Universidad Zaragoza
1542

Trabajo Fin de Grado

Estudio de redes definidas por software (SDN) y su aplicación a la Ingeniería de Tráfico

Study of Software-Defined Networking (SDN) and its application to Traffic Engineering

Autor

Alfonso Cay Delgado

Director

María Canales Compés

Escuela de Ingeniería y Arquitectura / Universidad de Zaragoza

2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Alfonso Cay Delgado,

con nº de DNI 73229694V en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Estudio de redes definidas por software (SDN) y su aplicación a la Ingeniería de Tráfico,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 22 de noviembre de 2018

Fdo: Alfonso Cay Delgado

Estudio de redes definidas por *software* (SDN) y su aplicación a la Ingeniería de Tráfico

RESUMEN

La complejidad física de la red actual y la gran cantidad de protocolos existentes en entornos en los que abundan las implementaciones propietarias dificultan en gran medida una gestión eficiente de la red. Esto ha hecho que se opte por la búsqueda de una solución más flexible y que permita gestionar de manera eficiente los recursos disponibles. De este modo, debido a las necesidades actuales, nacen las redes definidas por *software* (*Software Defined Networking* (SDN)). Este nuevo paradigma introduce una nueva forma de ver las redes, haciendo que el plano de datos y el de control se encuentren totalmente desacoplados. Así, SDN permite una centralización de la red por medio de un controlador central, además de facilitar en gran medida la gestión de los equipos en entornos *multi-vendor* gracias al protocolo OpenFlow. Todo esto hace que la complejidad de las redes se vea reducida y se incremente su agilidad y escalabilidad. Además, el proceso de innovación en este campo se ve beneficiado en gran medida debido al hecho de tratarse de redes programables.

En este TFG se ha estudiado la aplicación de las redes definidas por *software* a la Ingeniería de Tráfico (*Traffic Engineering* (TE)), que engloba la aplicación de la tecnología y los principios científicos a la medida, caracterización, modelado y control de tráfico de Internet. Se expondrán las limitaciones de las redes actuales y cómo SDN es capaz de solventar algunos de estos problemas mediante la integración de las ventajas que introduce este nuevo paradigma con las de las tecnologías utilizadas hasta ahora. De esta unión nacen arquitecturas como la basada en PCE (*Path Computation Element*), que permitirá introducir en el funcionamiento de SDN este elemento, encargado exclusivamente del aprovisionamiento de recursos a los equipos de red. Esta arquitectura servirá como referencia para realizar una implementación propia, que se utilizará para estudiar y analizar el potencial de la definición de redes mediante *software* y el control centralizado que permiten realizar sobre la red. De esta manera, se pretenden sentar las bases para futuros trabajos, creando herramientas modulares y flexibles que permitan añadir complejidad al entorno fácilmente.

En cuanto al desarrollo del trabajo, en primer lugar, se ha seleccionado el *software* necesario para poder realizar la implementación del escenario escogido. Mininet, un *software* especialmente orientado al estudio de SDN y el protocolo OpenFlow, ha permitido la creación de topologías de red virtuales controladas mediante el controlador de código abierto OpenDaylight (ODL), mientras que los demás módulos se han implementado mediante *script* en Python. Verificado el funcionamiento de la implementación realizada, comparando los resultados con escenarios de referencia sin SDN, se ha estudiado la viabilidad de la solución para realizar Ingeniería de Tráfico. Para ello, se ha estudiado el *overhead* de señalización generado en diversos escenarios, así como la capacidad de adecuar la provisión de recursos en la red a demandas dinámicas del tráfico y anomalías detectadas en la red, proporcionando selectivamente los caminos necesarios. En conclusión, se ha confirmado la flexibilidad de SDN, así como su potencial escalabilidad, generando un adecuado punto de partida para nuevos estudios de mayor profundidad.

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres y a mi hermano el apoyo y las fuerzas que me han dado en todo momento para llegar hasta aquí. Gran parte de lo que consigo es siempre gracias a vosotros.

En segundo lugar, agradecer a María Canales la implicación durante el desarrollo del trabajo, siempre muy pendiente y dispuesta a ayudar en lo que ha hecho falta.

Por último, mencionar a mis amigos. A los que llevan toda la vida conmigo y a los que han aparecido en estos últimos cuatro años en Zaragoza, en especial, a Asier Carbonel, un apoyo importante durante este tiempo.

Índice

Índice de Figuras

Lista de Acrónimos

ABNO: Application – Based Network Operations

API: Application Programming Interface

ARP: Address Resolution Protocol

ATM: Asynchronous Transfer Mode

CSPF: Constraint Shortest Path First

FEC: Forwarding Equivalence Class

GUI: Graphical User Interface

HTTP: Hypertext Transfer Protocol

IP: Internet Protocol

JSON: JavaScript Object Notation

LLDP: Link Layer Discovery Protocol

LSP: Label Switched Path

LSR: Label Switching Router

MPLS: Multiprotocol Label Switching

MPLS-TE: Multiprotocol Label Switching – Traffic Engineering

NBI: North – Bound Interface

NFV: Network Functions Virtualization

ODL: OpenDaylight

ONF: Open Networking Foundation

ONOS: Open Network Operating System

OSPF: Open Shortest Path First

OVS: Open vSwitch

PCC: Path Computation Client

PCE: Path Computation Element

PCEP: Path Computation Element Protocol

QoE: Quality of Experience

QoS: Quality of Service

REST: Representational State Transfer

RSVP-TE: Resource Reservation Protocol – Traffic Engineering

SBI: South – Bound Interface

SDN: Software Defined Networking

SDN-TE: Software Defined Networking – Traffic Engineering

STEM: SDN Traffic Engineering Management

TCP: Transmission Control Protocol

TE: Traffic Engineering

TED: Traffic Engineering Database

UDP: User Datagram Protocol

URL: Uniform Resource Locator

VM: Virtual Machine

XML: eXtensible Markup Language

1. Introducción

A medida que las aplicaciones que conocemos han ido evolucionando su naturaleza ha cambiado, siendo cada vez más dinámica y requiriendo unos anchos de banda mayores. Asimismo, Internet se está haciendo cada vez más presente en nuestras vidas, requiriendo los recursos de red para muchas de las tareas que realizamos a lo largo del día, lo que genera gran cantidad de tráfico.

Los numerosos protocolos y la complejidad física de la red junto con las implementaciones propietarias, que son un problema en un entorno en el que se cuenta con equipos de distintos fabricantes, dificultan en gran medida la gestión de esta. Todo ello, ha hecho que las tecnologías que se utilizaban hasta ahora se queden prácticamente obsoletas y se haya optado por la búsqueda de una nueva solución más flexible y que permita simplificar la gestión de la red por medio de equipos más sencillos e interoperables y utilizando sistemas inteligentes que permitan realizar una gestión más eficiente de los recursos disponibles en la red. Además, la administración de los equipos en las redes convencionales se realiza de manera distribuida, por lo que encontrar un método para hacer este trabajo en un entorno centralizado resultaría muy interesante.

De esta manera, la *Open Networking Foundation* (ONF) identifica las principales limitaciones que presentan las redes actuales [30]:

- **Complejidad:** En las últimas décadas, los protocolos de red han evolucionado, haciéndose más seguros y eficientes. Sin embargo, estas mejoras tienden a definirse de forma aislada, resolviendo problemas específicos y sin beneficiarse de una abstracción mediante la cual adoptar una solución global. Esto lleva a un estado de estancamiento de la red.

- **Políticas inconsistentes:** Para implementar políticas que abarquen la red completamente, los administradores de red deben configurar miles de equipos y mecanismos. La complejidad de las redes actuales hace que sea muy difícil aplicar políticas consistentes, lo que hace que las empresas se encuentren vulnerables en muchas situaciones.

- **Imposibilidad de escalar:** La red debe crecer tan rápido como lo hacen las demandas de los centros de datos. Sin embargo, la red se hace mucho más compleja con la suma de cientos de miles de equipos de red que deben ser configurados y gestionados.

- **Dependencia del fabricante:** La falta de estándares limita la capacidad de los operadores de red de adaptar la red a sus propios entornos. Este desacuerdo entre los requerimientos del mercado y las capacidades de la red ha llevado a la industria a un punto de inflexión.

Asimismo, la *Open Data Center Alliance* [29] define una serie de requerimientos críticos para las redes futuras, como son la adaptabilidad, la seguridad, el mantenimiento de las operaciones ante la introducción de nuevas características y capacidades, la capacidad de gestionar de forma conjunta la red, la escalabilidad y la automatización.

Debido a todas estas limitaciones de las redes actuales y las necesidades expuestas nace un nuevo paradigma de red conocido como "redes definidas por *software*" o "*Software Defined Networking*" (SDN), que permite esta centralización de la red por medio de un elemento externo, el controlador SDN, así como una gestión sencilla de los equipos de red en entornos *multi-vendor*.

1.1. Motivación

La situación actual de internet ha hecho que SDN se encuentre en auge y compañías punteras en el sector tecnológico hagan uso de estas técnicas para tener un mayor control sobre sus redes. Por ello, resultaba interesante realizar un estudio del funcionamiento de este tipo de redes por medio de la creación de escenarios sencillos, aprendiendo a utilizar el *software* necesario para ello. Así, se pretenden sentar las bases para futuros trabajos por medio de la creación de herramientas modulares y flexibles, que permitan añadir complejidad al entorno fácilmente.

1.2. Objetivos y metodología

El objetivo principal del proyecto es **estudiar y analizar el potencial de la definición de redes mediante software (SDN)** y, especialmente, el control centralizado que permiten realizar sobre la red. Mediante la separación del plano de control y datos se pueden **abordar soluciones de ingeniería de tráfico (*Traffic Engineering*, TE)** que faciliten una gestión eficiente de los recursos de red adaptándose de modo flexible a las demandas de servicio (calidad de servicio, balanceo de carga, …), ofreciéndose como alternativa a las tecnologías actuales, como MPLS (*Multiprotocol Label Switching*). Se verá como a pesar de que MPLS no es una tecnología para nada obsoleta, SDN consigue dar solución a algunos de los problemas que esta presenta. La visión global de la red que tienen las redes definidas por software permite mejorar aspectos como la computación de caminos o la actualización de las bases de datos necesarias para ello, ya que una mala sincronización con el estado real de la red puede desembocar en un cálculo erróneo de estos caminos. Del mismo modo, la utilización que hace MPLS de un protocolo *in-band* como RSVP-TE (*Resource Reservation Protocol – Traffic Engineering*) para la reserva de recursos, aumenta el tráfico de señalización en la red respecto a una arquitectura con lógica centralizada y señalización *out-of-band* como es SDN. Además, la utilización de

una tecnología que aporta un grado de programabilidad e interoperabilidad como el que proporciona este nuevo paradigma facilita en gran medida la gestión de los equipos y del comportamiento de las redes.

Teniendo esto en cuenta, se pretende establecer un entorno de emulación/virtualización que permita estudiar la aplicación de SDN-TE (*Software Defined Networking – Traffic Engineering*) en un escenario controlado, identificando las fortalezas y necesidades y facilitando la evaluación de mecanismos de optimización de red. Para abordar el trabajo, se han definido una serie de **objetivos parciales**, enumerados a continuación:

1) En primer lugar, se realizará un **estudio del estado del arte** relacionado con el diseño de redes definidas por software y de su aplicación a la ingeniería de tráfico, revisando posibles arquitecturas a implementar y las tecnologías que se utilizarán para ello.
2) Una vez adquiridos los conocimientos necesarios sobre esta tecnología y conociendo las distintas alternativas existentes en cuanto a las herramientas a utilizar, se **seleccionará el *software*** que hará posible la implementación de cada uno de los bloques de la arquitectura que se estudiará. El detalle de la selección se describirá posteriormente. De manera resumida, la metodología finalmente empleada requerirá de la utilización de las siguientes herramientas:
   - En primer lugar, Mininet [25] (instalado sobre una máquina virtual (VM) Ubuntu) permitirá emular redes virtuales mediante virtualización ligera de máquinas Linux. Se ha elegido este *software* debido a que está especialmente orientado al estudio de SDN y el protocolo OpenFlow [32]. Además, cuenta con una interfaz gráfica llamada Miniedit y permite la creación de topologías por medio de *script* en Python [35], lo que da una mayor flexibilidad a la hora de trabajar con ellas.
   - Se utilizará también OpenDaylight [37] (instalado sobre una VM Ubuntu Server); un controlador SDN de *software* libre externo a Mininet.
   - También se hará uso un generador de tráfico adaptable al entorno en que se está trabajando. Esta herramienta permitirá introducir distintos tipos de tráfico en la red para poder estudiar el comportamiento de esta.
   - Para poder analizar el tráfico que se está cursando a través de la red se utilizará Wireshark [38], un analizador de tráfico de *software* libre con el que se estaba familiarizado.
3) Tras verificar el correcto funcionamiento de los programas instalados, se realizarán **pruebas iniciales con topologías de red sencillas**, asegurando así que somos capaces de manejar con cierta soltura las herramientas escogidas. A medida que se vayan implementando nuevas funcionalidades se comparará el funcionamiento de los escenarios trabajando con SDN con otros escenarios de referencia sin esta tecnología. De esta manera se asegurará que los módulos implementados funcionan correctamente.
4) Por último, se realizará una **evaluación de la arquitectura implementada**, analizando el *overhead* introducido en la red debido al uso de SDN y la reacción de la red ante cambios.

Puede observarse el diagrama de Gantt que define los tiempos del proyecto en la Figura 1-1.

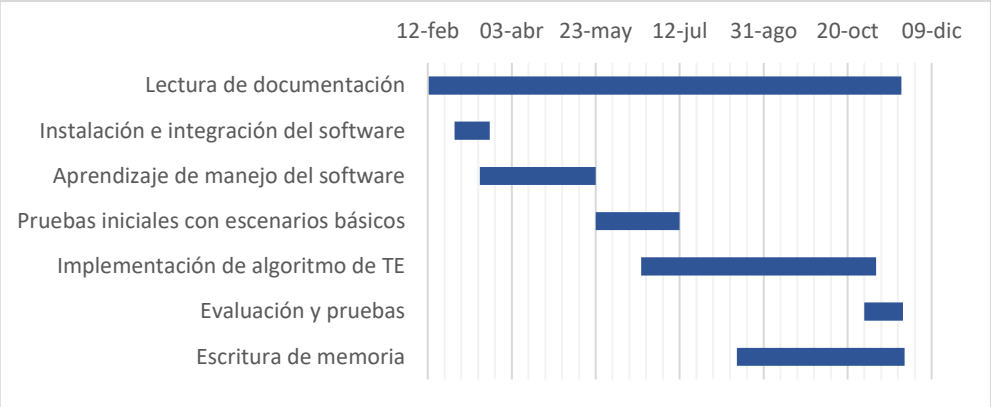


Figura 1-1. Diagrama de Gantt del proyecto

1.3. Estructura de la memoria

Tras esta introducción, en la que se ha expuesto la motivación que ha llevado a la realización de este proyecto, los objetivos propuestos y la metodología adoptada para llevarlos a cabo, se va a presentar la estructura que sigue este documento.

En el segundo capítulo se presenta el estado del arte de las distintas tecnologías con las que se ha trabajado durante la realización de este proyecto, particularmente de las redes definidas por software (SDN) y su aplicación a la ingeniería de tráfico (TE). Además, se presenta una posible solución para realizar ingeniería de tráfico de modo centralizado mediante el soporte de SDN. Una vez explicado su funcionamiento, en el capítulo tres se explica cómo se han utilizado estos paradigmas para diseñar la arquitectura SDN-TE (*Software Defined Networking – Traffic Engineering*) objeto de evaluación en el proyecto. En este sentido se explicarán los requisitos propuestos para cada uno de los bloques del sistema, así como la implementación de cada uno de ellos, diferenciando las capas a las que pertenecen (aplicación, plano de control o plano de datos). De acuerdo con dichos requisitos se expondrán los motivos por los que se ha elegido el *software* con el que se ha trabajado.

En el cuarto capítulo se validará el sistema propuesto analizando varias configuraciones de topologías y tráfico generado. De esta manera se podrá realizar un análisis del modo en que la red reacciona ante diferentes peticiones de recursos y cómo consigue atenderlas ubicando los flujos de tráfico en la red garantizando sus requisitos en la medida de lo posible. Además, se someterá a la red a un test en el que se podrá observar cómo reacciona ante la detección de anomalías como caídas de enlaces o nodos. También se realizará un análisis del *overhead* introducido en la red debido a la utilización de SDN.

Por último, en el quinto capítulo se exponen una serie de conclusiones a las que ha dado lugar el desarrollo de este trabajo, pudiendo comprender de forma algo más resumida la aportación de este tipo de redes al campo del que se ocupa este proyecto. Del mismo modo, se describe un apartado de líneas futuras en el que se proponen nuevas pruebas a las que someter al escenario estudiado y variantes del escenario implementado, para poder así comprender de manera más extensa hasta dónde pueden llegar las redes definidas por *software*.

