TRABAJO DE FIN DE GRADO EN

INGENIERÍA INFORMÁTICA

CURSO 2015-2016

DISEÑO E IMPLEMENTACIÓN DE UNA HERRAMIENTA DE VISUALIZACIÓN PARA ANÁLISIS EN TIEMPO REAL DE REDES SDN/OPENFLOW

Federico Ibáñez Moruno

Jesús Alejandro Lévano Lévano

Erik Steve Nieto Maldonado

Directores:

Luis Javier García Villalba

Ana Lucila Sandoval Orozco

Departamento de Ingeniería del Software e Inteligencia Artificial

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

Los abajo firmantes, matriculados en el Grado en Ingeniería Informática de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo Fin de Grado: "***Diseño e implementación de una herramienta de visualización para el análisis en tiempo real de redes SDN/Openflow***", realizado durante el curso académico 2015-2016 bajo la dirección de Luis Javier García Villalba y de Ana Lucila Sandoval Orozco en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Federico Ibáñez Moruno Jesús Alejandro Lévano Lévano Erik Steve Nieto Maldonado

Madrid, 17 de Junio de 2016

Abstract

Software Defined Networking allows the monitoring and centralized control of the network, so that administrators can have a real and complete view of it. The analysis and display of the different parameters represents the most feasible and practical way to program the network depending on user needs.

For this reason, in this project a modular architecture is developed, which aims to present real-time information that is monitored in a SDN network. First of all, the different monitored metrics (error, delay and data rate) are stored in a database. Then, these metrics are analyzed. Finally, the obtained results, of real time metrics and statistical data, are represented in a web application. The information is obtained through the REST interface that exposes the Floodlight controller and the analysis is done by means the comparison between the average and maximum values of the data set.

The results show graphically, in a clear and precise way, the different monitoring metrics. In addition, due to the modular nature of the architecture, it provides added-value to existing monitoring SDN solutions.

Keywords

API, Bootstrap, Controller, Floodlight, Highcharts, Mininet, OpenFlow, Python, REST, SDN, Software Defined Networking, Spring.

Resumen

Las Redes Definidas por Software (Software Defined Networking) permiten la monitorización y el control centralizado de la red, de forma que los administradores pueden tener una visión real y completa de la misma. El análisis y visualización de los diferentes parámetros obtenidos representan la forma más viable y práctica de programar la red en función de las necesidades del usuario.

Por este motivo, en este proyecto se desarrolla una arquitectura modular cuyo objetivo es presentar en tiempo real la información que se monitoriza en una red SDN. En primera instancia, las diferentes métricas monitorizadas (error, retardo y tasa de datos) son almacenadas en una base de datos, para que en una etapa posterior se realice el análisis de dichas métricas. Finalmente, los resultados obtenidos, tanto de métricas en tiempo real como de los datos estadísticos, son presentados en una aplicación web. La información es obtenida a través de la interfaz REST que expone el controlador Floodlight y para el análisis de la información se plantea una comparación entre los valores medios y máximos del conjunto de datos.

Los resultados obtenidos muestran gráficamente de forma clara y precisa las diferentes métricas de monitorización. Además, debido al carácter modular de la arquitectura, se ofrece un valor añadido a los sistemas actuales de monitorización SDN.

Palabras clave

API, Bootstrap, Controlador, Floodlight, Highcharts, Mininet, OpenFlow, Python, REST, Redes Definidas por Software, SDN, Spring.

Agradecimientos

Sírvase estas primeras líneas para agradecer especialmente a nuestro director Javier García Villalba, por la orientación que nos brindó en la selección de nuestro trabajo de fin de grado y por darnos la posibilidad de realizarlo.

Queremos dedicar unas líneas de agradecimiento a Lorena Barona y Leonardo Valdivieso, por el tiempo valioso compartido de manera generosa durante el desarrollo del presente trabajo, por la desinteresada colaboración y disponibilidad, también por proporcionarnos sus proyectos de investigación y amplia información bibliográfica que facilitó forjar la base de este trabajo.

No queremos olvidarnos de nuestros familiares y amigos, los cuales han estado con nosotros durante todo el transcurso del proyecto, ofreciéndonos su apoyo. Sin duda que, sin su apoyo, no habríamos llegado hasta dónde lo hemos hecho. De igual forma al personal de la biblioteca de la facultad, quienes nos han apoyado con amabilidad y buen trato en el transcurso de estos años.

Federico Ibáñez

Tania, gracias por haber hecho posible mi dedicación. Enzo, por haberte pasado horas mirándome callado como sabiendo lo que hacía. Kevin, por haber ayudado incondicionalmente a que esto sea posible. María, Fede, Mercedes, la mitad de mi tiempo ahora es vuestro.

Jesús Lévano

A Dora, por su paciencia y comprensión. Gracias por el sacrificio, por la fortaleza de salir adelante, por haberme formado como hombre de bien y por ser la mujer que me dio la vida. A Juana, mi ángel guardián y protectora incondicional. A José, por ser un ejemplo de vida y por los sabios consejos que me han sido de gran ayuda. A mi familia por el generoso apoyo que me han brindado durante toda mi vida.

Erik Nieto

Gracias a Itziar, Luis y Mónica por su apoyo, cariño y ánimo incondicional en los momentos complicados, sin ellos no habría podido llegar hasta donde estoy ahora. Además, a Rafael Pax por su amistad y ayuda desinteresada en todo momento.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ABREVIATURAS

API	*Application Programming Interface*
AJAX	*Asynchronous JavaScript and XML*
AOP	*Aspect-Oriented Programming*
ASP	*Active Server Pages*
BBDD	*Bases de Datos*
CSS	*Cascading Style Sheets*
CPOL	*Code Project Open License*
CORS	*Cross-Origin Resource Sharing*
DAO	*Data Access Objects*
DNS	*Domain Name System*
DoS	*Denial-Of-Service*
GASS	*Grupo de Análisis, Seguridad y Sistemas*
GIF	*Graphics Interchange Format*
GPL	*General Public License*
HDFS	*Hadoop Distributed File System*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IoC	*Inversion of Control*
IoT	*Internet of Things*
JEE	*Java Enterprise Edition*
JDBC	*Java Database Connectivity*
JDO	*Java Data Objects*
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
MANET	*Mobile ad hoc Networks*
MINA	*Multinetwork INformation Architecture*
MIME	*Multipurpose Internet Mail Extensions*
MPLS	*Multiprotocol Label Switching*
MVC	*Modelo Vista Controlador*
NAT	*Network Address Translation*

NFV	*Network Function Virtualization*
NOS	*Network Operating System*
REST	*Representational State Transfer*
ONF	*Open Networking Foundation*
ORM	*Object Relational Mapping*
OSI	*Open System Interconnection*
PDF	*Portable Document Format*
PHP	*Hypertext Preprocessor*
POJO	*Plain Old Java Object*
QoS	*Quality Of Service*
RPC	*Remote Procedure Call*
SDMN	*Software Defined Mobile Networks*
SDN	*Software Defined Networking*
SEO	*Search Engine Optimization*
SLA	*Service Level Agreement*
SOAP	*Simple Object Access Protocol*
SQL	*Structure Query Language*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*
XML	*eXtensible Markup Language*

1. INTRODUCCIÓN

El incremento de la cantidad de información que circula a través de la red y la diversidad de servicios requeridos por los usuarios han introducido la necesidad de buscar nuevos mecanismos que permitan flexibilidad y un control más eficiente de la red. Dentro de este contexto, se introduce el concepto de Redes Definidas por Software o Software Defined Networking (SDN). SDN facilita la programabilidad y gestión de la red de manera centralizada, a través de la separación del plano de datos del plano de control de los dispositivos de red. SDN se fundamenta en una arquitectura de tres capas:

1. Capa de control, que se encarga del control centralizado de la red, siendo esta característica la que permite que sea programable.
2. El plano de datos se encarga del reenvío de paquetes (en las distintas capas del modelo OSI).
3. Capa aplicación, SDN permite la creación de aplicaciones o políticas de red de alto nivel, con el fin de prestar un servicio óptimo a los clientes finales.

Para la comunicación de cada una de estas capas se definen dos tipos de interfaces: i) Interfaz de comunicación para capa de aplicación (Northbound) e ii) Interfaz de comunicación para capas inferiores (Southbound).

La interfaz de Southbound más conocida es el protocolo OpenFlow [PB13]. OpenFlow es un protocolo abierto y relativamente reciente (año 2008), el cual permite la conexión segura entre el controlador y los dispositivos de red. Con la ayuda de OpenFlow el controlador define el comportamiento de la red, independientemente de su tamaño o topología. Por ejemplo, se puede determinar el camino más óptimo en el proceso de transmisión de datos desde su origen hasta su destino.

Por un lado, en las redes tradicionales los dispositivos de red son gestionados a través de tecnologías propietarias, las cuales no se encuentran abiertas a los desarrolladores y administradores de red. Esto hace que la configuración de dichos dispositivos sea una tarea compleja [RBAB14]. Por su parte, SDN y OpenFlow brindan la posibilidad de personalizar el comportamiento de la red acorde a las necesidades del usuario, tanto para la gestión de tráfico real como para fines de investigación.

Por este motivo, es imprescindible proporcionar una herramienta de monitorización que proporcione información exacta tanto de la cantidad de tráfico que circula por la red, como del estado de la misma (errores, retardo y saturación). En este sentido, emerge una nueva necesidad en SDN: la capacidad de monitorizar y analizar el conjunto de métricas y datos que circulan por la red, con el fin de generar una interfaz gráfica fiable donde los resultados estén disponibles para un instante de tiempo determinado, independientemente del tamaño o características de la red.

1.1. OBJETO DE LA INVESTIGACIÓN

La correcta operación y control de una red se basa en la capacidad de monitorizar y analizar eventos específicos que permitan a priori o posteriori mejorar la gestión de la misma. SDN se posiciona como una alternativa prometedora. Para un administrador de red resulta de mucha utilidad el poder visualizar los parámetros monitorizados ya sean en tiempo real o producto del análisis estadístico de la información.

En este trabajo se presenta un diseño de arquitectura en la capa de aplicación para proporcionar una mejor experiencia al usuario, a través de la recopilación y presentación en un componente visual (aplicación web) de la información procesada. Las principales características de esta propuesta son la flexibilidad, modularidad y escalabilidad de los componentes del sistema. La arquitectura permite el desarrollo de aplicaciones (web y móviles) donde se muestre el tráfico a través de un enlace (data rate), las pérdidas (error rate) y el retardo (delay). Los objetivos que se pretenden con la arquitectura propuesta son:

- Brindar a los administradores de red una aplicación visualmente amigable para la monitorización SDN.
- Poner una API a disposición de los desarrolladores, donde puedan solicitar información de una red SDN simulada.
- Proveer información de las métricas monitorizadas en tiempo real, así como de la información estadística en un periodo de tiempo.

