FACULTAD DE INGENIERIA, UNIVERSIDAD DE LA REPUBLICA

Herramientas de simulación/emulación SDN

Tesis de licenciatura en computación
Perfil redes de computadoras

Mathias Duarte
Tutor: Eduardo Grampin
Co-Tutor: Martin Giachino

Julio 2016

Contenidos

Lista de Figuras

Lista de Tablas

1. Introducción y objetivos

SDN es una arquitectura muy nueva que aún se encuentra en fase de experimentación, por esto es de interés para este trabajo realizar un estudio sobre las distintas herramientas para la emulación y simulación de esta arquitectura. Para realizar un estudio completo sobre estas herramientas, como primer paso se realizará un estudio en detalle sobre SDN, así como también sobre sus componentes más importantes.
Como se mencionó en las líneas anteriores, este trabajo investigará herramientas de simulación/emulación de SND por lo que hay que definir que es un simulador, que es un emulador y cuales son sus diferencias.
Un simulador es un programa que reproduce por software el comportamiento del microcontrolador en la ejecución de un programa. Su empleo no exige ningún hardware y elimina muchos errores que agilizan el desarrollo posterior.
Mientras que un emulador es un software que permite ejecutar programas en una plataforma (sea una arquitectura de hardware o un sistema operativo) diferente de aquella para la cual fueron escritos originalmente. A diferencia de un simulador, que solo trata de reproducir el comportamiento del programa, un emulador trata de modelar de forma precisa el dispositivo de manera que este funcione como si estuviese siendo usado en el aparato original.

2. Software Defined Networks

2.1. ¿Que es SDN?

Las redes actuales cuentan cada vez más con configuraciones estáticas y complejas (lo que impide flexibilizarlas a los continuos cambios necesarios en los data center), esto hace que a la hora de crear una nueva red, se aborde su creación con una filosofía distinta a la que se usaba tradicionalmente. Los distintos proveedores y consumidores de servicios en la red, tienen cada vez más necesidades tecnológicas, las cuales son cada vez más difíciles de satisfacer por las redes tradicionales. Es decir, las redes tradicionales no son suficientemente ágiles como para reprogramarse, reconfigurarse y reajustarse a los despliegues de nuevos servicios. Las redes tradicionales generalmente presentan una arquitectura jerárquica, que están orientadas a tráfico norte-sur (cliente-servidor), lo cual, genera un importante problema, ya que las aplicaciones de hoy en día requieren de altas necesidades de tráfico este-oeste (tráfico entre servidores, no jerárquico), este tráfico supone la mayor carga de trabajo para la red. En el 2014, el 80 % de tráfico en el data center fue este-oeste. Por estos motivos, se necesita un cambio de arquitectura de red que se acerce a las necesidades actuales, que sea capaz de lograr flexibilidad, escalabilidad, automatización y control. Las redes definidas por software (SDN - software defined networking) son una nueva forma de abordar la creación de redes, en la cual el control se desprende del hardware y se le otorga a una aplicación de software, llamada controlador SDN. SDN presenta una arquitectura de red donde el plano del control se separa del plano de datos (logrando independencia en cada uno de los planos), lo cual permite obtener redes más programables, automatizables y flexibles. En definitiva, esta arquitectura le otorga al controlador SDN el control de la red. En la figura 1 se ilustran las arquitecturas mencionadas.

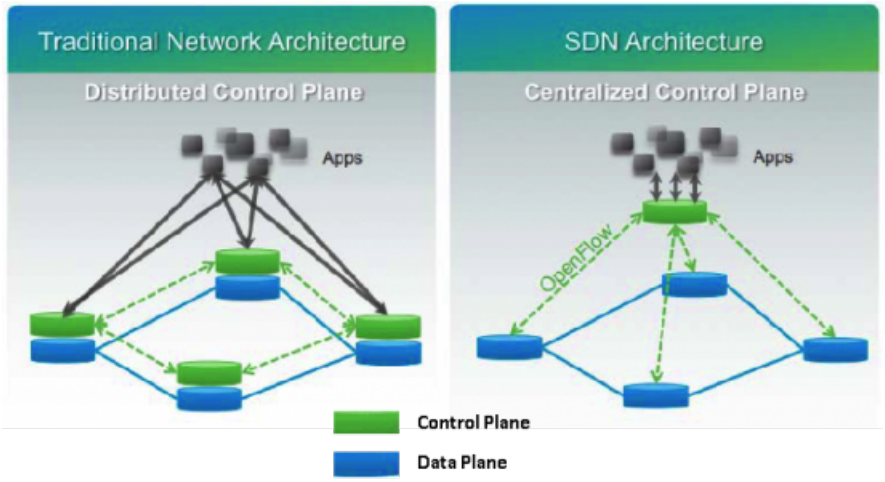

Figura 1: Comparación arquitecturas [1]

Para entender un poco más cómo funciona SDN, veamos que pasa cuando un paquete llega a un switch. En una red tradicional, las reglas de protocolo integradas al firmware propietario del switch le indican a dónde mover el paquete. El switch envía cada paquete al mismo destino por la misma trayectoria (y trata a todos los paquetes de la misma manera). En una SDN, un administrador de red puede manejar el tráfico de red de forma centralizada sin tener que configurar switches de manera individuales. El administrador puede cambiar cualquier regla de los switches cuando sea necesario, dando o quitando prioridad, o hasta bloqueando tipos específicos de paquetes con un nivel de control muy detallado. En definitiva con SDN se virtualiza la red independizándola de la infraestructura física subyacente y creando redes lógicas con objeto de cumplir los requisitos de rendimiento, escalabilidad y agilidad necesarios en modelos de cloud computing.[2]

2.2. Evolución Histórica SDN

SDN es fruto de una evolución que comenzó hace 20 años aproximadamente con la aparición de internet, debido a su éxito se creó la necesidad de gestionar y evolucionar las infraestructuras de redes, es decir; hacerlas programables. A partir de este momento, la historia se divide en tres etapas:

- Las redes activas (1995-2000 aprox.)
- La separación del plano de control del plano de datos (2001-2007 aprox.)
- La aparición de la interfaz de programación de aplicaciones de Openflow (2007-2010 aprox.)

En los inicios de Internet a principios de los 90, provocó un aumento del uso de las redes, haciendo que aplicaciones antiguas fueran sustituidas por otras más novedosas, llevando a los investigadores a diseñar y probar nuevos protocolos de red. Como las redes convencionales de aquel entonces no se podían programar surgieron las redes activas (Active Networking), conceptualizando una interfaz de programación con los diferentes recursos, para la gestión y administración de los distintos nodos que formarían la red individualmente. Esto añadía mucha complejidad, cosa que podría provocar que Internet no tuviese éxito, y por lo tanto, el programa de investigación de redes activas se dedicó a explorar alternativas a los servicios proporcionados por Internet vía IP o ATM. Ya en los 2000, el aumento del volumen de tráfico y la necesidad de redes más seguras y fiables, llevó a los investigadores a buscar mejores enfoques para la gestión de redes, como la ingeniería de tráfico, cuyos recursos y métodos usando protocolos de enrutamiento convencionales eran muy escasos. Los routers y switches convencionales, tenían una estrecha integración entre los planos de control y de datos, para enfrentarse a dicha tarea, la idea de separar ambos planos empezó a florecer con distintos enfoques. Posteriormente las empresas de equipos hardware comenzaron a implementar la lógica de reenvío de paquetes en hardware (plano de datos), separada del plano de control. Dentro del marco de las redes

definidas por software se tiene como precursor el proyecto Ethane [3] realizado en el año 2007. Este proyecto presenta una nueva arquitectura cuyo objetivo es que la red sea más administrable, donde la conectividad es gobernada a un nivel más alto y con políticas que permiten mayor granularidad del control de la red. Durante la próxima década diversos grupos de investigación y empresas empezaron a interesarse por la experimentación de redes a escala, en la universidad de Stanford, un grupo de investigadores creó el Clean Slate Program que dio lugar al protocolo Openflow. Gracias a la adopción del protocolo por las empresas, que abrieron sus APIs para permitir a los programadores controlar ciertos comportamientos de reenvío, la versión inicial de este protocolo se estableció en los switches a través de una simple actualización de firmware, sin necesidad de actualizar el hardware. En la figura 2 se observa la evolución histórica de SDN.

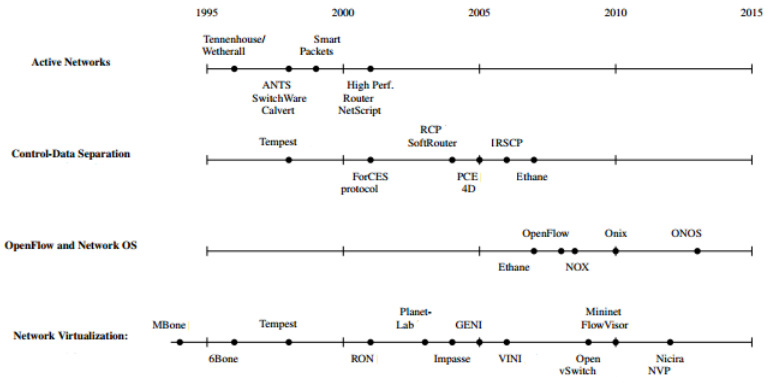

Figura 2: Evolución Histórica SDN [4]

2.3. Arquitectura

2.3.1. Capas de la arquitectura

Como se comentó anteriormente la arquitectura SDN, separa el plano de control, del plano de datos. Por esto existe una capa de infraestructura, la cual contiene a los dispositivos de red que son los encargados de realizar la conmutación y el encaminamiento de los paquetes. Además de la capa de infraestructura, se tiene la capa de control y la capa de aplicaciones, un esquema de la arquitectura se puede observar en la figura 3.

- Capa de infraestructura: En la capa de infraestructura se encuentran los dispositivos de red, son los encargados de la conmutación de los paquetes.
- Capa de control: En la capa de control, se encuentra el controlador SDN, el cual, es una entidad de software que tiene control exclusivo sobre el conjunto abstracto de recursos de plano de control, es decir, es la entidad que controla y configura los dispositivos de red para dirigir correctamente

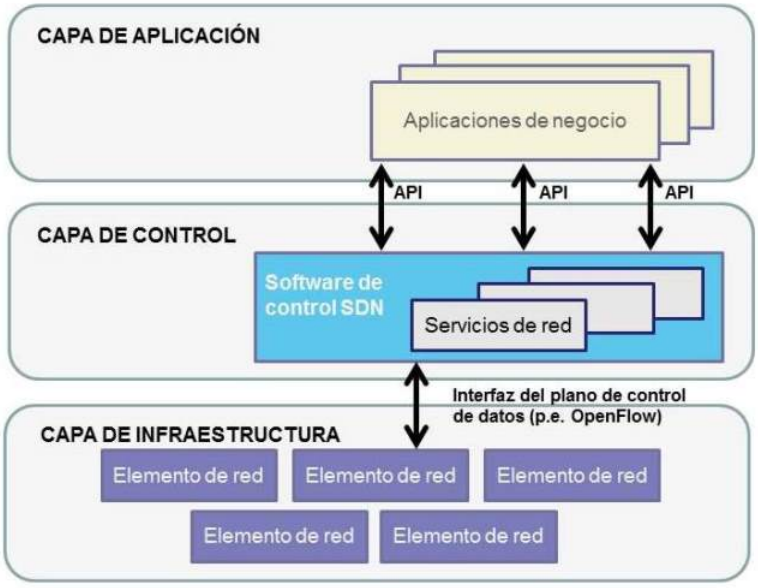

Figura 3: Arquitectura SDN [5]

los flujos de tráfico. El controlador SDN elimina la inteligencia de conmutación y encaminamiento de datos de los dispositivos, la cual en las redes tradicionales estaba en dichos dispositivos, de esta forma el controlador SDN es quien toma esas decisiones y selecciona el mejor camino para el tráfico. La capa de control se comunica con la capa de infraestructura a través de una API, llamada southbound. Mediante esta interfaz, el controlador SDN, logra interactuar con los dispositivos de red. Logrando así, descubrir la topología de red y realizar cambios cambios en los flujos de tráfico para satisfacer las demandas que surgen en tiempo real. Una especificación de la interfaz southbound, es el protocolo OpenFlow. Pero existen otras como Cisco OpFlex, Extensible Messaging and Presence Protocol (XMPP), Network Configuration Protocol (Netcong), OpenStack de Rackspace y la NASA.

- Capa de aplicación: A través de la capa de aplicaciones, se puede programar la capa de control, esto se realiza utilizando una API northbound.Por lo tanto, la capa de control y la capa de aplicación se comunican a través de la API northbound. Esta característica es muy importante para SDN, ya que al añadir esta tercera capa al diseño, es posible programar la red de la misma manera que las aplicaciones software, de modo que se pueden conseguir implementaciones que se integren ágilmente con el resto de componentes IT, por ejemplo, haciendo que la red se adapte automáticamente a los movimientos de máquinas virtuales entre Data Centers, permitiendo

grandes mejoras frente a los diseños tradicionales.

2.3.2. OpenFlow

Se puede definir OpenFlow como un protocolo, un mecanismo que permite a un servidor (controlador SDN) gestionar e interactuar con los dispositivos de red. Dicho de otra forma, es un protocolo que comunica los dispositivos OpenFlow con el controlador SDN. OpenFlow es open source y comenzó a desarrollarse en 2007, surgiendo como resultado de la colaboración de los sectores académico y empresarial. Fueron las universidades de Stanford y California en Berkeley quienes llevaron las riendas en primera instancia. En la actualidad, la Open Networking Foundation (ONF) se encarga de la definición del estándar, esta fundación, es un consorcio industrial que está a cargo de apoyar activamente a los avances de la SDN y la estandarización de OpenFlow. Este consorcio es una asociación de empresas como Google, Microsoft, Facebook y Yahoo con Verizon y Deutsche Telekom. Ha sido muy implementado por los fabricantes de dispositivos de red, lo que lo convierte en un referente de interfaz southbound. Por este motivo, nos tomamos un punto aparte para su estudio. La versión actual del protocolo OpenFlow es la 1.5.1 (Marzo 2015).

Switch OpenFlow

Para que el controlador pueda comunicarse con los dispositivos de la capa de infraestructura, estos deben admitir API Southbound (OpenFlow), llamaremos a estos dispositivos switch OpenFlow. Un switch OpenFlow se compone de un conjunto de tablas de flujo (las cuales indican cómo procesar la información que llega al dispositivo), un canal seguro, el cual es el encargado de comunicar el dispositivo con el controlador SDN, mediante el protocolo OpenFlow. El canal OpenFlow es usualmente encriptado utilizando TLS (Transport Layer Security), pero este canal puede correr directamente en TCP (Transmission Control Protocol). También se compone de una group table, la cual contiene acciones que afectan a un grupo de flujos y una meter table, la cual consiste en una serie de entradas que definen un conjunto de métricas por flujo que facilitan el control de la tasa de paquetes asignadas a ellas. Permiten implementar operaciones simples de calidad de servicio, como pueden ser limitaciones en el tráfico, o pueden ser combinadas con colas para implementar configuraciones de calidad de servicio más complejas como DiffServ. En la figura 4 se puede observar la organización de estos componentes en el switch OpenFlow.

Existen dos tipos de switches OpenFlow:

- **OpenFlow-only**: Estos swtiches solo soportan el procesamiento de paquetes a través del pipeline OpenFlow.
- **OpenFlow-hybrid**: Estos switches soportan tanto la operación OpenFlow, así como también la de un switch Ethernet "normal".

OpenFlow utiliza las tablas de flujos que contienen los routers y switches Ethernet modernos, estas tablas son generalmente distintas en dispositivos de

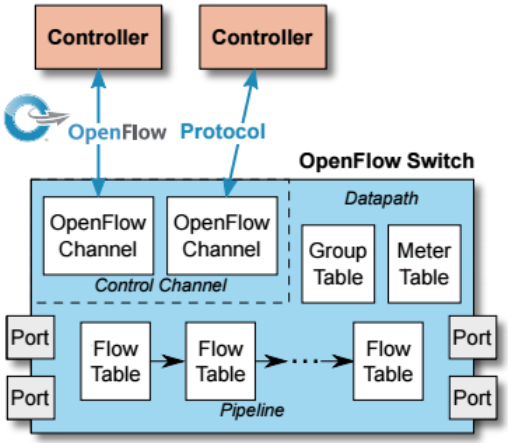

Figura 4: Componentes OpenFlow Switch [6]

distintos fabricantes, pero contienen un conjunto de funcionalidades en común. Estas tablas de flujo están generalmente construidas con TCAMs (Ternary content-addressable memory). Las tablas de flujos contienen una serie de entradas, con una acción asociada a cada una de estas entradas, de esta forma el switch sabe cómo manejar los distintos flujos de entrada.

Los campos de una entrada en la tabla de flujos, son los siguientes:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Figura 5: Campos entrada tabla de flujos [6]

- **Match Fields**: Definen un cierto flujo mediante el establecimiento de un conjunto de campos que se comparan con los paquetes recibidos en el switch.
- **Counters**: Muestran estadísticas sobre el flujo correspondiente a una cierta entrada, como por ejemplo el número de paquetes o bytes identificados.
- **Instructions**: Son acciones a realizar cuando se encuentra una coincidencia entre un paquete y una tabla de flujos (por ejemplo: modificar el conjunto de acciones o el procesamiento del pipeline).
- **Priority**: Orden de preferencia de "matching" del flujo de entrada.
- **Cookie**: Es una información añadida por el controlador que permite la identificación de la entrada de flujo.
- **Timeouts**: Permiten la eliminación de una cierta entrada de forma automática transcurrido un cierto periodo de tiempo.

- **Flags**: Estas flags alteran la forma en que se manejan los flujos de entrada.

Las entradas de cada tabla de flujos son identificadas por Match Fields y Priority, estos campos tomados en conjunto representan un único flujo de entrada, en una tabla de flujo específica.

Como se observa en la figura 6, el flujo de entrada pasa por varias tablas de flujos.

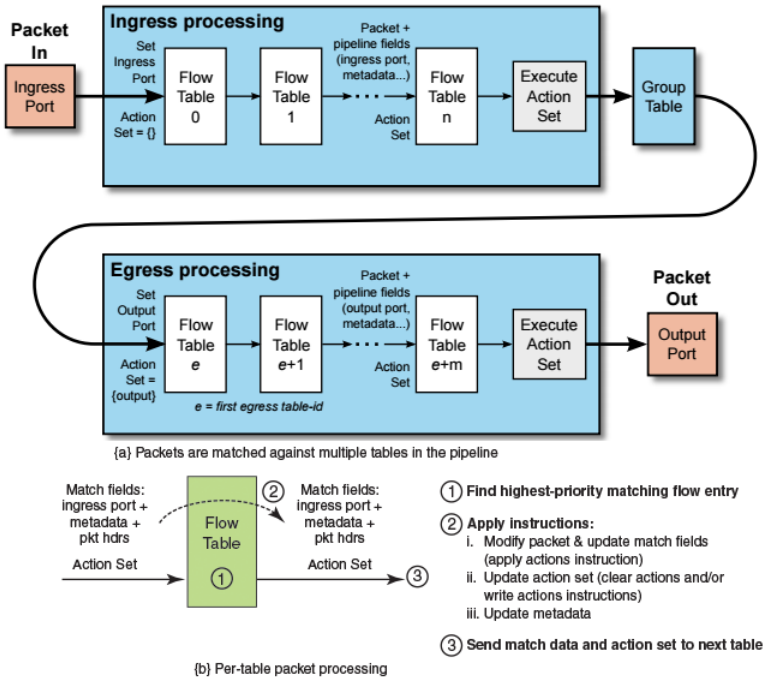

Figura 6: Tratamiento de flujos entrantes [6]

En el pipeline de openflow se realiza el proceso de decisión, donde, cada switch tiene una o varias tablas, en las cuales se realiza el siguiente proceso:
1. En primer lugar se busca el paquete entrante coincidente con la mayor prioridad.
2. Una vez detectado se aplican las distintas instrucciones.
a. Modificación del paquete y actualización de los campos coincidentes.
b. Actualización del conjunto de acciones.
c. Actualización del metadata.
3. Por último, se envía el dato coincidente a la siguiente tabla junto con el conjunto de acciones.

En caso de no hallar coincidencias en las tablas de flujo, el flujo de entrada será devuelto al Controlador SDN, que indicará al switch qué hacer con él. Como se mencionó anteriormente, cada entrada en la tabla de flujos tiene una

acción asociada. En OpenFlow hay tres tipos de acciones básicas que todos los dispositivos deben soportar:

- Reenvío de los paquetes de un flujo particular a un determinado puerto (o conjunto de puertos). Esto permite que los paquetes sean enrutados a través de la red.
- Encapsular y reenviar los paquetes de un flujo a un controlador. El paquete se entrega al canal seguro, donde se encapsula y se envía a un controlador. Normalmente se utiliza para el primer paquete en un flujo nuevo, para que el controlador pueda decidir si el flujo debe ser añadido a la tabla de flujos. Alternativamente, se podría utilizar como sniffer para transmitir todos los paquetes a un controlador para su procesado.
- Eliminar paquetes de un flujo. De esta forma, el controlador SDN podría actuar como un firewall, y bloquear paquetes sospechosos.

Protocolo OpenFlow

El switch identifica la conexión con el controlador a través de una Connection URI, esta debe tener una de las siguientes formas (RFC 3986):

protocol:name-or-address:port
protocol:name-or-address

Donde protocol, debe ser tls o tcp. name-or-address representa el nombre de host o la dirección IP del controlador, si la dirección IP es IPV6, esta deberá estar contenida en paréntesis rectos, tal como se recomienda en RFC 2732. Mientras que el port, es el puerto de capa de transporte utilizado. Si el puerto no es especificado, por defecto se usará el 6653 (A partir de la versión 1.4 (octubre 2013)).

Cuando una conexión es establecida, cada lado (Controlador SD, switch OpenFlow) envía un mensaje OFPT_HELLO con la versión más alta del protocolo OpenFlow soportado. Una vez recibido este mensaje, el receptor calcula la versión del protocolo a usar. Esta es, la más pequeña entre las soportadas por cada uno de los lados . Si la versión negociada es soportada, la conexión será iniciada, de lo contrario se enviará un mensaje HELLO_FAILED y se terminará la conexión.

Cada uno de los mensajes OpenFlow comienzan con la misma estructura de cabecera, esto permite su utilización independientemente de la versión del protocolo que se esté utilizando. En la figura 7 se observa como se distribuye el cabezal de un mensaje OpenFlow.

- **Version**: Indica la versión del protocolo que está siendo utilizada, el bit más significativo está reservado y tiene seteado el valor 0, mientras que los 7 menos significativos indican el número de versión del protocolo.

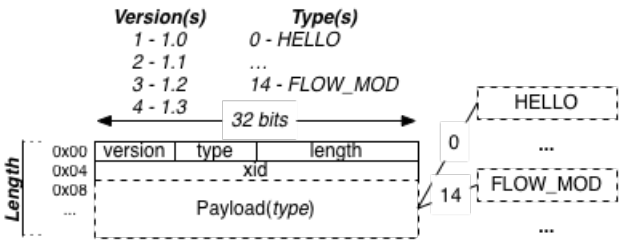

Figura 7: Cabezal OpenFlow Message [7]

- **Length**: Indica donde termina el mensaje OpenFlow proporcionando su longitud. El valor mínimo que puede tomar es 8, ya que 8 bytes es el tamaño fijo de header que contienen todos los paquetes OpenFlow.
- **Type**: Informa el tipo de mensaje, su contenido, se encuentra en el cuerpo del paquete.
- **Xid**:Proporciona un identificador de la transacción que se está realizando. Es un valor único que permite enlazar solicitudes con respuestas.

El protocolo OpenFlow soporta tres tipos de mensajes, de Controller-to-Switch (controlador-a-switch), Asynchronous (asíncrono) y Symmetric (simétrico), cada uno con múltiples subtipos. Los mensajes Controlador-a-switch son iniciados por el controlador y son utilizados para administrar o revisar el estado del switch. Los mensajes asíncronos son iniciados por el switch y utilizados para actualizar el controlador sobre eventos de red y cambios en el estado del switch. Los mensajes simétricos pueden ser iniciados por el switch o por el controlador, sin la necesidad de que exista una solicitud.

- **Controller-to-Switch**: Estos mensajes son iniciados por el controlador, y pueden requerir o no, respuesta por parte del switch. Algunos subtipos de este tipo de mensaje son:
  - **Features**: El Controlador le pide información al switch sobre sus características a través de este tipo de mensajes, lo hace a través de un features request; el switch responde con la información. Este tipo de mensajes se envía comúnmente luego de establecida la conexión a través del canal OpenFlow.
  - **Configuration**: El Controlador puede setear o pedir características de la configuración del switch, el switch solo responde, en caso de que se le solicite alguna característica.
  - **Modify-State**: Su principal propósito es añadir y eliminar entradas en las tablas de flujo y fijar características de los puertos.
  - **Read-State**: Interroga al switch sobre estadísticas del tráfico (contadores de las tablas de flujo).

- **Packet-out**: Envía paquetes por un puerto específico en el switch.
- **Barrier**: Usado para recibir notificaciones para operaciones completadas.
- **Role-Request**: Estos mensajes son utilizados cuando un switch se conecta con múltiples controladores, mediante estos mensajes el controlador setea el rol de su canal OpenFlow y el id del controlador. También pueden servir para consultar estos datos.
- **Asynchronous-Configuration**: Estos mensajes sirven para setear qué tipo de mensajes asincrónicos se desean recibir por el canal OpenFlow, y para consultar la configuración. Estos mensajes son útiles cuando el switch se conecta con varios controladores y comúnmente son enviados luego de que se establece la conexión.

- **Asynchronous**: Los switches envían mensajes al controlador a la llegada de un paquete, tras un error o tras un cambio de estado. Existen seis subtipos:
  - **Packet-in**: Enviado al controlador al recibir cualquier paquete que no tenga coincidencias en las tablas de flujo, o si la acción que corresponde a la entrada coincidente es reenviar el paquete al controlador.
  - **Flow-Removed**: Estos mensajes informan al controlador que fue eliminada una entrada de flujo de una tabla de flujos.
  - **Port-status**: Informan al controlador de un cambio en el estado de un puerto. Read-State: Interroga al switch sobre estadísticas del tráfico (contadores de las tablas de flujo).
  - **Role-status**: Informan al controlador de un cambio en su rol.
  - **Controller-Status**: Informan al controlador cuando el estado de un canal OpenFlow cambia.
  - **Flow-monitor**: Informan al controlador de un cambio en una tabla de flujos.

- **Symmetric**: Son enviados sin solicitud en cualquier dirección. Se definen cuatro tipos:
  - **Hello**: Estos mensajes son intercambiados entre el switch y el controlador cuando se está estableciendo una conexión.
  - **Echo**: Mensaje Request/Reply, qué puede ser enviado indistintamente por el switch o el controlador, a fin de comprobar la comunicación entre ellos. Pueden ser utilizados para medir la latencia o el ancho de banda de una conexión controlador-switch.
  - **Error**: Son mensajes enviados por el switch o el controlador para informar de algún problema a la otra parte de la conexión.
  - **Experimenter**: Proporciona una manera a los switches OpenFlow de ofrecer características adicionales. Está pensado para futuras revisiones de OpenFlow.

Versiones del Protocolo

Desde su primera especificación, el protocolo OpenFlow tuvo muchas mejoras, en este punto presentamos las más importantes que se dieron desde su primer versión. La ONF se hizo cargo de su especificación a partir de la versión 1.2.[6]

- **Versión 0.2.0 (Marzo 2008)**:
  Primera versión del protocolo.

- **Versión 0.8.0 (Mayo 2008)**:
  En esta versión se resalta el agregado de mensajes de error, se añade la prioridad a los flujos, se agrega el puerto virtual OFPP_TABLE (para el envío de paquetes packet-out a las tablas), se agrega una tabla de estadisticas asi como también mensajes con estadísticas de puertos.

- **Versión 0.8.2 (Octubre 2008)**:
  Se agregan los mensajes Echo Request y Echo Reply.

- **Versión 0.8.9 (Diciembre 2008)**:
  A partir de esta versión es posible que las entradas de flujos contengan máscaras IP. Se agrega más información a los mensajes que contienen estadísticas de puertos (ofp_port_stats). Se modifica la acción que envía un paquete por todos los puertos, para que no lo envíe por el puerto de entrada. Se agregan los mensajes del tipo OFPT_VENDOR, para que los vendedores agreguen sus propias extensiones, sin dejar de ser compatible con OpenFlow. Se pueden configurar switches que soporten STP. Se modifica el handshake para soportar la compatibilidad hacia atrás. Se incrementa a 65280 el número de puertos configurados.

- **Versión 0.9 (Julio 2008)**:
  En esta versión se incluye un mecanismo de conmutación por error simple, un switch se puede configurar con una lista de controladores, si la conexión con el primer controlador asignado falla, pasará a intentar conectarse con el segundo y así sucesivamente. Se agrega un flujo de emergencia en el cache para el caso de que se pierda la conexión con el controlador.
  La numeración de los puertos comienza en 1 y no en 0.
  A partir de esta versión se recomienda usar el puerto 6633 para OpenFlow.

- **Versión 1.0 (Diciembre 2009)**:
  A partir de esta versión se matchean campos IP en paquetes ARP. También se comienzan a soportar múltiples colas en los puertos de salida. Se incluye el campo cookie en los flujos.

- **Versión 1.1 (Febrero 2011)**:
  Se destaca la utilización de múltiples tablas de flujos en un mismo switch y la tabla de grupos, que proporcionan una mayor flexibilidad de acciones a ejecutar dado un paquete entrante en el switch. Al tener varias tablas de flujos es posible realizar el proceso de correspondencia por las entradas de múltiples tablas de flujos conectadas, lo que se conoce como pipeline.

Cuando no coincide ninguna entrada de una tabla de flujos con un cierto paquete se puede continuar examinando las tablas de flujos siguientes, ya que se encuentran en un cierto orden. Si por el contrario, en algún momento existe una coincidencia con una entrada, se dispone de una acción adicional que permite continuar el proceso de correspondencia en una tabla específica y así formar un conjunto de acciones a realizar, conocido como action set. Esto añade el concepto de metadata, que consiste en un registro de información utilizado durante el pipeline. La tabla de grupos, conocida como group table, está compuesta por entradas que identifican las acciones a realizar por un determinado grupo. Este concepto de grupo permite indicar de forma eficiente que un mismo conjunto de acciones deben ser procesadas por múltiples flujos, siendo útil para implementar multicast.

- **Versión 1.2 (Diciembre 2011)**:
  Se añade soporte para IPv6 y se crea una expresión flexible para el packet matching que permite la comparación con nuevos campos en los actuales y futuros protocolos. Esto reemplaza a la estructura fija que se estaba utilizando por una nueva estructura denominada OpenFlow Extensible Match (OXM). Con este nuevo avance se puede llegar a realizar el proceso de correspondencia con campos que se encuentren dentro del cuerpo del paquete, creando entradas de flujos sin límites. Además, se especifica la conexión de un switch con varios controladores. El controlador podrá asumir el rol de maestro o de esclavo, y dependiendo de ello tendrá mayor o menor privilegio en la toma de acciones sobre el switch.

- **Versión 1.3 (Junio 2012)**:
  Destaca la aparición de una nueva tabla en el plano de datos del switch, la tabla de medidas, meter table. Consiste en una serie de entradas que definen un conjunto de métricas por flujo que facilitan el control de la tasa de paquetes asignadas a ellas. Permiten implementar operaciones simples de calidad de servicio, como pueden ser limitaciones en el tráfico, o pueden ser combinadas con colas para implementar configuraciones de calidad de servicio más complejas como DiffServ.

- **Versión 1.4 (Octubre 2013)**:
  Aparece un nuevo mecanismo para realizar modificaciones en los switches garantizando atomicidad en ellas, bundles mechanism. Consiste en agrupar un conjunto de mensajes OpenFlow en uno solo y garantizar el éxito de todos ellos. De esta forma se realizarán todas las modificaciones juntas, o si falla alguna, ninguna modificación se realizará. Asimismo, proporciona una mejor sincronización en los cambios producidos. Se agrega la posibilidad de sincronizar tablas de flujo, tanto de manera unidireccional como bidireccional. Cuando una entrada de flujo es añadida, modificada o eliminada en una tabla origen, su correspondiente entrada sincronizada, en otra tabla de flujos, debe ser añadida, modificada o eliminada automáticamente. Este método permite tener varios flujos con la misma información

en diferentes puntos del pipeline. Además, a partir de este momento el puerto TCP utilizado por OpenFlow es el 6653, aprobado por la IANA.

- **Versión 1.5 (Diciembre 2014)**:
  Se extiende el pipeline hacia un conjunto de tablas de flujos que se basan en el contexto del puerto de salida del switch. Hasta ahora las entradas de flujo de las tablas dependían fundamentalmente de los campos que provienen en el paquete y del puerto origen en el switch. Esto añadía una limitación si se quería realizar el proceso de correspondencia teniendo en cuenta el puerto de salida de un cierto flujo. Así, ahora cuando un paquete sea redirigido hacia un puerto de salida, comenzará a procesar la primera tabla de flujos de salida y continuará por un pipeline similar al utilizado en versiones anteriores.

2.3.3. Controladores SDN

En el centro de la arquitectura de las SDN se encuentra la capa de control con el controlador SDN, que es quien gestiona los flujos de entrada. El controlador es el cerebro de la red, este controla todas las comunicaciones entre las aplicaciones y los dispositivos. El controlador SDN se encarga de traducir las necesidades o requisitos de la capa Aplicación a los elementos de red, y de proporcionar información relevante a las aplicaciones SDN, pudiendo incluir estadísticas y eventos. Existen múltiples controladores SDN OpenSource, como por ejemplo: NOX, POX, Ryu, Beacon, Floodlight, FlowER, Maestro, NodeFlow, OpenContrail, ONOS, OpenDaylight, Trema. Algunos de estos han sido desarrollados por la comunidad, y otros que han sido desarrollados por empresas que han liberado el código. También existen varios controladores comerciales, como por ejemplo: Big Switch, Cisco XNC, HP VAN SDN Controller Software, y los de las empresas NEC y Juniper. Como mencionamos anteriormente OpenFlow no es la única alternativa para una southbound API, pero si es la más fácil de explotar y sobre la que más información hay disponible en este momento para abordar las redes definidas por software, por lo que elegir un controlador con soporte OpenFlow ofrece muchas ventajas. El rendimiento es una de las características importantes a evaluar en un controlador, la cual se evalúa mediante el establecimiento de flujos. La medida que se utiliza en este caso es el tiempo de configuración de un flujo, y el número de flujos por segundo que el controlador puede configurar. Se debe garantizar que el rendimiento del controlador no produzca un cuello de botella en la red implantada. La configuración de flujos puede ser de dos formas: Proactiva o Reactiva (esto se define cuando se diseña la sdn).

Un controlador SDN debe manejar un gran número de switches, por lo que la escalabilidad a la hora de seleccionar un controlador es muy importante. Depende de las necesidades actuales y futuras de la red, pero un solo controlador SDN debería ser capaz de administrar al menos 100 switches sin problemas de rendimiento. Algunas de las otras características que se deben tener en cuenta a la hora de evaluar un controlador son: Soporte a la virtualización de la red, soporte

a ejecución en multiplataforma, seguridad en la red (soportar autenticación y autorización), entre otros. Otra diferenciación que tienen los controladores SDN, es el lenguaje en el que fueron programados, algunos de los lenguajes utilizados son: C/C++, Java, Python o Ruby entre otros.
Se describen algunos de los controladores open source que se mencionaron:

- **NOX**: Es una plataforma de controlador que permite escribir el controlador en Python, en C++ o en alguna combinación de estos lenguajes. NOX es una plataforma open source con la cual es muy facial crear programas que controlen o monitoren la red. Los programas escritos en NOX tienen control de nivel de flujo de la red, esto significa que ellos pueden determinar cuales flujos son permitidos en la red y qué camino deben tomar[8].

- **Beacon**: Es una plataforma controladora de Java creada en 2010, basada en OpenFlow que soporta tanto operaciones basadas en eventos como por thread. Beacon es fácil de desarrollar utilizando el entorno Eclipse, la máquina donde se pretenda realizar este controlador requerirá Java 6 JDK and JRE.[9]

- **Floodlight**: Es un controlador, basado en Java, de tipo empresarial que cuenta con licencia Apache. Es soportado por una comunidad de desarrolladores de todo tipo de backgrounds, incluidos ingenieros de Big Switch Networks[10].

- **Trema**: Es un controlador de OpenFlow basado en Ruby que también proporciona entornos de prueba. Se presenta como una opción completa y fácil de usar para ingenieros y diseñadores no solamente para Ruby sino con también soporte para C. Esta bajo licencia GNU General Public License version 2.0 o MIT License[11].

- **POX**:Es una plataforma controladora de OpenFlow basada en Python. Puede considerarse como .$^{\text{el}}$ hermano menor"de NOX. En su núcleo se considera una plataforma de rápido desarrollo y creación de prototipos de software de control de red que utiliza el lenguaje Python; actualmente es uno de los frameworks más usados (incluyendo NOX, Floodlight, Trema, entre otros) cuyo objetivo principal es configurar controladores OpenFlow. Características principales:

  - Interfaz OpenFlow basada en Python.
  - Componentes de muestra reutilizables para selección de camino, topología, descubrimiento, etc.
  - Tiene como objetivo poder ser compatible tanto con Linux y MAC como con Windows.
  - Soporta la misma GUI y las mismas herramientas de visualización que NOX.
  - Tiene un buen rendimiento si se compara con las aplicaciones de NOX escritas en Python.

[12]

- **NodeFlow**: Es el controlador SDN escrito en JavaScript para Node.JS creado inicialmente por Cisco. Este controlador proveía una librería asíncrona sobre JavaScript para programar las redes SDN. La idea de usar JavaScritp era descargar de CPU al controlador. Este controlador ha quedado descontinuado por la decisión de Cisco de abordar las redes SDN desde un punto de vista no OpenFlow[13].

- **FlowER**: Es un controlador SDN OpenFlow escrito en lenguaje de programación Erlang. Es un controlador menos conocido que nació en 2012 basado en un lenguaje de programación concurrente y un sistema de ejecución que incluye una máquina virtual y biblioteca[14].

- **Maestro**: Se define como un sistema operativo de orquestación de aplicaciones de control de red. En realidad, es una plataforma de controlador SDN OpenFlow. Ha sido diseñado en Java aunque requiere ANT. Es multi-thread, es decir explota la capacidad de CPU de los procesadores multi-core y se distribuye bajo licencia GNU[15].

- **Ryu**: Está escrito en Python. Dispone de un conjunto de APIs bien definidas, que permite el desarrollo de nuevas aplicaciones de gestión de red, y de varios protocolos para la gestión de los dispositivos de red, como OpenFlow y Netconf. Está escrito en Python soporta completamente desde la versión 1.0 hasta la 1.5 de OpenFlow[16].

- **OpenDayLight**: Ha surgido como un proyecto de la fundación Linux en el que contribuyen importantes empresas. Es un controlador multipropósito escrito en Java y originado a partir de Beacon. Además, es ampliamente reconocido y dispone de un gran número de módulos que pueden ser utilizados según las necesidades de la organización. Destaca sobre otros controladores al disponer de numerosas interfaces southbound, incluyendo OpenFlow, y de una capa de abstracción situada encima de dichas interfaces[17].

- **RouteFlow**: RouteFlow es un proyecto Open Source para proporcionar servicios de enrutamiento IP virtualizados, sobre hardaware con OpenFlow habilitado. RouteFlow esta compuesto por una aplicacion de controlador OpenFlow, por un servidor RouteFlow y por un entorno de red virtual, que reproduce la insfraestructura fisica y ejecuta los motores de enrutamineto IP(e.j. Quagga)[18].

Existen herramientas para evaluar la distintas características de los controladores, dos de ellas son[19]:

- **Cbench**: Cbench (Controller Benchmarker, por sus siglas en inglés) es una herramienta para monitorear controladores OpenFlow a través de la generación de eventos. Cbench emula un número configurable de switches

OpenFlow los cuales se comunican con un controlador OpenFlow para medir diferentes aspectos del rendimiento y latencia del mismo. Su funcionamiento básico consiste en que cada switch emulado envía un número configurable de mensajes de nuevos flujos (mensajes OpenFlow packet-in) al controlador OpenFlow, espera por las respuestas apropiadas de configuración de flujos (mensajes OpenFlow packet-out o mensajes OpenFlow de modificación de flujos flow-mod) y registra estadísticas de la diferencia de tiempo entre las solicitudes y las respuestas, así como otras métricas de desempeño.

- **Hcprobe**: Escrita en Haskell, permite crear con facilidad escenarios para pruebas de control SDN. Es capaz de simular un gran número de switch 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.
  - Un lenguaje específico de dominio embebido (EDSL) para la creación de pruebas.

  Además, Hcprobe proporciona un framework para la creación de varios casos de uso para estudiar el comportamiento de los controladores OpenFlow a través del procesamiento de diferentes tipos de mensajes. Con esta herramienta se puede generar todo tipo de mensajes OpenFlow switch-controlador y reproducir diferentes escenarios de comunicación entre ellos.

Por último vale la pena comentar que la mayoría de los controladores utilizan mensajes LLDP para descubrir la topología de la red, en la figura 8 se presenta gráficamente el mecanismo usado para el descubrimiento de la topología.

En la tabla 1 se presenta una comparación entre los distintos controladores SDN.

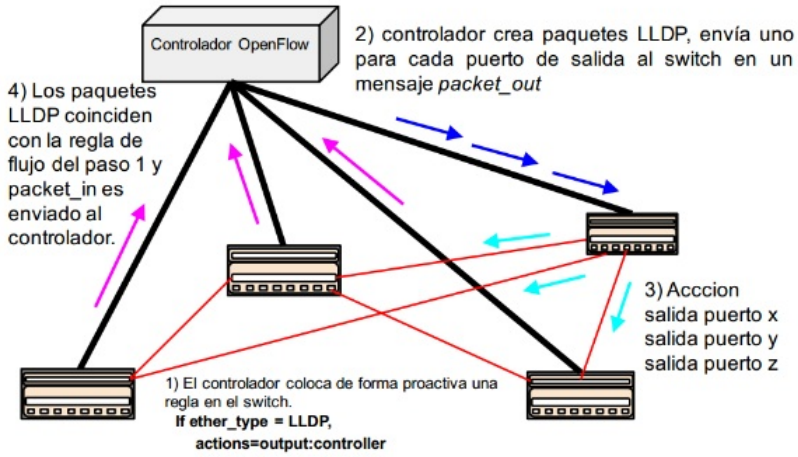

Figura 8: Descubrimiento de topología SDN [20]

	OF	Lenguaje	RestAPI	GUI	Plataforma	Documentación
NOX	1.0	C++	No	Python+, QT4	Linux	Media
Beacon	1.0	Java	No	Web	Linux, Windows, MAC OS, Android	Buena
Floodlight	1.0	Java	Si	Web	Linux, Windows, MAC OS	Buena
Trema	1.3	Ruby/C	Si	No	Linux	Media
POX	1.0	Python	No	Python+, QT4, Web	Linux, Windows, MAC OS	Buena
NodeFlow	1.0	Node.JS	No	No	Linux, Windows, MAC OS	Pobre
FlowER	1.0	Erlang	No	No	Linux	Pobre
Maestro	1.0	Java	No	No	Linux	Media
Ryu	1.0, 1.2, 1.3	Python	Si	Web	Linux	Buena
OpenDayLight	1.0	Java	Si	Web	Linux, Windows, MAC OS	Media
RouteFlow	1.0	?	?	Si	Linux	Pobre

Tabla 1: Tabla comparativa controladores.

2.3.4. Modelos de diseño SDN

Existen cinco modelos de diseño para SDN[21]

- **Estrategia de control**:
  - Centralizado: Un único controlador se hace cargo de la administración de todos los switches OpenFlow.
  - Distribuido: Se distribuye la administración de switches OpenFlow se tienen varios controladores que administran un conjunto de switches.
- **Manejo de flujos**:
  - Microflujos: Cada flujo es tratado individualmente, cada flujo tiene su entrada en la tabla de flujos. Se tiene un control más detallado de la red.

- Agregación: Una entrada cubre un grupo de flujos, con lo que se denomina entrada comodín. La tabla de flujos contiene una entrada por cada grupo. Es utilizado en para grandes cantidades de flujo.

- **Modelo de Población**:
  - Proactiva: La configuración proactiva de flujos se realiza antes de que el paquete llegue al switch. Así, cuando al switch llega el primer paquete, esté ya sabe que hacer con el mismo, esto da lugar a insignificantes retrasos, no existiendo límite real en el número de flujos por segundo que el controlador puede soportar. Idealmente, el controlador SDN pre-llena las tablas de flujo lo más posible.
  - Reactiva: La configuración reactiva de flujo se produce cuando el switch OpenFlow recibe un paquete que no coincide con las entradas de su tabla de flujo y, por tanto, tiene que enviarlo al controlador quien decide qué hacer con el mismo y almacena la nueva información en el switch en cuestión. El tiempo asociado con la configuración reactiva de flujo es la suma del tiempo que se necesita para enviar el paquete desde el switch OpenFlow al controlador SDN, el tiempo de procesamiento en el controlador y el tiempo que este tarda en llenar la tabla de flujo en el switch. Los factores claves que afectan al tiempo de establecimiento del flujo incluyen la capacidad de procesamiento de los switches que están conectados al controlador y el procesamiento y rendimiento de entrada / salida del controlador.

- **Forma de despliegue**:
  - Virtual: Asume la configuración de un switch dentro de un host. Es posible modificar arbitrariamente las capacidades, tal es el caso de la memoria, procesamiento, etc. La desventaja es que se encuentra limitado por el hardware sobre el que corre.
  - Físico: Usa un dispositivo físico que viene incorporado con la tecnología.

- **Consistencia**:
  - Completo: Se tiene certeza del estado de la red.
  - Eventual: La convergencia es más lenta, por tanto en ciertos momentos no se tiene certeza del estado de la red.

3. Emuladores/Simuladores SDN

En este punto se estudiarán algunos de los emuladores y simuladores de SDN que existen en la actualidad. Se escogieron cinco emuladores/simuladores a estudiar, todos ellos con características distintas. En primer lugar se seleccionó MiniNet, el cual es un emulador open source, su selección se debió a que es el emulador más usado por la comunidad y por esto, del que más material se dispone. Luego se eligió EstiNet el cual es un emulador y simulador de software propietario, su selección se debió a que interesaba comparar plataformas comerciales junto con las open source. De las otras plataformas comerciales, era la que ofrecía una mayor documentación. En tercer lugar se seleccionó NS-3, el cual es el simulador open source más usado por la comunidad y que dispone de mucho material para su uso. En cuarto lugar se eligió Distributed OpenFlow testebed (DOT), este es un emulador SDN distribuido, es open source, su selección se debe a su arquitectura distribuida. Por último se seleccionó KaanalNet, el cual es un emulador desarrollado en Node.JS, resultó interesante ver cómo funciona un emulador desarrollado con esta tecnología. Luego se presentarán otros emuladores/simuladores, los cuales no van a ser estudiados en este trabajo.

3.1. MiniNet

Es una plataforma de emulación de red, la cual es open source. Se pueden crear redes definidas por software (cuyas dimensiones pueden ser de hasta cientos de nodos, según la configuración deseada, dependiendo del HW donde se corra la plataforma) utilizando un único kernel linux. MiniNet permite crear, interactuar, personalizar y compartir de forma rápida un prototipo de red definido por software al mismo tiempo proporcionar un camino fácilmente adaptable para la migración a hardware. La primer versión de Mininet fue desarrollado por Bob Lantz y Brandon Heller, basado en el prototipo original creado por Bob Lantz en el 2009. La versión actual es 2.2.1 y corre sobre un sistema operativo Ubuntu 14.04 LTS. Mininet se ejecuta sobre una sola máquina y emula solo enlaces cableados, ofreciendo una virtualización parcial que limita su funcionalidad, ya que no puede manejar diferentes kernels del sistema operativo simultáneamente, por lo que todos los host comparten el mismo sistema de ficheros. MiniNet permite emular los siguientes componentes:

- **Links**: Es un enlace de Ethernet virtual con la función de actuar como un cable de red conectando dos interfaces virtuales, pudiendo enviar paquetes desde una interfaz a la otra, el comportamiento de este link es como el de un puerto de Ethernet real.
- **Hosts**: En Mininet un host es un proceso simple de Shell situado dentro de su propio espacio de nombres de red, cada host posee su propio enlace virtual.
- **Switches**: Son dispositivos con software OpenFlow que tienen el mismo comportamiento como un dispositivo real.

- **Controladores**: Un controlador puede estar en cualquier lugar de la red, tiene que tener conectividad ip con otros dispositivos (switch) cuando estos estén en ejecución.

La instalación de esta herramienta se puede realizar de manera muy sencilla, ya que desde su página web oficial[22], se puede descargar una máquina virtual preconfigurada (archivo ovf, para ser usado con VirtualBox). MiniNet cuenta con una gran documentación, muy detallada sobre su funcionamiento e instalación, lo cual es de gran utilidad para los usuarios. Una gran ventaja que posee esta esta herramienta, es que se pueden crear determinadas topologías, de forma muy rápida, utilizando un único comando.

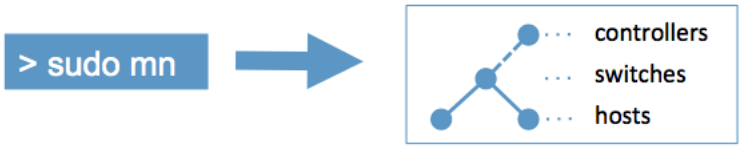

Figura 9: Mininet [22]

Algunas de las topologías que permite crear la herramienta son:

- Single: Un switch, N hosts
  # mn –topo=single,N
- Linear: N switches, cada uno conectado a un único host
  # mn –topo=linear,N
- Tree: Árbol, de profundidad N
  # mn –topo=tree,depth=N
- Definida por el usuario: A través de un archivo python el usuario puede crear su topología
  # mn –custom mytopo.py –topo mytopo

MiniNet por defecto soporta la versión 1.0 de OpenFlow, pero se puede activar el soporte para OpenFlow 1.1, 1.2 y 1.3. Cuenta con OpenVSwitch integrado.
mn –switch ovs,protocols=OpenFlow13

La estructura básica de un modelo de red en esta plataforma, se muestra en la figura 11.

La plataforma permite configurar el controlador SDN a utilizar, se puede utilizar un controlador gestionado por MiniNet o se puede utilizar un controlador gestionado por el usuario, indicando su dirección ip y su puerto.
mn –controller=remote,ip=127.0.0.1,port=6653
Si no se elige un controlador, Mininet utiliza el controlador que incorpora por defecto. La distribución OpenFlow de referencia incorpora un controlador que actúa como un switch clásico. En lenguaje OpenFlow, se le denomina Ethernet Learning Switch. Su funcionamiento es el de un switch tradicional: según vaya

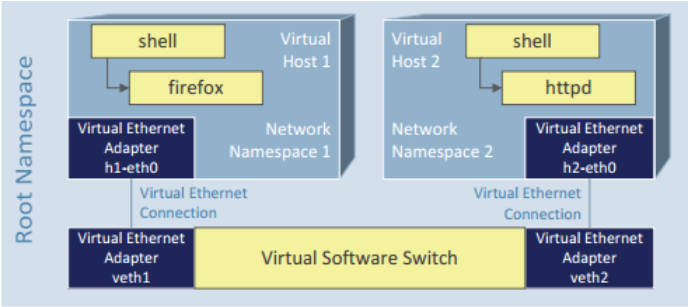

Figura 10: Estructura MiniNet [23]

recibiendo tramas irá creando una tabla donde se asociarán direcciones MAC con puertos. Éste controlador básicamente al asociar una MAC con un puerto, enviará un mensaje OpenFlow al switch, instaurando una nueva entrada en la tabla de flujo.

La plataforma cuenta con una GUI, llamada Miniedit, la cual se agregó desde la versión 2.2.0.1, está desarrollada en python, trayendo incorporados los elementos básicos de una red, como lo son, hosts, switches, controlador, links (cable virtual), routers, entre otros, permitiendo al usuario poder crear la topología escogiendo los dispositivos que desea. También permite poder exportar en python una topología creada para ejecutarla desde la línea de comandos de MiniNet.

3.2. EstiNet

EstiNet es una plataforma de emulación y simulación de varios protocolos de red; Uno de estos protocolos es OpenFlow. Esta plataforma es compatible con varios de los controladores SDN que se mencionaron más arriba. La versión actual de EstiNet es la 9.0 y tiene soporte para las versiones de OpenFlow 1.0, 1.1, 1.3, 1.3.2[24]. Es una herramienta de software propietario, utiliza una metodología de simulación innovadora, llamada "kernel reentering". Gracias a esto un único kernel soporta múltiples hosts y es capaz de simular múltiples switches OpenFlow, por esto, es una herramienta con la que se obtiene gran escalabilidad. EstiNet cuenta con buenas propiedades de simulación, gracias a la metodología "kernel reentering" se le permite a aplicaciones reales no modificados funcionar en máquinas simuladas. Debido a esta capacidad, los resultados de la simulación de EstiNet son tan precisos como los resultados obtenidos de un emulador[25]. La metodología "kernel reentering" para la simulación permite que un paquete entre, salga y reentre al kernel afectando el resultado en forma diferente en cada secuencia de la simulación. Por ejemplo: durante el inicio de una simulación el kernel trabaja como un host y en el siguiente paso de la simulación como un enrutador, y así consecutivamente. EstiNet utiliza su propio reloj de simulación para controlar el orden de ejecución de eventos de simulación, pero generando

resultados muy precisos. En una red OpenFlow simulada por EstiNet, controladores OpenFlow reales como NOX/POX, Ryu, y Foodlight se pueden ejecutar directamente en un host simulado para controlar switches OpenFlow simulados sin ninguna modificación. En cuanto a EstiNet como emulador, éste necesita llevar a cabo la emulación en tiempo real y permite que el controlador se ejecute en una máquina externa para controlar switches OpenFlow emulados. Debido a la utilización de la metodología del kernel reentering, EstiNet también permite al controlador y a los switches OpenFlow emulados ejecutarse en la misma máquina.

Cuenta con una interfaz de usuario (donde se diseña la topología de red a utilizar) y la animación de paquetes en el momento de la simulación, de esta forma se muestra de forma gráfica el comportamiento de la red. También cuenta con una buena forma de presentación de estadísticas de simulación.

Para descargar la herramienta, hay que registrarse en el sitio web oficial de EstiNet y solicitar la versión de prueba. Esta versión está licenciada por un mes. Junto con la descarga de la herramienta se incluyen algunos manuales para su utilización, los cuales cuentan con muy pocos ejemplos de uso. Al ser un producto comercial no se dispone de mucha información sobre su arquitectura. La plataforma corre sobre un sistema operativo Fedora 20 de 64 bits.

3.3. NS-3

NS-3[26] es un simulador de redes de eventos discretos:

- Cada evento ocurre en un instante determinado y modifica el estado del sistema.
- La simulación no recorre de forma progresiva el tiempo (simulador de tiempo continuo), sino que va saltando en el tiempo de un evento al evento siguiente.
- Las simulaciones son más rápidas.

NS-3 posee una implementación modular que contiene diferentes librerías que dan soporte al simulador (también es posible que los usuarios desarrollen sus propias librerías). Las más importantes:

- Core library: ofrece soporte para aspectos genéricos del simulador como generación de números aleatorios, utilizar punteros inteligentes, callbacks u objetos de depuración.
- Simulator library: define los parámetros requeridos para la ejecución de una simulación, como el tiempo de simulación, objetos, planificadores y eventos.
- Common library: define objetos independientes como paquetes genéricos.
- Node library: define las clases abstractas de objetos fundamentales del simulador como nodos, canales y dispositivos de la red.

- Internet-node: define los modelos relacionados con Internet, por ejemplo, los protocolos TCP y UDP.

La implementación modular, permite la compilación individual de pequeñas partes, de manera que a la hora de compilar solo se compile la del programa que ha cambiado. Los programas de ejecución de NS-3 pueden ser construidos estática o dinámicamente vinculados a las librerías.
NS-3 ofrece soporte para:

- Construcción de redes virtuales (nodos, canales, aplicaciones), planificadores de eventos, generadores de topologías, temporizadores, variables aleatorias y otro tipo de objetos para la simulación de sistemas de eventos discretos basados en Internet y en otros tipos de redes.
- Simular procesos que emiten y consumen paquetes de red reales.
- Simular múltiples procesos en diferentes máquinas.
- Animar las redes simuladas.
- Detección, registro, cálculo y estadísticas de la simulación.

El núcleo de simulación y modelos están implementados en C++ y tiene exportadas la mayoría de sus APIS a python, permitiendo de esta forma crear escenarios de simulación en este lenguaje. La estructura básica de un modelo de red en esta plataforma, se muestra en la figura 12.

- Node: Representa en la simulación un dispositivo de la red (Host, switch, router)
- Channel: Representa link que conecta los elementos de la red.
- NetDevice: Representa tarjeta de red de los elementos de la red.
- Protocol Stack: Se encarga de instalar en un nodo la pila de protocolos de internet.
- Application: Representan distintas aplicaciones contenidas en el nodo, para generar distintos tipos de tráfico.

