risti
Revista Ibérica de Sistemas e Tecnologias de Informação
Revista Ibérica de Sistemas y Tecnologías de Información

Recebido/Submission: 27/01/2019
Aceitação/Acceptance: 15/03/2019

Aplicación de Balanceo De Carga Dinámico Para Servidores, Basada En Redes Definidas Por Software.

Carlos Enrique MINDA GILCES[1], Rubén PACHECO VILLAMAR[2]

cminda@uees.edu.ec, rpachevov@uees.edu.ec

[1, 2] Universidad de Especialidades Espíritu Santo, Samborondón, Guayas, Ecuador.

DOI: 10.17013/risti.n32.67–82

Resumen: Un aspecto relevante en investigación a nivel de redes de datos es la optimización de recursos. Las redes deben gestionar grandes volúmenes de tráfico sin introducir tiempos de espera innecesarios y garantizar la disponibilidad de servicios. Este artículo propone un algoritmo de balanceo de carga basado en un sistema de criterios combinados que utiliza tecnologías de redes definidas por *software* para obtener en tiempo de ejecución, parámetros desde distintos puntos de la red y seleccionar el servidor con las mejores condiciones para responder. La aplicación propuesta está escrita en lenguaje Python utilizando el controlador Ryu; la implementación de este programa en un *switch*, lo convierte en un balanceador de carga. Se utilizó la herramienta de simulación Mininet para la evaluación resultados; el experimento demuestra que el balanceador de carga basado en criterios combinados, reduce, hasta en un 50%, el tiempo de respuesta del servidor en comparación con el método tradicional *Round Robin*.

Palabras-clave: SDN; balanceo de carga; redes; servidores; OpenFlow.

Dynamic Load Balancing Application for Servers, Based on Software Defined Networking.

Abstract: A relevant topic in current data network research is network resource optimization. Networks must cope with large volumes of traffic and ensure the availability of services without introducing unnecessary timeouts. This article presents a load balance algorithm based on *software* defined networks; it uses combined criteria to obtain at runtime, parameters from different nodes in the network to select the server with the best capability to deliver services. The proposed load balancer is written in Python language using the Ryu controller. The implementation of the python code on a *switch* allows it to act as a network load balancer. Mininet simulation tool was used in the experiment. Results demonstrate that the SDN load balancer obtains a response time up to 50% lower than *Round Robin* algorithm.

Keywords*:* SDN; load balancing; networks; servers; OpenFlow.

1. Introducción

La navegación web emplea un modelo cliente-servidor en donde un cliente solicita datos y el servidor responde. Dependiendo del tipo de servicio, pueden existir millones de conexiones al servidor, en un mismo instante. De acuerdo a (KLEINER PERKINS, 2018) hasta julio del 2018 se registraron 4,119 millones de usuarios de Internet. Esta creciente demanda genera sobrecarga; los aumentos repentinos de tráfico en la red ocasionan congestión en los enlaces, lo que a su vez compromete el despliegue de las aplicaciones y los servicios ofrecidos por los proveedores. Un estudio empírico de Kandula, Sengupta, Greenberg, Patel, y Chaiken (2009) demuestra que el 15% de la congestión en una red de centro de datos conformada por 150 *switches* y 1500 servidores dura más de 100 segundos, aun cuando muchos de los enlaces se encuentran subutilizados; esto implica una necesidad considerable de optimizar la asignación de recursos de red.

La mayoría de centros de datos opta por la implementación de un balanceador de carga con el fin de satisfacer el gran número de usuarios y los requerimientos que estos generan (Chen, Shang, Tian, & Li, 2015). Usualmente, las estrategias de balanceo implementadas a los servicios web (HTTP) suelen basarse en las métricas de enlace proporcionadas al momento de la instalación y no consideran las condiciones de red dinámicas relativas a la utilización del ancho de banda, la pérdida de paquetes y los retrasos, por lo tanto carecen de flexibilidad para ajustarse a los distintos requerimientos de la red, o del tráfico o de las aplicaciones que operan sobre ésta (Qilin & Weikang, 2015).

La problemática recae en que al momento de aplicar el balanceo, los elementos de juico disponibles para la toma de decisiones no contemplan parámetros variantes de la red que también influyen en el desempeño y pueden ser medidos dinámicamente (Kanungo, 2013), tales como la latencia, la variación de la latencia y las rutas alternas disponibles desde el punto de vista de procesos estocásticos. Como respuesta al requerimiento de alta disponibilidad, surgieron las Redes Definidas por *Software* (SDN, por sus siglas en inglés), que controlan la red de forma central o "programada" mediante aplicaciones de *software*. (Fernandez & L.Muñoz, 2016). Al implementar SDN, equipos de red de funcionalidad básica como los *switches*, son capaces de realizar funciones avanzadas.

Este artículo propone una solución de balanceo de carga que utiliza tecnologías de redes definidas por *software*. El mecanismo de balanceo de carga es aplicado a un grupo de servidores y está basado en criterios combinados, calculados a partir de la monitorización de múltiples parámetros de desempeño del servicio, que de acuerdo a estudios previos influyen en la calidad percibida por el usuario. El estudio está estructurado de la siguiente forma: La primera parte describe los Antecedentes y trabajos relacionados y la segunda el desarrollo de la investigación.