2. Estado del arte

En este capítulo se presenta el estado actual de las tecnologías que se han utilizado durante el desarrollo del trabajo. En primer lugar, se explica qué son las redes definidas por software y cómo funcionan. A continuación, se realiza un breve acercamiento a la ingeniería de tráfico y a la contribución de SDN en este ámbito. Por último, se da a conocer la arquitectura de red sobre la que se apoya este proyecto, una arquitectura basada en un PCE (*Path Computation Element*) [3] [4], encargado de calcular los caminos que debe seguir el tráfico dentro de la red y de proporcionar los recursos que le son solicitados, y un módulo orquestador [3],que se encargará de solicitar la asignación de recursos al PCE y de comunicarse con el controlador SDN para reservarlos.

2.1. SDN y el protocolo OpenFlow

En las redes tradicionales se pueden encontrar distintos elementos como *switch*, *router* o *firewall*, cada uno con sus tareas específicas. Todas estas tareas están separadas en distintos planos: plano de control, plano de datos y plano de gestión.

El plano de control es el encargado de intercambiar la información de encaminamiento. De este modo, funciones como la construcción de las tablas ARP (*Address Resolution Protocol*) o la ejecución de protocolos de encaminamiento como OSPF (*Open Shortest Path First*) se encuentran dentro de este plano. El plano de datos se encarga de realizar el reenvío de tráfico haciendo uso de las tablas de encaminamiento de los equipos. Por último, el plano de gestión es el encargado de hacer posible la gestión de los equipos, proporcionando una interfaz entre estos y los gestores de red.

Estos tres planos se encuentran habitualmente acoplados en cada uno de los elementos de red (por ejemplo, un *router* OSPF intercambia señalización con el resto de *router* para confeccionar dinámicamente la tabla de rutas que se utiliza posteriormente en el plano de datos). Por el contrario, en SDN, se separan los planos de control y de datos, encontrándose totalmente desacoplados, lo que quiere decir que la lógica de decisión y el reenvío de paquetes no se realiza en la misma entidad, como ocurría en las redes tradicionales

De esta forma, según la *Open Networking Foundation* [31], se puede decir que la arquitectura de SDN queda dividida en tres capas

lógicas diferenciadas (Figura 2-1): la capa de aplicación, la capa del plano de control y la capa del plano de datos.

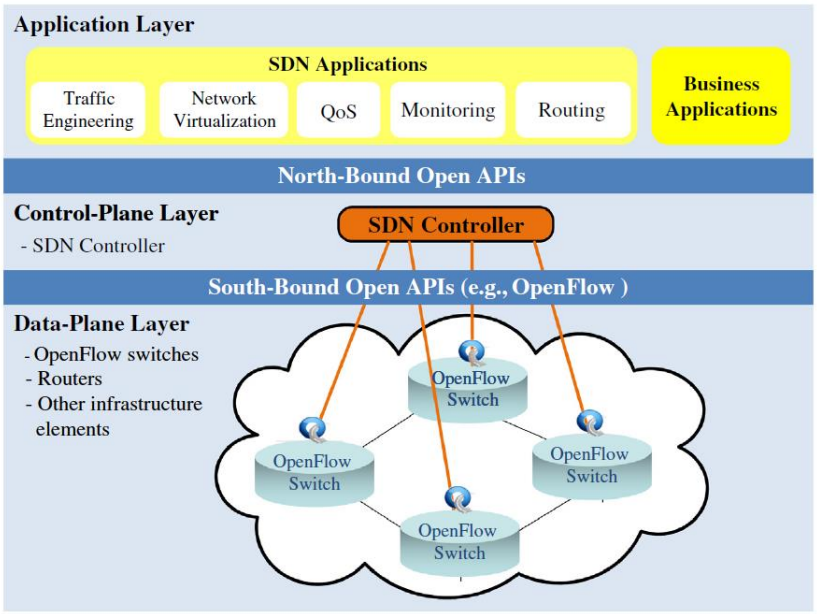


Figura 2-1. Arquitectura SDN [19]

La capa superior o capa de aplicación hace referencia a los programas que hacen uso de la funcionalidad de SDN para configurar la red y para obtener información de esta, de modo que el administrador de red comunica al controlador los requisitos de acuerdo con los cuales realizar dicha configuración y determinar así el comportamiento de la red. Dicha comunicación se realiza a través de interfaces denominadas *North-Bound Interface* (NBI), obteniendo así una visión abstracta de la red. Estas interfaces permiten automatizar la administración de la red por medio de *script* o a través de una GUI (*Graphical User Interface*) gracias a las *North-Bound* API (Figura 2-2). Los *script* de configuración pueden estar escritos en lenguajes como Java o Python y típicamente se utilizan REST API (*Representational State Transfer*) como solución para las *North-Bound* API. La REST API se encarga de enviar y recibir mensajes HTTP (*Hypertext Transfer Protocol*) para posibilitar la comunicación entre las aplicaciones y el controlador SDN. De esta manera, se puede solicitar información de la red al controlador o escribir en su base de datos a través de esta interfaz. Los dos lenguajes más utilizados para formar el cuerpo de estos mensajes son JSON (*JavaScript Object Notation*) y XML (*eXtensible Markup Language*).

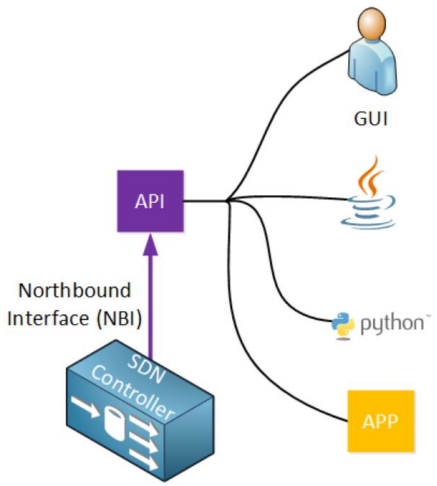


Figura 2-2. North-Bound API del controlador SDN [21]

La siguiente capa, el plano de control, la constituye la entidad denominada controlador SDN, que puede ser un equipo físico o encontrarse en una máquina virtual. El controlador se comunica mediante una *South-Bound Interface* (SBI) con los elementos de la red compatibles, obteniendo así información sobre estos o pudiendo modificar su comportamiento. Este elemento es el encargado de trasladar los requerimientos exigidos por la capa de aplicación a los dispositivos del plano inferior a través de las *South-Bound* API; además, proporciona a la capa superior esa visión abstracta de la red. OpenFlow y Cisco OpFlex, entre otras, son algunas de las South-Bound API más conocidas, pero debido a su generalización y a ser el protocolo mayoritario en las redes SDN, se utiliza OpenFlow en el desarrollo de este proyecto.

Por último, el plano de datos está formado por todos aquellos elementos de la red sobre los que el controlador SDN instalará los flujos pertinentes (*router*, *switch*, etc.) y de los que se extraerá la información necesaria para conocer el estado de la red en todo momento y poder actuar en consecuencia. Notar que, a diferencia de lo que ocurre en las redes convencionales, donde *router* y *switch* concentran las tareas relativas a los planos de datos y de control, en este caso solo se encargan de la funcionalidad de reenvío propia del plano de datos.

En conclusión, según la ONF [30], SDN junto con el protocolo OpenFlow proporciona un control centralizado en entornos *multi-vendor*, eliminando los protocolos específicos de cada fabricante y permitiendo controlar cualquier dispositivo compatible con el protocolo OpenFlow. La complejidad de las redes también se ve reducida gracias a la automatización de estas que SDN soporta, permitiendo realizar una orquestación inteligente de la red e incrementando su agilidad y escalabilidad. Por otro lado, el hecho de ser redes programables acelera el proceso de innovación, permitiendo a los gestores reprogramar en tiempo real el comportamiento de la red para que se ajuste a las necesidades que se tienen en cada momento, algo totalmente impensable en las redes tradicionales. La fiabilidad y la seguridad de la red también se ven reforzadas debido al hecho de que trabajar con SDN y OpenFlow elimina la necesidad de configurar los equipos de red individualmente, lo que da lugar a fallos debidos a políticas inconsistentes. Además, gracias al protocolo OpenFlow se puede realizar un control de la red más

granular y el hecho de ser una red centralizada mejora la experiencia de usuario al disponerse de información en tiempo real del estado de la red y poder adaptarse mejor a las necesidades de los usuarios.

No obstante, SDN también cuenta con algunos inconvenientes [16]. A pesar de las ventajas que aporta disponer de una red con el plano de control centralizado en el controlador SDN, este también es uno de los puntos débiles de esta tecnología. Todo entorno centralizado corre el peligro de que su gestor central caiga en algún momento, lo que sería fatal para el funcionamiento de la red. Por ello es importante tener en cuenta que se trata de una arquitectura muy sensible a ataques directos a este gestor, lo que hace que en ocasiones se introduzca redundancia instalando más de un controlador. También debe tenerse en cuenta la necesidad de cambios en las infraestructuras de red ya existentes, que deben ser actualizadas con equipos compatibles con los protocolos típicos de SDN. Asimismo, no debe olvidarse que se trata de una tecnología aún muy joven que se encuentra en proceso de mejora. Puede observarse un ejemplo de las diferencias de encaminamiento en las redes clásicas y en SDN en Figura 2-3, mostrándose el plano de control en rojo y el plano de datos en azul.

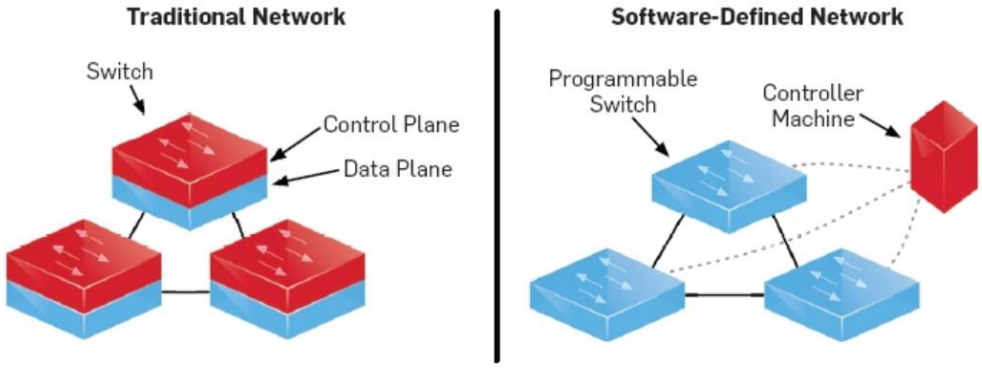


Figura 2-3. Encaminamiento redes tradicionales vs SDN [13]

2.2. Ingeniería de tráfico

Uno de los campos que se ha beneficiado de la aparición de este nuevo paradigma es la ingeniería de tráfico [9] [10]. La ingeniería de tráfico o *traffic engineering* (TE), se encarga de la evaluación y optimización del rendimiento de las redes IP (Internet Protocol) operativas. Engloba la aplicación de la tecnología y los principios científicos a la medida, caracterización, modelado y control del tráfico de Internet. Este paradigma ha sido muy utilizado con distintas tecnologías como ATM (*Asynchronous Transfer Mode*) o MPLS [15], pero como indican los autores de [19], la manera en que funcionaba no acaba de ser del todo adecuada para trabajar con las redes actuales. Hoy en día, se necesita una arquitectura de red que sea capaz de reaccionar en tiempo real a cambios en la red, así como adaptarse a distintos tipos de tráfico y gestionarlo de forma adecuada. Además, para poder hacer frente al rápido crecimiento del *cloud computing* y la gran cantidad de datos que se manejan son necesarias nuevas herramientas de TE más sofisticadas que

permitan mejorar la utilización de los recursos disponibles. De esta manera, la ingeniería de tráfico trata de minimizar parámetros como la congestión, el retardo extremo a extremo, la pérdida de paquetes y el consumo de energía, así como maximizar la QoE (*Quality of Experienc*e) o la experiencia percibida por el usuario final y la QoS (*Quality of Service*) o calidad de servicio.

Las cabeceras de los paquetes que se cursan en las redes IP convencionales contienen mucha más información de la necesaria para poder reenviar el paquete. Asimismo, la elección del próximo salto en la red puede verse como la composición de dos tareas. Una primera clasificación de los distintos paquetes con características similares en FECs (Forwarding Equivalence Classes) y la selección del próximo salto para cada uno de estos grupos de paquetes contenidos en una FEC. De esta manera, todos los paquetes que pertenezcan a una misma FEC viajarán por el mismo camino dentro de la red, permitiendo así realizar una clasificación del tráfico. En las redes convencionales, cada *router* reexamina el paquete comprobando la dirección destino en su tabla de encaminamiento y lo asigna a una FEC, lo que aumenta el tiempo de computación en cada salto. Sin embargo, en MPLS la asignación de un paquete a una FEC determinada se hace solo cuando el paquete se inserta en la red. Esto se consigue por medio de la adición de una etiqueta o *label*, que utilizarán los *router* compatibles con esta tecnología (denominados LSR (*Label Switching Router*)) para encaminar el tráfico en lugar de la dirección IP. El camino seguido por los paquetes a través de la red, determinado por las etiquetas asignadas a cada uno ellos, se denomina *Label Switched Path* (LSP). Por todo esto, MPLS no solo permite agilizar el reenvío por los caminos habituales gracias a la conmutación por etiquetas, sino que debido a la utilización de algoritmos y protocolos de señalización determinados permite seleccionar caminos adicionales específicos en función de parámetros adicionales a la dirección IP.

Para determinar los LSP en escenarios de ingeniería de tráfico se utiliza un algoritmo de cálculo de caminos denominado CSPF (*Constraint Shortest Path First*) [27]. Se trata de un algoritmo que pretende calcular el camino más corto en número de saltos entre origen y destino mediante Dijkstra [7], pero teniendo en cuenta unas restricciones asociadas a los requerimientos de cada tipo de tráfico. No hay un estándar que defina CSPF hasta ahora, aunque se basa en la idea de "*induced graph*", de la que se habla en [9].

El protocolo RSVP-TE (*Resource Reservation Protocol – Traffic Engineering*) [11] se encargará de realizar la reserva de recursos y de la distribución de las etiquetas que permitirán conformar todos los LSP. Es utilizado por los *router* de la red para intercambiarse esta información. Todos estos elementos ayudan a crear mecanismos que permiten aplicar ingeniería de tráfico en la red.

Teniendo en cuenta que el cálculo del camino adecuado mediante CSPF es uno de los elementos clave en las soluciones de TE propuestas, en [5], se propone una arquitectura estandarizada basada en el denominado PCE (Figura 2-4) [3] [4]. Esta arquitectura propone un elemento dedicado encargado exclusivamente de la computación de caminos dentro de la red, haciendo posible la aplicación de algoritmos complejos que requieren un uso de CPU más intensivo como CSPF. Para poder realizar estos cálculos el PCE debe tener accesible información actualizada del estado de la red. De este modo,

todos estos datos son almacenados en la TED (*Traffic Engineering Database*). La comunicación entre el PCE y un nodo de la red se realiza a través del protocolo PCEP (*Path Computation Element Protocol*) [23] y cualquier elemento que se comunique con el PCE para solicitar recursos se denomina PCC (Path Computation Client). Una vez un PCC recibe la información solicitada, el equipo informará a los demás nodos de la red a través del protocolo RSVP-TE.

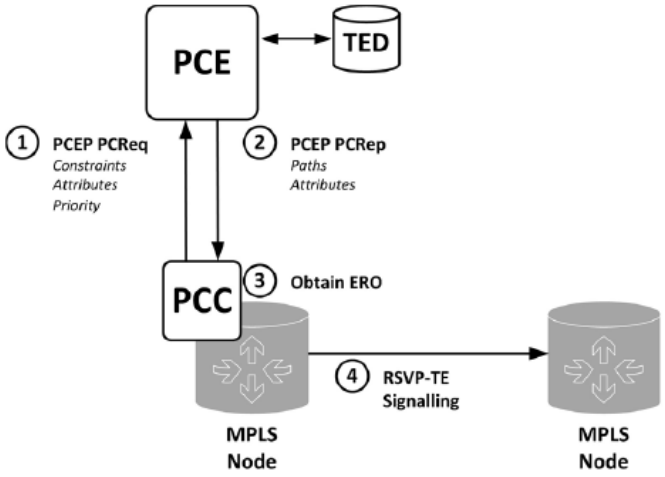


Figura 2-4. Arquitectura basada en PCE [5]

2.2.1. Contribución de SDN a la ingeniería de tráfico

Como dicen los autores de [5], la tecnología más utilizada hasta ahora en el ámbito de la ingeniería de tráfico, MPLS-TE (*Multiprotocol Label Switching – Traffic Engineering*), cuenta con algunas limitaciones que SDN es capaz de resolver. Todas estas debilidades se presentan a continuación, describiendo cómo afectan al rendimiento de la red y qué hacen las redes definidas por software para solucionarlo.