La herramienta es open source y su versión actual es la 3.25 (Marzo 2016). La plataforma NS-3 es compatible con las siguientes plataformas:

- Linux x86 and x86-64: gcc versions 4.2 through 4.8.
- FreeBSD x86 and x86-64: clang version 3.3, gcc version 4.2.
- Mac OS X Intel: clang-500.2.79, based on LLVM 3.3svn (OS X Mavericks and Xcode 5.0.1), and gcc-4.2 (available with Xcode version 4 or earlier)

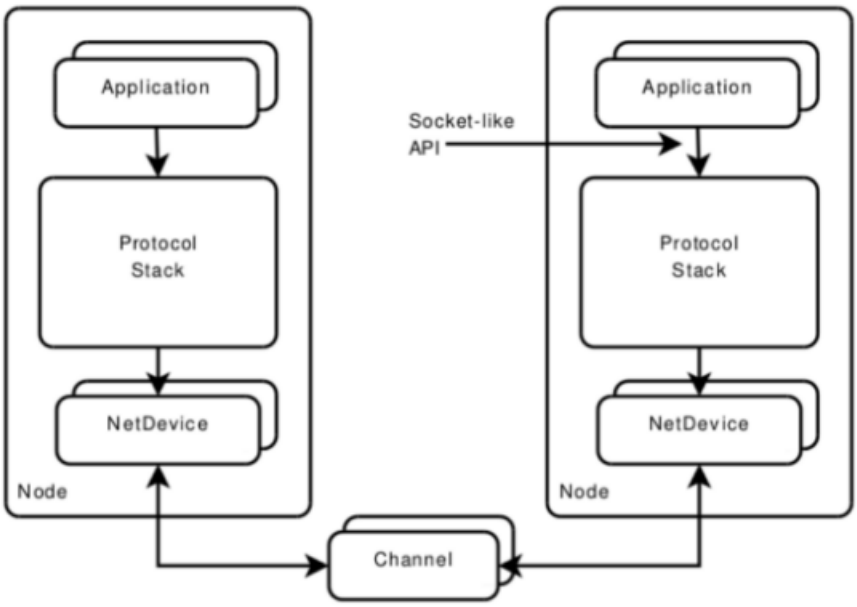

Figura 11: Estructura NS-3 [27]

Desde su versión 3.11 (Mayo 2011) cuenta con un módulo para la simulación de dispositivos OpenFlow, compatible con la versión 0.8.9 de OpenFlow. Ns-3 simula el funcionamiento de un switch OpenFlow compilando y enlazando un módulo C++. En la actualidad existe un módulo que da soporte a la versión 1.3 de OpenFlow (ofswitch13)[28]. A diferencia de MiniNet y EstiNet, ns-3 es únicamente simulador, no soporta las funcionalidades de emulación. Esta plataforma tampoco permite usar un controlador SDN real, externo a la plataforma, como si lo permiten EstiNet y MiniNet.
Se pueden simular dos tipos de controladores OpenFlow:

- DropController: Crea flujos para la tabla de flujos del switch e ignora paquetes individuales.
- LearningController: Analiza los paquetes de forma individual y es capaz de reconocer la dirección MAC de los dispositivos conectados.

El switch OpenFlow simulado con NS-3 tiene dos diferencias esenciales respecto a un switch OpenFlow "normal":

- No SSL: No cuenta con un canal seguro de comunicación con el controlador, esto no ocurre ya que se tiene un controlador embebido en la plataforma.
- Virtual Flow Table, TCAM: En un switch OpenFlow "normal" las tablas

de flujo típicamente están construidas con TCAMs, en NS-3 se implementa una librería que modela este tipo de hardware.

3.4. DOT

Distributed OpenFlow Testbed (DOT)[29] es un emulador de red, el cual está diseñado para implementar una emulación distribuida de una SDN OpenFlow. Para esto se despliega una infraestructura virtual de emulación (hosts, switches, enlaces) a través de múltiples máquinas físicas. DOT ofrece garantías de CPU time y ancho de banda para los elementos emulados. Así como también soporte a la hora de configurar y monitorear los componentes emulados.
Las características más importantes de este emulador son las siguientes:

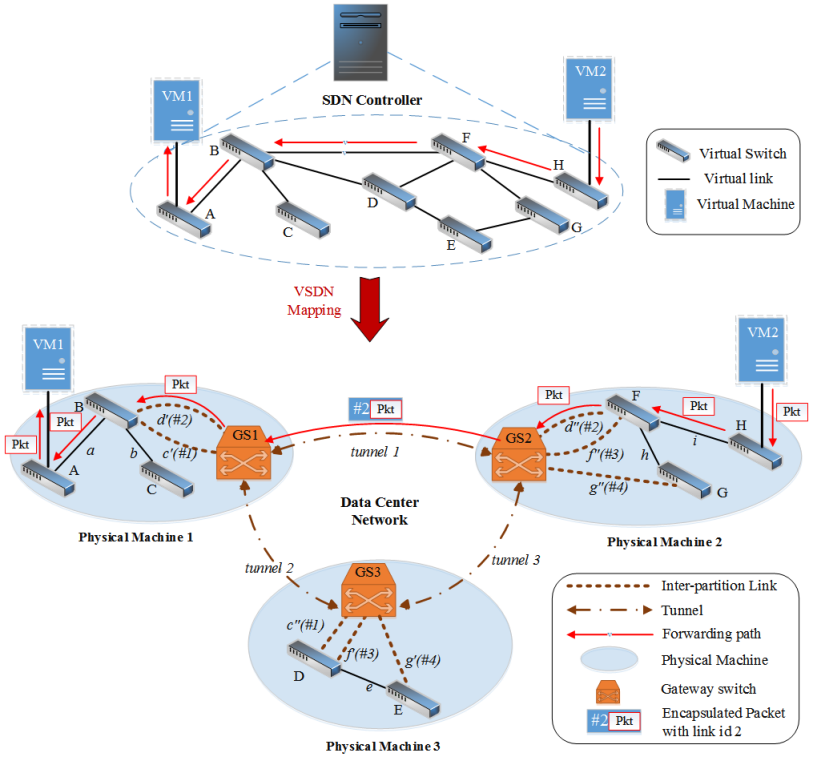

Figura 12: Arquitectura DOT [30]

- Escalabilidad: DOT puede ejecutarse sobre múltiples máquinas físicas, lo que hace que se logre una muy buena escalabilidad a la hora de la emulación.

- Performance garantizada: El algoritmo de emulación embebido en DOT ofrece garantías de recursos para todos los componentes emulados (hosts,

switches y links). Si la infraestructura física donde se corre la emulación es incapaz de satisfacer las necesidades de recursos para la emulación, el algoritmo no la inicia.

- Configuración Automatizada: El usuario a través de un archivo de configuración, define la infraestructura física a usarse y los requerimientos de emulación. El algoritmo de emulación toma este archivo de configuración y determina la ubicación óptima de los componentes a emular y los despliega a través del cluster de máquinas físicas.
- SDN desde cero: DOT permite la emulación de switches Open vSwitch y permite utilizar controladores SDN externos a la plataforma estableciendo los datos del mismo (IP, puerto). También provee controladores internos a la plataforma de emulación como por ejemplo Floodlight y POX.
- Transparencia: DOT conecta diferentes particiones de la infraestructura virtual a través de túneles IP. Se introducen varios componentes especializados para facilitar la implementación distribuida y la emulación. Todos estos componentes se mantienen ocultos al usuario, así como también al controlador SDN. Por lo tanto, a ambos se les da la ilusión de que sólo la infraestructura virtual solicitada está siendo emulada en DOT.

DOT cuenta con una arquitectura con dos componentes principales, los cuales tendrían que correr en máquinas físicas distintas. En la figura 12 se presenta dicha arquitectura.

- DOT Manager: Administran y monitorean la operación de los DOT Nodes. Es responsable de la asignación de recursos para la red emulada, segun la definicion del usuario en el archivo de configuracion. Cuenta con dos módulos, "provisioning module", el cual se encarga de ejecutar el algoritmo que realiza el mapeo entre componentes a emular y componentes físicos. Una vez que se determina la colocación de los componentes virtuales, la información se transmite a los nodos seleccionados. El otro módulo es el "statistics collection module", el cual se encarga de recoger diversos tipos de estadísticas de la red emulada.
- DOT Nodes: Alojan las instancias virtuales (hosts, switches , links) de la infraestructura emulada. Se tiene un DOT Node por cada máquina física a usar. Se compone de dos módulos: "host provisioning module" y "logging module". El primero es el responsable de la configuración y la asignación de recursos necesarios. Mientras que el segundo, recoge estadísticas de los dispositivos emulados en el DOT Node (utilización de recursos, tasa de paquetes, rendimiento, retardo, pérdida de paquetes y mensajes OpenFlow). DOT Node utilia Open vSwitch, libvirt y KVM para emular la topología configurada.

Cada uno de los componentes de la plataforma corren sobre un sistema operativo Ubuntu 12.04 LTS y tiene soporte para las versiones 1.0 y 1.3 de OpenFlow. Esta plataforma no cuenta con mucha documentación, solo se dispone de

un tutorial en la página web oficial de la misma. Su versión actual es la 1.0 y es de octubre del 2014.

3.5. KaanalNet

KaanalNet[31] es un emulador de red, open source, el cual se puede correr en un simple pc. Con este emulador se pueden emular tanto redes tradicionales como también redes definidas por software. La plataforma está desarrollada con Node.JS y corre bajo un sistema operativo Ubuntu 14.04. Su versión actual es la 0.2.0 (Noviembre 2015). Utiliza la tecnología LXC (Linux Containers)[32] para la emulación de los distintos dispositivos.
LXC es una tecnología de virtualización en el nivel de sistema operativo (SO) para Linux. LXC permite que un servidor físico ejecute múltiples instancias de sistemas operativos aislados, conocidos como Servidores Privados Virtuales (SPV o VPS en inglés) o Entornos Virtuales (EV). LXC no provee de una máquina virtual, más bien provee un entorno virtual que tiene su propio espacio de procesos y redes.

Es similar a otras tecnologías de virtualización en el nivel de SO como OpenVZ y Linux-VServer, asimismo se asemeja a aquellas de otros sistemas operativos como FreeBSD jail y Solaris Containers.

LXC se basa en la funcionalidad cgroups del Linux que está disponible desde la versión 2.6.29, desarrollada como parte de LXC. También se basa en otras funcionalidades de aislamiento de espacio de nombres, que fueron desarrolladas e integradas dentro de la línea principal del núcleo de Linux.
Algunas de sus características son:

- Soporte REST API.
- Soporte para redes tradicionales.
- Soporte SDN.
- Propiedades de enlaces configurables (ancho de banda, latencia, jitter, pérdida de paquetes).
- Soporte para controlar los nodos de la red emulada (start, stop, eliminar nodos).
- Testeo de tráfico (icmp,udp,tcp).
- Integración con los controladores RYU y POX

3.6. Otros emuladores/simuladores SDN

- OMNet++: Es un simulador modular de eventos discretos de redes orientado a objetos (desarrollado en C++), usado habitualmente para modelar el tráfico de redes de telecomunicaciones, protocolos, sistemas multiprocesadores y distribuidos, validación de arquitecturas hardware, evaluación

del rendimiento de sistemas software y, en general, modelar cualquier sistema que pueda simularse con eventos discretos. Esta herramienta está disponible tanto para sistemas operativos basados en UNIX como para Windows y se distribuye bajo la Licencia Pública Académica. Su versión comercial, denominada OMNEST, es desarrollada actualmente por Opensim Ltd[33]. Cuenta con un IDE basado en eclipse y un entorno de ejecución gráfica. Cuenta con una extensión para dar soporte OpenFlow en su versión 1.0.

- Trema: Es un framework para la programación de controladores OpenFlow open source desarrollado en C y Ruby, con el que se puede llevar a cabo un proyecto de emulación OpenFlow. Esta herramienta corre sobre el sistema operativo Fedora, su versión actual es la 0.4.8 y tiene soporte para las versiones 1.0, 1.3 y 1.3.1 de OpenFlow[11][34].

- Veryx PktBlaster SDN: Es una solución integrada que permite evaluar comparativamente el rendimiento de controladores SDN, a través de la emulación y simulación de diferentes topologías. Es un producto comercial y tiene soporte para las versiones 1.0 y 1.3 de OpenFlow.

- VNX: Como parte de los proyectos de investigación realizados por la Universidad Politécnica de Madrid, se desarrolló la herramienta de simulación open source VNX (Virtual Networks over Linux) que permite la creación de escenarios de prueba virtuales (testbeds) de forma automática. Su principal objetivo es brindar a los estudiantes un entorno donde puedan interactuar de forma más real con escenarios de red, de forma similar a los programas GNS3, NetKit, entre otros. Para poder dar soporte a la simulación de SDN se debe proveer a VNX de dispositivos que soporten OpenFlow. En primer lugar un switch (por ejemplo: Open vSwitch, que permite la simulación de un switch para ambientes virtualizados y que tiene soporte para el protocolo OpenFlow). Como segundo elemento un software externo para el control de la red (por ejemplo: POX, el cual es el controlador más reciente dentro del stack SDN y que además es el más utilizado en la actualidad). Finalmente como tercer elemento se encuentra el protocolo OpenFlow[35].

- MaxiNet: Es una extensión de MiniNet, en la cual se utiliza una arquitectura distribuida para la emulación de redes definidas por software. Con esta herramienta se pueden emular miles de máquinas virtuales en algunas pocas máquinas físicas. Cada una de las máquinas físicas ejecuta una instancia de MiniNet. Se les llama Workers a los equipos físicos que corren las instancias de MiniNet y Frontend al equipo se encarga de administrar los Workers[36].

3.7. Comparación de Emuladores/Simuladores

Plataforma	Versión	Developer	SO	OF	Emulador	Simulador	Documentos	Código
MiniNet	2.2.1	Universidad de Stanford	Ububtu 14.04 LTS	1.0, 1.1, 1.2, 1.3	SI	NO	Buena	OpenSource
EstiNet	9.0	EstiNet Technologies Inc	Fedora 20	1.0, 1.3	SI	SI	Media(oficial)	Propietario
NS-3	3.25	ns-3 Project	Linux, FreeBSD, MAC OS	0.8.9, 1.3	NO	SI	Buena	OpenSource
DOT	1.0	WatSDN group	Ubuntu 12.04 LTS	1.0, 1.3	SI	NO	Pobre	OpenSource
KaanalNet	0.2.0	Suresh Kumar (GitHub)	Ubuntu 14.04	1.0, 1.3	SI	NO	Pobre	OpenSource

Tabla 2: Tabla comparativa plataformas.

4. Emulando/Simulando topologías SDN

En este punto se mostrara como emular/simular una topología simple, en cada uno de los emuladores y simuladores presentados anteriormente.