1.2. TRABAJOS RELACIONADOS

En esta sección se analizan los principales trabajos de investigación directamente relacionados con la monitorización de redes SDN. Una red tradicional utiliza básicamente dos métodos de monitorización:

1. Activo: Se basa en la utilización de sondas de medición las cuales recopilan información específica de la red, tal es el caso del protocolo SNMP.
2. Pasivo: analiza la información que atraviesa la red, sin la necesidad de ondas de medición, como por ejemplo la utilización de SPAN o tcpdump.

Por su parte SDN permite la monitorización a través del intercambio de mensajes entre el controlador y los dispositivos, con lo cual se tiene información actualizada de rendimiento de la red, tal es el caso de Flowsense [YLZSJV13]. Este trabajo se basa en un método de monitorización pasivo, el cual captura y analiza los mensajes de control entre los conmutadores y el controlador. Flowsense evita el sondeo activo de los contadores de los switches.

El framework de monitorización PayLess [RBAB14] propone un algoritmo de recolección de estadísticas adaptativo, que disminuye la sobrecarga en una red SDN. Payless expone un API REST para la recopilación de las estadísticas de la monitorización en tiempo real y permite la utilización de cualquier lenguaje de programación para acceder a la API mencionada.

Otros trabajos se enfocan en la monitorización de aplicaciones multimedia en entornos SDN, para lo cual se propone la gestión y asignación de recursos de forma activa y dinámica [ACETV14]. Este trabajo toma en cuenta el tipo de tráfico que circula por la red (datos, video, audio) y compara el grado de satisfacción que obtienen los usuarios (Quality of Experience, QoE). Con este fin, desarrolla un algoritmo que optimiza la QoE para transmisiones multimedia de vídeo.

Por su parte en [AMCSM15] se presenta una arquitectura para mejorar la calidad de experiencia de los usuarios con respecto a los servicios multimedia recibidos a través de una red definida por software. Este trabajo obtiene información actualizada originada por los protocolos multimedia de alto nivel, la cual es enviada al controlador SDN. Con este fin, se implementa el Protocolo de Control Multimedia en SDN (PCMS), el cual proporciona al controlador información periódica actualizada y en tiempo real sobre la calidad de la transmisión multimedia.

Adicionalmente, la monitorización en redes SDN podría ser de utilidad en entornos virtualizados y en el campo de redes móviles [CINGNM14]. El controlador facilitaría las labores de administración y control de los recursos físicos y lógicos (por ejemplo, máquinas virtuales) que forman el entorno virtualizado. Además, se busca mejorar la conectividad 5G a través de la virtualización de las funciones de red (Network Function Virtualization, VFN).

1.3. ESTRUCTURA DEL TRABAJO

El presente trabajo está dividido en siete capítulos. En el capítulo 2 se describen los principales conceptos relacionados a Redes Definidas por Software. Además, se describe las necesidades en el campo de monitorización de redes SDN.

El capítulo 3 explora las herramientas necesarias para conseguir una implementación real y consistente que favorezca la monitorización en redes SDN.

En el capítulo 4, se detalla el proceso el proceso de implementación y desarrollo de la arquitectura del presente proyecto. Para ello, se explican los elementos y componentes que forman el diseño modular, desde la obtención de las métricas de red hasta la visualización y el análisis de la información en tiempo real.

El capítulo 5 introduce los experimentos realizados en los distintos escenarios para comprobar el funcionamiento de este trabajo. Se analiza los resultados obtenidos de las distintas gráficas y se los contrasta con los parámetros definidos en las topologías.

En el capítulo 6 se presentan las conclusiones a las que se ha llegado tras el desarrollo de este trabajo y se introducen las principales líneas de actuación para ampliar las funcionalidades de la arquitectura propuesta.

Finalmente, en el capítulo 7 se describe la participación común e individual que han tenido los integrantes del grupo de desarrollo de la presente arquitectura.

1.4. INTRODUCTION

The increasing of information that flowing through the network and the diversity of services required by users, have introduced the need for new mechanisms that allow flexibility and more efficient network control. In this context, the concept of Software Defined Networking (SDN) is introduced. SDN facilitates network management and programmability of the network through the separation of data and control plane in network devices. SDN is based on a three-tier architecture:

1. Control layer, which is responsible of the centralized control of the network, allowing its programmability.
2. Data plane, which is responsible of the packet forwarding process (in the different OSI model layers).
3. Application layer. SDN enables the creation of applications and high level network policies in order to provide optimal service to end customers.

To communicate each of these layers, two types of interfaces are defined: i) Communication interface for application layer (Northbound API) and ii) Communication Interface for lower layers (Southbound API).

The best known southbound interface is the OpenFlow protocol [PB13]. OpenFlow is an open and relatively new protocol (2008), which allows secure connection between the controller (control plane) and network devices (data plane). The controller obtains the status of each devices in order to determine the behavior of the network, regardless their size or kind of topology. For example, it is possible to determine the most optimal path in the data transmission process between the source and the destination point.

Traditional network devices are managed through proprietary technologies which are not open to developers and network administrators. This cause the configuration of these devices a complex task [RBAB14]. For its part, SDN and OpenFlow provide the ability to customize the network behavior according to user needs, for both the management of real traffic and research purposes.

In this context, it is essential to have a monitoring tool to provide accurate information of the traffic flowing through the network and its status (errors, delay, saturation, among others). In this sense, it emerges a new need in SDN: the ability to analyze and show the set of metrics obtained from the network. The objective of this work is to generate a reliable graphical interface where results are available for a specific period of time, regardless of size or network characteristics.

2. Redes Definidas por Software

2.1. Definición

Las Redes Definidas por Software, ofrecen una solución a las limitaciones de las arquitecturas tradicionales. Se busca optimizar los distintos servicios (tráfico de datos, video, nube, etc.) para garantizar un servicio más eficiente a los usuarios finales a través de la programabilidad de la red. Como se puede observar en la Figura 2.1, SDN se basa en la separación del plano de control del plano de datos, los cuales se comunican a través de un protocolo de comunicación, el más conocido el protocolo OpenFlow.

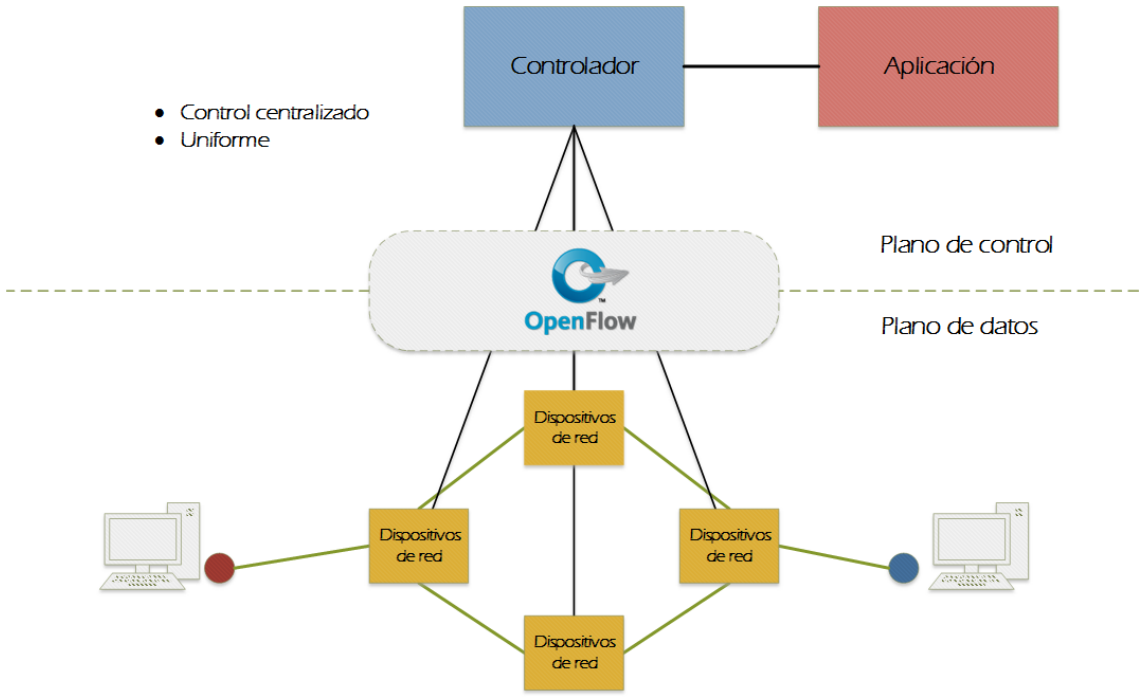


Figura 2.1: Concepto de SDN

2.2. LIMITACIONES EN ARQUITECTURAS TRADICIONALES

Las arquitecturas tradicionales de red poseen una serie de limitaciones que no permiten aprovechar sus características de forma sencilla y que además no están diseñadas para cubrir los requerimientos de los usuarios actuales. Una de las principales limitaciones es la dificultad y la gran cantidad de tiempo que se invierte a la hora de introducir o reconfigurar un elemento del sistema, tal es el caso de un nuevo dispositivo de red o de un servicio personalizado.

Por otra parte, la estandarización de un protocolo es un proceso lento que no permite la escalabilidad de los servicios acorde a las necesidades del mercado. En el caso de tecnologías como la computación en la nube o big data se requiere una respuesta ágil en demanda de los servicios, lo cual hace que la gestión de los sistemas sea una tarea compleja. En el caso específico de big data, se necesita un gran ancho de banda para poder transmitir grandes cantidades de información a través de la red.

En este contexto, SDN permite la personalización y la programabilidad de la red acoplándose de esta forma a las necesidades actuales. La principal diferencia entre una red tradicional y una red SDN es la separación del plano de datos del plano de control, como se muestra en la Figura 2.2.

