárquicamente distribuido	no	no	si			C, C++, Python				OF1.0		
LOOM				si		si							
Maestro	Centralizada Multi-hilos	no	no				Java			3	OF1.0	ad-hoc API	no
Meridian	Centralizada Multi-hilos	no	no				Java				OF1.0	API extensible	
MobileFlow											OF1.2	SDMN API	
MuL	Centralizada Multi-hilos	no	no				C			3	OF1.0	Interfaz multi-nivel	si
NEC					si	si		ALIADO					
NodeFlow	Centralizada					no					OF1.0		
NOX	Centralizada	no	no		no	no	C++			3	OF1.0	ad-hoc API	no
NOX-MT	Centralizada Multi-hilos	no	no				C++			3	OF1.0	ad-hoc API	
Nuage Networks					si	si		ALIADO					
Onix	Distribuida	Débil, fuerte	si				Python, C				OF1.0, OF1.3	NVP NBAPI	
ONOS	Distribuida Multi-hilos	Débil, fuerte	si	si		si	Java	no	si	5	OF1.0, OF1.3 NETCONF	RESTful API	no
OpenContrail		no	no	si		si	Python, C++, Java				OF1.0	REST API	
OpenDaylight	Distribuida Multi-hilos	Débil	no	si		si	Java		REST API		OF1.0, 1.3, 1.4, NETCONF/ YANG,OVSDB, PCEP, BGP/LS, LISP, SNMP	REST, RESTCONF	si
OpenMUL				si		si							
PANE	Distribuida	si										PANE API	
Plexxi					si	si			si				
PLUMgrid					si	si			si				
Pluribus					si	si			si	3			
POX	Centralizada	no	no	si		si	Python + QT4				OF1.0	REST API	no
ProgrammableFlow	Centralizada						C				OF1.3		
Rosemary	Centralizada										OF1.0	ad-hoc	
Ryu NOS	Centralizada Multi-hilos	no	no	si		si	Python			3	OF.1 .[0,2,3,4]	ad-hoc API	si
SMaRtLight	Distribuida	no	no				Java				OF1.0	RESTful API	

SNAC	Centralizada	no	no				C++		OF1.0	ad-hoc API
Sonus					si	si		si		
Trema	Centralizada Multi-hilos	no	no	si		si	C, Ruby		OF1.0	ad-hoc API
VMware NSX					si	si		si		
yanc	Distribuida									file system

Tabla 2.6. Clasificación de controladores.

ANEXO 7: CÓDIGO FUENTE DE SDN vs. RED TRADICIONAL

```
"""
Autores: ING. YANKO ANTONIO MARIN MURO
         DR. C. ING. FELIX ALVAREZ PALIZA

#PARA EJECUTAR LA TOPOLOGIA
#sudo mn --custom /home/ubuntu/mininet/
           custom/uclv_legacy.py --topo tradicional
#sudo mn --custom /home/ubuntu/mininet/
           custom/uclv_legacy.py --topo sdn
"""

from mininet.topo import Topo
from mininet.nodelib import LinuxBridge
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import Controller, RemoteController, OVSController


class UclvRedTradicional( Topo ):
    "Topologia de red tradicional."

    def __init__( self ):
        "Creando una topologia personalizada."

        # Inicializando la topologia
        Topo.__init__( self )


        # Creando host y switch
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        s1 = self.addSwitch( 's1', cls=LinuxBridge )
        s2 = self.addSwitch( 's2', cls=LinuxBridge )
        s3 = self.addSwitch( 's3', cls=LinuxBridge )

        # Adicionado los enlaces
        self.addLink( h1, s1 )
        self.addLink( s1, s2 )
        self.addLink( h2, s2 )
        self.addLink( s2, s3 )
        self.addLink( s3, h3 )
```

```
50 class UclvRedSDN( Topo ):
51     "Topologia de red SDN."
52 
53     def __init__( self ):
54         "Creando una topologia personalizada."
55 
56         # Inicializando la topologia
57         Topo.__init__( self )
58 
59         # Creando host y switch
60         h1 = self.addHost( 'h1' )
61         h2 = self.addHost( 'h2' )
62         h3 = self.addHost( 'h3' )
63         s1 = self.addSwitch( 's1', cls=OVSKernelSwitch )
64         s2 = self.addSwitch( 's2', cls=OVSKernelSwitch )
65         s3 = self.addSwitch( 's3', cls=OVSKernelSwitch )
66 
67         # Creando los enlaces entre los nodos
68         self.addLink( h1, s1 )
69         self.addLink( s1, s2 )
70         self.addLink( h2, s2 )
71         self.addLink( s2, s3 )
72         self.addLink( s3, h3 )
73 topos = { 'tradicional': ( lambda: UclvRedTradicional() ) ,
74           'sdn': ( lambda: UclvRedSDN() )}
75 
```

NOTACIÓN Y ACRÓNIMOS

Dado el uso generalizado del inglés como lenguaje universal de la ciencia, algunos términos característicos de este ámbito no han sido traducidos, por cuanto su equivalente en español pueda llevar a confusión o simplemente no existir. En tales casos, en esta tesis, se ha utilizado la terminología original siempre con caracteres en cursiva.

Acrónimos y Siglas

ACL	Access Control List
A-CPI	Application Controller Plane Interface
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARP	Address Resolution Protocol
ATIS	Alliance for Telecommunications Industry Solutions
BGP	Border Gateway Protocol
CDPI	Control to Data-Path Interface
CPU	Central Processing Unit
D-CPI	Data-Controller Plane Interface
DDoS	Distributed Denial of Service
DSL	Domain-Specific Language
ETSI	European Telecommunications Standards Institute
ForCES	Forwarding and Control Element Separation
ICMP	Internet Control Message Protocol
ICT	Information and Communication Technologies
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IOS	Internetwork Operating System
IP	Internet Protocol
IRTF	Internet Research Task Force
ISP	Internet Service Providers
IT	Information Technology
ITU-T	International Telecommunication Union
IVS	Indigo Virtual Switch
LAN	Indigo Virtual Switch

LTE	Long Term Evolution
MAC	Medium Access Control
MPLS	Multiple Protocol Label Switching
MPLS-TP	MPLS Transport Profile
NAT	Network Address Translation
NBI	Northbound Interface
NE	Network Element
NETCONF	Network configuration protocol
NFV	Network Function Virtualization
NOS	Network Operating System
NVGRE	network virtualization using generic routing encapsulation
OF	OpenFlow
ONF	Open Networking Foundation
OVSDB	The Open vSwitch Database Management Protocol
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RCP	Routing Control Platform
REST	Representational State Transfer
RTT	Round Time trip
SAL	Service Abstraction Layer
SANE	Secure Architecture for the Networked Enterprise
SDMN	Software Defined Mobile Network
SDN	Software Defined Network
SDWLAN	Software Defined Wireless Local Area Network
SDX	SDN-Based Exchange
SNMP	Simple Network Management protocol
SPB	Shortest Path Bridging
STP	Spanning Tree Protocol
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TE	traffic enginering
TIC	Tecnlogía de la Información y la Comunicaciones
TRILL	Transparent Interconnection of Lots of Links

VLAN	Virtual Local Area Network
VMM	Virtual Machine Monitor
VPN	Virtual Private Network
vSDN	Virtual Software Defined Network
VXLAN	Virtual Extensible LAN
WAN	Wide Area Network
WLAN	Wireless Local Area Network
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol