UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

Enrutamiento de paquetes en Redes Definidas por Software mediante Aprendizaje Automático

Santiago Rendón Bernot

Ingeniería en Sistemas

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Sistemas

Quito, 7 de mayo de 2020

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

HOJA DE CALIFICACIÓN
DE TRABAJO DE INTREGRACIÓN CURRICULAR

Enrutamiento de paquetes en Redes Definidas por Software mediante

Aprendizaje Automático

Santiago Rendón Bernot

Calificación:
Nombre del profesor, Título académico Noel Pérez, Ph.D.

Firma del profesor: _____

Quito, 7 de mayo de 2020

DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma del estudiante: _____

Nombres y apellidos: Santiago Rendón Bernot

Código: 00123786

Cedula de identidad: 1715315188

Lugar y fecha: Quito, 7 de mayo de 2020

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

La complejidad de las redes ha ido incrementando durante el transcurso de los años y el paradigma de las redes definidas por software (SDN) está contribuyendo a resolver las limitaciones y determinar nuevos requerimientos de las redes. Por esta razón, en este trabajo, se propone experimentar con un nuevo enfoque para mejorar el enrutamiento de paquetes en una red definida por software mediante la utilización de modelos de aprendizaje automático. Para el proceso de experimentación, se diseñó e implemento una topología de red básica en la cual se pueden realizar todas las pruebas. Al no disponer de equipos de red físicos compartibles con el paradigma de las redes definidas por software para crear la infraestructura de red, se utilizó Virtual Networks over LinuX (VNX) la cual es una herramienta de virtualización de red. Para poder gestionar el tráfico y lograr enrutarlo se utilizó el controlador SDN ryu, el cual utiliza OpenFlow como protocolo para la administración de la red. Para poder crear el data set de entrenamiento se diseñó e implemento un enrutamiento estático. Para este fin, se consideró como atributo esencial el ancho de banda causado por el tráfico generado en cada uno de los enlaces y, mediante este atributo, decidir cuál es el mejor puerto de salida. Por otro lado, se creó un data set de prueba para que los modelos de aprendizaje automático sean probados bajo las mismas circunstancias. Se escogieron dos modelos de aprendizaje automático, vecino más cercano (KNN) y redes neuronales artificiales (ANN), los cuales fueron entrenados y probados. Luego de realizar las pruebas respectivas, se determinó que al utilizar aprendizaje supervisado los paquetes son enrutados por los enlaces que presentan menor congestión, es decir, menor tráfico. En conclusión, ANN tuvo mejores resultados enrutando el tráfico en comparación con KNN.

Palabras clave: Redes definidas por software, Enrutamiento de paquetes, Ryu, Virtual Networks over Linux, K vecino más cercano, Red neuronal artificial.

Abstract

Network complexity has increased over the years and the paradigm of software-defined networks (SDN) is helping overcome limitations and determine new network requirements. For this reason, this study will analyze if, in any way, packet routing can be improved by using machine learning models. In order to demonstrate this, a basic network topology was designed and used to perform all tests. Not having physical network components compatible with the paradigm of software-defined networks to create the necessary network infrastructure, Virtual Networks over Linux (VNX) was used, which is a network visualization tool. To manage the traffic and manage how to route it, *ryu*, an SDN controller was used. *Ryu* uses OpenFlow as a protocol for network administration. To create the training data set, a static routing was designed, the bandwidth caused by the congestion generated on each of the links was taken as an essential attribute and therefore decide which is the best output port. On the other hand, a test data set was also created so that both machine learning models could be tested under equal circumstances. The two chosen machine learning models were k nearest neighbor (KNN) and artificial neural networks (ANN). These models were both trained and tested. It was determined that by using supervised learning, packages are routed by those links where there is less congestion. In conclusion, ANN had better results routing traffic than KNN.

Key words: Software defined networks, Packet routing, Ryu, Virtual Networks over Linux, K nearest neighbor, Artificial neural network.

Tabla de Contenidos

Índices de Figuras

1. Introducción

1.1 Descripción del problema

El desarrollo de las redes ha ido incrementando con el pasar de los años y el paradigma de las redes definidas por software (SDN) está ayudando a resolver las limitaciones que actualmente tienen las redes tradicionales. Esta siguiente generación de arquitectura de red, mediante la programabilidad, contribuye a que las redes sean más confiables, seguras y eficientes. Mediante esto los requerimientos de la red pueden ser configurados de manera más fácil sin necesitar infraestructura física o elementos de red de bajo nivel (Badotra y Singh, 2017).

Con lo dicho anteriormente es necesario determinar qué factores pueden contribuir a que una SDN proporcione ventajas en comparación con una red tradicional. Para ello se analizará si existe alguna mejora en cuanto al enrutamiento de los paquetes en la red mediante el uso de algoritmos basados en aprendizaje automático. Es necesario diseñar un mecanismo de enrutamiento que permita mejorar el control de tráfico en la red. Para este fin, se debe analizar los factores relacionados con la eficiencia del enrutamiento en la red e identificar los mejores enfoques. De esta manera, se pretende tener una visión general del estado de los enlaces en la red y enrutar el tráfico por aquellos enlaces que no se encuentren saturados. Como resultado, se balanceará la carga de tráfico en todos los enlaces de la red, optimizando el consumo de ancho de banda.

1.2 Objetivo General

Mejorar la toma de decisiones en el enrutamiento de paquetes en redes definidas por software mediante el uso de modelos de aprendizaje automático.

1.3 Objetivos Específicos

- Diseñar e implementar una topología de red definida por software para realizar experimentos de enrutamiento basado en aprendizaje automático
- Crear un data set de entrenamiento para algoritmos de aprendizaje automático supervisados.
- Proponer e implementar modelos de ANN y KNN para el enrutamiento de paquetes.
- Realizar pruebas sobre la topología de red implementada con el objetivo de observar comportamiento del enrutamiento y obtener resultados en diferentes contextos de ejecución.
- Analizar los resultados obtenidos y proponer futuras líneas de investigación.

2. Estado del Arte

2.1 Redes Definidas por Software

Una red definida por software (SDN - Software Defined Network) es una tecnología que se enfoca en la programabilidad de la red, contribuyendo a que se tenga la capacidad de controlar, cambiar y administrar el comportamiento de la red de manera dinámica a través de software. (Hakiri et al., 2014). De esta manera, se pueden tener redes más flexibles y centralizadas atribuyendo a que estas sean más consistentes. Una SDN tiene dos principios principales, el primero es que separa el plano de control con el plano de datos. El plano de control es aquel que contiene toda la lógica, mientras que el plano de datos es el que contiene toda la infraestructura de red de bajo nivel. Por otro lado, el segundo principio es que el plano de control actúa como un cerebro, por lo que tiene un control inmediato de los datos a ser trasmitidos y así, los elementos de red pueden ser manipulados dependiendo de las necesidades (Badotra y Singh, 2017).

La arquitectura de una SDN, sigue el esquema que se muestra en la Figura 1. Existen dos interfaces conocidas como Northbound y Southbound, estas pertenecen a distintas capas en la arquitectura, pero se encuentran constantemente en interacción. Un controlador SDN implementa la arquitectura mostrada en la Figura 1. Se ha escogido el controlador Ryu ya que el tiempo requerido para implementar prototipos es reducido. Ryu proporciona una interfaz Northbound con Python mientras que en la interfaz SouthBound se puede escoger distintas versiones de OpenFlow. Para este trabajo, se ha utilizado OpenFlow 1.3. OpenFlow es un protocolo de comunicación que da acceso al plano de reenvió que puede ser configurado y controlado por un controlador externo. Por otro lado, el funcionamiento de la SDN se presenta en la Figura 2. Como se observa, existen elementos de red que pueden contener una o más tablas de flujo, las cuales, a su vez, tienen entradas

de flujo. Cada una se encarga de determinar cómo los paquetes deben de ser procesados y posteriormente enviados dentro de la red (Flores, 2018).

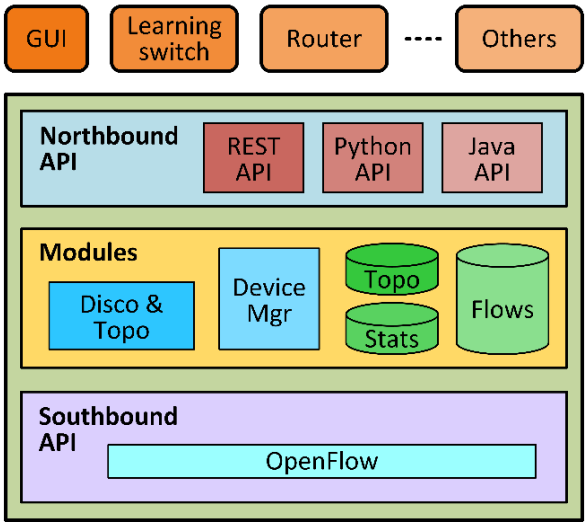

Figura #1: Arquitectura de SDN

(Goransson, Black, & Culver, 2017)

