UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Evaluación de Tecnología SDN
Autores: Marcelo Barreto
Director: Lic. Paula Venosa, Lic. Andres Barbieri
Codirector: -
Asesor profesional: -
Carrera: Licenciatura en Sistemas

Resumen

Las redes de computadoras actuales son complejas y difíciles de administrar, y están compuestas de una amplia variedad de equipamiento. Los routers y switches ejecutan software de control complejo y distribuído que generalmente es propietario y cerrado, encargado de implementar protocolos de red que requieren años de estandarización y pruebas de interoperabilidad.

Para que estas puedan operar los administradores de red deben configurar cada uno de los dispositivos utilizando interfaces de configuración, que varían según el vendedor del dispositivo e incluso entre productos del mismo vendedor.

Como consecuencia la innovación en el ámbito de las redes ha sido lenta, y ha aumentado la complejidad de su operación.

El objetivo del presente trabajo es realizar un análisis de las características y potencial de la tecnología SDN, aplicada a la administración y mantenimiento de redes de datos. Y realizar un prototipo funcional de una interfaz de usuario, capaz de mostrar la topología de una red y permitir realizar cambios en su funcionamiento de forma centralizada. Finalizando con la resolución de algunos problemas típicos en las redes a través del uso de esta tecnología.

Palabras Claves

Administración centralizada de redes , Arquitectura de Red, Frenetic, Link Layer Discovery Protocol, NetKat, OpenFlow, Plano de Aplicación, Plano de Control, Plano de Datos, Políticas de red, SDN, Software Defined Networks, Tabla de Flujos, Topología de Red.

Conclusiones

La tecnología SDN, incorpora un nuevo enfoque que brinda grandes oportunidades para la resolución de problemas típicos en el diseño, operatoria y mantenimiento en las redes de datos, favorece la innovación y permite que las redes de datos puedan implementarse a través de la composición de aplicaciones que resuelvan subconjuntos de problemas específicos.

Trabajos Realizados

Se desarrolló un producto que permite mantener y visualizar la topología de una red de datos, y la utilización de un lenguaje de alto nivel para especificar las funcionalidades y comportamientos que esta debe brindar.

Este producto fue desarrollado respetando los principios arquitectónicos definidos por la tecnología SDN.

Para finalizar se desarrollaron ejemplos de su utilización en problemas típicos de las redes de datos realizando una evaluación comparativa respecto de las soluciones normalmente utilizadas con el enfoque tradicional.

Trabajos Futuros

Se proponen las siguientes líneas de investigación y trabajo futuro: a) La implementación del descubrimiento y mantenimiento de los dispositivos finales a la topología de red. b) La implementación de diferentes algoritmos de grafos sobre la topología con el objetivo de ser la entrada de información para realizar tareas de Ingeniería de tráfico. c) La implementación de interfaces que permitan extender y relacionar los datos de la topología con herramientas externas como inventario y monitoreo. d) La investigación sobre la utilización de arquitecturas híbridas.

Fecha de la presentación: Febrero de 2017

EVALUACIÓN DE TECNOLOGÍA SDN

MARCELO BARRETO

UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TRABAJO DE TESINA DE GRADO

DIRECTORES
Lic. Paula Venosa
Lic. Andres Barbieri

La Plata, Febrero 2017

Quiero dedicar especialmente este trabajo a:

A mi esposa que me ha acompañado desde el inicio de mis estudios,
y me ha regalado dos hijas maravillosas.

A mi hija Sofia, que pese a su corta edad, ha tenido
la paciencia de entenderme y acompañarme.
A mi hija Lucia, que dando sus primeros pasos me
ha dado la fortaleza para concluir este ciclo.

Y a mis padres, que desde pequeño me inculcaron el
valor del esfuerzo y sacrificio.

Quien hace una pregunta es ignorante cinco minutos;
pero quien no la hace lo será toda la vida.

AGRADECIMIENTOS

Gracias a todos los profesores que me transmitieron sus conocimientos durante el transcurso de mi carrera.

A Nico y Ale que que fueron determinantes en la elección del tema de mi trabajo.

A mis directores Paula y Andres que me apoyaron y acompañaron desde la propuesta hasta la culminación de mi trabajo, y que sin su ayuda no hubiese podido concluir.

A Gustavo y Silvia, que además de haberme acompañado durante toda mi formación académica, me han dado la oportunidad de formar parte del laboratorio que es parte muy importante de mi vida.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

Parte I

INTRODUCCIÓN

1

ORGANIZACIÓN DEL TABAJO

MOTIVACIÓN

Las redes de computadoras actuales son complejas y difíciles de administrar, están compuestas de una amplia variedad de equipamiento, desde routers y switches a dispositivos intermedios como firewalls, balanceadores de carga y sistemas de detección de intrusos.

Los routers y switches ejecutan software de control complejo y distribuído que generalmente es propietario y cerrado, encargado de implementar protocolos de red que requieren años de estandarización y pruebas de interoperabilidad.

Para que estas puedan operar los administradores de red deben configurar cada uno de los dispositivos utilizando interfaces de configuración que varían según el vendedor del dispositivo e incluso entre productos del mismo vendedor.

Como consecuencia la innovación en el ámbito de las redes ha sido lenta, ha aumentado la complejidad de su operación, y los costos de capital y operación.

OBJETIVO

El objetivo principal del presente trabajo de grado es realizar un análisis de las características y potencial de la tecnología emergente conocida como SDN "Software Defined Networks", aplicada a la administración y mantenimiento de redes de datos.

Como objetivo secundario se realizará un prototipo funcional de una interfaz de usuario capaz de mostrar la topología de una red

SDN y permitir realizar cambios en su funcionamiento de forma centralizada.

Por último se plantean diversos requerimientos en las redes de datos modernos y cómo la tecnología SDN puede ayudar a resolverlos.

ESTRUCTURA DEL TRABAJO

En el capítulo 2 se realiza un análisis de la arquitectura utilizada en las redes de datos tradicionales y sus limitaciones en función de su evolución y sus actuales requerimientos.

En el capítulo 3 se presenta un análisis de la arquitectura planteada por SDN, su evolución histórica y las necesidades que impulsaron su desarrollo.

En el capítulo 4 se realiza la descripción detallada de la especificación OpenFlow en su versión 1.0.

En el capítulo 5 se describen las herramientas y frameworks utilizados en el trabajo.

En el capítulo 6 se plantean la motivación y requerimientos de tres problemas típicos a resolver en las redes de datos que serán utilizados para la comparación de su resolución tradicional versus su resolución a través de la arquitectura SDN.

En el capítulo 7 se detalla la arquitectura SDN desarrollada, y se realiza la comparación entre las soluciones tradicionales a los problemas del capítulo 6 y la solución implementada utilizando la arquitectura SDN desarrollada.

Para terminar en el capítulo 8 se desarrollan las conclusiones y se presentan las posibilidades de trabajo futuro.

Parte II

MARCO TEÓRICO

2

ARQUITECTURA TRADICIONAL DE LAS REDES

EVOLUCIÓN Y FUNCIONAMIENTO

En los años 50's, durante la era de la guerra fría, Paul Baran, un investigador de la corporación RAND fundada por el Departamento de Defensa de los Estados Unidos, notó que el método de transporte utilizado en las redes telefónicas no era lo suficientemente robusto como para tolerar un ataque militar, y un solo elemento comprometido podía dejar sin servicio a una parte importante de la red. A partir de la investigación de este hecho, sugirió que debía dotarse de autonomía a todos los elementos de la red, de forma tal que ante un fallo en alguno el resto pudiese utilizar la información de adyacencia para configurar caminos alternativos y recuperarse.

Este enfoque permitiría evitar la necesidad de la especificación estática de los caminos necesarios para interconectar dos elementos de la red.

Un tiempo después a finales de los 60's, Donald Davies, en el Laboratorio Nacional de Física del Reino Unido, utilizando estos conceptos formuló las bases de las comunicaciones modernas publicando los conceptos de la comunicación de intercambio de paquetes (packet switching).

La fusión de ambos conceptos sentaron las bases de las actuales redes de datos IP, y con el surgimiento del proyecto DARPA el nacimiento de Internet.

Hoy en día Internet está formada por un conjunto de redes de borde (edge networks) conocidas como sistemas autónomos (AS). Cada uno de estos sistemas autónomos posee sus propias políticas de fun-

cionamiento y acuerdos para la interconexión y comunicación con otros sistemas autónomos.

La infraestructura de estos AS está compuesta por un conjunto de dispositivos de red que permiten las comunicaciones dentro de él y con dispositivos de otros sistemas autónomos. Los principios de diseño de estos dispositivos de red descienden de las investigaciones de Paul Baran y Donald Davies, y se basan en garantizar la comunicación teniendo en cuenta la tolerancia a fallos en la red.

A fin de comprender y analizar las funcionalidades de estos dispositivos podemos abstraer sus funcionalidades y clasificarlas en tres capas o planos: plano de datos o forwarding, el plano de control y el plano de administración o management.

El plano de datos o forwarding implementa las funcionalidades encargadas de transportar los datos en la red.

El plano de control es el encargado de generar la topología de red para poder establecer los caminos necesarios, a fin de realizar el transporte de los datos entre dos dispositivos de la red y actualizar el plano de datos o forwarding para reflejar estas decisiones.

El plano de administración o management permite a los operadores de la red realizar las configuraciones necesarias para que los dispositivos funcionen adecuadamente y consultar la información necesaria para su gestión, auditoría y la resolución de problemas.

En función de los principios de diseño mencionados anteriormente, estas capas o planos se encuentran implementadas en cada uno de los dispositivos asegurando su independencia y autonomía en caso de un fallo en la red. Para generar la información necesaria para transportar los datos a través de la red, estos dispositivos ejecutan protocolos y algoritmos que se encargan de construir en cada dispositivo una topología, y a partir de esta determinan los caminos necesarios para entregar los datos.

Para descubrir los elementos en la red y mantener la topología, los dispositivos intercambian información a través de varios protocolos,

algunos ejemplos son Link Layer Discovery Protocol (LLDP) o Cisco Discovery Protocol (CDP).

Para evitar posibles bucles de comunicación, se utilizan protocolos que bloquean la comunicación en alguno de sus puertos como Spanning Tree Protocol (STP) o Shortest Path Bridging (SPB).

Para permitir la comunicación entre diferentes redes IP, dentro del AS, los dispositivos ejecutan protocolos de ruteo interiores (IGP), como OSPF o IS-IS, que construyen la topología intercambiando información del estado de sus enlaces con sus dispositivos adyacentes, y generando a partir de esta la información de reenvío necesaria para transportar los datos.

Los routers, que se encuentran en los bordes de los sistemas autónomos, utilizan protocolos de ruteo exteriores (EGP) como Border Gateway Protocol (BGP), para transportar el tráfico de diferentes AS respetando las políticas establecidas dentro del AS y sus acuerdos comerciales con otros.

Aunque, tanto las funcionalidades del plano de datos y el plano de control, se encuentran ambas dentro de cada dispositivo, a nivel de hardware se encuentran separadas y sus funciones se llevan a cabo por unidades de procesamiento diferentes.

En general, para implementar el plano de datos se utilizan circuitos integrados específicos de aplicación (ASIC), capaces de procesar y reenviar paquetes de datos a grandes velocidades. En cambio en el plano de control, la implementación generalmente es a través de CPUs de propósito general capaces de procesar los paquetes de control o aquellos que el plano de datos no sabe cómo manejar.

Además de estas funcionalidades, las redes de datos actuales incorporan otras tecnologías y dispositivos intermedios conocidos como middleboxes, que poseen características y funcionalidades que los dispositivos estándar de red no contienen. Algunos ejemplos son: firewalls para implementar políticas de seguridad más avanzadas, balanceadores de carga que permiten distribuir la carga en los enlaces de red y los servidores de aplicación, sistemas de detección de intru-

sos o IDS que ofrecen funcionalidades para prevenir o mitigar ataques de red.

