Framework de Monitorización para Redes Definidas por Software

Lorena Isabel Barona López, Jesús Antonio Puente Fernández,
Ángel Leonardo Valdivieso Caraguay, Luis Javier García Villalba, *Senior Member, IEEE*
E-mail: {lorebaro, jesusantoniopuente, angevald}@ucm.es, javiergv@fdi.ucm.es

***Abstract*—Software Defined Network (SDN) allows the control of the network behavior by means of an external software, separating the data and the control plane in network devices. The decisions taken by the Controller depend on the accuracy of the monitoring process of the network events like link failures, delays, overhead, among others. Nowadays, there are proposals to enable the monitoring of the network; however, these solutions requires additional or customized devices. In this context, this paper proposes an SDN Monitoring Framework using OpenFlow protocol. We provide different monitoring levels in order to customize the user requirement. This framework also provides a pluggable structure that allows the easy creation, updating and deleting of high-level metrics. For monitoring purposes, we take into account some performance metrics such as data rate, loss rate and delay. The proofs of concept were made with video streaming traffic and these demonstrate the efficiency of the framework.**

***Index Terms*—Framework, Monitoring, Multimedia, OpenFlow, QoS, Quality of Service, SDN, Software-Defined Network, Video Streaming.**

I. Introducción

El desarrollo de nuevos servicios y aplicaciones en línea, tanto en terminales fijos como en dispositivos móviles, ha dado lugar a un crecimiento exponencial del tráfico que circula por Internet. Por otro lado, los equipos de red y los protocolos tradicionales no fueron diseñados para soportar un alto nivel de escalabilidad y movilidad. Asimismo, la correcta operación de una red está ligada a la capacidad de monitorización y detección de eventos específicos como retardos, pérdidas de paquetes, etc. Consecuentemente, los mecanismos existentes no proporcionan el rendimiento esperado o requieren el despliegue de nuevo hardware. Esto no sólo genera un incremento en los costes de implementación y operación, sino también en el tiempo de despliegue de los servicios requeridos por el usuario. Por tal motivo resulta indispensable la creación de un ambiente abierto que permita la rápida experimentación con nuevas tecnologías y dispositivos que no estén limitados a un fabricante y que permitan la personalización del comportamiento de la red. Una posible solución son las Redes Definidas por Software (abreviadamente SDN, *Software-Defined Networks*), que proponen una arquitectura que elimina la rigidez presente en las redes tradicionales mediante el control centralizado de la red. SDN introduce la separación del plano de control (*controlador*) del plano de datos (*conmutadores*, que se comunican a través de un canal seguro SSL, *Secure Sockets Layer*). El controlador tiene la capacidad de gestionar los conmutadores remotamente a través de una interfaz *southbound*, siendo la más conocida OpenFlow [1]. El protocolo OpenFlow es ampliamente utilizado en diferentes campos de investigación: *Data Centers*, Seguridad, *Cloud Computing* [2] [3], siendo en todos ellos un pilar fundamental el proceso de monitorización.

Si bien SDN y OpenFlow abren nuevas perspectivas en el campo de la monitorización, también introducen algunos desafíos. El principal está relacionado con la carga de trabajo tanto del plano de datos como del plano de control. Si el controlador solicita información de cada conmutador de forma continua, proporcionará datos más precisos y podrá gestionar de mejor manera la red. Sin embargo, este proceso puede generar una utilización excesiva de los recursos de procesamiento del controlador y, en consecuencia, afectar negativamente al rendimiento de tareas prioritarias. Además, los conmutadores utilizarán mayor cantidad de recursos y energía para responder a los continuos requerimientos del controlador.

Por otro lado, no todos los servicios requieren las mismas métricas. Algunos servicios (como la transmisión de vídeo) se ven afectados por el retardo, mientras que para otros (como el correo electrónico) este factor es indiferente. Esta diversidad genera la necesidad de organizar las métricas en función de los requerimientos del usuario, proporcionando una estructura escalable.