En cuanto a la minimización de la congestión, se suelen utilizar múltiples caminos entre los que dividir el tráfico por paquetes. Esto puede resultar muy negativo, puesto que puede sobrecargar al nodo destino con un trabajo excesivo a la hora de reordenar los paquetes. Este problema es aún más notable en TCP (*Transmission Control Protocol*), donde la llegada de los paquetes en un orden incorrecto puede hacer que el mecanismo de *congestion avoidance* se active, degradando el rendimiento de la red. Además, se introduce *jitter* debido al almacenamiento de esos paquetes que deben ser reordenados. La división de tráfico por flujos puede ayudar a solventar este problema, sin embargo, la granularidad con la que se divide el tráfico viene determinada por los encaminadores de la red, por lo que se pueden estar exigiendo unos niveles de granularidad diferentes a los de los elementos de los que se dispone, lo que puede desembocar en una asignación inadecuada de los recursos. SDN dispone de una alta granularidad en los elementos de *forwarding*, proponiendo un mejor tratamiento de la congestión.

Los algoritmos de cálculo de caminos son otro de los aspectos que mejora SDN. En ocasiones se experimenta latencia en la red debida al algoritmo CSPF (*Constrained Shortest Path First*) junto con el mecanismo de *autobandwidth*, mediante el que se proporciona un ajuste automático del ancho de banda reservado por cada LSP en función de sus requerimientos. Esto desemboca en un cambio continuo de los *path*. No ocurre lo mismo en las redes definidas por software por el hecho de poder realizar esta computación de los caminos de forma centralizada.

Las bases de datos no siempre reflejan el estado de la red en tiempo real. Esta limitación se ve sobre todo en redes MPLS-TE. La arquitectura basada en PCE puede ofrecer algo de mejora en este aspecto al contar con una base de datos (TED) en la que se recoge la información de estado de la red. Sin embargo, esta base de datos no siempre está sincronizada con el estado real de la red, como se ve reflejado en [4] , lo que desemboca en un cálculo incorrecto de los caminos. Como en el punto anterior, la lógica centralizada de SDN y el hecho de tener una visión general de la red son los aspectos que ayudan a solventar este problema correctamente.

La dependencia del protocolo RSVP-TE que tienen tanto MPLS-TE como la arquitectura basada en PCE es otra de las limitaciones importantes en la ingeniería de tráfico actual. La distribución de los *path* de la red mediante un protocolo con señalización *in-band* como RSVP-TE puede demorarse más de lo que lo haría en una arquitectura de red con lógica centralizada y señalización *out-of-band*, además, puede suponer problemas de estabilidad y escalabilidad en la red. Este problema queda resuelto en SDN debido a la alta programabilidad de la red y a la señalización *out-of band*.

Por todo lo mencionado anteriormente, la inclusión de SDN en el campo de la ingeniería de tráfico parece el camino natural a seguir, combinando las ventajas que ofrece con las de las tecnologías utilizadas hasta ahora. Como se comenta en el apartado 2.3, comienza a hablarse de arquitecturas basadas en PCE que cuentan además con controladores SDN. De esta manera, se combina la capacidad computacional de un elemento como el PCE con la flexibilidad que aportan las redes definidas por software.

2.3. Arquitectura basada en PCE mediante SDN

Como se ha visto en apartados anteriores, SDN trata de separar el plano de control del plano de datos, pudiendo programar así el comportamiento de la red a través de una entidad central denominada controlador. De este modo, como puede verse en [3], esta arquitectura requiere de un elemento que sea capaz de realizar la asignación de recursos propia de la ingeniería de tráfico y de determinar cómo deben ser programados los dispositivos de *forwarding* de la red. Esta función es la desempeñada por el PCE, obteniéndose así la arquitectura de red presentada en Figura 2-5.

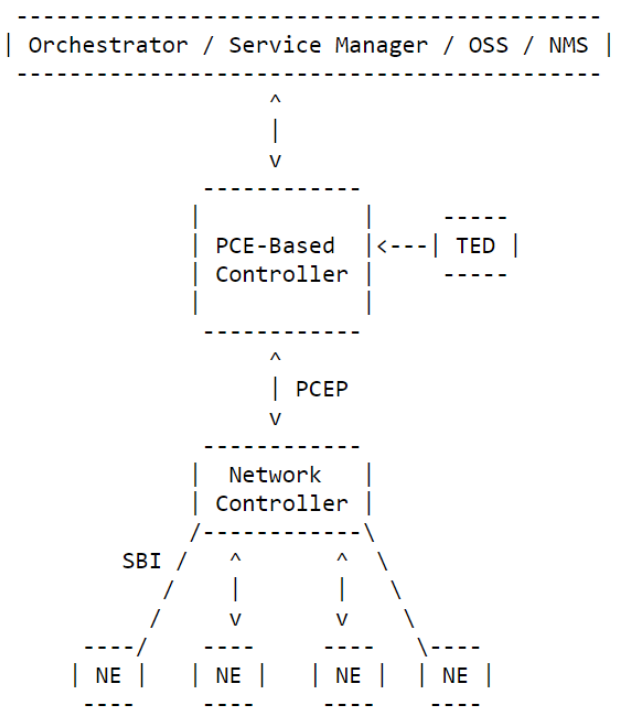


Figura 2-5. Arquitectura basada en PCE mediante SDN [3]

El elemento descrito en la Figura 2-5 como "*Orchestrator*" solicitaría al PCE unos recursos que, tras ser calculados haciendo uso de la información recogida en la TED, serían instalados en los equipos de red a través del controlador. Como puede observarse, la comunicación entre PCE y controlador se realiza mediante el protocolo PCEP, nombrado anteriormente. En [12] se define un entorno basado en PCE denominado ABNO (*Application-Based Network Operations*), que ya propone la utilización de distintos elementos que permitan coordinar el control de la red de una forma eficiente. Esta arquitectura puede llevarse a cabo con diversas tecnologías, pero los autores de [1] muestran una solución SDN para redes ópticas basada en esta filosofía. Así, sabiendo la posibilidad de introducir SDN en este tipo de arquitectura, se busca una implementación basada en el caso concreto de la utilización de SDN y de su aplicación a ingeniería de tráfico.

En [8], los autores presentan una solución basada en la arquitectura descrita, en la que el control central se divide funcionalmente en tres módulos: SDN T*raffic Engineering Module* (STEM), PCE y controlador, como se observa en la Figura 2-6.

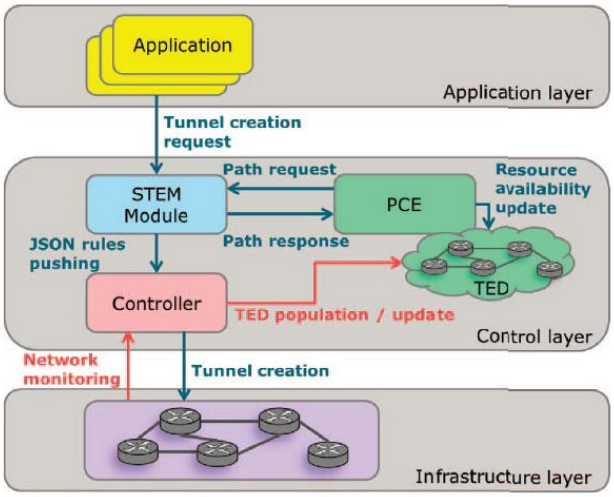


Figura 2-6. Arquitectura basada en STEM [8]

En dicha propuesta, de acuerdo a las aplicaciones que requieren establecer flujos de datos en la red, la capa de aplicación SDN (Figura 2-1) indica los requisitos mediante los cuales debe ser configurada la red, como pueden ser algoritmos de cálculo de caminos a utilizar, necesidades de ancho de banda o retardo, etc. Estas peticiones, especificadas en la figura como "*Tunnel creation request*", son enviadas al módulo STEM, en el plano de control.

El STEM se encarga de recibir esas peticiones de la capa de aplicación y "traducirlas" en las necesidades de cómputo de caminos para transmitírselas al PCE, que le responderá con los flujos a instalar en los equipos de la red. Una vez recibidos, el STEM se comunica con el controlador, que es el encargado de "hablar" con los equipos de red (*switch* o *router*) para realizar cualquier acción sobre estos (e.g., instalar o eliminar flujos).

Por su parte, el PCE recibe "*Path computation request*" de un PCC (*Path Computation Client*), en este caso el STEM, a través del protocolo PCEP. Para poder calcular los caminos en función de las variables expuestas por el PCC, el PCE debe tener un conocimiento de la red en tiempo real. Para ello, se comunica con el controlador SDN y almacena toda la información obtenida en la TED. Si bien los autores de [8] proponen una separación tanto funcional como modular del PCE y el controlador, dichas entidades podrían estar integradas.

La utilización de esta arquitectura puede resultar interesante debido a su modularidad, consecuencia de ajustarse al planteamiento de la arquitectura PCE, lo que facilita la modificación de los distintos módulos (utilización de controladores y protocolos de comunicación diferentes) de la arquitectura y la inclusión de nuevos elementos.

3. Diseño e implementación de la arquitectura SDN-TE

Una vez realizada una pequeña aproximación a las tecnologías en las que se basa este trabajo, y vistas las ventajas y el potencial de adoptar soluciones de ingeniería de tráfico basadas en SDN, concretamente mediante la arquitectura PCE, se describe a continuación la arquitectura finalmente implementada para su posterior estudio.

El escenario que se muestra a continuación se corresponde con una arquitectura basada en PCE, STEM y Controlador, siguiendo el mismo planteamiento que los autores de [8], descrito previamente. De este modo, se pretende programar de una manera modular las distintas funciones clave de la arquitectura, facilitando la flexibilidad de programación, la depuración y la futura extensión del sistema. A lo largo de este capítulo se describe la implementación de todos estos bloques. Para cada uno de ellos se seguirá la misma dinámica. En primer lugar, se describirá su funcionalidad brevemente para recordar al lector la finalidad de ese elemento concreto dentro de la arquitectura. A continuación, se presentará la herramienta por la que se ha optado para la implementación del bloque y la solución adoptada para ello. Finalmente, se realizará una breve valoración del resultado obtenido. El orden escogido para la descripción de cada módulo será ascendente, comenzando por el plano de datos, para proseguir con el plano de control y terminar con la capa de aplicación. La interpretación propia de la arquitectura puede observarse en la Figura 3-1.

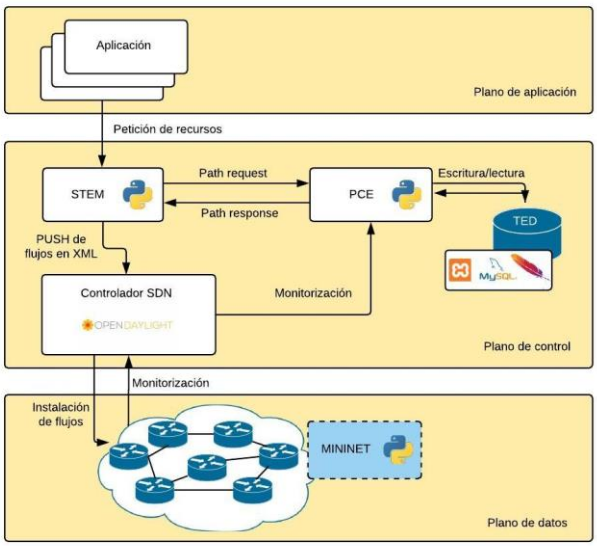


Figura 3-1. Arquitectura propia basada en STEM

3.1. Plano de datos

Dentro de este plano se encuentra la topología con la que se va a trabajar. Debe recordarse que se trata de una red cuya configuración se realizará de forma centralizada, por lo que debe contar con aquellos elementos que hagan posible su comunicación con la entidad central de SDN, el controlador. Una red acomodable a este nuevo paradigma necesita contar con equipos encaminadores compatibles con los protocolos de comunicación característicos del mismo, concretamente las denominadas interfaces *south-bound* descritas en el apartado 2.1. Tal y como se detalló en dicho apartado, el protocolo mayoritario, y por tanto prácticamente el estándar de facto, es OpenFlow. De este modo, para evaluar el diseño, teniendo en cuenta que el análisis se basará en simulación/emulación, se ha optado por utilizar un software de simulación que permita la creación sencilla de topologías que cuenten con estos elementos compatibles (específicamente *switch* gestionados mediante OpenFlow), además de permitir añadir a esta un controlador SDN externo. Este último punto se ha considerado relevante para realizar la distribución modular de funciones descrita previamente, permitiendo además que el elemento controlador se fácilmente intercambiable en futuros diseños.

Mininet, es un software libre especialmente orientado a este tipo de redes y permite emular distintas topologías mediante la virtualización ligera de máquinas Linux. Se pueden crear distintos escenarios de forma muy sencilla bien por línea de comandos, a través de Miniedit, su interfaz gráfica, o mediante la creación de *script* en Python, lo que da una mayor flexibilidad. Esta última opción ha sido la utilizada durante el desarrollo de este trabajo, concretamente mediante la versión 3.6.5 de Python, lo que permite, entre otras cosas, utilizar dicho lenguaje de programación en todos los módulos implementados, como se describe a lo largo de los apartados posteriores. En el ANEXO A se detallan los procedimientos de instalación y configuración necesarios para trabajar con la herramienta. A continuación, se muestra parte de un *script* de ejemplo (Figura 3-2) utilizado para configurar una topología, y posteriormente se detallan las funciones utilizadas en el mismo.

```
net = Mininet(topo=None, build=False, ipBase='10.0.0.0/8', autoStaticArp=True)

info('***Agregando controlador ODL\n')
OpenDayLight = net.addController(name='OpenDayLight', controller=RemoteController,
               ip='192.168.4.4', protocol='tcp', port=6633)

info('***Agregando switches\n')
s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
s5 = net.addSwitch('s5', cls=OVSKernelSwitch)

info('***Agregando hosts\n')
h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
h5 = net.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)

info('***Agregando enlaces\n')
net.addLink(s1,s2)
net.addLink(s1,s5)
net.addLink(s2,s5)
net.addLink(s2,s3)
net.addLink(s3,s4)
net.addLink(s4,s5)
net.addLink(h1,s1)
net.addLink(h2,s2)
net.addLink(h3,s3)
net.addLink(h4,s4)
net.addLink(h5,s5)
```

Figura 3-2. Configuración de topología de Mininet en Python

Se describen ahora los distintos elementos que permite añadir Mininet a una topología. Se pueden agregar todos estos dispositivos mediante comandos específicos que proporcionan las librerías de Mininet para Python. Además, todas estas funciones disponen de parámetros mediante los cuales se pueden especificar características como la clase de *switch* con la que se quiere trabajar, direcciones IP de los equipos, capacidad de los enlaces, etc. A continuación, se describe de manera más detallada el funcionamiento de los distintos elementos de los que se dispone.

La herramienta `Host` permite añadir a la topología distintos *host* que actuarán como los equipos finales en una comunicación. Esto se consigue mediante la función `addHost()` de la librería de Mininet, que devuelve el *host* añadido. Además, permite configurar parámetros tales como el nombre del mismo, su dirección IP y la de su router por defecto.

La herramienta `Switch` se corresponde con *switch* OpenFlow, que deben ir conectados al controlador. Son los que se utilizan en los escenarios sobre los que se ha trabajado. En este caso, la función que permite añadir *switch* Openflow a la topología es `addSwitch()`, que devuelve el *switch* añadido. Al igual que ocurre con los *host*, se dispone de distintos parámetros de entrada que permiten configurar aspectos como el nombre del *switch* o la clase de este. Durante el desarrollo de este trabajo se han utilizado *switch* OVS (Open vSwitch) [33], instalados en Mininet por defecto. Para poder trabajar con ellos, se debe dar el valor `OVSKernelSwitch` al parámetro `cls` de la función `addSwitch()` en el momento de su utilización para la creación de un *switch*. Mininet permite tanto modificar la versión de OVS a utilizar (siendo la 2.02 la preinstalada) como incorporar otro tipo de *switch* (como OfSoftSwitch o CPqD [17]. Teniendo esto en cuenta se valoró la elección más adecuada de *switch* para evaluar la propuesta. Dado que el

diseño de la arquitectura pretende aportar soluciones de ingeniería de tráfico, y que estas realizan en muchas ocasiones la configuración de los caminos mediante etiquetas MPLS, se consideró necesario el soporte a MPLS ofrecido por una versión más moderna de OVS que el kernel de trabajo (Ubuntu 14.04.4) permitía instalar, la 2.5.0. Respecto a los otros modelos de *switch* que, a priori, pueden dotar al escenario de algo más de flexibilidad, se desecharon debido a que se experimentaba una pérdida de rendimiento en la simulación. Por otra parte, a la hora de configurar la comunicación con el controlador SDN, hay que destacar que Mininet incorpora su propio controlador, pero permite la comunicación con un controlador externo a la topología, caso contemplado en este proyecto. Cuando el *switch* sea lanzado deberá especificarse el identificador del controlador con el que se quiere que se comunique (Figura 3-3).

```
info('***Iniciando switches\n')
net.get('s1').start([OpenDayLight])
net.get('s2').start([OpenDayLight])
net.get('s3').start([OpenDayLight])
net.get('s4').start([OpenDayLight])
net.get('s5').start([OpenDayLight])
```

Figura 3-3. Asociación de controlador SDN a un switch

La herramienta `NetLink` permite crear enlaces entre los distintos nodos dispuestos en el área de trabajo. Para ello se utiliza la función `addLink()`, pasándole como parámetros los nombres de los elementos que se conectan en los extremos del enlace. Además, esta función admite la introducción de más parámetros opcionales que permiten configurar propiedades del enlace tales como retardo, porcentaje de pérdidas, etc.

