Application for Load Balancing in SDN

Marlon Esteban Olaya
Escuela Politécnica Nacional
Ladrón de Guevara E11-253
Quito-Ecuador
(593) 2 2976-300
marlon.olayay@epn.edu.ec

Iván Bernal
Escuela Politécnica Nacional
Ladrón de Guevara E11-253
Quito-Ecuador
(593) 2 2976-300
ivan.bernal@epn.edu.ec

David Mejía
Escuela Politécnica Nacional
Ladrón de Guevara E11-253
Quito-Ecuador
(593) 2 2976-300
david.mejia@epn.edu.ec

ABSTRACT
This paper presents an application that implements three load balancers using a Software Defined Network (SDN). The application was developed for the Ryu controller. First, the principles of three queueing algorithms are presented; the selected algorithms are: FIFO, Round Robin and Deficit Round Robin. Then, the main aspects of the design and implementation of load balancing modules based on each of the aforementioned algorithms are outlined. The application is able to assign a client request to a server based on the criteria associated to a given queuing algorithm. Additionally, the tests that were carried out with the application using both a virtual SDN implemented with Mininet and a physical SDN using an OpenFlow switch are discussed. Finally, the results of the tests on the physical SDN that focused on proofing the functionality and estimating the response time of the application are included.

Categories and Subject Descriptors
C.2.1 [**Network Architecture and Design**]: Network Communications

General Terms
Algorithms, Management, Design.

Keywords
SDN, Ryu, Load Balancing, OpenFlow, Mininet.

1. INTRODUCCIÓN
La necesidad de transmitir gran cantidad de información de forma eficiente y segura ha causado el continuo avance y desarrollo de las tecnologías de la información. Uno de estos avances es la arquitectura conocida como SDN (*Software Defined Network*) [13] que permite un mayor control sobre todos los dispositivos de interconexión de la red desde un solo punto lógico de administración. Las SDN son redes que en este momento se encuentran en permanente desarrollo por todas las potencialidades que ofrecen en aspectos de seguridad y de servicios que pueden ofrecer.

EATIS'2016, April 27–29, 2016, Cartagena, Colombia.
978-1-5090-2435-3/16/$31.00 ©2016 IEEE.

En la actualidad, la mayor parte de la población mundial utiliza Internet para diferentes propósitos como realizar compras, leer, escuchar música, mirar vídeos, entre otros. Cada página web está alojada en un servidor web, y en los sitios populares se emplea más de un servidor web, ya que la cantidad de tráfico en un solo servidor podría causar una saturación y posterior caída del mismo, provocando pérdidas de tiempo y dinero. Para evitar este tipo de pérdidas se realiza una distribución del tráfico entre los diferentes servidores que alojan el mismo contenido, siendo este proceso transparente para el usuario. Esta distribución se realiza mediante software o hardware especializado conocido como "balanceador de carga"; estos balanceadores de carga tienen un alto costo en el mercado y probablemente no estén al alcance de las compañías que están comenzando a desarrollarse.

Lo mencionado ha motivado el desarrollo de la aplicación presentada en este artículo, para lo cual se implementó una SDN virtual y una SDN física, y se realizó el diseño, implementación y pruebas de la aplicación. El artículo está organizado de la siguiente manera: primero se presenta una breve descripción sobre las SDN, el protocolo OpenFlow, el controlador Ryu, y los algoritmos de encolamiento; luego se mencionan detalles de la aplicación desarrollada para balancear la carga de servidores web empleando el controlador de una SDN; a continuación se presentan las pruebas realizadas y los resultados obtenidos y, finalmente, se emiten conclusiones y se indica el trabajo futuro.

2. DESCRIPCIÓN DE CONCEPTOS Y HERRAMIENTAS

2.1 Redes Definidas por Software
La ONF (*Open Networking Foundation*) menciona que las SDN son redes en las que se ha separado el plano de datos del plano de control [12]. El plano de datos se ubica en cada dispositivo de la red (ej. router o switch). El plano de control, se centraliza en un solo punto lógico llamado controlador. Las SDN buscan optimizar y simplificar las operaciones en la red acercando más las interacciones entre las aplicaciones y servicios de la red con los elementos de la misma; se optimizan las interacciones ya que estos elementos trabajan en el plano de datos y las decisiones las realiza el controlador [1], de la misma manera se simplifican las operaciones, ya que las acciones son programadas de forma centralizada en el controlador en lugar de configurar cada dispositivo de forma manual. La interfaz de comunicación entre el controlador y los dispositivos de red la constituye el protocolo OpenFlow [11]. Distintos controladores han sido desarrollados como POX [15], Beacon [2], FloodLight [14], OpenDayLight [10] y Ryu [18]. La Fig. 1 presenta el esquema de una SDN.

2.2 OpenFlow

Es un protocolo que fue ideado e implementado con fines de investigación en la Universidad de Stanford. OpenFlow busca reemplazar todas las funcionalidades de los protocolos de las capas 2 y 3 en switches y routers comerciales [20].

La Fig. 2 muestra la cabecera del protocolo OpenFlow versión 1, conformada por doce campos; el primero corresponde al puerto de ingreso y los otros once corresponden a campos de Ethernet, VLAN, IP y TCP/UDP. En OpenFlow, los dispositivos de red se denominan switches OpenFlow. Un switch OpenFlow está constituido por una tabla de flujos y un canal seguro. La tabla de flujos establece la ruta de los paquetes para su reenvío (*forwarding*) y el canal se levanta entre el switch y un controlador que administrará el switch utilizando OpenFlow. La tabla de flujos contiene un conjunto de reglas para los flujos. Estas reglas están conformadas por campos de las cabeceras de los paquetes, valores de contadores y acciones, como se muestra en la Fig. 3. Se dice que un paquete hace *match* con una regla de flujo, cuando los valores de las cabeceras del paquete corresponden a los valores de la regla de flujo instalada. Los contadores tienen valores de parámetros como flujos activos, paquetes procesados, bytes procesados, tiempos de vida, entre otros. Finalmente, las acciones son las órdenes que tiene el switch para tratar al paquete que haya hecho *match* con una determinada regla de flujo [20].