4.1. MiniNet

- Como primer paso se debe instalar VirtualBox para poder importar la máquina virtual que ofrece el sitio oficial de MiniNet. Luego se importa la máquina virtual y se inicia.
- Luego se debe instalar un gestor de ventanas x11 (ej: Xming server) y un cliente ssh (ej: putty), de esta forma se podrán correr aplicaciones visuales desde el cliente ssh. Esto será útil para levantar wireshark en la máquina virtual donde corre MiniNet, y así poder capturar y observar el flujo de paquetes OpenFlow.
- Ahora se creará una topología single, con cinco hosts conectados a un switch OpenFlow, en este caso se utilizará el controlador SDN por defecto de MiniNet y se utilizara un switch Open vSwitch
  #sudo mn –topo single,5 –switch ovs
- El comando anterior se creará la topología y se accedera a la consola de administración de MiniNet. Mediante el comando "pingall"se puede probar la conectividad de todos los hosts, y con la herramienta wireshark se puede observar el intercambio de paquetes OpenFlow entre el controlador y el switch. Con la herramienta .ºvs-ofctl"podemos observar la tabla de flujos del switch, a través del comando:
  # sudo ovs-ofctl dump-flows s1

4.2. EstiNet

- Como primer paso se debe instalar VMware Player, en donde se creará una máquina virtual con el sistema operativo Fedora 20 de 64 bits, que viene incluido en la descarga de EstiNet.
- Luego se instala y se licencia EstiNet en la máquina virtual Fedora, como indica el manual de instalación (incluido en la descarga de la plataforma).
- Se levanta la interfaz de usuario de EstiNet, ejecutando los siguientes comandos en consola:
  # dispatcher
  # coordinator
  # estinetgui
- Los elementos para simular una red SDN se encuentran en la pestaña OpenFlow y son los siguientes:

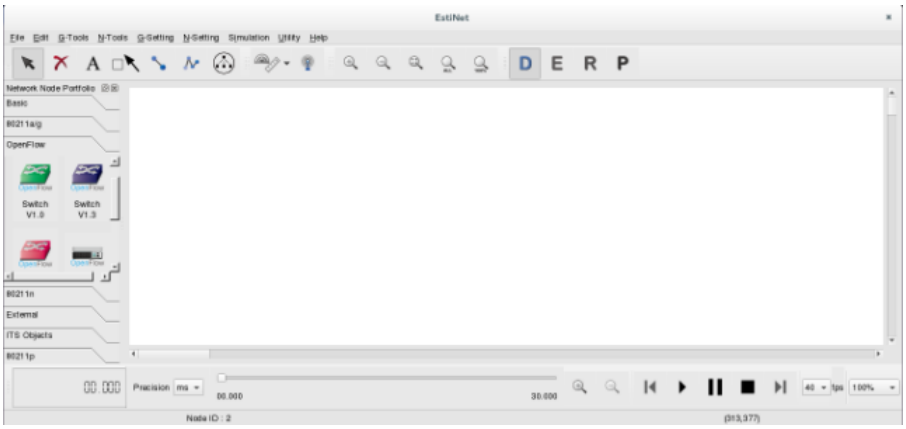

Figura 13: EstiNet GUI

Representa un controlador OpenFlow, que es ejecutado en el entorno de emulación.

Representa un controlador OpenFlow, ejecutado fuera del entorno de emulación.

Representan switches OpenFlow con soporte para las versiones 1.0 y 1.3 respectivamente, estos son dispositivos de capa 2.

Representan switches OpenFlow con soporte para las versiones 1.0 y 1.3 respectivamente, estos son dispositivos de capa 2.
En la pestaña basic, se encuentran los hosts, los cuales se conectaran a los switches OpenFlow. En la barra superior encontramos el icono de links con los cual unimos los elementos. Los links grises representan conexiones de la capa de control, mientras que los negros representan conexiones de la capa de datos. A los links se les puede configurar el delay, el ancho de banda y el BER.

- Una vez presentados los elementos SDN, se puede crear la topología deseada, para esto, se debe seleccionar la letra “D” en la barra superior de la interfaz de usuario y allí seleccionar cada uno de los elementos a utilizar. Se utilizará un controlador externo, el cual será pox, y correrá en la máquina donde se encuentra instalado EstiNet.

- Como último paso se ejecuta la simulación, para esto se selecciona la letra R, y en la barra superior se selecciona “Simulation” – “Run”.

4.3. NS-3

- Como primer paso se crea una máquina virtual (con virtual box) Linux, se usará el sistema operativo Ubuntu 14.04, luego se descarga, instala y compila la plataforma NS-3 con se indica en su manual de instalación[37].

- Se compila la plataforma para dar soporte a OpenFlow, siguiendo las instrucciones del manual “OpenFlow switch support”. De esta forma se tiene soporte para OpenFlow en su versión 0.8.9.

- Para crear la topología, se debe escribir un programa en c++, el cual utiliza el módulo OpenFlow. Se modifica la topología de ejemplo (se anexa openflow-linear5-topology.cc) que viene con la instalación del módulo OpenFlow para crear una topología “Linear, 5”. Y se programa la generación de tráfico udp entre los hosts de la topología. Luego se compila la nueva aplicación creada. Se debe agregar la aplicación al directorio de OpenFlow y editar el archivo wsscript para agregar la aplicación creada.
  # ./waf build

- Como último paso se ejecuta la simulación
  # ./waf –run .ºpenflow-linear5-topology -v”

4.4. DOT

- Como primer paso se crean dos máquinas virtuales (con virtaul box) con el sistema operativo Ubuntu 12.04. En una se instalará el componente DOT Manager y en la otra se instalará un componente DOT Node. Se sigue el manual de instalación de DOT[38].

- Como siguiente paso se debe levantar un controlador SDN a utilizar (por ejemplo RYU o POX). Puede ser en cualquiera de las maquinas, por ejemplo la que contiene el componente DOT Manager.

- Se crea el archivo de configuración donde se indica la infraestructura física y la topología a emular (Linear5.txt, se anexa archivo y se explica cada uno de sus tags). También se indica la versión de OpenFlow y donde se encuentra el controlador SDN a utilizar.

- En la maquina que contiene el componente DOT Manager se ejecuta la emulación. Se ingresa al directorio donde se instalo DOT, y se ejecuta el siguiente comando:
  # ./run.sh Linear5.txt

- Por último se accede a la consola de administración de DOT (# ./dot_console) y se prueba la conectividad de los hosts, mediante la aplicación ”pingAll”

4.5. KaanalNet

- Como primer paso se crea una máquina virtual (con virtual box) con el sistema operativo Ubuntu 14.04 y se instala la plataforma con sus dependencias, como indica su manual de instalación[39]
- Como siguiente paso se debe levantar un controlador SDN a utilizar (por ejemplo RYU o POX).
- Luego se debe iniciar la aplicación para la emulación, donde se debe indicar el switch OpenFlow (Open vSwitch, viene con la plataforma) a utilizar, la dirección ip y el puerto donde escucha el controlador.
  # sudo npm start – -S openvswitch -C tcp:ip:puerto
- Como siguiente paso se crea un archivo de configuración en formato json (Se anexa Linear5.json) en el cual se indica la topología a emular. Y se carga la configuración en la plataforma (A través de un request REST API).
  curl -XPOST -H 'Content-Type:application/json' -H 'Accept: application/json' –data-binary @Linear5.json http://localhost:5050/Topology -v -s
  Esto devolverá una respuesta json que contiene la información de los dispositivos creados y un identificador de la topología creada.
- Luego se puede obtener el estado de los dispositivos creados en la topología, a través de un request HTTP. Ej: curl http://localhost:5050/Topology/id-topologia
- Luego se lanzará un test de tráfico utilizando un REQUEST REST API. Ej: curl -XPOST http://localhost:5050/Topology/:id/Test
- Por ultimo se elimina la topología creada, mediante un REQUEST REST API. Ej: curl -XDELETE http://localhost:5050/Topology/:id

5. Experimentos

Se realizarán distintos tipos de pruebas en los emuladores/simuladores y veremos el rendimiento de cada uno. Las aplicaciones que se ejecutarán serán: spanning tree y network discovery. Se utilizaran los controladores Ryu y POX para ejecutar estas aplicaciones.

5.1. Controladores a utilizar

En este punto se presentarán los dos controladores a utilizar para realizar las pruebas.

Ryu

Como se mencionó anteriormente Ryu Está escrito en Python. Dispone de un conjunto de APIs bien definidas, que permite el desarrollo de nuevas aplicaciones de gestión de red, y de varios protocolos para la gestión de los dispositivos de red, como OpenFlow y Netconf. Está escrito en Python soporta completamente desde la versión 1.0 hasta la 1.5 de OpenFlow. Su versión actual es la 3.6. Para su instalación se debe clonar su código desde github y ejecutar el script de instalación.
git clone git://github.com/osrg/ryu.git
cd ryu
python ./setup.py install

Para darle determinada funcionamiento a un controlador ryu se deben escribir aplicaciones ryu, las cuales se escriben a través de scripts Python. Una vez escrita la aplicación ryu, se ejecuta el controlador con la aplicación de la siguiente manera:
ryu-manager app.py

POX

POX es un controlador derivado de NOX, que fue desarrollado por NICIRA y destinado a la comunidad científica. Proporciona un marco de desarrollo para la investigación sobre protocolos de comunicación y componentes en redes SDN. POX es una plataforma de código abierto especialmente pensada para la investigación y el desarrollo de controladores OpenFlow. Está desarrollado y permite la programación de componentes en Phyton. Los componentes desarrollados son los que dotan de funcionalidad al controlador. POX requiere de la versión 2.7 de python para su correcto funcionamiento y tiene soporte para plataformas Linux, Windows y MAC. Tiene soporte para la versión 1.0 de OpenFlow e incluye soporte para Open vSwitch. Para su instalación solo se requiere clonar su código desde github, mediante el comando:
git clone http://github.com/noxrepo/pox

Luego se ingresa al directorio obtenido mediante git (# cd pox) y ya se está en condiciones de ejecutar POX. La ejecución de POX se realiza a través del comando:
./pox.py componente
Dónde "componente" es un archivo python que que contiene la implementación del componente POX a utilizar. Como ya se dijo, estos componentes son los que determinan la funcionalidad del controlador. Es posible ejecutar POX con más de un componente. POX ofrece varios componentes ya implementados, a continuación presentamos algunos de ellos[40]:

- forwarding.hub: Este componente configura una entrada comodín, en cada uno de los switches de la red, esta entrada tiene como acción enviar el paquete recibido por todos los puertos del switch, convirtiendo el switch en un hub.
- forwarding.l2_learning: Con este componente los switches OpenFlow se comportan como switches de aprendizaje de capa 2. De esta forma el switch OpenFlow es capaz de reconocer la dirección MAC de los dispositivos conectados.
- forwarding.l3_learning: Con este componente el switch OpenFlow es capaz de analizar paquetes ARP, pudiendo así, construir respuestas y solicitudes ARP. Cabe aclarar que este componente no convierte al switch OpenFlow en un router.
- openflow.discovery: Este componente realiza el descubrimiento de la topologia de red mediante el uso de paquetes LLDP.
- openflow.spanning_tree: Este componente utiliza el componente de descubrimiento de la topología para realizar el spanning tree conformado por los switches OpenFlow de la red.
- openflow.of_01: Este componente da soporte al protocolo OpenFlow, por defecto se utiliza con el controlador, para no utilizar el soporte OpenFlow se debe agregar la opción –no-openflow

5.2. Switch OpenFlow a utilizar

En este punto se presentará el switch Open vSwitch, el cual se utilizara para realizar los experimentos.

Open vSwitch
Open vSwitch (OVS)[41] es un software (de código abierto, bajo licencia Apache 2.0) multicapa para switches, cuyo objetivo es brindar un switch con calidad de produccion, que pueda soportar interfaces de administración estándares y que abra a programación y control externo las funciones de transmisión. Open vSwitch es una de las implementaciones más populares de OpenFLow. Está bien adaptado para funcionar como un switch virtual en ambientes implementados

con máquinas virtuales. Además de exponer interfaces estándar de control y visibilidad con la capa de red virtual, fue diseñado para soportar una distribución a través de múltiples servidores físicos. Open vSwitch soporta numerosas tecnologías de virtualización basadas en Linux:

- Xen/XenServer: es un monitor de máquinas virtuales de código abierto.
- KVM (Kernel-Based-Virtual Machine): es una solución para implementar virtualización completa Linux.
- VirtualBox: es un software para realizar la virtualización de equipos.
- Proxmox VE: Es una solución completa de virtualización de servidores basada en sistemas de código abierto. Permite la virtualización tanto sobre KVM como contenedores y gestiona máquinas virtuales, almacenamiento, redes virtualizadas y clústeres HA.

También se ha integrado en muchos sistemas de gestión virtuales como OpenStack, OpenQRM, OpenNebula y oVirt. Open vSwitch permite más capacidades que los módulos regulares de kernel Linux, aun cuando el Datapath está dentro del propio kernel GNU/Linux, lo que hace ideal para la construcción de esquemas de redes virtuales para nubes o para investigación de nuevos protocolos de red. La mayor parte del código está escrito sobre una plataforma independiente en C y es fácilmente portable a otros entornos.
Su versión actual es la 2.5.0 y brinda soporte para OpenFlow desde su versión 1.0.

OVS consta de un diseño más complejo que los bridges, consta de varios componentes y se ejecuta tanto en el espacio de kernel, como en el espacio de usuario. En un sistema operativo existen dos separaciones en memoria, la primera corresponde al espacio Kernel donde se ejecutan los módulos y los drivers del sistema operativo, la segunda separación corresponde al espacio de usuario donde se encuentra la mayoría del software. Esta separación ofrece una mayor seguridad ya que protege al sistema de fallos o ataques. Una de las virtudes de OVS es que permite procesar paquetes "nuevos" eso significa que, en el caso de recibir un tipo de paquete que pertenece a un flujo desconocido es capaz de tomar la decisión de cómo procesarlo, ya que esta no aparece en el caché. Esta decisión se toma en el espacio de usuario, pero a partir de este, todos los paquetes del mismo tipo y que proceden del mismo flujo son enviados directamente al espacio Kernel, de esta forma se consigue mejorar el rendimiento y rebajar el tiempo de procesado.
Existe otras implementaciones de switches OpenFlow, como por ejemplo Pica8[42] e Indigo[43].

5.3. Aplicaciones a probar

En este punto se presentan las dos aplicaciones que se probarán en los distintos controladores. Para las pruebas se utilizarán los componentes de cada uno

de los controladores que implementan dichas aplicaciones.