La herramienta `Controller` permite añadir un controlador al escenario mediante la función `addController()`. Se pueden introducir parámetros como el nombre del controlador, el tipo (`Remote Controller` en el desarrollo de este TFG), la IP en la que se encuentra, el protocolo de nivel de transporte con el que funciona y el puerto en el que está escuchando.

Mininet también permite simular topologías clásicas de red no basadas en SDN haciendo uso de *switch* y *router* no configurables. Estas herramientas no han sido utilizadas durante el desarrollo del proyecto.

3.1.1. Protocolo OpenFlow y OVS

Como se ha comentado anteriormente, se ha utilizado el protocolo OpenFlow para comunicar el controlador con los equipos de red. Dicha comunicación es la que permite tanto monitorizar el estado de la red, como establecer el plano de datos en los elementos de la red. Dicho plano de datos es, en definitiva, la tabla de conmutación del *switch* virtual (OVS) que realizará los reenvíos de tráfico en función de las reglas o flujos establecidos. La Figura 3-4 muestra un ejemplo de dicha tabla:

flow_id	cookie	duration	table	n_packets	n_bytes	priority	matches	actions
1	0x1	44.397s	0	419	59203	10	ip,nw_dst= 10.0.0.4	push_mpls,set_field: 10
3	0x2	24.784s	0	2372	232313	100	in_port=2	output:3
…	…	…	…	…	…	…	…	…

Figura 3-4. Tabla de conmutación de un switch OVS

Así, se define un flujo en un *switch* OVS como un conjunto de campos que definen las acciones a realizar sobre un paquete entrante a ese *switch* con unas características determinadas. Los campos presentados en la Figura 3-4 son variables, dependiendo de la cantidad de información proporcionada en el momento de instalar el flujo. En este caso se presenta únicamente la información básica que caracteriza a un flujo.

El campo `flow_id` es un valor numérico que permite identificar a un flujo de forma unívoca dentro de una tabla. Por otro lado, el campo `cookie` ayuda a identificar un conjunto de flujos dentro de una tabla, por lo que no tiene por qué ser un valor único. Los campos `duration`, `n_packets` y `n_bytes` aportan información acerca del flujo instalado como el tiempo que lleva activo o el tráfico que se ha cursado relativo a ese flujo en número de paquetes o bytes. La columna `table` consta de un identificador numérico que indica la tabla a la que pertenece un flujo. De este modo, los flujos pueden agruparse en distintas tablas que pueden relacionarse entre sí. Los dos campos que definen el comportamiento del *switch* dado un paquete entrante son `matches` y `actions`. En el primero de ellos se define la información de las características que tiene que tener un paquete para aplicar sobre él las acciones relativas a ese flujo, definidas en `actions`.

Dentro del protocolo OpenFlow se pueden distinguir tres grandes grupos de mensajes:

1) El controlador envía mensajes al *switch* para poder extraer estadísticas de estos y conocer su estado, lo que se hace periódicamente, así como para modificar las entradas de las tablas de flujos de los *switch*, tanto añadiendo como eliminando entradas. Dentro de estos mensajes, el controlador envía paquetes LLDP (*Link Layer Discovery Protocol*) a todos los *switch* con los que tiene conexión, que se propagarán por la red y serán reenviados al controlador. De esta manera, el controlador podrá conocer la disposición de los nodos dentro de la red y cómo están interconectados.
2) Los *switch* pueden comunicarse con el controlador por varios motivos. En caso de que llegue a un *switch* un paquete sin coincidencias en su tabla de flujos, el *switch* envía un `Packet-in` al controlador preguntando qué hacer con ese paquete. Estos mensajes también pueden enviarse si la acción correspondiente a un *match* específico en la tabla de flujos corresponde con el reenvío del paquete hacia el controlador. Un *switch* también se comunica con el controlador cuando recibe un flujo y es instalado en su tabla de flujos correctamente. Esto se hace mediante un paquete `Flow-modify`.

3) Por último, se habla de mensajes simétricos, enviados tanto por los *switch* como por el controlador. Su principal funcionalidad es la negociación de la conexión y su mantenimiento.

3.2. Plano de control

Este plano recoge todos aquellos elementos que, combinando sus funcionalidades, concentran toda la inteligencia de la red. Como se ha indicado previamente, se ha seguido un modelo de arquitectura análogo al descrito en [8], de modo que se ha dividido la funcionalidad en tres módulos: STEM, PCE y Controlador. Los dos primeros se han desarrollado íntegramente mediante programación en Python, mientras que el último es el controlador de código abierto OpenDaylight.

La elección de Python, tratándose de un lenguaje dinámicamente tipado y que soporta orientación a objetos, ha facilitado en gran medida la implementación de algoritmos de cálculo de caminos como Dijkstra y CSPF. Además, según StackOverflow [36], la página web principal a nivel mundial de preguntas y respuestas sobre programación, y el IEEE [20], se trata del lenguaje más utilizado y que más crecimiento está experimentando en el momento de escritura de este TFG.

3.2.1. STEM

El módulo STEM es el encargado de recibir las peticiones procedentes de la capa de aplicación y de "traducirlas" en las necesidades de cómputo de caminos que se le trasladarán al PCE. Asimismo, debe comunicar los flujos recibidos del PCE al controlador SDN para que este los instale en los equipos encaminadores. Si bien la interacción STEM-PCE podría seguir un protocolo de comunicación estandarizado, como PCEP (tal y como se describió previamente), su utilización no se ha contemplado en el desarrollo de este trabajo realizando, como primera aproximación, una comunicación basada en *socket* UDP (*User Datagram Protocol*).

Este módulo consta de dos procesos diferenciados. El primero de ellos es el hilo principal, que se encontrará a la escucha de peticiones de recursos desde la capa de aplicación. Estas serán interpretadas y reenviadas hacia el PCE, donde se calcularán los caminos de acuerdo con los requisitos expuestos y con el algoritmo seleccionado por el gestor de la red. Dichos caminos se traducirán finalmente (a través de la configuración última por parte del controlador SDN) en LSP que fijen el camino a seguir por los flujos de tráfico que los demandaron. Cuando le llegue alguna petición de recursos para poder cursar tráfico con unas restricciones que determinarán el algoritmo de cálculo de caminos a utilizar, Dijkstra o CSPF en el caso de este proyecto, y las restricciones asociadas si este lo necesita, como mínimo ancho de banda requerido en el caso de CSPF, el STEM trasladará esta petición al PCE. A continuación, se espera a la respuesta (los flujos a establecer mediante SDN en los *switch* de la red). Una vez obtenidos los flujos y tras la detección del comando `OK`, recibido del PCE y que indica el fin de envío de los flujos que

determinan un LSP determinado, el STEM procede definitivamente a su instalación en los *switch* OVS a través de la comunicación con el controlador.

Para hacer posible esta interacción STEM/controlador, OpenDaylight cuenta con una interfaz REST mediante la que se puede establecer comunicación con sus distintas API, siendo las dos más populares "topology" e "inventory" (se verá más información sobre las distintas API de ODL y la instalación de flujos a través de la GUI de ODL en el ANEXO B). Para la instalación de los flujos en un nodo en concreto, el STEM envía a la URL que identifica a ese equipo dentro de ODL un mensaje con cuerpo en XML a través de HTTP. Este mensaje debe cumplir estrictamente con la estructura requerida por el controlador, de lo contrario, no podrá interpretarlo e instalarlo en el nodo correspondiente.

Para comprender mejor la información que se intercambian lo distintos módulos se presentan algunos mensajes de ejemplo a continuación. Como se ha indicado previamente, la metodología seguida para comunicar información entre el PCE y el STEM es de implementación propia, sin corresponderse con ningún protocolo estandarizado (como podría ser PCEP). En primer lugar, puede observarse la manera en que el STEM expone los requisitos de un tráfico determinado al PCE. Este mensaje consta de tres o cuatro campos (Figura 3-5), en función del algoritmo escogido. Los dos primeros hacen referencia al nodo origen y al nodo destino de la comunicación. El tercer campo guarda un valor que identifica el algoritmo a utilizar, que puede ser '1' (Dijkstra) o '2' (CSPF). El campo `Constraint` solo se utiliza en caso de que el algoritmo elegido sea CSPF, y en él se introducen los requisitos de ancho de banda que requiere el tráfico en *kbps*.

NodeSrc ID	NodeDst ID	Algorithm	(Constraint)

Figura 3-5. Formato de mensaje STEM -> PCE

Del mismo modo, los mensajes de respuesta que el PCE envía al STEM cuentan con algunos campos variables, aunque la información básica se puede ver en la Figura 3-6. El campo `MPLS instruction` sirve para indicar el tipo de flujo MPLS que debe instalar el STEM como puede ser `PUSH`, o adición de etiqueta, `FORWARD`, o reenvío del paquete con una etiqueta dada, y `POP`, o extracción de la etiqueta MPLS, para así poder realizar el envío de la información a equipos que trabajen con encaminamiento IP. El segundo campo del mensaje identifica la etiqueta del LSP al que pertenece un flujo concreto y `Switch ID` y `Flow ID` contienen la información relativa al ID del *switch* en el que debe instalarse el flujo y el ID de este en ese *switch*. Mediante el campo `Output Port` se indica el puerto de salida que debe asignarse en el flujo para un paquete dado, así, se reenviará el tráfico hacia el siguiente nodo que conforme el LSP. Para poder identificar fácilmente un flujo o conjuntos de flujos en la tabla de un *switch* se asigna un valor de `Cookie` a cada uno de ellos.

MPLS instruction	Label	Switch ID	Flow ID	Output Port	Cookie

Figura 3-6. Formato de mensaje PCE -> STEM

Una vez el STEM reciba un paquete con toda la información que le proporciona el PCE, se comunicará con ODL para instalar los flujos en los distintos *switch*. Recordar que esta comunicación se hace con la API REST de ODL gracias a la librería `requests` de Python, escribiendo el cuerpo de los mensajes en XML. Puede observarse un ejemplo de este mensaje en la Figura 3-7. Se puede ver en la primera línea la URL a la que se envía el flujo, especificando su ID y el del *switch* en el que se va a instalar. Deben definirse unos *header* para que ODL interprete bien el mensaje. El cuerpo del mensaje corresponde con todo lo contenido en *payload*, donde se pueden identificar los distintos campos que completará con la información recibida por el PCE. Por último, realiza un PUT HTTP para insertar el flujo en el *switch* a través del controlador.

```
url = 'http://192.168.4.4:8181/restconf/config/opendaylight-inventory\
      :nodes/node/openflow:' + switch_id + '/table/0/flow/' + str(flow)
headers = { 'content-type' : 'application/xml',
            'accept' : 'application/xml' }
payload =  '<?xml version="1.0" encoding="UTF-8" standalone="no"?>\
           <flow \
               xmlns="urn:opendaylight:flow:inventory">\
               <flow-name>FooXf10</flow-name>\
               <instructions>\
                   <instruction>\
                       <order>0</order>\
                       <apply-actions>\
                           <action>\
                               <pop-mpls-action>\
                                   <ethernet-type>2048</ethernet-type>\
                               </pop-mpls-action>\
                               <order>1</order>\
                           </action>\
                           <action>\
                               <output-action>\
                                   <output-node-connector>\
                                   openflow:' + switch_id + ':' + outputPort + '\
                                   </output-node-connector>\
                                   <max-Length>60</max-Length>\
                               </output-action>\
                               <order>2</order>\
                           </action>\
                       </apply-actions>\
                   </instruction>\
               </instructions>\
               <id>' + str(flow) + '</id>\
               <strict>false</strict>\
               <match>\
                   <ethernet-match>\
                       <ethernet-type>\
                           <type>34887</type>\
                       </ethernet-type>\
                   </ethernet-match>\
                   <protocol-match-fields>\
                       <mpls-label>' + label + '</mpls-label>\
                   </protocol-match-fields>\
               </match>\
               <idle-timeout>0</idle-timeout>\
               <cookie>' + str(cookie) + '</cookie>\
               <cookie_mask>255</cookie_mask>\
               <installHw>false</installHw>\
               <hard-timeout>0</hard-timeout>\
               <priority>10</priority>\
               <table_id>0</table_id>\
           </flow>'

r = requests.put(  url, data=payload, headers=headers, auth=('admin','admin'))
```

Figura 3-7. Ejemplo de mensaje STEM -> Controlador

El establecimiento de un LSP específico que obligue al tráfico de datos a seguir el camino deseado (ingeniería de tráfico) se ha realizado mediante etiquetado MPLS, facilidad soportada por los *switch* OVS utilizados y análogo al establecimiento que haría RSVP-TE en MPLS.

Puede verse un diagrama del funcionamiento descrito para es STEM en la Figura 3-8

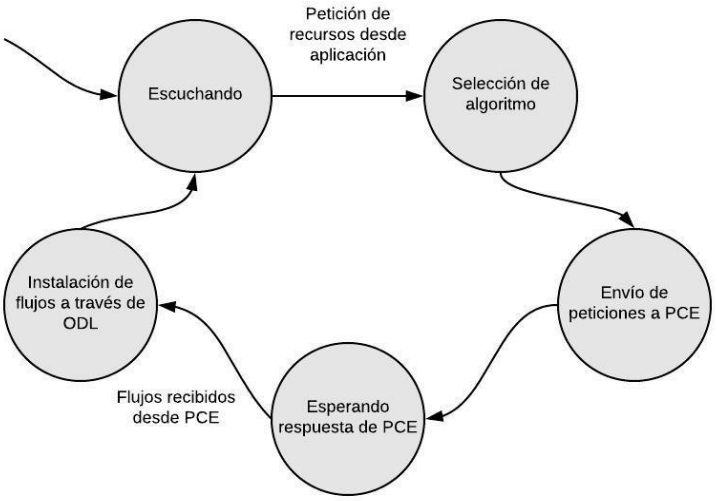


Figura 3-8. Diagrama de estados del hilo principal del STEM

La necesidad de configurar la red mediante ODL puede estar motivada no solo por la demanda específica de aplicaciones en su inicio (comunicación desde el STEM al PCE) sino como resultado de un cambio en la topología de la red (caída de nodos o enlaces, pérdida de QoS, …). Así pues, se requiere de una monitorización constante del estado de la red y la capacidad de recalcular nuevos caminos en el PCE. Para facilitar esta reconfiguración dinámica el módulo STEM incluye otro proceso o hilo (Figura 3-9). Este se encuentra continuamente a la escucha de mensajes con origen en el PCE, que será el responsable de monitorizar la red e identificar los eventos "de cambio", calcular los nuevos flujos a establecer e indicar su instalación al controlador (a través del STEM).

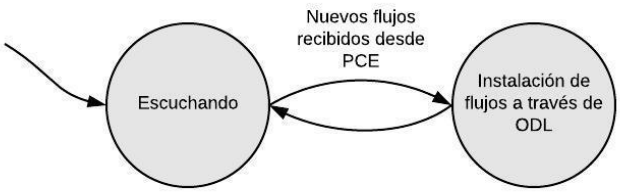


Figura 3-9. Diagrama de estados del hilo secundario del STEM

3.2.2. PCE

La función del PCE es realizar la computación de los caminos que le son expuestos en las peticiones que recibe del STEM, sirviéndose de los datos recogidos en la TED por medio de la comunicación con el controlador.

El PCE cuenta con tres procesos que se ejecutan de manera concurrente: 1) el hilo principal, responsable de las inicializaciones, la puesta en marcha de los otros dos hilos, y de la escucha permanente de peticiones del módulo STEM; 2) Un hilo de monitorización del estado de la red; 3) Un segundo hilo de monitorización del ancho de banda (relacionado con las demandas de QoS de los flujos activos en la red).

1) Hilo principal (Figura 3-10)

El hilo principal realiza, en primer lugar, la conexión con la base de datos TED (base de datos MySQL descrita a continuación), en la que almacenará y de la que extraerá información de la red. A continuación, el hilo principal crea dos procesos secundarios que se encargarán de la monitorización de la red. Hecho esto, se abre un *socket* UDP que estará escuchando continuamente peticiones procedentes del módulo STEM.

Cuando el PCE recibe datos, decodifica el origen y el destino del tráfico que se quiere insertar en la red, para poder así calcular el camino óptimo entre estos dos nodos. Además, también debe reconocer el algoritmo según el cual debe calcular esos caminos.

Para utilizar la información topológica más reciente, el PCE se comunica con el controlador SDN al recibir cada petición para actualizar la información de la red. En función de los parámetros que extrae de su base de datos y de las peticiones realizadas por el STEM, el PCE calcula los caminos que efectivamente debe seguir el tráfico para satisfacer sus requisitos. Específicamente: en primer lugar, el PCE calculará la matriz de adyacencia de la red, así como una matriz de puertos en la que quedan recogidos los puertos por los que debe salir un nodo para ir a otro nodo adyacente. Estas dos matrices varían en función de los enlaces extraídos de la red al monitorizarla a través de controlador y la utilización de Dijkstra o CSPF para el cálculo de caminos. CSPF se basa, al igual que Dijkstra, en la búsqueda del camino de mínimo coste, pero añadiendo restricciones al cómputo. Así, en caso de utilizar CSPF con requisitos de ancho de banda, si un enlace no dispone del ancho de banda suficiente para poder hacer frente a los requerimientos de un flujo no se tendrá en cuenta para el cálculo de los caminos.