En este contexto, el presente trabajo propone un *framework* que permite reducir la carga de CPU relativa al proceso de monitorización. Para ello se propone un sistema orquestador que determina el tiempo de sondeo de información en base a la carga del controlador y al tamaño de la red, proporcionándose además una estructura modular para la creación, actualización y continua mejora de los algoritmos de monitorización. También se presentan dos algoritmos que monitorean tres parámetros: la tasa de transmisión de datos, la tasa de pérdidas y el retardo. Dichos algoritmos fueron implementados y probados con el controlador Floodlight [4], el simulador Mininet [5] y un servidor de vídeo [6].

Este trabajo se divide en seis secciones, siendo la primera la presente introducción. En la sección II se presenta el estado del arte de la monitorización en redes SDN. La sección III describe el framework propuesto. El proceso de implementa-

Grupo de Análisis, Seguridad y Sistemas (GASS, http://gass.ucm.es), Departamento de Ingeniería del Software e Inteligencia Artificial (DISIA), Facultad de Informática, Despacho 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases 9, Ciudad Universitaria, 28040 Madrid, España.

ción se explica en la sección IV. En la sección V se muestran los experimentos y los resultados de las pruebas de concepto realizadas. Finalmente, la sección VI contiene las conclusiones y el trabajo futuro.

II. Trabajos Relacionados

El campo de la monitorización ha sido ampliamente estudiado en las redes tradicionales IP. Dependiendo de la estrategia utilizada, las soluciones pueden ser clasificadas como pasivas o activas. En los métodos activos se envían los paquetes de sondeo junto con el tráfico normal a través de la red. Estos paquetes se recopilan y analizan para estimar valores como el retardo, estado de las conexiones *end to end*, etc.

Por su parte, en los métodos pasivos no se añaden paquetes a la red y solamente se analiza la información que atraviesa la misma. Estas tareas se llevan a cabo por agentes que se encuentran en los dispositivos a ser monitorizados. Dichos agentes envían esta información a través de algún protocolo de monitorización conocido como SNMP (*Simple Network Management Protocol*) [7] o NETCONF (*Network Configuration Protocol*) [8]. Se utilizan también herramientas para la recolección de información como NetFlow [9], sFlow [10] ó jFlow [11]. OpenFlow permite que los conmutadores envíen información de su estado a través del intercambio de mensajes propios del protocolo. Con dicha información el controlador estima el estado actual de toda la red.

Por ejemplo, el proyecto PayLess [12] proporciona una API (*Application Programming Interface*) RESTFul para enviar estadísticas de flujos en diferentes niveles de agregación. Asimismo, propone un algoritmo de recolección de estadísticas adaptativo, que disminuye la sobrecarga de la red. El nivel de utilización de los enlaces se obtiene a través de los mensajes *flow removed* y *statistic request*. MonSamp [13] copia y reenvía el tráfico a un agente externo para el análisis (collector & analyzer). La información recolectada por el agente se envía al controlador SDN, que utiliza un algoritmo que optimiza la monitorización modificando la frecuencia de muestreo.

OpenNetMon [14] realiza la monitorización de las estadísticas de los flujos, basándose en parámetros como el retardo o la pérdida de paquetes. La frecuencia de monitorización depende de la variación de las mediciones con respecto al valor previamente calculado. OpenTM [15] analiza diferentes estrategias que son capaces de reducir la sobrecarga en los conmutadores OpenFlow. Los experimentos muestran que la estrategia *no uniforme* proporciona una precisión razonable. Este algoritmo selecciona aleatoriamente dos conmutadores de la ruta y consulta la información de los conmutadores más cercanos al destino.

La propuesta del presente trabajo incluye encapsulación, creación de perfiles de usuario, así como la medición de diferentes parámetros (tasa de transmisión, tasa de pérdidas y retardo). Además, no duplica la información en dispositivos externos, proporcionando una estructura modular para la generación de nuevos algoritmos de monitorización. Por otra parte, el módulo orquestador tiene en cuenta no solamente la carga del controlador sino también el tamaño de la red para la gestión de los diferentes algoritmos.