En la Figura 2.1 se puede ver la conformación de un AS, sus dispositivos y su conexión con otros AS, y también puede observarse como cada uno de los dispositivos posee su plano de datos y control. A través del plano de administración, se establecen y consultan las configuraciones de ruteo, vlans, controles de acceso y otras necesarias para implementar las políticas del AS. [1, 2]

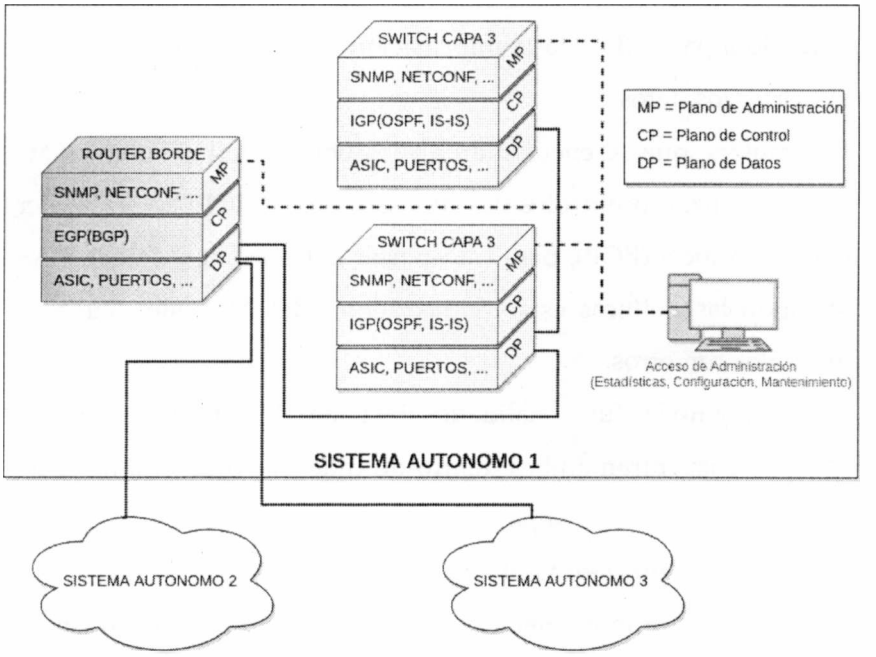

Figura 2.1: Descripción arquitectura tradicional

LIMITACIONES DE LA ARQUITECTURA

Aunque en principio este enfoque arquitectónico parece adecuado para la implementación y mantenimiento de las redes de datos y es el utilizado en la mayoría de las redes, con la evolución de la tecnología y sus nuevos requerimientos se comienzan a notar sus debilidades para cumplimentar de forma eficiente las funcionalidades hoy requeridas.

Entre sus debilidades podemos mencionar la complejidad para la implementación de políticas no tradicionales como la calidad de servicio (QoS), el enrutamiento basado en ingeniería de tráfico, la necesidad por parte de los operadores de red de realizar configuraciones de bajo nivel en cada uno de los dispositivos y la propensión a errores que esto conlleva, entre otras.

Como ejemplo podemos utilizar una aplicación bancaria que requiere que su tráfico sea enrutado a través de un camino seguro, mientras que una aplicación de videoconferencia requiere que su tráfico sea enrutado por los enlaces con el mayor ancho de banda disponible. La implementación de esta política de comunicación es compleja y requiere que los operadores configuren cada uno de los dispositivos que intervienen en la comunicación de forma individual y sin herramientas de alto nivel que le ayuden a verificar los efectos secundarios que pueden producirse.

El diagnóstico y solución de errores no escapa a estas limitaciones dado que los operadores de red deben utilizar el mismo conjunto de herramientas de bajo nivel para establecer y corregir los problemas de comunicación.

Un aspecto adicional derivado de la evolución y enfoque arquitectónico tradicional es que los dispositivos de red implementan de diversas formas las funcionalidades del plano de datos, control y administración generando que dispositivos de diferentes vendedores e incluso del mismo vendedor, soporten diferentes interfaces de configuración, diferentes niveles de soporte de protocolos de enrutamiento y diferentes funcionalidades en la capa de datos, esto implica la necesidad de tener recursos humanos altamente capacitados para las tecnologías de los diversos vendedores, y en muchos casos obliga a reemplazar o agregar nuevos dispositivos para sortear las limitaciones de los dispositivos instalados.

Como ejemplo para prevenir el ataque a un servidor desde un dispositivo específico, el operador debe identificar la IP o MAC del atacante y configurar las listas de control de acceso necesarias en cada

uno de los dispositivos que dirigen tráfico al servidor y es probable que en cada uno de estos dispositivos deba hacerlo a través de diferentes interfaces y sintaxis.

CONCLUSIÓN

El enfoque arquitectónico tradicional no es el más adecuado para abordar los nuevos requerimientos y funcionalidades en el campo de las tecnologías de comunicación. La complejidad generada por la evolución de los requerimientos en las comunicaciones no ha sido acompañada por una evolución en su enfoque arquitectónico y la implementación y operatoria requieren de la intervención y trabajo de un conjunto de recursos humanos. Además, la implementación de herramientas externas específicamente diseñadas para trabajar con dispositivos de un vendedor no favorecen la innovación ni la heterogeneidad en las redes de datos. Esta complejidad también trae aparejado un aumento los costos de implementación y mantenimiento debido a la necesidad de alta especialización y cantidad de recursos humanos junto a los costos de hardware asociados a los dispositivos de red.

Por lo tanto se debe rediseñar la arquitectura de modo de favorecer la administración y operación de las redes de datos teniendo en cuenta la escalabilidad, los actuales requerimientos, y la innovación para favorecer el desarrollo de mejoras y la adecuación a futuros cambios en los requerimientos de la red.

3

SOFTWARE DEFINED NETWORKS

HACIA EL CONCEPTO DE SDN

El concepto de redes programables no es un concepto nuevo, de hecho el concepto ha ido evolucionando desde hace más de 20 años.

El primer emprendimiento en este sentido fue el proyecto conocido como Active Networking. A diferencia de las redes tradicionales que son agnósticas al contenido de los paquetes Active Networking no lo era, los dispositivos podían ser configurados con paquetes especiales capaces de alterar su comportamiento. Aunque este proyecto no tuvo una adopción importante, fue el primero en proponer el concepto de "programabilidad" de la red.

Varios años después a mediados de los 2000s, se puso el foco en la separación del plano de datos y el plano de control, lo cual es uno de los conceptos centrales para la simplificación de la arquitectura. El proyecto pionero en poner a prueba este concepto fue el Forwarding and Control Element Separation (ForCES).

En ForCES los componentes de la red se clasificaban en dos tipos. Los Fordwaring Elements son componentes cuya única función es re-enviar y filtrar tráfico. Los Control Elements son los componentes encargados de definir cómo son procesados los paquetes. Ambos tipos de elementos están conectados a través de una interfaz abierta estandarizada.

Utilizando los principios de diseño establecidos por ForCES surgió un proyecto conocido como SANE que fue extendido por ETHANE.

En ETHANE existe un controlador encargado de comunicarse con los switches para actualizar sus tablas de flujos de acuerdo a las políticas establecidas por los operadores de red. A diferencia de el ruteo

basado en tablas, el ruteo basado en flujos no utiliza solo las direcciones IP o MAC, sino que utiliza los campos de las cabeceras de los paquetes para encontrar el puerto de destino. [2, 3]

OpenFlow [4] fue la evolución de los conceptos de ETHANE, diseñado inicialmente para favorecer la innovación y reducir los costos de las redes.

El éxito de OpenFlow en la industria fue posible gracias a la formación de la Open Networking Foundation (ONF) en el año 2011. Para el año 2015 la fundación estaba compuesta por más de 150 miembros de la industria, y tuvo implementaciones en dispositivos de vendedores como Cisco, Dell, Brocade y HP.

Tal fue su éxito que de hecho el término Software Defined Networking comenzó a asociarse como sinónimo del proyecto OpenFlow.

ARQUITECTURA SDN

Algunos autores refieren el término SDN como la capacidad de programar dispositivos de red de forma dinámica y así controlar su funcionamiento como un todo.

Otros autores la definen como un conjunto de técnicas útiles para facilitar el diseño, puesta en producción y operación de servicios de red de manera dinámica, determinista y escalable.

En la Figura 3.1 pueden observarse las capas de la arquitectura tradicional mientras que en la Figura 3.2 las de la arquitectura SDN.

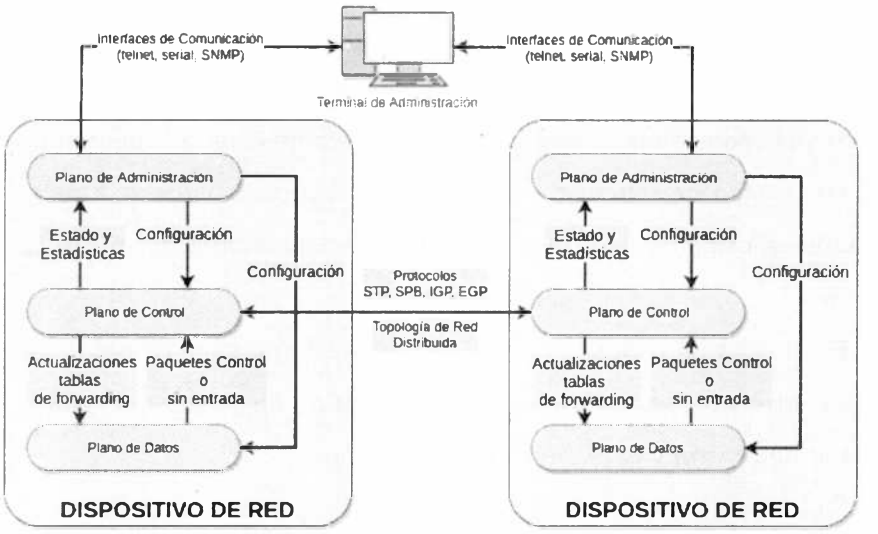

Figura 3.1: Capas arquitectura tradicional

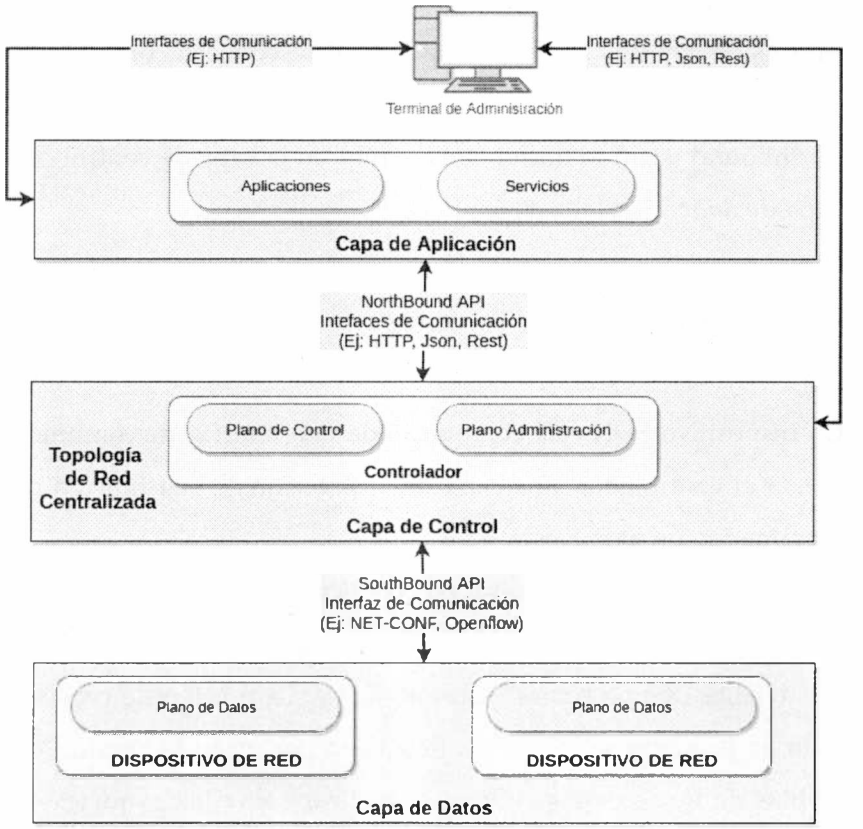

Figura 3.2: Capas Arquitectura SDN

Como puede observarse, en la arquitectura SDN las capas o planos de datos, control y administración se encuentran desacoplados y sus funciones se encuentran implementadas por diferentes dispositivos; a su vez aparece una nueva capa conocida como capa de aplicación.

En la capa de datos se encuentran los dispositivos de red encargados únicamente de implementar las funcionalidades del plano de datos.

En la capa de control aparece un nuevo componente conocido como controlador encargado de implementar de forma centralizada el plano de control y el plano de administración.

En la capa de aplicación se encuentran las programas de software que generan o consumen la información de la capa de control y permiten implementar distintas funcionalidades de red, un ejemplo es un software para realizar balanceo de carga.

La comunicación entre las distintas capas se realiza a través de interfaces de comunicación que en general se conocen como SouthBound (sentido sur) que comunican el plano de datos y el de control y NorthBound (sentido norte) que comunican la capa de control con la capa de aplicación.[2, 3, 5, 6, 7]

Dispositivos SDN

Un dispositivo SDN está compuesto de una interfaz de comunicación con el controlador, una capa de abstracción, y una función de procesamiento de paquetes.