Una vez obtenidas ambas matrices, se pasa a la aplicación del algoritmo, para el que se necesita saber el nodo fuente, nodo destino, algoritmo utilizado, matrices de adyacencia y puertos, *socket* mediante el cual se envían los datos al STEM (ya que la función de cálculo de caminos puede ser utilizada por el hilo principal o por el de monitorización) y dirección IP y puerto a los que debe enviarse (el hilo principal de STEM y el de monitorización escuchan en direcciones distintas). Con estos datos se calculan los dos caminos más cortos en número de saltos desde el nodo origen al nodo destino (Dijkstra, en cualquier caso) que cumplen con las restricciones dadas (si se aplica CSPF). Para ello, debe extraerse el camino completo que van a seguir los paquetes e instalar los flujos necesarios en los nodos que se vayan a atravesar. A continuación, se inserta la

información recogida en la base de datos, y se envían los flujos al STEM para ser instalados.

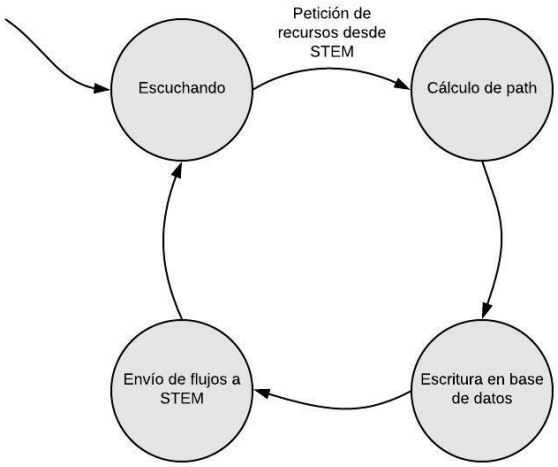


Figura 3-10. Diagrama de estados del hilo principal del PCE

2) Monitorización del estado de la red (Figura 3-12)

Como se ha dicho previamente, además del hilo principal, el PCE mantiene activos dos hilos de monitorización pendientes de las variaciones en la red.

El primero de estos dos procesos se encarga concretamente de la monitorización de cambios en la topología de red para ser capaces de detectar anomalías en esta, como pueden ser caídas de enlaces o nodos. Para ello, el PCE se comunica periódicamente con el controlador a través de su API REST, obteniendo así la información del estado de la red más reciente. Esta comunicación es posible gracias a la librería `requests` de Python, que permite realizar peticiones HTTP (Figura 3-11). De esta forma, será posible comunicarse con la API de ODL que se desee especificando la acción a realizar (PUT, GET, DELETE) y la URL de la API con la que se quiere realizar esa acción.

```
topology = 'http://192.168.4.4:8181/restconf/operational/network-topology:network-topology/'
r = requests.get(topology,auth=('admin','admin'))
```

Figura 3-11. Mensaje de monitorización del PCE

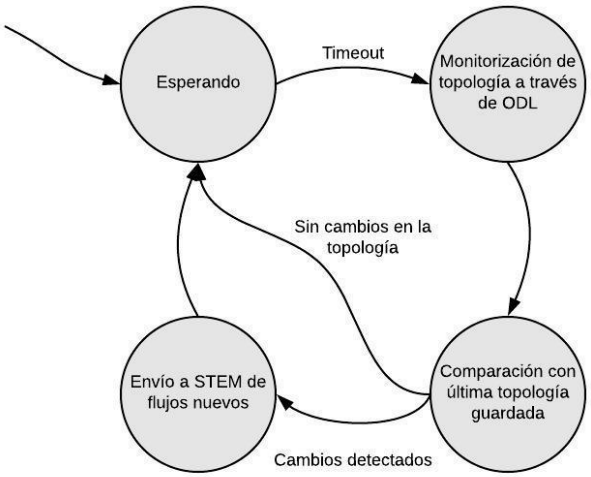


Figura 3-12. Diagrama de estados del hilo secundario de monitorización de red del PCE

3) Monitorización del ancho de banda (Figura 3-13)

El otro proceso en segundo plano monitoriza el ancho de banda de la red para evaluar las prestaciones de la red e identificar la posible necesidad de modificar los caminos previamente establecidos. En este proyecto, se ha evaluado como caso de uso un escenario de distribución equitativa del tráfico cursado (balanceo de carga). Así pues, la monitorización del ancho de banda trata de identificar específicamente el grado de utilización de los enlaces y el *throughput* de las aplicaciones cursadas. Por un lado, se almacena el estado de ocupación de cada uno de los enlaces individuales en la red. Además, también se guarda el ancho de banda ocupado en cada uno de los LSP creados.

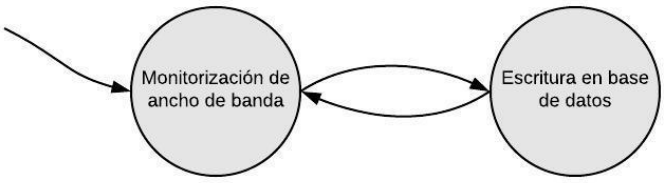


Figura 3-13. Diagrama de estados del hilo secundario de monitorización de ancho de banda del PCE

3.2.3. TED (*Traffic Engineering Database*)

La TED, tal y como se desprende de la arquitectura PCE [3] [4] descrita previamente, es una base de datos en la que el PCE recoge toda la información necesaria para estar al tanto de lo que sucede en la red (estado de los enlaces, ocupación, etc.) y así poder calcular los distintos caminos solicitados.

En este proyecto se buscaba una implementación sencilla de una base de datos, puesto que la creación de estas no es el objetivo principal del trabajo, pero se quería hacer una representación lo más fiel posible de la arquitectura PCE. Por ello se ha elegido XAMPP, un paquete de software libre que consiste principalmente en un servidor web Apache, intérpretes para lenguajes de *script* PHP y Perl y un sistema de gestión de bases de datos MySQL. Para poder hacer uso de la base datos que proporciona, únicamente debe lanzarse el servidor Apache y la base de datos MySQL. A continuación, se describe cómo se ha utilizado y qué tablas se han creado para conformar la TED.

- La tabla `topology_information` (Figura 3-14) recoge información más detallada para cada camino origen/destino que se va calculando, como número de saltos para llegar de uno nodo a otro, próximo salto, puerto por el que salir hacia el siguiente nodo, el camino completo que se va a seguir, prioridad del camino (1 o 2), etiqueta asociada a ese circuito virtual y ancho de banda experimentado por el tráfico que se cursa a través de este.

nodeSrc	nodeDest	nHops	nextHop	outPort	path	priority	label	bps
openflow:9	openflow:10	3	openflow:3	2	9 3 7 10	1	10	105648
openflow:9	openflow:10	4	openflow:2	1	9 2 1 8 10	2	11	0
openflow:2	openflow:7	2	openflow:3	2	2 3 7	1	12	0
openflow:2	openflow:7	3	openflow:1	1	2 1 8 7	2	13	0

Figura 3-14. Tabla topology_information de la TED

- La tabla `bw_n10` (Figura 3-15) recoge la capacidad de cada enlace en la topología. Esto servirá para poder calcular la ocupación de los enlaces en función del tráfico en la red.

node1 ▲ 1	node2	bps
openflow:1	openflow:2	50000
openflow:1	openflow:8	100000
openflow:10	openflow:6	100000
openflow:10	openflow:7	100000
openflow:10	openflow:8	100000
openflow:2	openflow:9	100000

Figura 3-15. Tabla bw_n10 de la TED

- La tabla `link_occupation_information` (Figura 3-16) recoge el tráfico que está circulando por cada enlace de la red, así como la ocupación de estos.

node1	node2	bps	lastTxBytes	occupation
openflow:1	openflow:2	227	1785	1
openflow:1	openflow:8	227	3549	0
openflow:2	openflow:1	227	1700	1
openflow:2	openflow:3	227	1904	0
openflow:2	openflow:9	227	3366	0
openflow:3	openflow:2	227	1530	0
openflow:3	openflow:4	227	15997	0
openflow:3	openflow:7	108461	121647	109

Figura 3-16. Tabla link_occupation_information de la TED

- La tabla `installed_flows_information` (Figura 3-17) recoge información de todos los flujos instalados en los *switch* OpenFlow. La información disponible es la siguiente: nodo en el que se instala el flujo, identificador de este, *match* (dirección IP destino o etiqueta MPLS), acción MPLS (PUSH, FORWARD, POP), puerto por el que se debe reenviar el paquete, etiqueta MPLS del paquete, prioridad del flujo, bits por segundo que se cursan de ese flujo en concreto y algoritmo mediante el cual ha sido calculado.

Node	nodeDest	flowID	matches	mplsAction	outputPort	label	priority	bps	lastTxBytes	algorithm
openflow:9	openflow:10	1	ip_dest->10.0.0.10	PUSH	2	10	1	105648	107909	1
openflow:3	-	1	mpls_label->10	FORWARD	3	10	1	108624	110957	1
openflow:7	-	1	mpls_label->10	FORWARD	4	10	1	108624	110957	1
openflow:10	-	1	mpls_label->10	POP	4	10	1	108624	110957	1
openflow:2	openflow:7	1	ip_dest->10.0.0.7	PUSH	1	13	2	0	0	1
openflow:1	-	1	mpls_label->13	FORWARD	2	13	2	0	0	1
openflow:8	-	1	mpls_label->13	FORWARD	2	13	2	0	0	1
openflow:7	-	2	mpls_label->13	POP	5	13	2	0	0	1

Figura 3-17. Tabla installed_flows_information de la TED

3.2.4. Controlador SDN

Un elemento clave del plano de control y por el que se caracterizan las redes definidas por software es el controlador SDN. Este controlador es el elemento central de la red y proporciona una visión general de esta. Además, permite una comunicación sencilla con los elementos de conmutación gracias al protocolo OpenFlow, eliminando las barreras que supone el software específico de algunos fabricantes. Gracias al desarrollo modular planteado en el trabajo, en el que la funcionalidad PCE se ha implementado de manera separada del controlador, no ha sido necesario programar funcionalidades adicionales sobre el mismo. Así, se ha podido optar por la utilización de un controlador de código abierto plenamente testado. Si bien existen múltiples controladores disponibles (FloodLight [34], OpenDaylight, ONOS (*Open Network Operating System*) [28], …), dado que en general ofrecen prestaciones similares en el entorno de pruebas para el que se ha desarrollado este trabajo, se ha elegido uno de los de mayor difusión según los autores de [6], OpenDaylight (ODL), un proyecto de código abierto llevado a cabo por *The Linux Foundation* cuyo objetivo es promover las redes SDN y la virtualización de funciones de red (NFV). ODL puede desplegarse fácilmente, en este caso en una máquina virtual Ubuntu Server en su versión 17.10.1. Para detalles de la instalación y de funcionamiento ver el ANEXO B. Una vez instalado se puede lanzar fácilmente con el siguiente comando:

```
./distribution-karaf-0.4.0-Beryllium/bin/karaf
```

La interacción de ODL con los módulos implementados se basa, como ya se ha indicado previamente, en la utilización de las correspondientes REST API que permiten indicar los flujos a establecer en los *swtich* OVS o acceder a información topológica. Así, ODL recibe indicaciones por parte del STEM y, a su vez, envía información de monitorización directamente al PCE. El detalle de dicha interacción puede observarse en los *script* previamente indicados.

3.3. Capa de aplicación

En esta capa se encuentran todas aquellas aplicaciones que, mediante las *North-Bound* API, trasladan los requisitos que necesitan para funcionar correctamente a la red. El objetivo principal de este trabajo es el estudio de la tecnología SDN y su potencial para implementar soluciones de ingeniería de tráfico. En este contexto, el esfuerzo del trabajo se ha puesto en el desarrollo de la capa de control de la red, utilizándose la capa de aplicación únicamente como fuente de tráfico para evaluarla. Así pues, en esta capa solo se ha considerado la generación de tráfico, de manera simulada, para generar demandas de recursos sobre la red. Estas demandas se comunican al módulo STEM para trasladar las peticiones al PCE y el Controlador SDN. Cuando de la interacción resultante con el PCE el STEM verifique que el tráfico entrante de una aplicación puede ser insertado en la red cumpliendo los requisitos expuestos en la petición, comunicará a la capa de aplicación que el tráfico puede ser cursado, en caso contrario, este se quedará a la espera de disponer de recursos o simplemente será rechazado.

Para poder simular esta generación aleatoria y secuencial de distintos tipos de tráfico, se ha creado un *script* en Python ubicado en la máquina virtual de Mininet que va creando flujos con distintos orígenes y destinos y de distinta índole. Para ello, se ha utilizado D-ITG (*Distributed Internet Traffic Generator*) [2] (ver ANEXO C), un generador de tráfico *open-source* muy completo creado por el DIETI (*Dipartamento di Ingegneria Elettrica e delle Tecnologie dell'Informazione*) de la Universidad Federico II de Nápoles (Italia). Esto ha sido posible gracias a las librerías de Mininet para Python que, tras crear una topología, permite ejecutar los comandos propios de Linux en todos los *host*, y por los tanto los relativos al generador de tráfico (Figura 3-18). Para más información sobre el *script* en Python ver el ANEXO A.

```
result_h10 = h10.cmd('cd /home/D-ITG-2.8.1-r1023/bin &&./ITGRecv &')
```

```
result_h9 = h9.cmd('cd /home/D-ITG-2.8.1-r1023/bin &&
              ./ITGSend -T UDP -a 10.0.0.10 -c 100 -C 100 -t 60000 -l sender.log -x receiver.log &')
```

Figura 3-18. Comandos de generación de tráfico en Mininet

3.4. Integración de los módulos para su ejecución

En este apartado se explican los pasos que se han seguido para la creación de cada uno de los escenarios de Mininet mediante *script* en Python. Además, estos *script* se utilizarán para hacer posible, de alguna manera, la comunicación entre los distintos *host* y el módulo STEM para la petición de recursos y la ejecución de distintos tráficos entre los elementos de la red para ver cómo reacciona esta ante los cambios que detecta. No se va a mostrar el *script* completo, simplemente se mostrarán las líneas relevantes, describiendo cada una de las funcionalidades que presenta y cómo ejecutarlo dentro de la máquina virtual de Mininet. Estos *script* deberán guardarse en la siguiente ruta:

```
/home/mininet/mininet/custom
```

Una vez dentro del fichero, se añaden las librerías necesarias (Figura 3-19) y se configuran todos los elementos que van a formar parte de la simulación, como son el controlador ODL, los *switch* OVS, los *host* y los enlaces que definen cómo están interconectados todos estos dispositivos (Figura 3-20).

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
import socket
import time
import os
```

Figura 3-19. Importación de librerías de Mininet para Python

```
net = Mininet(topo=None, build=False, ipBase='10.0.0.0/8', autoStaticArp=True)

info('***Agregando controlador ODL\n')
OpenDayLight = net.addController(name='OpenDayLight', controller=RemoteController,
               ip='192.168.4.4', protocol='tcp', port=6633)

info('***Agregando switches\n')
s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
s5 = net.addSwitch('s5', cls=OVSKernelSwitch)

info('***Agregando hosts\n')
h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
h5 = net.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)

info('***Agregando enlaces\n')
net.addLink(s1,s2)
net.addLink(s1,s5)
net.addLink(s2,s5)
net.addLink(s2,s3)
net.addLink(s3,s4)
net.addLink(s4,s5)
net.addLink(h1,s1)
net.addLink(h2,s2)
net.addLink(h3,s3)
net.addLink(h4,s4)
net.addLink(h5,s5)
```

Figura 3-20. Configuración de la topología en Mininet

A continuación, se inicia la topología y se pide al usuario que indique al programa mediante la pulsación de la tecla “Enter” el momento en el que vea la topología completa representada en la GUI de ODL, lo que quiere decir que los *switch* están corriendo correctamente y el controlador tienen conexión con ellos (Figura 3-21).

```
info('***Iniciando la red\n')
net.build()
info('***Iniciando controlador\n')
OpenDayLight.start()

info('***Iniciando switches\n')
net.get('s1').start([OpenDayLight])
net.get('s2').start([OpenDayLight])
net.get('s3').start([OpenDayLight])
net.get('s4').start([OpenDayLight])
net.get('s5').start([OpenDayLight])

info('***Esperando a que ODL reconozca la topologia\n')
info('***Cuando vea la  topologia completa en la GUI de ODL pulse enter\n')
raw_input()
```

Figura 3-21. Inicio de la topología en Mininet

Al pulsar la tecla mencionada, se realiza un test de alcanzabilidad (Figura 3-22) entre los distintos hosts de la simulación para verificar que todos los dispositivos están operativos. Es entonces cuando se pide la ejecución de los módulos PCE y STEM (Figura 3-23).

