PROYECTO FIN DE MÁSTER EN
SISTEMAS INTELIGENTES
CURSO 2014-2015

MONITORIZACIÓN DEL PLANO DE DATOS EN REDES DEFINIDAS POR SOFTWARE

Isaac Rojas García

Directores:

Luis Javier García Villalba

Ana Lucila Sandoval Orozco

Departamento de Ingeniería del Software e Inteligencia Artificial

Convocatoria de Septiembre

Calificación: 8 – Notable

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Abstract

Software Defined Networks (SDN) is a new network paradigm that removes the rigidity present on current architectures and improves flexibility and management in networks. SDN decouples the control plane and the data plane in routing devices and establishes an open communication interface between them. In addition, SDN proposes a centralized control of the network. The decision taken by the control plane depends on the quality of metrics about the networks performance (missing packets, delays, overhead …). Currently, there are several proposals to monitor the network, but these solutions involve the use of expensive external networks. This work proposes an efficient monitoring SDN framework using the OpenFlow protocol. OpenFlow is the first SDN standard that has been widely used in different research projects. The implementation provides different levels of monitoring, which allow the creation, updating and customization of high-level metrics. The experimental results show the efficiency of the developed framework.

Keywords

Floodlight, Framework, Metrics, Mininet, Monitoring, OpenFlow, SDN, Software Defined Networks.

Resumen

Las Redes Definidas por Software, abreviadamente SDN (de sus siglas en inglés), constituyen un nuevo paradigma que elimina la rigidez presente en las arquitecturas actuales y mejora la flexibilidad y administración de las redes. SDN desacopla el plano de control y el plano de datos en los dispositivos de encaminamiento y establece una interfaz abierta de comunicación entre ellos. Asimismo, SDN propone el control centralizado de la red. La decisión tomada por el plano de control depende de la precisión de la monitorización de la información sobre el rendimiento de las redes y de la detección de eventos en ella como errores en sus enlaces, pérdidas de paquetes, retardos o saturación de la red. En la actualidad, existen propuestas que posibilitan esta monitorización, pero estas soluciones tienen un alto coste al implicar redes externas que requieren la instalación de equipos adicionales. Este trabajo propone un framework SDN eficiente de monitorización usando el protocolo OpenFlow. OpenFlow es el primer estándar SDN que ha sido ampliamente utilizado en diferentes proyectos de investigación. La implementación proporciona diferentes niveles de monitorización, permitiendo la creación, actualización y personalización de métricas. La experimentación realizada demuestra la correcta eficiencia del framework implementado.

Palabras clave

Floodlight, Métricas, Mininet, Monitorización, OpenFlow, Redes Definidas por Software, SDN, Sistema.

Agradecimientos

A mis Directores, Luis Javier García Villalba y Ana Lucila Sandoval Orozco, por su orientación y ayuda y por su comprensión con mi situación personal.

A Ángel Leonardo Valdivieso Caraguay y a Jesús Antonio Puente Fernández por su gran ayuda. Sin ella este trabajo no hubiera sido posible.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Lista de acrónimos

3GPP	*Third Generation Partnership Project*
API	*Application Program Interface.*
BSD	*Berkeley Software Distribution*
CPD	*Data Processing Data*
CPU	*Central Processing Unit.*
GNU	*General Public License*
IP	*Internet Protocol.*
LAN	*Local Area Network.*
MAC	*Media Access Control.*
MFC	*MobileFlow Controller*
MFFE	*MobileFlow Forwarding Engine*
NAT	*Network Address Translation.*
NFC	*Near Field Communication*
ONF	*Open Networking Foundation.*
OTT	*Over the Top*
PID	*Proportional Integral*
QoS	*Quality of Service.*
SDMN	*Software Defined Mobile Network*
SDN	*Software Defined Networking.*
SO	*System Operative*
SSH	*Secure Shell*
SSL	*Security Socket Layer*

TCP	*Transfer Control Protocol.*
TSL	*Transmit Security Layer*
VLAN	*Virtual Local Area Network.*
VM	*Virtual Machine.*
VoIP	*Voice over Internet Protocol.*
XML	*Extensible Markup Language.*

1. INTRODUCCIÓN

1.1. Evolución de las Arquitecturas de Red Tradicionales

Las tecnologías de comunicación han sufrido una gran evolución desde la década de los 80 hasta llegar al término actual de Redes Definidas por Software (abreviadamente SDN, correspondiente a las siglas de su nomenclatura inglesa *Software-Defined Networks*). Sin embargo, hay dos conceptos que forman la base del desarrollo de esta tecnología. Estos avances son:

- Control Centralizado, Redes Activas y Virtualización de Redes.
- Separación del Plano de Control del Plano de Datos.

A continuación se analizan los aspectos fundamentales de cada uno.

Los orígenes del **control centralizado de la red** se remontan a principios de los 80 y se centran en la forma del control de las primeras redes telefónicas de American Telephone and Telegraph (AT&T), que aún sigue vigente. Inicialmente, el control que sufría este tipo de redes se llevaba a cabo en el mismo plano de datos ya que se transportaban en el mismo canal. Por ejemplo, en la red telefónica, en donde la voz y el control de las señales de ésta se realizaban sobre el mismo canal. Ciertas frecuencias, como por ejemplo 2,6 MHz, enviadas externamente en este canal podían inicializar el teléfono y truncar las líneas. Algunos pulsos en la línea podían ser usados para encaminar llamadas o cambiar opciones de los circuitos. Esto ofrecía muchas ventajas en términos de simplicidad; sin embargo, era bastante frágil, inseguro y vulnerable.

A mediados de la década de los 80, AT&T dio un giro particular separando los planos de control y datos en un componente llamado el *Network Control Point* (NCP). Este modelo fue desarrollado solamente para la red telefónica. En

su caso particular, lo que se separó fue la señal de control de la señal de voz. Por tanto, la idea fue que todas las señales de control irían al NCP. Éste se comunicaba con una base de datos que tenía información adicional sobre los clientes. Los beneficios de esta particular tecnología fueron la habilidad de desplegar servicios específicos bajo demanda y, algo más importante, la posibilidad de implementar nuevos servicios rápidamente. Estas ventajas eran prácticamente imposibles con la arquitectura anterior.

Otro avance importante que han experimentado las redes fue la aparición de las **redes activas** en los años 90. Las redes activas [SJSZRP98] [SJSZRP00] permiten realizar tareas personalizadas en los paquetes que viajan a través de los conmutadores. Un ejemplo de redes activas son los *middleboxes* o cajas en la red que realizan tareas de cortafuegos, funciones proxy, servicios de aplicaciones y tareas personalizadas en el tráfico de la red. Esta tecnología fue desarrollada debido a la dificultad para probar nuevos servicios de red en una infraestructura. Las Redes Definidas por Software (SDN) tienen la misma motivación que las redes activas, es decir, acelerar la innovación. En las redes activas hay dos enfoques diferentes: la encapsulación y los conmutadores programables. La encapsulación consiste en que cada mensaje o cada paquete lleva un programa y los nodos activos desplegados en el camino evalúan dicho código. Así, el código puede ser enviado a un entorno y ejecutarse en un conmutador con soporte para esta tecnología. El otro enfoque son los conmutadores programables, en los cuales los comandos son almacenados en los elementos de red que realizan procesamiento personalizado de paquetes. El procesamiento depende de los valores en el campo de la cabecera de los paquetes entrantes.

Es importante resaltar la aparición y el concepto de la **virtualización de redes**. El término virtualización se ajusta a la representación de una o más topologías lógicas de red en la misma infraestructura física subyacente. Hay diferentes instancias de virtualización de redes, algunas de ellas aparecieron en

los 90 como las *Virtual Local Área Networks* (VLAN). Los beneficios que presenta la virtualización son múltiples como, por ejemplo, la compartición de recursos. Usando virtualización se pueden instanciar múltiples encaminadores lógicos en un nodo físico. En otras palabras, se pueden instanciar varias redes virtuales en la misma infraestructura. Sin embargo, esta compartición requiere de la habilidad de aislar los recursos en términos de capacidad de procesamiento de *Central Processing Unit* (CPU), memoria, ancho de banda, tablas de encaminamiento, etc.

En todos los dispositivos encargados de encaminar los paquetes a través de la red se pueden distinguir lógicamente dos planos: el plano de control y el plano de datos.

El plano de control se refiere a la lógica que controla el comportamiento de la red. Ejemplos de este plano son los protocolos de encaminamiento, las configuraciones de *middleboxes* en la red como la de un *firewall*, balanceador de carga, etc. En otras palabras, se puede definir como el cerebro de la red.

Por otro lado, el plano de datos reenvía el tráfico aplicando la lógica del plano de control. Ejemplos del plano de datos son el reenvío de un paquete por un puerto, modificar la cabecera de un paquete IP, la lectura de direcciones MAC, etc. Este plano es normalmente implementado en hardware aunque existen diseños basados en software.

Pero, ¿por qué separar el plano de datos del de control? Una razón de la separación es la de permitir evolucionar y desarrollarse independientemente. Además, en un plano de control separado los dispositivos de red pueden ser controlados por un software de alto nivel independientemente del fabricante del hardware de la red. Adicionalmente, las actualizaciones de los algoritmos, protocolos y políticas pueden ser centralizadas y personalizadas por cada administrador de red.

Un caso de estudio de las ventajas de la separación entre el plano de control y el de datos es un centro de cálculo, donde es relativamente común la necesidad de mover máquinas virtuales de una localización física a otra. Por ejemplo, las instalaciones de Yahoo están compuestas de alrededor de 20.000 servidores en un clúster, resultando en total unas 400.000 máquinas virtuales que necesitan comunicarse entre sí [COU13].

1.2. Redes Definidas por Software

Las Redes Definidas por Software es un nuevo paradigma que reúne los dos avances analizados en el epígrafe anterior. En primer lugar, SDN separa el plano de datos del plano de control en los dispositivos de red. En segundo lugar, SDN propone un control centralizado del plano de control mediante una aplicación de software de alto nivel. De esta manera, los administradores pueden tener un control centralizado y programable del comportamiento del tráfico dentro de la red, sin requerir acceso físico a los dispositivos hardware de red.

Seguidamente se analiza las características básicas de este tipo de redes.

El Protocolo de Internet (*Internet Protocol*, IP) está basado en redes que fueron inicialmente construidas sobre la noción de Sistemas Autónomos Distribuidos (*Autonomous System*, AS) donde para enviar un mensaje desde una fuente A hacia un destino B no es necesario que desde el principio se conozca todo el camino. En la arquitectura actual, el mensaje (paquete IP) va circulando desde un dispositivo hacia otro hasta llegar a su destino. Dicho dispositivo de red tiene un plano de datos y un plano de control integrado y cerrado, que lee la cabecera del mensaje y ejecuta un algoritmo de encaminamiento para determinar el siguiente salto por donde enviar el mensaje, es decir, el camino entre fuente y destino se va estableciendo por medio de los dispositivos de red disponibles.

Por su parte, SDN, al ofrecer separación de planos y un control centralizado, puede establecer el camino más óptimo de la fuente hacia el destino en función de las condiciones de la red. En este paradigma un controlador central recibe la situación actual de la red (número de dispositivos, número de enlaces, ancho de banda disponible, …) y establece el camino entre fuente y destino. Este camino se envía a los encargados de transmitir el mensaje evitando que cada elemento tenga que volver a recalcular la ruta. En otras palabras, el controlador enviará las órdenes a los conmutadores y éstos únicamente transmitirán el paquete salto a salto por el camino previamente asignado. Además, SDN propone que el controlador tenga una interfaz abierta, de tal manera que los usuarios puedan programar sus propias aplicaciones y servicios de red y sean implementados directamente en toda la red.

SDN ofrece importantes ventajas respecto a las tradicionales tecnologías de red. A continuación se analizan los principales avances y campos de aplicación.

Una de las aplicaciones de SDN es la mejora en el rendimiento de los centros de datos. Por ejemplo, el concepto de infraestructura como servicio o IaaS (acrónimo del inglés *Infratructure as a Service*). En este caso, las organizaciones e individuos usan recursos de máquinas virtuales (*Virtual Machine*, VM) bajo demanda. A pesar de que físicamente las máquinas virtuales se encuentran en sitios diferentes, la conexión de dichos recursos tiene que ser transparente para el usuario, es decir, la infraestructura tiene que tener la capacidad de soportar la movilidad de las máquinas virtuales dentro de diferentes centros de datos sin afectar al servicio prestado a los clientes. En este contexto, las Redes Definidas por Software pueden ser programadas para coordinar el transporte de información de manera dinámica, sin necesidad de continuamente configurar los dispositivos de red individuales, como sucede actualmente.

Entre otras aplicaciones de SDN se encuentra el concepto Internet de las Cosas (*Internet of Things,* IoT) o Máquina a Máquina (*Machine to Machine*,

M2M). Estos términos hacen referencia al incremento de dispositivos que continuamente se conectan a la red y transmiten información. Estos dispositivos incluyen a "cosas" intercambiando información entre sí. Por ejemplo, domótica, coches, puertas, luces, monitores de salud personal, etc. En este escenario, la conectividad de dispositivos que continuamente cambian de posición, requiere que la red brinde alta conectividad y pueda modificar sus rutas dinámicamente. SDN, gracias a la administración dinámica y a su visión global de la red, puede implementar nuevos algoritmos que brinden una mejor eficiencia y conectividad.

1.3. Objetivos de la Investigación

Tendencias como la movilidad del usuario, la virtualización de servidores y los nuevos modelos de negocios, aplicaciones y servicios *online* plantean demandas importantes en seguridad, velocidad y rendimiento que las arquitecturas de red convencionales no pueden satisfacer. En este contexto, las Redes Definidas por Software proponen una nueva arquitectura que permite transformar las redes tradicionales en plataformas dinámicas de prestación de servicios.

Las actuales tendencias muestran que el futuro de las redes se basará cada vez más en software, lo que acelerará el ritmo de la innovación. Las Redes Definidas por Software desacoplan el plano de control del plano de datos en los dispositivos de red. De esta manera se realiza una abstracción de la infraestructura para que pueda ser directamente programable por software. Asimismo, fomenta el uso de herramientas de virtualización de redes, permitiendo al personal de Tecnologías de la Información (*Information Technology*, IT) gestionar de manera óptima sus servidores, aplicaciones y servicios. Las Redes Definidas por Software prometen transformar las redes estáticas actuales en plataformas programables flexibles con la inteligencia necesaria para asignar los recursos de forma dinámica.

Por otro lado, el continuo crecimiento de los dispositivos conectados a la red ha incrementado exponencialmente la cantidad de información que circula por la misma. Servicios multimedia *on-line* (youtube, VoIP, *e-commerce*) requieren que los servicios de telecomunicaciones brinden mayor velocidad, seguridad y flexibilidad. En especial, la transmisión de vídeo ocupa gran cantidad del total de información que circula por Internet. Sin embargo, la mayoría de protocolos no ofrecen distinción entre los diferentes tipos de tráfico.

Además, muchos servicios de Calidad de Servicio (*Quality of Service,* QoS) o el emergente concepto de Calidad de Experiencia (*Quality of Experience,* QoE) son propietarios y requieren que toda la infraestructura pertenezca a un determinado proveedor. La QoE toma en cuenta la percepción del usuario respecto a un determinado servicio o aplicación. En otras palabras, la QoE analiza el grado de satisfacción del cliente

Estos servicios requieren de herramientas potentes de monitorización del estado de la red que ayuden a brindar la calidad requerida. La presente investigación va a centrarse en la monitorización del estado de la red utilizando SDN y OpenFlow.

1.4. Estructura del Trabajo

El resto del trabajo está organizado en 7 capítulos con la estructura que se comenta a continuación:

El Capítulo 2 describe las Redes Definidas por Software (SDN), analizando su evolución en los últimos años y las oportunidades y retos que presenta dicha tecnología.

El Capítulo 3 presenta la arquitectura SDN denominada OpenFlow. OpenFlow es el estándar SDN más utilizado por la comunidad científica que ofrece un protocolo abierto de comunicación entre el controlador y el

conmutador. Se muestran los elementos de la arquitectura OpenFlow, haciendo énfasis en el conmutador OpenFlow y en el protocolo OpenFlow como canal de comunicación entre el conmutador y el controlador.

El Capítulo 4 analiza el Sistema Operativo de Red, sus características, ventajas y las principales herramientas disponibles en la actualidad. En otras palabras, se analizan los principales tipos de controladores que actualmente se utilizan en las Redes Definidas por Software. Se clasifican en grupos en cuanto al lenguaje en el que son implementados acompañados de las características propias de cada uno.

El Capítulo 5 especifica el framework de monitorización desarrollado, explicando cuáles son sus principales componentes y los algoritmos utilizados para llevar a cabo la realización del estudio estadístico. Se detallan asimismo los posibles estados de evaluación por los que va pasando el framework con el objetivo de obtener los datos estadísticos de la topología de red que se está analizando.

El Capítulo 6 se centra en los aspectos de implementación del framework de monitorización de redes SDN. Se introducen, asimismo, diferentes herramientas usadas para la simulación, haciendo hincapié en los conceptos principales del programa para la transmisión de vídeo denominado VideoLan y el simulador usado para crear las redes SDN denominado Mininet.

El Capítulo 7 contiene las pruebas realizadas y los resultados obtenidos al aplicar el framework sobre diferentes topologías de redes SDN. Para cada una de las topologías analizadas se muestran gráficas relativas a los parámetros de velocidad de transmisión, tasa de paquetes perdidos y retardo.

Finalmente, el Capítulo 8 muestra las principales conclusiones de este trabajo y las líneas futuras de investigación que se pueden derivar del mismo.

2. REDES DEFINIDAS POR SOFTWARE

2.1. Introducción

El nacimiento de nuevos servicios y aplicaciones *on-line*, tanto en terminales fijos como en dispositivos móviles han hecho de las redes de comunicación un punto estratégico, tanto en empresas, instituciones y hogares. La continua evolución de estos servicios y la creciente información que circula en internet han traído retos imprevistos a los desarrolladores y empresas. En especial, los nuevos dispositivos que, gracias a los avances en *Micro-Electro-Mechanical Systems* (MEMS), automáticamente guardan, procesan y envían información con datos relevantes relacionados con las actividades humanas a través de la red. Este tipo de dispositivos, principalmente constituidos por sensores y actuadores (RFID, dispositivos Bluetooth, redes de sensores, sistemas embebidos, …) han dado lugar al nacimiento de nuevos conceptos y paradigmas como es el de IoT.

En 2011 el número de dispositivos conectados en el planeta sobrepasó al número de habitantes. Actualmente, existen 9 billones de dispositivos conectados y se espera una cifra de 24 billones para el 2020 [GBMP13]. Estos dispositivos utilizan diferentes formas de conectarse a la red; entre otras, la infraestructura de red tradicional. Sin embargo, los equipos y protocolos de red tradicionales no fueron diseñados para soportar un alto nivel de escalabilidad, alta cantidad de tráfico y movilidad. Las actuales arquitecturas resultan poco eficientes y presentan limitaciones importantes para satisfacer estos nuevos requerimientos.

La infraestructura encargada de transmitir la información procedente de dispositivos IoT (encaminadores, conmutadores, redes 3G-4G, puntos de acceso) tiene que adaptarse a nuevos servicios post-PC (VoIP, Virtualización, QoS, Computación en la Nube, Aplicaciones de IoT) y, al mismo tiempo,

brindar seguridad, escalabilidad, rapidez y disponibilidad, entre otros. Algunos esfuerzos como SENSE [SSI14], *Internet of Things-Architecture* (IoT-A) [ELIP14] o *Cognitive Management Framework for IoT* [VGSKF13], así como nuevos protocolos como el DDRP [SZMM13] han tratado de obtener una conectividad más inteligente entre los elementos de red. Sin embargo, es posible que no sean la mejor opción para cada uno dominios de aplicación y dispositivos en particular (*Smart Grid, Intelligent Transportation, Smart Home, Health Care, Environmental Monitoring, …*). Por esta razón, en los últimos años ha surgido la idea de personalizar el comportamiento de la red y dar flexibilidad a los usuarios para utilizar los recursos de red según sus necesidades. Más aún, el desarrollo de algoritmos para la toma de decisiones en redes IoT requiere que diferentes métodos (algoritmos genéticos, redes neuronales, algoritmos evolutivos y otras técnicas de inteligencia artificial) puedan ser implementados rápidamente en los equipos de red de forma dinámica sin necesidad de esperar un estándar.

SDN es una arquitectura de red que elimina la rigidez presente en las redes tradicionales. Su estructura permite que el comportamiento de la red sea más flexible y adaptable a las necesidades de cada organización, campus o grupo de usuarios. Además, su diseño centralizado permite recopilar información importante de la red y usarla para mejorar y adaptar sus políticas dinámicamente. El desarrollo de SDN en los últimos años ha impulsado nuevos conceptos, como es el sistema operativo de red (*Network Operating System*, NOS), tratando de emular el avance que se ha tenido en sistemas de computación. Gracias a esta herramienta se ha logrado probar SDN en múltiples proyectos (*Home Networking, Data Centers, Multimedia*, entre otras iniciativas). De igual manera, SDN ha impulsado el diseño de modelos que finalmente integran y logran convergencia entre arquitecturas que tradicionalmente son independientes (WiFi – 4G – LTE). Sin embargo, todas estas oportunidades están aún lejanas de ser implementadas globalmente en equipos de producción. Temas importantes como la convergencia con redes

actuales, escalabilidad, rendimiento, seguridad, etc., son retos importantes que deben superarse para ser posicionados en el mercado.

2.2. Separación del Plano de Datos del Plano de Control

La idea de transmitir información entre dos puntos a través de una red hizo necesario el diseño de protocolos de comunicación (TCP/IP, HTTPS, DNS) y la fabricación de equipos especializados en la transmisión de información. Dichos equipos han evolucionado dando lugar a una gran variedad de dispositivos (*hub, switch, router, firewall, middlebox*, …). Esto ha causado un incremento exponencial en el número de dispositivos conectados.

Todos estos dispositivos encargados de transmitir información tienen características similares en su diseño y fabricación. En primer lugar, existe un hardware especializado en el tratamiento de paquetes (plano de datos) y, sobre el hardware, funciona un sistema operativo (generalmente Linux) que recibe la información del hardware y ejecuta una aplicación de software (plano de control). El software contiene miles de líneas de código y su objetivo es determinar el siguiente salto que debería tomar un paquete para llegar a su destino. El programa sigue las reglas definidas por un protocolo específico (actualmente existen unas 7000 RFCs) o alguna tecnología propia del fabricante. Los equipos modernos también analizan la información de los paquetes en búsqueda de información maliciosa o intrusiones (cortafuegos, sistemas de detección de intrusos). Sin embargo, todo el software o tecnología que se utiliza en la fabricación de estos dispositivos es rígido o simplemente cerrado para el administrador de red. El administrador está limitado a configurar únicamente algunos parámetros, generalmente a través de comandos de bajo nivel usando una interfaz de comandos (CLI). Por otro lado, cada nodo es un sistema autónomo que busca el siguiente salto que debe tomar un paquete para llegar a su destino. Algunos protocolos (OSPF, BGP) permiten que los nodos compartan información de control entre sí, pero únicamente con sus vecinos inmediatos y

de manera muy limitada, con el fin de evitar carga adicional en el tráfico de red. Esto significa que no existe una visión global de la red como un todo. Si el administrador necesita controlar y modificar un camino determinado, el administrador tiene que jugar con parámetros, prioridades o utilizar artilugios para lograr el comportamiento esperado en la red. Cada cambio en la política de red requiere la configuración individual, ya sea directa o de forma remota de cada uno de los equipos. Esta rigidez hace muy complicada la implementación de políticas de red de alto nivel que sean adaptativas, es decir, que sean flexibles y reaccionen dinámicamente según las condiciones de la red.

Al igual que los sistemas operativos evolucionan y se adaptan a las nuevas necesidades y tendencias tecnológicas (soporte multi-CPU, multi-GPU, 3D, soporte pantalla táctil, entre otras), la adaptabilidad de la red a nuevos requerimientos (VLAN, IPv6, QoS, VoIP) se materializa por medio de protocolos o RFCs. Sin embargo, a diferencia del sistema operativo que, gracias a su separación hardware, permite la continua actualización de aplicaciones o directamente su actualización completa, en el área de redes el período de diseño de una nueva idea hasta su publicación en un protocolo y posterior instalación en los equipos puede durar algunos años. Algunos servicios propietarios de los fabricantes requieren que toda la infraestructura de la red sea de la misma firma para funcionar apropiadamente. Esta limitación favorece la dependencia de una tecnología o firma específica.

2.3. Características

El concepto de SDN no es nuevo y completamente revolucionario, sino que más bien surge como el resultado de contribuciones, ideas y avances en la investigación en redes. En [ONF14] se determinan 3 estados importantes en la evolución de SDN: redes activas (de mediados de los 90 a principios de 2000), separación de los planos de datos y de control (2001-2007) y el API OpenFlow y NOS (2007-2010). Todos estos aspectos se analizan a continuación.