La interfaz de comunicación con el controlador implementa las formas válidas de comunicación entre ellos. La capa de abstracción oculta los detalles de una o más tablas de flujos. La función de procesamiento de paquetes se basa en la búsqueda por prioridad dentro de las tablas de flujos de las acciones a realizar sobre los paquetes de datos.

Al entrar un paquete al dispositivo se realiza una búsqueda en sus tablas de flujos, se selecciona la entrada que coincide con mayor prioridad y se realizan las acciones que se especifican en la tabla de flujos de forma local en el dispositivo.

En caso de no encontrar una entrada para procesar el paquete, este puede ser copiado al controlador para que esté realice las acciones necesarias.

En la Figura 3.3 puede observarse la anatomía de un dispositivo SDN.

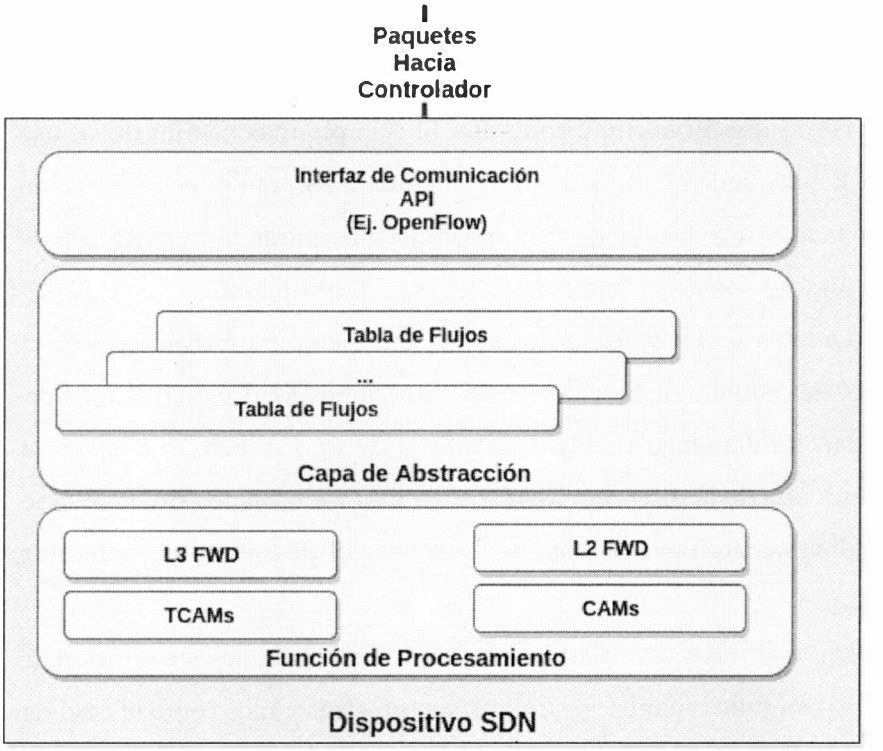

Figura 3.3: Anatomía de un dispositivo SDN

Función de Procesamiento

Los dispositivos de red actuales utilizan hardware dedicado y especialmente diseñado para la inspección de los paquetes entrantes y las acciones a realizar sobre estos.

Como puede verse en la figura este hardware incluye las tablas de forwarding de capa 2 (L2 FWD) y de capa 3 (L3 FWD), éstas gene-

ralmente están implementadas a través de memorias CAMs (content addressable memories) y memorias TCAMs (ternary content addressable memories).

La tabla de forwarding de capa 3 se utiliza para tomar las decisiones de ruteo a nivel IP, esta es la capacidad fundamental de los dispositivos de tipo router, basándose en la búsqueda de la dirección IP de destino y realizando las acciones necesarias en función del resultado (ej: enviar el paquete por la interfaz fe01).

Las memorias TCAM utilizadas para implementar esta funcionalidad, son memorias más complejas que las CAMs y permiten realizar la búsqueda utilizando además de la dirección IP de destino, una máscara que sirve como comodín. El ejemplo más común de su uso es la búsqueda de una dirección IP y una máscara de red, en función de la máscara pueden encontrarse múltiples entradas y en función de la lógica se selecciona la coincidencia más apropiada.

La tabla de forwarding de capa 2 se utiliza para tomar las decisiones de reenvío en función de las direcciones MAC, esta es la capacidad fundamental de los dispositivos de tipo switch, se basa en la búsqueda de la dirección MAC de destino y en función del resultado realiza las acciones necesarias (ej: enviar el paquete por la interfaz 15).

Las memorias CAM utilizadas para implementar esta funcionalidad, son útiles cuando los índices son precisos y fijos como el caso de las direcciones MAC de 48 bits.

Tablas de Flujos

Las tablas de flujos son las estructuras de datos fundamentales de los dispositivos SDN. Permiten que el dispositivo pueda evaluar los paquetes entrantes y realice las acciones necesarias basadas en el contenido del paquete.

En las redes de datos tradicionales esta evaluación se realiza basada en ciertos campos y como resultado de su evaluación en general

se toma una de las siguientes acciones: reenviar el paquete a través de un puerto específico, descartar el paquete, o reenviarlo por un conjunto de puertos (flooding).

Un dispositivo SDN no es muy diferente excepto por que estas operaciones básicas se han hecho más genéricas y más programables a través de las tablas de flujos y su lógica asociada.

Estas tablas de flujos están compuestas por un conjunto de entradas cada una con una prioridad, compuestas de dos componentes: criterios de coincidencia y acciones.

Los criterios de coincidencia se utilizan para evaluar los paquetes entrantes se evalúan en orden de prioridad y el primero en coincidir se selecciona, a partir de esta coincidencia se realizan las acciones sobre el paquete especificadas por la entrada de la tabla de flujos.

Controlador

Como describimos anteriormente el controlador mantiene la vista general de toda la red, implementa las políticas, controla todos los dispositivos SDN que componen la infraestructura y provee de una interfaz de comunicación (NorthBound API) a la capa de aplicación.

Cuando nos referimos a que implementa las políticas, estas pueden referirse a: políticas de ruteo, forwarding, redirección, balanceo de carga, y muchas más, algunos autores ubican dentro de la funcionalidad del controlador sólo las políticas de forwarding y ruteo dejando el resto de las funcionalidades a la capa de aplicación, pero lamentablemente todavía no existe un consenso ni estándar respecto a las funcionalidades dentro del controlador ni la capa de aplicación.

En general, el software de los controladores actuales implementa ciertos módulos básicos como la funcionalidad de switching y routing estático básico, dejando el resto de la funcionalidad para que sea implementada en la capa de aplicación.

Respecto a las interfaces de comunicación entre el controlador y los dispositivos SDN, hoy en día existen varios estándares como OpenFlow que las definen. Desafortunadamente las interfaces de comunicación entre el controlador y la capa de aplicación no los poseen y se considera una de las deficiencias actuales de SDN, aunque se están desarrollando propuestas para estandarizarla.

En conclusión en la arquitectura SDN el controlador es responsable de abstraer los detalles de los dispositivos SDN y la manera de comunicarse con ellos brindando una interfaz de comunicación no estandarizada a las aplicaciones para que estas implementen las funcionalidades de red para las que fueron desarrolladas.

Podemos sumarizar las características que un controlador SDN debería brindar como:

- Descubrimiento de los dispositivos de usuario como laptops, computadoras de escritorio, dispositivos móviles y otros.
- Descubrimiento de los dispositivos de red: encontrar los dispositivos de red que conforman la infraestructura de la misma como switches, routers, access points y otros.
- Mantenimiento y administración de la topología de red: mantener toda la información respecto de los detalles de interconexión entre los dispositivos de red y también los dispositivos de usuario.
- Administración de los flujos: mantener la base de datos de los flujos de paquetes, coordinando y sincronizando las entradas de las tablas de flujo de los dispositivos SDN.
- Mantener la información estadística necesaria para la búsqueda y resolución de errores y la generación de información de auditoría.

Capa de Aplicación

Las aplicaciones SDN se ejecutan en una capa superior al controlador, comunicándose con este a través de una interfaz de comunicación.

A través de esta interfaz las aplicaciones son capaces, entre otras cosas, de:

- Configurar los flujos para enrutar paquetes a través del mejor camino entre dos dispositivos.
- Balancear el tráfico a través de múltiples caminos.
- Reaccionar a los cambios en la topología como errores en un enlace o aparición de nuevos dispositivos y caminos.
- Redirigir tráfico con propósitos de inspección, autenticación o tareas de seguridad similares.

Es importante notar que estas capacidades son posibles debido al desacoplamiento de funcionalidades que permite la arquitectura SDN. Aunque existen herramientas y middleboxes en la arquitectura tradicional que brindan estas funcionalidades, no son lo suficientemente extensibles ni portables entre distintos vendedores, y tampoco permiten que las funcionalidades pueden irse agregando de forma gradual y en base a las necesidades de cada organización, debido a las políticas de ventas de los fabricantes.

Por ello la arquitectura SDN, teóricamente, permite a una organización implementar la funcionalidad mínima requerida para su operación e ir agregando funcionalidad a medida que es necesario, ya sea a través de la adquisición de módulos de software de propósito general de diversos vendedores como a través del desarrollo de aplicaciones de propósito específico para implementar las funcionalidades requeridas por la organización.

4

LA ESPECIFICACIÓN OPENFLOW

INTRODUCCIÓN

Aunque muchos autores y bibliografía [2, 3, 4] utilizan los términos OpenFlow y SDN como sinónimos, no lo son. Como vimos OpenFlow es un subconjunto de las tecnologías incluidas en SDN.

La especificación sólo describe los requerimientos que debe cumplir un dispositivo y el protocolo de comunicación entre éste y el controlador. Y no establece nada respecto al controlador ni a la capa de aplicación.

La especificación de OpenFlow ha evolucionado a través de los años, comenzando en su versión 1.0 [4] en diciembre de 2009, y encontrándose en su versión 1.5.0 [8] de diciembre de 2014.

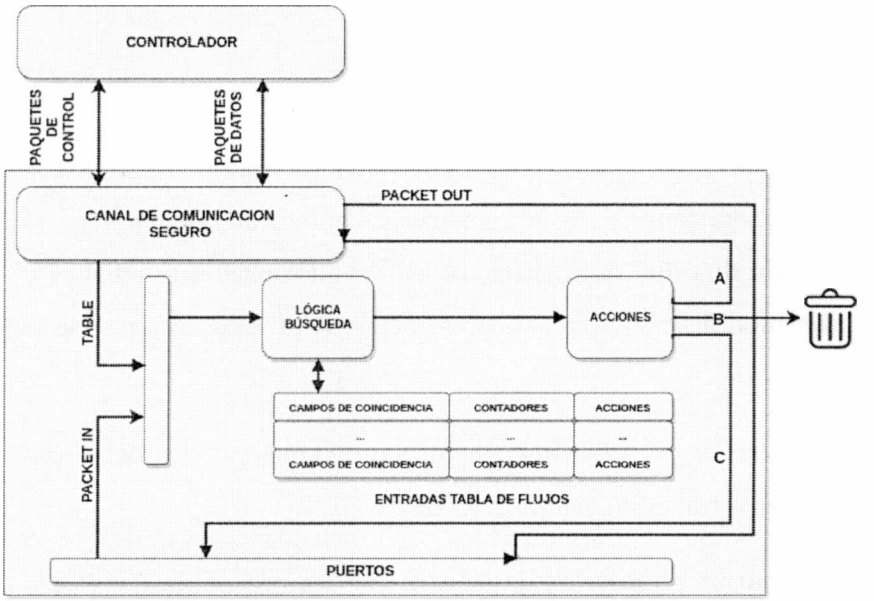

Figura 4.1: Dispositivo OpenFlow

La función principal de un dispositivo es procesar los paquetes entrantes y enviarlos a uno más puertos realizando las modificaciones que sean necesarias al paquete original.

Un dispositivo está compuesto por un conjunto de puertos por los que recibe y envía paquetes. Al ingresar un paquete por alguno de sus puertos a través del mensaje PACKET IN el dispositivo realiza una búsqueda sobre las entradas de su tabla de flujos utilizando las cabeceras del paquete y el identificador del puerto de ingreso y los compara con los valores de los campos de coincidencia establecidos en sus entradas.

Selecciona la primera entrada que genera una coincidencia y ejecuta las acciones especificadas en el campo acciones de la entrada.