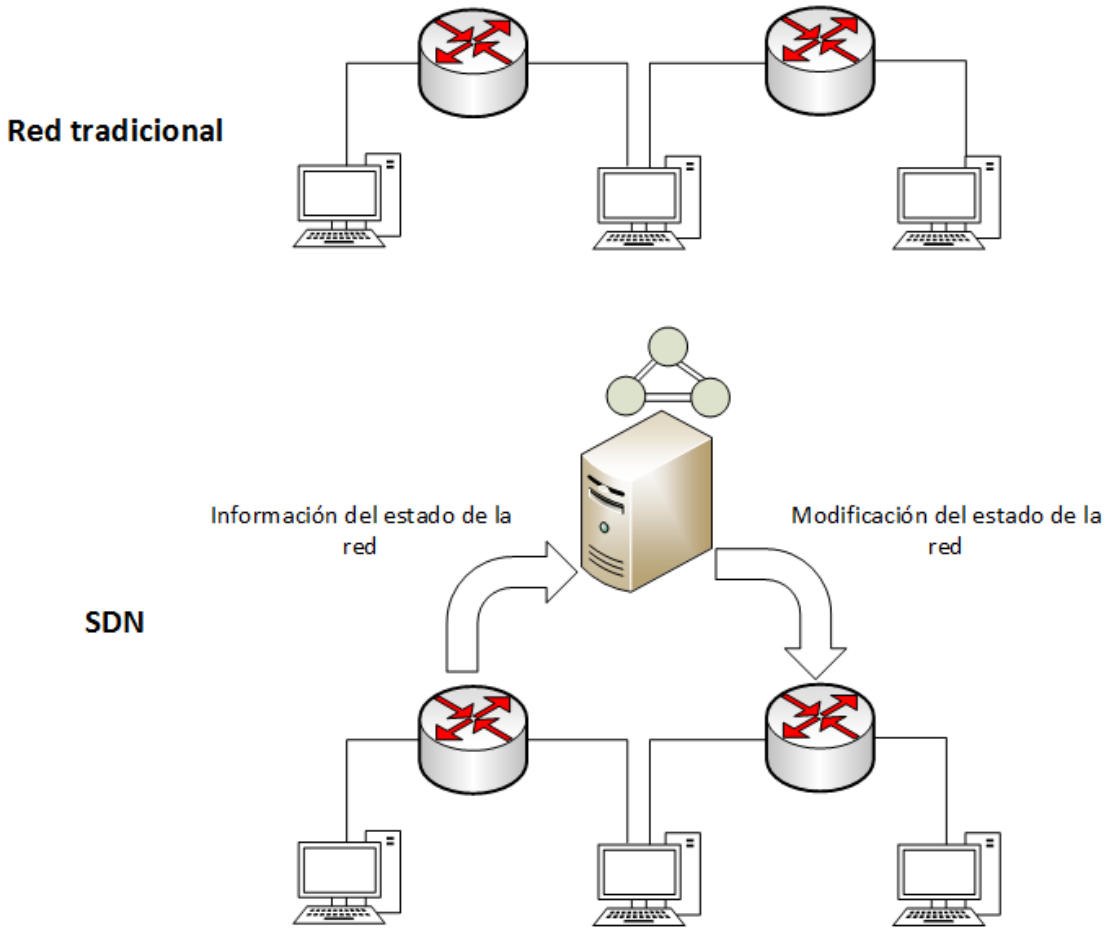


Figura 2.2: Comparación entre arquitectura tradicional y SDN.

En los dispositivos de red de una red tradicional el plano de datos y el plano de control se encuentran dentro del mismo dispositivo, a diferencia de una red SDN, en donde el plano de control se encuentra en un dispositivo exterior conocido como controlador.

2.3. Arquitectura de Redes Definidas por Software

La arquitectura en de una Red Definida por Software facilita la separación entre los dispositivos de red, las funciones de control y las aplicaciones del sistema, como se muestra en la Figura 2.3.

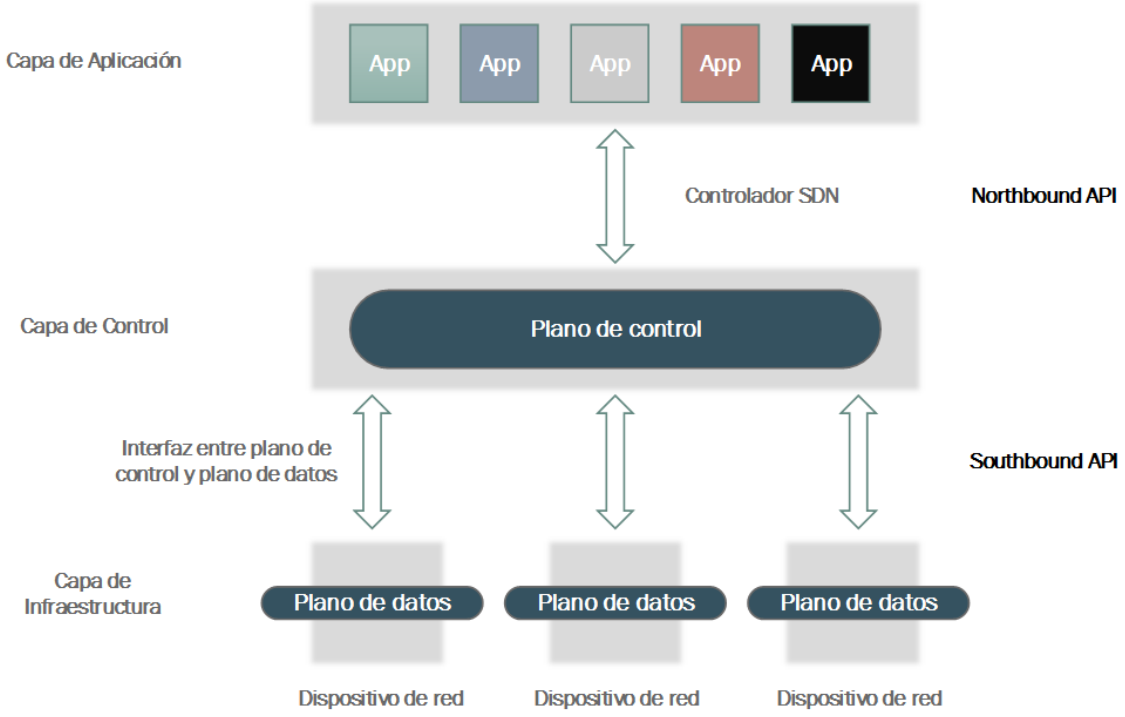


Figura 2.3: Arquitectura SDN

SDN define una arquitectura de tres capas:

1. **Capa de Infraestructura:** está compuesta por los distintos dispositivos hardware que forman una red. Estos dispositivos deben tener soporte del protocolo OpenFlow.
2. **Capa de Control:** está formada por el controlador, que se encarga de enviar las órdenes a los componentes hardware de la capa de infraestructura y proporciona una API abierta tanto con la capa superior (capa de aplicación) como con la capa inferior.
3. **Capa de Aplicación:** se ubica en el nivel más alto de la arquitectura y es la encargada de la implementación de las políticas y aplicaciones de alto nivel.

Cuando se utiliza políticas de alto nivel provenientes de la capa aplicación, estás son transmitidas al controlador (capa de control), el cual toma las decisiones en base al estado actual de la red. Finalmente, estas decisiones se transmiten a la capa de infraestructura.

2.3.1. EL PLANO DE DATOS

El desacoplamiento del plano de control del plano de datos, se hace con el propósito de que los dispositivos de red puedan ser programados y controlados por un agente externo. De esta forma se proporciona flexibilidad y escalabilidad a la hora de introducir nuevas funcionalidades.

El plano de datos se encarga de procesar los paquetes que llegan a los dispositivos de red a través del medio físico (comunicación por cable o inalámbrica). Cuando un paquete llega a un dispositivo, éste lo envía al controlador SDN para que tome el control del mismo y modifique la cabecera convenientemente. Después, el controlador envía las instrucciones específicas sobre el tratamiento del paquete (Ej. puerto de salida, siguiente salto), para que finalmente el conmutador las aplique. El controlador puede utilizar algoritmos programados previamente en el plano de control. De esta forma, se pueden definir de forma clara y sencilla una gran cantidad de funcionalidades nuevas, que con en una red tradicional sería difícil o imposible de implementar. Por ejemplo, se puede programar el reenvío de paquetes, desarrollar un control de acceso, marcar o inspeccionar datagramas o monitorizar el tráfico de la red para su posterior tratamiento. En la Figura 2.4. se ilustra el flujo del tratamiento de la información.

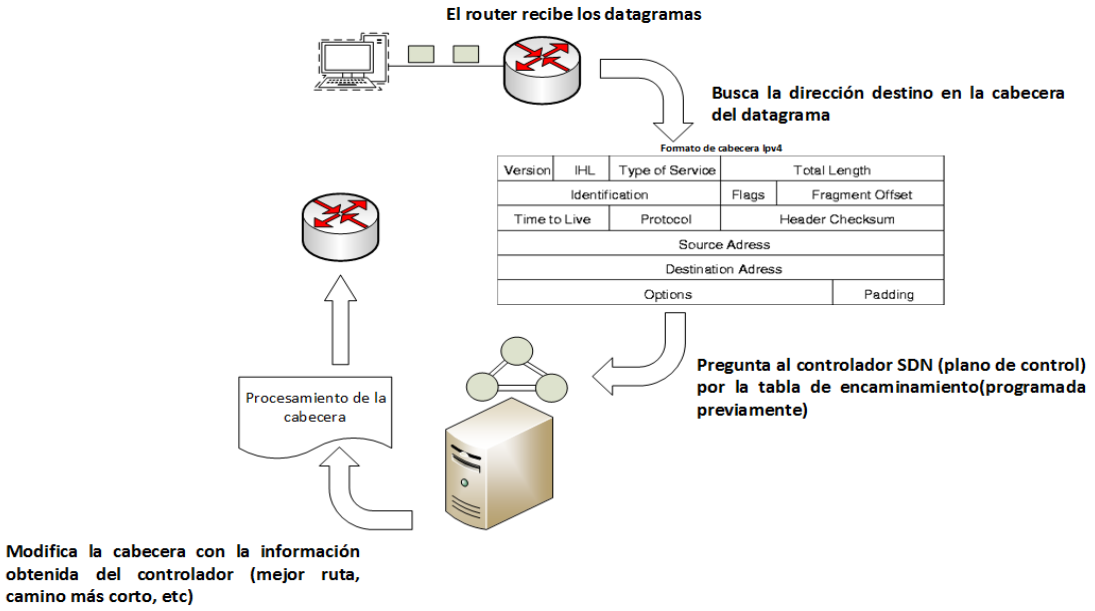


Figura 2.4: El Plano de Datos

2.3.1.1. Mininet

Mininet [Mi1] es una herramienta que permite la emulación de Redes Definidas por Software mediante la utilización del protocolo OpenFlow y un entorno Linux en una máquina virtual [ITMWEB]. Dicho entorno hace posible la creación de una red SDN virtual, emulando diferentes tipos de dispositivos tal es el caso de conmutadores, máquinas host, enlaces y controladores. Además, proporciona una ventana de comandos, la cual permite que el usuario genere de forma interactiva nuevas topologías de red o programas controlador.

Los dispositivos que se utilizan para crear las topologías se comportan como dispositivos reales, los paquetes que se intercambian lo hacen a través de lo que parecen interfaces ethernet reales, permitiendo inclusive definir características en los enlaces (retardo o la tasa de bits y errores).

Mininet permite la ejecución de funcionalidades Linux sobre la máquina virtual tal es el caso de wireshark. Se puede crear una topología de red de forma rápida y todas las características de la red son configurables. Además, Mininet es un proyecto de código abierto y está en constante desarrollo.

Por otra parte, Mininet presenta una serie de limitaciones como por ejemplo que sus recursos se ven limitados por las características del dispositivo físico en el que ejecuta. Además, que, al utilizar el kernel de Linux no se podrán lanzar aplicaciones que funcionen en otros sistemas operativos. En la Tabla 2.1, se pueden observar los resultados de una comparación entre diversas plataformas para simular/emular el funcionamiento de una red SDN (ESTINET, NS-3, MININET).