La dificultad de los investigadores para probar nuevas ideas en una infraestructura real y el tiempo, el esfuerzo y los recursos necesarios para estandarizar estas ideas en la *Internet Engineering Task Force* (IETF) hizo necesario dar cierta programabilidad a los dispositivos de red. Las redes activas proponen una interfaz programable o *network* API que abre al usuario los recursos individuales de cada nodo como procesamiento, recursos de memoria, procesamiento de paquetes y permitían incluir funcionalidades personalizadas a los paquetes que circulaban a través del nodo. La necesidad de utilizar diferentes modelos de programación en los nodos dio el primer paso para la investigación en virtualización de las redes, así como el desarrollo de *frameworks* o plataformas para el desarrollo de aplicaciones en el nodo. La *Architectural Framework for Active Networks* v1.0 [ONF14] [Ca99] contiene un sistema operativo de nodo (*Node Operating System*, NodeOS) compartido, un grupo de ambientes de ejecución (*Execution Environments*, EEs) y aplicaciones activas (*Active Aplications*, AAs). The NodeOS administra los recursos compartidos, mientras que los EE definen a una máquina virtual para las operaciones de paquetes. Las AA operan dentro de un EE y brindan el servicio extremo a extremo. La separación de paquetes a cada EE depende de un patrón en la cabecera de los paquetes entrantes al nodo. Este modelo fue utilizado en la plataforma *PlanetLab* [Pl14], en donde los investigadores realizaban experimentos en ambientes virtuales de ejecución y los paquetes eran demultiplexados a cada ambiente virtual en función su cabecera. Estos avances resultaron importantes, especialmente en la investigación de arquitecturas, plataformas y modelos de programación en redes. Sin embargo, su aplicabilidad en la industria fue limitada y criticada principalmente por sus limitaciones en rendimiento y seguridad. El trabajo presentado en [WoTu01] es un esfuerzo para brindar un mayor rendimiento a las redes activas; el *Secure Active Network Environment Architecture* [AAKS98] intentó mejorar su seguridad.

El crecimiento exponencial de los volúmenes de tráfico que circulan por la red acarreó la necesidad de mejorar la gestión y de utilizar mejores funciones de administración como es el manejo de los caminos o enlaces que circulan en la red (ingeniería de tráfico), predicción de tráfico, reacción y recuperación rápida a problemas en la red, entre otros. Sin embargo, el desarrollo de estas tecnologías se han visto fuertemente limitadas por la estrecha unión entre el hardware y software de los equipos de red. Además, el continuo incremento en las velocidades de enlace (*backbones*) hizo que todo el mecanismo de transmisión de paquetes (*packet forwarding*) fuese concentrado en el hardware, separando el control o la administración de red a una aplicación de software. Dichas aplicaciones funcionarían mejor en un servidor, ya que presenta mayores recursos de procesamiento y memoria que los disponibles en un solo dispositivo de red. En este sentido, el proyecto ForCES (*Forwarding and Control Element Separation*) [YDAG04] estandarizado por la IETF (RFC 3746) estableció una interfaz entre los planos de datos y de control en los nodos de red. El software *SoftRouter* [LNRS04] utilizaba esta interfaz para instalar *forwarding tables* en el plano de datos de los *routers*. Asimismo, el proyecto *Routing Control Plataform* (RCP) [CCFRS05] propuso un control lógico centralizado de la red. De esta manera se facilitaba la administración y se daba capacidad de innovación y programación de red. RCP tuvo una aplicabilidad inmediata, ya que aprovechó un protocolo de control existente, BGP (*Border Gateway Protocol*), para instalar entradas en las tablas de encaminamiento de los *routers*.

Con la separación de los planos de datos y control se desarrollaron arquitecturas "*clean-slate*" como es el proyecto 4D [GHM05] o Ethane [CFPL07]. La arquitectura 4D propone una arquitectura de 4 capas según su funcionalidad: *data plane, discovery plane, dissemination plane* y *decision plane*. Por su parte, el proyecto Ethane [CFPL07] propone un sistema de control centralizado de enlaces para redes empresariales. Sin embargo, la necesidad de conmutadores personalizados basados en Linux, OpenWrt, NetFPGA con

soporte para el protocolo Ethane, hizo difícil su aplicabilidad. Actualmente, el protocolo OpenFlow [MABP08] es el más utilizado en la comunidad científica y ha sido la base para la realización de diferentes proyectos. Empresas como Cisco también han presentado una propuesta de nueva arquitectura denominado *Cisco Open Network Environment* (Cisco ONE).

Simplificando el análisis previo, el término SDN propone algunos cambios a las redes de hoy en día. En primer lugar, establece la separación o desacople de los planos de datos y de control, permitiendo su independiente evolución y desarrollo. En segundo lugar, propone que el plano de control sea lógicamente centralizado teniendo de esta manera una visión global de la red. Finalmente, se instauran interfaces abiertas entre los planos de control y de datos. Las diferencias entre estas arquitecturas se presentan en la Figura 2.1.

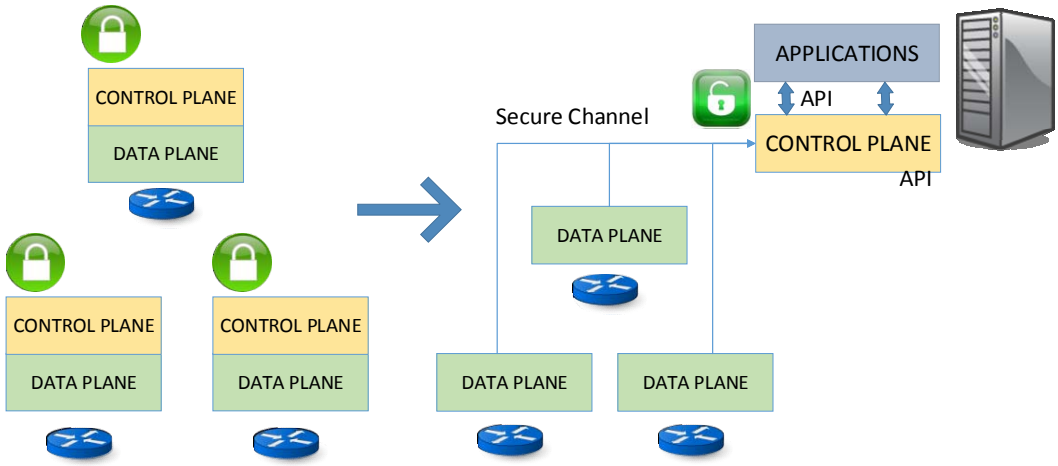

Figura. 2.1: Comparación entre la arquitectura tradicional y SDN.

La programabilidad que ofrece SDN a la red puede compararse como las aplicaciones móviles que hoy en día son ejecutadas sobre un sistema operativo (Android, Windows Mobile). Dichas aplicaciones utilizan los recursos del móvil (GPS, acelerómetro, memoria) gracias al API que ofrece el sistema operativo. De la misma forma, el administrador de red gracias a las API disponibles (propietarias o abiertas) en el controlador, puede gestionar y programar los recursos de la red según las necesidades de los usuarios.

2.4. Aplicaciones SDN

SDN brinda la capacidad de modificar el comportamiento de la red según las necesidades del usuario. Es decir, SDN por sí misma no resuelve ningún problema en concreto, sino que brinda una herramienta más flexible para gestionar de mejor manera las redes. Con el fin de probar las ventajas de esta arquitectura, la comunidad investigadora ha presentado múltiples proyectos de interés. A continuación se resumen algunas de estas aplicaciones.

- ***Home networking***. En el incipiente campo de la IoT, la gestión de los dispositivos y los recursos de red en redes residenciales resulta todo un desafío debido al número de usuarios y dispositivos conectados a un mismo punto (usualmente, un punto de acceso). En [KSXFE11] [KF13] se presenta una implementación de un sistema basado en Openflow que permite la monitorización y administración del acceso de usuarios a Internet basados en *usage caps*, es decir, una capacidad limitada de datos por usuario o dispositivo. El sistema permite visibilidad sobre los recursos de red, administración de acceso a nivel de usuario, grupo de usuarios, dispositivo, aplicación u hora del día e, incluso, el intercambio de capacidad de acceso con otro usuario. El control y monitorización de la red se realiza a través de una interfaz amigable de usuario *Kermit* y la administración de la capacidad y políticas de red por medio del *lenguage Resonance* [NRFC09].

- **Seguridad**. La seguridad también puede ser mejorada debido a la visión global de la red. La seguridad no puede basarse únicamente en la seguridad de los *hosts* (antivirus), ya que cuando éstos se encuentran comprometidos dichas defensas no son efectivas. En [RMTF09] se presenta el sistema *Pedigree* como alternativa de seguridad en el tráfico que circula por la red corporativa. Este sistema, basado en Openflow, permite al controlador analizar y autorizar el tráfico y conexiones que circula en la

red. Los *hosts* tienen un módulo de seguridad a nivel de kernel *(tagger)* que no se encuentra bajo el control del usuario. Este módulo etiqueta las conexiones que solicitan enviar información a través de la red (procesos, archivos, etc.). Dicha etiqueta se envía hacia el controlador (*arbiter*) al inicio de la comunicación. El controlador analiza y acepta o rechaza la conexión según sus políticas. Una vez que se autoriza la conexión, las tablas de flujo correspondientes se instalan en el conmutador. *Pedigree* presenta mayor resistencia a una variedad de ataques de evasión como los gusanos polimórficos. El sistema agrega una mayor carga al tráfico de red y al *host*. Sin embargo, esta carga no es mayor al de un software antivirus común.

- **Redes móviles**. Los dispositivos de la infraestructura de redes portadoras móviles (*mobile carrier networks*) comparten similares limitaciones que las redes de computadores. Las redes portadoras de igual forma siguen estándares y protocolos, por ejemplo los propuestos por el *Third Generation Partnership Project* (3GPP), así como implementaciones propietarias específicas de los vendedores. En este punto el paradigma SDN y su modelo basado en flujos (*flow-based forwarding model*) puede aplicarse a este tipo de infraestructura ofreciendo mejores herramientas. *Software-Defined Mobile Network* (SDMN) [PWH13] es una arquitectura que permite a los operadores apertura, innovación y programabilidad sin depender de un fabricante exclusivo o proveedores de servicios *Over The Top* (OTT). Este modelo consta de 2 elementos: *MobileFlow Forwarding Engine* (MFFE) y el *MobileFlow Controller* (MFC). MFFE es el plano de datos simple, estable y de alto desempeño. Tiene una estructura más compleja que un conmutador OpenFlow ya que soporta funcionalidades adicionales de portadoras como son la tunelización de capa 3 (por ejemplo GTP-U y GRE), funcionalidades de nodos de redes de acceso y de carga flexible. El MFC corresponde al plano de control de alta capacidad, en donde se desarrollan las aplicaciones de redes móviles. De igual manera, se establecen interfaces

3GPP para interconectarse con diferentes tipos de *Mobile Management Entity* (MME), *Serving Gateway* (SGW) *o Packet Data Network Gateway* (PGW).

- **Multimedia**. Los múltiples servicios *on-line* multimedia como, por ejemplo, transmisión de contenido en tiempo real, requieren altos niveles de eficiencia y disponibilidad por parte de la infraestructura de red. Según estudios presentados por CISCO [TZE13], para el 2017 el 73% de todo el tráfico IP (público y privado) será tráfico de vídeo IP (en 2012 era del 60%). Además, en los últimos años ha tomado fuerza el término de QoE [PaPe12], que intenta redefinir la QoS tomando en consideración el nivel de aceptación del usuario a un determinado servicio o aplicación multimedia. En este sentido, SDN permite optimizar las tareas de administración multimedia. Por ejemplo, en [KSDM12] se mejora la experiencia QoE a través de la optimización de rutas. Esta arquitectura consiste de los elementos: el QoS *Matching and Optimization Function* (QMOF) que lee los parámetros multimedia y determina la configuración apropiada para el enlace y el *Path Assignment Function* (PAF) que mantiene actualizada la topología de la red. En el caso de una degradación de la calidad en los enlaces, el sistema automáticamente modifica los parámetros de los enlaces tomando en cuenta las prioridades de los usuarios. Asimismo, el proyecto *Openflow-assisted QoE Fairness Framework* QFF [GEBMR13] busca las transmisiones multimedia que se encuentran en la red y ajusta dinámicamente las características de la transmisión en función de los dispositivos terminales y los requerimientos de la red.

- **Confiabilidad y Recuperación**. Uno de los problemas comunes en las redes tradicionales es la dificultad para recuperarse cuando falla un enlace. El tiempo de convergencia se ve afectado por la limitada información que posee el nodo para recalcular una ruta. En algunos casos, se requiere inevitablemente la intervención del administrador para que manualmente

restablezca los enlaces en la red. En este punto SDN, gracias a su visión global, permite la personalización de los algoritmos de recuperación. En [SSCP12] se propone un sistema basado en Openflow que utiliza los mecanismos de restauración y protección para buscar un camino alternativo. En restauración el controlador busca otro camino cuando recibe la señal de caída de enlace. Por su lado, el método de protección se anticipa a un fallo y calcula previamente un camino alternativo. Por otro lado, al igual que el mal funcionamiento de un conmutador o encaminador puede afectar gravemente la disponibilidad de la red, en SDN el mal funcionamiento del controlador (fallo del NOS, ataque DDoS, error de la aplicación) puede ocasionar un colapso de toda la red. En este sentido, la confiabilidad de la red puede garantizarse por medio de controladores de respaldo (*backup*). Sin embargo, es necesario que tanto el controlador principal como el secundario tengan actualizada y coordinada la misma información de control y configuración. El componente *CPRecovery* [FBMP12] es un mecanismo de *backup* primario que permite la replicación de información entre el controlador principal y de respaldo. El sistema usa la fase de replicación para mantener actualizado el controlador *backup* y la fase de recuperación que inicia el controlador de respaldo al momento de detectar un error en el controlador principal.

- **Virtualización**. El concepto de virtualización en redes tiene similitud con virtualización en sistemas de cómputo, donde diferentes sistemas operativos pueden compartir recursos hardware, es decir, en virtualización de redes se intenta que múltiples redes virtuales puedan operar sobre una misma infraestructura, cada una con una topología y lógica de encaminamiento propia. Inicialmente, las tecnologías VLAN y redes privadas virtuales permiten que varios usuarios compartan recursos de la red. Sin embargo, la separación se controla sólo por el administrador de red con parámetros limitados (puerto del conmutador) y únicamente opera

con protocolos de red conocidos. Con la separación de los planos de control y de datos que soporta SDN, las posibilidades de crear redes virtuales más avanzadas es prometedora. Por ejemplo, Flowvisor [GYAC09] es una plataforma de virtualización que utiliza OpenFlow [MABP08] y se ubica lógicamente entre las capas de control y encaminamiento. Flowvisor [GYAC09] actúa como un proxy transparente entre los controladores y conmutadores. Luego crea un plano virtual y transparente según las políticas establecidas por el administrador, asegurando aislamiento en términos de ancho de banda, *flowspace* y carga en el CPU del conmutador. El usuario puede observar y controlar únicamente su propio *slice*. Adicionalmente, es posible volver a dividir un slice virtual y tener de esta manera una jerarquía de redes virtualizadas, tal y como se muestra en la Figura 2.2.

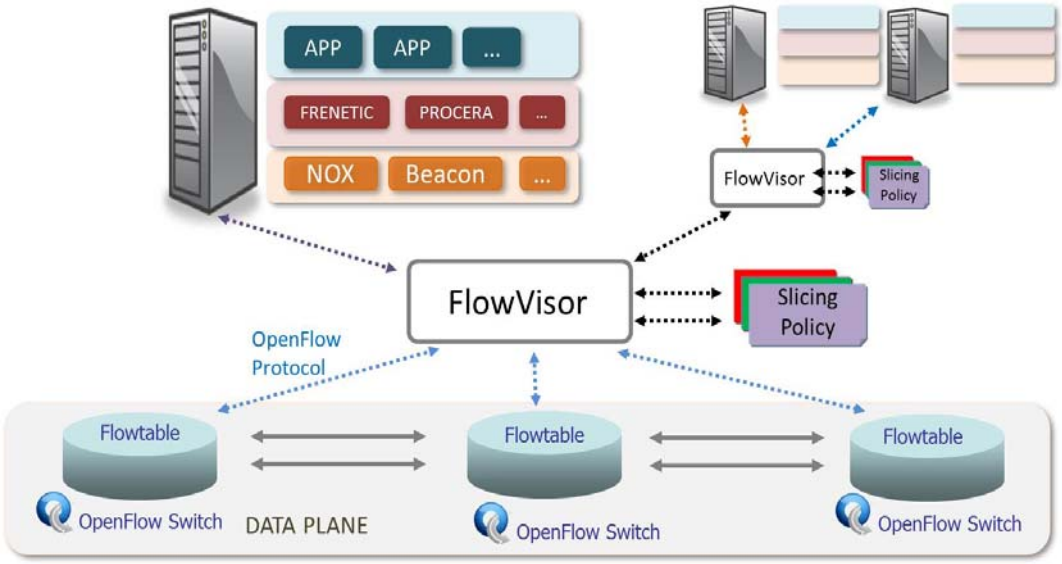

Figura 2.2: Protocolo OpenFlow, Virtualización y Sistemas Operativos de Red.

En [GYAC10] hay una demostración de cuatro exitosos experimentos usando *Flowvisor* [GYAC09] (balanceador de carga, transmisión de *video streaming*, ingeniería de tráfico y experimentos de hardware experimental), cada uno con su propio *slice*. Sin embargo, los experimentos muestran algunos problemas por resolver: interacción inesperada con otros

dispositivos de red instalados, incremento del tráfico de *broadcast* emitidos por dispositivos no OpenFlow y algunas violaciones del aislamiento en CPU, especialmente cuando un *slice* añade una regla de encaminamiento que es enviado por el conmutador a través de un camino lento.

Otro aspecto importante es la integración entre las diferentes operaciones de red y virtualización de sistemas operativos S.O. En virtualización de S.O., las diferentes máquinas virtuales VMs requieren una capa de acceso de red que permita interconexión entre VMs y fuera de él y además soporte funciones de red comunes a una capa física tradicional. El modelo común es establecer comunicación entre nodos virtuales y el NIC físico implementando un típico encaminamiento de capa L2 (*switching*) o L3 (*routing*). Esto dificulta la administración de la red en ambientes virtuales, por ejemplo al momento de migrar VMs entre diferentes servidores físicos. En este enfoque, SDN y virtualización de redes puede ayudar a lograr estos objetivos.

Open *vSwitch* [PPAC09] es un conmutador basado en software diseñado para ambientes virtuales. Este conmutador exporta una interfaz para un minucioso control de la red. Adicionalmente, tiene una partición lógica para el plano de encaminamiento basado en un flexible motor de encaminamiento basado en tablas. El plano de encaminamiento tiene una interfaz externa y puede ser administrado por ejemplo a través de OpenFlow [MABP08]. Con esta abstracción, el controlador puede obtener una vista lógica de múltiples Open vSwitches ejecutándose en servidores separados físicamente.

Otra aplicación interesante en virtualización es la *Virtual Network Migration* (VNM). En redes tradicionales, la migración o el cambio en un nodo de la red requiere la re-configuración y re-sincronización de los protocolos de encaminamiento. Esto causa altos retardos y pérdida de paquetes. En este

ámbito, el uso de nodos virtuales puede reducir significativamente el tiempo de inactividad.

En el sistema VNM propuesto en [PFCMC10], el controlador SDN crea nuevas entradas *flow* para el nuevo conmutador y redirecciona el camino del nodo inicial hacia el siguiente. Luego, el controlador elimina las entradas *flow* del conmutador antiguo permitiendo ser retirado con seguridad. Los resultados de experimentos muestran un tiempo total de migración de 5 ms sin aparente pérdida de paquetes. Más aún, el sistema podría ser reconfigurado dinámicamente para ubicar redes virtuales en diferentes nodos físicos según la hora del día o la demanda de tráfico para ahorrar energía (*green networks*).

2.5. Retos de la Tecnología SDN

Las ventajas que ofrece SDN como tecnología aplicable a las redes de producción masiva se encuentran cercanas pero no disponibles. Más aún, existen algunos retos en términos de seguridad, escalabilidad, confiabilidad, entre otros aspectos, que deben superarse con el fin de ser consideradas aceptables para usuarios comerciales. A continuación se analizan brevemente estos aspectos.

Como se explicó anteriormente, la separación de los planos de datos y de control permite su independiente desarrollo y evolución. En el plano de datos la velocidad de procesamiento de paquetes depende principalmente de la tecnología utilizada en el hardware, ya sea *Application-Specific Integrated Circuits* (ASIC), *Application-specific Standard Products* (ASSP), *Field Programmable Gate Array* (FPGA) o *multicore* CPU/GPP. Por su parte, en el plano de control el rendimiento depende principalmente del hardware y del NOS (Beacon, POX, Floodlight). Sin embargo, el bajo desempeño de uno de los dos puede ocasionar problemas significativos, como son la pérdida o retraso de paquetes,

comportamientos erróneos de la red o denegación de servicio. Por esta razón, es necesario que los diseños de hardware y software para componentes de redes SDN tengan balance en rendimiento, coste y facilidad de desarrollo.

Por otro lado, Openflow utiliza los recursos de hardware comunes en los equipos actuales mediante el uso de tablas de flujo. Sin embargo, SDN puede extenderse más allá de las tablas de flujo y utilizar otros recursos adicionales que ofrece actualmente el hardware [VBG13]. La integración y estudio de nuevas funcionalidades entre el plano de control y el plano de datos personalizado es un campo recién abierto. Aplicaciones como cifrado, análisis, clasificación de tráfico y dispositivos como *middleboxes*, procesadores de paquetes personalizados, entre otros, pueden integrarse y ser usados eficientemente con la tecnología SDN. Por otro lado, el número y la ubicación de los controladores dentro de la red es una pregunta abierta. El análisis presentado en [HSM12] expone que los elementos determinantes para la elección del número y ubicación del controlador son la topología de la red y el rendimiento que se espera de la red.

La seguridad es otro aspecto fundamental que también debe ser tomado en cuenta. Por ejemplo, no todos las aplicaciones de red deben contar con los mismos privilegios de acceso [SSCF13]. Es necesaria la asignación de perfiles, autenticación y autorizaciones para acceder a los recursos de la red. Por otro lado, Openflow establece el uso opcional de TLS (*Trasport Layer Security*) como herramienta de autenticación entre el controlador y el conmutador. Sin embargo, no existen especificaciones claras que brinden seguridad para sistemas de múltiples controladores que intercambian información entre sí y con los conmutadores. Asimismo, debido a que Openflow establece que un paquete desconocido sea enviado completamente o su cabecera al controlador, fácilmente se pueden ejecutar ataques de denegación de servicio mediante el envío de múltiples paquetes desconocidos al conmutador.

La transición de arquitecturas actuales hacia arquitecturas SDN es de igual forma un campo abierto. A pesar de que actualmente ya existen equipos con soporte para Openflow (NEC, IBM) en el mercado, es imposible remplazar toda la infraestructura ya instalada. El período de transición requiere de mecanismos, protocolos e interfaces que permitan coexistencia eficiente de ambas arquitecturas. Actualmente existen esfuerzos para lograr este objetivo: la *Open Networking Foundation* ONF publicó el protocolo IF-Config [OMC13] como primer paso para la configuración de equipos con soporte Openflow. De igual manera, el *European Telecommunications Standards Institute* (ETSI) y el *IETF´s Forwarding and Control Element Separation Working Group* (ForCES) trabajan en la estandarización de interfaces para el correcto desarrollo de esta tecnología.

3. ARQUITECTURA OPENFLOW

3.1. Introducción

OpenFlow es un estándar creado por la Universidad de Stanford, inicialmente diseñado para permitir a los investigadores ejecutar protocolos experimentales en las redes de un campus, que provee un mecanismo estandarizado para ejecutar experimentos sin requerir la exposición de la estructura interna de los dispositivos de red. Actualmente, OpenFlow tiene soporte en conmutadores Ethernet comerciales, *routers* y puntos de accesos inalámbricos.

3.2. Arquitectura OpenFlow

La arquitectura OpenFlow propone la existencia de un controlador, un conmutador OpenFlow y un protocolo seguro de comunicación entre ellos. Dichos elementos se muestran en la Figura 2.3. Cada conmutador OpenFlow está formado por tablas de flujo que son administradas desde el controlador. Cada tabla de flujo consta de tres elementos: *packet header*, *action* y *statistics*. El *packet header* es una máscara encargada de seleccionar los paquetes que van a ser procesados por el conmutador. Los campos que se utilizan para la comparación pueden ser indistintamente de la capa 2, 3 o 4 de la arquitectura TCP/IP. En otras palabras, no existe una separación entre capas como sucede en las arquitecturas actuales. Todos los paquetes que llegan al conmutador son filtrados por medio de este método. El número de campos que el conmutador puede procesar depende de la versión del protocolo OpenFlow utilizado. En la versión OpenFlow v1.0 [OSS09], que es la versión más utilizada, existen 12 campos, mientras que la última versión disponible OpenFlow v1.3 define la existencia de 40 campos incluyendo soporte para IPv6.