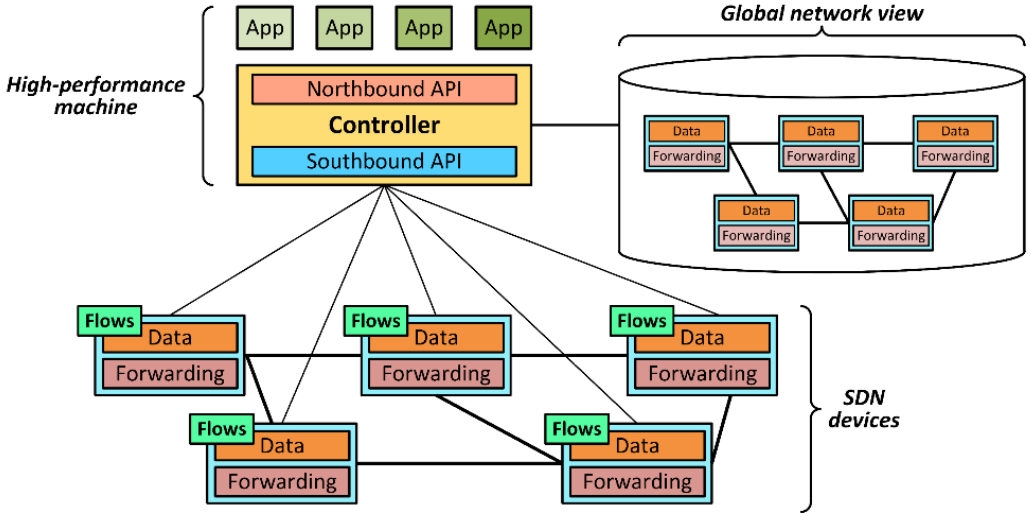

Figura #2: Funcionamiento de SDN

(Goransson, Black, & Culver, 2017)

2.2 Open vSwitch

Un Open vSwitch (OVS) es una implementación de un switch virtual multicapa. Este proporciona una pila de conmutación para entornos de virtualización de hardware. Su utilización permite un mejor manejo de las interfaces de administración estándar, obteniendo un control programático sobre todo el switch. En la Figura 3, se puede apreciar de mejor manera la arquitectura de un OVS. Este puede ser utilizado ya sea como un switch definido por software o como el stack de un switch dedicado (The Linux Foundation, 2016).

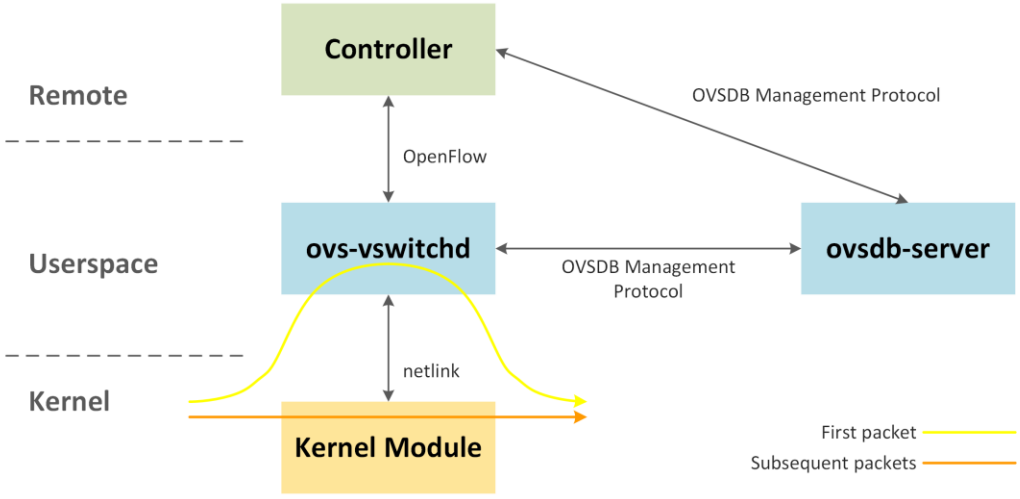

Figura #3: Open vSwitch
(PicOS, 2018)

2.3 Aprendizaje Automático (ML – Machine Learning)

Machine Learning (ML) es el estudio de la generación de algoritmos los cuales buscan mejorar la automatización con base en la experiencia. Para ello, existen distintos modelos que utilizan información de entrenamiento para poder realizar predicciones o decisiones sin ser explícitamente programadas. De esta manera, el modelo es capaz de tomar decisiones dependiendo

de las distintas variables que sean determinantes (Liu et al., 2018). Mediante la utilización de ML será posible predecir cual será la manera más eficiente de enrutar los paquetes dentro de la red. Para el proyecto se propone utilizar el modelo de redes neuronales artificiales y K vecino más cercano.

2.3.1 Redes Neuronales Artificiales (ANN – Artificial Neural Network)

Las redes neuronales (ANN) es un algoritmo de Machine Learning supervisado que simula una neurona biológica. Esta recibe un valor de entrada el cual es combinado con los estados de cada neurona y se produce un resultado. La representación de una red neuronal se presenta en la Figura 4. En esta se observan círculos, los cuales son nodos que representan neuronas, y estos se encuentran conectados entre sí representando la red (Mishra y Srivastava, 2014).

Los nodos reciben información, la combinan con su estado interno y producen una salida. Las neuronas pueden estar en tres distintas capas (entrada, ocultas y salida). En la Figura 4 se observa cómo está compuesta una red neuronal con sus distintas capas. La capa de entrada es la que recibe los datos externos. Las capas ocultas son aquellas que se encuentran entre la capa de entrada y de salida y estas se encargan de obtener un conjunto de entradas ponderadas y producen una salida a través de una función de activación. Por último, la capa de salida proporciona el resultado de la predicción (Mishra y Srivastava, 2014).

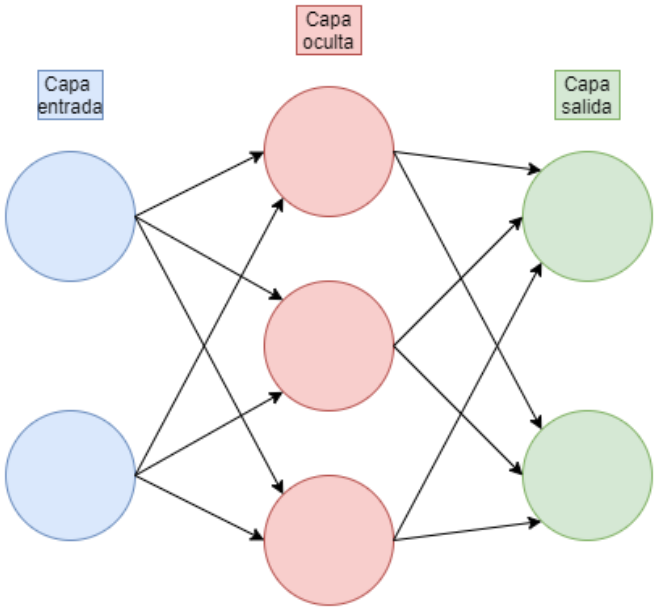

Figura #4: ANN

En un principio, a las conexiones entre nodos se les define pesos aleatorios y a los nodos que se encuentran en la capa oculta se les coloca un bias aleatorio. En la fase de entrenamiento, una vez que se obtiene un resultado de salida y al determinar un valor de error, realiza un procedimiento de *backpropagation* para ajustar los pesos y los bias. De esta manera, los pesos entre neuronas y los bias serán más precisos y se obtendrán una mejor predicción. (Mishra y Srivastava, 2014).

2.3.2 K vecino más cercano (KNN – K-Nearest Neighbors)

El algoritmo K vecino más cercano (KNN) es un método de ML supervisado que consiste en un data set de entrenamiento, el cual define un espacio de distintas características. En la Figura 5 se presenta un ejemplo con dos características distintas. Las características del data set son agrupadas en distintos grupos dependiendo de sus clases. Cuando se desea realizar una predicción se calcula la distancia (por ejemplo, Euclidiana) al k punto más cercano y se determina, mediante votación entre los puntos, a que clase pertenece (Zhao, et al., 2019).

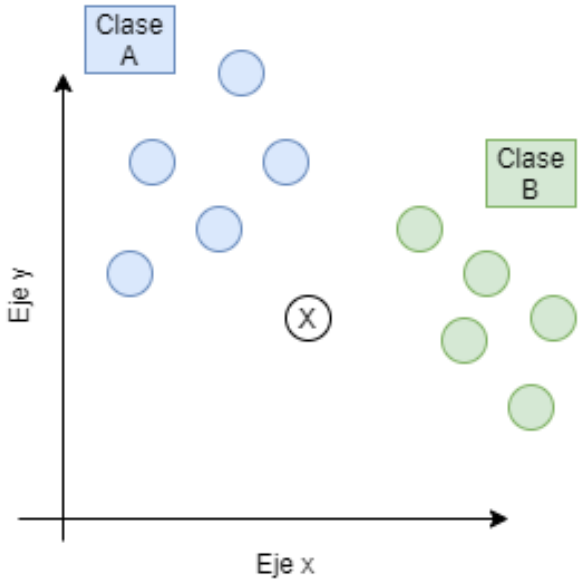

Figura #5: KNN

3. Desarrollo del prototipo

3.1 Requerimientos

Para el proceso de experimentación, es necesario disponer de una infraestructura de red con dispositivos físicos que soporten el paradigma SDN. Al no disponer de tales dispositivos, se ha optado por utilizar entornos virtuales. En este contexto, se ha decidido a utilizar una herramienta de virtualización de red llamada VNX (Virtual Networks over LinuX). VNX ayuda a construir escenarios de red automáticamente mediante archivos XML (Galán et al. 2019). De esta manera se podrá realizar la topología de red teniendo distintos dispositivos o sistemas finales y OVS's y así realizar las pruebas necesarias. Para los sistemas finales, se utilizará máquinas virtuales basadas en LXC. LXC es un método de virtualización de nivel de sistema operativo que permite ejecutar múltiples sistemas Linux aislados (contenedores) simplemente utilizando el núcleo de Linux (Fernández, et al., 2011).