Estas acciones son realizadas por el dispositivo sin la necesidad de la intervención del controlador. Las acciones básicas que puede realizar el dispositivo son:

- Enviar el paquete al controlador a través del canal de comunicación (A).
- Descartar el paquete (B).
- Modificar el paquete y re-enviarlo a uno o más puertos (C).

Por su parte el controlador a través del canal de comunicación puede realizar operaciones para administrar y configurar las entradas de la tabla de flujo del dispositivo, consultar información estadística y realizar modificaciones en la configuración del dispositivo. Y tiene la capacidad de:

- Enviar un paquete directamente a uno o más puertos del dispositivo a través de la acción PACKET OUT.
- Enviar un paquete de datos para que sea procesado por el dispositivo a través del puerto virtual TABLE.

LA TABLA DE FLUJOS

La tabla de flujos es uno de los componentes más importantes de un dispositivo OpenFlow, en ella se establecen las acciones a realizar sobre los paquetes de datos, en función de los flujos establecidos en el dispositivo.

Un flujo es una abstracción que permite agrupar paquetes que poseen características similares para luego procesarlos uniformemente. Su definición en OpenFlow es a través de la utilización del puerto de ingreso del paquete al dispositivo y los valores de los campos de coincidencia soportados por la especificación.

A partir de estas definiciones podemos establecer que la tabla de flujos del dispositivo está compuesta por un conjunto de entradas que definen los flujos y acciones a realizar sobre éstos, brindando así las herramientas necesarias para establecer las capacidades de comunicación en una red de datos.

Cada una de las entradas de la tabla de flujo está compuesta por un conjunto de campos que establecen los criterios para que un paquete sea asociado a un flujo, mantienen información estadística y establecen las acciones a realizar sobre los paquetes del flujo.

Así definimos los campos de una entrada de flujo como una n-tupla formada por: campos de coincidencia, contadores y un conjunto de acciones.

CAMPOS DE COINCIDENCIA	CONTADORES	ACCIONES
...	...	...
CAMPOS DE COINCIDENCIA	CONTADORES	ACCIONES

Figura 4.2: Tabla de Flujos

Campos de Coincidencia

Los campos de coincidencia son los que permiten la definición de los criterios que debe cumplimentar un paquete de datos para pertenecer al flujo.

En la versión 1.0 de la especificación se establecen los siguientes criterios:

- **IN PORT**: establece como criterio el puerto del dispositivo por el cual se recibió el paquete de datos.
- **DL SRC**: establece como criterio la dirección MAC de origen del paquete de datos.
- **DL DST**: establece como criterio la dirección MAC de destino del paquete de datos.
- **DL TYPE**: establece como criterio el valor del campo EtherType del paquete de datos.
- **DL VLAN**: establece como criterio el valor del identificador de vlan del paquete de datos.
- **DL VLAN PCP**: establece como criterio el valor de prioridad "PCP" de vlan del paquete de datos.
- **NW SRC**: establece como criterio el valor de la dirección ip de origen del paquete de datos.
- **NW DST**: establece como criterio el valor de la dirección ip de destino del paquete de datos.
- **NW PROTO**: establece como criterio el valor del campo IP Protocol del paquete de datos. También es utilizado para definir el código de operación para los paquetes ARP.
- **NW TOS**: establece como criterio el valor del campo ToS del paquete de datos.

- **TP SRC**: establece como criterio el valor del puerto de origen para paquetes UDP y TCP del paquete de datos. También es utilizado para definir el tipo en los paquetes ICMP.
- **TP DST**: establece como criterio el valor del puerto de destino para paquetes UDP y TCP del paquete de datos. También es utilizado para definir el código en los paquetes ICMP.

Además de permitir establecer valores para cada una de las tuplas, la especificación indica que se puede utilizar el valor ANY en cualquiera de ellas, en cuyo caso se producirá una coincidencia cualquiera sea el valor en el paquete de datos.

También define que varios de estos campos pueden enmascararse a través del uso de bits de máscara.

Un ejemplo son las IP de origen y destino donde se pueden especificar la cantidad de bits a descartar, desde el comienzo de la dirección, y que de esta forma solo coincidan las direcciones que comienzan con los bits hasta la longitud de la máscara.

Contadores

Los contadores permiten almacenar información estadística relevante acerca del dispositivo para luego poder ser consultada por el controlador.

La especificación en su versión 1.0 establece que los dispositivos deben mantener contadores para la tabla de flujos, cada una de sus entradas y los puertos del dispositivo.

En el caso de las tablas, define que se deben mantener contadores para: la cantidad de entradas, la cantidad total de búsquedas realizadas, y la cantidad total de búsquedas exitosas en las entradas de la tabla.

Para las entradas de la tabla define que se deben mantener contadores para: la cantidad de paquetes y bytes de los flujos procesados y la duración en segundos y nanosegundos desde que la entrada fue insertada en la tabla de flujos.

Para los puertos define que se deben mantener contadores para: la cantidad de paquetes y bytes recibidos y enviados por éste y contadores de varios tipos de errores.

Acciones

Las acciones establecidas en cada una de las entradas de la tabla de flujos son las que definen qué operaciones realiza el dispositivo con aquellos paquetes que coinciden con el flujo de datos, definido a través de los campos de coincidencia de la entrada.

Cada una de las entradas de la tabla puede tener asociadas cero o más acciones. La especificación define tres categorías de acciones: **Reenvío, Descarte, Modificación**.

REENVÍO

Las acciones de Reenvío permiten establecer que el paquete de datos debe ser enviado a uno o más puertos físicos o a un puerto virtual del dispositivo.

Según la especificación los puertos virtuales NORMAL y FLOOD son opcionales mientras que el resto debe estar implementado en todos los dispositivos.

La semántica de los puertos virtuales se define como:

- **ALL**: envía el paquete a todos los puertos físicos del dispositivo excepto por el puerto por el que se recibió el paquete.
- **CONTROLLER**: encapsula y envía el paquete al controlador a través del canal de comunicación.
- **LOCAL**: envía el paquete al propio dispositivo a través de su pila de red local.
- **TABLE**: este puerto virtual se utiliza cuando el controlador envía un paquete al dispositivo a través del canal de comunicación y desea que sea procesador a través de la tabla de flujos del dispositivo.

- **IN PORT**: envía el paquete al puerto por el cual fue recibido el paquete.
- **NORMAL**: este puerto virtual permite que el paquete de datos sea procesado por la lógica tradicional del dispositivo, envía el paquete a la instancia del plano de datos tradicional del dispositivo, evitando que este sea procesado a través de la lógica OpenFlow.
- **FLOOD**: envía el paquete a través del spanning tree mínimo mantenido por el dispositivo.

DESCARTE

Aunque la acción explícita de Descarte de paquete no está definida, se establece que, cuando la lista de acciones no incluye una acción de Forward, el paquete es descartado.

MODIFICACIÓN

Las acciones de Modificación son las que permiten cambiar los datos del paquete antes de reenviarlo.

Según la especificación, este tipo de acciones es opcional, pero se recomienda su implementación debido a que extienden la funcionalidad básica de un dispositivo permitiendo realizar las acciones necesarias en la mayoría de los escenarios sin la necesidad de la participación del controlador.

Las acciones definidas en la especificación son:

- **SET VLAN VID**: modifica el identificador de VLAN.
- **SET VLAN PCP**: modifica la prioridad de VLAN.
- **STRIP VLAN**: quita los encabezados de VLAN.
- **SET DL SRC**: modifica la dirección origen ethernet.
- **SET DL DST**: modifica la dirección destino ethernet.
- **SET NW SRC**: modifica la dirección origen IP.

- **SET NW DST**: modifica la dirección de destino IP.
- **SET NW TOS**: modifica los bits tipo de tipo de servicio IP.
- **SET TP SRC**: modifica el puerto origen de tcp o udp.
- **SET TP DST**: modifica el puerto de destino de tcp o udp.

PROTOCOLO DE COMUNICACIÓN

El protocolo de comunicación OpenFlow establece la semántica y formato de los mensajes entre los dispositivos OpenFlow y el controlador SDN.

La especificación establece que las comunicaciones entre el controlador y el dispositivo se realizan a través de un canal de comunicación TLS sobre TCP.

Cada mensaje intercambiado entre el controlador y el dispositivo comienza con un encabezado que especifica la versión de OpenFlow utilizada, el tipo de mensaje, la longitud del mensaje y un identificador de transacción del mensaje.

La especificación clasifica los tipos de mensajes en tres categorías en función de quién es el responsable de iniciar la comunicación: mensajes simétricos, mensajes asimétricos y mensajes controlador-dispositivo.[4]

Mensajes Simétricos

Este tipo de mensajes pueden ser iniciados por el controlador o el dispositivo indistintamente.

La especificación define cuatro mensajes en esta categoría: HELLO, ECHO REQUEST. ECHO REPLY Y VENDOR.

- **HELLO**: se intercambian luego del establecimiento del canal TLS, para determinar la mayor versión del protocolo OpenFlow

que soportan el controlador y el dispositivo. Se especifica que se debe utilizar la menor de las dos versiones.

- **ECHO REQUEST y ECHO REPLY**: son intercambiados para asegurar el funcionamiento de la conexión y calcular la latencia y ancho de banda de la conexión entre el controlador y el dispositivo.
- **VENDOR**: estos mensajes están reservados para experimentación y su utilización por vendedores específicos.

Mensajes Asimétricos

Este tipo de mensajes son enviados desde el switch al controlador sin que éste los haya solicitado.

La especificación define cuatro mensajes en esta categoría: PACKET IN, FLOW REMOVED, PORT STATUS y ERROR.

- **PACKET_IN**: este mensaje es utilizado por el dispositivo para enviar paquetes al controlador.
- **FLOW REMOVED**: este mensaje es utilizado por el dispositivo para informar al controlador que una entrada de la tabla ha sido eliminada debido a que expiró alguno de los tiempos establecidos para que esté instalada.
- **PORT STATUS**: este mensaje es utilizado por el dispositivo para informar al controlador sobre el cambio en el estado de alguno de sus puertos.
- **ERROR**: este mensaje permite al dispositivo informar sobre varios tipos de errores al controlador.

Mensajes Controlador-Dispositivo

Este tipo de mensajes son iniciados por el controlador, y pueden ser subdivididos en 6 categorías: Características, Configuración, Modificación de Estado, Lectura de Estado, Envio de Paquete y Barrera.

Características

Estos mensajes se intercambian luego del establecimiento del canal de comunicación y permiten al controlador conocer las capacidades del dispositivo.

La especificación define los mensajes: FEATURES REQUEST y FEATURES REPLY.

- **FEATURES REQUEST**: es enviado por el controlador para pedir la configuración al dispositivo.
- **FEATURES REPLY**: es enviado por el dispositivo en respuesta al pedido del controlador.

La información contenida en la respuesta contiene el datapath_id, la información de sus puertos y las acciones que soporta.

El datapath_id es el identificador unívoco del dispositivo en la red y está formado por 64 bits de los cuales los primeros 16 bits son para que sean definidos por el implementador y los restantes 48 para la dirección ethernet del dispositivo. Cuando el controlador desea comunicarse con alguno de los dispositivos conectados a él, utiliza este identificador.

Configuración

Estos mensajes se utilizan para obtener y modificar la configuración del dispositivo.

La especificación define tres mensajes: GET CONFIG REQUEST, GET CONFIG REPLY y SET CONFIG.

- **GET CONFIG REQUEST**: es enviado por el controlador para obtener la información de configuración del dispositivo.
- **GET CONFIG REPLY**: es enviado por el dispositivo en respuesta al pedido del controlador.
- **SET CONFIG**: es enviado por el controlador al dispositivo para modificar su configuración. El dispositivo no contesta a este mensaje.

La información de configuración incluye las acciones a realizar sobre los fragmentos IP y la cantidad de bytes de un paquete de datos que serán transmitidos por el dispositivo al controlador a través del mensaje PACKET IN.

Modificación de Estado

Estos mensajes se utilizan para agregar, modificar y eliminar entradas en la tabla de flujos y modificar la configuración de los puertos del dispositivo.

La especificación define los mensajes: FLOW MOD Y PORT_MOD.

Y cinco posibles comandos para el mensaje FLOW MOD: ADD, MODIFY, MODIFY STRICT, DELETE, DELETE STRICT.

- **ADD**: es enviado por el controlador para agregar un nuevo flujo a la tabla del dispositivo.
- **MODIFY y MODIFY STRICT**: son enviados por el controlador para modificar una entrada de la tabla del dispositivo.
- **DELETE Y DELETE STRICT**: son enviados por el controlador para eliminar una entrada de la tabla del dispositivo.
- **PORT MOD**: es enviado por el controlador para la modificar la configuración de alguno de los puertos del dispositivo.