III. Descripción del Framework

La principal ventaja de SDN en la monitorización de redes es la independencia del hardware, dado que no es necesaria la introducción de equipos especializados adicionales. Indiferentemente de la marca del equipo de red, si dicho dispositivo es compatible con alguna interfaz *southbound* (por ejemplo, OpenFlow), podrá ser gestionado por el controlador. La rápida respuesta del controlador es fundamental para garantizar el rendimiento de la red. Sin embargo, una monitorización excesiva de la red representa un serio riesgo tanto para el controlador como para los dispositivos de red. Asimismo, el controlador podría eventualmente utilizar la mayor parte de su capacidad de procesamiento para la monitorización, perjudicando a otras tareas. A su vez, el intercambio de mensajes de monitorización entre el conmutador y el controlador pueden generar una saturación de la red y de sus dispositivos. Otro aspecto a tener cuenta es la diferenciación de los requerimientos de los servicios, que deben formar parte del proceso de monitorización. De esta forma el controlador puede personalizar y gestionar correctamente la red.

Teniendo en cuenta todos estos aspectos, se modifica el controlador Floodlight [4], añadiéndose nuevos módulos al mismo. El detalle del framework propuesto se muestra en la Figura 1.

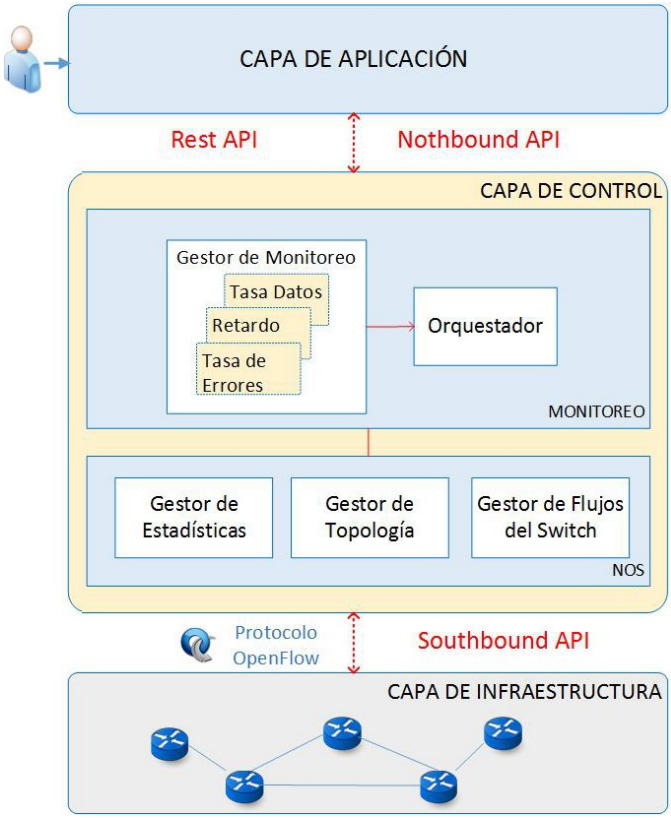

Figura 1. Framework de Monitorización.

La estructura del framework se basa en las tres capas funcionales de la arquitectura SDN:

Capa de infraestructura: contiene los dispositivos (*switches*, *routers*) responsables del reenvío de información.

Capa de control: se encarga del control de la red. Por un lado, envía instrucciones a los dispositivos de red a través del API southbound [1]. Por otro lado, permite la conexión con las funciones de la capa de aplicación a través de una interfaz northbound como Procera [16] [17] o Frenetic [18]. Los componentes funcionales de esta capa son los siguientes:

- *Gestor de Topología*: este módulo analiza los mensajes OpenFlow para identificar los equipos de red, su capacidad y sus enlaces. Esta información puede ser utilizada por otros módulos de la capa de control.
- *Gestor de Flujos del Conmutador*: recibe instrucciones del módulo de monitorización, ya sea para solicitar información de los conmutadores o para modificar la configuración de la tabla de flujos. Para realizar este procedimiento utiliza el intercambio de mensajes OpenFlow.
- *Gestor de Estadísticas*: recibe información de los mensajes OpenFlow para actualizar las estadísticas de los mismos. Esta información se utiliza por el módulo de monitorización.
- *Gestor de Monitorización*: este módulo registra y almacena los algoritmos de monitorización. Además, propone una estructura modular para que se puedan modificar o añadir nuevos algoritmos.
- *Orquestador*: se encarga de la planificación y organización de los diferentes módulos de monitorización en función del perfil de usuario. Tiene en cuenta la información del tamaño de la red y la ocupación del controlador para determinar el tiempo de monitorización de los dispositivos de la red.