	ESTINET	NS-3	MININET
Modo de simulación	Sí	Sí	No
Modo de emulación	Sí	No	Sí
Compatible con controladores reales	Sí	No	Sí
Resultado repetible	Sí	No	Sí
Escalabilidad	Alta (para un solo proceso)	Alta (para un solo proceso)	Media (para múltiples procesos)
Exactitud en los resultados de rendimiento	Sí	No admite protocolo árbol y ningún controlador real	Sin fidelidad de rendimiento
Soporte GUI	Para configuración y observación	Sólo para observación	Sólo para observación

Tabla 2.1: Comparación entre EstiNet, NS-3 y MiniNet [Mi1]

2.3.2. EL PLANO DE CONTROL

En una red tradicional, el plano de datos y el plano de control se encuentran dentro del mismo dispositivo de red, razón por la cual la escalabilidad y flexibilidad para la introducción de nuevas funcionalidades es limitada. Por su parte, SDN al tener el plano de control separado del plano de datos, permite la adaptación de nuevos servicios acorde a las necesidades de los usuarios.

El plano de control (controlador) es el encargado de la configuración de cada nodo y de la programación del reenvío de los flujos de forma automática, tomando en cuenta la situación actual de la red. Si se compara con una red tradicional, el administrador de red tendría que efectuar las modificaciones o ajustes de configuración en cada dispositivo de forma manual.

El controlador es también conocido como Sistema Operativo de Red (NOS), pudiendo ser éste centralizado o distribuido. Al ser el controlador el cerebro de la red, una falla del mismo produce un fuerte impacto en todo el sistema. Por tanto, se recomienda que exista una configuración distribuida con el fin de asegurar el funcionamiento continuo de la red. Para la comunicación entre la capa de control con la capa de aplicación y la capa de infraestructura, se definen dos tipos de interfaces (Northbound y Southbound). La interfaz de Northbound permite el desarrollo de aplicaciones de alto nivel, como, por ejemplo, la provisión de sistemas de seguridad, integración de middlebox, recursos para la administración y control, etc.

En el otro extremo se encuentra la interfaz Southbound, la cual comunica el plano de datos y el de control. Esta interfaz permite la exteriorización del estado de los dispositivos de red hacia el controlador (principio fundamental de SDN). Existen diferentes interfaces Southbound, tal como OpenFlow, el protocolo para Juniper's Contrail Controller (Linux) o el de Cisco's Open Network Environment.

Existen diferentes controladores los cuales pueden ser clasificados según el lenguaje de programación en el que han sido desarrollados. En la Tabla 2.2 se muestran algunos de éstos.

Lenguajes de programación	**Controladores**
C++	NOX
Java	Floodlight, OpenDayLight, Maestro
Python	POX, Ryu, Pyretic

Tabla 2.2: Clasificación Controladores SDN.

2.3.2.1. NOX y POX

NOX es la primera generación de controladores OpenFlow open source. Se encuentra escrito en el lenguaje de programación C++, lo que garantiza un código limpio y que adicionalmente tiene una gran comunidad de soporte. El modelo de programación es orientado a eventos, lo que permite tener un conjunto de manejadores (handlers), los cuales responden con acciones tras un evento. Por su parte, POX es similar a NOX con la diferencia de que fue desarrollado en Python. POX tiene compatibilidad con OpenFlow v.1.0, que permite un rápido prototipado y experimentación.

2.3.2.2. Floodlight

Floodlight (Figura 2.5) es un controlador open source desarrollado en Java y compatible con OpenFlow v.1.0. Nace a partir de su antecesor, el controlador Beacon y que además cuenta con el soporte de mantenimiento de Big Switch Networks.

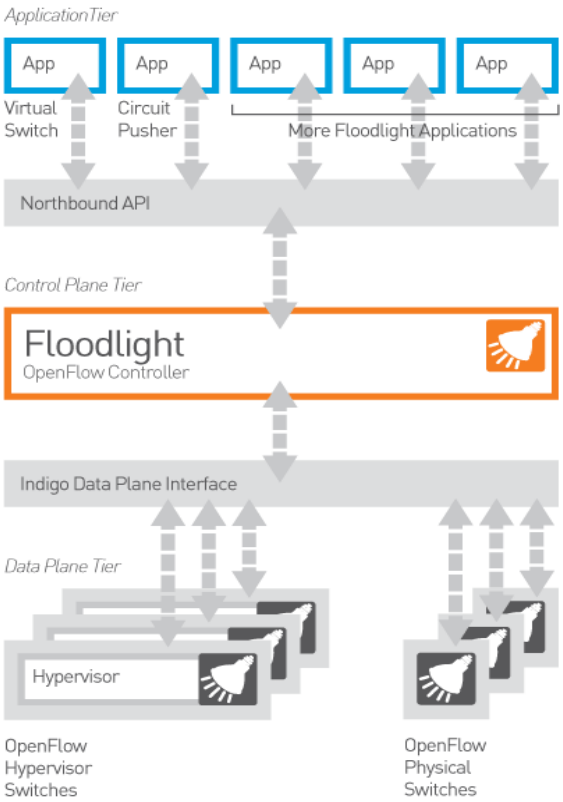


Figura 2.5: Visión global de Floodlight.

Entre las ventajas de Floodlight [PF14], es que tiene una extensa documentación y que expone una API REST, sin embargo, su principal desventaja es su curva lenta de aprendizaje. Floodlight [PF14] está compuesto por diferentes módulos que permiten la implementación de funcionalidades típicas de un controlador SDN, tales como: i) Descubrir y exponer los estados y eventos de la red (topología, dispositivos, flujos), ii) Permitir la comunicación del controlador con los dispositivos de red (protocolo OpenFlow), iii) Gestión de los módulos de Floodlight y recursos compartidos (hilos), iv) Externalización de una interfaz web de usuario y un servidor de depuración (Jython). Los componentes más importantes de la arquitectura de Floodlight [PF14] se muestran en la Figura 2.6.

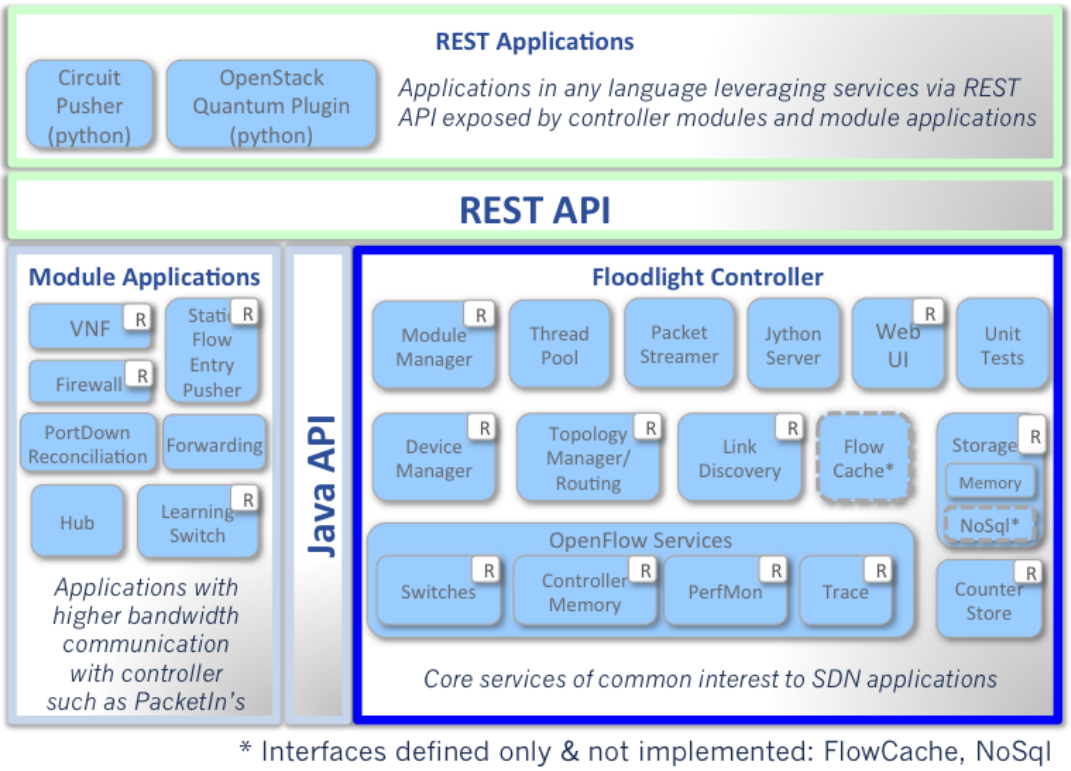


Figura 2.6: API de Floodlight [PF14].

- **FloodlightProvider:** Ofrece dos funcionalidades. Se ocupa de las conexiones con los dispositivos de red y convierte los mensajes OpenFlow en eventos que otros módulos pueden utilizar. Su segunda función es que decide el orden en que son enviados los mensajes OpenFlow.
- **OFSwitchManager:** Módulo diseñado para gestionar todos los conmutadores OpenFlow conectados al controlador Floodlight.

- **DeviceManagerImpl:** Realiza el seguimiento de los dispositivos y define el dispositivo destino para un nuevo flujo.
- **LinkDiscoveryManager:** Es el responsable de descubrir y mantener el estado de los enlaces en la red OpenFlow.
- **TopologyService:** Mantiene la información de la topología de la red, necesario para determinar el encaminamiento en la red.
- **RestApiServer:** Permite a los módulos exponer la API REST a través de HTTP.

Todas estas funcionalidades han permitido que Floodlight se convierta en uno de los controladores favoritos tanto para la experimentación con SDN, así como en despliegues reales.

2.3.2.3. OpenDaylight

OpenDaylight (ODL) es un controlador de código abierto, modular, robusto, extensible, escalable y que tiene una infraestructura de controlador multi-protocolo. Entre sus principales ventajas se puede mencionar la buena aceptación por parte de la industria, de sistemas en la nube y que permite la integración con OpenStack[1]. ODL proporciona una plataforma de abstracción de servicios basada en modelos que permiten a los usuarios escribir aplicaciones a través de una amplia variedad de hardware y protocolos Southbound. Adicionalmente, ODL se enfoca en la implementación de funcionalidades basadas tanto en SDN y NFV o su combinación. En febrero del 2016, ODL lanzó la cuarta versión del controlador conocida como “Beryllium”. En la Figura 2.7, se muestra un resumen de la arquitectura ODL Beryllium.

