JITEL 2017
XIII Jornadas de Ingeniería Telemática
27-29 Septiembre 2017, Valencia (España)

Actas de las XIII Jornadas de Ingeniería Telemática (JITEL 2017), Valencia (España), 27-29 de Septiembre de 2017.

ISBN: 978-84-9048-595-8
DOI: http://dx.doi.org/10.4995/JITEL2017.2017.6620

Diseño y evaluación de un servicio OpenFlow de provisión de Calidad de Experiencia sobre Mininet

Cristian Alfonso Prieto Sánchez, Pilar Andres-Maldonado, Jonathan Prados-Garzon, Juan José Ramos-Munoz,
Departamento de Teoría de la Señal, Telemática y Comunicaciones,
Universidad de Granada
Calle Periodista Daniel Saucedo Aranda s/n, E-18071 (Granada)
rps@ugr.es, pam91@correo.ugr.es, jpg@ugr.es, jjramos@ugr.es

***Resumen*—Las redes definidas por software suponen un nuevo paradigma de red que potencialmente pueden proporcionar nuevas soluciones para proporcionar calidad de experiencia.**

En este trabajo se propone un diseño de aplicación de redes definidas por software para proveer calidad de experiencia a flujos con distintos requisitos de red. Esta aplicación se implementa en OpenDayLight, una implementación de referencia, y se ejecuta en el emulador Mininet.

En este artículo se detallan los algoritmos implementados para proporcionar la aplicación para OpenDayLight.

***Palabras Clave*—Controlador, Dijkstra, Java, Mininet, OpenDayLight, OpenFlow, QoE, QoS, Redes, Routing, SDN.**

I. INTRODUCCIÓN

El crecimiento del tráfico multimedia experimentado en los últimos años, unido al crecimiento exponencial esperado (como las previsiones realizadas por Cisco en [1]), ahondan en la necesidad de diseñar una arquitectura de red capaz de soportar y cumplir los requisitos de calidad de experiencia (QoE) para soportar el tráfico multimedia y los nuevos tipos de aplicaciones.

Por otro lado, es recurrente la búsqueda de técnicas o arquitecturas de red capaces de reducir costes y aumentar capacidades sobre las que ofrecer nuevos servicios a los usuarios, por parte de operadores de red. Estas dos necesidades básicas en la industria han impulsado la adopción de redes definidas por software (Software Defined Networking, SDN) como solución con mayor aceptación, dadas las ventajas en las se hará hincapié a continuación.

La solución diseñada en este proyecto se basa en aprovechar las ventajas que ofrece disponer de un elemento de toma de decisiones centralizado, el controlador SDN, que tiene una visión global de toda la red, para poder desplegar rutas que satisfagan los requisitos de calidad de experiencia (QoE) de varias clases de flujos multimedia.

Concretamente, se ha adaptado un protocolo de encaminamiento de *estado de enlace*, para generar las rutas por cada flujo multimedia que llegue a la red SDN. Para ello, el protocolo de encaminamiento toma como costes distintas métricas, que dependen del tipo de flujo para el que se crea la ruta, y sus requisitos. Así, tras identificar el tipo de un nuevo flujo, se calculará y configurará una ruta a su destino, que minimice los parámetros de red que degradan la calidad que percibirá el destinatario.

Además, el servicio diseñado monitoriza el estado de los enlaces para recalcular nuevas rutas, por tipo de aplicación, si las condiciones de red empeoran.

II. DISEÑO DE LA SOLUCIÓN

La propuesta actual se basa en el funcionamiento de las aplicaciones SDN, en las que un controlador lógicamente centralizado monitoriza y programa los conmutadores SDN. De esta manera, las decisiones del controlador se pueden realizar con información completa de la red y, consecuentemente, se podrían tomar decisiones óptimas.

La solución propuesta se compone de varios elementos, que se especifican en las siguientes subsecciones:

A. *Clasificación de flujos paquetes*

Para poder seleccionar la ruta que mejor se ajuste a los requisitos de QoE de un flujo de paquetes, es necesario identificar a qué tipo de aplicación corresponde dicho flujo. La clasificación de paquetes se hace de acuerdo a la figura 1, atendiendo a las características bien conocidas de cada tipo de paquete.

Inicialmente, en este trabajo se distingue entre tráfico TCP, ICMP, y RTP con perfil de audio y vídeo.

B. Algoritmo de encaminamiento

Una vez reconocido el tipo de flujo, es necesario ejecutar un algoritmo que proporcione una ruta hacia el destino del flujo. Los parámetros de QoS de los enlaces que componen dicha ruta debería maximizar la calidad que el usuario final al que va destinado el flujo percibirá. Para ello hace falta estimar previamente cuál es el coste de cada enlace para el algoritmo. A partir de la estimación de los pesos para cada enlace, se aplica el algoritmo Dijkstra [2] que nos dará la "mejor" ruta en el momento de consulta.