Lectura de Estado

Este tipo de mensajes se utilizan para recolectar la información estadística del dispositivo.

La especificación define los mensajes: STATS REQUEST y STATS REPLY.

- **STATS REQUEST**: es enviado por el controlador para pedir la información estadística al dispositivo.
- **STATS REPLY**: es enviado por el dispositivo en respuesta al pedido del controlador.

Envio de Paquete

Estos mensajes se utilizan cuando el controlador desea enviar un paquete a través del dispositivo ya sea un paquete completo generado por él o algún paquete que recibió del dispositivo para ser procesado.

La especificación define el mensaje PACKET OUT.

Barrera

Estos mensajes se utilizan cuando el controlador desea asegurarse que los mensajes hasta el momento enviados al dispositivo han sido procesados por este.

La especificación define los mensajes: BARRIER REQUEST y BARRIER REPLY.

- **BARRIER REQUEST**: es enviado por el controlador para informar al dispositivo que desea asegurarse que todos los mensajes enviados hasta el momento han sido procesados. Al recibir este mensaje el dispositivo debe terminar de procesar todos los mensajes pendientes antes de aceptar nuevos mensajes.
- **BARRIER REPLY**: es enviado por el dispositivo para informar al controlador que ha terminado de procesar todos los mensajes pendientes.

Parte III

DESARROLLO DEL TRABAJO

5

HERRAMIENTAS

Con el objetivo de realizar el prototipo funcional de una interfaz de usuario capaz de mostrar la topología de una red SDN y permitir realizar cambios en su funcionamiento de forma centralizada se seleccionó un conjunto de herramientas que permitieran:

- La creación e instanciación de los ambientes.
- La implementación de la interfaz de usuario.
- La implementación de la capa de aplicación SDN.
- La implementación de la capa de control SDN.
- La implementación de la capa de datos SDN.

CREACIÓN E INSTANCIACIÓN DE AMBIENTES

Vagrant

Vagrant [9] es una herramienta que permite el aprovisionamiento de máquinas virtuales de forma agnóstica a la tecnología de virtualización subyacente.

A través de la definición abstracta de los recursos y características deseados para la máquina virtual, la tecnología se encarga de realizar todas las operaciones específicas y así obtener una máquina virtual ejecutando.

La forma de almacenar la especificación que utiliza la tecnología permite compartirla lo que favorece el trabajo colaborativo.

A su vez la herramienta posee integración nativa con la tecnología de automatización "Ansible".

Esta tecnología fue utilizada para generar la especificación de las máquinas virtuales necesarias para el desarrollo del trabajo, dada la facilidad que brinda para compartir los archivos de especificación, y la capacidad de crear e instanciar todas las máquinas virtuales de forma muy sencilla y ágil favoreciendo tanto el trabajo colaborativo como su migración a otra/s máquinas físicas.

Ansible

Ansible [10] es una herramienta para la automatización de tareas, fue utilizada para escribir el conjunto de tareas necesarias (receta) para instalar y configurar las dependencias de software que los ambientes virtualizados del trabajo requieren.

Estas recetas permiten obtener las plataformas requeridas, a través de la ejecución de la/s recetas, sin la necesidad de introducción de comandos de forma manual, agilizando el proceso de instanciación de la plataforma y favoreciendo su reproductibilidad, testeo y el trabajo colaborativo.

VirtualBox

VirtualBox [11] es una plataforma de virtualización open source disponible para un gran conjunto de sistemas operativos. Fue la herramienta seleccionada para la creación y ejecución de las máquinas virtuales debido a su facilidad de uso, integración con Vagrant y disponibilidad en sistemas operativos Linux y Windows.

IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO

AngularJS

AngularJS [12, 13] es un framework de desarrollo ágil de aplicaciones WEB sobre Javascript. Cuenta con un gran número de componentes integrados y plugins que agilizan y facilitan el desarrollo de aplicaciones e interfaces de usuario complejas.

Bootstrap

Bootstrap [14, 15] es un framework para aplicaciones WEB, que facilita y agiliza el desarrollo de interfaces de usuario.

D3js

D3js [16] es una librería desarrollada en JavaScript para la manipulación y visualización de datos. En el trabajo fue utilizada para generar la interfaz de usuario, encargada de mostrar la topología de la red a través de la creación del SVG que contiene los dispositivos y sus enlaces.

IMPLEMENTACIÓN DE LA CAPA DE APLICACIÓN SDN

La capa de aplicación fue desarrollada utilizando como lenguaje de programación Python, debido a la facilidad y agilidad de este lenguaje y su integración con el producto Frenetic [17, 18, 19].

NetworkX

NetworkX [20] es una librería para Python que permite generar, manipular y ejecutar algoritmos de grafos. La implementación del módulo encargado de responder a los eventos generados ante cambios en la topología, utiliza estas librerías para crear y mantener el grafo que modela la topología de la red.

Tornado

Tornado [21] es un framework web para Python, que fue utilizado para implementar: La comunicación entre la capa de aplicación y el controlador OpenFlow. La interfaz de comunicación HTTP que la capa de aplicación debe implementar para permitir su consulta por la interfaz de usuario.

IMPLEMENTACIÓN DE LA CAPA DE CONTROL SDN

Introducción

Para realizar la elección del framework de desarrollo, se realizó una evaluación de las diversas alternativas open source existentes. Y se tomaron como parámetros de evaluación la facilidad de escritura, el lenguaje de programación, las capacidades de abstracción y la actividad y crecimiento del proyecto.

Nox

Nox [22, 23] fue la primera herramienta evaluada, fue desarrollada por Nicira al mismo tiempo que se desarrollaba la especificación 1.0 de OpenFlow.

Nox está desarrollado en C++ e implementa una API de programación con el protocolo OpenFlow permitiendo implementar las capas de control y aplicación de la arquitectura SDN.

Aunque en la bibliografía y la documentación de la herramienta se utiliza el término controlador para describirlo, Nox es solo una herramienta de programación con la capacidad de desarrollar tanto componentes de la capa de control como de la capa de aplicación en arquitecturas SDN.

La actividad del proyecto se encuentra a cargo de la Universidad de Stanford ysu licencia en open source. Aunque fue una de las herramientas pioneras en las primeras etapas de OpenFlow su desarrollo no muestra actividad desde el año 2004.

Debido a que la documentación es escasa y que el lenguaje de programación está orientado a la performance y no a la facilidad de desarrollo, se descartó la utilización de éste. Sin embargo se lo menciona porque algunas de las herramientas hacen uso de algunas de sus funcionalidades.

Pox

Pox [24] es otro framework de desarrollo basado en Nox pero desarrollado en Python, lo que mejora la facilidad y agilidad de desarrollo, soporta la especificación OpenFlow 1.0 pero al igual que Nox su desarrollo está estancado.

Aunque no se lo utiliza explícitamente para el desarrollo, algunos de los componentes y frameworks utilizados reutilizan algunas de sus funcionalidades, y es una buena herramienta para realizar pseudocódigos comparativos.

Frenetic

Frenetic [17, 18, 19] fue el producto seleccionado para la implementación del controlador OpenFlow.

La mayoría de las herramientas anteriores proveen interfaces de programación de bajo nivel y con poca capacidad de abstracción respecto al hardware de la capa de datos y a la especificación OpenFlow, ya que básicamente son interfaces de programación que solo proveen funciones o procedimientos para facilitar el envío y recepción de mensajes OpenFlow.

Frenetic es un producto que implementa la capa de control de la arquitectura SDN e implementa un lenguaje de programación de alto nivel para redes NetKat [25], que se basa en las propiedades del algebra de Kleene y fue implementado a través del paradigma de programación funcional.

Con este enfoque, el programador utiliza abstracciones de alto nivel para definir la funcionalidad deseadas en la red, y el sistema genera las instrucciones de bajo nivel necesarias para lograrla.

El desarrollo de este framework es bastante activo, el núcleo del sistema esta desarrollado en OCaml, y posee interfaces de programación en Python y HTTP.

En el Apéndice A se pueden encontrar las propiedades mátematicas y la especificación de la sintaxis del lenguaje NetKat en un metalenguaje tipo BNF utilizado por Frenetic.

En las siguientes secciones se introducen las capacidades de Frenetic.

Composición Secuencial

La composición secuencial de políticas "**(p ; q)**" aplica primero la política **p** a los paquetes de entrada y luego aplica la política **q** a cada uno de los paquetes resultado de aplicar la política **p**. Dado este funcionamiento en mucha de la bibliografía se la compara con el funcionamiento de los pipes en los sistemas operativos.

Para ejemplificar este problema supongamos que deseamos enviar el tráfico de aquellos paquetes que tienen como dirección de destino 00:00:00:00:00:02 al puerto 4 y descartar el resto de los paquetes.

Para implementarlo a través de NetKat utilizamos el operador de composición paralela de la siguiente forma:

```
(filter ethDst = 00:00:00:00:00:02) ; port := 4
```

Composición Paralela

La composición paralela de políticas "**(p + q)**" genera la unión de los conjuntos de paquetes producidos por la aplicación de la política **p** y la aplicación de la política **q**.

Para ejemplificar este problema supongamos que deseamos enviar todo el tráfico al puerto ALL (65532) para implementar un hub y para aquellos paquetes que además tienen la dirección de destino 00:00:00:00:00:02 enviarlos también al controlador.

Para implementarlo a través de NetKat utilizamos el operador de composición paralela de la siguiente forma:

```
(port := 65532 )
+
(filter ethDst = 00:00:00:00:00:02 ; port := pipe("all"))
```

Notar cómo en esta composición también utilizamos el operador de composición secuencial para para aplicar el envío de tráfico al controlador de los paquetes seleccionados.

Operadores de comparación

La especificación OpenFlow provee los campos de coincidencia para definir los criterios de matcheo de los flujos, pero solo permite la utilización del operador de comparación "igual a".

Cuando se desea utilizar el operador "distinto a" deben generarse entradas adicionales con el objetivo de emularlo.

Esto agrega complejidad al programa y atenta contra la legibilidad, la agilidad y el mantenimiento del programa.

Frenetic implementa el comparador de forma nativa generando de forma transparente al programador las instrucciones necesarias para instalar las entradas en el dispositivo.

Para ejemplificar este problema, supongamos que deseamos enviar todo el tráfico al puerto 2 del dispositivo, excepto aquellos paquetes cuya dirección MAC de origen es 00:00:00:00:00:01, los cuales deseamos descartar.

Para conseguir esta funcionalidad, utilizando cualquiera de las herramientas OpenFlow de bajo nivel, debemos agregar dos entradas a la tabla de flujo.

1. Una que define el flujo de los paquetes que tienen como dirección MAC origen 00:00:00:00:00:01.
2. Y otra que maneje el resto de los paquetes.

Las instrucciones OpenFlow necesarias para su implementación son:

```
mininet> dpctl add-flow dl_src=00:00:00:00:00:01,actions=drop
mininet> dpctl add-flow actions:output=2
mininet> dpctl dump-flows
***s1 ----------------------------------------------------
dl_src=00:00:00:00:00:01 actions=drop
actions=output:2
-------------------------------------------------------------
```

Las instrucciones en lenguaje NetKat necesarias para su implementación son:

```
filter not (ethSrc = 00:00:00:00:00:01) ; port := 2
mininet> dpctl dump-flows
***s1 ----------------------------------------------------
dl_src=00:00:00:00:00:01 actions=drop
actions=output:2
-------------------------------------------------------------
```

Podemos observar como el resultado final de la tabla de flujos coincide con la implementación nativa.

Operadores Lógicos

La especificación OpenFlow establece un "and" implícito entre todos los valores de los campos de coincidencia de una entrada de la tabla de flujos.

Al igual que en el caso anterior cuando se desea utilizar el operador "or" deben generarse entradas adicionales.

Continuando con el ejemplo anterior, si quisiéramos agregar la condición de enviar todo el tráfico al puerto 2 del dispositivo excepto aquellos paquetes cuya dirección mac de origen es 00:00:00:00:00:01 o que ingresaron por el puerto 1, deberíamos agregar tres entradas a la tabla de flujo.

1. Una que define el flujo de los paquetes que ingresan por el puerto 1.
2. Otra que define el flujo de los paquetes que tienen como dirección mac origen 00:00:00:00:00:01.
3. Y una última que maneje el resto de los paquetes.