2.3 Ryu

Ryu es un *framework* basado en componentes, está formado por una gran cantidad de librerías y recursos que facilitan el desarrollo de aplicaciones. Ryu está escrito en Python y trabaja con la versión 2.6 o superior, aunque no es completamente funcional con la versión 3 [17]. Las aplicaciones de Ryu utilizan un solo hilo de ejecución e implementan alguna funcionalidad. El controlador utiliza una programación dirigida por eventos, es decir, las acciones que se realizan dependen de los eventos que ocurran en tiempo de ejecución. Cabe mencionar que los eventos son manejados como mensajes entre las aplicaciones. La API de Ryu permite crear eventos propios y manipuladores para eventos específicos o generar eventos preestablecidos. El decorador `ryu.controller.handler.set_ev_cls` permite crear los manipuladores de eventos. Se puede clasificar los componentes del *framework* Ryu [16] de la siguiente manera: a) ejecutables, permiten correr las aplicaciones de Ryu; b) componentes base, se encargan de la administración de las aplicaciones de Ryu; c) controlador OpenFlow, gestiona a los switches generando y direccionando los eventos hacia la aplicación; d) codificador/decodificador, provee las librerías necesarias para manipular los parámetros de las diferentes versiones de OpenFlow desde la versión 1.0 de OpenFlow hasta la versión 1.5; e) aplicaciones, se pueden utilizar de manera libre ya sea como esqueleto de una aplicación más compleja o por partes, utilizando solo ciertas funciones; f) librerías, para utilizar protocolos conocidos como TCP, UDP, ARP, entre otros, estas librerías permiten crear, modificar o hacer algún tipo de manipulación en los paquetes de los protocolos disponibles.

2.4 Sistemas de encolamiento

Un sistema de encolamiento es un modelo abstracto en el que se tienen dos conjuntos de participantes, los clientes y los servidores. En este modelo, los clientes solicitan servicios a los servidores, si los servidores no tienen capacidad de atender a todos los clientes en ese momento, se producirá un encolamiento, es decir se pondrá en cola a los clientes que no puedan ser atendidos en ese instante de tiempo hasta que un servidor termine de atender la solicitud en curso y pueda atender a otro cliente de la cola [9]. Los sistemas de encolamiento se emplean en servidores web. Un cliente en este sistema de encolamiento es cualquier navegador web, consola de comandos o aplicación capaz de realizar peticiones web, es decir peticiones HTTP o HTTPS. Cuando un servidor web tiene un gran número de clientes generando una gran cantidad de tráfico es natural que un solo equipo físico o virtual se sature y no sea suficiente para satisfacer las necesidades de todos los clientes; por lo tanto, es común tener un clúster de servidores o varios servidores con la misma información para poder responder a todas las peticiones. Suponiendo que no se cuenta con un clúster de servidores, sino solo con servidores réplicas, para los usuarios de una página web (un servidor web) sería molesto tener que cambiar la dirección en el navegador si el servidor principal los atiende con un retardo considerable o no los atiende, los usuarios buscarían la información o servicio que necesitan en otro lado. Esto implicaría pérdidas a corto y largo plazo para los propietarios del sitio web, para evitar estas pérdidas se puede utilizar un enmascaramiento de todos los servidores detrás de una única dirección o nombre, este trabajo lo puede realizar un Balanceador de Carga.

2.5 Balanceador de Carga

Un balanceador de carga puede estar implementado en software o en un equipo físico, su trabajo es distribuir la carga (tráfico generado por los clientes) entre varios servidores del mismo tipo. Los balanceadores de carga tienen diferentes métodos para realizar su trabajo, estos métodos pueden ser manuales, automáticos o una combinación de ambos [19]. En el método manual se programa directamente la regla en el balanceador, este método de balanceo puede llegar a ser poco eficiente en una red, ya que no se puede predecir el comportamiento del usuario y se podría llegar a saturar a uno de los servidores sin realizar un balanceo de carga propiamente dicho. Los métodos automáticos son más eficiente que los manuales ya que la carga se distribuye dinámicamente según algún patrón programado en el balanceador, estos patrones que realizan el balanceo se implementan de acuerdo a algún algoritmo de encolamiento [3], como: FIFO (*First In First Out*), Round Robin [5] y Deficit Round Robin [4].

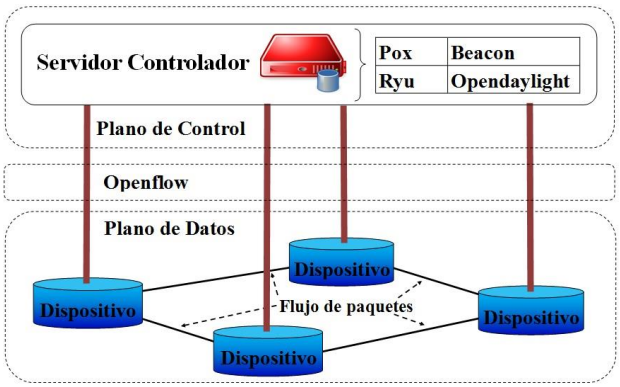

Figura 1. Planos de control y de datos en una SDN

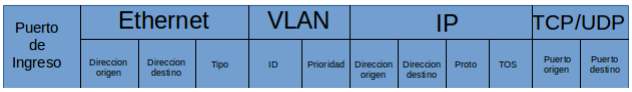

Figura 2. Cabecera OpenFlow v 1.0

Figura 3. Regla de Flujo en un switch OpenFlow

2.5.1 FIFO

El primero que entra es el primero que sale; en el contexto cliente-servidor quiere decir que el primer cliente que realice la petición será atendido primero. Cada cliente se pone en cola y una vez que esté disponible algún servidor el cliente que está primero en la cola sale de la misma, es atendido y el proceso continúa, no existe ningún tipo de priorización, todo depende de los tiempos de llegada.