Capa de aplicación: encargada de la implementación de las políticas y aplicaciones de alto nivel.

IV. Implementación

El diseño propuesto permite realizar el estudio estadístico del estado de la red, proporcionando información sobre la tasa de transmisión, la tasa de pérdidas y el retardo de los enlaces de la topología. El objetivo principal es reducir la carga tanto del plano de control como del plano de datos. Actualmente, existen controladores desarrollados en diferentes lenguajes de programación [19] [20] [21]. En el presente trabajo se utiliza el controlador Floodlight [4] basado en Java. Este controlador tiene una extensa comunidad de soporte y proporciona módulos compatibles con la versión 1.0 de OpenFlow [22]. Además, posee una API basada en Rest para la comunicación con la capa de aplicación. La implementación del framework se basa en la obtención de estadísticas y el cálculo de un tiempo óptimo de monitorización.

En primera instancia, los conmutadores y enlaces de la capa de infraestructura se representan por un grafo $G(N,A)$, donde N representa el número de dispositivos conectados y A los enlaces (arcos) entre dichos elementos. A su vez, el enlace entre los nodos i y j se representa como $arc(i,j) \in A$. Cada enlace ($arc(i,j)$) tiene asociado un coste de enlace (c_{ij}). Se asume que la red es directa (camino directo de un conmutador a otro) con un solo controlador y que no tiene ciclos negativos.

Una vez realizada la abstracción de la topología de red, el módulo orquestador se encarga de la planificación y ejecución de los diferentes algoritmos, permitiendo personalizar el comportamiento de la red $G(N,A)$. Luego, el gestor de monitorización registra y organiza los algoritmos de monitorización, de tal forma que es posible la adición de nuevos algoritmos o, en su defecto, modificar fácilmente los ya existentes. El módulo orquestador también tiene la función de controlar la carga de las tareas de monitorización, para lo cual calcula un tiempo óptimo de monitorización (t_{mon}) en función del tamaño de la red y de la capacidad del controlador (α), como se muestra en el Algoritmo 1.

Algoritmo 1: Función Orquestador

Input: Grafo $G(N,A)$
Período mínimo de monitorización t_{min}
Capacidad del controlador $0 < \alpha \leq 1$

Result: Tiempo óptimo de monitorización t_{mon}

```
1 procedure ORQUESTADOR
2     Calcular tiempo óptimo en función de la red
3     if α = 1 then
4         t_mon=t_min
      else if 0 < α ≤ 1 then
5             if A ≤ N then
6                 t_max = N * t_min
              else
7                 t_max = (N + A/N)
8         t_mon = (1 − α) * t_max
9 end procedure
```

El administrador asigna un tiempo mínimo de monitorización (t_{min}) y una capacidad de carga del controlador. Un valor de $\alpha = 1$ representa el 100% de la capacidad, mientras que un valor de α cercano a cero significa que la capacidad del controlador es limitada. Por su parte, el tamaño de la red se calcula en función del número de dispositivos (N) y de enlaces (A).