C. Definición de la matriz de costes por enlace

El algoritmo seleccionado para construir las rutas para cada flujo necesita una representación de la red (qué conmutadores SDN y qué enlaces existen). Esta representación será la matriz de costes. Para calcular dicha matriz, es necesario definir una métrica de calidad que se base en los parámetros objetivos de cada enlace, y que tenga en cuenta el tipo de flujo que debe seguir dicha ruta. La estimación de la matriz de costes se realiza con cuatro parámetros de calidad, que, para un enlace $Edge$, dan el peso total $C_{[}Edge]$ calculado en 2:

$$\begin{aligned} C_{[}Edge] = \alpha \cdot C_{latency}[Edge] \\ +\beta \cdot C_{jitter}[Edge] \\ +\gamma \cdot C_{load}[Edge] \\ +\omega \cdot C_{loss}[Edge] \end{aligned} \quad (1)$$

donde $C_{latency}[Edge]$ representa la componente del coste debido a retardo, $C_{jitter}[Edge]$ la componente del coste debido a la variación de retardo entre paquetes (*jitter*), y $C_{load}[Edge]$ la componente del coste debido a la carga en el enlace.

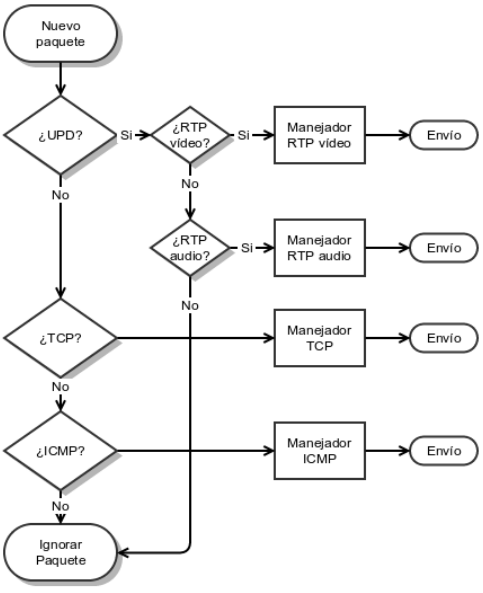

Fig. 1. Procedimiento de clasificación de los flujos de paquetes.

Cada componente está ponderado con un factor que representa la influencia de dicho parámetro sobre el coste final, y dependen del tipo de tráfico que demanda la nueva ruta.

D. Estimación de los parámetros de calidad de servicio por enlace

Para extraer los parámetros objetivos de calidad de cada enlace, es necesario definir cómo calcularlos a partir de la información que proporciona cada conmutador. El cálculo de los parámetros de calidad dará la información necesaria para el cálculo final del coste de cada enlace, y por tanto, de la estimación de la matriz de tráfico.

III. Función de evaluación de costes

Para la evaluación de costes de cada enlace E se propone usar 4 funciones diferentes, cada una relacionada con el tipo de evaluación que realiza: función de evaluación de latencias ($C_{latency}[E]$), función de evaluación de *jitter* ($C_{jitter}[E]$), función de evaluación de pérdidas de paquetes ($C_{loss}[E]$) y función de evaluación de carga ($C_{load}[E]$). Estas cuatro funciones se normalizan para que devuelvan valores de coste entre 0 y 100. Una vez definidas estas funciones de coste, es necesario ajustar los pesos α, β, γ y ω de cada uno de dichos componentes para obtener el coste total estimado para cada tipo de tráfico.

- *Función de evaluación de latencias*. Para este cálculo se empleará la latencia media estimada por un módulo de recolección de estadísticas. La latencia mínima encontrada en la red tendrá asociado un coste de 1. También se tendrá en cuenta la latencia máxima encontrada en la red pues será necesario acotar el coste resultante entre los límites establecidos (1 y 100), siendo en este caso 100 el coste máximo.
- *Función de evaluación de jitter*. La detección de *jitter* se lleva a cabo usando los valores instantáneos y medias de latencia obtenidas. Esta detección tiene ciertas complicaciones existen variaciones que no dependen únicamente del enlace en las medidas, sino que también dependen del tiempo de procesamiento en el controlador, como se comprobó experimentalmente en pruebas preliminares. Otro inconveniente de estas medidas está provocada por la relación entre *jitter* y las latencias extremo a extremo. Por ejemplo, un *jitter* de un milisegundo en una red con latencias promedio de 10 milisegundos tendría una relevancia muy importante. Sin embargo, este mismo *jitter* de un milisegundo en redes con latencias de 500 milisegundos tendrá una importancia mucho menor. No obstante, si solo se consideran *jitter* mínimo y máximo para el cálculo, se obtendrá el mismo coste para el enlace en ambos casos.
  Para solucionar estos dos problemas se establecen las siguientes medidas:
  1) Limitar una diferencia mínima entre los valores máximos y mínimos de *jitter*, a partir de la cual comenzar a calcular costes prefijados (si la diferencia fuera menor, los enlaces tendrían un coste asociado al *jitter* igual para todos).