[1] "Home » OpenStack Open Source Cloud Computing Software." 2012. 22 May. 2016 <https://www.openstack.org/>

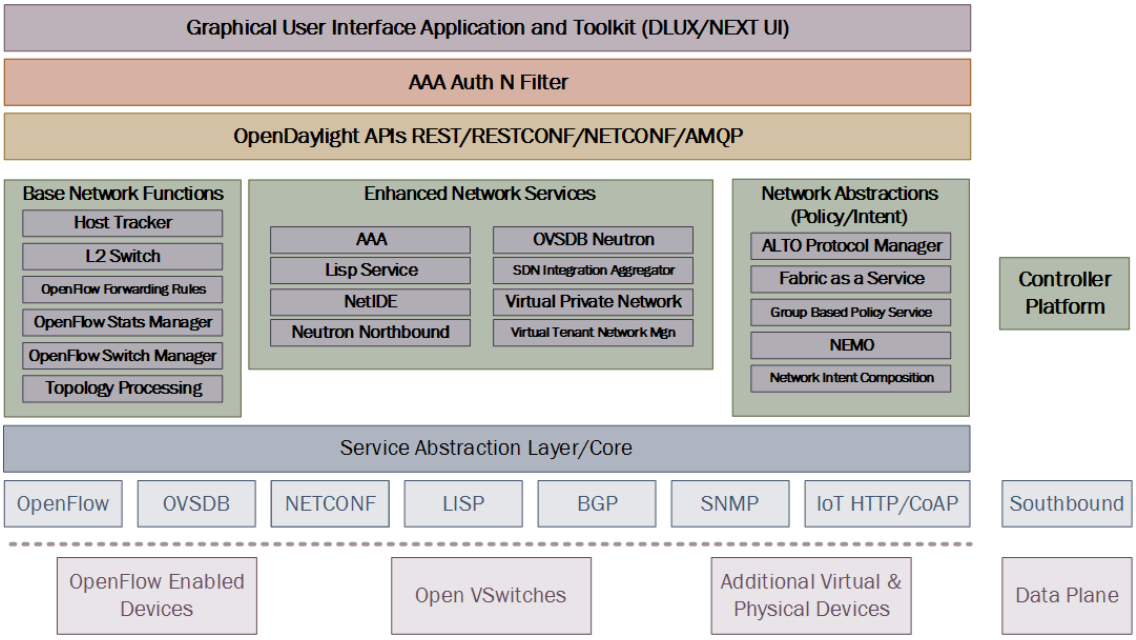


Figura 2.7: Arquitectura OpenDaylight [ODL16].

ODL Beryllium añade soporte para alta disponibilidad y clusterización en entornos OpenStack, así como un mejor soporte para la API Neutron y para NetIDE (Figura 2.8). NetIDE es un plug-in que permite la ejecución aplicaciones de otros controladores como por ejemplo Ryu, Pyretic, Floodlight, etc.

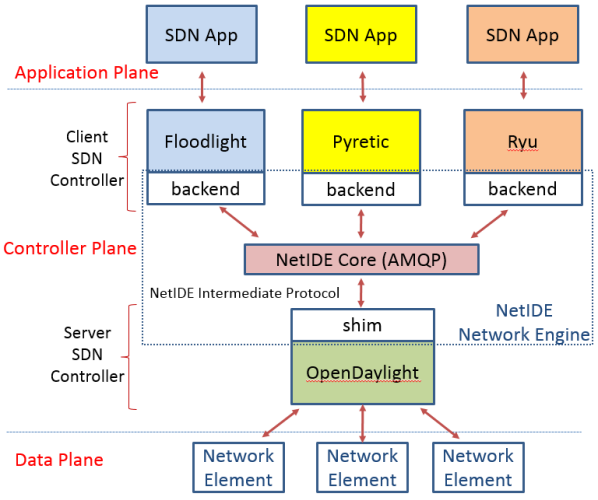


Figura 2.8: NetIDE OpenDaylight [ODL16].

2.3.3. EL PLANO DE APLICACIÓN

Este plano tiene como objetivo principal el desarrollo de aplicaciones de alto nivel usadas por los clientes para la gestión de los servicios, tal es el caso de aplicaciones de video, audio, seguridad, optimización y gestión de la información. Como se comentó anteriormente, la capa aplicación se comunica con el controlador a través de la interfaz de Northbound API (Figura 2.9).

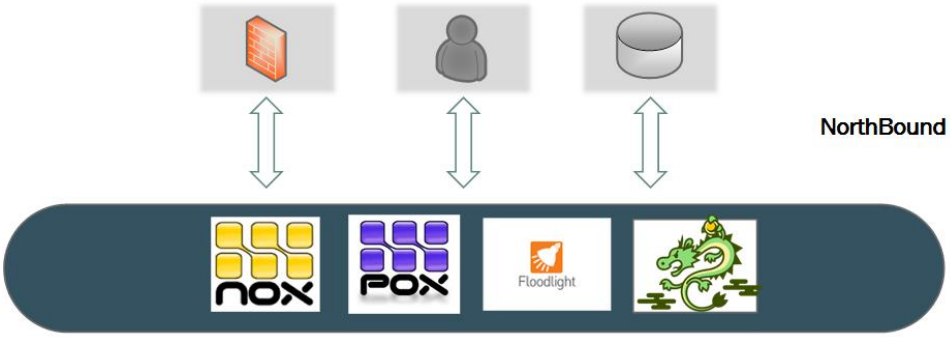


Figura 2.9: Plano de aplicación SDN

Como ejemplos de aplicaciones, se pueden nombrar:

1. *FlowVisor* permite la virtualización de una red SDN, donde el plano de datos es compartido por diferentes redes virtuales.
2. *ElasticTree,* caracteriza el consumo de energía de un Data Center.
3. *OpenPipes* es una plataforma que permite la implementación de sistemas hardware distribuidos, comunicados a través del protocolo OpenFlow.

La capa de aplicación busca: automatizar la configuración y gestión de los servicios, permitir el despliegue de nuevos servicios de red y mejorar la toma de decisiones.

2.4. APLICACIONES DE REDES DEFINIDAS POR SOFTWARE

El concepto de SDN ha sido aplicado en diferentes campos de acción, como por ejemplo el internet de las cosas (IoT), virtualización, monitorización, seguridad, computación en la nube, entre otros. Existen diferentes organizaciones que impulsan la promoción y adopción de SDN, tal es el caso de Open Networking Foundation [Op1].

ONF presenta las directrices en diferentes campos de estudio tal como se detalla a continuación:

1. **Calidad de servicio**

   SDN permite la personalización de los servicios requeridos por el usuario, de tal manera que se consigue una red flexible, escalable y abierta a las tendencias del mercado. En [BRABR13] se proporciona una interfaz para especificar acuerdos de nivel de servicio (SLA), los cuales son garantizados mediante la utilización del protocolo OpenFlow (monitorización y reconfiguración automática de los parámetros de la red).

2. **Internet de las cosas (IoT),** este concepto engloba una serie de soluciones de comunicación heterogéneas, por ejemplo: tecnologías móviles como Wifi, ZigBee[2], Bluetooth y protocolos de enrutamiento MANET[3]. En ese sentido, en [QDGBV14] se propone un sistema basado en SDN, cuyo objetivo es proporcionar niveles de calidad diferenciados para cada tarea en un ambiente IoT heterogéneo.

   Este trabajo utiliza también un middleware conocido como Multinetwork INformation Architecture (MINA). Las pruebas realizadas sobre esta propuesta se realizaron en escenarios IoT de gran escala como vehículos eléctricos, sitios de recarga eléctrica, infraestructuras de redes inteligentes, entre otros.

3. **Network Function Virtualization,** propone desacoplar las funciones de típicas de red del hardware (Ej. NAT, detección de intrusos, DNS, etc.), de tal forma que puedan ejecutarse en software [HGJL15]. SDN y NFV son tecnologías complementarias, ejemplo de ello es la gestión de las funciones virtualizados de red a través del controlador SDN.

[2] Alliance, ZigBee. "Zigbee specification." 1 Dec. 2006.
[3] Basagni, Stefano et al. *Mobile ad hoc networking*. Stefano Basagni et al. John Wiley & Sons, 2004.

4. **Software Mobile Networks,** la aplicabilidad de SDN en redes móviles se centra en la gestión eficiente de los dispositivos y aplicaciones, así como de la gran cantidad de información que generan dichos dispositivos. En [PWH13] se presenta un modelo tanto para arquitecturas tradicionales como para sistemas basados en SDN. Este trabajo define una arquitectura conocida como software-defined mobile network (SDMN), la cual permite mayor flexibilidad y capacidad de programación, sin depender de otros proveedores de servicios.

SDN proporciona innumerables ventajas en diversos campos de investigación, sin embargo, tiene también limitaciones que deben ser tomadas en cuenta, como se detalla en la siguiente sección.

2.5. LIMITACIONES ACTUALES DE REDES DEFINIDAS POR SOFTWARE

Las redes SDN presentan algunas limitaciones en términos de seguridad, rendimiento, escalabilidad, entre otras, que deben ser tomadas en cuenta antes de ser aplicada en sistemas reales. Entre los principales retos se puede mencionar:

1. **Seguridad:** La separación del plano de datos del plano de control convierte a los elementos de SDN en objetivos vulnerables a ataques maliciosos, tal es el caso de la denegación de servicio del controlador. En este tipo de ataque, si el controlador se ve comprometido se perderá el control de la red. Adicionalmente, se necesita mecanismos de autenticación y diferentes niveles de acceso para el uso adecuado de los recursos de red.
2. **Rendimiento y modelado de una red SDN**: actualmente no existe un acuerdo en la ubicación, el tipo de NOS o el número de controladores que deben ser desplegados en una red SDN.
3. **Integración con redes tradicionales:** si se requiere implementar una red SDN, los dispositivos deben tener soporte para dicha tecnología. Por tanto, es de vital importancia crear mecanismos, protocolos e interfaces que permitan la transición o coordinación entre equipos SDN y los dispositivos de las redes actuales.

2.5.1. MONITORIZACIÓN

El uso de herramientas para monitorizar surge de la necesidad de conocer lo que ocurre en un sistema ya sea en un momento concreto como en intervalos específicos de tiempo. Para llevar a cabo la labor de monitorización, existen numerosos procedimientos y tácticas que se centran en la utilización tanto de sistemas software como en el establecimiento dispositivos físicos destinados a este fin [NOMS14]. Debido al alto coste que presenta la implantación de estos sistemas, existen soluciones alternativas como las que propone SDN.

En este sentido, SDN presenta dos tipos de métodos para realizar la monitorización. El primero de ellos realiza una monitorización activa utilizando la técnica de envío de paquetes adicionales para conocer lo que ocurre en la red. Por el contrario, el segundo método se basa en la monitorización pasiva, en el cual se analiza únicamente el tráfico que circula por la red.

2.5.2. ANÁLISIS

Una vez detectadas las métricas de monitorización, éstas pueden ser analizadas con el objetivo de proporcionar información agregada o correlacionada del estado de la red, tal es el caso de los valores máximos y mínimos de retardo un enlace específico en un intervalo de tiempo. Por otra parte, SDN permite la implementación de técnicas de análisis tradicionales.

2.5.3. VISUALIZACIÓN

Tras realizar el análisis de las métricas y estadísticas en una red SDN, adquiere importancia la presentación de las mismas hacia el usuario final. Para ello existen frameworks (normalmente en versión web) [SOI15] que permiten la visualización del estado de la red en tiempo real, en una gran cantidad de formatos (gráficos y tablas). Sin embargo, la mayoría de estas herramientas son de pago.

3. TECNOLOGÍAS PARA LA MONITORIZACIÓN Y GESTIÓN EN REDES DEFINIDAS POR SOFTWARE

Uno de los principales retos en SDN es garantizar la correcta monitorización, análisis y seguimiento de los elementos y servicios de red. Con la información obtenida, es posible actuar de forma inmediata para subsanar problemas en la red. Esto debido al carácter dinámico y auto gestionable de SDN frente a los sistemas de monitorización tradicionales, en donde se requiere la intervención humana. La monitorización análisis de información en una red SDN tienen como objetivos: i) solucionar problemas esporádicos, ii) proveer capacidad de crecimiento bajo demanda (dispositivos y servicios) y iii) el ajuste dinámico de las políticas de red en tiempo real.