2.5.2 Round Robin (RR)

En este algoritmo se establece un listado integrado con todos los servidores y se les da un orden. La primera solicitud se la dirige al primer servidor del listado y se continúa asignando las solicitudes de forma ordenada a todos los servidores del listado. Cuando todos los servidores han recibido una solicitud, una nueva solicitud será enviada al primer servidor del listado y el proceso empieza de nuevo. Este proceso se ejemplifica con tres servidores en la Fig. 4, en la cual se observa que las solicitudes de los clientes han sido encoladas, la solicitud del cliente 1 (amarillo) se envía al servidor 1, la solicitud del cliente 2 (azul) se envía al servidor 2, la solicitud del cliente 3 (verde) al servidor 3, luego de lo cual se repetirá el ciclo.

2.5.3 Deficit Round Robin (DRR)

Este algoritmo está basado en Round Robin, pero tiene varios parámetros adicionales que proporcionan una forma de encolamiento que considera el número de bytes que puede aceptar cada servidor en una ronda. Los parámetros adicionales que se consideran en este algoritmo son: a) quantum, es el tamaño máximo de un paquete que puede ser recibido en la ronda actual por un servidor; b) *deficit-counter*, es un contador de déficit, que aumenta si el tamaño del paquete es mayor al quantum, es decir cuando el paquete no puede ser procesado en la ronda actual.

Figura 4. Round Robin con tres servidores

Para explicar el proceso de DRR se considera que cada servidor tiene su propia cola y solo procesará los paquetes que estén en ella; además se asume que el quantum es fijo para cada servidor. Para empezar a distribuir los paquetes se debe fijar los valores del quantum en un valor estático y el *deficit-counter* en cero ($q = valor \wedge dc = 0$). El proceso DRR se realiza de la siguiente manera: Se actualiza el *deficit-counter* a su valor más el valor del quantum ($dc = dc + q$); se compara si el tamaño del paquete es menor que el *deficit-counter* ($paquete \leq dc$), si es menor, el paquete se envía al servidor y el *deficit-counter* se resetea ($dc = 0$) y el servidor termina su turno; si es mayor, el paquete se queda en la cola del servidor y espera para ser enviado en la siguiente ronda, el *deficit-counter* se iguala al valor del quantum ($dc = q$) y el servidor termina su turno.

3. APLICACIÓN PARA BALANCEO DE CARGA

3.1 Introducción

Un sitio web con un contenido novedoso y actualizado, será visitado por un número de usuarios cada vez mayor, y llegará el momento donde un solo servidor no será suficiente para atender de manera apropiada a todos los clientes, por lo que se podría replicar al servidor para ofrecer una mejor atención al usuario. Para asignar las peticiones entre los servidores, se utilizaría un balanceador de carga. Por lo mencionado, se desarrolló una aplicación que permita balancear la carga entre los distintos servidores web.

3.2 Requerimientos

Los requerimientos para esta aplicación se determinaron en base a: ambiente físico (infraestructura física donde funcionará la aplicación); usuarios; y, funcionalidad (lo que se necesita que haga la aplicación y como lo debe hacer [7]).

Los requerimientos de la aplicación son: debe trabajar con switches que soporten el protocolo OpenFlow; debe permitir que el administrador pueda configurar la aplicación desde un archivo externo a la aplicación; debe trabajar como balanceador de carga para los servidores web; debe enmascarar las direcciones de los servidores, es decir, mostrar solo una dirección a los clientes; debe tener tres modos de operación, basados en FIFO, RR y DRR; y, debe actuar cada vez que llegue un nuevo cliente.

Antes de elegir un criterio para el balanceo de carga, se consideró que la aplicación trabajaría en una SDN, por lo tanto, debe generar reglas de flujos que el controlador enviará a los switches para que los introduzcan en sus tablas de flujos. Por otro lado, los algoritmos de encolamiento mencionados en la Sección anterior generalmente se implementan a nivel de paquetes, pero en el escenario que se maneja para esta aplicación el procesamiento de cada paquete no sería algo eficiente, ya que cada paquete debería ir hasta el controlador, que es la entidad que realiza el procesamiento, y aumentaría el retardo en cada uno de estos. Por lo mencionado, los mecanismos de encolamiento están orientados a distribuir clientes a cada servidor, no paquetes. Un cliente se diferencia de otro por su dirección IP y puerto (origen) empleados para comunicarse con el servidor web.

3.3 Balanceador de carga basado en FIFO

Este mecanismo se presenta en la Fig. 5. En este caso solo importa el orden de llegada, pero al tratarse de una aplicación

que distribuye los pedidos de los clientes (Fig. 5a), no solo basta con tener en cuenta el orden de llegada, por este motivo se decidió implementar un mecanismo de distribución a la salida del switch; este mecanismo distribuye los clientes de manera aleatoria entre los servidores disponibles (Fig. 5b), es decir, se conserva la base de FIFO en el switch, que es direccionar primero al cliente que llega primero, pero el servidor que lo atiende puede ser cualquiera de los disponibles, sin un orden específico. La generación del valor aleatorio se realizó mediante un método de la clase Random, que escoge un elemento al azar de una lista de elementos, utilizando el generador Mersenne Twister, que es un generador de números pseudo-aleatorios.