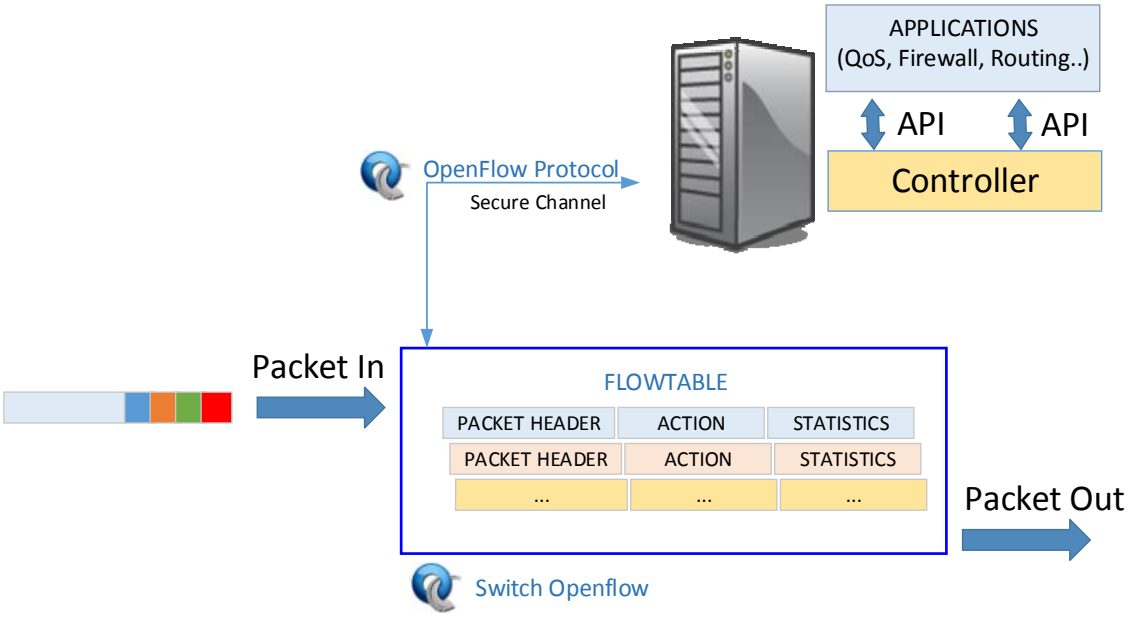

Figura 2.3: Elementos de la arquitectura OpenFlow.

Una vez que la cabecera de un paquete entrante coincide con el *packet header* del *flowtable*, las acciones correspondientes para esa máscara son ejecutadas por el conmutador. Existen acciones principales y opcionales. Las acciones principales son: reenviar el paquete a un puerto determinado, encapsular el paquete y enviarlo hacia el controlador y descartar el paquete. Finalmente, el campo de *statistics* contabiliza entre otros la información del número de paquetes por cada flujo y se utiliza para fines de administración. En el caso de que la cabecera de un paquete entrante no coincide con el *packet header* del *flowtable*, el conmutador (según su configuración) envía dicho paquete hacia el controlador para su análisis y tratamiento.

3.3. Conmutador OpenFlow

Un conmutador OpenFlow consiste en una tabla de flujo (*flow table*) y un canal externo (*secure channel*) que se conecta al controlador. Estos componentes se pueden apreciar en la Figura 2.4. El controlador maneja el comportamiento del conmutador a través del canal seguro utilizando el protocolo OpenFlow. El

controlador puede añadir, actualizar y borrar información de la tabla de flujo, tanto reactivamente (en respuesta a paquetes) como proactivamente (generando acciones).

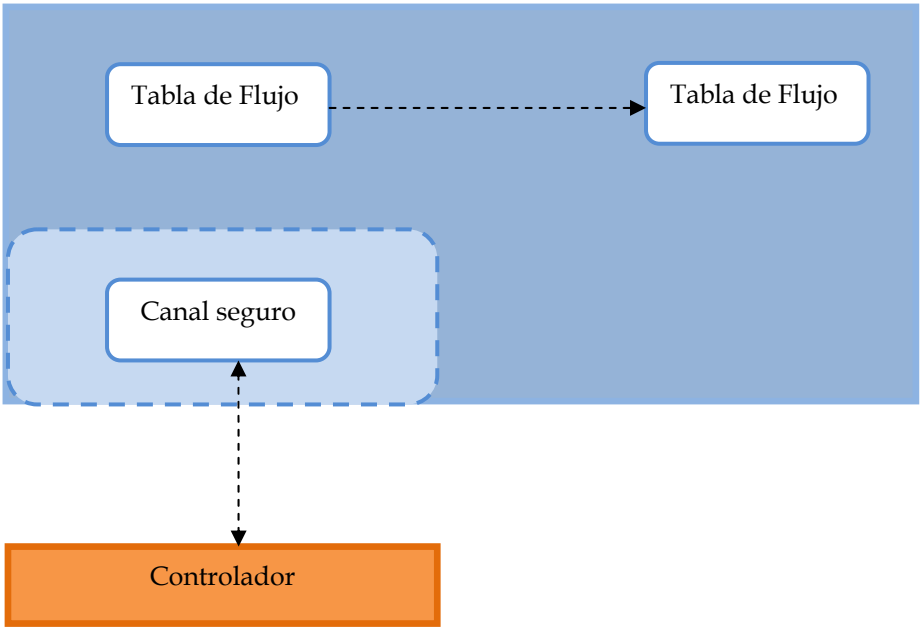

Figura 2.4: Componentes de un conmutador OpenFlow.

Cada tabla de flujo en el conmutador contiene un conjunto de entradas (*flow entries*). Éstas, a su vez, consisten en valores de cabecera (*header values*), contadores de actividad y un conjunto de cero o más acciones para aplicar a los paquetes. Cada vez que entra un paquete, el conmutador compara la cabecera del paquete con las entradas de la tabla de flujo. Si los campos coinciden, las instrucciones asociadas a ese flujo se ejecutan y, en caso contrario, se envía el paquete al cont Protocolo Openflow anal seguro. Por tanto, el controlador es responsable de determinar como se manejan los paquetes sin entrada de flujo válida. Dichas instrucciones se envían al conmutador para reconfigurar la tabla de flujo, permitiendo que se envíen directamente los siguientes paquetes. Las acciones asociadas a las entradas son: envío del paquete por un puerto determinado, re-escritura de la cabecera del paquete y descartar paquete.

3.4. Tablas OpenFlow

En este apartado se describe los componentes de las tablas de flujo, además del mecanismo de comprobación de coincidencia y manejo de las acciones.

3.4.1. Tabla de Flujo

Una tabla de flujo, tal y como se muestra en la Tabla 2.1, es una estructura que contiene 3 campos:

- *Campos de Cabecera*: se usan para hacer la comprobación de coincidencia de los paquetes entrantes.
- *Contadores*: se utiliza para registrar el número de paquetes coincidentes.
- *Instrucciones*: determinan las acciones que se ejecutarán con los paquetes cuyas cabeceras son idénticas a los campos coincidentes.

Campos de Cabecera	Contadores	Instrucciones

Tabla 2.1: Cabeceras de una tabla de flujo.

La Tabla 2.2 muestra los campos de cabecera que pueden ser utilizados para la comparación con los paquetes entrantes. Cada entrada contiene un valor específico o el valor ANY para un valor arbitrario. Los campos coincidentes pueden ser indistintamente de la capa 2, 3 o 4 de la arquitectura TCP/IP.

Campo	Bits	Aplicable a	Notas
Ingress Port	(depende de la implementación)	Todos los paquetes	Representación numérica del puerto de entrante, empezando en 1.
Ethernet Source Address	48	Todos los paquetes en puertos habilitados	
Ethernet Destination Address	48	Todos los paquetes en puertos habilitados	
Ethernet Type	16	Todos los paquetes en puertos habilitados	Un conmutador OpenFlow es requerido para la comprobar la coincidencia del tipo tanto en el estándar Ethernet como 802.2 con una cabecera SNAP y OUI de valor 0x000000. El valor especial 0x05FF es usado para coincidir con todos los paquetes 802.3 sin cabeceras SNAP.
VLAN Id	12	Todos los paquetes del tipo Ethernet 0x8100	
VLAN Priority	3	Todos los paquetes del tipo Ethernet 0x8100	Campo PCP de VLAN.
IP Source Address	32	Todos los paquetes IP y ARP	Puede ser enmascarado por subred.
IP Destination address	32	Todos los paquetes IP y ARP	Puede ser enmascarado por subred.
IP Protocol	8	Todos los paquetes IP e IP sobre Ethernet. Paquetes ARP	Sólo los 8 bits menos significativos son usados para el *ARP opcode*.
IP ToS Bits	6	Todos los paquetes IP	Especifica un valor de 8 bits y está localizado en los 6 bits superiores de ToS.
Transport Source Port / ICMP Type	16	Todos los paquetes TCP, UDP e ICMP	Sólo los 8 bits menos significativos son usados para tipo ICMP.
Transport Destination Port / ICMP Code	16	Todos los paquetes TCP, UDP e ICMP	Sólo los 8 bits menos significativos son usados para tipo ICMP.

Tabla 2.2: Parámetros utilizados en el Experimento 1 de identificación de la fuente.

El tratamiento que un paquete recibe cuando entra en el conmutador se describe en la Figura 2.5. Como se explicó anteriormente, el conmutador compara el paquete con los campos de la tabla de flujo; en caso de coincidencia ejecuta las acciones, actualiza contadores y busca en la siguiente tabla. En caso de que el paquete sea desconocido, el conmutador (según su configuración) descarta o encapsula el paquete y lo envía al controlador.

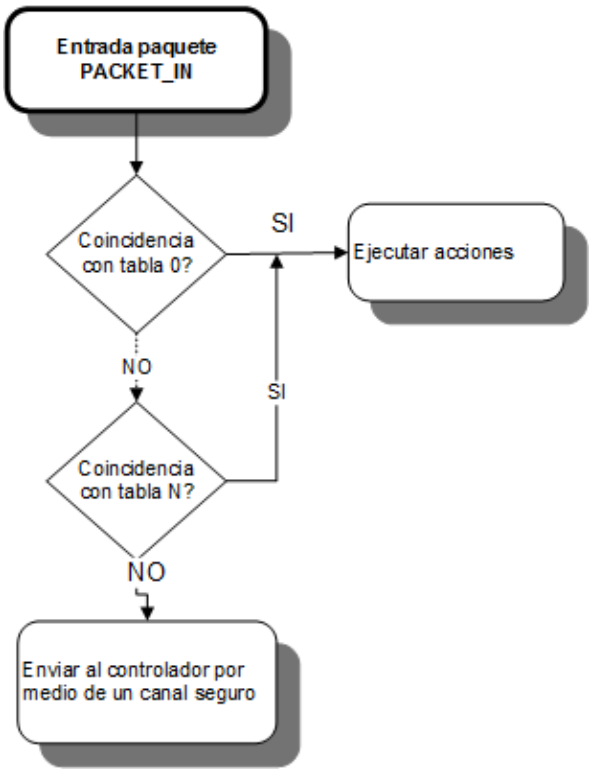

Figura 2.5: Procesamiento de un paquete en un conmutador OpenFlow.

Las diferentes acciones que se pueden ejecutar se dividen en requeridas y opcionales:

- **Acción Requerida**: *Forward*. Esta acción envía un paquete por un puerto específico (tanto físico como virtual). Los puertos estándar son definidos como: puertos físicos, virtuales (definidos por el controlador) y el puerto LOCAL. Además, los conmutadores OpenFlow soportan estas opciones adicionales de envío:

- ALL: envía el paquete de salida a todos los puertos estándares, menos el puerto de entrada.
- CONTROLLER: encapsula y envía el paquete al controlador.
- LOCAL: envía el paquete a la pila de red del conmutador local.
- TABLE: realiza acciones en la tabla de flujo. Sólo para mensajes *Packet-Out*.
- IN PORT: envía el paquete al puerto de entrada.

- **Acción Opcional:** *Forward*. El conmutador tiene la opción de soportar las siguientes acciones en los puertos virtuales:
  - NORMAL: procesa el paquete usando algoritmos tradicionales (no-OpenFlow) del conmutador. El conmutador comprueba el campo VLAN para determinar si se puede o no enviar el paquete a través de la ruta normal de procesamiento. En caso contrario, el conmutador envía un mensaje indicando que no soporta esta acción.
  - FLOOD: inunda el paquete sobre toda la red, excluyendo la interfaz de entrada.
- **Acción Opcional**: *Enqueue*. Este tipo de acción envía un paquete a través de la cola adjunta a un puerto. El comportamiento de envío está determinado por la configuración de la cola y suele proveer soporte básico QoS.
- **Acción Requerida**: *Drop*. El conmutador descarta todos los paquetes que coinciden con una tabla de flujo configurada sin acciones.
- **Acción Opcional:** *Modify-Field*. Este tipo de acción modifica los valores de las respectivas cabeceras en un paquete.

3.5. Canal OpenFlow

El canal OpenFlow es la interfaz que conecta el conmutador OpenFlow con el controlador. Dicha interfaz es conocida como protocolo OpenFlow y por medio de ella se realizan las siguientes acciones:

- Configura y actualiza el conmutador.
- Recibe eventos procedentes del conmutador.
- Envía paquetes al conmutador.

3.5.1. Protocolo OpenFlow

El protocolo OpenFlow define los siguientes tipos de mensajes entre el conmutador y el controlador: *controller to switch*, *symmetric* y *asynchronous*. Los mensajes tipo *controller to switch* gestionan el estado del conmutador, los *asynchronous* actualizan el control de los eventos de la red y cambios al estado del conmutador. Los *symmetric* son enviados ya sea por el controlador o por el conmutador para iniciar la conexión o intercambio de mensajes. De igual manera, se definen 2 tipos de conmutadores: *Openflow-only* y *Openflow-enabled*, según tengan la capacidad de trabajar únicamente con OpenFlow o puedan también procesar el paquete utilizando algoritmos tradicionales de conmutación o de encaminamiento.

En resumen, el protocolo OpenFlow [OSS09] soporta tres tipos de mensajes. Éstos son:

- **Controlador a conmutador**: iniciado por el controlador y usado para inspeccionar el estado del conmutador.
- **Asíncronos**: iniciado en el conmutador y usado para actualizar la información del controlador cuando se producen eventos en la red y cambios de estado en el conmutador.
- **Simétricos:** iniciados por ambos y enviados sin petición.

3.5.1.1. Mensajes Controlador a Conmutador

Este tipo de mensajes son iniciados por el controlador y pueden o no tener respuesta por parte del conmutador. Estos tipos de mensajes son:

- ***Features***: el controlador solicita las capacidades de un conmutador enviando una petición de características (*features request*). El conmutador responde con una respuesta a la petición (*features reply*).
- ***Configuration***: el controlador está capacitado para cambiar y hacer peticiones sobre parámetros de configuración en el conmutador. El conmutador sólo responde a una petición generada desde el controlador.
- ***Modify-State***: estos mensajes son enviados por el controlador para administrar estados en los conmutadores. Su propósito principal es añadir, eliminar o modificar flujos en las tablas OpenFlow. Además, permiten cambiar las propiedades de los puertos del conmutador.
- ***Read-State***: son usados por el controlador para coleccionar estadísticas del conmutador.
- ***Send-Packet***: estos mensajes son usados por el controlador para enviar paquetes por un puerto específico del conmutador y para reenviar paquetes previamente recibidos.
- ***Barrier***: los mensajes de petición/respuesta de barrera son usados por el controlador para asegurar dependencias que haya tenido o para recibir notificaciones de operaciones completadas.

3.5.1.2. Mensajes Asíncronos

Estos mensajes son enviados sin la petición del controlador al conmutador. Estos mensajes denotan la llegada de un paquete, cambios de estado en el conmutador o errores. Los cuatro principales tipos de mensajes asíncronos son:

- ***Packet-in***: Este mensaje encapsula un paquete y lo envía al controlador

para su procesamiento. Este mensaje es enviado cuando no existe una tabla asociada a la cabecera del paquete.

- ***Flow-removed***: Informa al controlador que un elemento de la tabla de flujo ha sido eliminado del conmutador.
- ***Port-status***: el conmutador envía mensajes de este tipo al controlador para informar sobre cambios de estados en la configuración del puerto.
- ***Error***: el conmutador notifica al controlador de problemas existentes.

3.5.1.3. Mensajes Simétricos

Estos mensajes son enviados bidireccionalmente sin petición. Los mensajes simétricos son:

- ***Hello***: son intercambiados entre el conmutador y el controlador a la hora de establecer la conexión.
- ***Echo***: los mensajes *Echo* pueden ser enviados tanto por el controlador como por el conmutador y se envían siempre en respuesta a una petición. Son usados para medir la latencia o el ancho de banda de una conexión controlador- conmutador.
- ***Vendor***: este tipo de mensajes proveen a los conmutadores OpenFlow una forma estándar de ofrecer funcionalidad adicional dentro del espacio del mensaje OpenFlow.

3.6. Ventajas de OpenFlow

Como se ha comentado anteriormente, OpenFlow fue inicialmente propuesto como alternativa para el desarrollo de protocolos experimentales en campus universitarios, donde se puedan probar nuevos algoritmos sin necesidad de interrumpir o interferir con el funcionamiento normal del tráfico de otros usuarios. Hoy en día, la Open *Networking Foundation* ONF [ONF14] es la

organización encargada de la publicación del protocolo Openflow, entre otros protocolos SDN como, por ejemplo, *OF-Config* [OMC13].

La ventaja que OpenFlow presenta con respecto a protocolos SDN previos radica en que OpenFlow aprovecha elementos y funciones de hardware ya disponibles en la mayoría de los equipos de red. Estos elementos son las tablas de encaminamiento y las funciones comunes como leer la cabecera, enviar el paquete a un puerto, descartar paquete, entre otros. Openflow abre estos elementos y funciones para que puedan ser controlados externamente. Esto implica que basta con una actualización de *firmware* para que el mismo hardware pueda ser ya compatible con Openflow. De esta manera las empresas no necesitan realizar un cambio completo de su hardware para implementar SDN en sus productos y servicios.

El controlador recibe la información de los diferentes conmutadores y configura remotamente las tablas de flujo de los conmutadores. Es en el controlador, donde el usuario puede literalmente programar el comportamiento de la red. A diferencia de las redes activas que proponían un *Node Operating System*, OpenFlow abre la noción de un *Network Operating System* (NOS). En este aspecto, en [FRZ13] se define al NOS como el software que abstrae la instalación del estado en los conmutadores de red de la lógica y aplicaciones que controlan el comportamiento de la red. En los últimos años los NOS han ido evolucionando según las necesidades y aplicaciones de los investigadores y administradores de red.

4. SISTEMA OPERATIVO DE RED

4.1. Introducción

Un Sistema Operativo de Red o NOS (acrónimo del inglés *Network Operating System*) describe a todas las herramientas de software que permiten crear aplicaciones y controlar el comportamiento de la red. Los NOS se clasifican según el lenguaje de programación que utilizan. Cada lenguaje tiene sus ventajas en función de administración de la memoria, soporte multiplataforma y rendimiento. En la Tabla 4.1 se resumen los principales NOS.

Lenguaje	Nombre del Controlador
C/C++	NOX, Trema, MUL
Haskell	Nettle, McNettle, NetCore
Java	Maestro, Floodlight, Beacon
OCaml	Mirage, Frenetic
Python	POX, Pyretic, RYU

Tabla 4.1: NOS en función del lenguaje de programación [E13].

4.2. Evolución de los Sistemas Operativos de Red

El concepto de Sistema Operativo de Red (*Network Operating Systems*, NOS) se basa en la función de un sistema operativo en el campo de computación, es decir, el sistema operativo permite al usuario crear aplicaciones usando abstracción de alto nivel de los recursos de información y de hardware. En SDN algunos autores [RFRS12] [SSCF13] [KF13] han clasificado las abstracciones de los recursos de red como interfaces *southbound* y *northbound* (Figura 4.1). Las interfaces tipo *southbound* tienen la función de abstraer la funcionalidad de los conmutadores programables y conectarse con el software controlador. Un claro

ejemplo de interfaz *southbound* es OpenFlow. Sobre las interfaces *southbound* se ejecuta un NOS. Ejemplos de NOS son: NOX [GKPC08], Beacon [E13], Floodlight [PF14], entre otros. Por otro lado, las interfaces *northbound* permiten crear aplicaciones o políticas de red de alto nivel y transmiten dichas tareas al NOS. Ejemplos de estas interfaces son: Frenetic [FHFM11] [FGR13], Procera [VKF12] [KF13], Netcore [MFHW12]; McNettle [VW12]. A continuación se analizan los principales NOS e interfaces *northbound*.

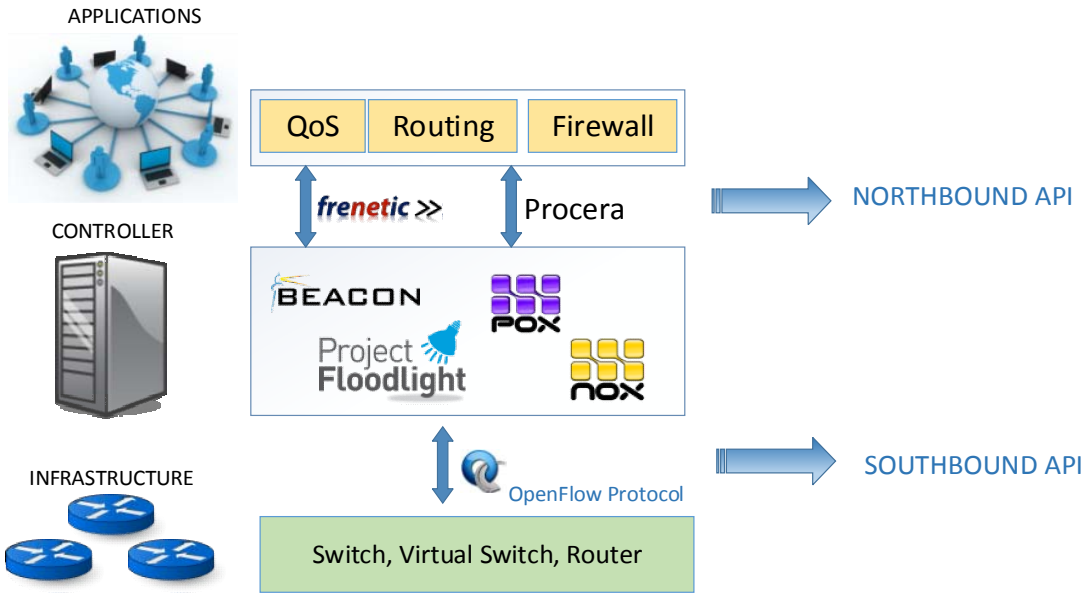

Figura 4.1: NOS e interfaces northbound y southbound.

El software NOX [GKPC08] es el primer NOS para OpenFlow y está constituido por 2 elementos: procesos del controlador y una visión global de la red. En función del estado actual de la red, el usuario puede tomar decisiones y configurar el comportamiento de la red por medio de dichos procesos. En NOX el tráfico se maneja a nivel de flujos, es decir, todos los paquetes con la misma cabecera, son tratados de manera similar. El controlador inserta, elimina entradas y lee los contadores que se encuentran en las tablas de flujo (*flow tables*) de los conmutadores. Por otro lado, debido a la naturaleza dinámica del tráfico NOX usa eventos (*event handlers*) que son registrados con diferentes prioridades para ejecutarse cuando se produce un evento específico en la red. Los eventos más utilizados son: *switch join, switch leave, packet received and switch statistics*

received. Asimismo, NOX incluye *system libraries* con implementaciones y servicios de red comunes. Finalmente, NOX es implementado en C++ ofreciendo alto rendimiento. Existe una implementación enteramente en Python denominado POX, que proporciona un lenguaje de desarrollo más amigable.

Beacon [E13] es un controlador OpenFlow basado en Java. Su interfaz es simple y sin restricciones, es decir, el usuario puede usar libremente los constructores disponibles en Java (*threads, timers, sockets*, …). Por otro lado, Beacon es un NOS basado en eventos, es decir, el usuario configura los sucesos que monitoriza el controlador. La interacción con los mensajes Openflow del conmutador se realiza mediante la librería *OpenflowJ*, una implementación del protocolo OpenFlow 1.0 [OSS09], y la interfaz *IBeaconProvider* que contiene los *listeners IOFSwitchListener*, *IOFInitializerListener* y *IOFMessageListener*. Beacon tiene soporte *multithreading* y facilita implementaciones APIs importantes (*Device Manager, Topology, Routing, Web UI*), así como la capacidad de iniciar, agregar y terminar aplicaciones sin terminar completamente un proceso (*runtime modularity*).

A pesar que un NOS puede manejar las tablas de flujo de los conmutadores, existen algunos problemas que pueden ocasionar el mal funcionamiento de la red [RFRS12] [SSCF13] [GRF13]. Por ejemplo, el controlador recibe el primer paquete que llega al conmutador y que no tiene un *flow table* asignado. Luego, el controlador lo analiza, asigna acciones y reenvía esas instrucciones al conmutador para que los demás paquetes similares tengan el mismo camino. Sin embargo, durante ese tiempo puede llegar el segundo, tercer o cuarto paquete similar al controlador y ocasionar un funcionamiento errático. En otras palabras, virtualmente existen dos procedimientos en ejecución: uno en el controlador y otro en el conmutador, y dichos procedimientos no se encuentran completamente sincronizados.