2. Antecedentes y Trabajos Relacionados

En lugar de utilizar un algoritmo estático como el *Round Robin*, los estudios de (Qilin, 2015);(Cui & Xu, 2016; Zhang & Guo, 2014) proponen a las redes definidas por *software* (SDN) como un nuevo acercamiento al balanceo de carga a través de un controlador de red centralizado. El trabajo de Fernandez & L. Muñoz (2016) menciona que existen dos métodos por los cuales se pueden agregar entradas a una tabla de flujos. La primera es de forma proactiva, esto se refiere a que la tabla es poblada por entradas mucho antes de

que los paquetes transiten por la red. El segundo método es el reactivo, en este método el dispositivo de red se inicializa sin ninguna regla de flujo prestablecida. Todo paquete entrante es enviado al controlador y posterior a tomar la decisión de qué hacer con el paquete, actualiza la tabla de flujo con la regla respectiva en el *switch*. La segunda vez que un paquete perteneciente al mismo flujo ingrese al dispositivo de red, este ya conocerá que acción debe realizar.

Por lo tanto, existen dos tipos de balanceadores de carga SDN que pueden ser desarrollados (Golinelli, 2015). La diferencia está en la manera en que el controlador aplica las reglas de flujo en el *switch*. En el método proactivo se establece previamente una división de las distintas direcciones IP que harán los requerimientos y dependiendo de la fuente se selecciona el servidor. En el método reactivo, la selección del servidor que atenderá el requerimiento es tomada cada vez que un paquete entra al *switch* y es enviado al controlador.

3. Métricas de Calidad

Según la investigación de Changhui (2016), el uso efectivo de los limitados recursos de red para satisfacer las necesidades de Calidad de Servicio de los negocios es un problema clave en el desarrollo de Internet. Conforme a las investigaciones de Kandula et al., (2009); Tashtarian, Erfanian, & Varasteh, (2018) los parámetros con mayor influencia en el establecimiento de niveles satisfactorios de QoE y QoS en una red de datos se reducen a: la capacidad de la red para acomodar el tráfico (rendimiento de procesamiento o throughput), la latencia o retraso que existe en la transferencia de

Estándar de Ancho de Banda (Throughput Standards)	**Categoría**	**Ancho de Banda Disponible (%)**
	Excelente	100
	Medio	75
	Bueno	50
	Pobre	<25
Estandár Pérdida de Paquetes (ETSI)	**Categoría**	**Pérdida de Paquetes (%)**
	Excelente	0
	Bueno	3
	Medio	15
	Pobre	> 25
Estandár Latencia (ITU-T-G.114)	**Categoría**	**Intervalos (ms)**
	Bueno	0-5
	Medio	150 – 400
	Pobre	> 400
Estandár Jitter (ITU-T-G.114)	**Categoría**	**Intervalos (ms)**
	Bueno	0-20
	Medio	20 – 50
	Pobre	> 50

Tabla 1 – Valores Basados en los Estándares revisados para Ancho de Banda, Pérdida de Paquetes, Latencia y Jitter.

datos de un punto a otro, el jitter, la variación en un periodo de tiempo que tiene la latencia y la pérdida de paquetes ocasionada por fallas en la transmisión de paquetes IP de un nodo a otro. Los estudios de (Tashtarian et al., 2018; Wawge & Tijare, 2014) expanden estas investigaciones y correlacionan el nivel de los parámetros medidos con el nivel de satisfacción que se genera en el usuario en referencia (TIPHON, 2002) como se detalla en la tabla 1.

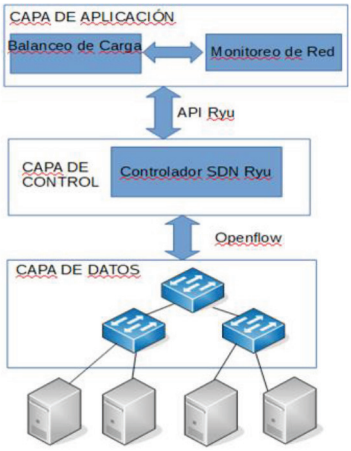


Figura 1 – Diagrama esquemático de los componentes lógicos. Fuente: Autores.

La figura 2 muestra la separación en capas en una arquitectura SDN, como se explica en el trabajo de Barrientos, Rico, Coronel y Cuesta(2019) estas capas se comunican por medio de una interfaz de programación de aplicaciones API lo cual brinda a las nuevas redes emergentes una mayor flexibilidad, fiabilidad y seguridad.

3. Desarrollo

Las métricas a tomar en consideración para el diseño de sistemas combinados son las de: latencia (ms) y ancho de banda disponible (kb/s), por la facilidad que se tiene para monitorizarlos desde el controlador. Adicional a estos parámetros de red se considera el consumo de CPU y memoria del servidor, debido a que, de acuerdo a Tashtarian et al., (2018) y Wawge & Tijare, (2014) estos afectan directamente al tiempo de respuesta del servicio, por el hecho de que de ellos depende la capacidad del servidor de procesar y atender los requerimientos. Estas métricas tienen un mayor peso en consideración con los parámetros medibles en red, ya que su influencia decrece directamente con la capacidad de la red de entregar correctamente un paquete. Para establecer los rangos permitidos para los valores medidos se hace referencia a estándares revisados. Utilizando estos estándares, los rangos y pesos para el sistema de criterios combinados escogidos se muestran en la tabla 2.

Sistema de Criterios Combinados				
	Parámetro	**Valor Óptimo**	**Valor No Óptimo**	**Incidencia en Calificación**
Servidor				
	CPU	0%	90%	50
	Memoria	0%	90%	30
Red				
	Latencia	0 ms	400 ms	10
	Ancho de Banda Disponible	100 %	50 %	10

Tabla 2 – Sistema de Criterios Combinados

3.1. Descubrimiento de equipos

Para relacionar las direcciones físicas e IP de los equipos se implementó dentro del controlador una tabla ARP. El controlador genera paquetes ARP que llevan un ARP REQUEST a todas las direcciones especificadas en la configuración. Al recibir un ARP_ REPLY se agrega en un diccionario la dirección IP y su correspondiente dirección física. La figura 3 muestra una captura del intercambio de paquetes ARP.