3.4 Balanceador de carga basado en RR

La Fig. 6 presenta un esquema de este mecanismo, el cual se diferencia con el basado en FIFO en la distribución a la salida del switch. Los servidores (Fig. 6b) han sido ordenados y los clientes (Fig. 6a) se reparten entre los servidores de forma cíclica, de uno en uno empezando por el primer servidor.

3.5 Balanceador de carga basado en DRR

Después de analizar el funcionamiento del algoritmo DRR, se concluyó que este solo podía ser implementado completamente a nivel de paquetes; sin embargo, como se mencionó anteriormente, no es conveniente una implementación a nivel de paquetes, por lo tanto, solo se utilizaron ciertos aspectos de DRR, como el control del número de bytes y el orden de tratamiento de clientes para la selección del criterio.

En el mecanismo basado en DRR (ver Fig. 7) se pueden diferenciar tres partes: el orden de llegada de los clientes (Fig. 7a); la interacción entre el controlador Ryu y el switch OpenFlow (Fig. 7b), ya que se realiza una constante retroalimentación entre estos; y, el direccionamiento de cada cliente al servidor con menos carga en el momento de la petición (Fig. 7c).

En este caso, si llega una gran cantidad de solicitudes, de distintos clientes, se producirá un retardo, ya que se realiza un procesamiento adicional con cada cliente para verificar el número de bytes procesados por cada servidor. Pero este retardo solo será evidente en la respuesta de la primera solicitud de cada cliente, ya que una vez generada e instalada la regla, no habrá más retrasos en las solicitudes posteriores provenientes de dicho cliente.

3.6 Diseño e Implementación de la Aplicación

La aplicación se escribió usando el lenguaje de programación Python y la API de Ryu. La aplicación está diseñada con tres módulos, un módulo de recepción de paquetes, uno de procesamiento y, finalmente, uno de creación de reglas.

3.6.1 Módulo de Recepción de Paquetes

Para este módulo se considera si el paquete es ARP o no. Las direcciones de los servidores físicos están ocultas tras una dirección IP conocida así como una dirección MAC conocida. La respuesta ARP se necesita para que los clientes puedan llenar sus tablas ARP, por lo tanto, no es necesario que estas peticiones lleguen a los servidores físicos. Las únicas peticiones que llegarán a los servidores serán de tipo HTTP o HTTPS. Para el tratamiento de las peticiones ARP (ver Fig. 8a), la aplicación trabaja de la siguiente manera: recibe la petición ARP; comprueba si el paquete tiene como dirección destino la dirección IP conocida de los servidores; construye un paquete ARP con la dirección IP conocida; y, se lo envía al cliente que hizo la petición. Para los paquetes que no son ARP (ver Fig. 8b), la aplicación realiza lo siguiente: revisa la dirección IP destino del paquete, si no es la dirección conocida, se descarta el paquete; si es la dirección conocida, revisa el puerto destino, que debe ser o bien HTTP o HTTPS, en caso afirmativo, el paquete pasa al módulo de procesamiento, caso contrario se descarta.

Para gestionar los paquetes en el controlador se implementó la función `_packet_in_handler`, y para gestionar los paquetes ARP la función `_arp_handler`. El decorador `@set_ev_cls` se emplea en la función `_packet_in_handler` para que sea invocada siempre que se produzca un evento `OFPPacketIn`, que se produce cuando ingresa un paquete OpenFlow al controlador.

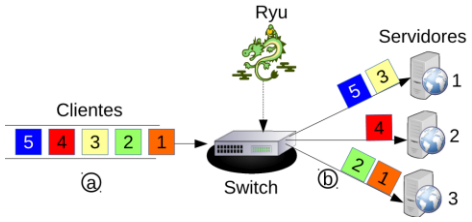

Figura 5. Mecanismo basado en FIFO

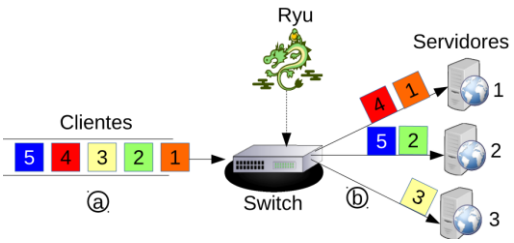

Figura 6. Mecanismo basado en Round Robin

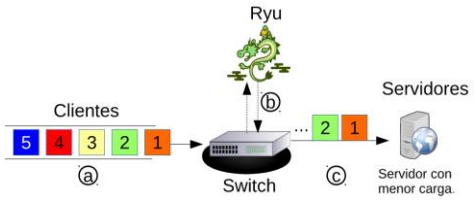

Figura 7. Mecanismo basado en Deficit Round Robin

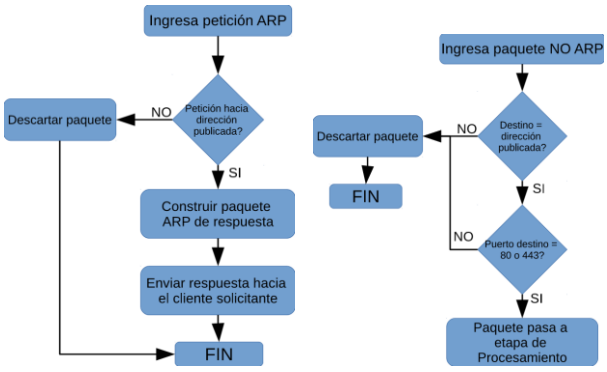

Figura 8. Diagrama de flujo: recepción de paquetes

Además, `_packet_in_handler` comprueba si el paquete recibido es del tipo ARP, en cuyo caso invoca a `_arp_handler`; caso contrario revisa si el paquete es IP y si tiene como destino la dirección IP de los servidores, en cuyo caso pasa al módulo de procesamiento; si nada de esto se cumple, el paquete no se procesa. Si la petición ARP está dirigida a la dirección IP conocida de los servidores, se crea una respuesta que tendrá como destino la dirección IP del cliente que envió la petición y como origen las direcciones IP y MAC conocidas.