Otra limitación es la composición, es decir, si el usuario desea configurar dos

servicios diferentes en el mismo conmutador (por ejemplo, encaminamiento y monitorización), se tiene que combinar manualmente ambas acciones en el conmutador, asignar prioridades, mantener la semántica según cada elemento de la red. Esto hace muy difícil el diseño, coordinación y reutilización de librerías. Además, el conmutador tiene que manejar 2 tipos de mensajes simultáneamente: paquetes y mensajes de control. Cualquier descoordinación puede ocasionar que un paquete sea procesado con una política inválida y, consecuentemente, causar un problema de seguridad importante en la red. Por ejemplo, si en una tabla de flujo existen dos entradas con la misma prioridad, el comportamiento del conmutador podría ser no determinístico, ya que la ejecución dependería del diseño del hardware del conmutador. Para superar este tipo de inconvenientes, la comunidad investigadora ha trabajado en el desarrollo de interfaces más simples que interactúen y coordinen el correcto funcionamiento en el conmutador (*northbound*).

Procera [VKF12] [KF13] es un *framework* que permite expresar políticas o configuraciones de red de alto nivel. Esta arquitectura establece diferentes dominios de control y acciones con las que el usuario programa el comportamiento de la red. Los principales dominios de control son: *Time, Data Usage, Status* y *Flow*. Con estos dominios el usuario puede determinar un comportamiento dependiendo, por ejemplo, de la hora del día, cantidad de datos transmitidos, privilegios o grupos de usuarios, tipo de tráfico transmitido, etc. Las acciones pueden ser temporales o reactivas y están expresadas en un lenguaje de alto nivel basado en *Functional Reactive Programming* (FRP) y *Haskell*. En [KF13] se encuentran los detalles de este lenguaje, así como ejemplos del uso de Procera en aplicaciones de monitorización y control de usuarios en un campus universitario.

Frenetic [FHFM11] [FGR13] es un lenguaje de alto nivel para redes SDN desarrollado en Python. Está estructurado en 2 sub-lenguajes: *Network Query Language* y *Reactive Network Policy Management Library*. *Network Query Language*

permite al usuario leer el estado de la red. Esta tarea se realiza mediante la instalación de reglas de bajo nivel (*low-levels rules*) en el conmutador que no afectan al funcionamiento normal de la red. Por otro lado, el *Network Policy Management Library* es diseñado en base a un lenguaje para robots, *Yampa* [CNP03] y librerías para programación web en *Flapjax* [MGBC09]. Las acciones usan un constructor tipo *Rule* que contiene un patrón o filtros y lista de acciones como argumentos. Las acciones principales son: enviar a un puerto determinado, enviar paquete al controlador, modificar la cabecera del paquete, y acción en blanco, que se interpreta como descartar el paquete. La instalación de estas políticas se realiza mediante la generación de *policy events* (similar a *querys*), *primitive events* (*Seconds, SwitchJoin SwitchExit, PortChange*) y *listener (Print, Register*). El resultado de experimentos [FHFM11] muestra que *Frenetic* proporciona simplicidad, así como un ahorro significativo en código y menor consumo de los recursos de la red en comparación a NOX.

Una de las ventajas adicionales de este lenguaje es la composición, es decir, se pueden escribir módulos funcionales independientes y el *runtime system* coordina su correcto funcionamiento en el controlador y en el conmutador. Existen 2 tipos de composición: secuencial y paralela. En la composición secuencial la salida de un módulo es la entrada del siguiente. Por ejemplo, un balanceador de carga que primeramente modifica el destino IP de un paquete y luego busca el puerto de salida en función de la nueva cabecera IP. En la composición paralela ambos módulos son ejecutados virtualmente al mismo tiempo en el controlador. Por ejemplo, si el balanceador envía un paquete con destino IP A hacia el puerto 1 y el paquete con destino IP B hacia el puerto 2. Esta composición resultaría en una función que envía los paquetes entrantes por los puertos 1 y 2.

McNettle [VW12] es un controlador diseñado especialmente para ofrecer alta escalabilidad a la red SDN. Esto se logra mediante un conjunto de manejadores de mensaje (*message handlers*), uno por cada conmutador, que tienen una

función que gestiona las variables *switch-local* y *network state* y administra las acciones de suministro de los flujos de la red. El principio es que los mensajes de un conmutador individual se manejen secuencialmente, mientras que los mensajes de conmutadores diferentes sean manejados concurrentemente. De igual manera, *McNettle* intenta que cada mensaje sea procesado en un único *core* CPU. De esta manera, se reduce al máximo el número de conexiones y sincronizaciones *inter-cores*, mejorándose el rendimiento. Las pruebas realizadas en [VW12] muestran que *McNettle* tienen un desempeño *multicore* considerable en comparación a NOX o Beacon.

El controlador propuesto en [GRF13] se basa en la verificación de las políticas establecidas, en lugar de buscar *bugs* monitorizando el funcionamiento del controlador. Para realizar la verificación, en primer lugar se utiliza el lenguaje de alto nivel denominado *Netcore* [MFHW12], en donde se expresa únicamente el comportamiento de la red, más no su modo de implementación en el controlador. Luego, el *NetCore Compiler* expresa dichas políticas en configuraciones a nivel de conmutador (tablas de flujo). La información de las tablas de flujo es analizada nuevamente por el *Verified Run-time System*, que traduce dicha configuración en un nivel de abstracción más bajo denominado *Featherweght Openflow*. *Featherweght Openflow* es un modelo que permite asegurar que las reglas instaladas en el conmutador son consistentes con la tabla de flujo y que, gracias a primitivas de sincronización, aseguran que el funcionamiento del conmutador sea el correcto. Asimismo, en [RFRS12] se presenta la herramienta *Kinetic*, que ofrece mantener consistencia de actualizaciones en la red mediante dos mecanismos: de paquete y de flujo. En el primero de ellos se garantiza que cuando existe una actualización el paquete que circula en por la red se procesa por una misma configuración. En el segundo se asegura que todos los paquetes pertenecientes al mismo flujo (por ejemplo, la misma conexión TCP) sean tratados de forma similar por los conmutadores de la red.

4.2. NOX / POX

NOX es uno de los primeros controladores OpenFlow de código abierto. En este apartado se agrupan NOX y POX debido a que tienen estructura similar, pero un entorno diferente de implementación (NOX es desarrollado en C++ y POX en Python).

NOX incluye una vista sobre la red en la que incluye la topología a nivel de conmutadores, la localización de los usuarios, *hosts*, *middleboxes*, servicios (por ejemplo, *HyperText Transfer Protocol* (HTTP)) y otros elementos de la red. La vista también incluye todos los enlaces entre nombres y direcciones.

La interfaz de programación de NOX es simple, centrándose sobre eventos, espacio de nombres y vista sobre la red.

- **Eventos:** las redes en general no son estáticas, es decir, los flujos llegan y se van, igual que los usuarios y los enlaces. Para alcanzar este cambio de eventos, las aplicaciones de NOX usan un conjunto de *handlers* o manejadores de eventos que están registrados para ejecutarse cuando una interrupción particular sucede. Estos *handlers* son ejecutados en el orden de prioridades (especificado durante el registro del *handler*). El *handler* devuelve un valor indicado a NOX si hay que parar la ejecución de ese evento, continuarlo o pasar dicho evento al siguiente *handler* registrado. Algunos eventos son generados directamente por los mensajes OpenFlow como *switch join, switch leave, packet received* y *switch statistics received*. Otros eventos son generados por las aplicaciones NOX como resultado de un procesamiento de eventos de bajo nivel. Por ejemplo, NOX incluye aplicaciones que autenticarán a un usuario a través de la redirección de tráfico HTTP con el evento *packet received*.

- **Vista de la red y espacio de nombres:** NOX incluye un número de aplicaciones *base* las cuales construyen la vista de la red y mantienen un

espacio de nombres de alto nivel que puede ser usado por otras aplicaciones. Estas aplicaciones manejan la autenticación del usuario y del *host* para obtener los nombres de los *hosts* a través de una monitorización DNS. La vista de la red debe ser consistente y hacerla disponible para todas las instancias de controladores NOX, por lo que la escritura dirigida a ellos se torna compleja. Debido a esto, las aplicaciones NOX sólo deben avisar cuando se detecta un cambio en la red y no para todos los paquetes recibidos.

- **Servicios de alto nivel:** NOX incluye un sistema de librerías para proveer implementaciones eficientes de funciones comunes a muchas aplicaciones de la red. Éstas incluyen un módulo de encaminamiento, clasificación rápida de paquetes, servicios estándar como *Dynamic Host Control Protocol* (DHCP) y DNS, y un módulo de filtrado basado en políticas de la red.

4.3. Floodlight

Floodlight [PF14] es un controlador OpenFlow desarrollado en Java. Aparece como evolución del controlador Beacon [E13], por lo que igualmente comparten similar estructura. Java brinda soporte multiplataforma y sencillez de programación. El usuario puede hacer uso de herramientas típicas de Java, como son *threads*, temporizadores, *sockets*, etc.

Floodlight incluye la librería *OpenFlowJ* para trabajar con mensajes OpenFlow. Esta librería es una implementación Java orientada a objetos de la especificación de la versión 1.0 de OpenFlow. Los *listeners* se utilizan para notificar cuando los conmutadores son añadidos o eliminados (*IOFSwitchListener*), y para recibir distintos tipos de mensajes específicos OpenFlow (*IOFMessageListener*).

Además, *Floodlight* contiene aplicaciones básicas de referencia las cuales

constituyen el *core*. Esta API adicional es la que se muestra en la Figura 4.2.

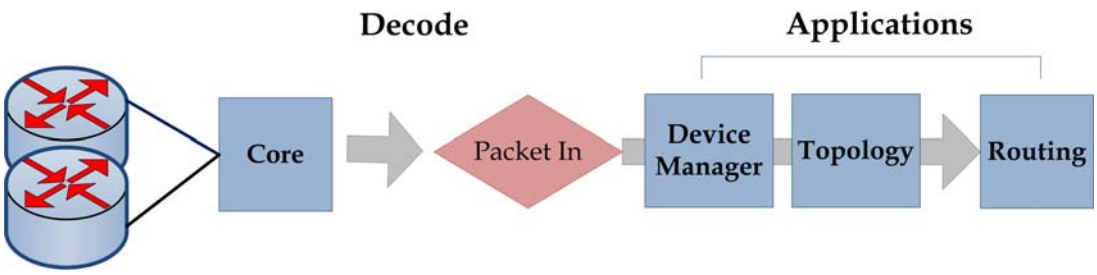

Figura 4.2: *Pipeline* del *thread IOFMessageListener* de Beacon [E13].

Las aplicaciones básicas disponibles son:

- ***Device Manager*:** muestra los dispositivos que aparecen en la red, incluyendo sus direcciones (Ethernet e IP), fecha de último uso, el conmutador y el puerto que han sido vistos por última vez. El *Device Manager* provee una interfaz (*IDeviceManager*) para buscar dispositivos conocidos y la habilidad de registrarlos para recibir eventos cuando nuevos dispositivos sean incluidos, actualizados o eliminados.
- ***Topology*:** descubre los enlaces entre los conmutadores OpenFlow. Su interfaz (*ITopology*) permite la recuperación de una lista de los enlaces y el registro de eventos para ser notificados cuando los enlaces son incluidos o eliminados.
- ***Routing*:** provee enrutamiento de la capa L2 con el camino más corto entre dos dispositivos de la red. Esta aplicación exporta la interfaz *IRoutingEngine*, permitiendo implementaciones con diferentes lógicas de encaminamiento. La implementación incluida usa todos los métodos de computación de camino más corto entre dos pares. Esta aplicación depende tanto de la *Topology* como del *Device Manager*.
- **Web:** provee una *Web User Interface* (UI) para Floodlight. La aplicación web provee la interfaz *IWebManageable*, permitiendo a los desarrolladores añadir sus propios elementos UI.

Por otro lado, Floodlight contiene funciones adicionales como se muestra en la Figura 4.3. Se presenta un Java API para el desarrollo de aplicaciones que residen dentro del controlador y requieren alta eficiencia de procesamiento (por ejemplo, procesamiento de paquetes tipo PACKET_IN). Adicionalmente, el REST API (por las siglas de *Representational State Transfer*) está disponible para configuración remota (puerto 8080 por defecto) de los diferentes servicios del controlador. De esta manera, los usuarios pueden crear aplicaciones que invoquen servicios del controlador mediante peticiones web (HTTP REST).

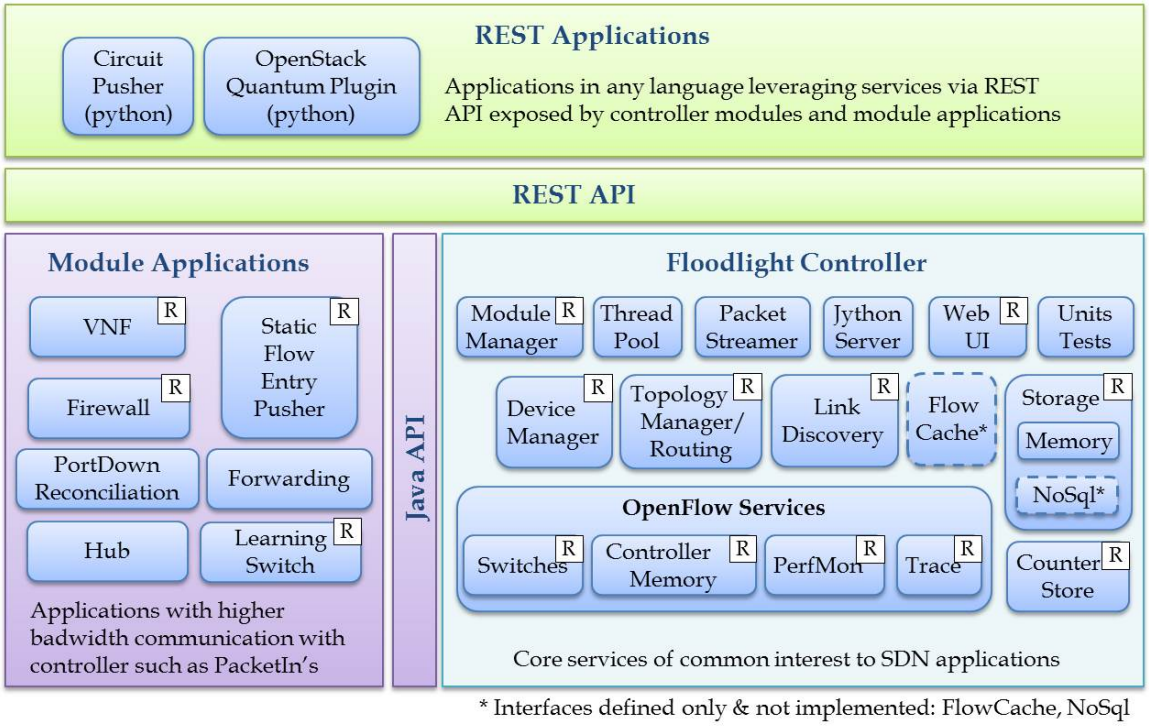

Figura 4.3: API de Floodlight [PF14].

4.3.1. Modularidad en Tiempo de Ejecución

Muchos de los controladores OpenFlow tienen la habilidad de seleccionar qué aplicaciones compilar (modularidad en tiempo de compilación) y qué aplicaciones ejecutar cuando el controlador comienza a actuar (modularidad en tiempo de inicio). Floodlight tiene la capacidad adicional de comenzar y finalizar aplicaciones mientras se está ejecutando, además de añadirlas y

eliminarlas (modularidad en tiempo de ejecución) sin que el proceso de Floodlight se termine.

4.4. Pyretic

Pyretic [MRFRW13] es uno de los miembros de la familia de lenguajes de programación SDN. Surge como resultado de uno de los lenguajes de programación de redes como es Frenetic [FO14] pero con sintaxis de Python, ofreciendo una manera simple para expresar políticas de alto nivel, las cuales compilan las reglas de coincidencia OpenFlow. Permite a los programadores especificar políticas en lo que son *paquetes localizados*. Una política está basada en la combinación de un paquete y su localización en la red.

4.4.1. Características

Pyretic ofrece muchas características entre las cuales se encuentran la habilidad de escribir políticas de red como funciones. En otras palabras, Pyretic permite al programador escribir una función que recoge como entrada un paquete, devolviéndolo en diferentes localizaciones en la red.

Pyretic provee predicados booleanos en contraste a otros controladores. Además, a diferencia de manejar reglas de acciones por medio de coincidencias, Pyretic permite la creación de políticas usando conjunciones y predicados como *and* y *not*. En la Tabla 4.2 se muestra una lista de algunas políticas del lenguaje Pyretic.

Sintaxis	Descripción
None	Devuelve el conjunto vacío.
Identidad	Devuelve el paquete original.
Match	Devuelve la identidad si el campo del paquete coincide con un valor particular; en otro caso, devuelve el conjunto vacío.
Mod	Devuelve el mismo paquete pero cambia el campo de la cabecera virtual del paquete a un valor específico o envía el tráfico a través de un puerto de salida particular modificando solamente el atributo que se encarga de dicho puerto en el paquete.
Flood	Devuelve un paquete por cada puerto en el árbol de recubrimiento de la red.

Tabla 4.2: Políticas atómicas del lenguaje Pyretic [MRFRW13].

Por otro lado, Pyretic ofrece registros o campos virtuales en la cabecera del paquete, que habilitan al programador a referirse tanto a las cabeceras actuales como a las virtuales.

Por último, Pyretic provee operadores de composición tanto paralelos como secuenciales para hacer posible que el operador de red escriba políticas complejas compuestas por otras más sencillas.

4.4.2. Predicados

En OpenFlow el conmutador comprueba si la cabecera de paquetes coincide con una regla; en caso contrario, busca la tabla siguiente. Esto hace muy difícil expresar las políticas que envuelven reglas complejas como son conjunciones o negaciones. En este contexto, Pyretic permite analizar los paquetes directamente por medio de predicados. Por ejemplo, en la Figura 4.4 se muestra una conjunción de dos predicados de coincidencia que devolverá los paquetes que tengan como dirección IP destino 10.0.0.3 ó 10.0.0.4.

$$match(dstip = 10.0.0.3) \mid match(dstip = 10.0.0.4)$$

Figura 4.4: Conjunción de predicados

Las cabeceras virtuales del paquete son una manera unificada de representar metadatos en un paquete. En Pyretic un paquete es un diccionario que mapea un nombre de un campo a un valor. Se puede aplicar el predicado de coincidencia a paquetes que llegan a un puerto de entrada particular de un conmutador o que tiene una dirección MAC destino.

Como se ha especificado anteriormente, una composición secuencial realiza la primera operación y, si se cumple, realiza la segunda. Un ejemplo que envía paquetes a un puerto de salida dependiendo de la dirección IP destino es el que se aprecia en la Figura 4.5.

$$match(dstip = 2.2.2.8) \gg fwd(1)$$

Figura 4.5: Composición secuencial de dos predicados.

Por otro lado, la composición paralela realiza ambas operaciones simultáneamente. En la Figura 4.6 se muestra un ejemplo que envía paquetes de un puerto de salida a otro, dependiendo del valor de la dirección IP destino.

$$match(dstip = 2.2.2.8) \gg fwd(1) + match(dstip = 2.2.2.9) \gg fwd(2)$$

Figura 4.6: Composición paralela de dos predicados.

5. FRAMEWORK DE MONITORIZACIÓN

5.1. Introducción

Tradicionalmente, se han usado muchas técnicas de monitorización sobre los sistemas de redes. Estos métodos de monitorización requieren de la instalación de dispositivos físicos o de la configuración a través de software [NOMS14]. Este tipo de configuración suele ser generalmente una tarea tediosa y costosa de llevar a cabo. El protocolo OpenFlow posibilita el uso de interfaces necesarias para implementar monitorizaciones de red sin un alto grado de personalización. Hay diferentes tipos de técnicas de monitorización. Las principales están basadas en medidas activas y pasivas de la red. La presente propuesta está basada en técnicas de medición activa y pasiva. La medición pasiva se basa en la observación del tráfico de red sin realizar inyección de paquetes. Por el contrario, la medición activa se basa en la inyección de paquetes adicionales en la red para monitorizar el funcionamiento de ésta. SDN y OpenFlow ofrecen la posibilidad de implementar este tipo de monitorización sin ningún coste económico.

Con los precedentes que ofrecen SDN y OpenFlow, las ventajas que aportan estas tecnologías a un sistema de monitorización de red son las siguientes:

- Evitan la compra/instalación de equipos adicionales repercutiendo en un menor coste de administración de la red. SDN permite realizar actualizaciones en las redes sin requerir cambios importantes en el hardware. SDN realiza cambios y actualizaciones mediante software, lo que permite que los cambios sean implementados en los equipos de manera automática.

- Al ser OpenFlow un estándar de facto para redes SDN, el sistema es compatible con cualquier red que use dicho estándar,

independientemente de la marca del fabricante del hardware. Además, OpenFlow puede ser usado de momento en cualquier sistema operativo basado en Linux.

- Ofrecen el servicio del estado de los enlaces en tiempo real a otros módulos del controlador, lo que permite una eficiente toma de decisiones en el instante que se presenten errores o degradaciones de la calidad.

5.2. Trabajos Relacionados

El estado del arte en SDN, en general, y dentro de éste, el aspecto de monitorización en particular, es bastante escaso, dado el carácter novedoso de esta temática. Seguidamente se presentan los trabajos más representativos al respecto relativos al objeto de investigación planteado en este Trabajo Fin de Máster:

En [IEEE14] se presenta PayLess, una interfaz basada en OpenFlow para llevar a cabo la supervisión del estado de la red desatendiendo los detalles a bajo nivel. El mecanismo de recopilación de estadísticas se ha implementado como un servicio basado en consulta, donde las aplicaciones de red y el controlador tienen que consultar de manera periódica los conmutadores. La frecuencia de consulta determina el nivel de precisión de monitorización y sobrecarga de la red. Payless ofrece una API REST flexible para la recopilación de estadísticas de flujo a diferentes niveles de agregación, utilizando un algoritmo de recopilación de estadísticas que ofrece una información precisa y en tiempo real del estado de la red. Este trabajo plantea una disyuntiva entre la precisión de seguimiento, la puntualidad y la sobrecarga de la red.

MonSamp [MON14] estudia el flujo de muestreo para la vigilancia del tráfico de red que circula por las redes SDN, describiendo cómo se realiza la comunicación con el controlador SDN a través del uso de la API SDN. Dicha

vigilancia se realiza mediante la creación de un prototipo SDN denominado MonSamp. MonSamp consiste en un algoritmo que realiza duplicación del tráfico de red, enviándose a un monitor denominado collector & analyzer. La información recolectada por el algoritmo de MonSamp aumenta o disminuye el número de elementos en las tablas de encaminamiento en un dispositivo concreto. Este aumento o disminución en el número de elementos depende de la capacidad con la que se encuentren los enlaces o colectores, pues si éstos se encuentran llenos no se añaden más elementos. Las pruebas realizadas demuestran un funcionamiento correcto y, particularmente, viable.

[NOMS14] presenta una aplicación de código abierto para realizar la monitorización de métricas en flujos de red denominada OpenNetMon. Esta monitorización de métricas se basa en la obtención de parámetros como el retardo y la pérdida de paquetes en redes OpenFlow. OpenNetMon proporciona el control necesario para determinar la calidad de servicio de los parámetros que se encuentran en los nodos de una red determinada. Estos parámetros se van ajustando en función de la tasa de tráfico de red y de la sobrecarga de la CPU. OpenNetMon además se encarga de verificar el rendimiento obtenido en las mediciones de los parámetros citados.

En [JCN14] se expone el diseño de un esquema para medir de manera activa el rendimiento de los enlaces físicos. Esta medición se realiza mediante el uso de una función que controla el flujo de tráfico individual en redes OpenFlow. Para poder llevar a cabo dicha medición realizan un cálculo para poder cubrir de manera exhaustiva todos los enlaces definidos en el algoritmo de encaminamiento por parte del controlador OpenFlow, así como monitorizar todo el tráfico de red en forma de paquetes.

5.3. Objetivos

El diseño del framework pretende alcanzar los siguientes objetivos:

- Disminuir la carga en el controlador: El sistema debe minimizar el análisis de información por parte del controlador OpenFlow. Esto repercute favorablemente en la carga de procesamiento que se realiza sobre la CPU y la memoria. Al descender dicha carga computacional, también desciende la energía usada. Este ahorro energético repercute de manera directa en un ahorro económico.

- Disminuir la carga en el conmutador: El sistema debe minimizar el número de peticiones de información al conmutador para evitar la posible saturación de este dispositivo de red. Este aspecto garantiza que la información se está enviando de manera correcta a través de los nodos y enlaces que conforman la topología de red.

- Disminuir la carga en el enlace: Las medidas activas, es decir, aquellas que implican agregar tráfico de monitorización en los enlaces, deben minimizar el tráfico agregado.

- Rápida detección: Una detección rápida de problemas en la capa IP se traduce en un mejor funcionamiento de la red. Mientras más tardía sea dicha detección, mayor perjuicio ocasionará a las capas superiores.

5.4. Generalidades

En este apartado se señalan algunos aspectos básicos a tener en cuenta en relación al diseño y a la implementación del algoritmo con el que se realiza la monitorización de estadísticas en redes SDN usando el protocolo OpenFlow.

Como se ha comentado anteriormente, se pretende hacer un estudio estadístico del uso de los enlaces que compone una determinada topología de

red OpenFlow. Este estudio estadístico proporciona información sobre la velocidad de transmisión, la tasa de pérdida de datos y la tasa de retardo en los enlaces de una topología dada durante el proceso de envío de un fichero de vídeo.