El módulo gestor de monitorización contiene tres algoritmos que permiten obtener diferentes métricas: la tasa de transmisión (ts), la tasa de paquetes perdidos (pl) y el retardo (d). En primer lugar, se comprueba si existen dispositivos conectados. Luego se activa un temporizador en función del número de dispositivos presentes y se obtienen las estadísticas de los mismos. Para el cálculo de la tasa de transmisión, el controlador envía el tiempo óptimo de monitorización (t_{mon}) a través de un mensaje OFPT_STATS_REQUEST. El conmutador responde a través de un mensaje OFPT_STATS_REPLY. El controlador utiliza la función OFP_PORT_STATS para recibir las respuestas e identificar los contadores del conmutador ($srcnode$) y del respectivo puerto ($srcport$) de la topología. Esta información

se almacena en s^k. Luego, la tasa de transmisión (ts_{ij}) por cada puerto del nodo se calcula mediante la diferencia de bytes enviados en t_{mon} (Algoritmo 2).

Algoritmo 2: Tasa Transmisión y de Paquetes Perdidos

Input: Grafo $G(N,A)$
Tiempo óptimo de monitorización t_{mon}
Result: Tasa de transmisión de cada $arc(i,j)$, ts_{ij}
Tasa de paquetes perdidos pl_{ij}

1 **procedure** CALCULAR TASAS
2 Iniciar temporizador k con período t_{mon}
3 **foreach** *período* $k = 0, 1, 2, 3, \ldots \in t_{mon}$ **do**
4 **foreach** $arc(i,j)$ **do**
5 Leer bytes enviados s_i y bytes recibidos r_j
$s^k = tx_{bytes}(src\ node,\ src\ port(i))$
$r^k = tx_{bytes}(dst\ node,\ dst\ port(j))$
6 Bytes enviados y recibidos en t_{mon}
$\Delta s_i = s_i^k - s_j^{k-1}$
$\Delta r_j = r_i^k - r_j^{k-1}$
if $k > 0$ *(en cada período)* **then**
7 Calcular tasa de transmisión
$ts_{ij} = \frac{\Delta s_i}{t_{mon}}$
8 Calcular tasa de paquetes pérdidos
$pl_{ij} = \frac{\Delta s_i - \Delta r_j}{t_{mon}}$
9 **end procedure**

Por otra parte, la tasa de paquetes perdidos se calcula de forma similar a la tasa de transmisión. Se utiliza el módulo gestor de topología para descubrir los enlaces. Adicionalmente, en esta ocasión el controlador obtiene los bytes recibidos de cada conmutador destino (*dst node*) y por cada puerto (*dst port*) de la topología. Esta información es guardada en r^k. Cada enlace $arc(i,j)$ es representado mediante la clave (*src node - src port - dst node - dst port*). La tasa de paquetes perdidos se obtiene de la diferencia (*src port/node - dst port/node*) entre los bytes enviados y los bytes recibidos en el período de tiempo de monitorización t_{mon} (Algoritmo 2).

Para el cálculo del retardo, se realizada la lectura del valor actual de la marca de tiempo t_{st} (marca de tiempo) del controlador, cuyo valor es encapsulado en un paquete de sondeo P_p. Luego, el controlador solicita a los conmutadores enviar dicho paquete por los enlaces de los nodos (conmutador origen i al conmutador destino j). El controlador envía instrucciones a cada conmutador para que identifique el paquete P_p y envíe este valor de vuelta hacia el controlador, como se muestra en el Algoritmo 3.

Para identificar el paquete P_p se utiliza el número de protocolo experimental 253. Una vez que el paquete llega al otro extremo del enlace (destino j), el conmutador identifica, encapsula y envía este paquete (pkt_{in}) de vuelta al controlador. El controlador identifica el paquete, recupera la marca inicial de tiempo t_{st} y lo compara con el tiempo real de llegada (t_{ctr}). El valor de retardo se calcula mediante la diferencia entre las dos marcas de tiempo para cada enlace $arc(i,j)$.