En la Figura 6, se presenta el flujo de operación de VNX. El primer paso es el diseño de la topología de la red por parte del usuario. Una vez planteada la topología, se realiza las especificaciones del escenario en XML. A continuación, se procesa las especificaciones, se las crea y libera sobre la maquina host. Por último, se crea el escenario de red virtual y el usuario tiene acceso a todos los sistemas finales (Fernández, et al., 2011).

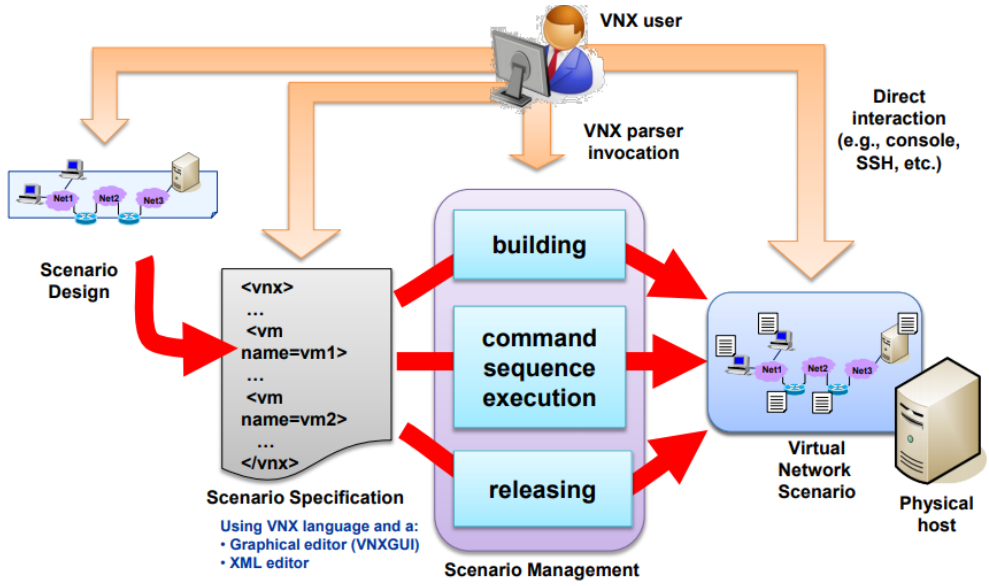

Figura #6: Flujo de operación VNX

(Fernández et al., 2011).

3.2 Diseño

3.2.1 Topología de red

Como topología de red inicial, se ha decidido considerar una red redundante compuesta por tres nodos de conmutación. Cada nodo de conmutación está representado por un OVS y en cada uno de ellos se conecta un sistema final. En la Figura 7, se observa la topología anteriormente explicada. En ella se encuentran tres OVS's (Net0, Net1 y Net2), cada uno con su sistema final. Los sistemas finales son designados con la letra H seguida de un número, por ejemplo H1. Es importante recordar, que los sistemas finales son máquinas virtuales LXC. Los números que se encuentran en los enlaces de red, representan los números de puerto de los diferentes OVS's que forman parte de la topología.

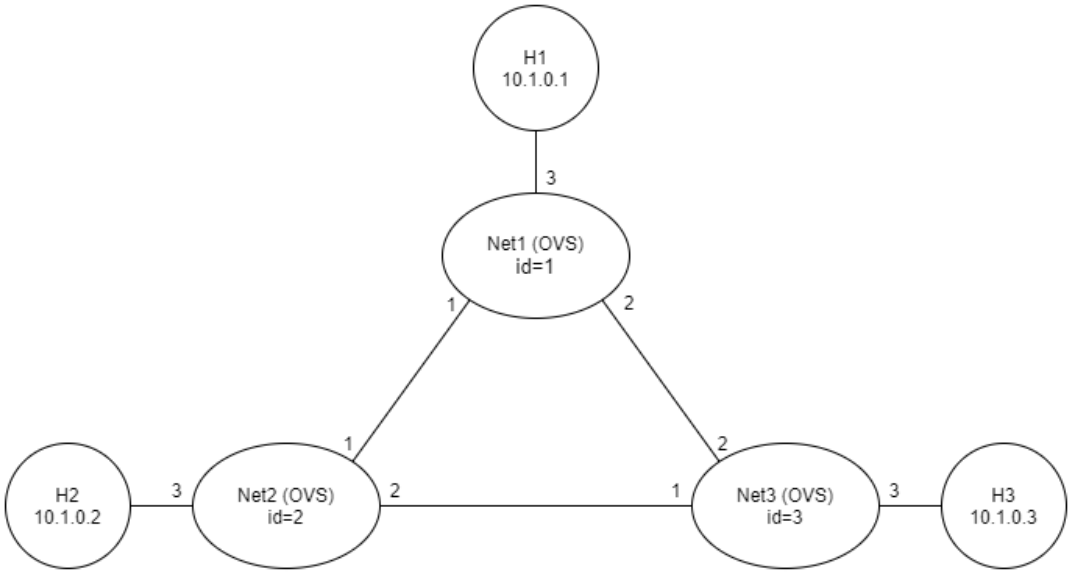

Figura #7: Topología de red

La topología mostrada en la Figura 7, al ser simple, permite comprender de mejor manera el proceso de enrutamiento de paquetes. Al analizar la topología, se observa que el tráfico que se origina en *H1* tiene dos opciones para llegar hasta *H2*. La primera es directa a través del enlace entre *Net1* y *Net3* y la otra es indirecta a través de *Net2*. Por otra parte, si el enlace *Net1* y *Net3* esta congestionado y *H1* desea enviar un paquete a *H3*, el enrutamiento puede realizarse por la ruta *Net1-Net2-Net3*. De esta manera, se facilita el análisis de los resultados mediante la utilización de aprendizaje automático.

3.2.2 Controlador

Para controlar el tráfico dentro de la SDN es necesario tener un controlador. Como controlador se utilizará Ryu, el cual, como se mencionó anteriormente, utiliza OpenFlow como protocolo para el control de los OVS's que forman parte de la red. Ryu proporciona un API para el desarrollo de aplicaciones (apps) que se encargan de determinar el comportamiento de la red mediante la instalación de flujos en los OVS's.

Ryu tiene dos métodos importantes para el desarrollo de la aplicación basada en aprendizaje automático que se utilizará en este proyecto. El primero es *packet_in_handler*, el cual se encarga de recibir paquetes de tráfico desconocido (packet_in) que envía el OVS. Posteriormente, al analizar este packet_in, la aplicación que ejecuta Ryu determina cómo realizar el enrutamiento. Al analizar el *packet_*in se puede saber cuál es la IP de origen y destino, el puerto de entrada, *datapath id* (ID del OVS), entre otra información. Este método será utilizado para determinar el puerto de salida para el enrutamiento más eficiente del paquete. Por otro lado, el segundo método *_port_stats_reply_handler*, se encarga de proporcionar estadísticas de los puertos de cada OVS de forma periódica. En esas estadísticas se encuentra el *datapath id*, número de puerto, número de paquetes recibidos y enviados, número de bytes recibidos y enviados, cuenta de errores de paquetes enviados y recibidos (RYU Project Team, s/f.). Con estas estadísticas será determinar el nivel de congestión del tráfico en la red y determinar el mejor puerto de salida para el enrutamiento de los paquetes.

3.3 Implementación

3.3.1 Topología de red

Tomando en cuenta el proceso de creación de la red virtual con VNX y el diseño de topología de red planteada se procedió a la creación de la especificación de la red en XML. Para ello, primero se creó cada uno de los OVS como se muestra en la Figura 8. Podemos ver que al elemento de red se le asigna un nombre, un modo (en este caso es OVS), un controlador el cual es ryu que se encuentra ejecutándose en *127.0.0.1:6633* y la versión de OpenFlow que utilizará. Por último, se realiza la conexión a otro OVS especificando un nombre de enlace, a que OVS estará conectado y el tipo de conexión. El tipo de conexión *veth* es para utilizar una interfaz de ethernet

virtual la cual facilita la captura el tráfico y el control de ancho de banda mediante controladores de tráfico Linux (Fernández, et al., 2011).

```
<net name="Net1" mode="openvswitch" controller="tcp:127.0.0.1:6633"
of_version="OpenFlow13" >
   <connection name='net1-net2' net='Net2' type="veth">
   </connection>
</net>
```

Figura #8: Creación de OVS

Una vez especificados los OVS's, se procede a la creación de sistemas finales. En la Figura 9 se puede observar cómo se crea un sistema final. Primero se le debe de especificar un nombre, de que tipo (en este caso es de LXC) y su sistema operativo. A continuación, se especifica el directorio donde se encuentra el filesystem. Se realiza la conexión entre el sistema final y un OVS (en este caso Net1), se especifica el IPv4 y su gateway. Para todos los sistemas finales se utilizó el mismo gateway.

```
<vm name="H1" type="lxc" arch="x86_64">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu64</filesystem>
  <if id="1" net="Net1">
    <ipv4>10.1.0.1/24</ipv4>
  </if>
  <route type="ipv4" gw="10.1.0.254">default</route>
</vm>
```

Figura #9: Creación de sistema final

Antes de ejecutar la topología de red, se instaló *iperf* en el root filesystem (Fernández, et al., 2011). *Iperf* es una herramienta de red utilizada para medir el rendimiento de la red (Chapman, 2016). Al ser una red virtual, las conexiones son interproceso por lo que se alcanza un ancho de banda aproximadamente de 30 Gbits/seg. Con el objetivo de emular fielmente una red de datos, es necesario controlar el ancho de banda de los enlaces entre los OVS's. Para este fin, se utilizará el *traffic control* de Linux, utilidad que permite realizar un conformado de tráfico al nivel de kernel

(Hubert, et al., 2019). El conformado permitirá establecer un ancho de banda por interfaz. Para ello, se está utilizando *Hierarchy Token Bucket* el cual es nos permite establecer límites en el ancho de banda. En este caso se limitará el ancho de banda a 10 Gbits/seg.