La implementación del framework se ha realizado en Java [SUNJ]. Java permite tener un framework totalmente portable. Asimismo, cuenta con infinitud de librerías de clases bastante completas.

5.5. Estructura del Framework

El principal componente del framework es el controlador, que se encarga de monitorizar la red. La estructura del framework se ilustra en la Figura 5.1.

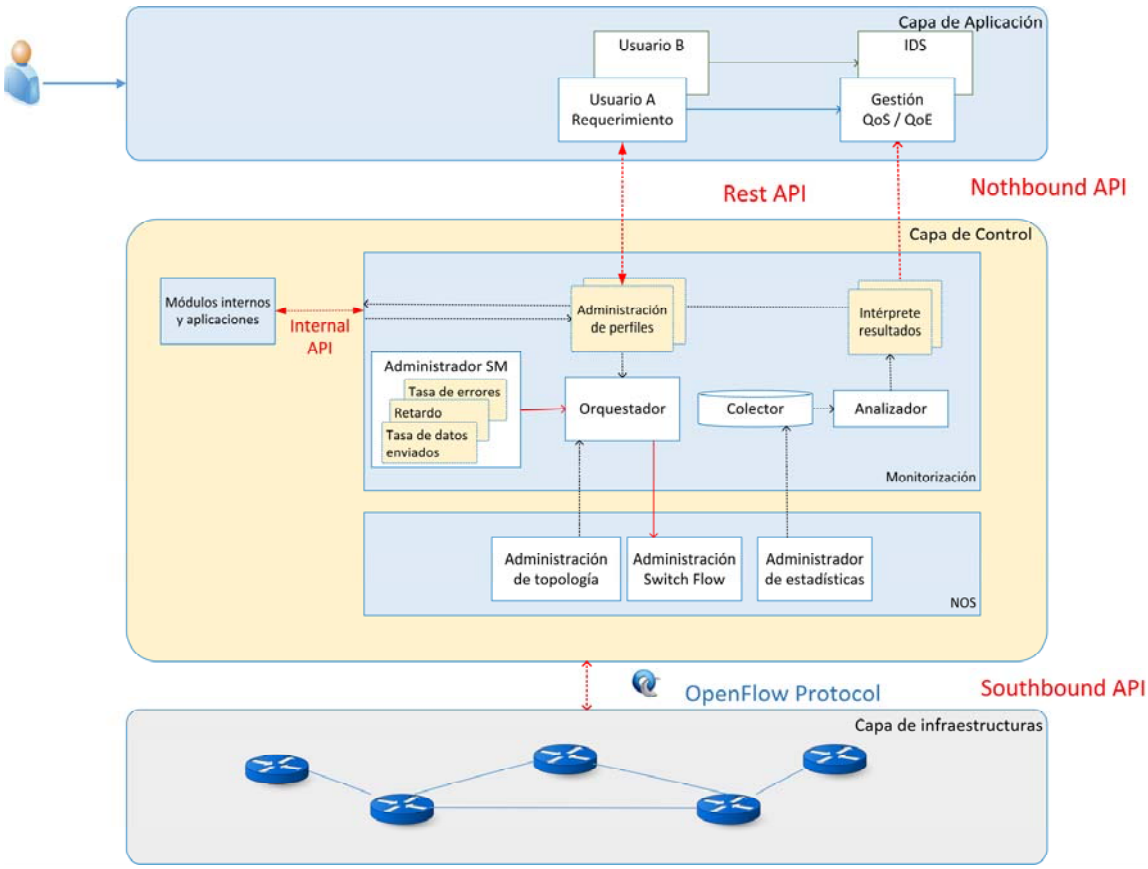

Figura 5.1: Estructura del Framework.

Los componentes de este framework se encuentran en tres capas:

- **Capa de Control**: Responsable de la toma de decisiones ante cualquier comportamiento de la red. Envía instrucciones a la capa de infraestructura. Dentro de la capa de control podemos encontrar los siguientes elementos:

    - Administración de Perfil: Módulo encargado de recibir las peticiones de las aplicaciones, así como de crear un perfil de monitorización para cada usuario registrado en la red.

    - Administración de Topología: Módulo encargado de identificar los dispositivos de red, sus capacidades y los enlaces presentes en la capa de infraestructura. Envía información a otro módulo perteneciente a la misma capa.

    - Administración SM: Contenedor que registra y almacena los algoritmos disponibles para la monitorización. Estos algoritmos pueden ser lanzados, añadidos, modificados y actualizados fácilmente.

    - Orquestador: Módulo encargado de la planificación y organización de los diferentes módulos de monitorización en función de los requisitos demandados por la Administración de Perfil.

    - Administración Switch Flow: Recibe instrucciones de los algoritmos de monitorización organizados por el Orquestador.

    - Administración de Estadísticas: Recibe información proporcionada por los mensajes OpenFlow y además extrae estadísticas de dichos mensajes. Esta información se envía al módulo Colector.

- Colector: Este módulo se encarga de organizar la información proporcionada por el administrador de estadísticas para almacenar aquellas que le interese.

- Analizador: Establece las métricas de actuación basadas en la información almacenada por el Colector. Ejemplo: alertas de acceso detectadas en el cortafuegos de la red.

- Intérprete de Resultados: Los resultados del proceso de monitorización se distribuyen a los usuarios ubicados en la capa de aplicación o capa de control.

- **Capa de Aplicación:** Encargada de las implementaciones de las políticas y de las aplicaciones de alto nivel. Esta capa contiene el siguiente módulo:

  - Requisitos de Usuario: Cuando la aplicación necesita información acerca de la red, crea un nuevo perfil de monitorización. Este perfil viene especificado con un identificador de usuario, tipo de métrica y nivel de precisión requerida. Esta información se envía a través de la capa de control.

  - Capa de Infraestructura: Contiene los diferentes elementos físicos.

5.6. Algoritmo de Monitorización

El algoritmo diseñado para la recolección de estadísticas va pasando por diversos procesos y estados, representados en la Figura 5.2.

a. Lectura de los dispositivos conectados en la topología.

b. Comprobación del número de conmutadores conectados. En el caso de que no hubiera dispositivos conectados, el procedimiento finaliza

mostrando el correspondiente mensaje de error.

c. Activación de un temporizador dinámico en función del número de dispositivos leídos.

d. Obtención de estadísticas a nivel de puerto en cada conmutador.

e. Iteración a través de los diferentes puertos que contienen los dispositivos de red conectados a la topología.

f. Visualización de estadísticas correspondientes a: número de puerto, número de paquetes y transmisiones.

g. Añadir la información obtenida en f a una nueva iteración.

h. Iterar por todos los enlaces que conectan los diferentes conmutadores encontrados en la topología.

i. Cálculo de estadísticas de los enlaces obtenidos en g para los tiempos T y T-1.

j. Cálculo de la velocidad de transmisión entre los nodos origen y destino.

k. Cálculo de la tasa de paquetes perdidos entre los nodos origen y destino.

l. Cálculo del retardo observado entre los nodos origen y destino.

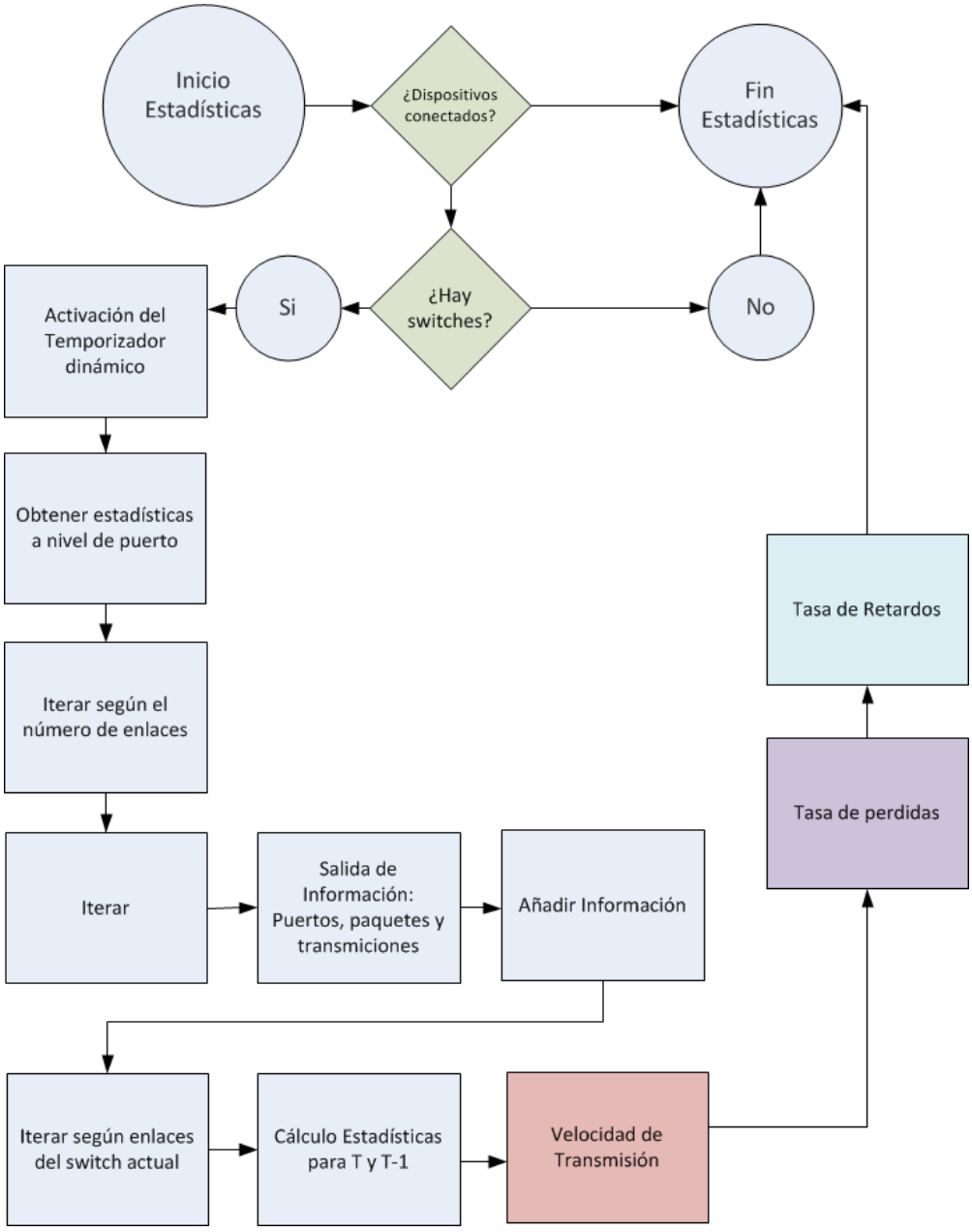

Figura 5.2: Esquema del Algoritmo de Monitorización.

5.6.1. Procedimiento para el Cálculo de Estadísticas

Las topologías de red usadas para llevar a cabo las simulaciones para el cálculo de estadísticas se encuentran representadas gráficamente por un grafo completo. Este grafo se encuentra determinado por *G(N, A)*, donde *N* representa el número de dispositivos conectados y *A* los arcos o enlaces que conectan esos dispositivos. Los enlaces serán representados en el algoritmo como *arc (i, j)* $\in A$, donde se simboliza la existencia de un enlace entre los nodos *i* y *j*. Cada enlace presente en la topología tendrá asociado un valor de peso o un coste de enlace. El coste de un enlace *arc (i, j)* vendrá determinado por c_{ij}. La conexión de enlaces entre los dispositivos usados en las topologías es directa y no presentan ciclos. La conexión entre el controlador y cada uno de los conmutadores es individual y directa.

El procedimiento de cálculo de estadísticas se realiza mediante la modificación de los módulos Orquestador (*Orchestrator*) y el Administrador SM (*SM Manager*). Estos módulos se encuentran representados en la Figura 5.3.

Los módulos Orquestador y Administrador SM son los responsables de organizar los algoritmos utilizados para gestionar el comportamiento de la red. Estos algoritmos se encuentran implementados de tal manera que permiten la modificación o eliminación de una manera más sencilla sin interferir entre sí. El Administrador SM se encarga de registrar y organizar los módulos utilizados para realizar la monitorización de la red. El Orquestador lleva a cabo la autorización de la ejecución de los módulos SM. Además, también tiene que realizar el control de las peticiones de solicitudes que se produzcan en los conmutadores.

La tarea de gestionar los algoritmos presentes en el Administrador SM es sumamente compleja, debido a que la supervisión de la red puede afectar al rendimiento de la misma. Ello requiere de un módulo Orquestador que pueda

controlar de manera óptima la carga de tareas de supervisión. Este control depende del número de solicitudes en función del tamaño que posea la red. Este tamaño será determinante en las simulaciones y en la carga de capacidad que sufrirá el controlador. La lógica de esta técnica se explica en el Algoritmo 1. El usuario asigna un período de tiempo mínimo t_{min} y una capacidad de carga del controlador α. Si α = 1, se indica al controlador que debe soportar un alto nivel de carga en su capacidad. Por el contrario, si α = 0, la capacidad de soporte del controlador se establece a un valor limitado. Por otra parte, el tamaño de la red se estima en función del número de dispositivos *N* y el número de enlaces *A* presentes en la topología leída. *N* y *A* serán variables usadas para llevar a cabo el cálculo del equilibrado en los tiempos de carga t_{orc}.

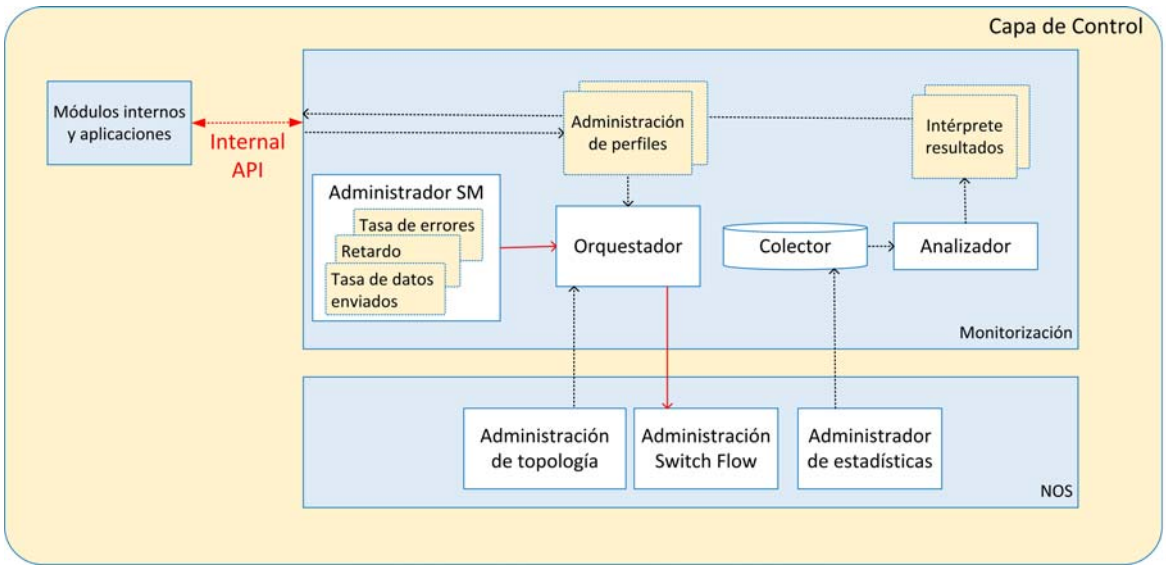

Figura 5.3: Orquestador y Administrador SM.

5.6.2. Algoritmos de Cálculo

La implementación del sistema de monitorización se traduce en la implementación de diferentes algoritmos sobre un grafo *G (N, A)*.

El primero de los algoritmos se denomina *Función Orquestador*. Esta función se encarga de organizar los algoritmos usados para establecer el comportamiento de una red *G(N, A)*. Dicha función se apoya en el elemento *Administrador SM* para añadir, modificar o eliminar los algoritmos encargados de la monitorización de la red SDN. Además, esta función también realiza el control de rendimiento para llevar a cabo la monitorización y la comprobación de la carga de la red. Estas tareas se realizan mediante el cálculo del tiempo optimizado de monitorización que vendrá determinado por el tamaño del grafo entrante (topología).

Así, el Algoritmo 1 realiza la medición del nivel de carga observado en una red *G(N, A)*.

Algoritmo 1: Función Orquestador

Entrada: grafo *G(N,A)*
periodo mínimo de monitorización $\boldsymbol{t_{min}}$
capacidad controlador $0 < a \leq 1$

Resultado: tiempo optimizado de monitorización $\boldsymbol{t_{orc}}$

1. **procedure** OrchFunction
2. calcular tiempo optimizado en función del grafo
3. **if** $a = 1$ **do**
4. $\quad \boldsymbol{t_{orc} = t_{min}}$
5. **else if** $0 < a < 1$ **do**
6. $\quad$ **if** $A \leq N$ **do**
7. $\quad\quad \boldsymbol{t_{max} = N . t_{min}}$
8. $\quad$ **else**
9. $\quad\quad \boldsymbol{t_{max} = \left(N + \frac{A}{N}\right) . t_{min}}$
   $\quad$ **end if**
10. $\boldsymbol{t_{orc} = (1 - \alpha) . t_{max}}$
    end if
11. **end procedure**

El Algoritmo 2 lleva a cabo la medición de la velocidad de transmisión en los diferentes enlaces.

La velocidad de transmisión hace uso de los mensajes de tipo *OFPT_STATS_REQUEST*.

Estos mensajes se utilizan para realizar solicitudes a los puertos de los nodos que constituyen la red sobre la que se está realizando el análisis. Mediante los parámetros *nodo_src* y *port_src* se realiza la recolección de información almacenada en los nodos y en cada uno de los puertos que éstos posean.

Una vez recolectada dicha información, se procede a realizar el cálculo de la velocidad de transmisión. Este cálculo se encuentra basado en la realización del cálculo de la diferencia de bytes observados en los períodos *t*. Esta diferencia es calculada para cada uno de los enlaces presentes en la topología.

Algoritmo 2: Velocidad de transmisión (Administrador SM)

Entrada: red grafo *G(N,A)*
Tiempo optimizado de monitorización $\boldsymbol{t_{orc}}$

Resultado: send data rate $\boldsymbol{dr_{ij}}$ **para cada** $\boldsymbol{arc(i,j) \in A}$

1. **procedure** SendDataRate
2. Iniciar timer *k* con periodo $\boldsymbol{t_{orc}}$
3. **foreach** periodo $\boldsymbol{k = 0, 1, 2, 3 \ldots . \in t_{orc}}$ **do**
4. **foreach** $\boldsymbol{arc(i,j)}$ **do**
5. Leer bytes enviados $\boldsymbol{s_i}$ desde el puerto del enlace de inicio con $\boldsymbol{s_i^k}$ = *tx_bytes* in *ofp_port_stats (src node, src port(i))*
6. **if** k > 0 **do**
7. Calcular data rate del enlace

$$\boldsymbol{dr_{i,j} = \frac{s_i^k - s_i^{k-1}}{t_{orc}}}$$

end if
end for
8. **end procedure** SendDataRate

El Algoritmo 3 es el responsable de realizar la medición de la tasa de paquetes perdidos. El cálculo que realiza es similar al realizado en el Algoritmo 2. Así, realiza el cálculo de la diferencia de bytes enviados desde nodo y puerto origen, con los bytes recibidos en el nodo y puerto destino en un período *t*.

Algoritmo 3: Tasa Paquetes Perdidos (Administrador SM)

Entradas: red grafo *G(N,A)*
Tiempo optimizado de monitorización $\boldsymbol{t_{orc}}$

Resultado: packet loss rate $\boldsymbol{lr_{ij}\ para\ cada\ arc(i,j) \in A}$

1. **procedure** PacketLossRate
2. Iniciar timer *k* con periodo $\boldsymbol{t_{orc}}$
3. **foreach** periodo $\boldsymbol{k = 0, 1, 2, 3 \ldots . \in t_{orc}}$ **do**
4. **foreach** $\boldsymbol{arc(i,j)}$ **do**
5. Leer bytes enviados $\boldsymbol{s_i}$ del puerto del enlace inicio con $\boldsymbol{s_i^k}$ = *tx_bytes* in *ofp_port_stats(src node, src port(i))*
6. Leer bytes recibidos $\mathbf{r_j}$ del enlace final con $\boldsymbol{r_j^k}$= *rx_bytes* in *ofp_port_stats (dst node, dst port(j))*
7. **if** k > 0 **do**
8. Calcular packet loss rate del enlace

$$\boldsymbol{lr_{i,j} = \frac{(s_i^k - s_i^{k-1}) - (r_j^k - r_j^{k-1})}{t_{orc}}}$$

end if
end for
end if
9. **end procedure** PacketLossRate

El Algoritmo 4 determina el retardo. El controlador empieza leyendo el registro del tiempo actual del sistema. Este registro de tiempo se encapsula en un paquete de prueba (Pp). Se procede a enviar dicho paquete desde el conmutador origen *i* al conmutador destino *j*. Una vez llegado el paquete al conmutador destino *j*, el conmutador lo identifica, lo encapsula y lo envía de vuelta al controlador. El controlador procede a realizar la identificación del paquete para obtener los datos referentes al tiempo inicial y al tiempo de llegada. Una vez obtenidos dichos tiempos, se realiza una comparación con ellos para obtener el registro del tiempo real de llegada. El retardo se calcula

mediante la diferencia entre los registros de tiempo para el inicio del envío y el tiempo de llegada del paquete.

Algoritmo 4: Retardo (Administrador SM)

Entrada: red grafo *G(N,A)*
Tiempo optimizado de monitorización $\boldsymbol{t_{orc}}$
Resultado: delay $\boldsymbol{d_{ij}}$ **para cada** $\boldsymbol{arc(i,j) \in A}$

1. **procedure** Delay
2. **procedure** SentProbePacket
3. Iniciar timer *k* con periodo $\boldsymbol{t_{orc}}$
4. **foreach** periodo $\boldsymbol{k = 0, 1, 2, 3 \ldots . \in t_{orc}}$ **do**
5. **foreach** $\boldsymbol{arc(i,j)}$ **hacer**
6. Leer timerstamp actual $\boldsymbol{t_{st}}$ del controlador
$\boldsymbol{t_{st}}$=*Date.getTime()*
7. Encapsular $\boldsymbol{t_{st}}$ con Probe packet $\boldsymbol{Pp}$
$\boldsymbol{Pp_{i\rightarrow j}}$ = new packetOutMessage*()*
$\boldsymbol{Pp_{i\rightarrow j}}$*.setProtocol(253)*
$\boldsymbol{Pp_{i\rightarrow j}}$.setData($\boldsymbol{t_{st}}$*)*
8. Enviar Probe packet al switch fuente
switch(i).write($\boldsymbol{Pp_{i\rightarrow j}}$*)*
end for
end for
9. **end procedure** SentProbePacket

10. **procedure** ReceiveProbePacket
11. Verificar que el paquete mensaje entrante es parte de probe packet
if (packetInMessage $\boldsymbol{pkt_{in}}$ is a Probe packet ($\boldsymbol{Pp_{i\rightarrow j}}$)) **do**
12. Read source, destination and timestamp
$\boldsymbol{t_{st} = pkt_{in}}$ *.getData()*
$\boldsymbol{i = pkt_{in}}$ *.getSource()*
$\boldsymbol{j = pkt_{in}}$ *.getDestination()*
end if
13. Leer timestamp actual del controlador
$\boldsymbol{t_{ctr}}$=*Date.getTime()*
14. Calcular el valor de delay del enlace *arc(i,j)*
$\boldsymbol{d_{ij}}$=$\boldsymbol{t_{ctr} - t_{st}}$
15. **end procedure** ReceiveProbePacket
16. **end procedure** Delay

6. IMPLEMENTACIÓN

6.1. Entorno

Se ha utilizado el simulador Mininet. En dicho simulador se ha procedido a la carga de las diferentes topologías que se han analizado. Las topologías han sido implementadas mediante un lenguaje de programación de alto nivel denominado Python [PYT]. Cada topología creada ha sido ajustada con unos parámetros y valores por defecto para la velocidad de transmisión, tasa de paquetes perdidos y tasa de retardo.

Una vez implementadas las topologías y ajustados sus parámetros de manera correcta, se procedió a cargarlas en el simulador, mediante el uso de la herramienta terminal y el siguiente comando:

Usuario1@mininet: ~$ sudo ./cargarTopologiaEjemplo1.py

Cargada una topología determinada dentro del simulador de Mininet, se procede a realizar la ejecución de los algoritmos implementados dentro del framework de monitorización. La ejecución de estos algoritmos durante el envío de un archivo de vídeo permite obtener los valores estadísticos. Los algoritmos recolectaran información a medida que el vídeo va pasando por cada uno de los enlaces que conforman el camino.

Una vez finalizada la transmisión de vídeo, todos los datos recolectados por el algoritmo de monitorización se almacenan en ficheros de texto plano. Estos ficheros de texto contienen información referente a nodos origen, nodos destino, puertos de salida, puertos de destino, velocidades de transmisión, tasas de paquetes perdidos y tasas de retardo.