3.6.2 Módulo de Procesamiento

Este módulo analiza el paquete y decide a que servidor enviarlo de acuerdo al criterio de balanceo seleccionado. Se tienen tres diferentes tipos de procesamiento, uno para FIFO, otro para RR y uno para DRR. Este módulo realiza un análisis de las direcciones origen y destino; almacena las direcciones IP y MAC del cliente; selecciona al servidor mediante uno de los mecanismos de balanceo de carga, reenvía el paquete al servidor escogido; y, pasa el resultado al módulo de creación de reglas (ver Fig. 9). Si la aplicación emplea el mecanismo FIFO, se procesa a los paquetes en el orden de llegada y para reenviarlos se selecciona un servidor de forma aleatoria (ver Fig. 10). En caso de trabajar con RR el proceso es el siguiente: se ordena la lista de servidores, cada vez que llega un nuevo paquete se selecciona al primer servidor de la lista ordenada, una vez hecho esto, el servidor pasa al final de la lista (ver Fig. 11).

Al usar DRR, el módulo realiza el siguiente proceso: cuando el controlador recibe un paquete enviado por el switch, el módulo envía al switch una petición de estadísticas de los flujos. De la respuesta del switch se extrae la cantidad de bytes que ha procesado el switch por cada flujo, se calcula la cantidad de bytes que se ha enviado a cada servidor y se selecciona el servidor con el menor número de bytes procesados hasta el momento (ver Fig. 12). Cabe mencionar que cada regla que se instale en el switch tendrá un tiempo de vida corto para que las estadísticas del flujo de un cliente inactivo no afecten a la selección del servidor.

El módulo de procesamiento dispone de la función `_ip_handler`. Cuando la opción elegida en el archivo de configuración es FIFO, dentro de la función `_ip_handler` se llama a la función auxiliar `fifo`, que seleccionará de forma aleatoria al servidor que atenderá al cliente actual. Si se escoge el mecanismo RR, `_ip_handler` utiliza la función auxiliar `round_robin` para seleccionar al que será enviado el paquete que está siendo procesado, esta función ordena los servidores en una lista y toma el primero de esta, asimismo toma el puerto correspondiente a ese servidor y devuelve los dos valores, además emplea un contador para que en la siguiente ronda se escoja al siguiente servidor y cuando todos los servidores hayan sido escogidos, se reinicia el contador para que nuevamente se escoja el primero y continúe el ciclo. El procesamiento que se realiza para el caso basado en DRR tiene varias partes, ya que se realiza una retroalimentación entre la aplicación del controlador y el switch cada vez que llega un paquete, esta retroalimentación se da a través de mensajes que se envían en los dos sentidos. En primer lugar, el controlador envía un mensaje de petición del estado de todos los flujos en el switch. El switch le contesta con un mensaje donde están todas las estadísticas de todos los flujos. El controlador analiza el mensaje de estadísticas y calcula el número de bytes procesados por cada servidor. El servidor elegido será el que haya procesado el menor número de bytes.

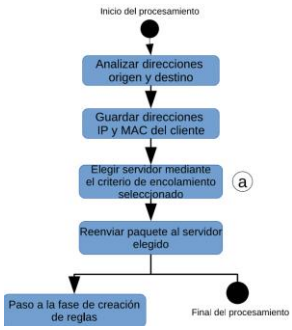

Figura 9. Diagrama de flujo del Módulo de Procesamiento

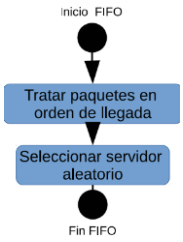

Figura 10. Diagrama de flujo del mecanismo FIFO

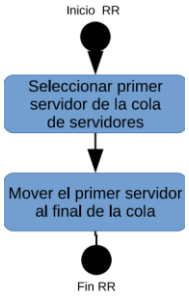

Figura 11. Diagrama de flujo del mecanismo RR

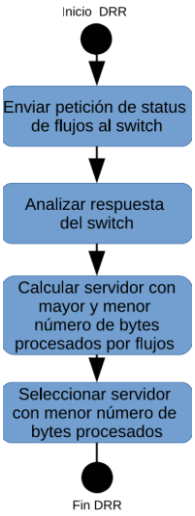

Figura 12. Diagrama de flujo del mecanismo DRR

3.6.3 Módulo de Creación de Reglas

El objetivo principal de las reglas que crea la aplicación es asignar un servidor a cada nuevo cliente, y que ese cliente solo sea atendido por el servidor asignado, mientras la conexión de ese cliente siga activa. Para la creación de las reglas se requiere:

dirección IP del cliente; dirección MAC del cliente; dirección IP del servidor asignado; dirección MAC del servidor asignado; acciones a tomar con el paquete; y, tiempo de vida del flujo. Todos estos datos son recolectados en el módulo de procesamiento, con excepción del tiempo de vida que se define en el archivo de configuración. Las reglas creadas en la aplicación se enviarán al switch para que este instale dichas reglas, de modo que los paquetes que cumplan con las condiciones del flujo sean procesados directamente por el switch y no por el controlador. La aplicación crea reglas en dos sentidos, cliente-servidor y servidor-cliente, ya que se debe procesar por separados estos dos flujos. Las acciones para las reglas de cliente a servidor son: cambiar las direcciones destino IP y MAC (direcciones publicadas) con las direcciones del servidor asignado y reenviar el paquete hacia ese servidor por el puerto correcto del switch. Las acciones para las reglas de servidor a cliente son: cambiar la dirección IP del servidor asignado por la dirección IP conocida, cambiar la dirección MAC del servidor asignado por la dirección MAC conocida y reenviar el paquete hacia el cliente. Las direcciones MAC e IP del servidor que atiende a cada cliente son asignadas por el módulo de procesamiento.