Las instrucciones OpenFlow necesarias para su implementación son:

```
mininet> dpctl add-flow in_port=1,actions=drop
mininet> dpctl add-flow dl_src=00:00:00:00:00:01,actions=drop
mininet> dpctl add-flow actions:output=2
mininet> dpctl dump-flows
***s1 ----------------------------------------------------
in_port=1 actions=drop dl_src=00:00:00:00:00:01 actions=drop
actions=output:2
------------------------------------------------------------
```

Las instrucciones en lenguaje NetKat necesarias para su implementación son:

```
filter not (port=1 or ethSrc = 00:00:00:00:00:01);port := 2
mininet> dpctl dump-flows
***s1 ----------------------------------------------------
in_port=1 actions=drop dl_src=00:00:00:00:00:01 actions=drop
actions=output:2
-----------------------------------------------------------
```

Podemos observar como el resultado final de la tabla de flujos coincide con la implementación nativa.

La implementación de este comparador también permite la comparación de un valor de los campos de coincidencia con una lista de valores, que en la implementación de bajo nivel implicaría la inserción de un conjunto de entradas.

IMPLEMENTACIÓN DE LA CAPA DE DATOS SDN

Open vSwitch

Open vSwitch [26] es una implementación de un switch virtual multicapa con licencia open source capaz de ejecutarse en un sistema operativo Linux. Implementa la especificación OpenFlow en varias de sus versiones y permite la conexión de máquinas virtuales al dispositivo, cuenta con herramientas de administración que permiten gestionar las tablas de flujo del dispositivo y se encuentra muy bien integrado a la herramienta de emulación de redes elegida para el desarrollo del presente trabajo.

Mininet

Mininet [27] es capaz de emular y ejecutar una colección de hosts y dispositivos de red permitiendo establecer la interconexión entre estos.

Utiliza tecnologías de virtualización ligera permitiendo instanciar una gran cantidad de dispositivos y utilizando pocos recursos del sistema para ejecutarlos.

Cuenta con una interfaz de consola que permite la administración en tiempo real de los dispositivos y links emulados, la consulta y modificación de las tablas de flujo de los dispositivos y la ejecución de comandos dentro de los dispositivos emulados.

Miniedit

Miniedit [28] es una interfaz para Mininet que permite diseñar una topología de red de forma gráfica y generar el código necesario para ejecutarla en mininet. Esta herramienta fue utilizada para generar las topologías de ejemplo utilizadas en el trabajo.

6

PROBLEMAS PROPUESTOS

INTRODUCCIÓN

En este capítulo se presenta un conjunto de problemas y funcionalidades comunes en las redes de datos que serán utilizados en el capítulo siguiente para evaluar su resolución a través de las herramientas disponibles en las arquitecturas tradicionales versus su resolución a través de arquitecturas SDN.

TOPOLOGÍA DE LA RED

Motivación

Uno de los problemas recurrentes con los que me he encontrado durante el desarrollo de mi tarea profesional, es la necesidad de contar con la información de la topología completa de la red de la organización.

Esta información es necesaria entre otras cosas para:

- La evaluación y resolución de problemas de conectividad.
- La implementación de las diversas políticas de la red, como políticas de seguridad, de calidad de servicio y otras que requieren de esta información para determinar en qué lugar de la red deben ser implementadas.
- El rápido dimensionamiento del impacto ante la caída de un dispositivo o enlace.

- La evaluación de los SPOFs "Single point of failures" en la red, y la toma de decisiones respecto a cómo mitigarlos.
- La ingeniería de tráfico precisa de esta información para poder analizar la utilización, carga y patrones de tráfico, y a partir de ésta generar opciones de mejora.
- Las tareas de los administradores de red no solo incluyen tareas técnicas específicas sino que en general también involucran tareas de gestión, que incluyen diversos y variados tipos de reportes hacia niveles superiores de la organización. Estos necesitan de la información de topología para ser entendidos por los diversos actores destino de los mismos.
- La implementación de políticas de alarma y monitoreo por parte del NOC (Network Operation Center), encargado de monitorear y asegurar el correcto funcionamiento de la red.
- La creación y mantenimiento de los sistemas de inventario y la generación de la documentación mínima necesaria dentro de una organización a partir de la información de la topología de la red.

Requerimientos

Podemos enumerar los requerimientos mínimos al problema de la generación de la información topológica de la red de datos en una organización como:

- Descubrir y mantener la totalidad de los dispositivos de red de la organización ya sean estos switches, routers u otro tipo de dispositivos.
- Descubrir y mantener la totalidad de los links de intercomunicación entre los dispositivos del punto anterior.

- Descubrir y mantener la totalidad de los dispositivos finales de la organización como servidores, laptops, dispositivos móviles y pc's de escritorio.
- Tener la capacidad de consultar la información obtenida para que la misma pueda ser utilizada por diversos sistemas dentro de la organización.

Y además podemos definir los siguientes requerimientos deseables:

- Brindar una interfaz de usuario capaz de mostrar la topología de la red de forma visual.
- Permitir la integración de esta información con los sistemas externos de inventario.
- Permitir la integración de esta información con las herramientas del NOC.

POLÍTICAS DE ACCESO

Motivación

La implementación de políticas de acceso es una tarea habitual dentro de las tareas operativas normales de los administradores y operadores de la red.

A través de la implementación de las políticas de acceso en la/s redes de una organización los operarios y administradores definen las reglas para permitir o denegar las comunicaciones entre diversos tipos de dispositivos y usuarios.

Podemos clasificar a estas reglas en varios grupos en función del momento de su definición y su propósito.

- **Reglas estáticas**: en general este tipo de reglas puede derivarse a través del análisis y las especificaciones de las políticas de comunicación y seguridad de la organización. En este grupo podemos incluir las políticas de comunicación y seguridad

aceptables entre: los diferentes usuarios y áreas dentro de la organización, entre la organización y otras organizaciones con las que se poseen acuerdos, y entre la organización y el resto del mundo.

- **Reglas dinámicas**: además de las reglas mencionadas anteriormente en muchas organizaciones existe la necesidad de generar reglas de acceso y seguridad que no se desprenden naturalmente de las políticas especificadas. Su naturaleza más dinámica responde al hecho de que deben implementarse en respuesta a estímulos o eventos que no siempre pueden ser anticipados. En este grupo podemos mencionar como ejemplo las políticas a implementar ante un ataque de denegación de servicio o la aparición de un virus o malware entre otros.

Requerimientos

Podemos enumerar los requerimientos mínimos al problema de la implementación de políticas de acceso en una organización como:

- Permitir su implementación de forma rápida a fin de utilizar eficientemente los recursos humanos y que su impacto sea reflejado en la red de forma rápida.
- Permitir la verificación y completitud de la política completa entendiendo a ésta como el conjunto total resultado de la aplicación de sus reglas individuales.
- Permitir el testeo tanto de reglas individuales como de la política completa con el fin de la verificación de su correctitud.

7

DESARROLLO Y SOLUCIONES PROPUESTAS

En este capítulo se presenta el desarrollo de software realizado para la evaluación de la tecnología SDN, se realiza una comparación entre las soluciones comúnmente utilizadas en las arquitecturas tradicionales versus la utilización de arquitecturas SDN utilizando el producto desarrollado para implementar las soluciones.

El código fuente de la aplicación desarrollada se puede descargar desde el respositorio:

```
git@gitlab.com:chelobarreto/tesis.git
```

El código fuente de la interfaz de usuario puede descargarse desde el repositorio:

```
git@gitlab.com:chelobarreto/tesis-frontend.git
```

Los archivos de especificación de las máquinas virtuales junto con los pasos necesarios para replicar la arquitectura pueden descargarse desde el repositorio:

```
git@gitlab.com:chelobarreto/tesis-laboratorio.git
```

ARQUITECTURA DEL PRODUCTO DESARROLLADO

En la Figura 7.1 puede observarse cómo esta compuesta la arquitectura de la aplicación desarrollada sus componentes y sus interfaces de comunicación y puede observarse como se corresponde con la arquitectura SDN teórica presentada en el Capítulo 3.

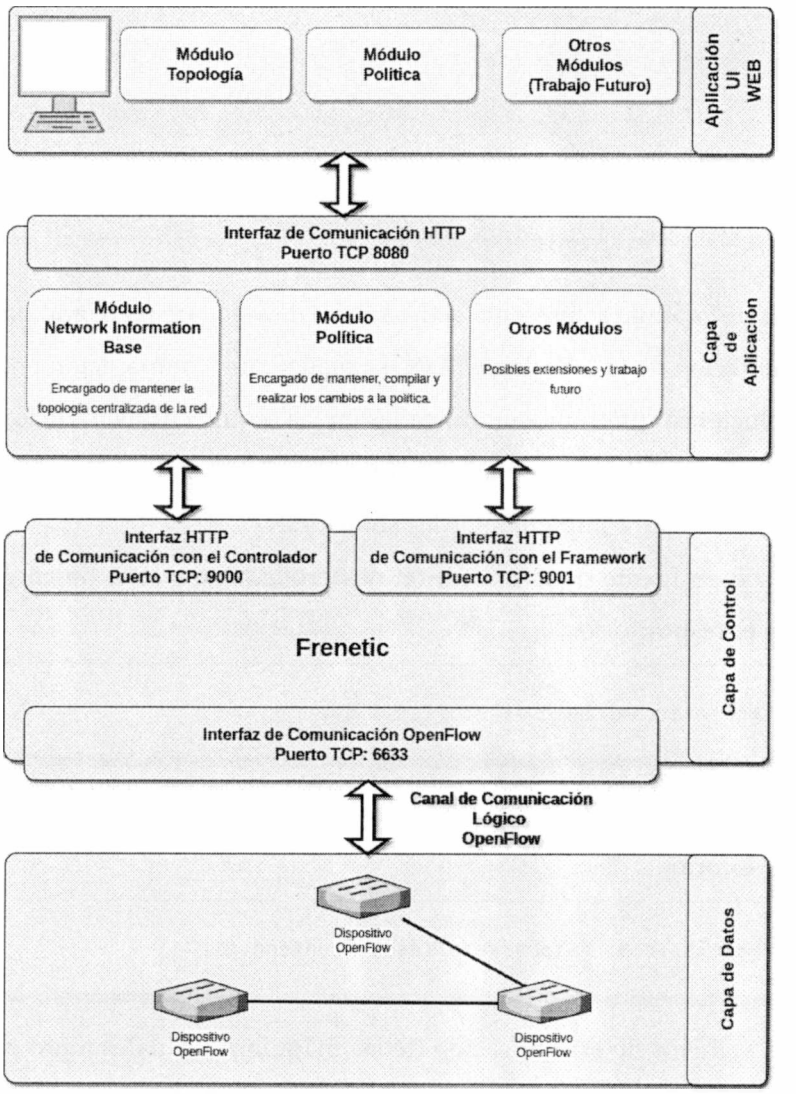

Figura 7.1: Arquitectura del producto desarrollado

SOLUCIÓN AL PROBLEMA DE TOPOLOGÍA DE RED

Soluciones tradicionales

El problema de obtener y contar con la topología de la red en una organización es un problema recurrente y que generalmente no tiene respuestas satisfactorias ni confiables en todas las organizaciones en las que he desempeñado funciones y/o he tenido contacto.

La forma tradicional de mantener esta información se basa en la utilización de documentación que debe ser generada y actualizada

por los operadores y administradores de la red, utilizando para ello herramientas gráficas como "Microsoft Visio".

Las organizaciones más grandes suelen adquirir software específico y costoso que en general produce la información correcta solo cuando los dispositivos de red cumplen con requerimientos rigurosos como: fabricante y soporte de protocolos específicos. En general estas herramientas no se adaptan a ambientes heterogéneos con dispositivos de diferentes vendedores.

En efecto mucha de la documentación de estos vendedores recomienda el uso de estas herramientas estáticas y la actualización de esta información cuando se realiza un cambio.

En conclusión actualmente la solución a este problema varía de organización en organización, especialmente en función su tamaño y los recursos que está dispuesta a invertir en este sentido.

Solución desarrollada a través de SDN

Para atacar este problema utilizando la tecnología SDN se desarrolló un módulo dentro de la capa de aplicación y un módulo en la interfaz de usuario. Como objetivo general se estableció que se debía contar con la capacidad de:

"Permitir consultar gráficamente la información de la topología de la red, incluyendo los dispositivos de la capa de datos OpenFlow y los enlaces entre ellos en tiempo real."

El módulo de la capa de aplicación responsable de reaccionar frente a los cambios producidos en la topología de la capa de datos es el denominado Network Information Base (NIB), e implementa tres funcionalidades básicas:

1. Responde a los eventos de conexión y desconexión de los dispositivos y sus puertos, que son generados por la capa de control (Frenetic), y a partir de estos mantiene una estructura de datos de tipo grafo que representa la topología de la red.