No.	Time	Source	Destination	Protocol	Length	Info
26	1.388584467	00:00:00_00:00:03	aa:bb:cc:dd:00:11	ARP	42	Who has 10.0.0.10? Tell 10.0.0.3
28	1.386518612	00:00:00_00:00:03	aa:bb:cc:dd:00:11	ARP	42	Who has 10.0.0.10? Tell 10.0.0.3
29	1.388603851	aa:bb:cc:dd:00:11	00:00:00_00:00:03	ARP	60	10.0.0.10 is at aa:bb:cc:dd:00:11
31	1.388579712	00:00:00_00:00:03	aa:bb:cc:dd:00:11	ARP	42	Who has 10.0.0.10? Tell 10.0.0.3
84	3.692158397	00:00:00_00:00:04	aa:bb:cc:dd:00:11	ARP	42	Who has 10.0.0.10? Tell 10.0.0.4
90	3.692162622	00:00:00_00:00:04	aa:bb:cc:dd:00:11	ARP	42	Who has 10.0.0.10? Tell 10.0.0.4
91	3.690274314	00:00:00_00:00:04	aa:bb:cc:dd:00:11	ARP	42	Who has 10.0.0.10? Tell 10.0.0.4
92	3.692187604	aa:bb:cc:dd:00:11	00:00:00_00:00:04	ARP	60	10.0.0.10 is at aa:bb:cc:dd:00:11

Figura 2 – Intercambio de Mensajes ARP. Fuente: Autores.

3.2. Cálculo de Ancho de Banda Disponible

Se utiliza una aplicación de monitoreo de tráfico desarrollada por Machi y Cheng(2018). La obtención del ancho de banda disponible se calcula en base a los valores de la capacidad de ancho de banda del *switch* y la tasa de transferencia del puerto. La capacidad de ancho de banda es consultada directamente al *switch* por el controlador por medio del mensaje OFPPORT_FEATURES. La tasa de transferencia del puerto se calcula por medio de la variación de paquetes transmitidos en un periodo de tiempo determinado, la fórmula es la siguiente:

$$Ttp = \frac{Pq_t - Pq_{t-1}}{1/t} \quad (1)$$

Donde *Pq* es el número de paquetes transmitidos y *Ttp* la tasa de Transferencia del Puerto. Luego, el ancho de banda disponible se calcula a partir de (2), donde *Cp* representa la capacidad del puerto. El ancho de banda queda expresado en las unidades de kb/s.

$$B = \frac{(Cp - Ttp)\ 8}{100} \tag{2}$$

3.3. Cálculo de Latencia

Para el cálculo de latencia se desarrolló un módulo dentro de la aplicación de balanceo de carga que construye y envía paquetes *ICMP* hacia cada dirección IP definida en el archivo de configuración del balanceador. Este cálculo se realiza desde el punto de vista del balanceador hacia los servidores, utilizando el tiempo de ida y vuelta (RTT) provisto por la aplicación PING (*ICMP*).

La Latencia, se calcula determinando el lapso transcurrido desde que el balanceador envió el paquete de prueba *ICMP* (tiempo de envío paquete), hasta el momento en que le llega la respuesta correspondiente (tiempo actual). Se adjunta fórmula utilizada:

$$l = T_{actual} - T_{ICMP} \tag{3}$$

3.4. Datos del Servidor

Con el objetivo de simplificar el diseño y evitar el uso de herramientas adicionales para la medición de las métricas del servidor se implementó una pequeña aplicación en Python. Esta aplicación utiliza la librería *psutil* para consultar los porcentajes de consumo de CPU y memoria del servidor y es compatible con sistemas operativos Windows o Linux (Rodola, 2018). Luego de consultar las métricas el controlador construye un paquete UDP con la información requerida, el cual es enviado a la dirección IP virtual designada para el balanceador de carga. Cuando este paquete llega al controlador la información de las métricas es almacenada. La figura 4 muestra una captura de tráfico en el que se verifica el envío de los paquetes hacia la dirección IP del balanceador.

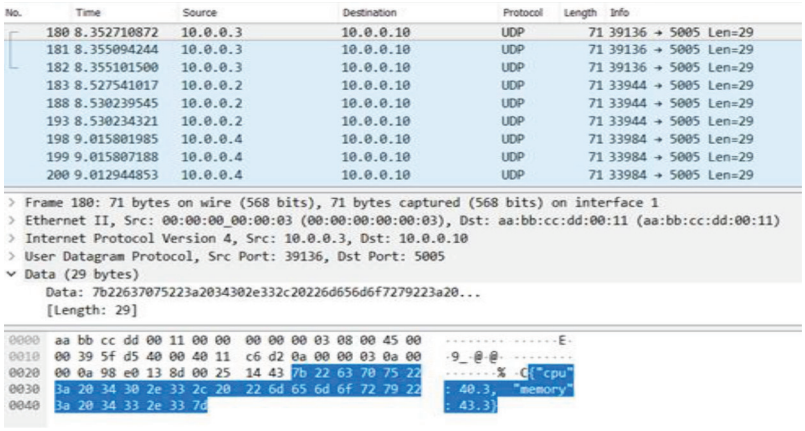


Figura 3 – Captura de tráfico UDP de los servidores al controlador. Fuente: Autores.

3.5. Normalización de datos

Debido a que las métricas que conforman el sistema de criterios combinados poseen unidades y rangos de valores distintos es necesario realizar la normalización de los datos previo a realizar el cálculo del peso de cada servidor. La función implementada dentro de la aplicación de balanceo de carga para normalizar los datos está dada por (4):

$$V_N = \left(\frac{V_M - V_{min}}{V_{max} - V_{min}}\right) 100 \tag{4}$$

Donde V*min* y V*max* son los valores mínimos y máximos respectivamente, VM el valor medido y VN el valor normalizado. De este modo, de cada métrica se obtiene una calificación sobre 100 puntos, como se muestra en la Tabla 3.