Para la creación de estas reglas se desarrolló la función `add_flow`, en esta función se crea un encabezado OpenFlow que servirá para hacer *match* con los paquetes entrantes, también se establecen las acciones que el switch tomará en caso de que un paquete haga *match*, ya sean para cambiar una dirección por otra o reenviar el paquete por un puerto específico del switch. Todos los flujos tienen el mismo proceso de creación, lo que cambia en cada uno es la regla o el encabezado OpenFlow y las acciones a realizarse con los paquetes que cumplan con la regla.

3.7 Red

Se implementaron dos redes para poder probar la aplicación. La primera red es de tipo virtual, fue utilizada para pruebas menores durante el desarrollo temprano de la aplicación; esta red se levantó en el simulador Mininet. La segunda red fue implementada usando un switch con soporte de OpenFlow.

3.7.1 Red Virtual

Para implementar la red virtual se utilizó el simulador Mininet, instalado en una máquina virtual que se puede descargar de [8]. Sobre Mininet se creó la red virtual conformada por: un switch OpenFlow y seis computadoras, tres de ellas actúan como clientes (h1 ah3) y tres como servidores (h4 a h6), el controlador corre en otra máquina virtual. Ambas máquinas virtuales se ejecutan sobre una máquina física, que posee las características que se muestran en la Tabla 1. El diagrama de la red se presenta en la Fig. 13.

3.7.2 Red Física

Para la implementación física de la red se utilizaron siete computadoras y un switch HP con soporte para el protocolo OpenFlow. Una computadora se emplea como controlador, cuatro computadoras (PC1, PC2, PC3 y PC4) se utilizan como clientes, en la sexta computadora se virtualizan tres clientes y en la última se virtualizan tres servidores web. Las características de estos equipos se resumen en la Tabla 1. Los tres servidores están virtualizados en una computadora, cada servidor web se implementó sobre una máquina virtual utilizando el Sistema Operativo Ubuntu 14.04 de 64 bits y se instaló el CMS JOOMLA. En el primer servidor web se modificó la página de inicio, se incorporó a la página un reproductor de audio y un reproductor de vídeo utilizando una extensión de JOOMLA denominada Allvideos [6]. Se agregó un enlace a un vídeo de 48.4 MB en formato MP4 con una duración de aproximadamente 25 minutos, y a una canción de 4 MB en formato MP3 de 4 minutos de duración.

Para el uso de OpenFlow en el switch es necesario configurar los parámetros: VLAN de OpenFlow, instancia OpenFlow, y la información del controlador. Al momento de realizar la configuración de este switch se debe recordar que para la conexión con el controlador no se puede usar la VLAN que está asociada a la instancia OpenFlow.

En resumen, la página web cuenta con texto plano, una imagen en el encabezado de la página, un reproductor de audio y un reproductor de vídeo. Para cada servidor se asignó 1GB de RAM, 1 núcleo de la máquina física y 25GB de disco duro. Los clientes son físicos y virtuales con sistemas operativos Windows y Linux; Los clientes Linux virtualizados son: Fedora con el navegador Chrome, Linux Mint con el navegador Chrome y Elementary OS con el navegador Midori.

4. PRUEBAS Y ANÁLISIS DE RESULTADOS

En esta sección se presentan los resultados de las pruebas que se realizaron en la red física, con la aplicación usando cada criterio de encolamiento y un breve análisis de los resultados obtenidos de estas pruebas.

Tabla 1. Características de los equipos empleados

Equipo	Controlador	PC1	PC2	PC3	PC4	PC donde se virtualizaron clientes	PC donde se virtualizaron los servidores
Sistema Operativo	Ubuntu 14.04 LTS 64 bits	Windows 7 Pro 32 bits	Windows 8.1 Pro 64 bits	Windows 7 Pro 64 bits	Windows 7 Ultimate 64 bits	VMware ESXI 5.1.0	VMware ESXI 5.1.0
Procesador	Intel Core i7	Intel Core Duo E6750	Intel Pentium D820	Intel Core 2 Quad Q8200	Intel Core 2 i3 M330	Intel Core 2 i7 3770	Intel Core 2 i7 3770
Velocidad del procesador	1,73 GHz	2.66 GHz	2.8 GHz	2.33 GHz	2.13 GHz	3.4 GHz	3.4 GHz
RAM	8 GB	2 GB	4 GB	4 GB	4 GB	8 GB	8 GB

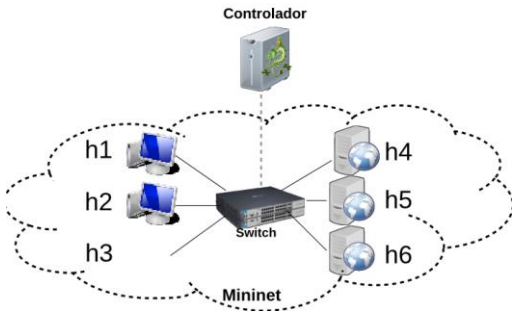

Figura 13. Red virtual implementada con Mininet

En primer lugar es necesario manipular el archivo de configuración de la aplicación para especificar: las direcciones IP y MAC de los servidores; la dirección IP conocida; la dirección MAC conocida; las direcciones IP y MAC del switch OpenFlow que se utilizarán para descartar paquetes innecesarios; los tiempos de vida de los flujos; y, la opción del mecanismo de encolamiento que se utilizará. El proceso de configuración no debería tardar más de un par de minutos ya que solo se requiere especificar alrededor de 20 valores.