Spanning Tree
Esta aplicación se encarga de formar un árbol en una red switcheada de forma de evitar bucles por causa de enlaces redundantes, esto lo logra mediante el uso del protocolo STP (spanning tree protocol). Este es un protocolo de gestión de capa de enlace. El objetivo del protocolo es que en cada instante exista un solo camino activo entre dos switches (pueden existir loops físicos pero no lógicos). Para esto se define un árbol a través del cual se alcanza a todos los switches pero el árbol se "poda" de tal forma que algunos puertos quedan bloqueados a la espera de algún cambio topológico y los restantes puertos están en estado forwarding. Para probar esta aplicación se utilizará el componente de POX openflow.spanning_tree y se utilizará una aplicación ryu (simple_switch_stp.py) ya desarrollada por el RYU project team

Network Discovery
Mediante esta aplicación el controlador SDN puede realizar el descubrimiento de la topología de red, como ya se comentó anteriormente los controladores que se utilizaran para las pruebas (pox y ryu) utilizan paquetes LLDP para realizar el descubrimiento de la topología. Tal como se muestra en la figura 8.
Para probar esta aplicación se utilizará el componente de POX openflow.discovery, la cual utiliza mensajes LLDP para realizar el descubrimiento de la topología Y por otro lado, se modificara la aplicación simple_switch.py la cual viene incorporada con la instalación de ryu para agregarle el descubrimiento de la red a traves de paquetes LLDP (simple_switch_nd.py), esto se logra con la libreria Topology de ryu.
LLDP es un Protocolo de Descubrimiento de Vecindario (Neighbor Discovery Protocol NDP), el cual ha sido diseñado para dispositivos de redes Ethernet (como switches y routers), Los NDP se usan para recibir y/o transmitir información relacionada con los dispositivos de otros nodos de la red, también para almacenar la información aprendida acerca de otros dispositivos. LLDP es un protocolo de "un salto"; es decir que la información LLDP sólo se puede enviar y recibir por medio de dispositivos adyacentes, los cuales están conectados directamente entre sí por el mismo enlace, dichos dispositivos se denominan "vecinos". La información anunciada nunca se reenvía a otros dispositivos en la red. Los controladores encapsulan el mensaje LLDP en un mensaje OpenFlow.

5.4. Pruebas

Las pruebas sobre MiniNet y NS-3 se ejecutaron en máquinas virtuales corriendo sobre un equipo con sistema operativo windows 7, con un procesador Intel(R) Core(TM) i7 CPU 860 @ 2.80 GHz y 4GB de memoria RAM. Las pruebas sobre KaanalNet se intentaron ejecutar sobre la infraestructura anterior, pero al no resultar satisfactorias (los recursos de hardware no eran suficientes para la emulación de los escenarios) se utilizó la infraestructura de facultad, virtualizando la plataforma con una máquina virtual KVM con 32GB de memoria

RAM y un procesador AMD Opteron 23xx (Gen 3 Class Opteron). Las pruebas sobre DOT no resultaron satisfactorias en ninguna de las infraestructuras antes mencionadas ya que el módulo encargado de la emulación, no encontraba recursos necesarios para realizarla.

Las pruebas se realizarán sobre conjunto definido de topologías, a continuación se definen las topologías a utilizar:

- **Topología 1**:
  Cantidad de Switches: 10
  Cantidad de hosts: 20
  Cantidad de links: 65
  Cantidad de Controladores: 1
  Esta será una topología completa (cada switch se conectara al resto) y cada switch tendrá conectados dos hosts.

- **Topología 2**:
  Cantidad de Switches: 20
  Cantidad de hosts: 40
  Cantidad de links: 230
  Cantidad de Controladores: 1
  Esta será una topología completa (cada switch se conectara al resto) y cada switch tendrá conectados dos hosts.

- **Topología 3**:
  Cantidad de Switches: 30
  Cantidad de hosts: 30
  Cantidad de links: 59
  Cantidad de Controladores: 1
  Esta será una topología lineal (cada switch se conectará a dos switches, salvo los switches de los extremos, que se conectan a un único switch) y cada switch tendrá conectados un único host.

- **Topología 4**:
  Cantidad de Switches: 50
  Cantidad de hosts: 50
  Cantidad de links: 99
  Cantidad de Controladores: 1
  Esta será una topología lineal (cada switch se conectará a dos switches, salvo los switches de los extremos, que se conectan a un único switch) y cada switch tendrá conectados un único host.

- **Topología 5**:
  Cantidad de Switches: 30
  Cantidad de hosts: 60
  Cantidad de links: 90
  Cantidad de Controladores: 1
  Esta será una topología en forma de anillo (cada switch se conectará a

dos switches, y formarán un anillo) y cada switch tendrá conectados dos hosts.

- **Topología 6**:
  Cantidad de Switches: 30
  Cantidad de hosts: 120
  Cantidad de links: 150
  Cantidad de Controladores: 1
  Esta será una topología en forma de anillo (cada switch se conectará a dos switches, y formarán un anillo) y cada switch tendrá conectados cuatro hosts.

- **Topología 7**:
  Cantidad de Switches: 50
  Cantidad de hosts: 300
  Cantidad de links: 350
  Cantidad de Controladores: 1
  Esta será una topología en forma de anillo (cada switch se conectará a dos switches, y formarán un anillo) y cada switch tendrá conectados seis hosts.

- **Topología 8**:
  Cantidad de Switches: 73
  Cantidad de hosts: 512
  Cantidad de links: 584
  Cantidad de Controladores: 1
  Esta será una topología en forma de árbol con profundidad tres y cada nodo tendrá ocho hijos. De esta forma los switches del último nivel tendrán conectado ocho hosts.

- **Topología 9**:
  Cantidad de Switches: 127
  Cantidad de hosts: 128
  Cantidad de links: 254
  Cantidad de Controladores: 1
  Esta será una topología en forma de árbol con profundidad siete y cada nodo tendrá dos hijos. De esta forma los switches del último nivel tendrán conectado dos hosts.

- **Topología 10**:
  Cantidad de Switches: 1
  Cantidad de hosts: 1000
  Cantidad de links: 1000
  Cantidad de Controladores: 1
  Esta será una topología single (un único switch conectado a todos los hosts).

- **Topología 11**:
  Cantidad de Switches: 1
  Cantidad de hosts: 2000
  Cantidad de links: 2000
  Cantidad de Controladores: 1
  Esta será una topología single (un único switch conectado a todos los hosts).

- **Topología 12**:
  Cantidad de Switches: 1
  Cantidad de hosts: 5000
  Cantidad de links: 5000
  Cantidad de Controladores: 1
  Esta será una topología single (un único switch conectado a todos los hosts).

MiniNet

La máquina virtual (en VirtualBox) para las pruebas sobre MiniNet utilizando los controladores POX y Ryu tiene instalado un sistema operativo Ubuntu 14.04, 3GB de memoria RAM y la versión de mininet instalada es la 2.2.1, se utiliza la versión 0.2.0 de POX y se utiliza Ryu en su versión 4.3. En las pruebas a realizar ambos controladores se encuentran corriendo en la misma máquina donde se ejecuta MiniNet. Los switches emulados son Open vSwitch en su versión 2.0.2. Las pruebas se realizarán sobre la versión 1.0 del protocolo OpenFlow, ya que es la unica version del protocolo que soporta POX. Para probar la conectividad entre hosts se utiliza el comando pingall de MiniNet. Los escenarios para las distintas pruebas son sin pérdida de paquetes y un retraso y ancho de banda variable, dependiente de la carga de los equipos.
Para crear las topologías se ejecutan los comandos:

- **Topología 1**: sudo mn –custom topo1.py –topo Topo1 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 2**: sudo mn –custom topo2.py –topo Topo2 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 3**: sudo mn –topo=linear,30 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 4**: sudo mn –topo=linear,50 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 5**: sudo mn –custom topo5.py –topo Topo5 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 6**: sudo mn –custom topo6.py –topo Topo6 –switch ovs,protocols=OpenFlow10 –controller remote

- **Topología 7**: sudo mn –custom topo7.py –topo Topo7 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 8**: sudo mn –topo=tree,depth=3,fanout=8 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 9**: sudo mn –topo=tree,depth=7,fanout=2 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 10**: sudo mn –topo=single,1000 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 11**: sudo mn –topo=single,2000 –switch ovs,protocols=OpenFlow10 –controller remote
- **Topología 12**: sudo mn –topo=single,5000 –switch ovs,protocols=OpenFlow10 –controller remote

Se anexan todas las topologías realizadas (topo1.py, topo2.py, topo5.py, topo6.py y topo7.py). Para levantar los controladores se ejecutan los comandos:

- **POX**
  Spanning tree: sudo ./pox.py forwarding.l2_learning openflow.discovery openflow.spanning_tree openflow.of_01
  Network Discovery: sudo ./pox.py forwarding.l2_learning openflow.discovery openflow.of_01
- **Ryu**
  Spanning tree: sudo ryu-manager –ofp-tcp-listen-port 6633 simple_switch_stp.py
  Network Discovery: sudo ryu-manager –ofp-tcp-listen-port 6633 simple_switch_nd.py –observe-links

Pruebas :
Cada una de las pruebas se ejecutaron 10 veces y aquí se muestra un promedio del tiempo de esas 10 ejecuciones.
Las topologías 11 y 12 no fueron posibles emularlas con la infraestructura física propuesta por lo que se intentaron emular con la infraestructura brindada por la facultad, la cual consta de una máquina virtual (en KVM) con 32GB de memoria RAM y un procesador AMD Opteron 23xx (Gen 3 Class Opteron), pero tampoco fueron satisfactorias las pruebas ya que no se lograron emular mas de 1700 hosts.

EstiNet

La máquina virtual (en VMware Player) para las pruebas sobre EstiNet tiene instalado un sistema operativo Fedora 20, 3GB de memoria RAM y la versión de EstiNet instalada es la 9.0 en su versión de prueba (licenciada por un mes), se utiliza la versión 0.2.0 de POX y se utiliza Ryu en su versión 4.3. Al momento de comenzar a ejecutar las pruebas se encontró con la dificultad que la versión de de prueba que ofrece EstiNet no permite simular una topología con gran cantidad de nodos, ni siquiera se llegó a probar con la primer topología

	Crear Topología	STP(Pox-Ryu)	ND(Pox-Ryu)	Eliminar Topología
Topo1	3s	8s - 41s	7s - 2s	1.5s
Topo2	9s	10s - 79s	9s - 3s	4s
Topo3	4s	10s - 39s	10s - 3s	2s
Topo4	6s	11s - x	11s - 2s	2s
Topo5	5s	10s - 42s	10s - 1.5s	4s
Topo6	6s	10s - 45s	10s - 2s	4s
Topo7	16s	11s - x	11s - 2.5s	12s
Topo8	32s	14s - 43s	13s - 4s	18s
Topo9	16s	18s - 50s	18s - 4.5s	13s
Topo10	60s	x	x	44s
Topo11	x	x	x	x
Topo12	x	x	x	x

Tabla 3: Pruebas MiniNet.

propuesta. Por lo que no se pudo avanzar con las pruebas sobre EstiNet

NS-3
La máquina virtual (en VirtualBox) para las pruebas sobre NS-3 tiene instalado un sistema operativo Ubuntu 14.04, 3GB de memoria RAM y la versión de NS-3 instalada es la 3.25 con soporte para OpenFlow en su versión 0.8.9. Las pruebas a realizar en NS-3 son limitadas, ya que no soporta un controlador SDN externo. Por este motivo no se podran realizar pruebas sobre las aplicaciones propuestas, por lo tanto se probara unicamente su escalabilidad. Se utilizará un controlador del tipo LearningController.
Para crear las topologías se ejecutan los comandos:
./waf –run "topoN -v"
Donde N indica el número de topología a crear (N = 1..12).
Pruebas :
Cada una de las pruebas se ejecutaron 10 veces y aquí se muestra un promedio del tiempo de esas 10 ejecuciones.

	Crear Topología	STP(Pox-Ryu)	ND(Pox-Ryu)	Eliminar Topología
Topo1	0.08s	x	x	0.01s
Topo2	0.13s	x	x	0.01s
Topo3	0.09s	x	x	0.02s
Topo4	0.10s	x	x	0.00s
Topo5	0.11s	x	x	0.02s
Topo6	0.16s	x	x	0.02s
Topo7	0.34s	x	x	0.04s
Topo8	0.55s	x	x	0.06s
Topo9	0.21s	x	x	0.07s
Topo10	0.95s	x	x	0.05s
Topo11	1.85s	x	x	0.10s
Topo12	4.60s	x	x	0.29s

Tabla 4: Pruebas NS-3.

DOT

Para las pruebas sobre DOT se utilizaran máquinas virtuales para los componentes DOT Manager y DOT nodes, estas máquinas se virtualizaron utilizando VMWare. Se utilizaran los controladores POX y Ryu instalados en el componente DOT Manager. Cada una de las máquinas virtuales cuenta con un sistema operativo Ubuntu 12.04, 512GB de memoria RAM y la versión de DOT instalada es la 1.0, se utiliza la versión 0.2.0 de POX y se utiliza Ryu en su versión 4.3. Los switches emulados son Open vSwitch en su versión 2.1.3(se instalan con la instalación de DOT Node). Las pruebas se realizarán sobre la versión 1.0 del protocolo OpenFlow, ya que es la unica version del protocolo que soporta POX. Para probar la conectividad entre hosts se utiliza el comando pingAll desde la consola de DOT. Los escenarios para las distintas pruebas son sin pérdida de paquetes y un retraso y ancho de banda variable, dependiente de la carga de los equipos.
Para crear las topologías se ejecutan los comandos (en el DOT Manager, dentro del directorio donde se encuentra instalado DOT):
sudo ./run.sh topoN.txt
Donde N indica el número de topología a crear (N = 1..12).
Para levantar los controladores se ejecutan los comandos:

- **POX**
  Spanning tree: sudo ./pox.py forwarding.l2_learning openflow.discovery openflow.spanning_tree openflow.of_01
  Network Discovery: sudo ./pox.py forwarding.l2_learning openflow.discovery openflow.of_01

- **Ryu**
  Spanning tree: sudo ryu-manager –ofp-tcp-listen-port 6633 simple_switch_stp.py
  Network Discovery: sudo ryu-manager –ofp-tcp-listen-port 6633 simple_switch_nd.py

–observe-links

Pruebas :
Al intentar crear las distintas topologías propuestas, DOT genera inconvenientes, la infraestructura física donde se corre la emulación es incapaz de satisfacer las necesidades de recursos para la emulación, por lo que procede a instalar 5 componentes DOT Nodes más, de esta forma se tiene un componente DOT Manager con 512MB de memoria RAM y 6 componentes DOT Nodes con 512MB de memoria RAM, todos ellos virtualizados con VMWare, corriendo sobre un equipo con 4GB de memoria RAM. Esta infraestructura también resulta insuficiente para DOT, por lo que de esta forma no se puede avanzar con las pruebas en las topologías propuestas. Luego se modificaron las topologías, para que no se emulen los hosts y se emulan solamente switches y links, para poder probar las aplicaciones sobre este emulador. Pero esto tampoco resultó satisfactorio, por lo que se dejaron las pruebas en DOT de lado.
Al no resultar satisfactorias las pruebas con las especificaciones anteriores se pasó a utilizar una máquina virtual (en KVM) con 32GB de memoria RAM y un procesador AMD Opteron 23xx (Gen 3 Class Opteron) para emular un DOT Node. De esta forma se intentan emular las topologías propuestas.