Algoritmo 3: Función Retardo

Input: Grafo $G(N,A)$
Tiempo óptimo de monitorización t_{mon}
Result: Retardo d_{ij} for each $arc(i,j) \in A$

1 **procedure** RETARDO
2 **procedure** ENVIAR PAQUETE DE SONDEO
3 **foreach** *período* $k = 0,1,2,3,\ldots \in t_{mon}$ **do**
4 Leer marca de tiempo actual t_{st} del controlador
5 Encapsular t_{st} y crear paquete P_p
6 Enviar P_p al conmutador origen
7 **end procedure**
8 **procedure** RECIBIR PAQUETE DE SONDEO
9 Verificar que el paquete entrante contiene P_p
if pkt_{in} *is* $(P_{p_{i\to j}})$ **then**
10 Identificar el enlace al que pertenece P_p
11 $i = pkt_{in}.getSource()$
12 $j = pkt_{in}.getDestination()$
13 Leer la marca de tiempo del controlador
14 $t_{ctr} = Date.getTime()$
15 Calcular el retardo de cada enlace $arc(i,j)$
16 $d_{ij} = t_{ctr} - t_{st}$
17 **end procedure**
18 **end procedure**

V. EXPERIMENTOS Y RESULTADOS

Para comprobar la factibilidad de los algoritmos se genera una topología lineal con el simulador Mininet [5]. Mininet es el simulador más utilizado para la experimentación en SDN. Mininet permite el despliegue de un variado número de topologías de manera automática. Para la fase de pruebas se realiza la transmisión de un archivo de vídeo entre un *host* servidor y un *host* cliente, para lo cual se utiliza el reproductor multimedia VLC [6] (Figura 2).

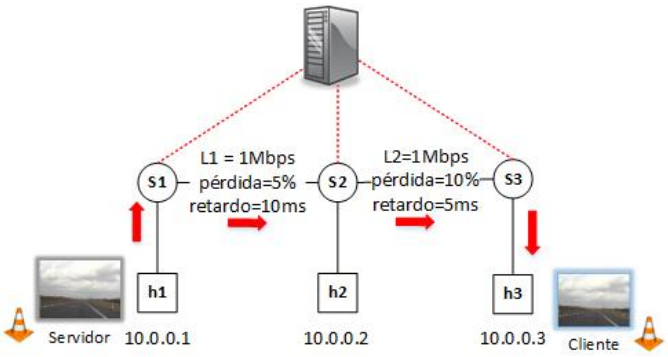

Figura 2. Topología de Prueba

La topología se compone de tres host virtuales (h_1, h_2 y h_3), tres conmutadores (S_1, S_2 y S_3) y el controlador Floodlight. Cada uno de los enlaces L_1 y L_2 (S_1-S_2 y S_2-S_3) son configurados con valores específicos de tasa de transmisión, tasa de pérdidas y porcentaje de retardo, como se muestra en la Tabla I .

Tabla I
PARÁMETROS DE LA TOPOLOGÍA

Enlace	Tasa de Transmisión	Tasa de Pérdida	Retardo
L_1	1 Mbps	5%	10 ms
L_2	1 Mbps	10%	5 ms

Se envía un archivo de vídeo de 80 segundos de duración, 2.97 MB de tamaño, 2000 fotogramas (*frames*) y en formato MPEG (highway_cif) [23] desde h_1 a h_3. Para el envío del vídeo se utiliza el protocolo RTP/UDP. El controlador Floodlight ejecuta el módulo *learning_switch* (emulación del comportamiento de un conmutador tradicional) para establecer el camino entre la fuente y el destino y se ejecuta también el módulo con los algoritmos para la obtención de estadísticas. El tiempo de monitorización del conmutador se configura en 200 ms y con una capacidad $\alpha = 1$. El experimento se repite 20 veces y con estos valores se calcula el promedio de cada una de las métricas, como se puede observar en las Figuras 3, 4 y 5.

Para todas las mediciones el controlador realiza 400 solicitudes durante el tiempo total de transmisión del vídeo (80 s), teniendo en cuenta que cada 200 ms se realiza la monitorización y el cálculo de las métricas respectivas.

La Figura 3 representa la tasa de transmisión promedio tanto del enlace L_1 como del enlace L_2. Se utiliza el Algoritmo 2 para obtener dichos valores en bps. Como es lógico, cuando inicia la transmisión se produce un incremento en el tráfico en los enlaces y una vez que termina la transmisión el controlador calcula la tasa de transmisión en ambos enlaces.