Parámetro	Valor Medido	Valor Mínimo	Valor Máximo	Valor Normalizado
Latencia (ms)	20	400	0	95.00
Latencia (ms)	50	400	0	87.50
Ancho de Banda Disponible (%)	80	50	100	60.00
CPU (%)	10	90	0	88.89
Memoria (%)	40	90	0	55.56

Tabla 3 – Ejemplo de normalización de datos medidos.

Dado que cada parámetro medido, por su nivel de importancia en el tiempo de respuesta, debe tener el peso específico que le corresponde en la determinación de la métrica combinada (peso), se utilizó una escala basada en la experiencia y el criterio de expertos, desarrollada a través de encuestas e investigaciones predecesoras, y se determinó la ponderación de acuerdo a la fórmula (5) donde W representa el Peso, *Ccpu* el consumo de cpu, *M* la memoria, *l* la latencia y *B* el ancho de banda.

$$W = 0.50C_{cpu} + 0.30M + 10l + 10B \tag{5}$$

3.6. Aplicación de Balanceo de Carga

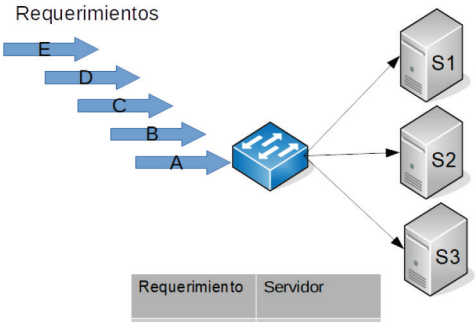
La aplicación de balanceo de carga es de tipo reactivo. Todos los paquetes entrantes al *switch* son enviados al controlador para la selección del servidor. Los paquetes entrantes levantan un evento del tipo OFPPacketIn. Si la dirección de destino coincide con la del servicio, el paquete pasa a ser procesado, caso contrario el paquete es ignorado y no se realiza ninguna acción.

El controlador verifica que el paquete corresponda a un paquete TCP y en ese momento realiza la selección del servidor. El destino escogido se decanta por el servidor que posee la mejor calificación, basado en el sistema de criterios combinados, calculado en ese instante de tiempo. Una vez seleccionado el servidor el controlador construye dos entradas con las que actualiza la tabla de flujo del *switch* utilizando la dirección IP y

dirección física del servidor seleccionado y del cliente. Una entrada corresponde al flujo cliente-servidor para los requerimientos entrantes. La segunda entrada corresponde al flujo servidor-cliente para la entrega de las respuestas del servidor.

Como se visualiza en la Figura 5, una vez establecida la conexión con el *switch*, para la monitorización de las estadísticas del equipo de red, el controlador genera un paquete del tipo *OFPPortStatsRequest*. La respuesta, un paquete *OFPPORTStatsReply*, es enviada por el *switch* al controlador y contiene información de los puertos como: la capacidad de ancho de banda, velocidad de transmisión, paquetes transmitidos y recibidos.

La información del uso de CPU y memoria llegan al controlador por medio de unpaquete UDP generado por el servidor. En este caso, el evento OFPPacketIn inicia el gestionador de paquetes UDP implementado en la aplicación del balanceo de carga y extrae de la trama UDP la información y la dirección IP de origen.



Requerimiento	Servidor
A	S1
B	S2
C	S3
D	S1
E	S2

Figura 4 – Intercambio de mensajes Openflow. Fuente: Autores.

La actualización de la tabla de flujos es realizada por medio del paquete OFPFlowMod enviado desde el controlador hacia el *switch*. Una vez ingresadas las entradas en la tabla de flujo del *switch* el paquete es enviado al destino seleccionado y se establece la conexión TCP. Una vez establecida la conexión, se realiza el intercambio de información por medio del protocolo HTTP.

3.7. Ambiente de pruebas

La aplicación de balanceo de carga fue desarrollada e implementada sobre una red virtualizada en *Mininet*,[1] utilizando *OpenVswitch* y el controlador *Ryu*. Se escogió este

[1] Mininet es un *software* de código abierto que permite la virtualización de redes y hosts para la simulación de distintas topologías de red. Esta herramienta le provee al desarrollador un ambiente de pruebas para todo tipo de aplicaciones. *Mininet* permite describir y levantar topologías de manera programática.

controlador porque de acuerdo a (Shah, Faiz, Farooq, Shafi, & Mehdi 2013) es el que cubre un mayor número de especificaciones técnicas para el desarrollo de aplicaciones. La figura 6 denota la topología del entorno de pruebas, el *OVSwitch* actúa como balanceador de carga y se conecta a los servidores S1, S2, S3. El equipo H1 funciona como cliente. Para la implementación del balanceador de carga se desarrolló el archivo *lb-topology.py* para automatizar el despliegue de los servidores, los enlaces y la inicialización del módulo de estadísticas del servidor y el servicio web.

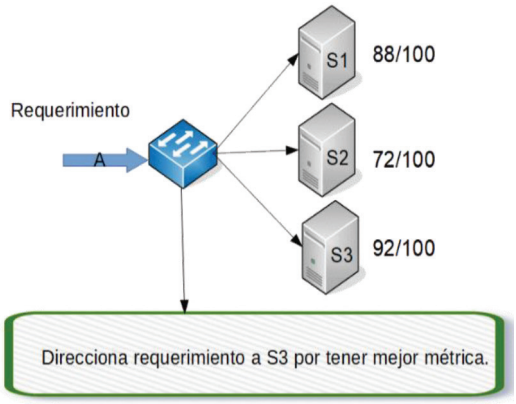


Figura 5 – Topología de Red Virtualizada en *Mininet* para entorno de pruebas. Fuente: Autores.