2. Implementa un algoritmo encargado de realizar el descubrimiento de los links de comunicación entre los dispositivos de la red, y partir de esta información mantiene las aristas del grafo que representan la topología.

3. Permite la consulta de la topología de la red a través de la exposición de una interfaz HTTP que es utilizada por la interfaz de usuario para generar el gráfico de la topología.

La elección de una estructura de datos de tipo grafo para el mantenimiento de la topología de red permite la ejecución de algoritmos de grafos como los de conectividad, caminos mínimos y otros, aunque esta capacidad no fue implementada en la interfaz de usuario.

Para implementar el mantenimiento de los nodos, el módulo implementa las siguientes funciones:

- **connected(switches)**: esta función es invocada cuando la aplicación es iniciada y puede establecer la comunicación con el controlador Frenetic a través de HTTP. Recibe como parámetro los dispositivos OpenFlow conectados al controlador junto con la información de los puertos de cada uno de ellos. Su función es invocar a la función switch_up por cada uno de los dispositivos conectados.

- **switch_up(switch, ports)**: esta función es invocada tanto por la función anterior como cuando se recibe un evento asincrónico desde el controlador para informar que un nuevo dispositivo se ha conectado. Su función es agregar, como un nodo, del grafo la información del dispositivo.

- **switch_down(switch)**: esta función es invocada cuando se recibe un evento de desconexión de un dispositivo desde el controlador. Su función es eliminar del grafo el nodo asociado al dispositivo y las aristas que lo incluyen.

- **port_up(switch, port)**: esta función es invocada tanto por la función switch_up como cuando se recibe un evento asincrónico

desde el controlador para informar que un nuevo puerto en un dispositivo se ha conectado. Cuando esta función es invocada se realiza el proceso encargado de inyectar paquetes LLDP en el puerto para implementar el descubrimiento de posibles links de comunicación con otros dispositivos, el algoritmo completo de esta funcionalidad se detalla en la próxima sección.

- **port_down(switch, port)**: esta función es invocada cuando se recibe un evento de desconexión de un puerto en un dispositivo desde el controlador. Su función es eliminar las aristas del grafo que incluyen al puerto desconectado.

Algoritmo para el descubrimiento de enlaces

Para realizar el descubrimiento de los enlaces entre dispositivos se utilizó el siguiente algoritmo:

1. Cuando se verifica que la aplicación puede conectarse con el controlador se instala una política por defecto en los dispositivos. Esta se encarga de que todos los paquetes LLDP recibidos en los dispositivos sean enviados al controlador.

2. Cuando se recibe un evento de conexión de un dispositivo o de un puerto en la aplicación desde el controlador, se procede a generar un paquete LLDP con los siguientes valores: en el campo chasis.id del TLV se coloca el datapath_id del dispositivo conectado, y en el campo port.id el numero de puerto del dispositivo. Luego se procede a enviarlo al puerto que acaba de conectarse.

3. Dada la política establecida en el Paso 1, si en el otro extremo de la conexión hay un dispositivo OpenFlow, al recibir un paquete LLDP éste lo envía al controlador. La aplicación recibe el paquete y además recibe como información adicional el identificador del dispositivo y número de puerto por el cual ingresó el paquete al dispositivo con esta información se puede agregar la arista al grafo que modela la topología.

Para ejemplificar el algoritmo se utilizará la topología de la Figura 7.2.

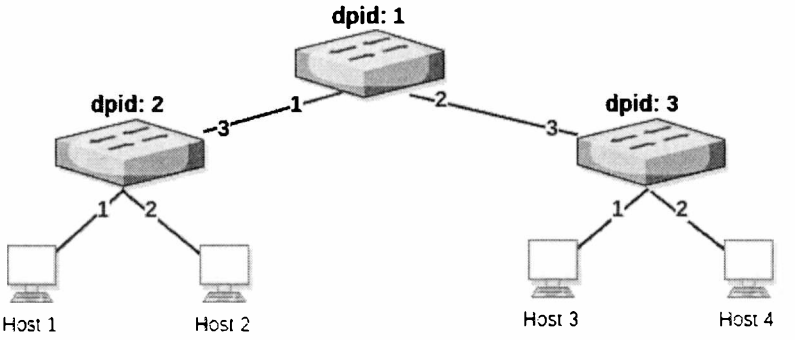

Figura 7.2: Topología de Ejemplo

Cuando se encienden los dispositivos, estos establecen el canal de comunicación con el controlador, luego se inicia la aplicación desarrollada y ésta se comunica con el controlador a través de su interfaz HTTP.

Al conectarse con el controlador la aplicación instala la política detallada en el Paso 1 en los 3 dispositivos, indicando que los paquetes LLDP recibidos serán enviados al controlador. Como paso siguiente, se agrega un nodo al grafo por cada uno de los dispositivos, y se generan los paquetes LLDP mostrados en la Tabla 7.1.

#PAQUETE	SWITCH	PUERTO	LLDP DPID	LLDP PORT
A	1	1	1	1
B	1	2	1	2
C	2	1	2	1
D	2	2	2	2
E	2	3	2	3
F	3	1	3	1
G	3	2	3	2
H	3	3	3	3

Table 7.1: Generación de Paquetes

Las columnas LLDP DPID y LLDP PORT especifican los datos que contendrá el paquete LLDP y las columnas SWITCH Y PUERTO indican en qué dispositivo y número de puerto se inyectará el paquete.

- **Paquete A**: cuando el paquete es enviado a través del dispositivo "1" en su puerto "1" en el otro extremo es recibido por el dispositivo "2" en su puerto "3". Este paquete es procesado a través de la tabla de flujos del dispositivo "2" y dada la política instalada se lo envía al controlador. Cuando el controlador lo recibe lo reenvía a la aplicación desarrollada junto con el identificador del dispositivo "2" y el identificador del puerto por el que lo recibió "3". Allí la aplicación inspecciona el contenido del paquete y encuentra los campos TLV chasis.id con valor "1" y port.id con valor "1". De esta manera se descubre el enlace entre los dos dispositivos. La aplicación utiliza estos datos para agregar la arista 1@1 => 2@3.

- **Paquete B**: el procesamiento de este paquete es análogo al paquete A y genera el descubrimiento de la arista 1@2 => 3@3.

- **Paquete C**: cuando el paquete es enviado a través del dispositivo "2" en su puerto "1", es recibido por el Host 1 y no se realiza ninguna acción debido a que este no atraviesa ningún dispositivo OpenFlow.

- **Paquete D**: el procesamiento de este paquete es análogo al del paquete C pero llegando al Host 2.

- **Paquete E**: el procesamiento de este paquete es análogo al paquete A y genera el descubrimiento de la arista 2@3 => 1@1.

- **Paquete F**: el procesamiento de este paquete es análogo al del paquete C pero llegando al Host 3.

- **Paquete G**: el procesamiento de este paquete es análogo al del paquete C pero llegando al Host 4.

- **Paquete H**: el procesamiento de este paquete es análogo al paquete A y genera el descubrimiento de la arista 3@3 => 1@2.

Para terminar de complementar el objetivo se desarrolló una interfaz HTTP en la aplicación que es utilizada por la interfaz de usuario para mostrar gráficamente la topología de la red.

En la Figura 7.3 se puede observar una captura de pantalla con una topología semejante a la del ejemplo utilizado para describir el algoritmo.

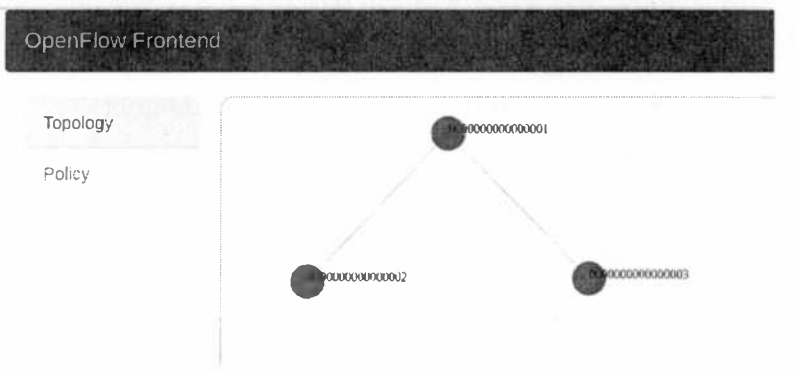

Figura 7.3: Interfaz de Usuario de la Topología

SOLUCIÓN AL PROBLEMA DE POLÍTICA DE ACCESO

Soluciones tradicionales

Tradicionalmente las soluciones a este tipo de problema se implementan a través de la definición de reglas de filtrado o ACLs, que definen las acciones a realizar sobre los paquetes cuando cumplen con un conjunto de criterios.

Este enfoque tiene como desventaja que cada fabricante y sistema operativo posee su propia sintaxis y semántica, y requiere la introducción manual de estas reglas a través de la interfaz de administración soportada por el dispositivo.

Esto obliga a realizar el análisis abstracto de la política, para luego introducirla manualmente en cada uno de los dispositivos de la red

donde debe aplicarse utilizando la sintaxis y semántica apropiada para el dispositivo.

Para ejemplificar el algoritmo se utilizará la topología de la Figura 7.4.

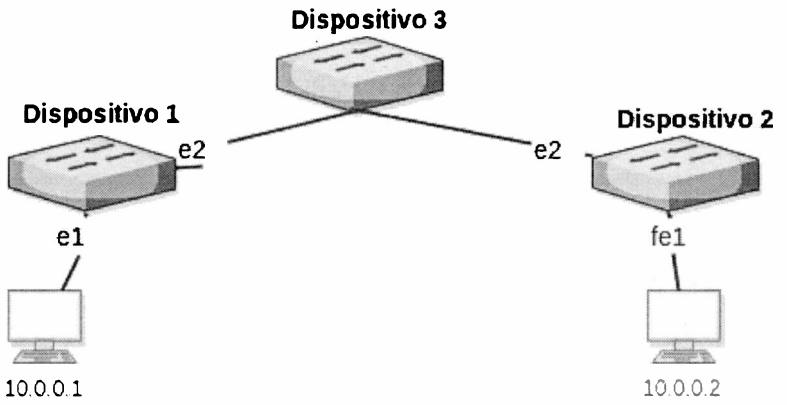

Figura 7.4: Topología de Ejemplo

Y se define la siguiente política de ejemplo:

"Permitir la comunicación entre los hosts 10.0.0.1 y 10.0.0.2 y denegar el resto del tráfico en la red"

Suponiendo que el Dispositivo 1 es del fabricante Juniper y que el Dispositivo 2 es del fabricante Cisco, para cumplimentar con la política de ejemplo definida necesitan ejecutar las siguientes instrucciones.

En el Dispositivo 1:

```
edit firewall family inet filter politica-1
set term politica-1 from source-address 10.0.0.1/32
set term politica-1 from destination-address 10.0.0.2/32
set term politica-1 then accept
set term politica-1 then drop
set interface e1 filter input politica-1
edit firewall family inet filter politica-2
set term politica-2 from source-address 10.0.0.2/32
set term politica-2 from destination-address 10.0.0.1/32
set term politica-2 then accept
set term politica-2 then drop
set interface e2 filter input politica-2
```

En el Dispositivo 2:

```
interface fe1 ip access-group 101 in
access-list 101 permit ip source 10.0.0.2 dest 10.0.0.1
access-list 101 deny any any
interface e2 ip access-group 102 in
access-list 102 permit ip source 10.0.0.1 dest 10.0.0.2
access-list 102 deny any any
```

Como puede observarse tanto la semántica como la sintaxis de las instrucciones para los dos tipos de dispositivos es muy distinta.

Solución desarrollada a través de SDN

Para atacar este problema utilizando la tecnología SD N se desarrolló un módulo dentro de la capa de aplicación y un módulo en la interfaz de usuario. Como objetivo general se estableció que se debía contar con la capacidad de:

" Permitir la utilización del lenguaje de alto nivel "NetKat" para la definición de la política y brindar una interfaz de usuario con la capacidad de obtener la traducción necesaria entre la política de alto nivel y las instrucciones OpenFlow de bajo nivel que la implementan. Y permitiendo la instalación de nuevas políticas en la red. "

Con el objetivo de realizar la comparación de la solución SDN respecto a la solución tradicional se utiliza la misma topología y la misma definición de política del apartado anterior.

Las instrucciones necesarias para lograr la implementación de la política en lenguaje NetKat son:

```
filter (ip4Src = 10.0.0.1 and ip4Dst = 10.0.0.2)
       or (ip4Src = 10.0.0.2 and ip4Dst = 10.0.0.1) ;
port := 65532
```

Puede observarse como esta implementación de alto nivel es mucho más natural, y tiene muchas semejanzas con un lenguaje de programación tradicional.