3.3.2 Modelos de aprendizaje automático

En este caso, por ser Ryu un framework escrito en Python, se ha decidido que los modelos de aprendizaje automático se encuentren en el mismo lenguaje de programación. En Python, existe la librearía *sklearn*, la cual tiene implementado distintos modelos de aprendizaje automático. *sklearn* será utilizado para implementar los modelos de ANN, KNN, normalizar los datos, entrenar los modelos y realizar las predicciones. Para esto, se desarrolló scripts para ANN y KNN que sean capaces de realizar las funcionalidades anteriormente mencionadas. Antes de normalizar y entrenar a los modelos los datos son aleatoriamente mezclados para que el orden de los datos no afecte el aprendizaje y los resultados sean independientes al orden. Por último, los modelos de aprendizaje determinarían el mejor puerto de salida para el enrutamiento del paquete.

3.3.3 Aplicación SDN

En el controlador, se ha desarrollado una aplicación que utiliza el método "packet_in_handler" para realizar el enrutamiento de paquetes que no incorpora aprendizaje automático. La aplicación sigue un enfoque de enrutamiento "estático", donde dependiendo del *datapath id* del OVS, la IP origen y la IP destino se determina el camino más corto y posteriormente se escoge el puerto de salida del OVS para el paquete.

La aplicación también utiliza el método *_port_stats_reply_handler* para determinar el ancho de banda en cada uno de los enlaces entre los OVS’s. Una solicitud de estadísticas de puertos se envía a los OVS's desde la aplicación que se ejecuta en Ryu de forma periódica cada tres segundos. Al recibir la respectiva respuesta, se considera el total de bytes de entrada y de salida de

cada puerto. Al conocer el total de bytes de la ejecución previa y de la ejecución actual se puede calcular cual es el ancho de banda. Para ello, se resta el valor de la ejecución previa del valor de la ejecución actual y se divide para el tiempo transcurrido. El resultado obtenido representa la cantidad de bytes transmitidos por unidad de tiempo.

3.3.4 Data set para modelos de aprendizaje

La creación del data set es muy importante para que las predicciones realizadas tengan un alto grado de exactitud. Para la creación del data set de enrutamiento, se ha considerado los siguientes parámetros: la IP de origen y destino, *datapath id* del OVS, tráfico entre *Net1-Net2*, *Net1-Net3* y *Net2-Net3* y el puerto de salida representa el resultado esperado. Adicionalmente, se ha dividido el enrutamiento en dos tipos, enrutamiento directo e indirecto los cuales son enrutamientos estáticos (Amaral et al., 2016). En la Figura 10, se muestra un ejemplo de enrutamiento directo e indirecto. Consideremos que *H1* desea enviar un paquete a *H2*, el enrutamiento directo es *Net1-Net2* (color azul) mientras que el enrutamiento indirecto es *Net1-Net3-Net2* (color rojo). Si existe una mayor congestión en el enrutamiento directo, el paquete es enviado por el enrutamiento indirecto. De esta manera se determina el puerto de salida del paquete.

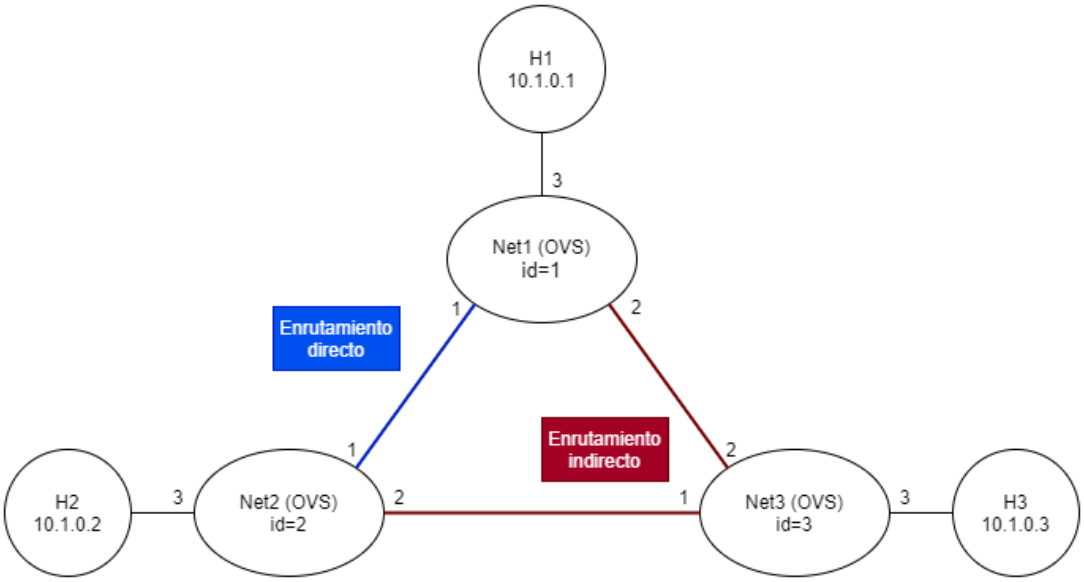

Figura #10: Enrutamiento directo e indirecto

A partir de lo anterior, se generó distinto tráfico entre los enlaces y se comenzó a recolectar los datos de la siguiente manera:

1) *_port_stats_reply_handler* es ejecutado.
2) Se calcula la congestión en cada uno de los enlaces.
3) Para cada OVS se crea un supuesto "mensaje" entre todos los sistemas finales.
4) Se determina que tipo de enrutamiento es el óptimo (directo o indirecto).
5) Se añaden los parámetros con el puerto de salida al data set

Al realizar la recolección de datos de esta forma se asegura que haya la misma cantidad de datos recolectados para cada uno de los posibles mensajes que puede haber en cada OVS. Una vez que se terminó de generar distinto tráfico en los enlaces se logró recolectar un total de 11250 datos de entrenamiento.

3.4 Resultados

Luego de realizar el procedo de entrenamiento de los modelos, se comenzó a realizar pruebas generando tráfico en la red y guardando la información del tráfico. A su vez, se generó el data set de aprendizaje. De esta forma, ambos modelos fueron probados bajo las mismas circunstancias y pudieron ser comparados. Idealmente, cualquiera de los dos modelos debe de ser agregado al controlador y utilizarlo en el método *packet_in_handler, para* determinar el mejor puerto de salida del paquete. Al concluir el proceso de generación de tráfico se obtuvo un total de 2250 datos para realizar las pruebas con los modelos de aprendizaje automático.

3.4.1 Resultados con ANN

Para determinar cuál es la configuración de la red neuronal óptima, se decidió usar dos capas ocultas ya que al emplear más de dos no afecta de gran manera a los resultados (Mishra y Srivastava, 2014). Para ello, se hizo que cada capa tenga un numero de nodos de 5 a 300, realizando todas las combinaciones posibles y aumentando los nodos de 5 en 5, determinando cuál de todas las combinaciones tiene el porcentaje de precisión más alto. Al realizar esta prueba se obtuvo que la red neuronal con 30 nodos en la primera capa y 25 nodos en la segunda capa tiene un porcentaje de precisión de 99.82 %. Esto quiere decir que al utilizar este modelo de aprendizaje automático se logró obtener que el 99.82% de los paquetes sean enrutados óptimamente.

3.4.2 Resultados con KNN

Para determinar cuál es el valor óptimo de *k,* se encontró el porcentaje de precisión desde *k=1* hasta *k=20* utilizando el data set de prueba. Realizando estas pruebas, se obtuvieron los porcentajes que se muestra en la Figura 11. En la figura podemos ver, que el mayor cambio de pendiente ocurre en *k=1* y *k=2*. Posteriormente, los cambios de pendiente son aproximadamente iguales. De esta manera, se obtiene un comportamiento óptimo es cuando *k=3*. Con este valor de

k se obtuvo un porcentaje de precisión de 96.31 % lo cual quiere decir que el 96.31% de los paquetes fueron enrutados por los enlaces óptimos.