3.8. HTTPerf

Para la generación de tráfico se optó por utilizar la herramienta *httperf*, un *software* de código abierto que permite realizar pruebas de servicios *HTTP*. Con *httperf* se realiza

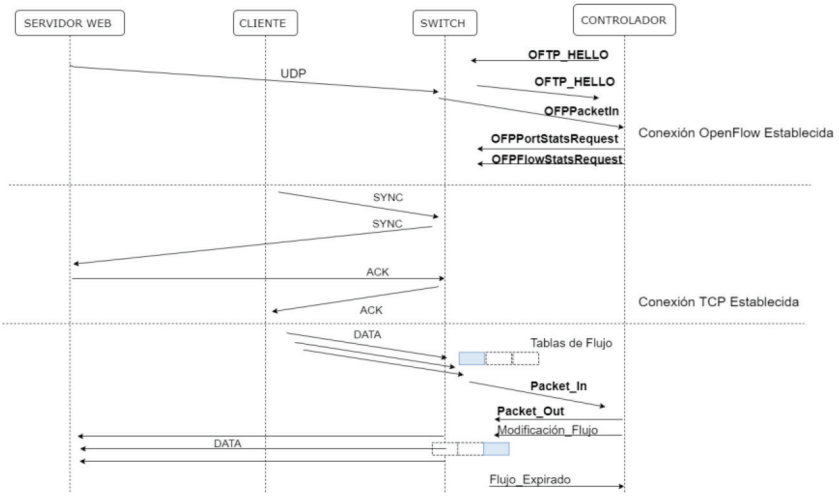


Figura 6 – Método Round Robin. Fuente: Autores.

la medición del tiempo de respuesta promedio de un servidor web y el número de requerimientos exitosos y fallidos.

El objetivo de las pruebas es constatar el funcionamiento y medir los tiempos de respuesta del balanceo de carga basado en criterios combinados. Para este efecto, se ejecutó la virtualización del balanceador y se obtuvieron varias mediciones del tiempo de respuesta bajo diferentes condiciones. Luego se implementó el método de balanceo de carga, *Round Robin*, para comparar los resultados de rendimiento de éste versus el balanceo de carga basado en criterios combinados sobre el mismo ambiente. En las figuras 7 y 8, se muestra el proceso de selección con cada técnica de balanceo.

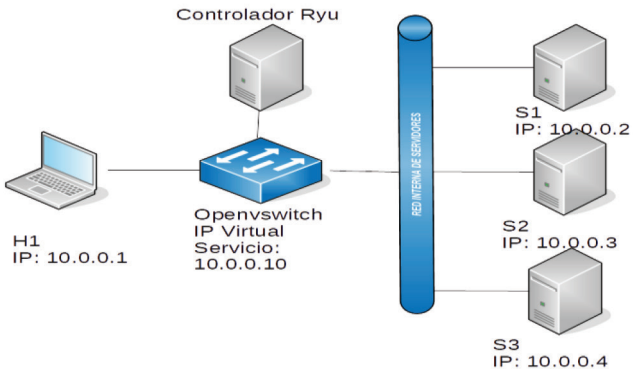


Figura 7 – Método de selección basado en el sistema de criterios combinados. Fuente: Autores.

4. Resultados

Para confirmar el funcionamiento del balanceo de carga, dentro de la etapa de implementación, se capturó el tráfico generado dentro de *Mininet* con la herramienta

No.	Time	Source	Destination	Protocol	Length	Info
2	0.000380361	127.0.0.1	127.0.0.1	OpenFl…	208	Type: OFPT_PACKET_IN
3	0.001138020	127.0.0.1	127.0.0.1	OpenFl…	206	Type: OFPT_PACKET_OUT
7	0.002323224	127.0.0.1	127.0.0.1	OpenFl…	208	Type: OFPT_PACKET_IN
8	0.002337498	127.0.0.1	127.0.0.1	OpenFl…	209	Type: OFPT_PACKET_IN
13	0.003346844	127.0.0.1	127.0.0.1	OpenFl…	206	Type: OFPT_PACKET_OUT
14	0.003368116	127.0.0.1	127.0.0.1	OpenFl…	207	Type: OFPT_PACKET_OUT
22	0.395649023	127.0.0.1	127.0.0.1	OpenFl…	178	Type: OFPT_PACKET_OUT
25	0.396019504	127.0.0.1	127.0.0.1	OpenFl…	180	Type: OFPT_PACKET_IN
26	0.396682754	127.0.0.1	127.0.0.1	OpenFl…	178	Type: OFPT_PACKET_OUT
33	1.004266277	127.0.0.1	127.0.0.1	OpenFl…	208	Type: OFPT_PACKET_IN
34	1.005127995	127.0.0.1	127.0.0.1	OpenFl…	209	Type: OFPT_PACKET_IN
36	1.006247755	127.0.0.1	127.0.0.1	OpenFl…	208	Type: OFPT_PACKET_IN
38	1.007550168	127.0.0.1	127.0.0.1	OpenFl…	206	Type: OFPT_PACKET_OUT
40	1.007589421	127.0.0.1	127.0.0.1	OpenFl…	207	Type: OFPT_PACKET_OUT

```
> Frame 40: 207 bytes on wire (1656 bits), 207 bytes captured (1656 bits) on interface 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 6633, Dst Port: 44018, Seq: 774, Ack: 955, Len: 139
> OpenFlow 1.3
```

Figura 8 – Captura de tráfico demostrando el intercambio de paquetes Openflow. Fuente: Autores.