```
info('***Probando conectividad\n\n')
net.pingAll()
```

Figura 3-22. Test de alcanzabilidad en Mininet

```
info('***Esperando lanzamiento de PCE y STEM\n')
info('***Cuando sean lanzados pulse enter\n')
raw_input()
```

Figura 3-23. Espera a comienzo de ejecución de PCE y STEM

Una vez lanzados, comienzan las peticiones de recursos al STEM para poder insertar tráfico en la red. Para ello, se abre un *socket* UDP (Figura 3-24) mediante el que se enviará la información del tráfico que se quiere cursar al STEM. Una vez recibida la confirmación de que el flujo ha sido instalado correctamente se insertan los datos en la red. Se intercala la generación de tráficos con distintos orígenes y destinos con la modificación de la topología, tirando y levantando enlaces, lo que nos permitirá comprobar cómo gestiona la red los recursos disponibles y si lo hace correctamente. Para más información sobre el *script* en Python ver ANEXO A.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
address = ('192.168.4.1', 12001 )
```

Figura 3-24. Apertura de socket UDP para comunicación con STEM

4. Análisis de resultados

En este apartado de la memoria se va a realizar un análisis de la arquitectura diseñada, con objeto de evaluar el potencial real de SDN para ofrecer soluciones de Ingeniería de Tráfico. Las pruebas se han realizado sobre topologías de distinto tamaño, con objeto de comparar, especialmente, cómo afectan las dimensiones de la red a la gestión de esta. De esta manera, se comprobará por un lado la cantidad de información que es necesario guardar en la TED para tener el conocimiento de la red necesario para que el PCE pueda ejecutar los algoritmos de cálculo de caminos. Asimismo, se analizará el *overhead* introducido en la red debido a la utilización de SDN, así como las limitaciones debidas a este paradigma en cuanto a tiempos de convergencia en caso de cambios en la red. A continuación, se comprobará cómo actúa la red al detectar caídas de enlaces o la introducción de nuevo tráfico. Por último, se comparará el encaminamiento resultante de utilizar el algoritmo de Dijkstra para calcular los caminos con el de utilizar CSPF.

4.1. Cantidad de información recogida en la TED

En primer lugar, se va a comentar la cantidad de información que ha sido necesario guardar en la TED para poder llevar a cabo los escenarios descritos a continuación. En el apartado 3.2.3 se recoge el detalle de la información almacenada en esta base de datos. Tomando como referencia los ejemplos allí mostrados, dada la poca cantidad de información almacenada en ellas al tratarse de un escenario reducido y la falta de resolución, para cada una de las tablas (`topology_information`), (`bw_n10`), (`link_ocupation_information`) y (`installed_flows_information`) se obtiene un valor de almacenamiento de 16 kBytes de información, el valor estándar dado por la base de datos. En definitiva, y puesto que el número de nodos y de flujos que han conformado las topologías de prueba es reducido y que el volumen de información necesario a alojar en la base de no es muy alto, en la TED utilizada se han guardado menos de 64 kB de datos.

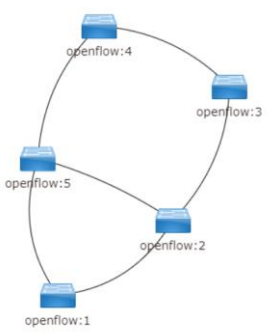
4.2. *Overhead* de señalización introducido en la red

Uno de los parámetros más importantes a la hora de analizar la eficiencia de la red es el *overhead* de señalización introducido. Como se dijo en 2.2.1, la señalización de control en las redes definidas por *software* es *out-of band*, es decir, a través de circuitos distintos a los que cursan el tráfico en la red. Sin embargo, en ese proceso de reconocimiento de la red que hace el controlador SDN, este envía paquetes LLDP a los

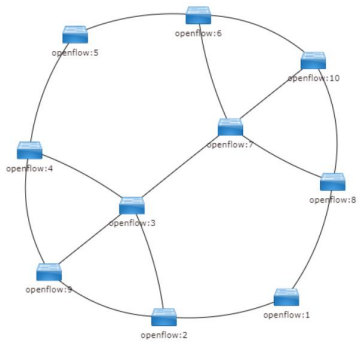
switch que han establecido conexión con este. Cada vez que un *switch* recibe un paquete LLDP lo reenvía por todos sus puertos, introduciendo señalización dentro de los circuitos utilizados para la transmisión de datos útiles. De este modo, se pretende analizar cómo se comporta el tráfico de señalización en las redes definidas por *software*

4.2.1. Reconocimiento de la red y recopilación de estadísticas

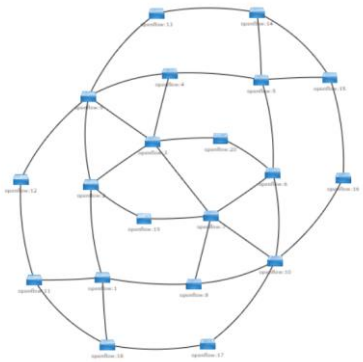
El objetivo de este apartado es estudiar cómo afecta el aumento de equipos en la red a la señalización utilizada por el controlador para reconocer la red y recopilar estadísticas de esta. Para realizar este estudio se han utilizado tres topologías de red distintas, con cinco (Figura 4-1 (a)), diez (Figura 4-1 (b)) y veinte nodos (Figura 4-1 (c)). En los tres escenarios se captura con *Wireshark* el tráfico intercambiado entre ODL y los *switch* de Mininet y los paquetes LLDP intercambiados entre los distintos nodos en toda la red. Los resultados obtenidos pueden consultarse en la Figura 4-2.



(a) cinco nodos



(b) diez nodos



(c) veinte nodos

Figura 4-1. (a) Topologías utilizadas para la evaluación del overhead de señalización

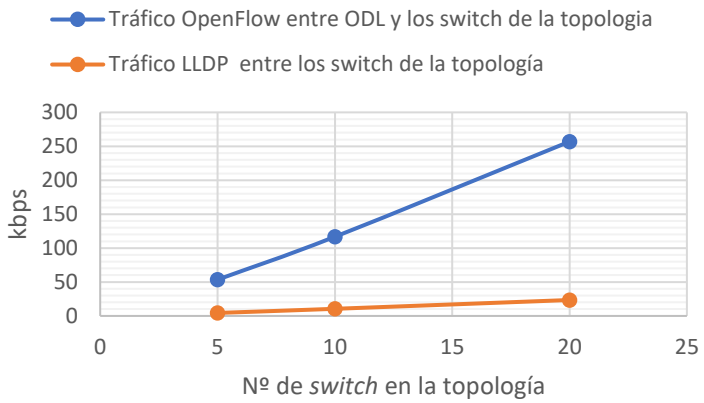


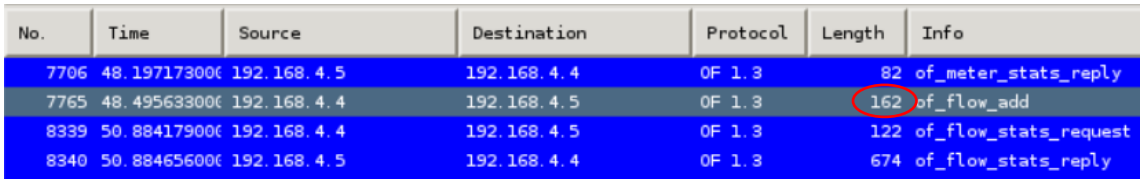
Figura 4-2. Tráfico de señalización en red SDN

Como puede observarse en la Figura 4-2, siendo mucho mayor el tráfico intercambiado entre los *switch* y el controlador, en ambos casos se observa un resultado similar. Doblando el número de nodos en la topología se duplica el tráfico de señalización asociado a SDN, tratándose de un incremento directamente proporcional al crecimiento de la red. Este incremento lineal contrasta, por ejemplo, con el tráfico generado debido a protocolos de encaminamiento dinámicos que requieran difusión de señalización entre todos o parte de los nodos, ya que, en ese caso, el incremento de señalización puede llegar a ser exponencial con el tamaño de la topología (si bien, existen alternativas para su reducción, creando topologías jerárquicas, no llegando en ningún caso a la reducción lineal mostrada).

Por otro lado, resulta interesante realizar una breve comparación entre el tráfico de señalización *in-band* intercambiado entre los *switch* en un escenario SDN y el que se observa entre los distintos *router* en un entorno funcionando con un protocolo de encaminamiento distribuido como puede ser, por ejemplo, OSPF. En los enlaces entre *switch* en SDN tan solo se observa el tráfico asociado al protocolo LLDP, lo que supone unos pocos cientos de bits por segundo. Sin embargo, en el caso de OSPF los *router* se intercambian con sus nodos vecinos mensajes de HELLO periódicos cada diez segundos. Esto supone incluso algo menos de tráfico que en el caso de SDN, aunque debe tenerse en cuenta que en caso de caída de algún enlace la señalización asociada a la actualización de las tablas de encaminamiento supondrá un ligero aumento en estos valores de forma temporal, algo que en las redes definidas por *software* no ocurre al ser este tipo de señalización *out-of-band*, entre controlador y *switch*.

4.2.2. Comunicación con controlador y *switch* OVS

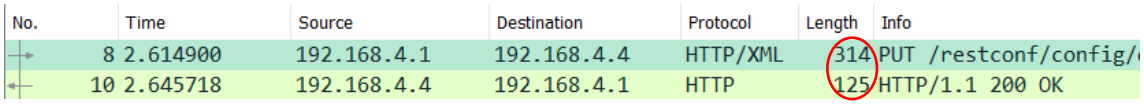
Otro punto de análisis tiene que ver con el *overhead* que se introduce en la red en el momento de instalar o eliminar flujos en un *switch*. Al realizar la instalación de un flujo, el controlador envía a un *switch* un paquete `of_flow_add`, que puede verse en la Figura 4-3. El tráfico generado en esta comunicación es mínimo, siendo de 162 bytes en este caso.



No.	Time	Source	Destination	Protocol	Length	Info
7706	48.19717300(	192.168.4.5	192.168.4.4	OF 1.3	82	of_meter_stats_reply
7765	48.49563300(	192.168.4.4	192.168.4.5	OF 1.3	162	of_flow_add
8339	50.88417900(	192.168.4.4	192.168.4.5	OF 1.3	122	of_flow_stats_request
8340	50.88465600(	192.168.4.5	192.168.4.4	OF 1.3	674	of_flow_stats_reply

Figura 4-3. Instalación de flujo (comunicación controlador/switch)

Analizando el tráfico generado entre el STEM y ODL en el momento de instalar un flujo se obtiene el resultado de la Figura 4-4.

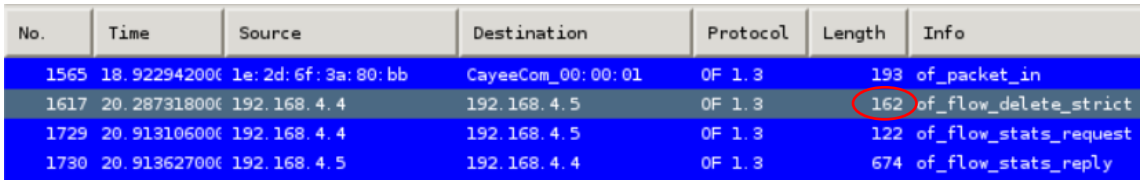


No.	Time	Source	Destination	Protocol	Length	Info
8	2.614900	192.168.4.1	192.168.4.4	HTTP/XML	314	PUT /restconf/config/
10	2.645718	192.168.4.4	192.168.4.1	HTTP	125	HTTP/1.1 200 OK

Figura 4-4. Instalación de flujo (comunicación STEM/controlador)

Puede observarse como este tráfico es algo mayor que el que se intercambiaba entre el controlador y la red, aunque sigue siendo bastante moderado.

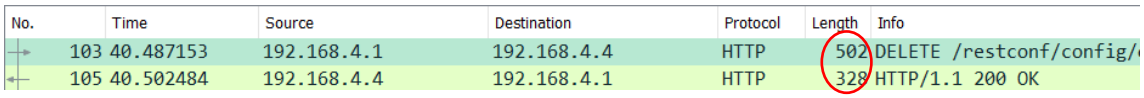
Se comprueba a continuación el tráfico que genera la eliminación de un flujo. En este caso se pretende analizar el paquete `of_flow_delete_strict`. Al igual que ocurría con la instalación de flujos, el tráfico generado para eliminar una entrada en la tabla de flujos de un *switch* es mínimo (Figura 4-5).



No.	Time	Source	Destination	Protocol	Length	Info
1565	18.92294200(	1e:2d:6f:3a:80:bb	CayeeCom_00:00:01	OF 1.3	193	of_packet_in
1617	20.28731800(	192.168.4.4	192.168.4.5	OF 1.3	162	of_flow_delete_strict
1729	20.91310600(	192.168.4.4	192.168.4.5	OF 1.3	122	of_flow_stats_request
1730	20.91362700(	192.168.4.5	192.168.4.4	OF 1.3	674	of_flow_stats_reply

Figura 4-5. Eliminación de flujo (comunicación controlador/switch)

Analizando el tráfico generado entre STEM y ODL en el momento de eliminar un flujo se obtiene el resultado de la Figura 4-6.



No.	Time	Source	Destination	Protocol	Length	Info
103	40.487153	192.168.4.1	192.168.4.4	HTTP	502	DELETE /restconf/config/
105	40.502484	192.168.4.4	192.168.4.1	HTTP	328	HTTP/1.1 200 OK

Figura 4-6. Eliminación de flujo (comunicación STEM/controlador)

Del mismo modo que ocurre con el PUT, el tráfico es mayor que el que se generaba entre el controlador y la red, pero no es excesivo.

Para terminar este estudio se va a realizar una consulta al controlador mediante una petición GET. Este tipo de mensaje es el utilizado por el PCE para extraer información de la topología a través del controlador y poblar la base de datos ubicada en la TED. En este caso, no se genera tráfico añadido en la comunicación entre el controlador y los nodos de la red debido a que la información que se le pide a ODL se encuentra en su base de datos, que se va actualizando gracias a la información de señalización periódica analizada anteriormente. Analizando la comunicación entre el PCE y ODL se obtiene la información presentada en la Figura 4-7, de la que se extrae un resultado similar al de los dos casos anteriores.

No.	Time	Source	Destination	Protocol	Length	Info
7	2.634021	192.168.4.1	192.168.4.4	HTTP	459	GET /restconf/operational
13	2.685088	192.168.4.4	192.168.4.1	HTTP/XML	61	HTTP/1.1 200 OK

Figura 4-7. Consulta del PCE al controlador

Cabe destacar que estos resultados pueden variar ligeramente en función del tipo de información que se pide al controlador o del contenido de los flujos a instalar, pero nunca se generan grandes cantidades de tráfico.

4.3. Convergencia de la red ante cambios

Una de las características más importantes de las redes es el tiempo de convergencia ante fallos, es decir, la rapidez con la que son capaces de detectar un cambio en la topología, como caídas de enlaces o nodos, para poder reaccionar ante esa anomalía e intentar que afecte lo menos posible al tráfico que se está cursando.

De la misma manera que se hizo en apartados anteriores, se va a estudiar el comportamiento de tres escenarios, formados por cinco, diez y veinte nodos, en cuanto a tiempo de reacción ante la caída de enlaces en la red. Así, se podrá comprobar cómo afecta el número de nodos gestionados por un mismo controlador SDN en este aspecto.

Para la consecución de este experimento se ha lanzado cada uno de los escenarios y se han tirado enlaces para comprobar el tiempo que tarda el controlador en percatarse de ese cambio en la red. Se ha repetido el proceso varias veces actuando sobre distintos enlaces de las distintas topologías para asegurar un resultado fiable.