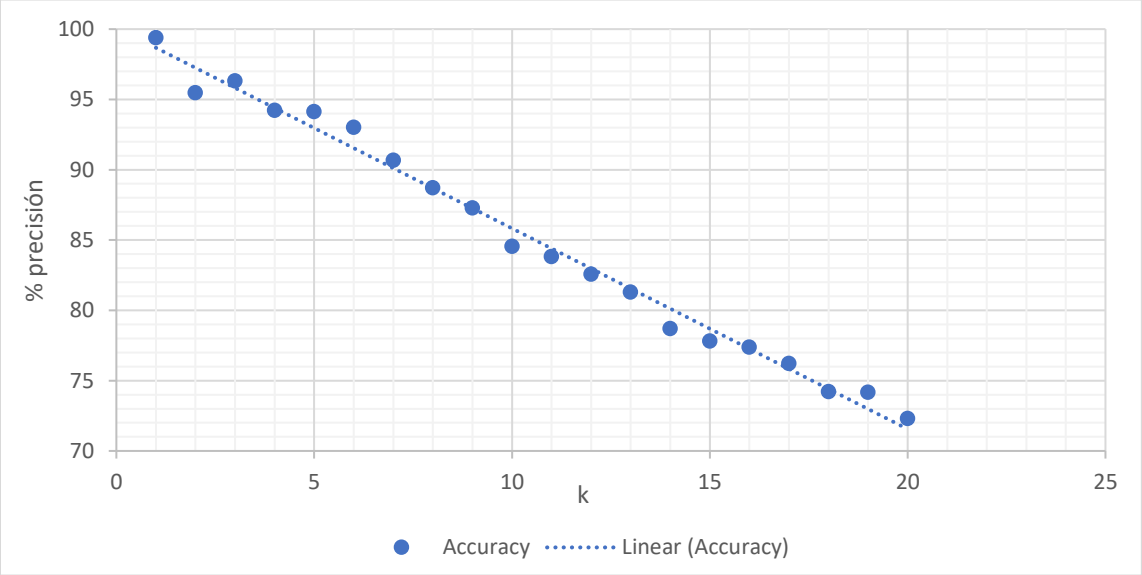

Figura #11: Resultados para distintos k's para KNN

4. Conclusiones y trabajo futuro

4.1 Conclusiones

En conclusión, los objetivos del proyecto fueron logrados mediante la construcción de un prototipo que realice el enrutamiento de los paquetes utilizando aprendizaje automático. Al no tener los recursos para construir la red, VNX fue una gran herramienta para construir la red de manera virtual. Por otro lado, Ryu como controlador fue una gran herramienta para poder enrutar y determinar las estadísticas necesarias para poder realizar en enrutamiento óptimo de los paquetes.

Para la creación de los data set de entrenamiento y prueba, se utilizaron rutas estáticas para de esta manera poder comprobar si el uso de aprendizaje automático contribuye a un enrutamiento más eficiente de paquetes. Al utilizar enrutamiento estático, la agregación o eliminación de más OVS's requiere modificar manualmente la aplicación que se encarga de configurar los flujos. Se puede enfrentar esta limitación se debería de considerar un enfoque dinámica para la creación de las rutas.

Mediante la utilización de los dos modelos de aprendizaje automático (ANN y KNN) se logró demostrar que los paquetes fueron enrutados por las rutas que tienen un menor tráfico. Para ANN se logró determinar cuál es la mejor configuración de la red obteniendo un 99.82% de precisión. Por otro lado, para KNN se logró determinar el *k* más optimo es 3. Esto se logró mediante la utilización de la Figura 11 observando el cambio en las pendientes. El valor de *k* tiene sentido ya que a partir de *k=3* la pendiente no varía de forma significante. Por otro lado, *k* al ser un valor pequeño no representa un alto costo computacional. Al comparar los resultados de los dos modelos de aprendizaje automático, se puede ver que ANN tiene un mejor porcentaje de precisión, y, por tanto, es más indicado para realizar el enrutamiento de paquetes dentro de la red.

4.2 Trabajo futuro

A pesar de que se logró determinar que el uso de aprendizaje automático ayuda que el enrutamiento de los paquetes sea el más optimo, durante todo el proyecto se utilizaron rutas estáticas. Como se ha mencionado anteriormente, esto presenta una limitación. Al saber que aprendizaje automático si contribuye puede ser utilizado basándose en la capa 2 o 3 de la red. Por ejemplo, se podría utilizar *Spanning Tree Protocol* para resolver los problemas de red que tienen una estructura de loop, como la utilizada (Figura 7), y reducir los tiempos de enrutamiento en dicho protocolo. Al usar aprendizaje automático con la capa 2 o 3 se podría optimizar el enrutamiento de los paquetes sin importar la topología de la red.

En este caso, solamente se utilizaron dos modelos (ANN y KNN) de aprendizaje automático, pero se podría considerar usar otros modelos y probar con cuál se obtienen mejores resultados para este tipo de proyectos. Por otro lado, se pueden utilizar modelos de aprendizaje automático que no sean supervisados. De esta manera, el modelo aprendería cual es el mejor enrutamiento mediante la experiencia y sin importar la topología podría optimizar el enrutamiento de los paquetes.

5. Referencias bibliográficas

Amaral, P., Dinis, J., Pinto, P., Bernardo, L., Tavares, J., y Mamede, H. S. (2016). Machine learning in software defined networks: Data collection and traffic classification. *IEEE 24th International Conference on Network Protocols* (ICNP): 1-5.

Badotra, S., y Singh, J. (2017). A review paper on software defined networking. *International Journal of Advanced Research in Computer Science, 8*(3).

Chapman, C. (2016). *Network Performance and Security: Testing and Analyzing Using Open Source and Low-cost Tools*. Syngress.

Fernández, D., Cordero, A., Somavilla, J., Rodríguez, J., Corchero, A., Tarrafeta, L., & Galán, F. (2011). Distributed virtual scenarios over multi-host Linux environments. *5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud* (SVM): pp. 1-8.

Flores, R. (2018). *Contribución a las arquitecturas de virtualización de funciones de red y redes definidas por software aplicadas a las redes residenciales con gestión centrada en el usuario*. Universidad Politécnica de Madrid. Tesis de doctorado. Madrid, España.

Goransson , P., . Black, C., y Culver , T. (2017) Software defined networks: a comprehensive approach. Segunda edición. Morgan Kaufmann. Massachusetts, Estados Unidos.

Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D. C., y Gayraud, T. (2014). Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks*, 75, 453-471.

Hubert, B., Maxwell, G., Mook, R., Oosterhout, M., Schoreder, P. y Spaans, J. (2019). *Linux Advanced Routing and Traffic Control: How To.* Segunda edición. Independent Publisher. Países Bajos.

Liu, W. X., Zhang, J., Liang, Z. W., Peng, L. X., y Cai, J. (2017). Content popularity prediction and caching for ICN: A deep learning approach with SDN. *IEEE access*, 6, 5075-5089.

Mishra, M., & Srivastava, M. (2014). A view of artificial neural network. *International Conference on Advances in Engineering & Technology Research* (ICAETR-2014): 1-3.

PicOS. (2018). *Introduction to Open vSwitch.* Recuperado el 24 abril 2020 desde https://docs.pica8.com/display/PicOS21116cg/Introduction+to+Open+vSwitch

RYU Project Team. (s/f). *RYU SDN Framework: Using OpenFlow 1.3.*

The Linux Foundation. (2016). *What is Open vSwitch?* Linux Foundation Collaborative Projects. Recuperado el 1 de abril de 2020 desde http://docs.openvswitch.org/en/latest/intro/what-is-ovs/

Zhao, Y., Li, Y., Zhang, X., Geng, G., Zhang, W., y Sun, Y. (2019). A Survey of Networking Applications Applying the Software Defined Networking Concept Based on Machine Learning. *IEEE Access*, 7, 95385-95405.

6. Anexos

Anexo A: Código VNX

```
<?xml version="1.0" encoding="UTF-8"?>

<!--

-->

<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
  <global>
    <version>2.0</version>
    <scenario_name>ryu_controller</scenario_name>
    <automac/>
    <vm_mgmt type="none" />
    <vm_defaults>
        <console id="0" display="no"/>
        <console id="1" display="yes"/>
    </vm_defaults>

  </global>

  <net name="Net1" mode="openvswitch" controller="tcp:127.0.0.1:6633" of_version="OpenFlow13" >
    <connection name='net1-net2' net='Net2' type="veth">
    </connection>
  </net>

  <net name="Net2" mode="openvswitch" controller="tcp:127.0.0.1:6633" of_version="OpenFlow13">
    <connection name='net2-net3' net='Net3' type="veth">
    </connection>
  </net>

   <net name="Net3" mode="openvswitch" controller="tcp:127.0.0.1:6633" of_version="OpenFlow13">
    <connection name='net3-net1' net='Net1' type="veth">
    </connection>
  </net>

  <vm name="H1" type="lxc" arch="x86_64">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc_ubuntu64</filesystem>
    <if id="1" net="Net1">
      <ipv4>10.1.0.1/24</ipv4>
    </if>
    <route type="ipv4" gw="10.1.0.254">default</route>
  </vm>
```

```
43
44    <vm name="H2" type="lxc" arch="x86_64">
45      <filesystem type="cow">/usr/share/vnx/filesystems/
    rootfs_lxc_ubuntu64</filesystem>
46      <if id="1" net="Net2">
47        <ipv4>10.1.0.2/24</ipv4>
48      </if>
49      <route type="ipv4" gw="10.1.0.254">default</route>
50    </vm>
51
52    <vm name="H3" type="lxc" arch="x86_64">
53      <filesystem type="cow">/usr/share/vnx/filesystems/
    rootfs_lxc_ubuntu64</filesystem>
54      <if id="1" net="Net3">
55        <ipv4>10.1.0.3/24</ipv4>
56      </if>
57      <route type="ipv4" gw="10.1.0.254">default</route>
58    </vm>
59
60    <host>
61      <hostif net="Net1">
62         <ipv4>10.1.0.4/24</ipv4>
63      </hostif>
64      <route type="ipv4" gw="10.1.0.254">10.1.0.0/16</route>
65    </host>
66
67  </vnx>
68
```