Wireshark. En la figura 9, se visualiza el intercambio de mensajes del Protocolo Openflow. Se muestran los paquetes que corresponden a los eventos de *OFPT_PACKET_IN* y *OFP_PACKET_OUT* que confirman que los paquetes entrantes al *switch* están siendo enviados hacia el controlador. Adicionalmente, se verificó que al enviar un paquete hacia la dirección virtual, este sea redirigido hacia uno de los servidores.

La figura 10 muestra una captura de pantalla de la secuencia de eventos en ese caso. Aquí se verifica que al realizar un requerimiento desde el cliente a la dirección del servicio posteriormente se selecciona uno de los servidores para atender el requerimiento. También se puede observar el status de respuesta del paquete.

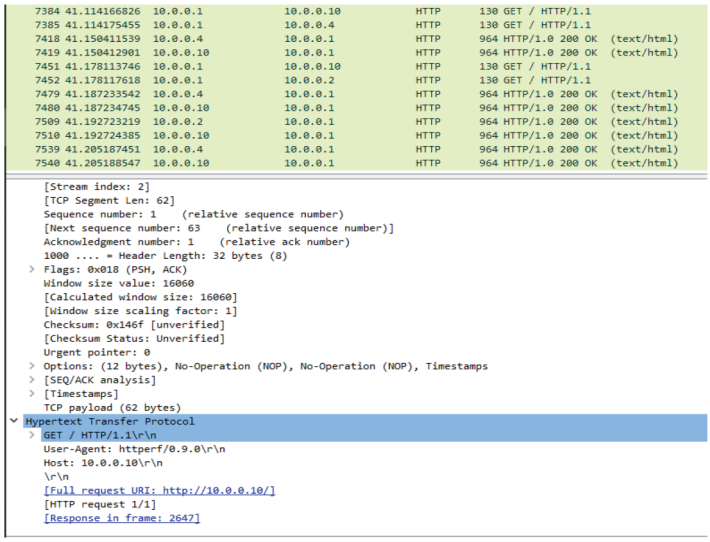


Figura 9 – Redirección de paquetes HTTP. Fuente: Autores.

4.1. Resultados Rendimiento de balanceo de carga

Con el propósito de realizar pruebas de rendimiento del balanceo de carga sobre la topología previamente establecida, se utilizó la herramienta httperf para generar tráfico HTTP hacia la dirección virtual del servicio. Para el primer conjunto de pruebas todos los enlaces están virtualizados bajo condiciones ideales, es decir con latencia mínima y cero pérdida de paquetes. En el segundo conjunto de pruebas se agrega latencia a dos de los enlaces que se conectan del *switch* al servidor.

Previo a la ejecución de cada una de las pruebas se reinició la aplicación de balanceo de carga, y la de la topología virtualizada en Mininet para evitar que datos residuales en memoria afecten el rendimiento de los algoritmos del balanceo de carga. Así mismo, previo a iniciar la herramienta HTTPerf se ejecutó un comando *pingall* en la línea de comando de *Mininet* para confirmar que todos los hosts virtualizados fueron alcanzados.

Los dos conjuntos de pruebas consisten de un número total de conexiones (100, 200 y 300) a establecerse. Para cada caso se varía la cantidad de conexiones que se establecen

por segundo. Por último se limita de manera equitativa el porcentaje de CPU al que pueden acceder los servidores para evitar que el host se sature. HTTPerf calcula un promedio del tiempo de respuesta para cada requerimiento y nos muestra una estadística del tiempo de respuesta promedio para cada prueba. Con estas estadísticas, se calculó la disminución porcentual, es decir, la proporción de la cantidad por la cual el tiempo de respuesta se ha reducido al emplear el sistema de criterios combinados en comparación con el tiempo del sistema *Round Robin*. Cuando el valor de disminución porcentual es positivo, implica que el tiempo de respuesta se ha reducido en dicho porcentaje. Cuando el valor es negativo se debe entender que ha habido un incremento en el tiempo de respuesta en dicho porcentaje. La Tabla 4 y el Gráfico 1 resumen los resultados obtenidos para el primer conjunto de pruebas en condiciones ideales.

En cualquiera de los tres escenarios (100, 200 o 300 conexiones) se puede observar que el balanceo de carga basado en el sistema de criterios combinados (línea naranja) obtiene un menor tiempo de respuesta comparado con el método *Round Robin* (línea azul) para la mayoría de tasas de conexión. Así mismo, la disminución porcentual es positiva para la mayoría de casos, indicando que el tiempo de respuesta se reduce con la aplicación del sistema de criterios combinados. En promedio, el tiempo de respuesta se reduce en un rango del 20 hasta 50% utilizando el sistema de criterios combinados. La única excepción se da cuando la tasa de conexiones por segundo es de 50. Esto puede ser atribuido al hecho de que en el ambiente de pruebas los recursos de CPU y memoria son limitados por la maquina host.

		Round Robin	Sistema de Criterios Combinados	
Total de Conexiones	Tasa de Conexiones por Segundo	Tiempo de Respuesta (ms)	Tiempo de Respuesta (ms)	Disminución porcentual
100	5	73.7	54.5	26.05
	10	70.8	31.5	55.51
	15	51.8	25	51.74
	20	65.6	40.5	38.26
	50	44	63.5	-44.32
200	5	61.9	56.2	9.21
	10	70.2	45	35.90
	15	58.6	29.7	49.32
	20	54	54.1	-0.19
	50	38.7	66.5	-71.83
300	5	59.9	48.8	18.53
	10	68.7	28.3	58.81
	15	61.5	32.7	46.83
	20	73.7	49.2	33.24
	50	49.7	68.2	-37.22

Tabla 4 – Comparativo Condiciones Ideales

El método *Round Robin* es un método simple el cuál utiliza poco recursos del controlador a diferencia del sistema de criterios combinados. Esto se debe a que a una mayor cantidad de conexiones por segundo, decrece la efectividad del algoritmo de selección propuesto ya que el ambiente sobre el cuál se está realizando la simulación debe hacer uso de tiempos de CPU y memoria del host para la virtualización de los elementos, la creación de la carga o conexiones y en generar la respuesta de los servidores.