Los recursos humanos responsables de la actualización y diseño de las políticas solo necesitan capacitarse en el lenguaje de alto nivel y no como en el caso anterior en dos sintaxis y semánticas totalmente diferentes.

Además la introducción de esta política a través del controlador realiza las modificaciones necesarias en todos los dispositivos de la red sin la necesidad de que el operador acceda a cada uno para realizar los cambios en la política.

Parte IV

CONCLUSIONES Y TRABAJO FUTURO

8

CONCLUSIONES Y TRABAJO FUTURO

CONCLUSIONES

El objetivo teórico principal de este trabajo ha sido realizar una evaluación de los problemas recurrentes en el diseño, mantenimiento y operatoria de las redes de datos, y evidenciar las deficiencias y complejidades que estas tareas presentan actualmente debido a los avances tecnológicos en el área de las telecomunicaciones, que introducen nuevos y variados tipos de problemas y requerimientos para los que las arquitecturas y paradigmas tradicionales no fueron diseñados.

Para abordar esta problemática se realizó una investigación de las oportunidades que puede brindar la tecnología Software Defined Network "SDN", al utilizar otro enfoque arquitectónico diseñado con estos objetivos desde su concepción.

El objetivo práctico del trabajo fue evaluar las distintas herramientas y estándares disponibles para la implementación de redes basadas en la arquitectura SDN. Para ello se realizó una investigación de las especificaciones y herramientas más utilizadas y aceptadas; y se seleccionó un conjunto de éstas para desarrollar un producto que pudiera ser utilizado para implementar algunos de los requerimientos y comportamientos típicos de una red, y así realizar una comparación con las soluciones actualmente utilizadas.

La tecnología SDN, incorpora un nuevo enfoque que brinda grandes oportunidades para la resolución de problemas típicos en el diseño, operatoria y mantenimiento en las redes de datos, favorece la innovación y permite que las redes de datos puedan implementarse a través de la composición de aplicaciones que resuelvan subconjuntos de problemas específicos. La evaluación de lenguajes de alto

nivel para describir las funcionalidades deseadas en una red de datos, demuestra que su utilización puede traer grandes beneficios en múltiples áreas de las telecomunicaciones.

La adopción de esta tecnología, puede ayudar a reducir los costos de diseño, implementación y mantenimiento de las redes de datos y los asociados a la capacitación de los recursos humanos.

La utilización de este enfoque arquitectónico para el diseño de aplicaciones de red, favorece la utilización de principios y herramientas de la Ingeniería de software.

La aplicación de la tecnología en el ámbito educativo, puede favorecer la comprensión del funcionamiento de las redes de datos, permitiendo el aprendizaje y experimentación a través del desarrollo de aplicaciones que implementen las funciones objeto del tema de estudio.

TRABAJO FUTURO

Algunas de las líneas de investigación y trabajos futuros incluyen:

- La implementación del descubrimiento y mantenimiento de los dispositivos finales a la topología de red.
- La implementación de diferentes algoritmos de grafos sobre la topología con el objetivo de ser la entrada de información para realizar tareas de Ingeniería de tráfico.
- La implementación de interfaces que permitan extender y relacionar los datos de la topología con herramientas externas como inventario y monitoreo.
- La implementación del producto en dispositivos físicos reales a fin de evaluar su funcionamiento y desempeño.
- La investigación sobre las posibilidades de las nuevas especificaciones respecto de tecnologías de la capa de datos como

medios ópticos y medios inalámbricos, donde la arquitectura podría brindar grandes oportunidades.

- La investigación sobre la factibilidad de utilizar arquitecturas híbridas (combinación de arquitecturas tradicionales y arquitecturas SDN), que pueden favorecer la transición de las tecnologías o que pueden ser utilizadas en conjunto para mejorar las prestaciones que cada una puede ofrecer individualmente.

- El desarrollo de modulos de aplicación que permitan realizar validaciones acerca de la completitud de las políticas de conectividad y acceso.

Parte V

APÉNDICES

A

SINTAXIS NETKAT

PROPIEDADES MÁTEMATICAS DE NETKAT

Sean:

- $f = f_1, .. , f_n$ el conjunto de los campos de un paquete.
- $p_k = \{p_1, .. , p_k\}$ un conjunto de paquetes.

Definimos un predicado **a**, **b** o **c** como:

- **id** : Identidad
- **drop** : Descarte
- **f = n** : Comparación
- **a or b** : Disyunción
- **a and b** : Conjunción
- **not a** : Negación

Una política **p**, **q** o **r** como:

- **filter a** : Filtro
- **f := n** : Modificación
- **p + q** : Composición paralela
- **p ; q** : Composición secuencial

Y a partir de estas definiciones los siguientes axiomas son válidos:

- **a or (b and c) ≡ (a or b) and (a or c)**
- **a or id ≡ id**
- **a or not a ≡ id**
- **a and b ≡ b and a**
- **a and not a ≡ drop**
- **a and a ≡ a**
- **p + (q + r) ≡ (p + q) + r**
- **p + q ≡ q +p**
- **p + drop ≡ p**
- **p + p ≡ p**
- **p ; (q ; r) ≡ (p ; q) ; r**
- **id ; p ≡ p**
- **p ; id ≡ p**
- **p ; (q + r) ≡ p ; q + p ; r**
- **(p + q) ; r ≡ p ; r + q ; r**
- **drop ; p ≡ drop**
- **p ; drop ≡ drop**

SINTAXIS DE NETKAT

Para describir la sintaxis de NetKat utilizaremos una sintaxis de tipo BNF omitiendo las formalidades estrictas del metalenguaje.

Los números enteros pueden ser representados en notación decimal o hexadecimal siempre y cuando agreguemos "ox" al comienzo del número. Y utilizaremos la notación "h" para especificar números hexadecimales y "x" para especificar números enteros.

Un predicado es una condición booleana que deben tener los paquetes de datos.

La política es la que genera las acciones sobre los paquetes de datos que cumplen algún predicado.

Tipos

```
<switch-id> ::= 64-bit integer
<port-id> ::= 16-bit integer
<mac-address> ::= xx:xx:xx:xx:xx:xx
<ip-add> ::= xxx.xxx.xxx.xxx
<mask> ::= 1 ... 32
<ip-masked> ::=<ip-add>/<mask>
<vlan-id> ::= none | 12-bit integer
<tcp-port> ::= 16-bit integer
<vlan-pcp> ::= 16-bit integer
<frame-type> ::=
    arp (* abreviatura de 0x806 *)
    | ip (* abreviatura de 0x800 *)
    | 8-bit integer
<ip-protocol> ::=
    icmp (* abreviatura de 0x01 *)
    | tcp (* abreviatura for 0x06 *)
    | udp (* abreviatura for 0x11 *)
    | 8-bit integer
```

Predicado

```
<pred-atom> ::= ( <pred> )
      | true
      | false
      | switch = <switch-id>
      | port = <port-id>
      | vlanId = <vlan-id>
      | vlanPcp = <vlan-pcp>
      | ethTyp = <frame-type>
      | ipProto = <ip-protocol>
      | tcpSrcPort = <tcp-port>
      | tcpDstPort = <tcp-port>
      | ethSrc = <mac-address>
      | ethDst = <mac-address>
      | ip4Src = <ip-masked> | <ip-add>
      | ip4Dst = <ip-masked> | <ip-add>

<not-pred> ::= <pred-atom>
      | not <not-pred>

<and-pred> ::= <not-pred>
      | <and-pred> and <not-pred>

<or-pred> ::= <and-pred>
      | <or-pred> or <and-pred>

<pred> ::= <or-pred>
```

Política

```
<pol-atom> ::= ( <pol> )
      | id
      | drop
      | filter <pred>
      | switch := <switch-id>
      | port := <port-id>
      | vlan := <vlan-id>
      | vlanPcp := <vlan-pcp>
      | ethTyp := <frame-type>
      | ipProto := <ip-protocol>
      | tcpSrcPort := <tcp-port>
      | tcpDstPort := <tcp-port>
      | ethSrc := <mac-address>
      | ethDst := <mac-address>
      | ip4Src := <ip-address>
      | ip4Dst := <ip-address>

<seq-pol> ::= <pol-atom>
      | <seq-pol> ; <pol-atom>

<union-pol> ::= <seq-pol>
      | <union-pol> + <seq-pol>

<cond-pol> ::= <union-pol> |
     if <pred> then <cond-pol> else <cond-pol>

<pol> ::= <cond-pol>

<program> ::= <pol>
```

BIBLIOGRAFÍA

[1] K. W. R. James F. Kurose, *Computer Networking a Top-Down Approach 6th Edition*. 2013. ISBN: 978-0-13-285620-1.

[2] C. B. Paul Göransson, *Software Defined Networks: A Comprehensive Approach*. MK, 2014. ISBN: 978-0-12-416675-2.

[3] Internet Research Task Force (IRTF), "Request For Comments 7426," 2015. http://www.rfc-base.org/txt/rfc-7426.txt.

[4] Open Networking Fundation, "OpenFlow Switch Specification 1.0," 2009. https://opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf.

[5] Taylor & Francis Group, *Network Innovation through OpenFlow and SDN*. Taylor & Francis Group, 2013. ISBN: 978-1-4665-7210-2.

[6] U. D. D. Idris Z. Bholebawa, Rakesh Kumar Jha, "Performance Analysis of Proposed Network Architecture: OpenFlow vs. Traditional Network," *International Journal of Computer Science and Information Security (IJCSIS)*, 2016.

[7] Jay Shah, "Implementation and Performance Analysis of Firewall on Open vSwitch," https://pdfs.semanticscholar.org/9bb1/4f91b269e72a98d9063d22da2092dc5f552d.pdf.

[8] Open Networking Fundation, "OpenFlow Switch Specification 1.5," 2014. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[9] HashiCorp, *Vagrant Documentation*. https://www.vagrantup.com/docs/.

[10] RedHat, *Ansible Documentation.* http://docs.ansible.com/ansible/index.html.

[11] Oracle, *VirtualBox Documentation.* https://www.virtualbox.org/wiki/Documentation.

[12] AngularJS, *AngularJS Tutorial.* https://docs.angularjs.org/tutorial.

[13] AngularJS, *AngularJS API Documentation.* https://docs.angularjs.org/api.

[14] BootStrap, *BootStrap Components Documentation.* http://getbootstrap.com/components/.

[15] BootStrap, *BootStrap CSS Documentation.* http://getbootstrap.com/css/.

[16] D3.js, *D3.js Documentation.* https://github.com/d3/d3/wiki.

[17] M. L. M. Nate Foster, Rob Harrison, "Frenetic: A network programming language," http://frenetic-lang.org/publications/frenetic-icfp11.pdf.

[18] M. L. M. Nate Foster, Rob Harrison, "Frenetic: A high-level language for openflow networks," http://frenetic-lang.org/publications/frenetic-presto10.pdf.

[19] W. R. J. Harrison, "Frenetic: A network programming language, masters thesis," Master's thesis, 2009. http://frenetic-lang.org/publications/harrison-mse-thesis.pdf.

[20] NetworkX, *NetworkX 1.11 Documentation.* http://networkx.readthedocs.io/en/networkx-1.11/.

[21] Tornado, *Tornado Web Server Documentation.* http://www.tornadoweb.org/en/stable/.

[22] S. S. Arsalan Tavakoli, Martin Casado, "Applying nox to the datacenter," https://www.cs.duke.edu/courses/cps296.4/fall13/838-CloudPapers/hotnets2009-final103.pdf.

[23] J. P. Natasha Gude, Teemu Koponen, "Nox: Towards an operating system for networks," http://benpfaff.org/papers/nox.pdf.

[24] J. S. Sukhveer Kaur and N. S. Ghumman, "Network programmability using pox controller," http://www.sbsstc.ac.in/icccs2014/Papers/Paper28.pdf.

[25] G. A. Anderson Carolyn Jane, Foster Nate, "Netkat: Semantic foundations for networks," 2013. https://ecommons.cornell.edu/bitstream/handle/1813/34445/netkat-tech-report.pdf?sequence=2&isAllowed=y.

[26] L. Foundation, *OpenVSwitch Documentation.* http://docs.openvswitch.org/en/latest/.

[27] Mininet, *Mininet Documentation.* https://github.com/mininet/mininet/wiki/Documentation.

[28] Mininet, *Miniedit Source Code.* https://github.com/mininet/mininet/blob/master/examples/miniedit.py.