KaanalNet
La máquina virtual (en VirtualBox) para las pruebas sobre KaanalNet utilizando los controladores POX y Ryu tiene instalado un sistema operativo Ubuntu 14.04, 3GB de memoria RAM y la versión de KaanalNet instalada es la 0.2.0, se utiliza la versión 0.2.0 de POX y se utiliza Ryu en su versión 4.3. En las pruebas a realizar ambos controladores se encuentran corriendo en la misma máquina donde se ejecuta KaanalNet. Los switches emulados son Open vSwitch en su versión 2.0.2. Las pruebas se realizarán sobre la versión 1.0 del protocolo OpenFlow, ya que es la unica version del protocolo que soporta POX. Para probar la conectividad entre hosts se utilizan los tests que expone KaanalNet. Los escenarios para las distintas pruebas son sin pérdida de paquetes y un retraso y ancho de banda variable, dependiente de la carga de los equipos.
Para crear las topologías se ejecutan los comandos:

Primero se levanta Kaanalnet:
sudo npm start – -S openvswitch -C tcp:0.0.0.0:6633

y luego se crean las topologías:
curl -XPOST -H 'Content-Type:application/json' -H 'Accept: application/json' –data-binary @topoN.json http://localhost:5050/Topology -v -s
Donde N indica el número de topología a crear (N = 1..12).
Para levantar los controladores se ejecutan los comandos:

- **POX**
  Spanning tree: sudo ./pox.py forwarding.l2_learning openflow.discovery openflow.spanning_tree openflow.of_01

Network Discovery: sudo ./pox.py forwarding.l2_learning openflow.discovery openflow.of_01

- **Ryu**
  Spanning tree: sudo ryu-manager –ofp-tcp-listen-port 6633 simple_switch_stp.py
  Network Discovery: sudo ryu-manager –ofp-tcp-listen-port 6633 simple_switch_nd.py –observe-links

Pruebas :
Cada una de las pruebas se ejecutaron 10 veces y aquí se muestra un promedio del tiempo de esas 10 ejecuciones.

Al intentar crear las distintas topologías propuestas, KaanalNet genera inconvenientes, ya que no es capaz de emular las topologias con las capacidades provistas a la maquina virtual que contiene la plataforma. Se proceden a eliminar los hosts de la topologia, de modo de solo emular switches y links. De esta formase logran emular las topologias propuestas y los resultados obtenidos, son solo emulando estos componentes. Al momento de probar las aplicaciones de los controladores en el entorno de emulación, estas no responden adecuadamente. Al ejecutar el comando .ºvs-ofctl show sX"(este comando muestra las propiedades de un switch openflow, donde sX indica el switch openflow a observar.) se observa que todos los puertos de los switches se inician en estado "PORT_DOWN", por tal motivo, por este motivo las aplicaciones de los controladores no responden. Para solucionar este problema se implementó un script por cada topología a emular (up_ports_topoX.sh), el cual levanta los puertos y links de cada uno de los switches openflow emulados. Al no resultar satisfactorias las pruebas con las especificaciones anteriores para emular hosts en la topología, se pasó a utilizar una máquina virtual (en KVM) con 32GB de memoria RAM y un procesador AMD Opteron 23xx (Gen 3 Class Opteron) para emular las topologías con hosts. Con estas especificaciones tampoco es posible emular hosts en la topologia. Por lo que las pruebas de las topologías 10, 11 y 12 no se realizaron.

	Crear Topología	STP(Pox-Ryu)	ND(Pox-Ryu)	Eliminar Topología
Topo1	12s	7s - 49s	8s - 3s	5s
Topo2	62s	12s - 120 s	10s - 3.5s	15s
Topo3	18s	13s - 35s	11s - 3s	4s
Topo4	20s	13s - x	14s - 4s	6s
Topo5	14s	16s - 48s	13s - 4s	4s
Topo6	15s	16s - 48s	13s - 4s	4s
Topo7	18s	18s - x	9s - 3s	6s
Topo8	27s	20s - 44s	10s - 4s	11s
Topo9	106s	21s - 49s	22s - 6s	15s

Tabla 5: Pruebas KaanalNet.

6. Conclusiones

Lo primero que hay que decir es que SDN es una tecnología nueva, muy prometedora que viene a solucionar problemas actuales que ocurren sobre las redes tradicionales. Como ya se mencionó, las arquitecturas de red tradicionales no están optimizadas para satisfacer los requerimientos actuales y futuros de los distintos actores, por lo que con SDN se logra la automatización de la gestión y provisión de la red. En cuanto a herramientas de emulación/simulación SDN, aún no existen gran cantidad de herramientas open source y la mayoría de las que existe cuentan con muy poca documentacion, asi como tambien con muy pocas pruebas sobre ellas, tales son los casos de DOT y KaanalNet, estas herramientas fueron seleccionadas para realizar pruebas sobre distintas topologías y ambas generaron numerosos inconvenientes. Con DOT no se pudo realizar la emulación de ninguna de las topologías planteadas, ya que requería de más infraestructura física, lo cual no parece razonable a la hora de realizar un trabajo de investigación. Se probó con distintas infraestructuras físicas, y ninguna de ellas resultó satisfactoria para las pruebas a realizar. La única documentación que existe sobre este emulador es un tutorial muy escueto en el sitio web oficial, el cual en estos momentos no se encuentra online y su última actualización había sido en octubre del 2014. En cuanto a KaanalNet se lograron emular las topologías, pero solo switches y links, con la máquina en donde se realizaban las pruebas fue imposible emular hosts, por el mecanismo de emulación que utiliza la herramienta. A la hora de utilizar los controladores sobre el ambiente emulado en KaanalNet se generaron gran cantidad de problemas, ya que los puertos de los switches inicialmente están cerrados. KaanalNet es un proyecto de github, y cuenta con muy poca información, su última actualización es de noviembre del 2015. En cuanto a NS-3 (es uno de los pocos simuladores compatible con OpenFlow) las pruebas fueron bastante limitadas ya que no es posible utilizar un controlador externo, igualmente es un simulador muy potente con el que se pueden logran simular topologías muy pesadas en muy poco tiempo. Mientras que EstiNet parece ser una herramienta potente, pero al ser un software propietario, y no contar con licencias no se lograron emular las topologías planteadas ya que la licencia de prueba no tiene capacidades de emular topologías con demasiados componentes. MiniNet es la herramientas más usada a la hora de emular redes definidas por software, cuenta con muy buena documentación y es muy fácil de utilizar, se lograron realizar las pruebas sobre casi todas las topologías, el problema que se dio con MiniNet fue que no pudo emular topologías muy grandes en el equipo físico utilizado para las pruebas (por ejemplo las topologías 11 y 12), por lo que se paso a utilizar la infraestructura de la facultad para tratar de emular las topologías restantes. Con esta infraestructura tampoco se pudieron terminar de emular las dos topologías restantes. Igualmente las pruebas en MiniNet fueron satisfactorias, y se lograron emular diversas topologías en muy poco tiempo y de manera muy simple. En definitiva aún queda mucha para avanzar en cuanto a herramientas para simulación/emulación, y en la actualidad la más recomendable a usar es MiniNet.

7. Anexos

Se crea un proyecto en GitHub, donde se encuentran los desarrollos de todas las topologías emuladas/simuladas, así como también las aplicaciones de los controladores utilizados. También se anexan los scripts encargados de levantar los puertos de los switches openflow para el entorno de emulación kaanalnet [44].

Referencias

[1] Joe Silver, *SDN 101: What It Is, Why It Matters, and How to Do It.* Cisco Blogs, 2013.

[2] thegreek80, "Investigación de redes sdn." http://principiatechnologica.com/category/principia-technologica/redes/sdn, 2014.

[3] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 1–12, Aug. 2007.

[4] E. Z. Nick Feamster, Jennifer Rexford, "The Road to SDN: An Intellectual History of Programmable Networks," 2015.

[5] Ramón Jesús Millán Tejedor, *SDN: el futuro de las redes inteligentes.* Conectrónica nº 179, GM2 Publicaciones Técnicas, 2014.

[6] Open Networking Foundation, *OpenFlow Switch Specification.* Open Networking Fundation, 2014.

[7] Flowgrammable, *OpemFlow Message Layer.* Flowgrammable, 2014.

[8] Nicira Networks, Stanford University, International Computer Science Institute, UC Berkeley, "Nox." http://www.noxrepo.org/. [Sitio web oficial].

[9] Erickson, David, "The Beacon OpenFlow Controller," in *HotSDN*, ACM, 2013.

[10] Project Floodlight, "Floodlight." http://www.projectfloodlight.org/floodlight/. [Sitio web oficial].

[11] Trema, "Trema." https://trema.github.io/trema/. [Sitio web oficial].

[12] POX, "POX." http://www.noxrepo.org/pox/. [Sitio web oficial].

[13] Gary Berger, "NodeFlow." https://github.com/gaberger/NodeFLow. [Sitio web oficial].

[14] travelping, "FlowER." http://travelping.github.io/flower/. [Sitio web oficial].

[15] Zheng Cai (Rice), "Maestro." http://zhengcai.github.io/maestro-platform/. [Sitio web oficial].

[16] Ryu SDN Framework Community, "Ryu." https://osrg.github.io/ryu/. [Sitio web oficial].

[17] Linux Fundation, "OpenDayLight." https://www.opendaylight.org/lithium. [Sitio web oficial].

[18] CPqD(Brazil), "RouteFlow." https://sites.google.com/site/routeflow/. [Sitio web oficial].

[19] Alejandro García Centeno, Carlos Manuel Rodríguez Vergel, Caridad Anías Calderón, Frank Camilo Casmartiño Bondarenko, *Controladores SDN, elementos para su selección y evaluación.* Revista Telem@tica. Vol. 13. No. 3, 2014.

[20] NSRC, InCNTRE, "Introducción a OpenFlow." https://www.nsrc.org/workshops/2015/walc/routing/raw-attachment/wiki/Agenda/15-Introducción-SDN-Openflow.pdf.

[21] B. Heller, "OpenFlowTutorial_ONS_Heller.pdf." https://wangxliang.wikispaces.com/file/view/OpenFlowTutorial_ONS_Heller.pdf.

[22] "MiniNet." http://mininet.org/. [Sitio web oficial].

[23] Te-Yuan Huang, Vimalkumar Jeyakumar Bob Lantz, Brian O'Connor Nick Feamster Keith Winstein, Anirudh Sivaraman, "Teaching computer networking with mininet," *ACM SIGCOMM 2014*.

[24] "EstiNet." http://www.estinet.com/ns/. [Sitio web oficial].

[25] Shie-Yuan Wang, National Chiao Tung University, Chih-Liang Chou and Chun-Ming Yang, EstiNet Technologies, Inc., *EstiNet OpenFlow Network Simulator and Emulator.* IEEE Communications Magazine (Volume:51 , Issue: 9), 2013.

[26] "NS-3." http://ns3simulation.com/. [Sitio web oficial].

[27] Mathieu Lacage, "Experimentation with ns-3." http://www.slideshare.net/mathieu$_l acage/ns3 - tutorial$.

[28] "OFSWITCH13." http://www.lrc.ic.unicamp.br/ofswitch13/.

[29] "DOT." http://dothub.org/. [Sitio web oficial].

[30] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba, "DOT: Distributed OpenFlow Testbed," in *Proceedings of the ACM SIGCOMM 2014 Conference on SIGCOMM*, August 2014.

[31] "KaanalNet." https://github.com/sureshkvl/kaanalnet. [Sitio web oficial].

[32] "LXC." https://linuxcontainers.org/. [Sitio web oficial].

[33] "OMNet++." https://omnetpp.org/. [Sitio web oficial].

[34] Thomas Dietz, *Trema Tutorial.* 2012.

[35] "VNX." http://web.dit.upm.es/vnxwiki/index.php/Main$_P age.[Sitiowebof icial]$.

[36] "MaxiNet." http://maxinet.github.io/. [Sitio web oficial].

[37] "Instalación NS-3." https://www.nsnam.org/wiki/Installation.

[38] "Instalación DOT." http://dothub.org/installation/.

[39] "Instalación DOT." https://github.com/sureshkvl/kaanalnet/wiki/Installation.

[40] "Componentes POX." https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-ComponentsinPOX.

[41] "OVS." http://openvswitch.org/. [Sitio web oficial].

[42] "Pica8." http://www.pica8.org/solutions/new-software.php. [Sitio web oficial].

[43] "Indigo." https://floodlight.atlassian.net/wiki. [Sitio web oficial].

[44] "Anexos." https://github.com/elchobo5/licenciaturaSDN.

[45] C.-L. C. Shie-Yuan Wang, Hung-Wei Chiu, "Comparisons of SDN OpenFlow Controllers over EstiNet: Ryu vs. NOX," ICN 2015 : The Fourteenth International Conference on Networks, 2015.

[46] E. B. Y. Mohammed Basheer Al-Somaidai, "Survey of Software Components to Emulate OpenFlow Protocol as an SDN Implementation," American Journal of Software Engineering and Applications. Vol. 3, No. 6, 2014.

[47] L. J. G. V. Lorena Isabel Barona López, Ángel Leonardo Valdivieso Caraguay, "Extending OpenFlow in Virtual Networks," ICIT 2015 The 7th International Conference on Information Technology, 2015.

[48] P. B. Mukta Gupta, Joel Sommers, "Fast, Accurate Simulation for SDN Prototyping,"

[49] ryu development team, "ryu Documentation." https://media.readthedocs.org/pdf/ryu/latest/ryu.pdf.

[50] Martin Casado, "Software SDN." http://yuba.stanford.edu/ casado/of-sw.html.