La Figura 4 representa la tasa de paquetes perdidos promedio tanto del enlace L_1 como del enlace L_2, calculada en base al Algoritmo 2. Los datos teóricos (nominales) establecen que la tasa de pérdida del enlace L_2 (10%) es mayor que la del enlace L_1 (5%). Esta tendencia se mantiene si se compara con los valores calculados por el controlador. Sin embargo, el valor medio tanto de L_1 (9.3%) como de L_2 (12.6%) se incrementa con respecto al valor nominal.

El retardo promedio tanto del enlace L_1 como del enlace L_2 se representa en la Figura 5. De igual forma que la métrica de tasa de pérdidas, el valor de retardo calculado por el controlador (Algoritmo 3) difiere de los valores nominales. En este caso la diferencia es menor. Así, el enlace L_1 varía de 10 ms (valor nominal) a 12,6 ms (valor calculado) y para el enlace L_2 los valores varían del 5 a 8,1 ms.

Dicha variación puede atribuirse a la latencia adicional que introduce el paquete de prueba (P_p) entre los conmutadores y el controlador. Por ejemplo, en la topología de la Figura 2 no se tiene en cuenta los valores de retardo procedente de la comunicación entre el controlador con el conmutador origen (controlador - S_1) y el conmutador destino (controlador - S_3). Para realizar un cálculo más preciso en [14] se propone el cálculo de dichos valores. Sin embargo, esto puede generar mayor cantidad de tráfico de monitorización y más carga de trabajo para el controlador.

Los valores obtenidos en las diferentes métricas permiten tener un control más exacto de los cambios que se producen en la red y, en consecuencia, tomar las decisiones adecuadas para reducir el impacto en el funcionamiento de la red.