La aplicación de balanceo de carga basado en criterios combinados, al monitorear en tiempo real las estadísticas de la red y los servidores y realizar el cálculo con las métricas obtenidas hace fuerte uso de estos recursos, y el tiempo de respuesta del servidor sufre un incremento debido al tiempo que le toma al controlador realizar los cálculos, el cual aumenta al existir menor cantidad de recursos disponibles.

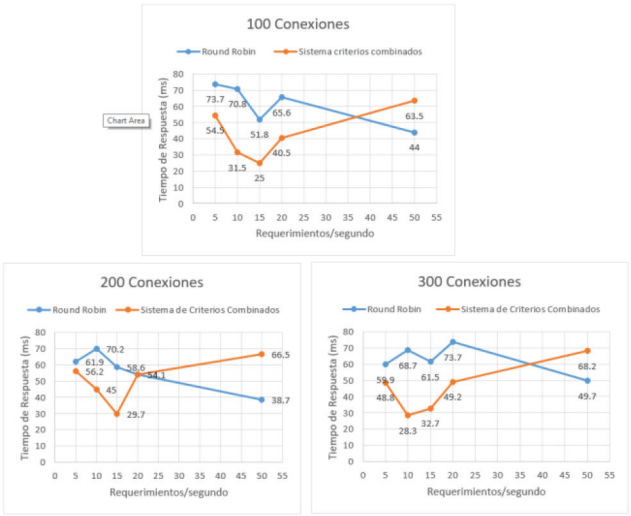


Gráfico 1 – Tiempo de Respuesta vs Requerimientos/Segundos del primer set de pruebas.
Fuente: Autores.

En el segundo conjunto de pruebas se aprovechó la flexibilidad que brinda la herramienta de Mininet para agregar latencia en los enlaces que conectan al dispositivo de red, el OVS *switch*, con los servidores. En estas pruebas a dos de los enlaces se les agregó una latencia correspondiente a 50 ms y 80ms, mientras que el enlace restante no es afectado por latencia. En la Tabla 5 y Gráfico 2 se puede observar los tiempos de respuesta obtenidos para ambos algoritmos de selección. Nuevamente se observa una disminución en el tiempo de respuesta al utilizar el sistema de criterios combinados. Incluso agregando la latencia, la disminución porcentual es positiva para casi todos los casos a excepción del de 5 conexiones por segundo, en el grupal de 100 conexiones. La tabla denota una mejora general en los tiempos de respuesta en todos los casos, reduciéndolo hasta en más del 50% del tiempo de respuesta obtenido por el método *Round Robin*.

También se observa que cuando existe una gran cantidad de conexiones por segundo el tiempo de respuesta del sistema de criterios combinados tiende a aumentar con respecto

a los casos anteriores. Aún cuando existe latencia en los enlaces el algoritmo propuesto demuestra un mejor desempeño comparado al *Round Robin*. Esto se debe, en gran parte a que, cuando se incluye latencia en los enlaces, el balanceo de carga propuesto tenderá a escoger el enlace que presente menor latencia hasta que la carga sobre este servidor cause que el consumo de CPU y memoria aumenten y reduzcan la calificación de este servidor bajo el sistema de criterios combinados, pasando a seleccionar uno de los enlaces restantes.

		Round Robin	Sistema de Criterios Combinados	
Total de Conexiones	Tasa de Conexiones por Segundo	Tiempo de Respuesta (ms)	Tiempo de Respuesta (ms)	Disminución Porcentual
100	5	72.4	74.3	-2.62
	10	147.2	92.2	37.36
	15	146.1	86.7	40.66
	20	132.3	116.1	12.24
	50	123.7	65.1	47.37
200	5	96.1	90	6.35
	10	147	93.6	36.33
	15	58.6	29.7	49.32
	20	134.6	82.4	38.78
	50	122.3	100.5	17.83
300	5	149.5	109.4	26.82
	10	154.8	80.3	48.13
	15	137.4	91.1	33.70
	20	137.3	65.9	52.00
	50	126.4	116.9	7.52

Tabla 5 – Comparativo con latencia en enlaces

5. Conclusiones y recomendaciones

El experimento demostró que el algoritmo de selección que toma en consideración parámetros de la red y servidores para la técnica de balanceo de carga permite obtener un menor tiempo de respuesta y en consecuencia, incrementa el rendimiento de los servicios web. Al comparar el sistema de criterios combinados versus el algoritmo *Round Robin*, se concluye que para entornos de red en los que existe latencia en los enlaces, el balanceador de carga basado en criterios combinados funciona mejor, con un tiempo de respuesta hasta 50% menor que el obtenido en comparación con la estrategia *Round Robin*. Adicionalmente, los tiempos de respuesta obtenidos usando el sistema de criterios combinados van desde valores de 25 a 68.2 ms en el caso de enlaces sin latencia y 29.7 a 116.9 ms en los enlaces con latencia. Valores que se encuentran por debajo al límite recomendado por Google de los 200ms para tiempo de respuesta de un servidor. Basado en esto se puede concluir que la implementación del balanceo de carga basado en criterios combinados ayuda a mejorar la experiencia de usuario al utilizar un servicio web.