3.1. MONITORIZACIÓN DE REDES DEFINIDAS POR SOFTWARE

Como se ha mencionado anteriormente existen implementaciones (basadas en OpenFlow) de sistemas de monitorización en redes SDN. A continuación, se detallan los sistemas más destacados.

- **Lightweight** [LPL14]**:** Se trata de un mecanismo que tiene como objetivo principal detectar ataques de denegación de servicio (DoS) en sistemas distribuidos. Proporciona datos sobre el tráfico que circula por una red usando para ello métricas basadas en el flujo. Lightweight utiliza un controlador NOX que monitoriza cada cierto tiempo, los dispositivos de red. Los datos que se almacenan en la tabla de entradas de cada conmutador, permiten determinar si se está produciendo un ataque o no. En la figura 3.1 se muestra el funcionamiento de Lightweight.

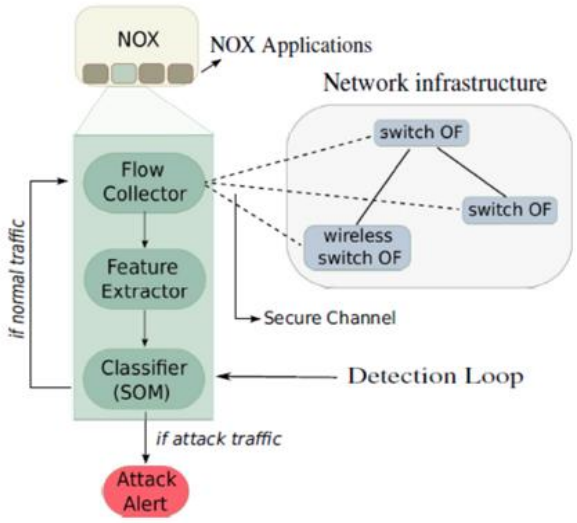


Figura 3.1: Diagrama Lightweight.

- **FRESCO** [MCSS13]**:** Este sistema se caracteriza por la creación de aplicaciones de seguridad en forma de librerías modulares. Ofrece una API de scripts para la creación de estos módulos con el fin de proveer un sistema de monitorización eficiente y que sea capaz de detectar posibles amenazas. El carácter modular de FRESCO (Figura 3.2) facilita la implementación de mecanismos de defensa como sistemas de detección de intrusión (IDS) o firewalls [SAO12].

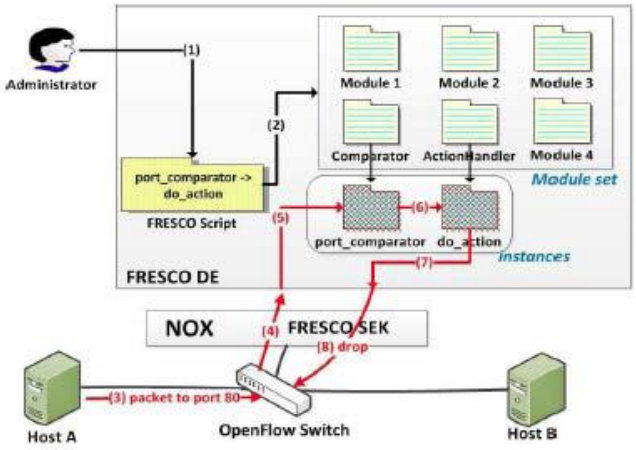


Figura 3.2: Diagrama FRESCO.

- **PayLess** [IEEE14]**:** Es un entorno que monitoriza una red SDN mediante la consulta constante de los elementos de red. PayLess ofrece una API REST de alto nivel, la cual permite abstraer al administrador de red de la implementación del sistema. También proporciona estadísticas en tiempo real del estado de la red. En la Figura 3.3 se muestra la arquitectura de Payless.

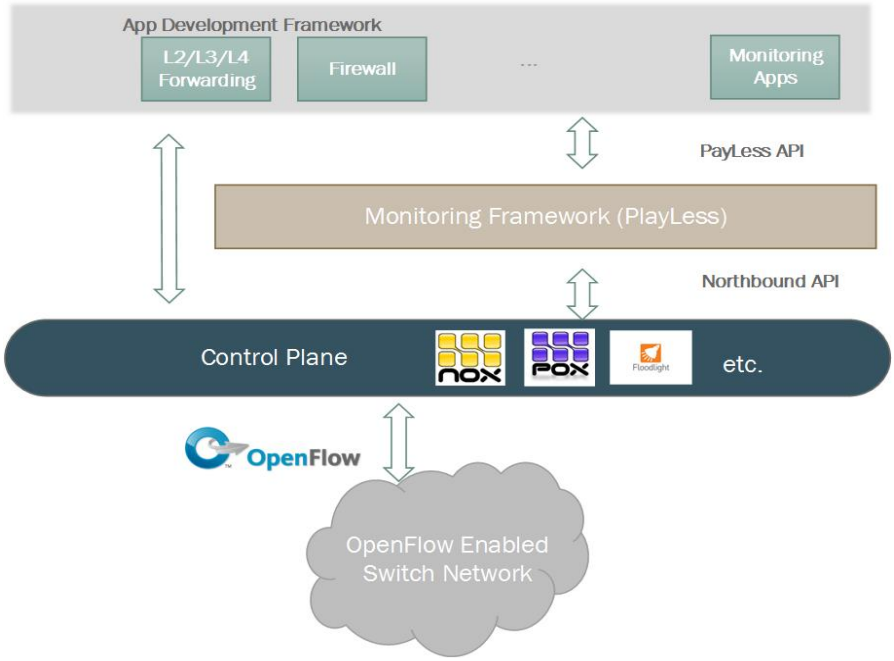


Figura 3.3: Diagrama de PayLess.

- **CloudWatcher** [NSMOFC12]: Gracias a la utilización del controlador permite el despliegue de dispositivos de seguridad en partes concretas de la red, con el objetivo de monitorizar el tráfico de la misma. Para ello, el controlador ejecuta algoritmos de monitorización que se encargan de inspeccionar lo que está ocurriendo en los dispositivos de seguridad. Su ámbito de actuación abarca redes de gran tamaño y dinámicas (Figura 3.4).

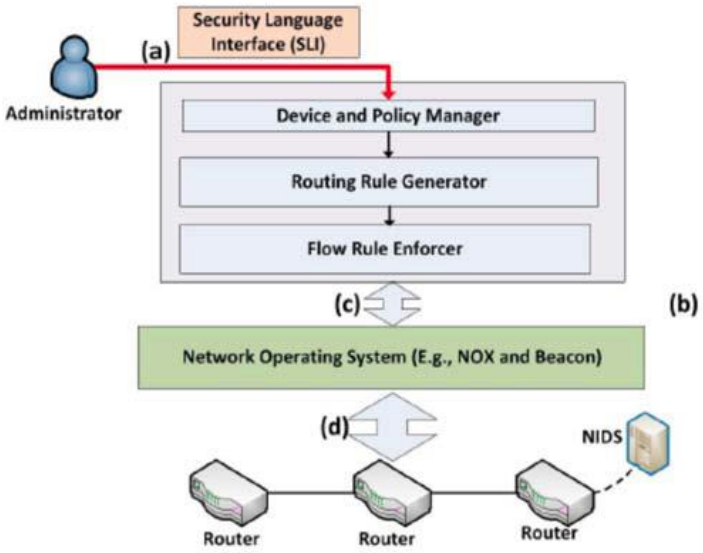


Figura 3.4: Diagrama de CloudWatcher.

- **OpenNetMon** [NOMS14]: este trabajo tiene como propósito principal monitorizar y analizar las distintas características que componen la red. Como ya se ha mencionado, se centra en el estudio y análisis de la calidad de servicio (QoS), la pérdida de paquetes y el retardo de la red. OpenNetMon facilita la verificación de los parámetros medidos y su ajuste dinámico. En la figura 3.5 se muestra la arquitectura de OpenNetMon.

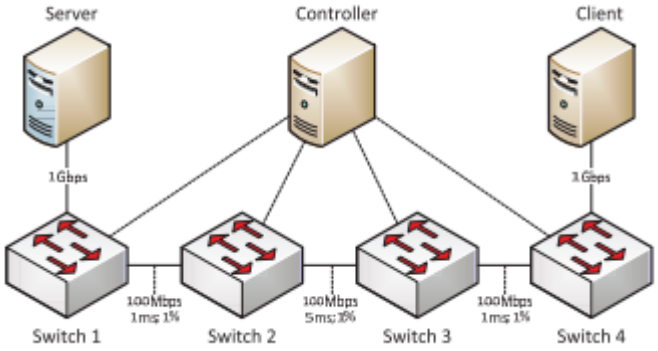


Figura 3.5: Diagrama OpenNetMon.

Además, OpenNetMon presenta estadísticas de rendimiento de las métricas monitorizadas.

- **Manager Interactivo de redes SDN** [HABRZ15]: Este trabajo tiene como objetivo proporcionar un sistema eficiente de monitorización basado en el sondeo. Para esto, se implementa una herramienta visual interactiva (versión web), que ofrece la posibilidad de definir una serie de intervalos configurables. En la Figura 3.6 se muestra la arquitectura propuesta por este trabajo de investigación.

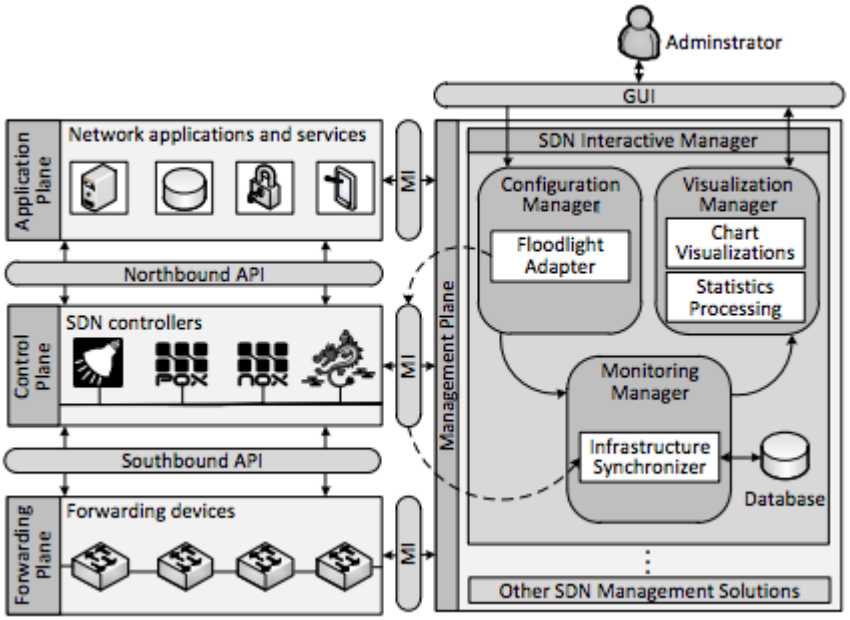


Figura 3.6: Modelo de Arquitectura de MI.