Para comprobar el funcionamiento del mecanismo FIFO se realizó el siguiente procedimiento: mediante el navegador web de cada cliente se realizó un pedido a la dirección IP conocida de los servidores; se comprobó que cada cliente fue atendido en el orden que realizó la petición; y, la atención la realizó un servidor al azar como era esperado. Una vez que todos los clientes fueron atendidos se revisó la tabla de flujos del switch, en la cual se encontraron tres flujos: flujos para descartar paquetes; flujos que tienen como destino la dirección IP conocida y cuyas acciones son cambiar las direcciones IP y MAC destino y reenviar el paquete; y, flujos que tienen como origen la dirección IP de un servidor y cuyas acciones son cambiar las direcciones MAC e IP de origen y reenviar el paquete.

Para probar el mecanismo RR se realizó el mismo procedimiento que para FIFO, pero utilizando cinco clientes. Los resultados de las primeras pruebas no fueron los esperados, debido a que podían llegar varios paquetes generados por un mismo cliente, esto debido a que debe establecerse una conexión TCP la cual requiere del *handshake* de tres vías, antes de que el cliente pueda enviar un pedido HTTP, y dado que la aplicación de balanceo debe procesar el paquete y luego instalar la regla de flujo en el switch, es posible que el navegador web haya generado una nueva solicitud, asumiendo que el servidor web está ocupado o que la solicitud se perdió. Estos pedidos eran encolados y provocaban que se asignen varios servidores y se sobrescriban las reglas de flujo produciendo un funcionamiento inesperado. Por lo que fue necesario modificar la aplicación para llevar un registro de la dirección IP del cliente del paquete que está siendo procesado en este momento y se utiliza para comprobar que los siguientes paquetes IP no provengan del mismo origen del registro almacenado, si provienen del mismo origen no se les asigna un nuevo servidor. Esta comprobación resolvió el problema ocasionado por el comportamiento inesperado debido a solicitudes sucesivas desde un mismo cliente.

Para comprobar el funcionamiento del mecanismo DRR se realizó lo siguiente: el primer cliente hizo la petición a la dirección IP publicada, el controlador le asignó un servidor, una vez que el cliente estableció conexión con el servidor web asignado y comenzó a navegar entre las páginas disponibles para tener un número elevado de bytes procesados por el servidor, entonces se realizaron solicitudes desde dos clientes más, pero no se realiza navegación alguna entre las páginas disponibles, esto con el objetivo de que el primer par cliente-servidor web tenga el mayor número de bytes procesado y, finalmente, se realizaron solicitudes desde tres clientes más. Al terminar el procedimiento de pruebas, los resultados fueron los esperados, la asignación se realizó en función de los bytes procesados por cada flujo.

Se realizaron pruebas de la aplicación para comprobar ciertos aspectos como: el funcionamiento de la aplicación con HTTPS, tráfico de audio y vídeo usando HTTP y HTTPS, y los tiempos de respuesta. Para comprobar el funcionamiento de la aplicación utilizando HTTPS, simplemente se realizaron solicitudes utilizando este protocolo y se obtuvieron resultados satisfactorios. Se comprobó el funcionamiento de la aplicación con el tráfico de audio y vídeo alojados en los servidores utilizando HTTP y HTTPS. Para cada mecanismo se realizaron pruebas del tiempo que tarda en cargar la página principal del servidor web desde que un cliente realiza la solicitud, se utilizó HTTP y HTTPS para las pruebas y se realizaron las mediciones del tiempo mediante temporizadores. Se realizaron pruebas borrando la caché y el historial de navegación, y sin borrarlos. Se realizaron 20 mediciones; los tiempos promedio obtenidos para cada uno de los mecanismos (FIFO, RR y DDR) y limpiando la caché y el historial se presentan en la Tabla 2; como era de esperarse los tiempos usando HTTP son menores a los obtenidos usando HTTPS; los resultados usando cada mecanismo y sin limpiar la caché ni el historial se presentan en la Tabla 3. Los tiempos obtenidos al usar el mecanismo DDR son mayores que con los otros dos ya que el procesamiento que realiza el controlador es mayor. Debe mencionarse que para cada una de las mediciones se eliminó las reglas de flujo del switch instaladas previamente.

También cabe mencionar que al no limpiar la caché ni el historial del navegador se obtuvieron tiempos menores que en el caso en el que se los limpio, esto debido a que los navegadores guardan información en su caché, y por lo tanto la carga es más rápida. Finalmente, se realizaron pruebas de descarga de audio y vídeo, el tiempo promedio de descarga de audio para los tres clientes fue 53.87s y para descarga de vídeo fue 699s. Si se realizan peticiones posteriores desde un cliente para el cual ya existe una regla de flujo en el switch, los tiempos de carga promedio son: 0.64s para HTTP y 0.9s para HTTPS.

Al analizar los tiempos de respuesta de todos los mecanismos, se puede notar que existe un incremento de casi dos segundos al emplear HTTPS en lugar de HTTP, sobre todo en los clientes Fedora y Mint, mientras que para Elementary OS el tiempo se incrementa aproximadamente 1 segundo.

5. CONCLUSIONES

La aplicación desarrollada con Ryu realiza el balanceo de carga para servidores web, haciendo uso de tres mecanismos de encolamiento: FIFO, RR o DRR. FIFO resulta el más rápido de los tres, puesto que se escoge a un servidor de forma aleatoria. DRR resulta el más lento de los tres pues se requiere primero conocer cuál es la carga en bytes procesada por cada servidor. RR tiene un tiempo intermedio entre los dos, puesto que se debe ir manipulando una lista para la asignación.