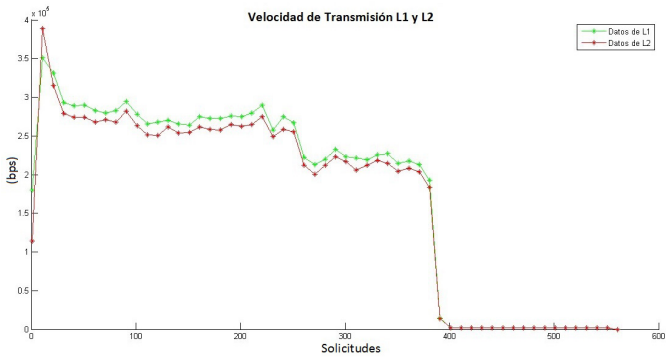

Figura 3. Tasas de Transmisión L_1 y L_2

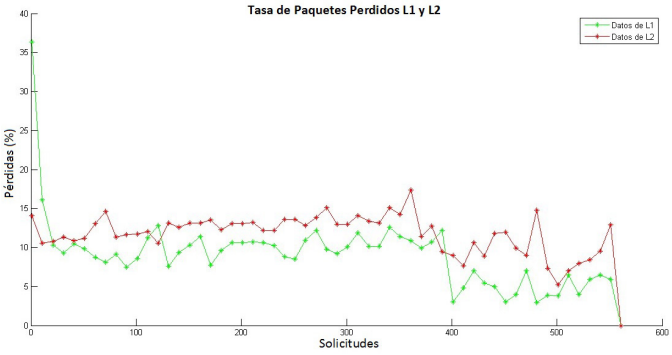

Figura 4. Tasas de Pérdidas L_1 y L_2

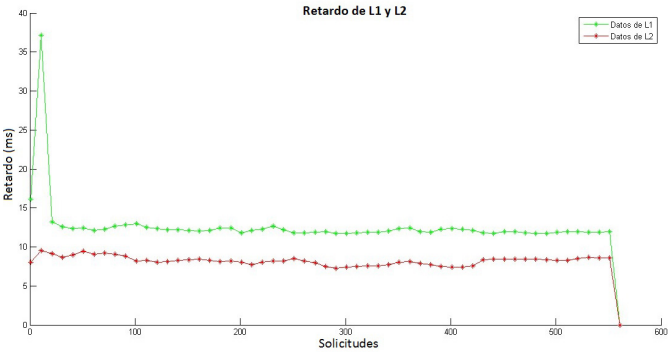

Figura 5. Retardos L_1 y L_2

VI. Conclusiones y Trabajos Futuros

El presente trabajo utiliza el paradigma de SDN como herramienta para el desarrollo de aplicaciones avanzadas de red. Se utiliza la información que brinda el protocolo OpenFlow para estimar diferentes métricas relacionadas con la situación actual de la topología y los enlaces presentes en la red. Los algoritmos presentados permiten estimar aproximadamente los valores de tasa de transmisión, tasa de paquetes perdidos y retardo presente en un enlace. Las pruebas de concepto demuestran la efectividad del framework desarrollado.

Como trabajo futuro está mejorar la solución propuesta, intentando minimizar el número de solicitudes a los conmutadores para disminuir la carga de los mismos. Además, se contempla la integración con otras soluciones de monitorización tradicionales para mejorar los resultados. Finalmente, se propone el uso de mecanismos de monitorización para la detección de amenazas de red tales como los ataques de denegación de servicio.

Agradecimientos

Los autores agradecen el apoyo brindado por el "Programa de Financiación de Grupos de Investigación UCM validados de la Universidad Complutense de Madrid Banco Santander".

Lorena Isabel Barona López y Ángel Leonardo Valdivieso Caraguay son auspiciados por la Secretaría Nacional de Educación Superior, Ciencia y Tecnología e Innovación SENESCYT (Quito, Ecuador) bajo el Programa de Becas Convocatoria Abierta 2013 y 2012, respectivamente.

Referencias

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, March 2008.

[2] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 493–512, February 2014.

[3] A. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona López, and L. J. García Villalba, "SDN: Evolution and Opportunities in the Development IoT Applications," *International Journal of Distributed Sensor Networks*, vol. 2014, article ID 735142, May 2014.

[4] "Floodlight." http://www.projectfloodlight.org/.

[5] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for. Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, (New York, NY, USA), pp. 1–6, ACM, October 2010.

[6] "Videolan." http://www.videolan.org/index.es.html.

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol (SNMP)." RFC 1157 (Historic), May 1990.

[8] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)." RFC 6241 (Historic), June 2011.

[9] B. Claise, "Cisco Systems NetFlow Services Export Version 9." RFC 3954, October 2004.

[10] P. Phaal and M. Lavine, "Sflow version 5," July 2004.

[11] A. C. Myers, "JFlow: Practical Mostly-static Information Flow Control," in *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, (New York, NY, USA), pp. 228–241, ACM, January 1999.

[12] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A Low Cost Netowrk Monitoring Framework for Software Defined Networks," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, (Krakow, Poland), May 2014.

[13] D. Raumer, L. Schwaighofer, and G. Carle, "MonSamp: A Distributed SDN Application for QoS Monitoring," in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 961–968, IEEE, September 2014.

[14] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks," in *Proceedings of the Network Operations and Management Symposium (NOMS)*, pp. 1–8, IEEE, May 2014.

[15] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks," in *Proceedings of the 11th International Conference on Passive and Active Measurement*, pp. 201–210, Springer, January 2010.

[16] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Communications Magazine*, vol. 51, pp. 114–119, February 2013.

[17] A. Voellmy, H. Kim, and N. Feamster, "Procera: A Language for High-Level Reactive Network Control," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, pp. 43–48, ACM, August 2012.

[18] N. Foster, A. Guha, M. Reitblatt, A. Story, M. Freedman, N. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison, "Languages for Software Defined Networks," *IEEE Communications Magazine*, vol. 51, pp. 128–134, February 2013.

[19] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 87–98, April 2014.

[20] D. Erickson, "The Beacon Openflow Controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pp. 13–18, ACM, August 2013.

[21] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," in *Proceedings of the ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 105–110, July 2008.

[22] "OpenFlow Switch Specification v.1.0.0," February 2011.

[23] "Highway." http://www2.tkn.tu-berlin.de/research/evalvid/qcif.html.