El Manager Interactivo se compone de tres componentes que realizan tareas independientes:

- **Monitorización:** recibe, gestiona y almacena la información del estado de la red en cada momento (enlaces, conmutadores, estadísticas, etc.). Esto es posible gracias a que la información recibida proviene directamente del controlador (Floodlight) el cual expone su interfaz REST.

- **Visualización:** Este componente se encarga tanto del procesamiento de estadísticas como de la visualización de gráficos en tiempo real. Utiliza la información almacenada para mostrar al usuario las siguientes estadísticas: procesos activos, procesos ociosos, datos de los ordenadores, información de conmutadores, cuellos de botella, cantidad de tráfico que circula por la red, etc. Adicionalmente para la aplicación web utiliza D3.js[4] en modo de tablero de mando (dashboard).

- **Configuración:** permite al administrador de red controlar y configurar los distintos parámetros que conforman el sistema, a través, del componente de visualización.

- **MonSamp** [CSIS14]: Este trabajo permite la monitorización SDN a través de la duplicación del tráfico de la red hacia un agente externo. La información recolectada por el agente se envía al controlador SDN, que utiliza un algoritmo el cual optimiza la monitorización modificando la frecuencia de muestreo. MonSamp determina si el dispositivo es capaz de gestionar las tablas de encaminamiento, ya sea añadiendo o quitando elementos (Figura 3.7).

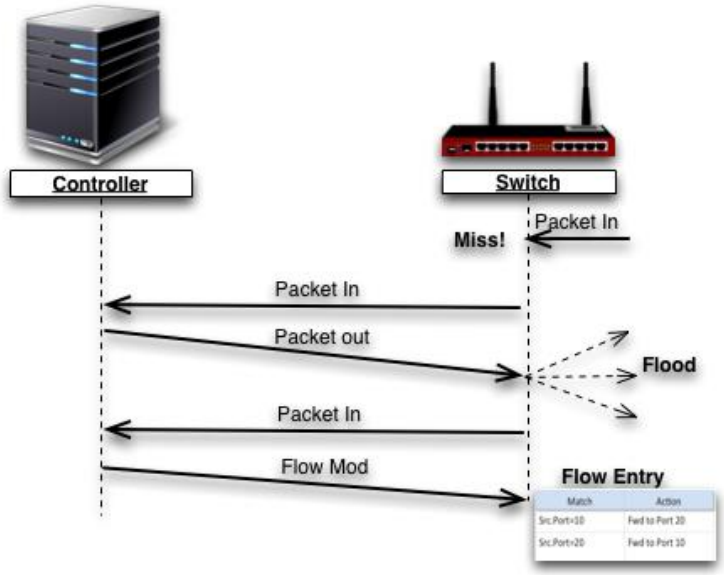


Figura 3.7: Diagrama MonSamp.

[4] "D3.js - Data-Driven Documents." 2015. 27 May. 2016 <https://d3js.org/>

De los sistemas expuestos se deduce que en la monitorización en redes SDN es primordial la necesidad de determinar el estado de la red, preservar su seguridad y analizar las distintas métricas que la componen.

3.2. GESTIÓN Y ANÁLISIS DE LA INFORMACIÓN

Las propuestas expuestas dan una visión global sobre las herramientas desarrolladas para la monitorización y su alcance. A continuación, se detallan algunos requerimientos que deben ser tomados en cuenta en el diseño de sistema de gestión y análisis de la información.

En lo que respecta a la capacidad de almacenamiento de información, es importante destacar que la escalabilidad de un sistema depende en gran medida del tamaño y funcionamiento de la base de datos back-end del controlador [SSHC+13].

Por su parte, las bases de datos basadas en gráficos están siendo adoptadas como atractivas alternativas a las bases de datos tradicionales en diversas aplicaciones, tal es el caso de la computación en la nube [CRS15]. Además, el uso de metadatos para describir cómo los datos se interconectan por medio de estándares de web semántica están ganando cada vez más terreno. En particular utilizan una base de datos escalable Neo4J[5].

Por otro lado, una de las principales metas es la captura de datos en tiempo real de la red (y a gran escala) para su supervisión y posterior análisis. En [LLA14], construyen un sistema de archivos distribuidos aprovechando HDFS[6,] un componente de Hadoop y HBase[7] para la gestión estructurada. Además para el análisis del tráfico capturado, desarrollan un conjunto de programas basados en el potente modelo de programación: MapReduce[8].

[5] "Neo4j: The World's Leading Graph Database." 2009. 22 May. 2016 <http://neo4j.com/>
6 "HDFS Architecture Guide - Hadoop - Apache." 2013. 22 May. 2016 <https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html>
7 "Apache HBase – Apache HBase™ Home." 2011. 22 May. 2016 <https://hbase.apache.org/>
8 Dean, J. "Google Research Publication: MapReduce - Research at Google." 2007. <http://research.google.com/archive/mapreduce.html>

Finalmente, no se pueden descartar el uso de herramientas y componentes de software tradicionales, ya que también han demostrado buenos resultados a nivel aplicación [HABRZ15].

La unión de estos conceptos y la variedad de componentes software permiten que se desarrollen propuestas novedosas que se ajustan a los requerimientos del usuario, como, por ejemplo, en la gestión eficiente del tráfico de la red. En los siguientes apartados, se estudia de forma independiente los principales componentes para el desarrollo de aplicaciones y que en su conjunto forman arquitecturas que permiten la gestión y el análisis de la información.

3.2.1. BASES DE DATOS

Dada la gran cantidad de datos que se pueden obtener a la hora de monitorizar redes definidas por software, surge la necesidad de almacenar los mismos de forma persistente. Hasta el momento, los trabajos relacionados con la materia, se encargan de almacenar las estadísticas monitorizadas en ficheros de texto plano de tipo "log". De esta forma, se deja a un lado la seguridad de la información, ya que el propósito principal era conseguir dicha información. En este punto se puede usar varios tipos de bases de datos, relacionales o no relacionales:

Bases de datos relacionales

Este tipo de bases de datos cumplen con el modelo relacional, el cual es el más utilizado en sistemas donde el tipo de datos varía muy poco. Se tienen una serie de tablas donde se almacena la información y se permite establecer relaciones entre ellas, de tal manera que no se pueden tener varias tablas con el mismo nombre. Cada tabla contiene un número de registros organizados por filas y columnas y las relaciones entre las tablas se realiza mediante claves primarias o claves foráneas. Los datos identificados por la clave primaria son únicos, de tal manera que se garantiza la integridad de la información.

MySQL

MySQL [MSQL1] es uno de los sistemas de gestión de bases de datos relacionales open source más utilizados y está desarrollado bajo una licencia dual GPL/Licencia Comercial por Oracle Corporation. MySQL es utilizado en aplicaciones web, especialmente con la tecnología PHP. Es una base de datos que se comporta de manera muy eficiente en la lectura de información, pero con muchos problemas de integridad cuando existe una concurrencia muy alta (modificaciones). En las aplicaciones web, lo que predomina son las lecturas de datos, habiendo menos concurrencia de modificaciones. Por tanto, MySQL es una base de datos ideal para aplicaciones web.

MySQL es una base de datos aceptada por múltiples plataformas y sistemas operativos y se comporta de manera robusta, ofreciendo una conectividad segura. Para la conexión segura con una base de datos MySQL una de las herramientas más utilizadas es XAMPP. Esta herramienta es un servidor independiente de plataforma que contiene los siguientes elementos:

- X: Puede ejecutarse en cualquier plataforma o sistema operativo
- A: contiene un servidor web **A**pache
- M: utiliza un sistema gestor de bases de datos **M**ySQL
- PP: utiliza los lenguajes de script **P**HP y **P**erl.

Bases de datos no relacionales

Las bases de datos noSQL, se diferencian principalmente de las bases de datos relacionales, en que no requieren esquemas de información fijos y que además escalan horizontalmente. Esto significa que no es un sistema dividido en tablas, sino que la información se guarda de forma íntegra en la base de datos, sin una estructura definida. Por lo tanto, es posible utilizar estructuras de datos sencillas como arrays asociativos y pares clave-valor. Una de las tecnologías más utilizadas para el almacenamiento es el esquema semi estructurado JSON.

MongoDB

MongoDB [MDB1] es una de las bases de datos no relacionales más utilizadas en el mercado actual, ya que tiene muchas características que proporcionan flexibilidad y agilidad a las empresas a la hora de realizar modificaciones en sus aplicaciones. MongoDB se adapta perfectamente a sistemas donde se trabaja con información que cambia constantemente. Entre sus principales características se destacan: i) alto nivel de rendimiento tanto para las lecturas como las escrituras, ii) alto grado de escalabilidad y iii) fiabilidad debido a su capacidad automática de tolerancia a fallos.

3.2.2. REPRESENTATIONAL STATE TRANSFER (REST)

REST es un estilo de arquitectura (no un estándar para el diseño de servicios web), el término fue introducido en la tesis doctoral de Roy Fielding en el año 2000, quien es uno de los principales autores de la especificación de HTTP.

En este estilo de arquitectura la idea de "servicio" como tal desaparece, en su lugar se establece el término recurso. Se puede acceder a este recurso mediante una URI, finalmente la URI entregará una representación del recurso que modificará el estado del cliente. En los últimos años, la utilización de REST se ha incrementado porque es más simple que otras alternativas, tal es el caso de SOAP.

SOAP está orientado a RPC, por lo que tiene que invocar métodos sobre un servicio remoto. Sin embargo, la implementación ya sea de SOAP o REST dependerá de las características del proyecto en el que se encuentre implicado. Para seguir este estilo se debe cumplir con ciertas premisas, por ejemplo:

- Arquitectura Cliente – Servidor: La W3C define servicios web *como un sistema software diseñado para soportar una interacción interoperable entre diferentes equipos de red.* La definición para servicios web propone diferentes tipos de sistemas, siendo el más usado para la comunicación el cliente – servidor, donde la independencia y responsabilidades de cada lado es delimita mediante una interfaz, lo que permite alta flexibilidad. En REST como el único canal de comunicación es el intercambio de mensajes JSON o XML, el cliente tiene total libertad para realizar cambios, y de igual forma el servidor.

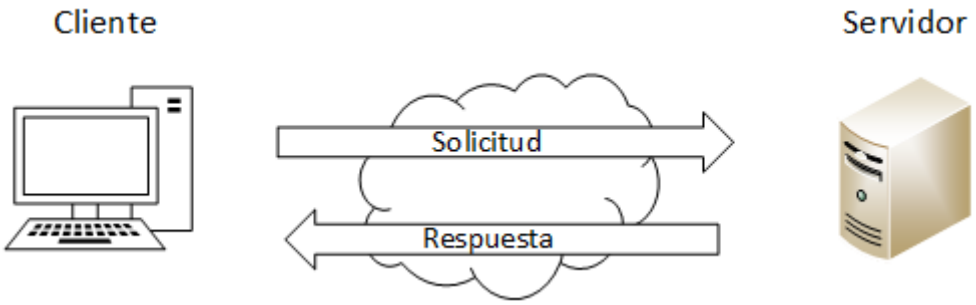


Figura 3.8: Arquitectura cliente-servidor.