Después de haber obtenido todos los archivos de texto con los datos recogidos, se procesa mediante una aplicación desarrollada con el programa de

Análisis Matemático Matlab [MAT], generándose las gráficas para las métricas analizadas. Asimismo, se generan una serie de tablas con los valores obtenidos.

6.2. VideoLAN

VideoLan (VLC) es un software desarrollado en Francia para realizar transmisión de vídeo. VLC se encuentra desarrollado dentro de las políticas y requerimientos establecidos por GNU.

VLC es un reproductor multimedia y con soporte para los principales sistemas operativos del mercado (Windows, MAC OS X y Linux). Su descarga es gratuita [VLC]. Mediante el uso de este reproductor se hace el envío del vídeo para la simulación del tráfico de red en las topologías analizadas. El motivo de la elección de este reproductor como método de transferencia de vídeo reside en su versatilidad y compatibilidad para cualquier tipo de formato de vídeo. VLC es capaz de reproducir formatos como: AVI, MPEG-4, FLV o MKV.

6.3. Herramientas

Para llevar a cabo las simulaciones de las implementaciones realizadas en el framework de monitorización se han hecho uso de diferentes herramientas. Las simulaciones realizadas mediante el simulador de Mininet han sido instanciadas usando máquinas virtuales. En concreto, se ha dispuesto de una máquina virtual Linux. Esta máquina virtual ha sido necesaria ejecutarla con el programa Oracle Virtual Box [OVB]. La implementación se ha basado en un proyecto denominado Floodlight [PFL] donde se ha insertado dicho framework mediante un entorno de programación Eclipse [ECL]. Todos los detalles de las herramientas usadas para realizar las simulaciones se recogen en las Tablas 6.1 y 6.2.

Hardware	Software
MacBook Pro Core i5" 2.4 Mid-2010 15” 8GB RAM DDR3 2x240 GB SSD	MAC OS X Lion Build 10.7.5 Oracle Virtual Box 4.3.14

Tabla 6.1: Componentes Hardware y Software

Hardware VM	Software VM
1 procesador 1024 RAM 12MB memoria gráfica	Xubuntu 13.04 Simulador SDN Mininet 2.1 Eclipse Kepler Floodlight 0.90. VLC 2.0.8

Tabla 6.2: Componentes Hardware y Software Máquina Virtual

Finalmente, señalar que el vídeo se envía mediante el uso del protocolo RTP/UDP. Sus características se recogen en la Tabla 6.3.

Nombre	**highway_cif**
Codificación	MPEG ½,
Número de Marcos	2000 marcos
Duración	80 s
Tamaño	2.97 MB

Tabla 6.3: Propiedades de la información enviada

Cada simulación se compone de veinte repeticiones completas del algoritmo. Posteriormente, se obtienen los valores promedios de las métricas consideradas.

6.4. Mininet

Mininet es un emulador de redes SDN. Dicho emulador ejecuta una colección de *hosts*, *switches*, *routers* y enlaces en un solo núcleo de Linux. Usando la tecnología de virtualización se puede construir todo un sistema que funcione

como una red completa. Este sistema se ejecuta sobre el mismo núcleo, sistema, y código de usuario.

Mininet se comporta como una máquina real; se puede acceder a una red Mininet a través de tecnología SSH y ejecutar programas arbitrarios (incluidos todos los programas que contenta Linux de manera interna) en dicha red.

Los programas ejecutados mediante el simulador Mininet pueden hacer uso de envío de paquetes justo como se hace a través de una interfaz real Ethernet, mediante el uso de un router o middlebox [MBX]. Cuando dos programas como un cliente *iperf* y un servidor se comunican a través de Mininet la medición del rendimiento debe coincidir con el de dos máquinas nativas.

Los host virtuales, switches, enlaces y controladores de Mininet se crean mediante software en un dispositivo físico. El comportamiento de estos dispositivos creados virtualmente es similar al de sus idénticos en hardware. Mininet permite la creación de redes que se asemejen a redes físicas, y una red física puede ser creada para asemejarse a una red virtual creada en Mininet. Esta semejanza permite la ejecución de código binario y aplicaciones en ambas. Mininet cuenta con una serie de comandos los cuales permiten la ejecución de aplicaciones dentro de Mininet [MWT], como conocer la información IP de los enlaces que componen una red simulada dentro de Mininet.

Actualmente, Mininet es un proyecto de código abierto, cuyos archivos fuentes pueden ser descargados para su visualización o incluso modificación [MGB].

6.4.1. ¿Por qué usar Mininet?

Los motivos que han llevado a la elección de Mininet como simulador por defecto para el uso de redes SDN son los siguientes:

- Rapidez: poner en marcha una red simple toma escasos minutos, con lo que gestionar la depuración de una red puede ser una tarea sumamente sencilla.

- Mininet soporta la creación de topologías totalmente personalizadas. Se pueden crear desde redes simples que contengan un solo controlador a redes más complejas que se asemejen a centros de procesamiento de datos.

- Se pueden ejecutar programas reales, ya que cualquier programa que pueda ser ejecutado en Linux, también está soportado en redes que hayan sido creadas mediante Mininet.

- Los *switches* pertenecientes al simulador de Mininet pueden ser programados mediante el protocolo OpenFlow. Los diseños personalizados de redes SDN que se ejecutan en Mininet pueden ser transferidos fácilmente a los *switches* físicos de OpenFlow.

- Este simulador se puede ejecutar en ordenadores portátiles, ordenadores de sobremesa, servidores, máquinas virtuales, máquinas Linux o incluso también en sistemas operativos que se encuentren alojados en la nube. Una de las mayores virtudes del simulador Mininet es, pues, su versatilidad en la ejecución en diferentes plataformas.

- Mininet permite la creación y ejecución de experimentos mediante la creación de *scripts* simples o complejos en lenguaje Python.

6.4.2. Limitaciones

Aunque simulador Mininet destaca por su versatilidad, eficiencia y sencillo uso, también presenta algunas limitaciones que es interesante detallar.

Mininet hace uso de un solo núcleo Linux para todos los host virtuales, lo que significa que no se puede ejecutar software que dependa de BSD, Windows u otros sistemas operativos.

Para llevar a cabo personalizaciones ajustadas a los requerimientos de un proyecto concreto es necesario realizar la escritura del código del controlador OpenFlow.

Por defecto, las redes que hacen uso de Mininet se encuentran aisladas de redes LAN e Internet, aunque se puede hacer uso de tecnologías NAT para interconectar redes Mininet con redes LAN.

7. EXPERIMENTACIÓN REALIZADA

Para comprobar el rendimiento de los algoritmos en el framework se han llevado a cabo una serie de pruebas con el objetivo de obtener los valores de las métricas analizadas: velocidad de transmisión de datos, tasa de paquetes perdidos y tasa de retardo.

7.1. Topologías Analizadas

Para llevar a cabo la comprobación de rendimiento de los algoritmos implementados en el Framework de monitorización se han analizado diferentes topologías.

La topología básica (véase Figura 7.1) se compone de tres host virtuales (*H1, H2* y *H3*), tres switches (*S1, S2* y *S3*) y el controlador Floodlight que se encarga de la tarea de obtención de estadísticas. La conexión de los switches se encuentra constituida por una conexión en serie, donde el host 1 (*H1*) se encuentra conectado al switch 1 (*S1*), el switch *S1* se encuentra conectado al switch 2 (*S2*), el host 2 (H2) se encuentra conectado a *S2*, *S2* se encuentra conectado al switch 3 (*S3*) y éste, a su vez, al host 3 (*H3*).

El valor de retardo establecido para cada uno de los enlaces que compone la unión de los switches es de 10 milisegundos y el porcentaje de pérdida se encuentra establecido en un 5% (para más detalles de configuración véase la Tabla 7.1).

La comprobación del rendimiento de esta topología se realizó mediante el envío de vídeo desde el host virtual *H1* (servidor) al host virtual *H3* (cliente). El camino que recorrió el vídeo desde *H1* hasta *H3* se encuentra compuesto por los enlaces de red *L1* y *L2 (*línea de color verde*)*.

La topología en serie de seis switches (véase Figura 7.2). Estos switches están determinados por S1, S2, S3, S4, S5 y S6. Cada switch se encuentra conectado con su correspondiente host virtual. El número de hosts virtuales conectados asciende a seis (H1, H2, H3, H4, H5 y H6). A su vez, el controlador Floodlight se encuentra conectado a cada uno de los switches. La conexión entre los switches se realiza mediante cinco enlaces (L1, L2, L3, L4 y L5) conectados en serie. Cada uno de estos enlaces fue configurado con parámetros de velocidad de transmisión, tasa de pérdidas de paquetes y tiempo de retardo determinados.

El envío del vídeo se hace desde el host virtual H1 (servidor) al host virtual H6 (cliente). Dicho vídeo va recorriendo cada uno de los enlaces que conectan H1 con H6. Los enlaces presentan diferencias de configuración entre sí para los parámetros de velocidad de transmisión, tasa de paquetes perdidos y retardo (Tabla 7.2).

La topología en forma de estrella, donde cinco de los seis switches totales (S1, S2, S3, S4, y S5) se conectan de manera directa a un switch central (S6). Cada switch se encuentra conectado de forma directa con su host virtual correspondiente, excepto el switch S6 que, al tratarse de un switch central, carece de host virtual ya que es el switch intermedio que conecta a los demás switches entre sí. A su vez, el controlador Floodlight se encuentra conectado de manera directa con cada uno de los switches que componen la topología en forma de estrella.

El envío de vídeo se hace desde el host virtual H1 (servidor) al host virtual H5 (cliente). El camino por el que transcurre el vídeo se encuentra especificado en la Figura 7.3 en forma de flechas de color verde. El camino se encuentra compuesto por los enlaces L1 y L5.

La Tabla 7.3 presenta los valores establecidos por defecto para cada uno de los enlaces que conforman esta topología.

La última de las topologías presentadas es una topología en forma de centro de procesamiento de datos o "data center". Este tipo de topologías, pero a mayor escala, son usadas en entornos reales del sector empresarial. En concreto, esta topología está formada por 7 switches (S1, S2, S3, S4, S5, S6 y S7), conectados en una formación 1-2. Cada switch se encuentra conectado a otros 2 switches. Por ejemplo, S2 y S3 se encuentran conectados a S1. A su vez, el controlador Floodlight se conecta a cada uno de los switches de manera individual. Al tratarse de una conexión que intenta simular un "Data Center", sólo los switches S4, S5, S6 y S7 poseerán host virtuales conectados directamente a ellos. En concreto, los host virtuales creados para esta topología son cuatro: H1, H2, H3 y H4. Para lograr una simulación realista se ha creado esta topología usando diferentes tipos de switches:

- El switch *S1* representa un switch núcleo (*core*) que interconecta los dispositivos de la capa de distribución.

- Los switches *S2* y *S3* representan a switches agregación (*aggreation*). Este tipo de switches se usan para agregar políticas de conectividad entre capas.

- Los switches *S4*, *S5*, *S6* y *S7* están representados por switches de tipo borde (*edge*), encargados de realizar la conexión con los equipos finales.

El vídeo se envía desde el host virtual H1 (servidor) al host virtual H3 (cliente). El camino establecido se compone de los enlaces L3, L1, L2 y L5, a través de los switches S4 (switch origen), S2, S1, S3 y, finalmente, S6 (switch destino). Este camino se encuentra representado en la Figura 7.4 mediante las flechas de color verde.

Los parámetros referentes a velocidad de transmisión, tasa de pérdida de paquetes y tasa de retardo se encuentran establecidos con los valores de la Tabla 7.4.

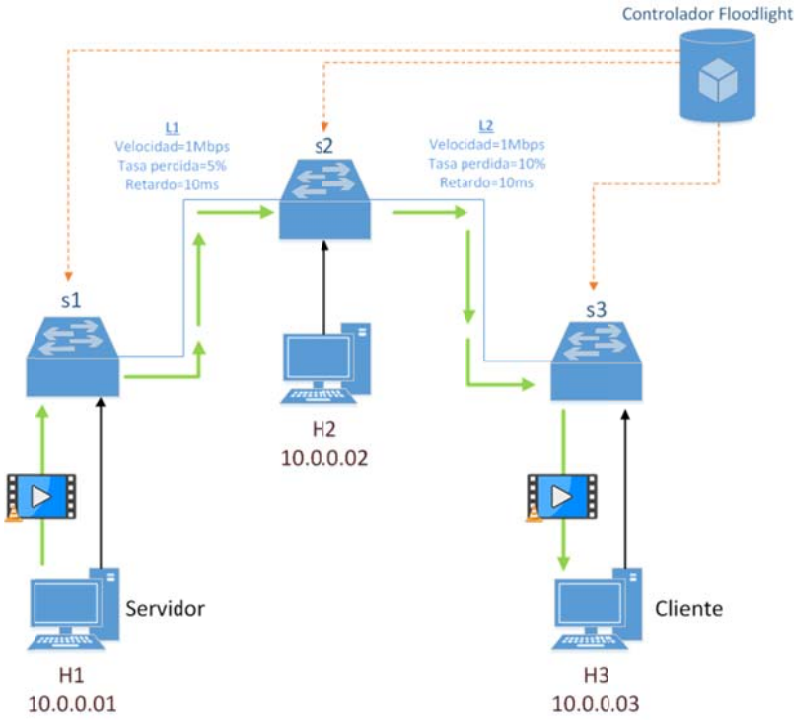

Figura 7.1: Topología básica.

Enlace	Velocidad Transmisión Datos	Tasa Paquetes Perdidos (loss rate)	Retardo (delay)
L1	1 Mbps	5%	10 ms
L2	1 Mbps	10%	10 ms

Tabla 7.1: Parámetros de configuración Topología básica

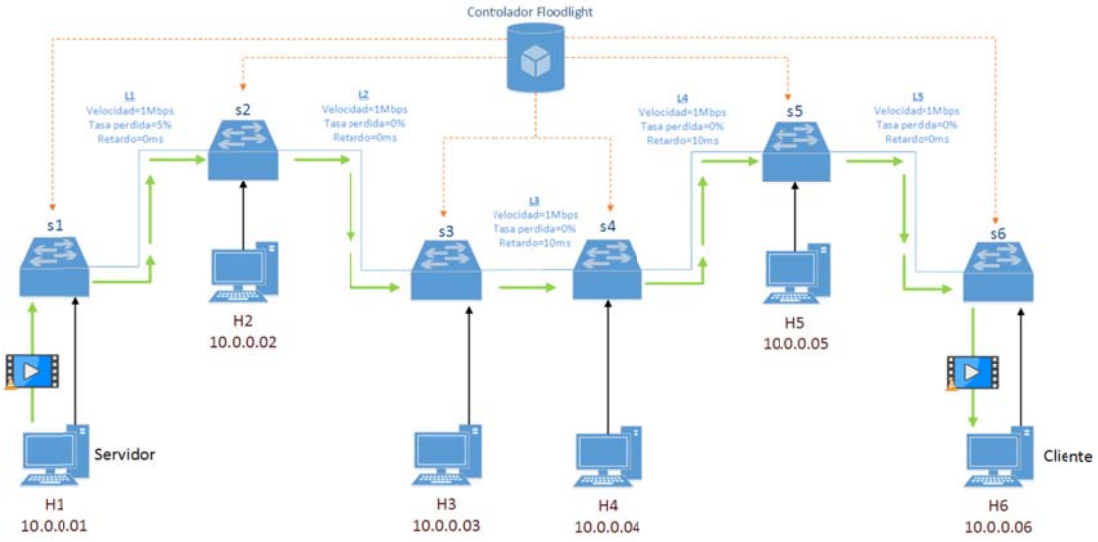

Figura 7.2: Topología en Serie.

Enlace	Velocidad Transmisión Datos	Tasa Paquetes Perdidos (loss rate)	Retardo (delay)
L1	1 Mbps	5%	0 ms
L2	1 Mbps	0%	0 ms
L3	1 Mbps	0%	10 ms
L4	1 Mbps	10%	10 ms
L5	1 Mbps	0%	0 ms

Tabla 7.2: Parámetros de configuración Topología en Serie

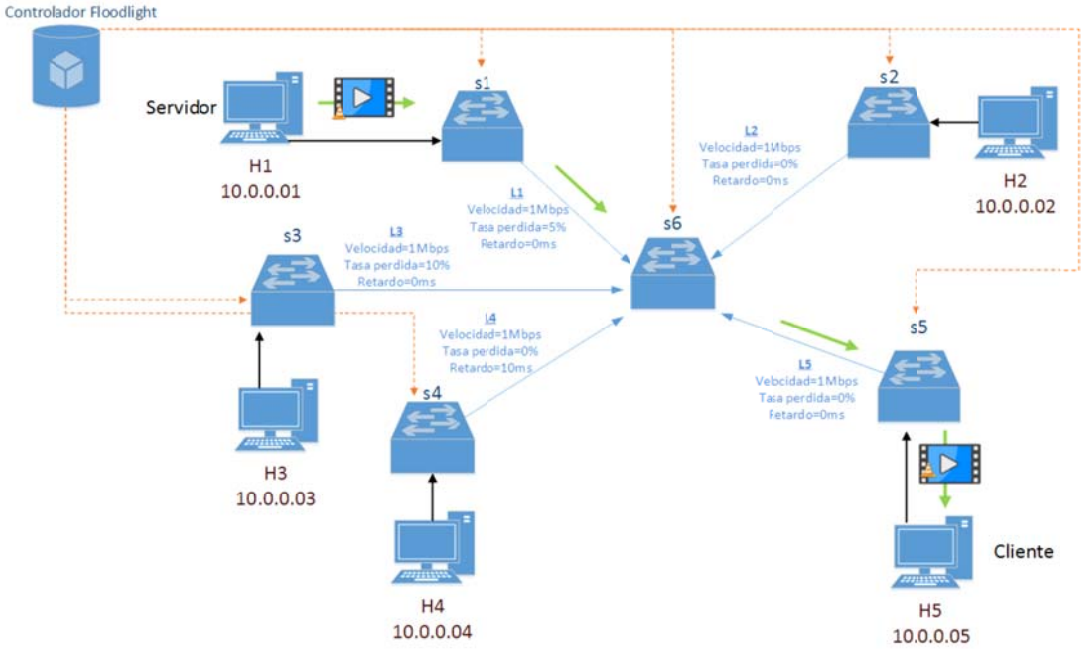

Figura 7.3: Topología en Estrella.

Enlace	Velocidad Enlace	Tasa de paquetes perdidos	Retardo
L1	1 Mbps	5%	0 ms
L2	1 Mbps	0%	0 ms
L3	1 Mbps	10%	0 ms
L4	1 Mbps	0%	10 ms
L5	1 Mbps	0%	0 ms

Tabla 7.3: Parámetros de configuración Topología en Estrella

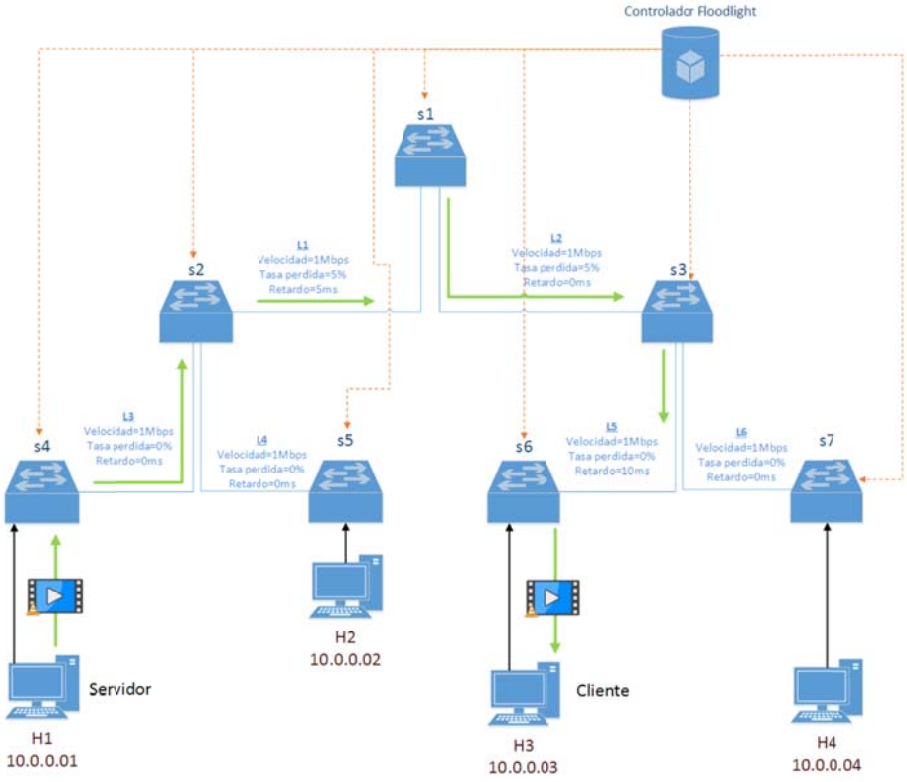

Figura 7.4: Topología Data Center.

Enlace	Velocidad Enlace	Tasa de paquetes perdidos	Retardo
L1	1 Mbps	5%	5 ms
L2	1 Mbps	5%	0 ms
L3	1 Mbps	0%	0 ms
L4	1 Mbps	0%	0 ms
L5	1 Mbps	0%	10 ms
L6	1 Mbps	0%	0 ms

Tabla 7.4: Parámetros de configuración Topología en Data Center

7.2. Métricas

7.2.1. Velocidad de transmisión datos

La Figura 7.5 muestra los valores de velocidad de transmisión en los enlaces L1 y L2. Mientras el vídeo se está enviando, se produce tráfico de red. Una vez finalizada la transmisión del vídeo, los datos descienden a valores cercanos al 0.

Este hecho, se produce a partir de la solicitud número 400, solicitud en la cual se finaliza por completo el envío del vídeo desde el host servidor al host cliente.

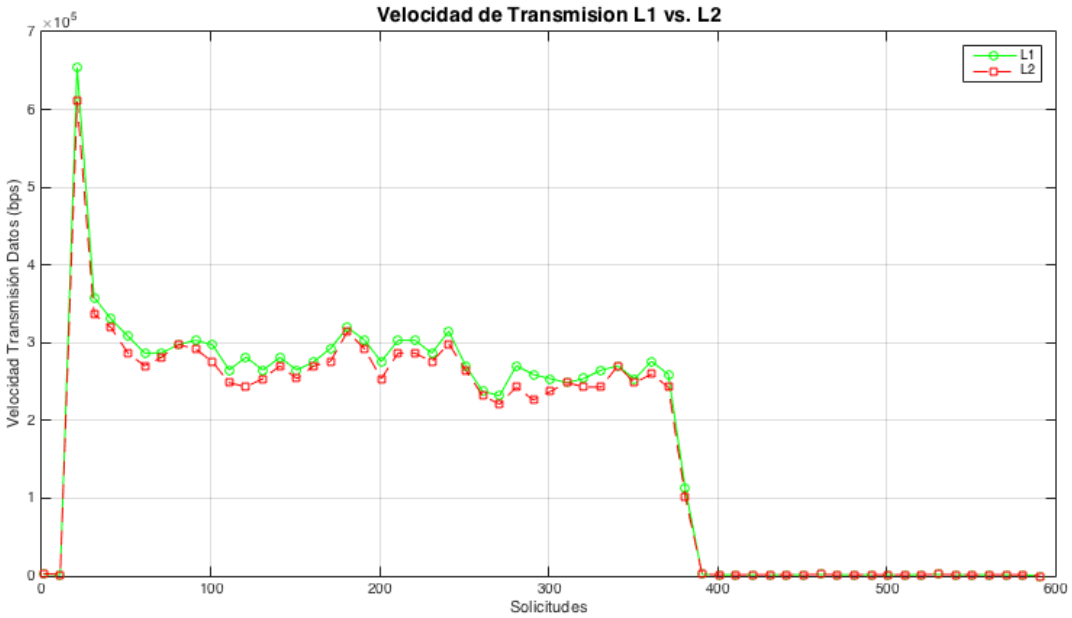

Figura 7.5: Velocidades de Transmisión Datos de L1 vs L2

Los datos obtenidos tanto para L1 como L2 son bastantes similares. Este hecho se debe a que los dos enlaces han sido configurados con la misma velocidad de transmisión de datos. La similitud observada en la gráfica de la Figura 7.5 prueba el buen funcionamiento del algoritmo implementado y que la velocidad de transmisión a través de estos enlaces ha sido satisfactoria. L1 es representado en la gráfica por la línea de color verde y L2 es representado por la línea de color rojo. Las velocidades de transmisión observadas en los enlaces participantes en el envío de vídeo son prácticamente similares desde el inicio hasta el fin.

La gráfica de la Figura 7.6 presenta los valores de transmisión de datos para los enlaces L1, L2, L3, L4 y L5. Cada uno de los enlaces se representa con un color diferente. L1 es representado con la línea de color verde, L2 con la línea de color rojo, L3 con la línea discontinua de puntos de color azul, L4 con la línea discontinua de rayas de color marrón, y L5 con la línea de color amarrillo.

Para una mejor comprensión de los datos representados en la gráfica de la Figura 7.6, se ha optado por mostrar un rango concreto de solicitudes (eje X). Los datos recogidos para la velocidad de transmisión de datos abarcan desde la solicitud número 50 hasta la solicitud número 380 aproximadamente.