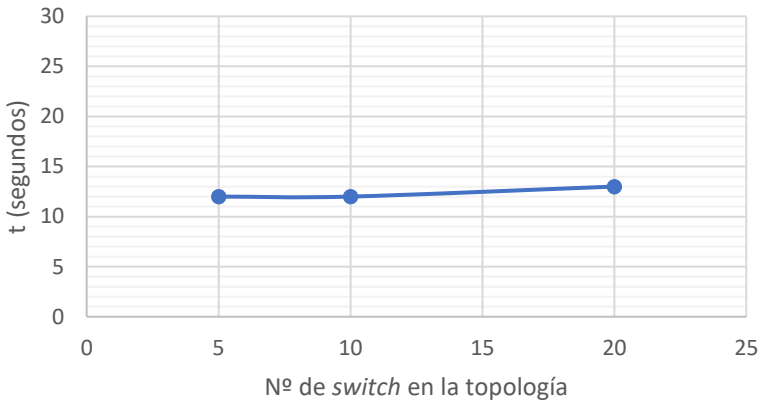


Figura 4-8. Tiempo de convergencia de la red ante cambios

De los resultados presentados en la Figura 4-8 puede deducirse que un incremento de la cantidad de nodos en la topología gestionada por el controlador no afecta al tiempo de convergencia de la red antes cambios en ella, situándose el valor medido entre los diez, quince segundos para los escenarios estudiados. Este tiempo está condicionado, en el despliegue realizado, por el tiempo de actualización de la información topológica de la red (establecido, como se ha dicho, en quince segundos). Este resultado es especialmente relevante en comparación con el funcionamiento clásico de los algoritmos de encaminamiento clásicos. En ese caso, tomando como referencia, por ejemplo, un protocolo habitual, OSPF, la detección de un enlace caído depende de la periodicidad de la señalización entre vecinos (considerando un enlace caído cuando no se reciben mensajes HELLO de un vecino, por defecto, en un intervalo de cuarenta segundos [22], si bien el valor es configurable). Dicho valor condiciona el tiempo de reacción. No obstante, hay que tener en cuenta además el funcionamiento distribuido del protocolo: la identificación de la nueva topología de red por parte de todos los nodos (y en consecuencia el nuevo cómputo de rutas) depende del envío de información de actualización a través de toda la red (en el caso de OSPF, mediante difusión controlada). El tamaño de la topología, por lo tanto, condiciona el tiempo de convergencia, como ilustra de un modo sencillo la Figura 4-9: R4 no identificará la caída del enlace hasta que reciba la información de actualización de R2. Así, cuanto más lejos en número de saltos se encuentre un nodo del lugar de la caída más tiempo tardará en actualizar sus tablas de encaminamiento y, en definitiva, más se tardará, en global, en alcanzar el estado estacionario en relación a la información topológica de la red.

Figura 4-9. Caída de enlace en entorno con OSPF

4.4. Evolución de tráfico en la red

Una vez analizados el *overhead* en la red debido a la utilización de SDN y cómo afecta la cantidad de nodos en la red al tiempo de convergencia de esta frente a cambios, se va a realizar un test sencillo del funcionamiento de la arquitectura implementada. El objetivo de este apartado es comprobar cómo gestionan los elementos de control de la arquitectura (STEM, PCE y controlador) la distribución de tráfico en la red ante peticiones de recursos de distinto tipo y cómo son capaces de reubicar los flujos ante anomalías detectadas en la red como pueden ser caídas de enlaces.

4.4.1. Ubicación equitativa de tráfico

El objetivo de este test de funcionamiento es validar la arquitectura como mecanismo para gestionar un reparto equitativo de los recursos de la red. Para ello, se ha modelado un algoritmo básico de reparto consistente sencillamente en ubicar los flujos en los enlaces menos ocupados para balancear el tráfico en la red. Así, ante la petición de recursos para un tráfico dado, el PCE siempre intenta ubicar los flujos en los caminos con menor número de saltos de origen a destino y con menor ocupación. En este caso se va a trabajar con un escenario de red formado por diez nodos (Figura 4-1 (b)).

En primer lugar, se va a lanzar la topología y se insertarán distintos flujos. El diseño del PCE se ha hecho de manera que, tras calcular los dos caminos óptimos (utilizando Dijkstra o CSPF) para un tráfico dado, se utiliza el que tiene menor ocupación. Así, se intenta hacer una distribución equitativa del tráfico en la red.

Para comprobar el correcto funcionamiento del algoritmo se van a generar dos tráficos UDP entre `openflow:9` y `openflow:10` y entre `openflow:2` y `openflow:7`. Pueden verse los caminos más cortos para llegar a su destino en la Figura 4-10.

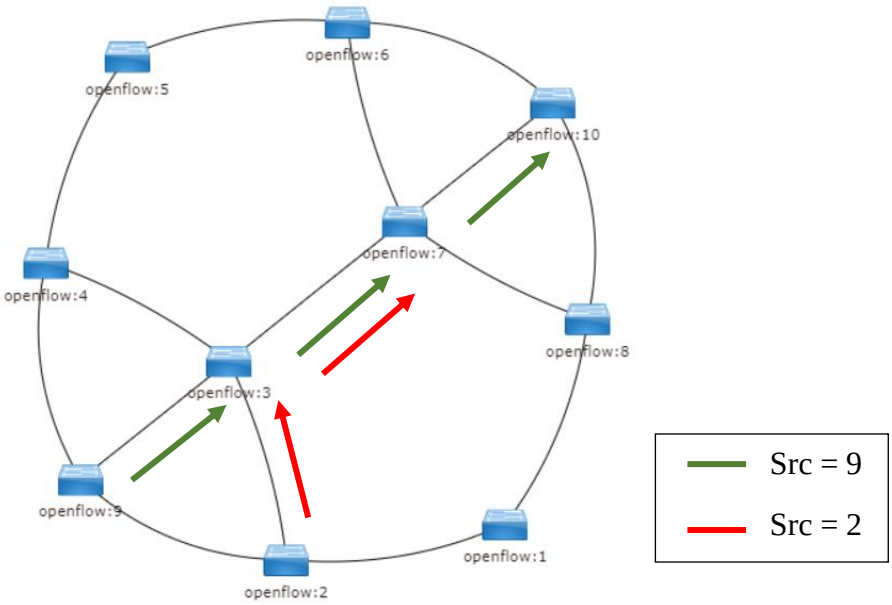


Figura 4-10. Caminos más cortos de origen a destino

El tráfico generado por `openflow:9` ya está cursándose en la red cuando se piden recursos para el tráfico generado por `openflow:2`. De este modo, al observarse que el camino principal para cursar el tráfico de `openflow:2` tiene una ocupación mayor que la del secundario, se instalan los flujos necesarios para cursar el tráfico por este segundo camino a pesar de tener un salto más hasta el nodo destino. Así, el encaminamiento final resultante queda como se muestra en la Figura 4-11.

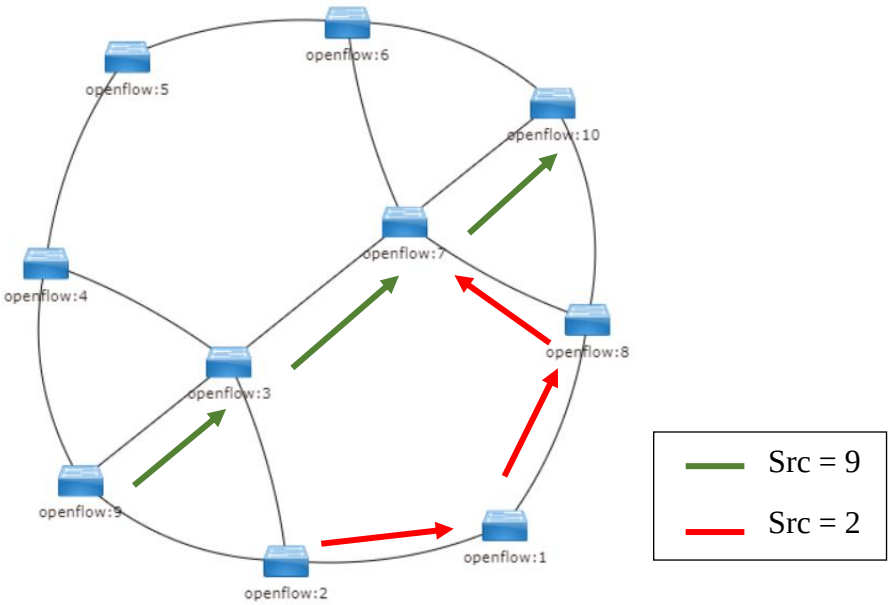


Figura 4-11. Encaminamiento resultante mediante Dijkstra

Pueden observarse en la tabla `installed_flows_information` (Figura 4-12) de la base de datos los flujos que se han instalado en cada *switch* para formar los LSP.

Node	nodeDest	flowID	matches	mplsAction	outputPort	label	priority	bps	lastTxBytes	algorithm
openflow:9	openflow:10	1	ip_dest->10.0.0.10	PUSH	2	10	1	11360	20579	1
openflow:3	-	1	mpls_label->10	FORWARD	3	10	1	11680	21167	1
openflow:7	-	1	mpls_label->10	FORWARD	4	10	1	11680	21167	1
openflow:10	-	1	mpls_label->10	POP	4	10	1	11680	21167	1
openflow:2	openflow:7	1	ip_dest->10.0.0.7	PUSH	1	13	2	11360	7089	1
openflow:1	-	1	mpls_label->13	FORWARD	2	13	2	11680	7297	1
openflow:8	-	1	mpls_label->13	FORWARD	2	13	2	11680	7297	1
openflow:7	-	2	mpls_label->13	POP	5	13	2	11680	7297	1

Figura 4-12. Tabla installed_flows_information resultante del encaminamiento mediante Dijkstra

4.4.2. Reubicación de flujos ante caídas de enlaces

Una de las características más importantes de las redes es su capacidad de reacción ante eventos inesperados en la red. Por ello, se requieren mecanismos que permitan detectar lo más rápido posible estos eventos y reaccionar ante ellos, afectando mínimamente al tráfico que se está cursando. Al igual que en el apartado anterior, se trabaja con una topología formada por diez nodos. Del mismo modo, se insertarán dos tráficos UDP y en cierto momento de la simulación se tirará un enlace para ver cómo la propia red es capaz de detectar un cambio en su topología y de reubicar los flujos por caminos distintos.

En primer lugar, se piden recursos para encaminar tráfico entre `openflow:9` y `openflow:10`. El encaminamiento resultante es el que se presenta en Figura 4-13. Mientras el tráfico se está cursando, se tira el enlace entre `openflow:3` y `openflow:7` (Figura 4-14).

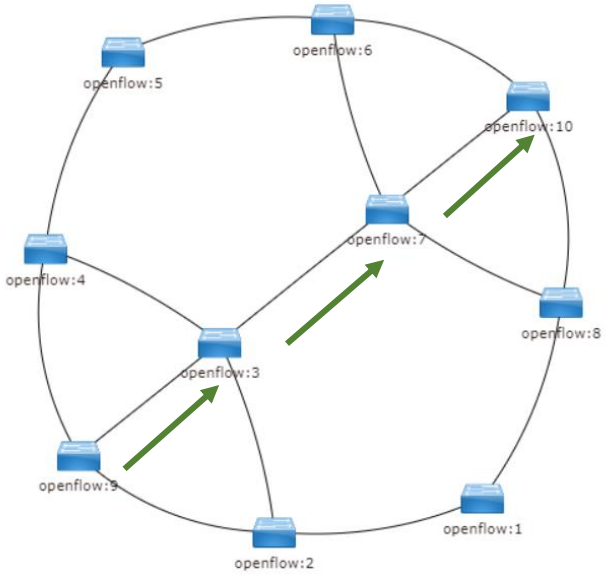


Figura 4-13. Tráfico entre openflow:9 y openflow:10

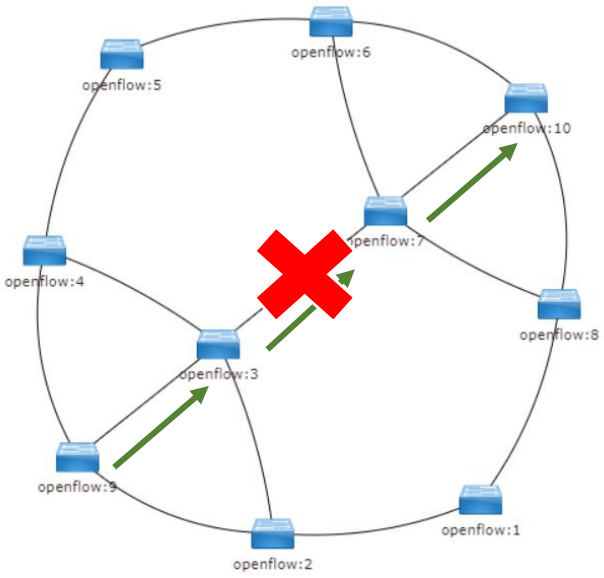


Figura 4-14. Caída de enlace entre openflow:3 y openflow:7

El PCE se encuentra monitorizando continuamente el estado de la red a través del controlador cada quince segundos, un valor estimado por los tiempos de convergencia observados en el controlador, aunque no se ha estudiado un valor óptimo del tiempo de este intervalo. Así, el PCE debería darse cuenta de esta caída y recalcular los LSP. Para comprobar esto se va a capturar el tráfico en las tres interfaces de `openflow:9` (Figura 4-15).

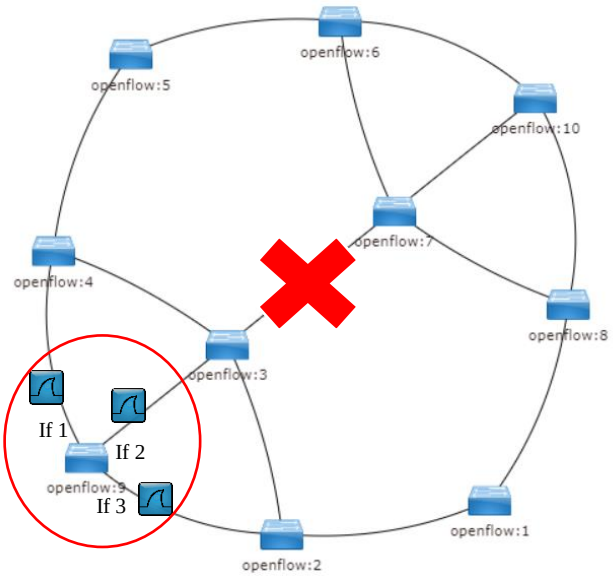


Figura 4-15. Captura de tráfico en las interfaces de openflow:9

Se representa en la Figura 4-16 el tráfico cursado por las tres interfaces de `openflow:9` durante el tiempo de captura.

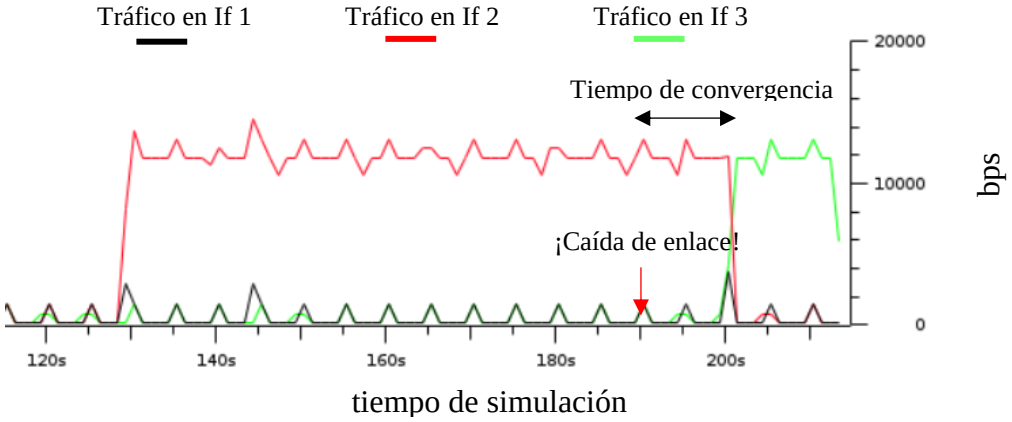


Figura 4-16. Representación del tráfico cursado por las interfaces de openflow:9

Como puede observarse en Figura 4-16, el tráfico entre `openflow:9` y `openflow:10` se cursa por la interfaz relativa al camino que se mostraba en la Figura 4-13, sin embargo, en el momento en el que la red detecta la caída del enlace, un tiempo después al instante del suceso, es capaz de reconfigurar los caminos en la red para seguir cursando el tráfico. En este caso, los nuevos flujos instalados por el PCE serían los que se presentan en Figura 4-17 y Figura 4-18.

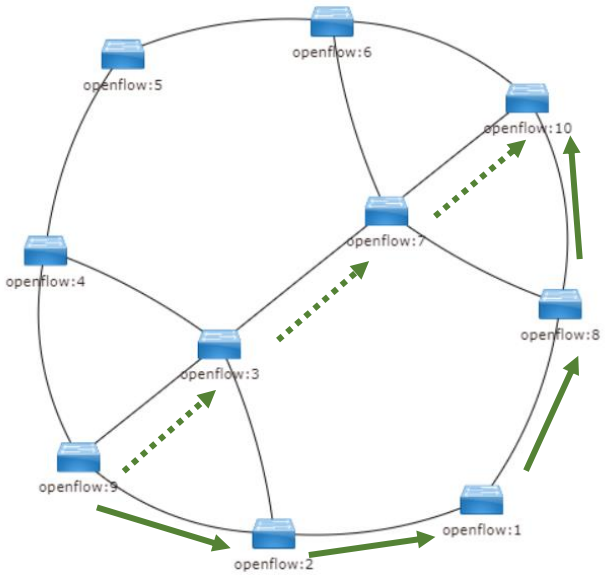


Figura 4-17. Encaminamiento tras la caída del enlace

Node	nodeDest	flowID	matches	mplsAction	outputPort	label	priority	bps	lastTxBytes	algorithm
openflow:9	openflow:10	1	ip_dest->10.0.0.10	PUSH	1	10	1	10981	6390	1
openflow:2	-	1	mpls_label->10	FORWARD	1	10	1	11291	6570	1
openflow:1	-	1	mpls_label->10	FORWARD	2	10	1	11291	6570	1
openflow:8	-	1	mpls_label->10	FORWARD	3	10	1	13237	7300	1
openflow:10	-	1	mpls_label->10	POP	4	10	1	11291	6424	1

Figura 4-18. Tabla installed_flows_information tras la caída del enlace

4.5. Encaminamiento con Dijkstra o CSPF

Para finalizar con la parte de análisis de resultados se van a comparar dos escenarios, uno con encaminamiento mediante Dijkstra y otro mediante CSPF. De este modo, se tomará como referencia el escenario estudiado en 4.4.1, en el que se realizaba un encaminamiento mediante Dijkstra (Figura 4-11), y se comparará con el mismo escenario con encaminamiento mediante CSPF.