2) Escalar el coste del *jitter* en función de las latencias máximas y mínimas encontradas, de modo que un *jitter* muy grande en una red de latencias pequeñas (y cercanas), será mucho más significativo en cuanto a coste que un *jitter* grande en una red lenta y de grandes diferencias.

Con estas consideraciones se ha decidido calcular el *jitter* como la ecuación 2:

$$jitter = |latencia_{instantanea} - latencia_{promedio}| \quad (2)$$

- Función de evaluación de pérdidas.
  Para poder evaluar las pérdidas en la red, se usará el porcentaje de pérdidas experimentados en el enlace. Este porcentaje equivaldrá al coste de forma directa, con lo que estará limitado entre 0 y 100. Para obtener el valor asociado al porcentaje de pérdidas se hará uso de las estadísticas mencionadas hasta ahora, y se usará la ecuación 3.

$$C_{loss}(E) = \frac{sentBytes(E) - receivedBytes(E)}{sentBytes(E)} \cdot 100 \quad (3)$$

IV. Recolección de estadísticas

Para recolectar las estadísticas, se hará uso de la API (*Application Programming Interface*) que proporciona el controlador OpenDayLight [3].

El controlador *OpenDayLight* cuenta en su núcleo con un recolector de estadísticas que almacena información sobre todos los elementos de red susceptibles de contar con estadísticas de red (paquetes enviados por un puerto, bytes recibidos por un conmutador, paquetes que pertenezcan a un flujo instalado en un conmutador, etc.). Haciendo uso de este almacén de estadísticas conseguiremos obtener datos que se usarán para la construcción de los costes asociados a cada enlace.

V. Detección y actualización de cambios en la topología

En esta sección se describe el proceso que consigue responder ante cambios en la red tales como caídas de enlaces o nodos. Es uno de los pilares en los cuales se apoya todo el proceso para proveer de QoE y QoS a las aplicaciones que usan los usuarios. Para conseguir el objetivo, el procedimiento diseñado se divide en dos fases: detección y actualización de rutas.

A. *Detección de cambios*

Mediante el procedimiento de detección de cambios de la topología de red, la solución propuesta es capaz de lanzar el procedimiento que actualice la información sobre la red, y recalcular las rutas que sean necesarias. Estos cambios comprenden la modificación de las características de los enlaces, la caída de nodos, etc. Si bien el propio controlador *OpenDayLight* puede detectar los cambios en la topología (existe un módulo capaz de avisar a las aplicaciones de red de cambios en la topología), se decidió prescindir de esta posibilidad, ya que nuestro diseño contempla el tratamiento de los flujos en el mismo módulo donde se reciben los paquetes.

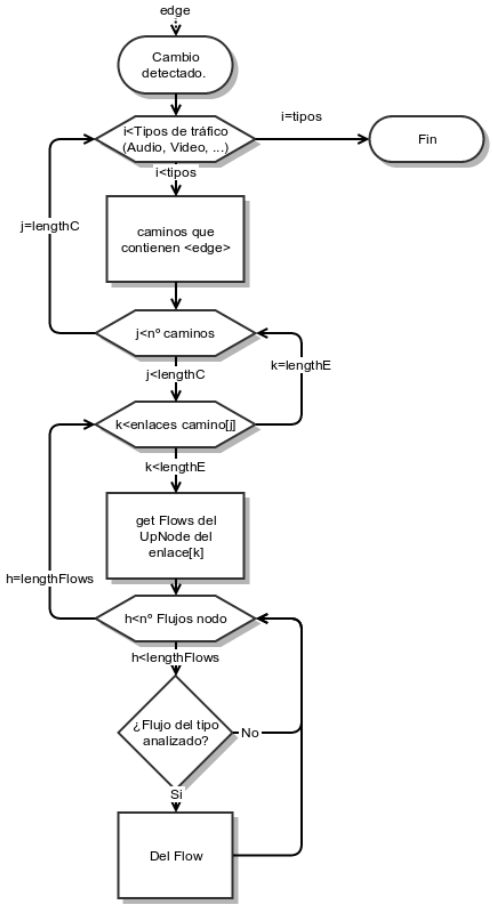

Fig. 2. Diagrama de flujo del proceso de actualización de topología

La detección se lleva a cabo mediante consultas sucesivas al estado de la red. El tiempo entre consultas es el mismo tiempo que el de actualización de estadísticas, t_{update}. El valor de este tiempo de actualización se propone como 100 ms, pudiendo ser modificado según las necesidades de red o usuario. La propuesta está basada en:

- Se establece un período de 100 ms, que en la experimentación preliminar se observó que era suficiente para que se completara la consulta sobre cambios en la topología que proporcionaba ODL.
- No se pretende sobrecargar el controlador eligiendo un tiempo menor, puesto que podría darse una situación en la cual se están actualizando estadísticas, mientras se detectan cambios en la topología, provocando dos accesos a un dato al mismo tiempo. Suponemos que con esta elección de periodos de tiempo se garantiza que el tiempo de procesamiento es mucho menor a la actualización, evitando posibles problemas.