Como se puede observar, los conjuntos de datos presentados para cada uno de los enlaces presentan una gran similitud entre sí. Esta similitud se debe a que todos los enlaces han sido configurados con la misma velocidad de transmisión de datos, es decir, 1 Mbps.

La gráfica demuestra que la velocidad de transmisión de datos se mantuvo en valores correctos cuando el vídeo fue transcurriendo por cada uno de los enlaces.

Debido a que la transferencia del vídeo pasa por todos los enlaces presentes en esta topología en serie, todos los enlaces presentan tráfico de red, es decir, el vídeo se va transmitiendo por cada uno de los enlaces participantes de manera correcta. Conforme el vídeo va transcurriendo por la topología y a su vez va finalizando la transferencia de vídeo, la velocidad de transmisión de datos va disminuyendo también. Una vez transmitido el vídeo desde el host servidor al host cliente, la velocidad de transmisión de datos decrece a 0. Este hecho se produce a partir de la solicitud número 380 aproximadamente.

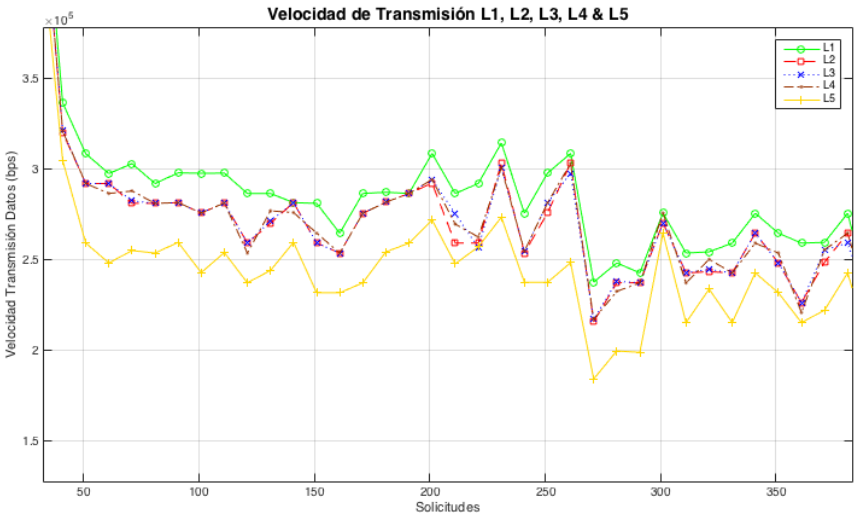

Figura 76: Velocidades de Transmisión Datos de L1, L2, L3, L4 y L5.

La Figura 7.7 muestra el parámetro velocidad de transmisión en los conjuntos de datos de los enlaces L1, L2, L3, L4 y L5. Todos los enlaces han sido configurados con la misma velocidad de transmisión de datos, es decir, 1 Mbps.

Los únicos enlaces participantes de esta topología en la trasmisión de vídeo, son L1 y L5. Se puede observar como estos dos enlaces son los únicos que presentan velocidad de transmisión de datos. La velocidad de transmisión observada en estos dos conjuntos de datos se presenta con una gran similitud entre ellos. Esta similitud se presenta desde el inicio de la transferencia del vídeo (solicitud número 0) hasta el final de la misma en la solicitud número 400 aproximadamente. Se observa además cómo el inicio de la transferencia coincide con las mayores máximos de velocidad de transmisión de datos, y los mínimos con el final de ésta. Los datos del enlace L1 se presentan con la línea de color verde y los datos del enlace L5 con la línea de color amarillo.

La Figura 7.7 también representa los conjuntos de datos referentes a los enlaces L2, L3 y L4. L2 es representado mediante la línea de color rojo, L3 mediante la línea de color azul y L4 por la línea de color marrón. Los datos para estos enlaces presentan líneas constantes e iguales entre sí debido a que por ellos no se produce la transferencia de vídeo, es decir, cuando el vídeo se está

transfiriendo desde el nodo origen al nodo destino, éste no recorre ninguno de esos enlaces ya que previamente se ha establecido cual deberá ser el camino que tiene que recorrer la transferencia de información en forma de vídeo. Es por ello que estos enlaces presentan valores iguales a 0 prácticamente desde el inicio de la transferencia del vídeo hasta su finalización en la solicitud número 530 aproximadamente.

A la vista de los resultados observados en la velocidad de transmisión de datos se puede concluir que éstos son buenos. El buen desempeño del framework consigue captar a la perfección la velocidad de transmisión de datos en los enlaces participantes y en los que no.

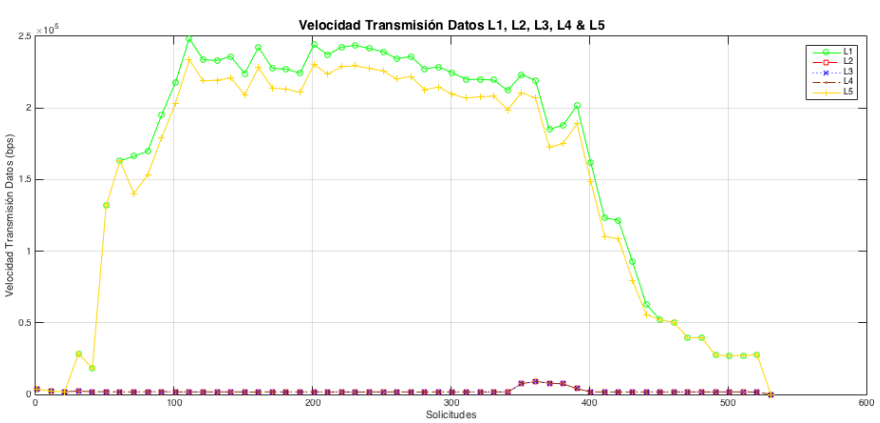

Figura 7.7: Velocidades de transmisión de L1 vs L5.

La Figura 7.8 presenta los valores de velocidad de transmisión de datos en los enlaces L1, L2, L3, L4, L5 y L6.

Como se observa, presenta valores más altos en los enlaces L1 (línea de color verde), L2 (línea de color rojo), L3 (línea de color azul) y L5 (línea de color amarillo). Este hecho se produce debido a que estos enlaces son participantes directos en el camino que recorre la transferencia del vídeo.

Aun así, se puede observar cómo los enlaces L4 (línea de color marrón) y L6 (enlace de color gris) presentan velocidades de transmisión de datos. Esto es

debido a que cada uno de los enlaces de esta topología ha sido parametrizado con valores de transmisión de datos (véase Tabla 7.4); en concreto, 1 Mbps. En el caso de haber asignado 0 Mbps a los enlaces no participantes, sus velocidades serían iguales a 0. En general, las velocidades observadas para los enlaces participantes se encuentran en los valores esperados.

Los máximos que se pueden observar en la Figura 7.8 coinciden con las mayores tasas de velocidades de transmisión de datos registradas durante la transferencia del vídeo. Estos máximos se pueden observar durante el inicio de las primeras solicitudes (rango 0-50). Estas velocidades registradas en los primeros instantes coinciden con el inicio de la transferencia del vídeo. Por lo tanto, la topología se ve en la coyuntura de tener que desplegar mayores velocidades de transmisión de datos.

Conforme la transferencia del vídeo va llegando a su fin, las velocidades de transmisión de datos empiezan a descender a valores cercanos al 0. Este hecho se produce a partir de la solicitud número 300. Este hecho no se puede apreciar en la Figura 7.13, ya que se ha optado por mostrar una vista aumentada de la gráfica para maximizar la comprensión de ésta.

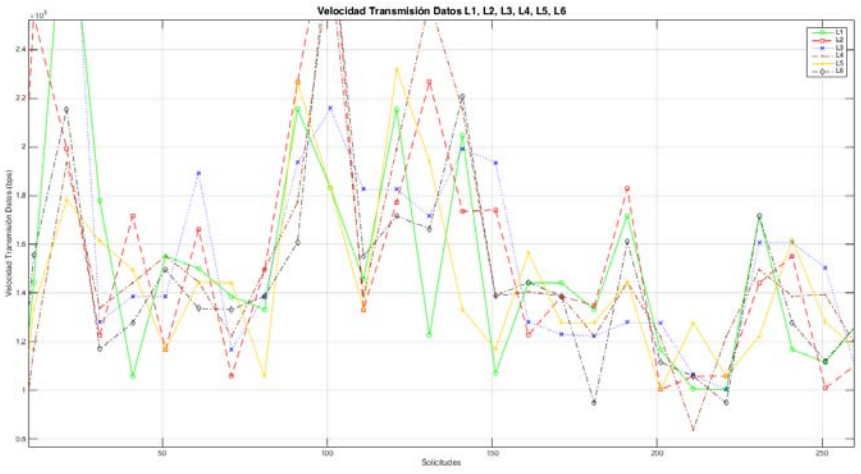

Figura 7.8: Velocidades de transmisión de L1, L2, L3, L4, L5 y L6.

7.2.2. Tasa de paquetes perdidos

La Figura 7.6 muestra la tasa de paquetes perdidos en el enlace L1 mediante la línea de color verde. Además, se representa mediante una línea horizontal discontinua de color azul el error teórico establecido para L1. Este error teórico es configurado en el emulador de Mininet o plano de datos.

Los datos obtenidos para el enlace L1 son bastantes óptimos. Sus valores se aproximan al error teórico establecido, con lo cual la tasa de paquetes perdidos obtenidos se encuentra dentro de los valores previstos. El valor de error teórico establecido para L1 se encuentra en un 5% de tasa de pérdidas de paquetes.

La gráfica de la Figura 7.9 demuestra el buen comportamiento del framework. Los valores observados para el enlace L1 se sitúan en torno a 10 y 5% de pérdidas. Por lo tanto, estos valores observados entran dentro del rango establecido para la configuración dada de L1.

Los máximos observados en la Figura 7.9 coinciden con la mayor carga que se produce en la topología cuando el vídeo se está transmitiendo. A su vez, lo mínimos que se registran en dicha Figura 7.9, concuerdan con la finalización del envío del vídeo. Este hecho se observa a partir de la solicitud número 400.

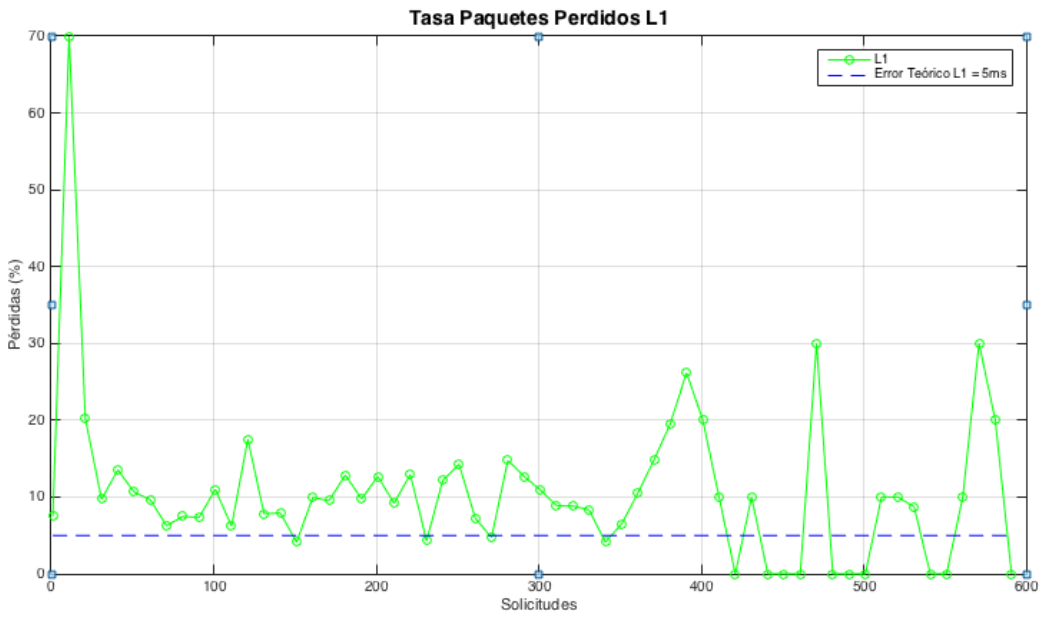

Figura 7.9: Tasa de paquetes perdidos de L1.

La Figura 7.10 representa la tasa de paquetes perdidos para el enlace L2 mediante la línea de color rojo. Además, se representa mediante la línea horizontal discontinua de color azul el error teórico establecido para dicho enlace. Este error teórico ha sido establecido en un 10% de pérdidas de paquetes.

En el caso de los valores de tasa de pérdidas de paquetes para el enlace L2, se observa que también se aproximan al error teórico establecido. La gráfica de la Figura 7.10 presenta algunos máximos y mínimos que se encuentran bastante alejados del error teórico establecido. Esos máximos se registran justo al inicio del envío del vídeo y en el tramo final. Por otra parte, los mínimos registrados coinciden de manera coherente con la finalización del envío del vídeo. Cuando se finaliza el periodo de solicitudes, ya no hay tráfico de red referente al envío del vídeo y por lo tanto no se puede producir pérdida de paquetes.

Puede afirmarse que los valores obtenidos para L2 son unos valores óptimos y que concuerdan con la salida esperada.

Estos valores óptimos son coherentes con los parámetros de tasa de paquetes perdidos establecidos en cada uno de los enlaces que conforman la topología básica.

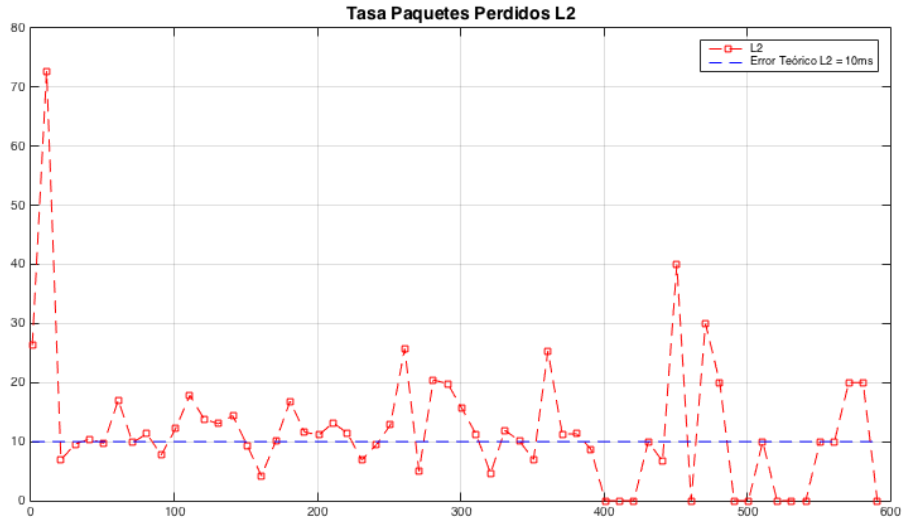

Figura 7.10: Tasa de paquetes perdidos de L2.

Los resultados obtenidos en la Figura 7.9 y la Figura 7.10 para los enlaces L1 y L2 arrojan que la tasa de paquetes perdidos obtenida para L2 es mayor que para L1. No obstante esto, observando de manera detenida dichas gráficas, se aprecia una gran similitud en los datos de ambos enlaces. Por lo tanto, a pesar de dicha similitud, se concluye que la transferencia del vídeo tuvo menor tasa de pérdidas cuando recorrió el enlace L1 que cuando recorrió el enlace L2.

Los valores de tasa de paquetes perdidos se muestran en la Figura 7.11. La gráfica muestra cómo la tasa de paquetes perdidos es bastante baja, lo que es indicativo de que la transferencia desde el host virtual servidor al host virtual cliente no ha sufrido pérdidas de información en su contenido. Los enlaces L1 y L4 se han configurado con tasas de pérdidas de paquetes del 5 y del 10%, respectivamente.

Observando de manera detenida la gráfica, se comprueba como los conjuntos de datos de L1 se sitúan en torno a valores cercanos al 5% de pérdidas. En el

caso de L4 sus valores se sitúan alrededor del 10%. Por lo tanto, a la vista de los resultados obtenidos, se puede afirmar el buen rendimiento del framework para esta topología respecto al parámetro de tasa de paquetes perdidos. Los resultados se ajustan con precisión a la tasa de paquetes perdidos establecido por defecto en el simulador para L1 y L4. Con respecto a los enlaces L2, L3 y L5, sus configuraciones no presentan asignaciones de porcentajes de pérdidas. Sus valores se observan dentro de la normalidad de pérdidas esperadas, aunque, a diferencia de L1 y L4, sus valores no presentan similitud entre ellos.

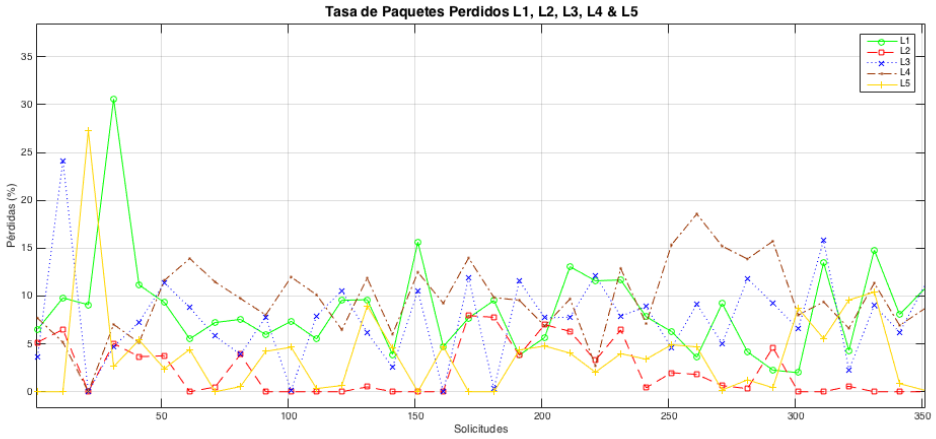

Figura 7.11: Tasas de paquetes perdidos de L1, L2, L3, L4 y L5.

La Figura 7.12 muestra la tasa de paquetes perdidos. Por un lado, se observa un primer grupo de datos: el conformado por el conjunto de datos de L1 (línea de color verde) y por el conjunto de datos de L5 (línea de color amarillo). El segundo grupo está formado por los datos de los enlaces L2 (línea de color rojo), L3 (línea de color azul) y L4 (línea de color marrón). También se representa mediante una línea gruesa discontinua de color azul el error teórico establecido en el simulador. Dicho error se encuentra ajustado a 10 ms.

La primera observación se centra en la representación de los enlaces L2, L3 y L4. De este primer conjunto, el único enlace que ha sido parametrizado con porcentaje de pérdidas de datos es L4. L4 se ha ajustado con un 10% de tasa de paquetes perdidos. Debido a que L2, L3 y L4 no son enlaces participantes en la

transferencia de vídeo, sus tasas de paquetes perdidos son próximas a tasas iguales al 0%. Por lo tanto, ninguno de estos enlaces presenta pérdidas de paquetes que pudieran afectar a la propia transferencia del vídeo que va desde el host servidor al host cliente.

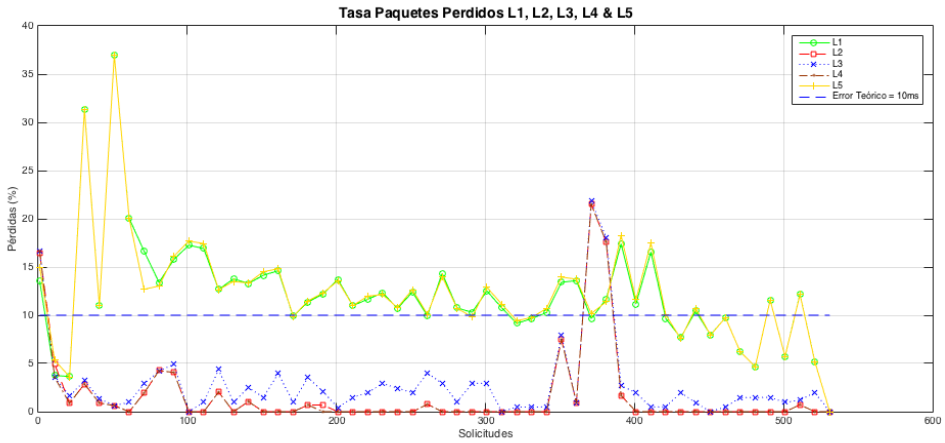

Figura 7.12: Tasas de paquetes perdidos de L1, L2, L3, L4 y L5.

La segunda observación se centra en los enlaces L1 y L5. Estos enlaces son protagonistas directos en la transferencia del vídeo. De estos dos enlaces el único que ha sido configurado con un porcentaje de tasa de pérdida de paquetes es L1. En concreto, esta configuración se sitúa en el 5% de pérdidas de paquetes. L1 y L5 presentan tasas de pérdidas de paquetes con valores bajos y similares entre los dos enlaces. Los valores registrados para L1 y L5 se aproximan al error teórico establecido en el simulador. Esta proximidad es un claro indicativo de que la transferencia de vídeo se ha producido de manera correcta cuando ha recorrido estos enlaces. Los datos obtenidos para estos enlaces corroboran el buen funcionamiento del framework implementado.

La Figura 7.13 representa la tasa de paquetes perdidos en los enlaces L1, L2, L3, L4, L5 y L6. Cada uno de estos enlaces ha sido representado con un color diferente. El enlace L1 es representado mediante la línea de color verde, L2 mediante la línea de color rojo, L3 con la línea de color azul, L4 con la línea discontinua de color marrón, L5 con la línea de color amarillo y L6 con la línea

de color gris. Además, se representa el error teórico establecido en el emulador mediante la línea gruesa discontinua de color azul.

El análisis de la tasa de paquetes perdidos para los enlaces presentes en esta topología ha sido dividido en dos grupos:

El primero de los grupos a analizar es el formado por los enlaces *L1*, *L2* y *L3*. *L1* y *L2* han sido configurados con tasas de pérdidas del 5%. Estos enlaces participantes en la transferencia del vídeo presentan buenos resultados. Los resultados de *L1* y *L2* se aproximan al error teórico establecido en el simulador. *L3* tiene también valores bastante bajos, con lo que puede concluirse que para este conjunto de enlaces el vídeo no sufrió pérdidas de paquetes.

El segundo conjunto de enlaces está compuesto por *L4*, *L5* y *L6*, donde el único enlace que recibe transmisión de información es el enlace *L5*. Por ello es el enlace que presenta menor tasa de pérdidas de paquetes acercándose sus valores a 0. Puede concluirse para *L5* que no hubo pérdida de información cuando el vídeo pasó por este enlace. En el caso de los enlaces *L4* y *L6*, la gráfica presenta altas tasas de pérdidas de paquetes ya que son enlaces no participantes en la transferencia de vídeo.

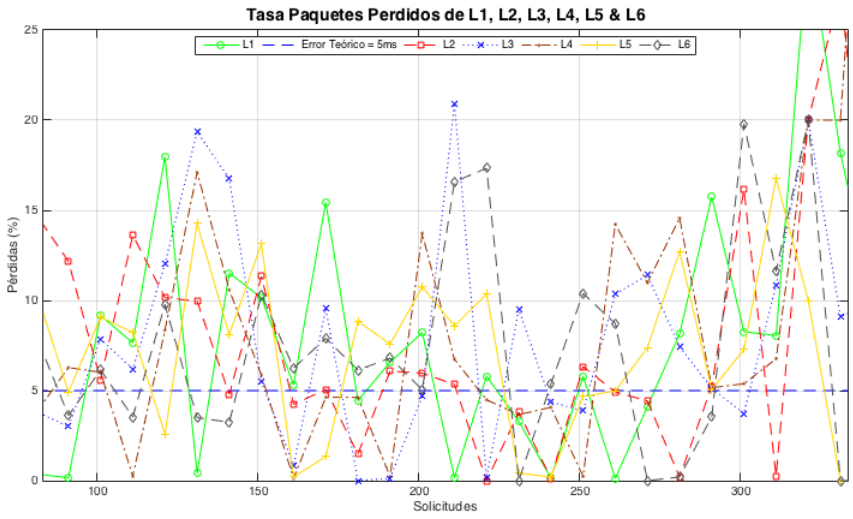

Figura 7.13: Tasas de paquetes perdidos de L1, L2, L3, L4, L5 y L6.

7.2.3. Retardo

La Figura 7.14 muestra los valores obtenidos para el parámetro de retardo o *delay* para los enlaces L1 y L2. Para L1 la medida establecida por el plano de datos se ajusta en 10 ms y para L2 se establece en 5 ms. Establecer medidas diferentes en el plano de datos para los enlaces L1 y L2 persigue corregir los errores de latencia que se producen entre los switches y el controlador. De esta forma se consiguen unos resultados más coherentes y realistas.

Los valores obtenidos para L1 representados con la línea de color verde y los de L2 representados con la línea de color rojo son correctos. Los errores teóricos se encuentran representados en la gráfica de la Figura 7.14 por una línea discontinua de color verde para L1 y una línea discontinua de color rosa para L2. El retardo observado para el enlace L1 cumple con creces la salida esperada a pesar de ser el enlace que cuenta con mayor error teórico establecido. Los máximos observados en el enlace L1 coinciden con el inicio del envío del vídeo y con el rango de solicitudes donde hubo mayor tasa de pérdidas de paquetes. Una vez que el vídeo ha entrado en el rango donde se va finalizando la transferencia, se observa como su retardo es parejo a 10 ms.