En este caso, cuando un flujo solicita recursos comunica al STEM unas restricciones de ancho de banda que deben cumplirse para asegurar los requisitos del tráfico en cuestión. Para poder ver cómo actúa el algoritmo CSPF, todos los enlaces de la red disponen de una capacidad de 100 kbps, excepto el que une `openflow:2` con `openflow:1`, que es de 50 kbps. Se generan dos tráficos con una tasa de 80 kbps, de `openflow:9` a `openflow:10` y de `openflow:2` a `openflow:7`. En este caso, el encaminamiento es distinto al obtenido mediante Dijkstra, ya que uno de los enlaces por los que se quiere cursar el tráfico no tiene la capacidad suficiente para hacerlo (Figura 4-19).

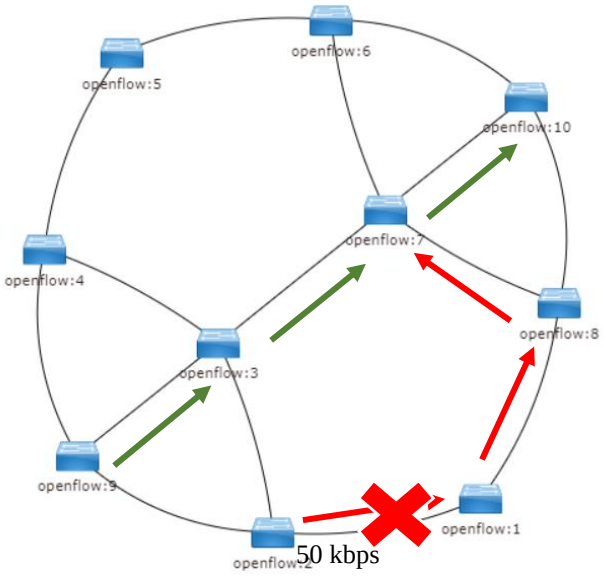


Figura 4-19. Representación del enlace que no cumple con los requisitos del tráfico

Así, a pesar de que el tráfico generado por `openflow:9` se cursa por el mismo camino que se hacía mediante Dijkstra, cuando se piden recursos para el tráfico generado por `openflow:2` el PCE debe buscar una alternativa que, contando con el menor número de saltos hasta su destino y utilizando los enlaces menos ocupados para conseguir una distribución equitativa del tráfico en la red, cumpla con los requisitos expuestos para este tráfico. De este modo, se puede observar en la Figura 4-20 el encaminamiento resultante para la utilización de CSPF como algoritmo de encaminamiento.

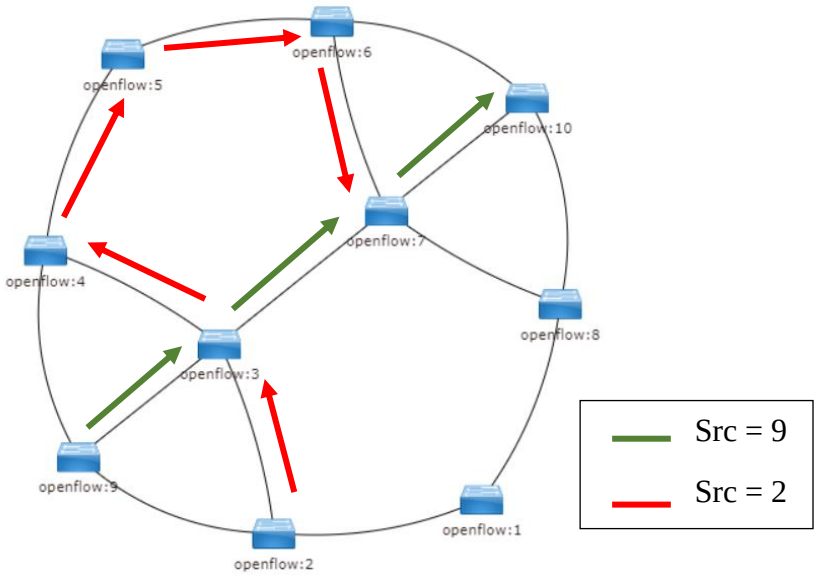


Figura 4-20. Encaminamiento resultante mediante CSPF

Pueden observarse en la tabla `installed_flows_information` (Figura 4-21) de la base de datos los flujos que se han instalado en cada *switch* para formar los LSP.

Node	nodeDest	flowID	matches	mplsAction	outputPort	label	priority	bps	lastTxBytes	algorithm
openflow:9	openflow:10	1	ip_dest->10.0.0.10	PUSH	2	10	1	200315	527235	2
openflow:3	-	1	mpls_label->10	FORWARD	3	10	1	205957	535525	2
openflow:7	-	1	mpls_label->10	FORWARD	4	10	1	223477	542095	2
openflow:10	-	1	mpls_label->10	POP	4	10	1	205957	542095	2
openflow:2	openflow:7	1	ip_dest->10.0.0.7	PUSH	2	12	1	148	148	2
openflow:3	-	2	mpls_label->12	FORWARD	2	12	1	156	156	2
openflow:4	-	1	mpls_label->12	FORWARD	2	12	1	156	156	2
openflow:5	-	1	mpls_label->12	FORWARD	2	12	1	156	156	2
openflow:6	-	1	mpls_label->12	FORWARD	2	12	1	156	156	2
openflow:7	-	2	mpls_label->12	POP	4	12	1	156	156	2

Figura 4-21. Tabla installed_flows_information resultante del encaminamiento mediante CSPF

5. Conclusiones y líneas futuras

En este capítulo se van a presentar las conclusiones extraídas del estudio de las redes definidas por *software* y su aplicación a la Ingeniería de Tráfico y de los resultados obtenidos tras realizar los distintos experimentos. Además, se incluye un apartado de líneas futuras en el que se exponen algunas variantes de este proyecto y nuevos casos de uso.

5.1. Conclusiones

Como se ha visto, las redes definidas por *software* son un paradigma de red muy joven, pero que ayuda a solventar algunos de los problemas que presentan las redes tradicionales debidos a la aparición de nuevas aplicaciones mucho más dinámicas con requerimientos mayores de ancho de banda, sobre todo en cuanto a flexibilidad y facilidad de gestión.

El estudio realizado en este trabajo sobre SDN se ha centrado en su utilización en ingeniería de tráfico, pudiendo comprobar cómo se consiguen solventar algunas de las limitaciones que las redes presentaban hasta ahora. Así, se ha realizado una búsqueda de arquitecturas modulares que permitieran analizar la aplicación de SDN a un entorno de ingeniería de tráfico realizando una implementación propia de algunos de los módulos que componen el plano de control de la arquitectura e integrándolos con Mininet, el *software* que ha permitido la simulación de escenarios SDN, y con OpenDaylight, el controlador escogido para la realización de este proyecto.

Para terminar, se han analizado distintos escenarios en los que se ha verificado la viabilidad de la solución SDN propuesta como alternativa al empleo de soluciones distribuidas (como las ofrecidas por protocolos de encaminamiento clásicos) y observando que, de manera general, la centralización del control permite una visión global con mayor capacidad de reacción, siendo por tanto más flexible y, a su vez, más escalable.

5.2. Líneas futuras

Tomando como punto de partida la prueba de concepto llevada a cabo en este proyecto, pueden definirse nuevos objetivos que, aprovechando el desarrollo del entorno implementado, faciliten el estudio de nuevos casos de uso de las redes definidas por *software*.

En primer lugar, se propone la utilización de protocolos estandarizados como PCEP para la comunicación entre los módulos presentes en el plano de control (controlador, PCE y STEM), ya que en este proyecto se ha simplificado esta comunicación para poder ajustarse a unos tiempos razonables. Además, sería interesante incluir la utilización de tecnologías como *Segment Routing* donde, básicamente, se aplica en el nodo de ingreso a la red una pila de etiquetas a cada paquete, compatibles con el plano de datos de MPLS, que definen las acciones a realizar con ese paquete a lo largo de la red. De esta manera, sólo el nodo de ingreso mantiene información de estado, eliminando señalización entre los distintos nodos de la red y reduciendo la carga del plano de control, así como simplificando el proceso de aprovisionamiento de recursos.

En cuanto a modificaciones de los módulos implementados, se propone proporcionar una interfaz más amigable para el gestor de red del módulo STEM, mostrando información de los distintos tráficos entrantes a la red y de los recursos que se proporciona a cada uno de ellos. Además, se puede dotar al PCE de mayor inteligencia mediante el desarrollo de algoritmos más sofisticados de optimización, así como técnicas de *Machine Learning*. Sobre esta base, se pueden plantear objetivos de planificación y ajuste dinámico adecuados para entornos de movilidad o QoS (facilitando, por ejemplo, la reubicación de flujos para adaptarse a degradaciones en las prestaciones obtenidas).

Por otra parte, como solución a posibles limitaciones del escenario propuesto se plantea acoplar el controlador SDN y el PCE. Al encontrarse desacoplados y contar con bases de datos individuales, pueden darse faltas de sincronismo respecto al estado de la red pudiendo provocar que los tiempos de reubicación de flujos ante cambios en la red tarden algo más de lo deseado. Un funcionamiento conjunto de los módulos (incluyendo *plugin* que incorporen la intercomunicación entre el controlado y sus bases de datos con el PCE, o analizando controladores alternativos que proporcionen esta funcionalidad) podría solventarlo. Otra posible limitación, la elevada dependencia de un único elemento central, plantea el estudio futuro de soluciones con controladores de respaldo o arquitecturas jerárquicas de SDN.

Por otro lado, este proyecto ha consistido en una prueba de concepto básica que ha permitido validar la arquitectura SDN basada en PCE, pero se plantea un futuro análisis de prestaciones estudiando el comportamiento de un mayor número de topologías, de mayor tamaño, inviables en el trabajo actual debido a las limitaciones de *hardware* existentes. Del mismo modo, se propone la sustitución de los *switch* OVS por *router* virtuales compatibles con el protocolo OpenFlow, para estudiar escenarios más realistas contemplando la interconexión de diversas redes.

Para terminar, durante el desarrollo de este proyecto solo se ha estudiado uno de los casos de uso de las redes definidas por *software*, su aplicación a la Ingeniería de Tráfico. Una de las tecnologías que se está desarrollando paralelamente a SDN es la virtualización de funciones de red (NFV), por lo que sería interesante plantear un escenario de integración entre estas dos tecnologías para analizar las ventajas que aporta su trabajo conjunto.

Bibliografía

[1] A. Aguado, V. López, J. Marhuenda, O. González de Dios, J. P. Fernández-Palacios. (n.d.). ABNO: a feasible SDN approach for multi-vendor IP and. *Telefónica I+D*.

[2] A. Botta, A. Dainotti, A.Pescapè. (2012). A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks (Elsevier), 56*.

[3] A. Farrel, Ed., Juniper Networks, Q. Zhao, Ed., R. Li, Huawei Technologies, C. Zhou, Cisco Systems. (2017). An Architecture for Use of PCE and the PCE Communication Protocol (PCEP). *Request for Comments, RFC 8283*.

[4] A. Farrel, J.-P. V. (2006). A path computation element (PCE)-based architecture. *Request for Comments, RFC 4655*. Retrieved from Internet Requests for Comments, RFC 4655.

[5] Alaitz Mendiola, Jasone Astorga, Eduardo Jacob, Senior Member, IEEE, Marivi Higuero. (2017). A Survey on the Contributions of Software-Defined Networking to Traffic Engineering. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*.

[6] Andrei Bondkovskii, John Keeney, Sven van der Meer, Stefan Weber. (n.d.). Qualitative Comparison of Open-Source SDN Controllers. *School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland* .

[7] *BRILLIANT. Dijkstra's Shortest Path Algorithm*. (2018, November 21). Retrieved from https://brilliant.org/wiki/dijkstras-short-path-finder/

[8] Cedric Morin, Géraldine Texier, Cao-Thanh Phan. (2017). On demand QoS with a SDN Traffic Engineering Management (STEM) module.

[9] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus, UUNET (MCI Worldcom). (1999). Requirements for Traffic Engineering Over MPLS. *Request for Comments, RFC 2702*.

[10] D. Awduche, Movaz Networks, A. Chiu, Celion Networks, A. Elwalid, I. W. (2002). Overview and Principles of Internet Traffic Engineering. *Request for Comments, RFC 3272*.

[11] D. Awduche, Movaz Networks, Inc., L. Berger, D. Gan, Juniper Networks, Inc., T. Li, Packet Networks, Inc., V. Srinivasan, Cosine Communications, Inc., G. Swallow, Cisco Systems, Inc. (2001). RSVP-TE: Extensions to RSVP for LSP Tunnels. *Request for Comments, RFC 3209*.

[12] D. King, Old Dog Consulting, A. Farrel, Juniper Networks. (2015). A PCE-Based Architecture for Application-Based Network Operations. *Request for Comments, RFC 7491*.

[13] *Datacomm. SDN*. (2018, November 21). Retrieved from https://www.datacomm.co.id/en/telco/sdn/

[14] DIETI. Universidad Federico II de Nápoles. (n.d.). *D-ITG Documentation*. Retrieved from D-ITG (Distributed Internet Traffic Generator): http://www.grid.unina.it/software/ITG/documentation.php

[15] E. Rosen, Cisco Systems, Inc., A. Viswanathan, Force10 Networks, Inc., R. Callon, Juniper Networks, Inc. (2001). Multiprotocol Label Switching Architecture. *Requests for Comments, RFC 3031*.

[16] Fernández, J. P. (2014). Estudio aplicabilidad SDN en red de agregación EoMPLS. *Universidad Carlos III de Madrid*.

[17] *GitHub. CPqD*. (2018, November 21). Retrieved from http://cpqd.github.io/ofsoftswitch13/

[18] Hopps, C. (2000). *Analysis of an Equal-Cost Multi-Path Algorithm*. Retrieved from Internet Requests for Comments, RFC 2992: https://tools.ietf.org/html/rfc2992

[19] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, Wu Chou. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks (Elsevier)*.

[20] *IEEE*. (2018, November 21). Retrieved from https://spectrum.ieee.org/ns/IEEE_TPL_2017/methods.html

[21] *Introduction to SDN (Software Defined Networking)*. (2018, November 21). Retrieved from NetworkLessons.com: https://networklessons.com/cisco/ccna-routing-switching-icnd2-200-105/introduction-to-sdn-software-defined-networking/

[22] J. Moy, Acend Communications, Inc. (1998). OSPF Version 2. *Request for Comments, RFC 2328*.

[23] JP. Vasseur, Ed., Cisco Systems, JL. Le Roux, Ed. France Telecom. (2009). Path Computation Element (PCE) Communication Protocol (PCEP). *Requests for Comments, RFC 5440*.

[24] Michigan Technological University. (n.d.). *Michigan Tech*. Retrieved from GEOS-Chem Resources: http://www.geo.mtu.edu/geoschem/docs/putty_install.html

[25] Mininet Team. (2018, November 21). *Mininet. An Instant Virtual Network on your Laptop (or other PC)*. Retrieved from http://mininet.org/

[26] Mininet Team. (n.d.). *Download/Get Started With Mininet*. Retrieved from http://mininet.org/download/

[27] *OMNETPP. The CSPF Algorithm*. (2018, November 21). Retrieved from https://omnetpp.org/doc/inet/api-current/neddoc/cspf-algorithm.html

[28] *ONOS Project*. (2018, November 21). Retrieved from https://onosproject.org/

[29] Open Data Center Alliance. (2014). Open Data Center Alliance: Software-Defined Networking rev. 2.0.

[30] Open Networking Foundation. (2012). Software-Defined Networking: The New Norm for Networks.

[31] Open Networking Foundation. (2013). SDN Architecture Overview.

[32] Open Networking Foundation. (2018, November 21). *ONF*. *Open Datapath*. Retrieved from https://www.opennetworking.org/technical-communities/areas/specification/open-datapath/

[33] *OpenvSwitch*. (2018, November 21). Retrieved from https://www.openvswitch.org/

[34] *Project Floodlight*. (2018, November 21). Retrieved from http://www.projectfloodlight.org/floodlight/

[35] Python Software Foundation. (2018, November 21). *Python*. Retrieved from https://www.python.org/

[36] *StackOverflow*. (2018, November 21). Retrieved from https://stackoverflow.blog/2017/09/06/incredible-growth-python/

[37] The Linux Foundation. (2018, November 21). *OpenDayLight*. Retrieved from https://www.opendaylight.org/

[38] *Wireshark*. (2018, November 21). Retrieved from https://www.wireshark.org/