B. *Actualización de rutas*

El proceso de actualización para las rutas sigue el flujo presentado en la figura 2.

1) Se recibe el enlace que ha cambiado según la detección (cuando un nodo cae, todos sus enlaces son detectados y pasan por este proceso de forma individual).
2) Por orden se buscan caminos afectados por cada uno de los 4 tipos de tráfico afectados.
3) Por cada tipo de tráfico:
   a) Se comprueba si alguno de los $lengthC$ caminos que contienen el enlace, están afectados por el cambio.
   b) En caso positivo se recorren los nodos del camino, compuesto de $lenghtE$ enlaces. Suponemos estos caminos ordenados, al haber sido almacenados tras una reordenación.
   c) En cada nodo se comprueban los $lenghtFlows$ flujos instalados buscando flujos coincidentes con los que se deben eliminar. Por cada flujo se comprueba si su acción de envío coincide con el enlace que corresponda del camino. Se podría intentar eliminar solo el enlace afectado, pero en ese caso surgen problemas con el cálculo de caminos extremo a extremo al haber flujos instalados ya.
4) Finalmente se devuelve el control al programa de detección que actualizará mapas y estadísticas de de los manejadores.

VI. Entorno experimental

Para evaluar el funcionamiento del servicio propuesto, se elige el emulador de redes basadas en software Mininet [4], y el controlador OpenDayLight.

Además, se configura mediante guiones programados en Python, la topología representada en la Figura 3. El objetivo es tener distintas rutas posibles a la hora de ejecutar los algoritmos de reencaminamiento, a la hora de adaptarse cuando caiga o se modifique uno de los enlaces.

Además, para comprobar el funcionamiento, se generan tráfico de audio y vídeo mediante la herramienta VLC.

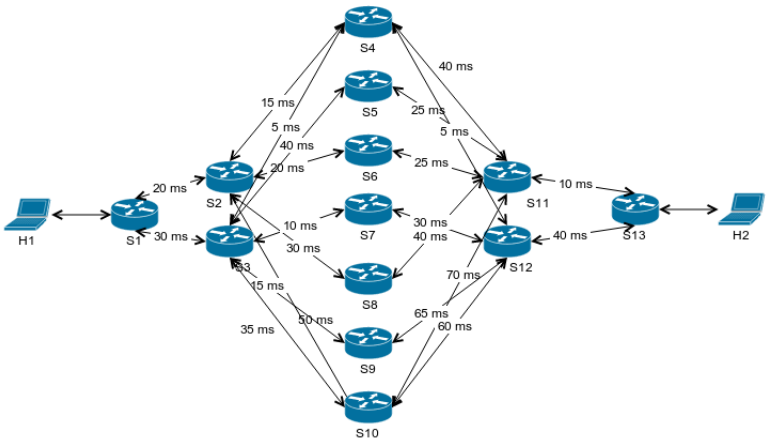

Fig. 3. Topología para realizar evaluación sobre el sistema.

VII. Conclusiones

En este trabajo se presenta el diseño de una aplicación SDN que proporcione calidad de experiencia a flujos con distintos requistos. Para ello, se diseña un módulo para OpenDayLight que monitoriza los enlaces de la red, y reprograman los encaminadores para crear nuevas rutas que cumplan los requisitos de los flujos multimedia.

Resultados preliminares muestra que la implementación permite la atomatización de este servicio en la red, pues se consiga encaminar en tiempo real y reaccionar ante caídas de enlaces, de manera automática. No obstante, aún es necesario ejecutar una batería de experimentos que permitan obtener resultados sólidos.

VIII. Agradecimientos

Este trabajo está parcialmente financiado por el Ministerio de Economía, Industria y Competitividad y el Fondo Europeo de Desarrollo Regional FEDER (proyectos TEC2016-76795-C6-4-R y TIN2013-46223-P).

Referencias

[1] CISCO, "Cisco visual networking index: Forecast and methodology, 2014-2019," May 2015.
[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Dijkstra's algorithm," in *Introduction to Algorithms 2nd edition*. MIT Press, ch. 24, pp. 595–599.
[3] "Página oficial de opendaylight." [Online]. Available: https://wiki.opendaylight.org/view/Main_Page
[4] 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*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466