Anexo B: Código controlador (Ryu)

```
 1 from ryu.base import app_manager
 2 from ryu.controller import ofp_event
 3 from ryu.controller.handler import CONFIG_DISPATCHER,
   MAIN_DISPATCHER, DEAD_DISPATCHER
 4 from ryu.controller.handler import set_ev_cls
 5 from ryu.lib import hub
 6 from ryu.lib.packet import ethernet
 7 from ryu.lib.packet import packet
 8 from ryu.ofproto import ofproto_v1_3
 9
10 from ann import ANN
11 from knn import KNN
12 from port_controller import PortController
13 from stats_analyzer import StatsAnalyzer
14
15
16 class MLOVSController(app_manager.RyuApp):
17     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
18
19     def __init__(self, *args, **kwargs):
20         super(MLOVSController, self).__init__(*args, **kwargs
   )
21         self.datapaths = {}
22
23         # For port controller
24         self.__use_ANN = False
25         if self.__use_ANN:
26             print('Training ANN')
27             self.__ann = ANN('dataset.csv', False)
28             print('ANN trained')
29
30         self.__use_KNN = False
31         if self.__use_KNN and not self.__use_ANN:
32             print('Training KNN')
33             self.__knn = KNN('dataset.csv', False)
34             print('KNN trained')
35
36         self.__port_controller = PortController()
37
38         # For port stats
39         self.__port_analyser = StatsAnalyzer()
40         self.__stats_timer = 3
41
42         # Last speed stats
```

```
43          self.__last_speeds = {
44              'net1-net2': 0,
45              'net1-net3': 0,
46              'net2-net3': 0
47          }
48
49          # Starts monitor thread
50          self.monitor_thread = hub.spawn(self._monitor)
51
52      @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
    CONFIG_DISPATCHER)
53      def switch_features_handler(self, ev):
54          datapath = ev.msg.datapath
55          ofproto = datapath.ofproto
56          parser = datapath.ofproto_parser
57
58          match = parser.OFPMatch()
59          actions = [parser.OFPActionOutput(ofproto.
    OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]
60          self.add_flow(datapath, 0, match, actions, 0)
61
62      def add_flow(self, datapath, priority, match, actions,
    idle_time):
63          ofproto = datapath.ofproto
64          parser = datapath.ofproto_parser
65
66          inst = [parser.OFPInstructionActions(ofproto.
    OFPIT_APPLY_ACTIONS, actions)]
67          mod = parser.OFPFlowMod(datapath=datapath,
    idle_timeout=idle_time, priority=priority, match=match,
68                                  instructions=inst)
69          datapath.send_msg(mod)
70
71      def _monitor(self):
72          while True:
73              for dp in self.datapaths.values():
74                  self._request_stats(dp)
75              hub.sleep(self.__stats_timer)
76
77      @set_ev_cls(ofp_event.EventOFPStateChange,
78                  [MAIN_DISPATCHER, DEAD_DISPATCHER])
79      def _state_change_handler(self, ev):
80          datapath = ev.datapath
81          if ev.state == MAIN_DISPATCHER:
```

```
 82             if datapath.id not in self.datapaths:
 83                 self.logger.debug('register datapath: %016x'
    , datapath.id)
 84                 self.datapaths[datapath.id] = datapath
 85         elif ev.state == DEAD_DISPATCHER:
 86             if datapath.id in self.datapaths:
 87                 self.logger.debug('unregister datapath: %
    016x', datapath.id)
 88                 del self.datapaths[datapath.id]
 89
 90     def _request_stats(self, datapath):
 91         self.logger.debug('send stats request: %016x',
    datapath.id)
 92         ofproto = datapath.ofproto
 93         parser = datapath.ofproto_parser
 94
 95         req = parser.OFPFlowStatsRequest(datapath)
 96         datapath.send_msg(req)
 97
 98         req = parser.OFPPortStatsRequest(datapath, 0,
    ofproto.OFPP_ANY)
 99         datapath.send_msg(req)
100
101     @set_ev_cls(ofp_event.EventOFPPortStatsReply,
    MAIN_DISPATCHER)
102     def _port_stats_reply_handler(self, ev):
103         body = ev.msg.body
104
105         # Calculates new speeds
106         self.__last_speeds = self.__port_analyser.
    manage_port_stats(ev.msg.datapath.id, body, self.
    __stats_timer,
107
            self.__last_speeds)
108
109     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
110     def packet_in_handler(self, ev):
111         msg = ev.msg
112         datapath = msg.datapath
113         ofproto = datapath.ofproto
114         parser = datapath.ofproto_parser
115         in_port = msg.match['in_port']
116
117         pkt = packet.Packet(msg.data)
```

```
118         eth = pkt.get_protocols(ethernet.ethernet)[0]
119
120         dst = eth.dst
121         src = eth.src
122
123         dp_id = datapath.id
124
125         src_id = self.__port_controller.get_host_id(src)
126         dst_id = self.__port_controller.get_host_id(dst)
127
128         # Uses ANN
129         ids = self.__port_controller.get_host_ids()
130         if self.__use_ANN and src_id in ids and dst_id in
    ids:
131             out_port = int(self.__ann.predict([[src_id,
    dst_id, dp_id, self.__last_speeds['net1-net2'],
132                                                self.
    __last_speeds['net1-net3'], self.__last_speeds['net2-net3'
    ]]])[0])
133
134         # Uses KNN
135         elif self.__use_KNN and src_id in ids and dst_id in
    ids:
136             out_port = int(self.__knn.predict([[src_id,
    dst_id, dp_id, self.__last_speeds['net1-net2'],
137                                                self.
    __last_speeds['net1-net3'], self.__last_speeds['net2-net3'
    ]]])[0])
138         # Uses direct routing
139         else:
140             out_port = self.__port_controller.direct_routing
    (dp_id, dst_id)
141
142
143         if out_port is not None:
144             actions = [parser.OFPActionOutput(out_port)]
145
146             if out_port != ofproto.OFPP_FLOOD:
147                 match = parser.OFPMatch(in_port=in_port)
148                 self.add_flow(datapath, 1, match, actions,
    self.__stats_timer)
149
150             out = parser.OFPPacketOut(datapath=datapath,
151                                       buffer_id=ofproto.
```

```
151 OFP_NO_BUFFER,
152                                         in_port=in_port,
153                                         actions=actions,
154                                         data=msg.data)
155             datapath.send_msg(out)
```

Anexo C: Código aprendizaje automático

```
 1 import numpy
 2 import csv
 3 from sklearn import metrics
 4
 5
 6 class MachineLearning:
 7     def __init__(self, dataset_path: str, model, test_data
   : bool):
 8         self.__model = model
 9
10         self.__x_train = numpy.array([])
11         self.__y_train = numpy.array([])
12
13         self.__test = test_data
14         self.__test_percentage = 0.1
15         self.__x_test = numpy.array([])
16         self.__y_test = numpy.array([])
17
18         self.__read_data(dataset_path)
19
20     def _change_model(self, model):
21         self.__model = model
22         self.__train_model()
23
24     def __read_data(self, file_path):
25         with open(file_path) as ds:
26             reader = csv.reader(ds, delimiter=',')
27             data_set = numpy.array(list(reader))
28
29         # This will help the order not to affect the
   learning
30         numpy.random.shuffle(data_set)
31
32         data_set = data_set.astype(numpy.float64)
33
34         if self.__test:
35             test_rows = int(len(data_set) * self.
   __test_percentage)
36             data_set = data_set[test_rows:]
37             test_data = data_set[:test_rows]
38
39             self.__x_test = test_data.transpose()[:-1].
   transpose()
40             self.__y_test = test_data.transpose()[-1:].
   transpose()
41
42         self.__x_train = data_set.transpose()[:-1].
```

```
42 transpose()
43         self.__y_train = data_set.transpose()[-1:].
   transpose()
44
45         self.__train_model()
46
47     def set_testing_data(self, test_data: list):
48         test_data = numpy.array(test_data).astype(numpy.
   float64)
49         self.__x_test = test_data.transpose()[:-1].
   transpose()
50         self.__y_test = test_data.transpose()[-1:].
   transpose()
51
52
53     def __train_model(self):
54         self.__model.fit(self.__x_train, self.__y_train.
   ravel())
55
56     def predict(self, predict_data: list):
57         to_predict = numpy.array(predict_data).astype(numpy
   .float64)
58         return self.__model.predict(to_predict)
59
60     def test_model(self):
61         predictions = self.__model.predict(self.__x_test)
62
63         return metrics.accuracy_score(predictions, self.
   __y_test) * 100
64
65
```