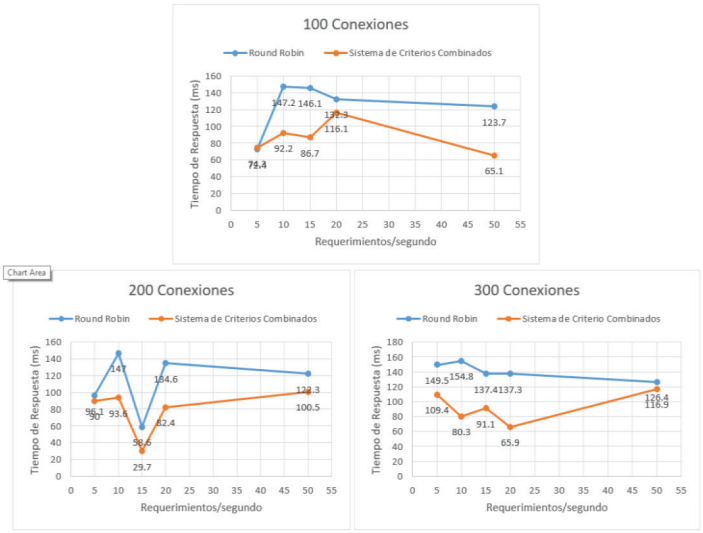


Gráfico 2 – Tiempo de respuesta vs Requerimientos/Segundos. Resultados obtenidos pruebas que incluyen latencia en los enlaces. Fuente: Autores.

El algoritmo propuesto en este documento, además de disminuir los tiempos de respuesta para proporcionar una mejor experiencia de usuario, permite también una utilización más eficiente del CPU y la memoria del servidor y por lo tanto ofrece un método de balanceo de carga más eficaz en comparación a las estrategias tradicionales.

La implementación actual del balanceo de carga basado en un sistema de criterios combinados recolecta y procesa datos utilizando la memoria del equipo en el que corre el controlador, esta configuración puede ser mejorada almacenando los parámetros monitoreados en una base de datos y no en memoria. La arquitectura SDN presenta un punto de fallo en el controlador, si éste falla, la red quedaría incapacitada, debido a esto las aplicaciones que se ejecuten dentro del controlador deben de ser eficientes en cuanto al consumo de CPU y memoria. Una solución a este problema es utilizar dos controladores dentro de la arquitectura de la red. La primera etapa de balanceo determinaría el controlador que realiza la segunda etapa de balanceo correspondiente a las aplicaciones del usuario. En trabajos futuros se sugiere modificar el esquema empleando dos controladores.

Experimentar sobre otro tipo de servicios, como FTP o flujo (streaming) de video, para comparar el desempeño del algoritmo de selección propuesto son sugerencias para trabajos de investigación futuros.

Referencias

Barrientos-Avendaño, E., Rico-Bautista, D., Coronel-Rojas, L. A., & Cuesta-Quintero, F. R. (2019). Granja inteligente: Definición de infraestructura basada en internet de las cosas, IpV6 y redes definidas por software. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação, (E17), 183-197.

Changhui, C. (2016). Research on QoS flow control method based on OpenFlow technology. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação, (17B), 14. DOI:10.17013/risti.17B.14-26

Chen, W., Shang, Z., Tian, X., & Li, H. (2015). Dynamic Server Cluster Load Balancing in Virtualization Environment with OpenFlow. International Journal of Distributed Sensor Networks, July 2015. https://doi.org/10.1155/2015/531538

Cui, C. X., & Xu, Y. Bin. (2016). Research on load balance method in SDN. International Journal of Grid and Distributed Computing, 9(1), 25–36. https://doi.org/10.14257/ijgdc.2016.9.1.03

Fernandez, C.L., & Muñoz, J. (2016). Software Defined Networking (SDN) with OpenFlow 1.3, Open vSwitch and Ryu, (June 2010), 183.

Golinelli, E. S. (2015). A Software Defined Networking evaluation approach to distributing load. Retrieved from https://www.duo.uio.no/bitstream/handle/10852/45126/Golinelli-master.pdf

Ikram, A., Arif, S., Ayub, N., & Arif, W. (2018). Load Balancing In Software Defined Networking (SDN).

Kandula, S., Sengupta, S., Greenberg, A., Patel, P., & Chaiken, R. (2009). The Nature of Datacenter Traffic : Measurements & Analysis. Provider. (pp 202–208). Microsoft Research. https://doi.org/10.1145/1644893.1644918

Kanungo, P. (2013). Study of Server Load Balancing Techniques. International Journal of Computer Science & Engineering Technology, 4(11), 1383–1389.

Kleiner, P. (2018). Internet Trends. Kpcb.

Machi, H., & Cheng, L. (2018). PureSDN. Github.Com.

Qilin, M., & Weikang, S. (2015). A Load Balancing Method Based on SDN. In Proceedings - 2015 7th International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2015. https://doi.org/10.1109/ICMTMA.2015.13

Rodola, G. (2018). Psutil. Github.Com.

Shah, S. A., Faiz, J., Farooq, M., Shafi, A., & Mehdi, S. A. (2013). An architectural evaluation of SDN controllers. In IEEE International Conference on Communications, 1, 3504–3508. https://doi.org/10.1109/ICC.2013.6655093

Tashtarian, F., Erfanian, A., & Varasteh, A. (2018). S2VC: An SDN-based framework for maximizing QoE in SVC-based HTTP adaptive streaming. Computer Networks, 146, 33–46. https://doi.org/10.1016/j.comnet.2018.09.007

TIPHON. (2002). Etsi tr 101 329-7 Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON) Release 3; End-to end Quality of Service in TIPHON systems; Part 7: Design guide for elements of a TIPHON connection from an end-to-end speech transmission per. Intellectual Property, 1, 1–72.

Wawge, P., & Tijare, P. (2014). Implementing Parameterized Dynamic Load Balancing Algorithm Using CPU and Memory, 3(5), 12828–12834.