La primera petición hacia un servidor utilizando esta aplicación, siempre será la que más tarde en ser atendida, pero una vez que se instala una regla de flujo en el switch para los paquetes que forman parte de una conexión, el tiempo de respuesta será mucho menor para la navegación en dicho servidor, esto se debe a la forma en la que la SDN funciona. La SDN provee flexibilidad en este escenario puesto que permite que el administrador decida como encolar las solicitudes de los clientes mediante un simple cambio en la aplicación. De igual forma, agregar un nuevo mecanismo de encolamiento involucra cambios en la programación, y no la adquisición de un nuevo producto y los posibles problemas que esto traiga consigo, como la instalación, mantenimiento y operación del nuevo equipo.

El trabajo futuro incluye el uso de otras técnicas de encolamiento, pruebas con alta carga, y el empleo de hilos para procesamiento paralelo.

6. REFERENCIAS

[1] Cisco, *¿Pueden las redes definidas por software (SDN) mejorar la rentabilidad del operador?* [Online]. Disponible en: https://www.cisco.com/web/ES/assets/pdf/networking_sdn_enhance_operator_monetization_wp.pdf

[2] D. Erickson, The Beacon OpenFlow Controller, HotSDN Hong Kong, 2013

[3] IBM, *Patterns for the Edge of Network*, IBM Redbooks, New York, 2002

[4] J. C. Cuéllar. *Algoritmos de planificación en redes de paquetes. S & T. Sistemas y Telemática*. [Online]. 7(14), pp. 91-107. Disponible en: http: //ow.ly/UHMBe

[5] J. J. García, *Estudio, adaptación y evaluación de algoritmos de arbitraje para entornos con control de flujo a nivel de enlace de red*, Proyecto de Fin de Carrera, Dept. Ing. Sistemas, Universidad de Castilla - La Mancha, Castilla-La Mancha, Julio 2007.

[6] JOOMLAWorks, *Allvideos*. [Online]. Disponible en: http://www.joomlaworks.net/extensions/free/allvideos

[7] M. Gómez, *Introducción, Notas del Curso: Análisis de Requerimientos*, 1st ed., vol. 1, M. Gómez, Ed. México D.F: Universidad Autónoma Metropolitana, 2011, pp. 3-11.

[8] Mininet Team. *Download/Get Started With Mininet*. [Online]. Disponible en: http://ow.ly/ UHPCL

[9] N. Benvenuto and M. Zorzi, *Queueing Theory, Principles of Communications Networks and Systems*, 1st ed., vol. 1, N. Benvenuto, Ed. New Jersey: Wiley, 2011, pp. 517-596.

[10] OpenDayLight, [Online]*;* Disponible en: https://www.opendaylight.org/about-sdn-and-nfv

[11] Open Networking Foundation, *OpenFlow Switch Specification*. [Online]. Disponible en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[12] Open Networking Foundation. *Software-Defined Networking (SDN) Definition*. [Online]. Disponible en: https://www.opennetworking.org/sdn-resources/sdn-definition

[13] P. Göransson and C. Black. Defined Networks: A Comprehensive Approach. Waltham, MA, USA: Morgan Kaufmann, 2014

[14] P. Lappas. Introducing Project Floodlight. Project Floodlight. [Online]. Disponible en: http://www.projectfloodlight.org/blog/2013/03/25/introducingproject-floodlight

[15] POX. POX Wiki. [Online]. Disponible en: https://openflow.stanford.edu/display/ONL/POX+Wiki

[16] Ryu Framework Development Community. *Components of Ryu* [Online]. Disponible en: http://ryu.readthedocs.org/en/latest/ components.html

[17] Ryu Framework Development Community. *What's Ryu?* [Online]. Disponible en: http://ryu.readthedocs.org/en/latest/getting_started.html#what-s-ryu

[18] Ryu Project Team. RYU SDN Framework. [Online]*;* Disponible en: https://osrg.github.io/ryu-book/en/Ryubook.pdf

[19] S. Bhatt. *Load Balancing and Clustering Best Practices*. 1st Ed. [Online]. Disponible en: http://pdf.th7.cn/down/files/1312/Liferay%20Portal%20Performance%20Best%20Practices.pdf

[20] T. D. Nadeau and K. Gray, *OpenFlow, SDN: Software Defined Networks*, 1st ed., vol. 1, M. Loukides and M. Blanchette, Ed. Sebastopol: O'Reilly, 2013, pp. 47-71

Tabla 2. Tiempos de respuesta para cada criterio limpiando historial y cache

Cliente	**FIFO**		**RR**		**DDR**	
	HTTP	**HTTPS**	**HTTP**	**HTTPS**	**HTTP**	**HTTPS**
Fedora	3.58 s	5.15 s	3.62 s	5.98 s	9.12 s	10.04 s
Mint	3.51 s	5.81 s	3.12 s	6.01 s	7.59 s	13.61 s
Elementary OS	4.58 s	5.01 s	4.18 s	4.54 s	8.19 s	9.29 s

Tabla 3. Tiempos de respuesta para cada criterio sin limpiar historial y cache

Cliente	**FIFO**		**RR**		**DDR**	
	HTTP	**HTTPS**	**HTTP**	**HTTPS**	**HTTP**	**HTTPS**
Fedora	2.39 s	4.51 s	3.74 s	12.22 s	8.60 s	9.07 s
Mint	2.56 s	4.30 s	4.41 s	5.16 s	6.80 s	12.02 s
Elementary OS	1.78 s	2.33 s	2.45 s	3.13 s	7.85 s	8.92 s