Anexo D: Código red neuronal artificial (ANN)

```
 1 import csv
 2 import threading
 3
 4 from sklearn.neural_network import MLPClassifier
 5
 6 from machine_learning import MachineLearning
 7
 8
 9 class ANN(MachineLearning):
10     def __init__(self, dataset_path: str, test_data: bool):
11         model = MLPClassifier(solver='lbfgs', alpha=1e-5,
   hidden_layer_sizes=(30, 25), random_state=1)
12         super().__init__(dataset_path, model, test_data)
13
14     def change_layers(self, layer_size, random_state: int):
15         self._change_model(MLPClassifier(solver='lbfgs',
   alpha=1e-5, hidden_layer_sizes=layer_size,
16                                          random_state=
   random_state))
17
18
19 def test_combinations(name, start, end, jumps):
20     print(name)
21     max_random_state = 5
22     max_accuracy = 0
23     max_attributes = []
24
25     ann_test = ANN('dataset.csv', False)
26     ann_test.set_testing_data(test_data)
27
28     for node_size1 in range(start, end, jumps):
29         for node_size2 in range(start, end, jumps):
30             for random_sta in range(1, max_random_state):
31                 ann_test.change_layers((node_size1,
   node_size2), random_sta)
32                 accuracy = ann_test.test_model()
33                 if accuracy > max_accuracy:
34                     max_accuracy = accuracy
35                     max_attributes = [node_size1,
   node_size2, random_sta]
36             print(name, node_size1, node_size2)
37
38     print(max_accuracy, max_attributes, '
   ______________________________')
39
40
41 def get_best_combination():
```

```
42      max_layer = 300
43      layer_per_thread = 50
44      jumps = 5
45      initial_layer = 10
46      start_layer = initial_layer
47      end_layer = layer_per_thread
48
49      coun = 0
50      while start_layer + layer_per_thread != max_layer:
51          coun += 1
52          threading.Thread(target=test_combinations, args=(
    coun, start_layer, end_layer, jumps)).start()
53
54          if start_layer == initial_layer:
55              start_layer -= initial_layer
56          start_layer += layer_per_thread
57          end_layer += layer_per_thread
58
59      test_combinations(coun + 1, start_layer, end_layer,
    jumps)
60
61
62  if __name__ == '__main__':
63      with open('test_dataset.csv') as data:
64          reader = csv.reader(data, delimiter=',')
65          test_data = list(reader)
66
67      ann = ANN('dataset.csv', False)
68      ann.set_testing_data(test_data)
69      print(ann.test_model())
```

Anexo E: Código k vecino mas cercano (KNN)

```
 1 from sklearn.neighbors import KNeighborsClassifier
 2 import csv
 3 from machine_learning import MachineLearning
 4
 5
 6 class KNN(MachineLearning):
 7     def __init__(self, dataset_path: str, test_data: bool):
 8         # Metrics: 'euclidean' 'manhattan'
 9         model = KNeighborsClassifier(n_neighbors=3, metric=
   'euclidean')
10         super().__init__(dataset_path, model, test_data)
11
12     def change_neighbors(self, n_neighbors: int):
13         self._change_model(KNeighborsClassifier(n_neighbors
   =n_neighbors, metric='euclidean'))
14
15
16 if __name__ == '__main__':
17
18     with open('test_dataset.csv') as data:
19         reader = csv.reader(data, delimiter=',')
20         test_data = list(reader)
21
22     knn = KNN('dataset.csv', False)
23     knn.set_testing_data(test_data)
24     print(3, knn.test_model())
25
26     for i in range(2, 21):
27         knn.change_neighbors(i)
28         print(i, knn.test_model())
```

Anexo F: Código de tipos de enrutamiento

```
1
2
3  class PortController:
4      def __init__(self):
5          # OVS IDs
6          self.__ovs1 = 1
7          self.__ovs2 = 2
8          self.__ovs3 = 3
9
10         # Hosts IPs
11         self.h1 = 1
12         self.h2 = 2
13         self.h3 = 3
14         self.__hosts_ids = {
15             '02:fd:00:00:00:01': 1,
16             '02:fd:00:00:01:01': 2,
17             '02:fd:00:00:02:01': 3
18         }
19
20     def get_hosts_list(self) -> list:
21         hosts = []
22         for key in self.__hosts_ids:
23             hosts.append(key)
24         return hosts
25
26     def get_host_ids(self):
27         return [self.h1, self.h2, self.h3]
28
29     def get_host_id(self, host_ip: str):
30         if host_ip not in self.__hosts_ids:
31             return host_ip
32         return self.__hosts_ids[host_ip]
33
34     def get_ovs_list(self) -> list:
35         return [self.__ovs1, self.__ovs2, self.__ovs3]
36
37     def direct_routing(self, dp_id: int, dst_id: int):
38         out_port = None
39         if dp_id == self.__ovs1:
40             if dst_id == self.h1:
41                 out_port = 3
42             elif dst_id == self.h2:
43                 out_port = 1
44             elif dst_id == self.h3:
45                 out_port = 2
46         elif dp_id == self.__ovs2:
47             if dst_id == self.h1:
```

```
48                 out_port = 1
49             elif dst_id == self.h2:
50                 out_port = 3
51             elif dst_id == self.h3:
52                 out_port = 2
53         elif dp_id == self.__ovs3:
54             if dst_id == self.h1:
55                 out_port = 2
56             elif dst_id == self.h2:
57                 out_port = 1
58             elif dst_id == self.h3:
59                 out_port = 3
60         return out_port
61
62     def indirect_routing(self, dp_id: int, src_id: int,
   dst_id: int):
63         out_port = None
64         if dp_id == self.__ovs1:
65             if dst_id == self.h1:
66                 out_port = 3
67             elif src_id == self.h2 and dst_id == self.h3:
68                 out_port = 2
69             elif src_id == self.h3 and dst_id == self.h2:
70                 out_port = 1
71             elif src_id == self.h1 and dst_id == self.h2:
72                 out_port = 2
73             elif src_id == self.h1 and dst_id == self.h3:
74                 out_port = 1
75
76         elif dp_id == self.__ovs2:
77             if dst_id == self.h2:
78                 out_port = 3
79             elif src_id == self.h1 and dst_id == self.h3:
80                 out_port = 2
81             elif src_id == self.h3 and dst_id == self.h1:
82                 out_port = 1
83             elif src_id == self.h2 and dst_id == self.h1:
84                 out_port = 2
85             elif src_id == self.h2 and dst_id == self.h3:
86                 out_port = 1
87
88         elif dp_id == self.__ovs3:
89             if dst_id == self.h3:
90                 out_port = 3
91             elif src_id == self.h1 and dst_id == self.h2:
92                 out_port = 1
93             elif src_id == self.h2 and dst_id == self.h1:
```

```
 94                 out_port = 2
 95             elif src_id == self.h3 and dst_id == self.h1:
 96                 out_port = 1
 97             elif src_id == self.h3 and dst_id == self.h2:
 98                 out_port = 2
 99
100         return out_port
```

Anexo G: Código analizador de estadísticas de puertos

```
 1 from ryu.ofproto import ofproto_v1_3
 2
 3
 4 class StatsAnalyzer:
 5     def __init__(self):
 6
 7         # OVS IDs
 8         self.__ovs1 = 1
 9         self.__ovs2 = 2
10         self.__ovs3 = 3
11
12         # Stats been updated
13         self.__link_stats = {
14             'ovs1': False,
15             'ovs2': False,
16             'ovs3': False,
17             'net1-net2': 0,
18             'net1-net3': 0,
19             'net2-net3': 0
20         }
21
22         # Saves last bytes count
23         self.__last_bytes_count = {
24             'net1-net2': None,
25             'net1-net3': None,
26             'net2-net3': None
27         }
28
29     def manage_port_stats(self, dp_id: int, body: list,
   timer: int, speeds: dict):
30
31         for stat in body:
32             port = stat.port_no
33
34             if port == ofproto_v1_3.OFPP_LOCAL:
35                 continue
36
37             receive_bytes = stat.rx_bytes
38             send_bytes = stat.tx_bytes
39
40             if dp_id == self.__ovs1:
41                 self.__link_stats['ovs1'] = True
42                 if port == 1:
43                     self.__link_stats['net1-net2'] +=
   receive_bytes + send_bytes
44                 elif port == 2:
45                     self.__link_stats['net1-net3'] +=
```

```
45 receive_bytes + send_bytes
46             elif dp_id == self.__ovs2:
47                 self.__link_stats['ovs2'] = True
48                 if port == 1:
49                     self.__link_stats['net1-net2'] +=
   receive_bytes + send_bytes
50                 elif port == 2:
51                     self.__link_stats['net2-net3'] +=
   receive_bytes + send_bytes
52             elif dp_id == self.__ovs3:
53                 self.__link_stats['ovs3'] = True
54                 if port == 1:
55                     self.__link_stats['net2-net3'] +=
   receive_bytes + send_bytes
56                 elif port == 2:
57                     self.__link_stats['net1-net3'] +=
   receive_bytes + send_bytes
58
59         if self.__link_stats['ovs1'] and self.__link_stats[
   'ovs2'] and self.__link_stats['ovs3']:
60             if self.__last_bytes_count['net1-net2'] is not
   None:
61                 speeds['net1-net2'] = (self.__link_stats['
   net1-net2'] - self.__last_bytes_count['net1-net2'])/timer
62                 speeds['net1-net2'] = round(speeds['net1-
   net2'] * 8/2000000.0, 3)
63
64                 speeds['net1-net3'] = (self.__link_stats['
   net1-net3'] - self.__last_bytes_count['net1-net3'])/timer
65                 speeds['net1-net3'] = round(speeds['net1-
   net3'] * 8/2000000.0, 3)
66
67                 speeds['net2-net3'] = (self.__link_stats['
   net2-net3'] - self.__last_bytes_count['net2-net3'])/timer
68                 speeds['net2-net3'] = round(speeds['net2-
   net3'] * 8/2000000.0, 3)
69
70             self.__last_bytes_count['net1-net2'] = self.
   __link_stats['net1-net2']
71             self.__last_bytes_count['net1-net3'] = self.
   __link_stats['net1-net3']
72             self.__last_bytes_count['net2-net3'] = self.
   __link_stats['net2-net3']
73
74             self.__link_stats = {
75                 'ovs1': False,
76                 'ovs2': False,
```

```
77                  'ovs3': False,
78                  'net1-net2': 0,
79                  'net1-net3': 0,
80                  'net2-net3': 0
81              }
82
83          return speeds
84
85
```