Por otra parte, el retardo observado en el enlace L2 también se encuentra parejo al error teórico establecido por defecto en dicho enlace. Los máximos observados coinciden con el inicio de la transferencia de vídeo y la pequeña tasa de pérdidas de paquetes que sufre la topología se encuentra en el rango 200-400.

Se concluye que, a pesar de poseer valores por defecto para el error teórico en cada uno de los enlaces, estos presentan una ligera similitud y se adecuan a los valores óptimos esperados.

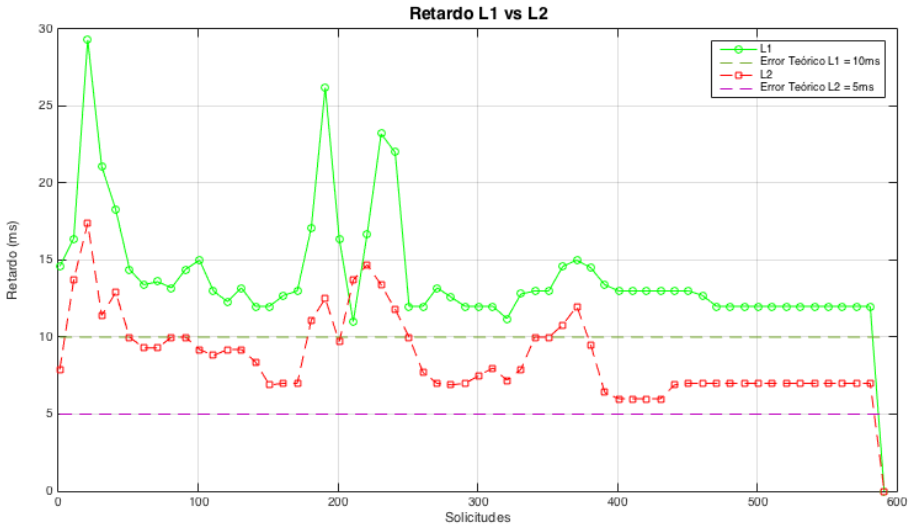

Figura 7.14: Retardos de L1 vs L2.

La Figura 7.15 representa la tasa de retardo para la topología serie. Los valores representados en la gráfica concuerdan con los parámetros establecidos por defecto en la configuración de los enlaces. Los conjuntos de datos para los enlaces L1, L2 y L5 presentan valores próximos a 0. Estos enlaces han sido configurados sin tasa de retardo, con lo que presentar valores cercanos a 0 es indicativo de que no se han producido retardos en dichos enlaces cuando se produjo la transferencia del vídeo desde el host virtual servidor al host virtual cliente.

Las configuraciones del retardo establecido para los enlaces L3 y L4 se parametrizaron con un retardo de 10 ms. Los restantes enlaces (L1, L2 y L5) no poseen tasa de retardo establecidas por defecto. Los datos de L3, representados con la línea de color azul, presentan valores cercanos a los 10 ms. Por el contrario, los datos del enlace L4 (línea de color marrón) configurado con una tasa de 10 ms presenta unos resultados por debajo de dicho valor. A pesar de tener una tasa de retardo pre-establecida, cuando el vídeo recorrió L4 la tasa de retardo fue muy baja. Por lo tanto, la mayor tasa de retardo se dio cuando el vídeo pasó por el enlace L3.

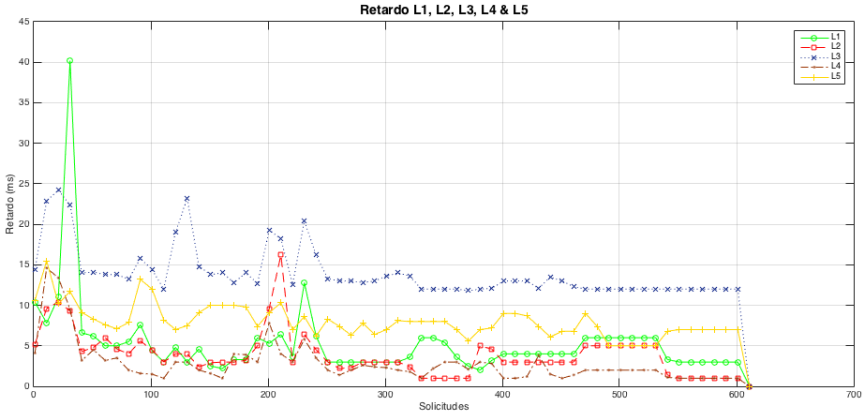

Figura 7.15: Retardos de L1, L2, L3, L4 y L5.

La Figura 7.16 presenta los valores de retardo en los enlaces L1, L2, L3, L4 y L5. Los datos obtenidos para los enlaces L1 (línea de color verde) y L5 (línea de color amarillo) son correctos según los valores establecidos por defecto. Por lo tanto, el retardo es mínimo cuando el vídeo se encuentra recorriendo estos enlaces. Además, según el vídeo va avanzando en la transferencia, el retardo para estos enlaces va descendiendo de manera gradual hasta la finalización.

En la Figura 7.16 también se representa el retardo observado en los enlaces no participantes en la transferencia como L2 (línea de color rojo), L3 (línea de color azul) y L4 (línea de color marrón). Los trazados que se pueden observar en la gráfica para L2, L3 y L4 muestran anomalías debido a que por ellos no se produce transferencia de información. En la práctica, este tipo de circulación de tráfico no se tiene en cuenta ya que dicha circulación podría deberse a circulación de tráfico residual o tráfico que se haya quedado almacenado de otras transferencias o instancias anteriores.

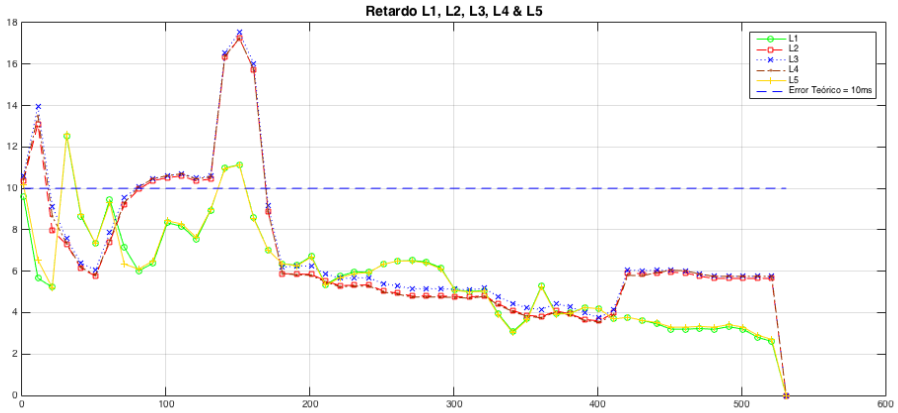

Figura 7.16: Retardos de L1, L2, L3, L4 y L5.

La Figura 7.17 presenta el retardo obtenido en los enlaces L1, L2, L3, L4, L5 y L6. Los datos presentan más irregularidades en comparación con el resto de topologías analizadas. Aun así, los datos obtenidos se encuentran dentro de los rangos esperados para esta topología.

La Figura 7.17 muestra los máximos que se producen en la tasa de retardo en los instantes donde se inicia la transferencia del vídeo desde el host virtual al host cliente. Conforme el vídeo va recorriendo la topología, se puede observar como las tasas de retardo se aproximan al error teórico establecido dentro del simulador.

Puede afirmarse que, a pesar de producirse en esta topología los mayores picos de retardo en comparación con el resto de topologías, las tasas de retardo obtenidas se encuentran dentro de los valores esperados.

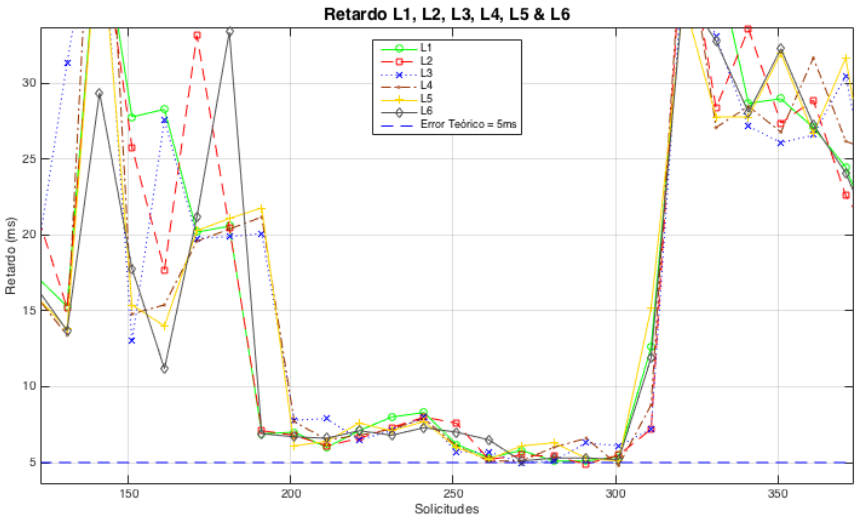

Figura 7.17: Retardos de L1, L2, L3, L4, L5 y L6.

7.3. Resumen

Este apartado presenta una comparativa analítica de los resultados observados en las gráficas resultantes para cada una de las topologías analizadas. Este resumen presenta los datos obtenidos para los parámetros de tasa de paquetes perdidos y retardo. Estos valores serán representados para cada uno de los enlaces presentes en las topologías. Las columnas están conformadas por los valores del plano de dato, mínimos, máximos, medias y desviaciones estándares.

	Tasa de paquetes perdidos (%)					Retardo (ms)				
	Plano Datos	min	max	Media	Deviación estándar	Plano Datos	min	max	Media	Deviación estándar
L1 (s1-s2)	5	8	11	9,323	5,716	10	10	13	12,626	5,474
L2 (s2-s3)	10	10	13	12,623	6,158	5	7	9	8,147	0,674

Tabla 7.5: Tabla Resumen datos obtenidos Topología Básica.

	Tasa de paquetes perdidos (%)					Retardo (ms)				
	Plano Datos	min	max	Media	Deviación estándar	Plano Datos	min	max	Media	Deviación estándar
L1 (s1-s2)	5	0	30	6.15347	10.70913296	0	0	40	5.522	10.5489372
L2 (s2-s3)	0	0	8	1.82643	5.98274963	0	0	17	4.22	3.80533324
L3 (s3-s4)	0	0	42	7.64058	14.46522342	10	0	24	14.44	4.53649596
L4 (s4-s5)	10	0	30	7.52817	12.47778576	0	0	14	3.164	4.01613778
L5 (s5-s6)	0	0	80	3.54353	9.375166512	5	0	15	8.374	3.05954991

Tabla 7.6: Tabla Resumen datos obtenidos en Topología en Serie.

	Tasa de paquetes perdidos (%)					Retardo (ms)				
	Plano Datos	min	max	Media	Deviación estándar	Plano Datos	min	max	Media	Deviación estándar
L1 (s1-s6)	5	0	30	7,8327	10,7091	0	0	15	5,068	10,5498
L2 (s2-s6)	0	0	9	0,0471	5,9827	0	0	41	13,928	3,8053
L3 (s3-s6)	0	0	70	0,8403	14,4652	10	0	36	18,068	4,5365
L4 (s4-s6)	10	0	60	0,6007	12,4778	0	0	34	19,004	4,0161
L5 (s6-s5)	0	0	70	8,2566	9,3752	5	0	27	10,132	6,2486

Tabla 7.7: Tabla Resumen datos obtenidos en Topología en Estrella

	Tasa de paquetes perdidos (%)					Retardo (ms)				
	Plano de Datos	min	max	Media	Deviación estándar	Plano de Datos	min	max	Media	Deviación estándar
L1 (s1-s2)	0	0	30	2.6211	8.3634	5	0	41	16.056	21.9275
L2 (s1-s3)	5	0	26	2.2918	7.8845	0	0	41	16.123	22.002
L3 (s2-s4)	0	0	44	2.5436	8.7285	0	0	43	16.339	22.5703
L4 (s2-s5)	0	0	45	2.1793	7.3425	0	0	52	15.897	21.9127
L5 (s3-s6)	0	0	42	2.6957	8.2369	10	0	41	15.984	21.1838
L6 (s3-s7)	0	0	40	2.5784	8.485	0	0	36	15.559	20.6645

Tabla 7.8: Tabla Resumen datos obtenidos en Topología Data Center

8. CONCLUSIONES Y TRABAJO FUTURO

8.1. Conclusiones

SDN/OpenFlow, al separar la capa de control del plano de datos, permite el diseño (e implementación) de un controlador (en Floodlight) para monitorizar una red. La implementación realizada permite leer los dispositivos que se encuentran conectados en una red y monitorizar su tráfico. Esta monitorización de tráfico aporta información referente al número de bytes que se están transmitiendo y recibiendo, así como valores de velocidad de transmisión, tasa de paquetes perdidos o tasa de retardo. Estos datos arrojan información del correcto o mal desempeño de una red.

El framework se ha desarrollado con una arquitectura totalmente escalable y rentable en cuanto a su utilización, ya que se hace uso del protocolo OpenFlow. La aplicación diseñada para obtener todos los datos del tráfico de red que va ocurriendo en redes SDN se extraen directamente desde el plano de datos. De esta manera, se contribuye a las políticas de código abierto que persigue el canon SDN. El framework implementado ha sido ampliamente experimentado.

Este framework de monitorización consigue obtener resultados precisos y reducir al mínimo la carga sobre los dispositivos de red como switches. La tasa de retardo es medida mediante la inyección de paquetes. Estos paquetes van recorriendo cada uno de los enlaces que conforman una red SDN, determinando de dicha manera el retardo que se produce desde los nodos extremos.

8.2. Trabajo Futuro

Como posible trabajo futuro se podría plantear la inclusión como entrada y de manera dinámica de cualquier tipo de topología. El framework admitiría la entrada de una topología con *N* switches y *M* hosts virtuales unidos por *J* enlaces, llevando a cabo el despliegue de dicha topología y procediendo de manera automática a la ejecución de un número *Y* de simulaciones, donde *Y* también sería pasado como parámetro de entrada.

Además, se puede utilizar este trabajo como fuente, ya que es abierto, con lo que puede servir a otros investigadores y desarrolladores para seguir impulsando la comunidad de SDN. También, se vislumbra una idea interesante, que consiste en demostrar la eficacia del framework en aplicaciones de calidad de servicio. Este framework persigue además la idea de mejorar su desarrollo para ser compatible con plataformas distribuidas.

9. Otro trabajo futuro interesante sería la transferencia de nuestro framework a un entorno real de pruebas, que por norma general poseen más de un switch. Así, se podría ampliar el valor de nuestra arquitectura más allá de la simple funcionalidad de este trabajo. La transferencia a un entorno real permitiría realizar una evaluación mucho más sofisticada e incluso transferir el framework a otras plataformas de controladores SDN como Ryu [RYU] y poder así combinarlo con otros sistemas de monitorización de tráfico de red como el que propone Braun [BRA].

BIBLIOGRAFÍA

[AAKS98] D. Alexander, W. Arbaugh, A. Keromytis, and J. Smith, "A Secure Active Network Environment architecture: Realization in SwitchWare," IEEE Network, vol.12, no.3, pp. 37-45, May/Jun 1998.

[BRA] L. Braun, C. Diekmann, N. Kammenhuber, and G. Carle, "Adaptive Load-Aware Sampling for Network Monitoring on Multicore Commodity Hardware," in IFIP Networking 2013, New York, NY, May 2013. [Online]. Available: http://dx.doi.org/10.1.1.395.9415.

[Ca99] K. Calvert. "Architectural Framework for Active Networks Version 1.0". 1999

[CCFRS05] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of a Routing Control Platform," in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2, NSDI'05, (Berkeley, CA, USA), pp. 15–28, USENIX Association, May 2005.

[CFPL07] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, (New York, NY, USA), pp. 1–12, ACM, August 2007.

[CNP03] A. Courtney, H. Nilsson, and J. Peterson, "The Yampa Arcade," in Proceedings of the 2003 ACM SIGPLAN workshop on Haskell, (New York, NY, USA), pp. 7–18, ACM, August 2003.

[COU13] Control and Data Together, Software Defined Networking course, https://www.coursera.org/

[E13] D. Erickson, "The Beacon Openflow Controller," in Proceedings of the Second ACM SIGCOMM Workshop on Hot topics in Software Defined Networking, (New York, NY, USA), pp. 13–18, ACM, August 2013.

[ECL] http://eclipse.org/downloads/

[EDTBT12] Egilmez, Hilmi E and Dane, S Tahsin and Bagci, K Tolga and Tekalp, A Murat "OpenQoS: An OpenFlow Controller Design for Multimedia Delivery with end-to-end Quality of Service over Software-Defined Networks" in Asia-Pacific Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 1-8, December 2012.

[ELIP14] European Lighthouse Integrated Project, Internet of Things Architecture, http://www.iot-a.eu/public

[FBMP12] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A Replication Component for Resilient OpenFlow-based Networking," in Proceedings of the 2012 IEEE Network Operations and Management Symposium, pp. 933–939, IEEE, April 2012.

[FGR13] N. Foster, A. Guha, M. Reitblatt, A. Story, M. Freedman, N. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison, "Languages for Software Defined Networks," IEEE Communications Magazine, vol. 51, pp. 128–134, February 2013.

[FHFM11] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in Proceedings of the The 16th ACM SIGPLAN International Conference on Functional Programming, (New York, NY, USA), pp. 279–291, ACM, September 2011.

[FO14] Frenetic Organization, https://www.frenetic-lang.org/pyretic.

[FRZ13] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," 2013.

[GBMP13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," Future Generation Computer Systems - The International Journal of Grid Computing and Science, vol. 29, pp. 1645–1660, September 2013.

[GEBMR13] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards Network-wide QoE Fairness using Openflow-assisted Adaptive Video Streaming," in Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking, FhMN '13, (New York, NY, USA), pp. 15, August 2013.

[GHM05] A. Greenberg, G. Hjaimtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A Clean Slate 4D Approach to Network Control and Management," ACM SIGCOMM Computer Communication Review, vol. 35, pp. 41–54, October 2005.

[GKPC08] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," ACM SIGCOMM Computer Communication Review, vol. 38, pp. 105–110, July 2008.

[GRF13] A. Guha, M. Reitblatt, and N. Foster, "Machine-verified Network Controllers," ACM SIGPLAN NOTICES, vol. 48, pp. 483–494, June 2013.

[GYAC09] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A Network Virtualization Layer," tech. rep., OpenFlow Switch Consortium, October 2009.

[GYAC10] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the Production Network be the Testbed?," in Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, (Berkeley, CA, USA), pp. 1–6, USENIX Association, October 2010.

[HSM12] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement problem," in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, (New York, NY, USA), pp. 7–12, ACM, August 2012.

[KF13] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," IEEE Communications Magazine, vol. 51, pp. 114–119, February 2013.

[KSDM12] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards QoE-driven Multimedia Service Negotiation and Path Optimization with Software Defined Networking," in Proceedings of the 20th International Conference on Software, Telecommunications and Computer Networks, vol. 1, pp. 1–5, IEEE, September 2012.

[KSXFE11] H. Kim, S. Sundaresan, M. Chetty, N. Feamster, and W. K. Edwards, "Communicating with Caps: Managing Usage Caps in Home Networks," ACM SIGCOMM Computer Communication Review, vol. 41, pp. 470– 471, August 2011.

[LNRS04] T. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The Softrouter Architecture," in Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networking, November 2004.

[MABP08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," ACM SIGCOMM Computer Communication Review, vol. 38, pp. 69–74, March 2008.

[MAT] http://www.mathworks.com/help/matlab/

[MBX] Middlebox Communication Architecture and Framework. P. Srisuresh, Kuokoa Networks J. Kuthan, Fraunhofer Institute FOKUS, J. Rosenberg, dynamicsoft, A. Molitor, Aravox Technologies, A. Rayhan. Ryerson University, August 2002.

[MFHW12] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A Complier and Run-time System for Network Programming Languages," ACM SIGPLAN NOTICES, vol. 47, pp. 217–230, January 2012.

[MGB] https://github.com/mininet

[MGBC09] L. A. Meyerovich, A. Guha, J. Baskin, G. H. Cooper, M. Greenberg, A. Bromfield, and S. Krishnamurthi, "Flapjax: A Programming Language for Ajax Applications," ACM SIGPLAN NOTICES, vol. 44, pp. 1–20, October 2009.

[MRFRW13] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software Defined Networks," in Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, (Berkeley, CA, USA), pp. 1–14, USENIX Association, April 2013.

[MWT] http://mininet.org/walkthrough/

[NOMS14] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, 'Opennetmon: Network Monitoring in OpenFlow 1.0.0 Software-defined Networks', in Network Operations and Management Symposium (NOMS), 2014, IEEE.

[NRFC09] A. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic Access Control for Enterprise Networks," in Proceedings of the 1st ACM workshop on Research on enterprise networking, (New York, NY, USA), pp. 11–18, ACM, August 2009.

[OSS11] OpenFlow Switch Specification v.1.1.0, pp. 1–56, 2011.

[OVB] https://www.virtualbox.org

[PaPe12] S. M. Patrick Le Callet and A. Perkis, "Qualinet White Paper on Definitions of Quality of Experience," European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003), March 2012.

[PF14] Project Floodlight, http://www.projectfloodlight.org/floodlight/

[PFCMC10] P. S. Pisa, N. C. Fernandes, H. E. Carvalho, M. D. Moreira, M. E. M. Campista, L. H. M. Costa, and O. C. M. Duarte, "OpenFlow and Xen-Based Virtual Network Migration," in Communications: Wireless in Developing Countries and Networks of the Future, vol. 327, pp. 170–181, Springer Berlin Heidelberg, September 2010.

[PFL] "Project Floodlight" http://www.projectfloodlight.org/floodlight/

[Pl14] PlanetLab, https://www.planet-lab.org/

[PPAC09] B. Pfaff, J. Pettit, K. Amidon, and M. Casado, "Extending Networking into the Virtualization Layer," in Proceedings of the ACM SIGCOMM HotNets, ACM, October 2009.

[PWH13] K. Pentikousis, Y. Wang, and W. Hu, "MobileFlow: Toward Software- Defined Mobile Networks," IEEE Communications Magazine, vol. 51, pp. 44–53, July 2013.

[PYT] https://www.python.org

[RFRS12] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for Network Update," ACM SIGCOMM Computer Communication Review, vol. 42, pp. 323–334, October 2012.

[RMTF09] A. Ramachandran, Y. Mundada, M. B. Tariq, and N. Feamster, "Securing Enterprise Networks Using Traffic Tainting," tech. rep., August 2009.

[RYU] http://osrg.github.io/ryu/

[SJSZRP00] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge, “Smart packets: Applying Active Networks to Network Management," ACM Transactions on Computer Systems, vol. 18, pp. 67, February 2000.

[SJSZRP98] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge, “Smart Packets for Active Networks," in Proceedings of the IEEE Second Conference on Open Architectures and Network Programming, March 1998.

[SSCF13] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, “Are We Ready for SDN? Implementation Challenges for Software-Defined Networks,” IEEE Communications Magazine, vol. 51, pp. 36–43, July 2013.

[SSCP12] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, “A Demonstration of Fast Failure Recovery in Software Defined Networking,” in Testbeds and Research Infrastructure. Development of Networks and Communities, vol. 44, pp. 411–414, Springer Berlin Heidelberg, June 2012.

[SSI14] SENSEI, Integrated EU Project, http://www.ict-sensei.org/

[SUNJ] https://www.java.com

[SZMM13] L. Shi, B. Zhang, H. T. Mouftah, and J. Ma, “DDRP: An Efficient Data-driven Routing Protocol for Wireless Sensor Networks with Mobile Sinks,” International Journal of Communication Systems, vol. 26, pp. 1341–1355, October 2013.

[TZE13] The Zettabyte EraTrends and Analysis, tech. rep., CISCO, May 2013.

[VBG13] A. Valdivieso, L. Barona and L. Villalba, “Evolution and Challenges of Software Defined Networking”, in Proceedings of the Workshop on Software Defined Networks for Future Networks and Services, pp. 61-67, IEEE, November 2013.

[VGSKF13] P. Vlacheas, R. Giaffreda, V. Stavroulaki, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A. R. Biswas, and K. Moessner, “Enabling Smart Cities through a Cognitive Management Framework for the Internet of Things,” IEEE Communications Magazine, vol. 51, pp. 102 – 111, June 2013.

[VKF12] A. Voellmy, H. Kim, and N. Feamster, “Procera: A Language for High- Level Reactive Network Control,” in Proceedings of the First Workshop on Hot topics in Software Defined Networks, (New York, NY, USA), pp. 43–48, ACM, August 2012.

[VLC] http://www.videolan.org/index.html

[VW12] A. Voellmy and J. Wang, “Scalable software defined network controllers,” ACM SIGCOMM Computer Communication Review, vol. 42, no. 4, pp. 289–290, August 2012.

[WoTu01] T. Wolf and J. Turner, “Design Issues for High-performance Active Routers,” IEEE Journal on Selected Areas in Communications, vol. 19, pp. 404–409, March 2001.

[YDAG04] L. Yang, R. Dantu, T. Anderson, and R. Gopal, “Forwarding and Control Element Separation (ForCES) Framework.” RFC 3746 (Informational), April 2004.