Anexo H: Código para grabar data set

```
1  from csv import writer
2
3  from port_controller import PortController
4
5
6  class DatasetCreator:
7
8      @staticmethod
9      def __get_out_port(direct_speed, indirect_speed, ovs_id
   : int, src_id: int, dst_id: int,
10                        port_controller: PortController) ->
   int:
11         # Direct routing has less traffic
12         if direct_speed <= indirect_speed:
13             out_port = port_controller.direct_routing(
   ovs_id, dst_id)
14         # Indirect routing has less traffic
15         else:
16             out_port = port_controller.indirect_routing(
   ovs_id, src_id, dst_id)
17         return out_port
18
19     def __add_dataset(self, speeds: dict):
20         port_controller = PortController()
21
22         with open('dataset.csv', mode='a+', newline='') as
   ds:
23             csv_writer = writer(ds)
24
25             for ovs_id in port_controller.get_ovs_list():
26                 # Decision for H1
27                 # h1 -> h2
28                 direct_speed = speeds['net1-net2']
29                 indirect_speed = speeds['net1-net3'] +
   speeds['net2-net3']
30                 src_id = port_controller.h1
31                 dst_id = port_controller.h2
32
33                 out_port = self.__get_out_port(direct_speed
   , indirect_speed, ovs_id, src_id,
34                                                dst_id,
   port_controller)
35
36                 csv_writer.writerow([src_id, dst_id, ovs_id
   , speeds['net1-net2'], speeds['net1-net3'],
37                                      speeds['net2-net3'],
   out_port])
```

```
38
39                  # h1 -> h3
40                  direct_speed = speeds['net1-net3']
41                  indirect_speed = speeds['net1-net2'] +
   speeds['net2-net3']
42                  src_id = port_controller.h1
43                  dst_id = port_controller.h3
44
45                  out_port = self.__get_out_port(direct_speed
   , indirect_speed, ovs_id, src_id,
46                                                 dst_id,
   port_controller)
47
48                  csv_writer.writerow([src_id, dst_id, ovs_id
   , speeds['net1-net2'], speeds['net1-net3'],
49                                       speeds['net2-net3'],
   out_port])
50
51                  # Decision for H2
52                  # h2 -> h1
53
54                  direct_speed = speeds['net1-net2']
55                  indirect_speed = speeds['net1-net3'] +
   speeds['net2-net3']
56                  src_id = port_controller.h2
57                  dst_id = port_controller.h1
58
59                  out_port = self.__get_out_port(direct_speed
   , indirect_speed, ovs_id, src_id,
60                                                 dst_id,
   port_controller)
61
62                  csv_writer.writerow([src_id, dst_id, ovs_id
   , speeds['net1-net2'], speeds['net1-net3'],
63                                       speeds['net2-net3'],
   out_port])
64
65                  # h2 -> h3
66
67                  direct_speed = speeds['net2-net3']
68                  indirect_speed = speeds['net1-net2'] +
   speeds['net1-net3']
69                  src_id = port_controller.h2
70                  dst_id = port_controller.h3
71
72                  out_port = self.__get_out_port(direct_speed
   , indirect_speed, ovs_id, src_id,
```

```
 73                                                  dst_id,
    port_controller)
 74
 75                 csv_writer.writerow([src_id, dst_id,
    ovs_id, speeds['net1-net2'], speeds['net1-net3'],
 76                                      speeds['net2-net3'],
    out_port])
 77
 78                 # Decision for H3
 79                 # h3 -> h1
 80
 81                 direct_speed = speeds['net1-net3']
 82                 indirect_speed = speeds['net1-net2'] +
    speeds['net2-net3']
 83                 src_id = port_controller.h3
 84                 dst_id = port_controller.h1
 85
 86                 out_port = self.__get_out_port(
    direct_speed, indirect_speed, ovs_id, src_id,
 87                                                  dst_id,
    port_controller)
 88
 89                 csv_writer.writerow([src_id, dst_id,
    ovs_id, speeds['net1-net2'], speeds['net1-net3'],
 90                                      speeds['net2-net3'],
    out_port])
 91
 92                 # h3 -> h2
 93
 94                 direct_speed = speeds['net2-net3']
 95                 indirect_speed = speeds['net1-net2'] +
    speeds['net1-net3']
 96                 src_id = port_controller.h3
 97                 dst_id = port_controller.h2
 98
 99                 out_port = self.__get_out_port(
    direct_speed, indirect_speed, ovs_id, src_id,
100                                                  dst_id,
    port_controller)
101
102                 csv_writer.writerow([src_id, dst_id,
    ovs_id, speeds['net1-net2'], speeds['net1-net3'],
103                                      speeds['net2-net3'],
    out_port])
104
105
```

Anexo I: Bash para limitar ancho de banda

```
#!/bin/bash
#-------------------------------------------------------------------
# THIS SCRIPT CONFIGURES A GIVEN BANDWIDTH ON INTERFACES,
#-------------------------------------------------------------------

#-------------------------CONFIGURATION PARAMETERS
-----------------------------
MAX_RATE=10000000            # Rate q0
PORT_1=net3-net1-1           # Interface 1 Net0
PORT_2=net1-net2-0           # Interface 2 Net0
PORT_3=net1-net2-1           # Interface 1 Net1
PORT_4=net2-net3-0           # Interface 2 Net1
PORT_5=net2-net3-1           # Interface 1 Net2
PORT_6=net3-net1-0           # Interface 2 Net2
#-------------------------------------------------------------------

#---------------------------------ACTIONS TO DO
--------------------------------
start() {

    ovs-vsctl -- set Port $PORT_1 qos=@newqos -- \
    --id=@newqos create qoS type=linux-htb other-config:max-
rate=$MAX_RATE \
        queues:0=@q0 -- \
    --id=@q0 create queue other-config:max-rate=$MAX_RATE

    ovs-vsctl -- set Port $PORT_2 qos=@newqos -- \
    --id=@newqos create qoS type=linux-htb other-config:max-
rate=$MAX_RATE \
        queues:0=@q0 -- \
    --id=@q0 create queue other-config:max-rate=$MAX_RATE

    ovs-vsctl -- set Port $PORT_3 qos=@newqos -- \
    --id=@newqos create qoS type=linux-htb other-config:max-
rate=$MAX_RATE \
        queues:0=@q0 -- \
    --id=@q0 create queue other-config:max-rate=$MAX_RATE

    ovs-vsctl -- set Port $PORT_4 qos=@newqos -- \
    --id=@newqos create qoS type=linux-htb other-config:max-
rate=$MAX_RATE \
```

```
36          queues:0=@q0 -- \
37      --id=@q0 create queue other-config:max-rate=$MAX_RATE
38
39      ovs-vsctl -- set Port $PORT_5 qos=@newqos -- \
40      --id=@newqos create qoS type=linux-htb other-config:max-
    rate=$MAX_RATE \
41          queues:0=@q0 -- \
42      --id=@q0 create queue other-config:max-rate=$MAX_RATE
43
44      ovs-vsctl -- set Port $PORT_6 qos=@newqos -- \
45      --id=@newqos create qoS type=linux-htb other-config:max-
    rate=$MAX_RATE \
46          queues:0=@q0 -- \
47      --id=@q0 create queue other-config:max-rate=$MAX_RATE
48 }
49
50 stop() {
51
52      sudo ovs-vsctl clear Port $PORT_1 qos
53      sudo ovs-vsctl clear Port $PORT_2 qos
54      sudo ovs-vsctl clear Port $PORT_3 qos
55      sudo ovs-vsctl clear Port $PORT_4 qos
56      sudo ovs-vsctl clear Port $PORT_5 qos
57      sudo ovs-vsctl clear Port $PORT_6 qos
58      ovs-vsctl -- --all destroy QoS
59
60 }
61
62 restart() {
63
64      stop
65      sleep 1
66      start
67
68 }
69
70 show() {
71
72      ovs-vsctl list qos
73 }
74
75 case "$1" in
76
77   start)
```

```
 78
 79      echo -e "\n---Starting BW shaping--- "
 80      start
 81      echo -e "---done!!!\n"
 82      ;;
 83
 84    stop)
 85
 86      echo -e "\n---Stopping BW shaping--- "
 87      stop
 88      echo -e "---done!!!\n"
 89      ;;
 90
 91    restart)
 92
 93      echo -e "\n---Restarting BW shaping--- "
 94      restart
 95      echo -e "---done!!!\n"
 96      ;;
 97
 98    show)
 99
100      echo -e "\n---QoS status for $SHAPER_PORT:\n"
101      show
102      echo ""
103      ;;
104
105    *)
106
107      pwd=$(pwd)
108      echo "Usage: $(/usr/bin/dirname $pwd)/tc.bash {start|
    stop|restart|show}"
109      ;;
110
111 esac
112
113 exit 0
114
115
116
```