- **Sin estado:** Entre dos llamadas cualesquiera, el servicio pierde su estado. Esto constituye uno de los puntos claves de REST ya que permite la escalabilidad.
- **Cacheable:** Es recomendable mantener en caché la petición que un usuario haya realizado, ya que favorece el rendimiento y la escalabilidad. En la cabecera de HTTP tiene la etiqueta "Cache-Control".
- **Servicios uniformes:** Todos los servicios REST invocan métodos uniformes: GET, POST, PUT, DELETE.
- **Arquitectura en capas:** Como no guarda el estado, permite que sea escalable, siendo una oportunidad para poner un balanceador.

  Un balanceador permite que la redirección de la petición a otro servidor sea un proceso transparente para el cliente, es decir que el usuario no encuentra diferencia en la respuesta.

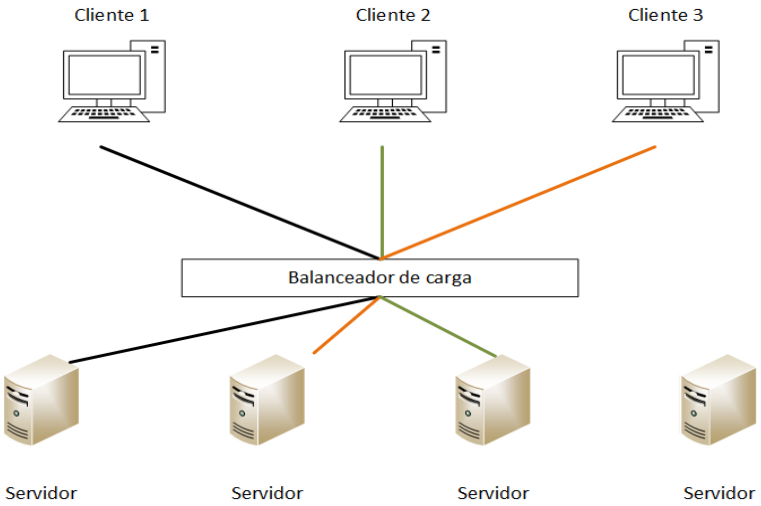


Figura 3.9: Balanceador de carga.

Una API REST es una interfaz software a software, es decir, no existe intervención del usuario. Esta puede ser consumida desde una aplicación web, móvil u otras aplicaciones software, la cual es efectiva gracias a la cohesión con HTTP. Este hecho permite realizar acciones diferenciadas a través de sus distintos verbos, y que cualquier cliente pueda interactuar sin requerir una configuración específica. REST se ha convertido en una propuesta atractiva para el desarrollo software, sin embargo, para establecer un API REST se debe cumplir con niveles de calidad, los cuales son:

1. **Correcto uso de URIs**

   Una URI permite identificar de forma uniforme un recurso para su posterior acceso. Las URIs para HTTP (con una pequeña diferencia con las conocidas URLs) solicitan recursos para realizar ciertas acciones directamente sobre el servidor.

{protocolo}://{dominio o hostname}[:puerto]/{ruta del recurso de forma jerárquica }?{filtrado para la consulta}

Figura 3.10: Estructura de una URI.

Existen algunas reglas para establecer el nombre de la URI para un recurso:

- Evitar el uso de verbos ya que no es una acción
- Deben ser únicas
- Deben ser independientes del formato
- Deben mantener una jerarquía lógica

2. **Correcto uso de HTTP**

Las acciones que una URI puede realizar, son bajo los verbos que aporta HTTP. Para el desarrollo de una API REST se debe tener en cuenta ciertos aspectos básico de HTTP:

- **Métodos HTTP**

  Para la manipulación de los recursos, HTTP establece los siguientes verbos:

  - GET: Solicita consulta y lectura de recurso
  - POST: Crear recursos
  - PUT: Edición de recursos
  - DELETE: Eliminar un recurso del servidor

  Cuando se realiza una petición mediante estos verbos, en realidad se trabaja sobre un esquema o representación ya que no se manipula la base de datos directamente (u otro sistema de almacenamiento).

3. **Códigos de estado**

Para conocer el estado de la lectura, modificación o creación de un recurso, HTTP define distintas cabeceras. El estado indica si la petición fue aceptada o rechazada mediante el intercambio de mensajes (códigos). En la tabla 3.1 se muestra los códigos de estado de HTTP.

Estado	Mensaje
200	Comunicación correcta.
201	Nuevo recurso creado correctamente.
204	Recurso eliminado correctamente.
304	El cliente no puede utilizar los datos en caché.
400	Petición incorrecta.
401	El recurso requerido necesita autorización.
403	El servidor entendió la solicitud pero no se ha permitido el acceso.
404	No hay recursos detrás de la URI .
500	Error interno de servidor.

Tabla 3.1: Códigos de estado HTTP.

4. **Aceptación de tipo de contenido**

   HTTP permite especificar en qué formato se quiere recibir el recurso, el API mostrará el primer formato disponible y en el caso que no se pueda resolver, éste deberá enviar un mensaje de error. Por ejemplo, en la cabecera HTTP se indica mediante *Accept* el tipo de contenido o MIME que se espera de la respuesta y mediante Content-Type, indica el tipo de contenido que se envía.

5. **Implementar Hypermedia**

   El objetivo es informar en la respuesta de la petición, las URIs para acceder a otros recursos que tengan relación con el solicitado.

Las representaciones de los recursos obtenidos mediante REST, puede tener formato XML o JSON, siendo el último el más utilizado debido a la flexibilidad y agilidad en la transferencia.

Diferencias entre REST y SOAP

REST no necesita de una capa adicional para transmitir datos como es necesario en SOAP. Suele compararse REST con servicios web, sin embargo, se debe tener en cuenta que REST es un estilo y los servicios web son sistemas software. Se puede ver en la siguiente Tabla 3.2 las características, ventajas y posibles desventajas entre REST y SOAP.

	REST	**SOAP**
Características	Las operaciones se definen en los mensajes. Una dirección única para cada instancia del proceso. Cada objeto soporta las operaciones estándares definidas. Componentes débilmente acoplados	Las operaciones son definidas como puertos WSDL. Dirección única para todas las operaciones. Múltiples instancias del proceso comparten la misma operación. Componentes fuertemente acoplados.
Ventajas declaradas	Bajo consumo de recursos. Las instancias del proceso son creadas explícitamente. El cliente no necesita información de enrutamiento a partir de la URI inicial. Generalmente fácil de construir y adoptar.	Fácil (generalmente) de utilizar. La depuración es posible. Las operaciones complejas pueden ser escondidas detrás de una fachada. Envolver APIs existentes es sencillo. Incrementar la privacidad. Herramientas de desarrollo
Posibles desventajas	Gran número de objetos. Manejar el espacio de nombre (URIs) puede ser engorroso. La descripción sintáctica/semántica muy informal (orientada al usuario). Pocas herramientas de desarrollo.	Los clientes necesitan saber las operaciones y su semántica antes del uso. Los clientes necesitan puertos dedicados para diferentes tipos de notificaciones. Las instancias del proceso son creadas implícitamente.

Tabla 3.2. Características REST y SOAP.[9]

[9] ELP-DSIC-UPV, RNM. "REST vs Web Services - Upv." 2009. <http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>

3.2.3. LENGUAJES DE PROGRAMACIÓN Y FRAMEWORKS

En el desarrollo de alto nivel, existen diferentes lenguajes que se adaptan según las necesidades del proyecto. Existen lenguajes de programación orientados para el desarrollo web, el desarrollo de servicios, orientados a los videojuegos, entre otros. Para las diferentes áreas existen soluciones en cualquier lenguaje de programación, por ejemplo: PHP, Python, Ruby, .NET, Java, Node/JavaScript. De estos PHP y Ruby están vinculados básicamente en la web, Python se adapta a cualquier ámbito, .NET y Java son importantes en el ámbito empresarial y por último Node/ Javascript está pensado para facilitar el desarrollo de aplicaciones basadas en red (rápidas y fiables). En los próximos apartados, se presta especial atención a los lenguajes de programación Java y Python.

Java

Java EE define una infraestructura común básica para el acceso a bases de datos, gestión de persistencia, control de seguridad, gestión de transacciones, entre otros, por lo cual permite la separación clara entre presentación (interfaz), modelo (lógica) de negocio y datos. En la Figura 3.11 se muestra los componentes de Java EE 7.

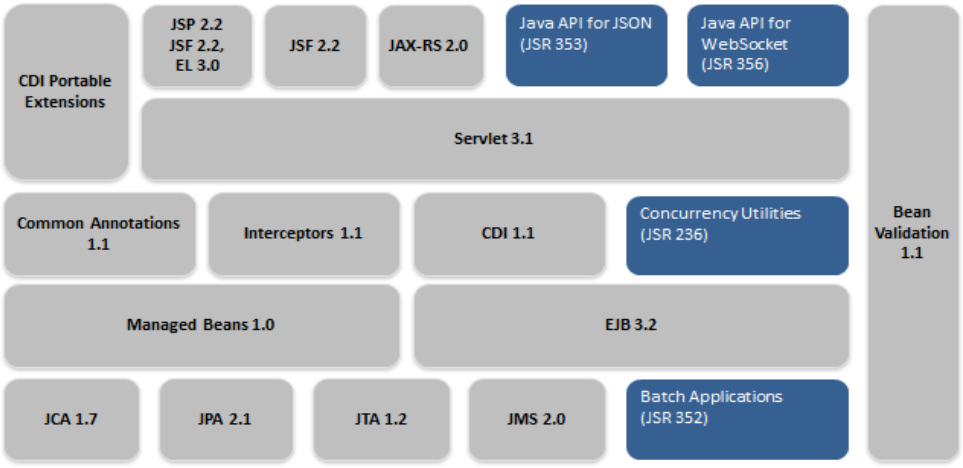


Figura 3.11: Esquema general Java EE 7.

Python

Python es un lenguaje de programación orientado a objetos, clara y potente, comparable a Perl, Ruby, Scheme o Java [BGOPWEB]. Algunas de las características notables de Python son:

- Utiliza una sintaxis elegante, por lo que los programas se escriben y leen fácilmente.
- Es ideal para el desarrollo de prototipos y otras tareas de programación ad-hoc, sin comprometer la capacidad de mantenimiento.
- Tiene incorporado una biblioteca estándar que soporta muchas de las tareas de programación comunes, tales como la conexión a los servidores web, la búsqueda de texto con expresiones regulares, leer y modificar archivos, entre otras.
- Potencia para el análisis y procesamiento de datos debido a la flexibilidad.

Spring

Spring se centra sobre "la fontanería" de las aplicaciones, de modo que los equipos se puedan centrar en la lógica de negocio a nivel de aplicación, sin tener que depender de entornos de desarrollo específicos [S13WEB], Spring puede eliminar la proliferación del patrón singleton (visto en muchos proyectos), también permite eliminar la necesidad de usar una variedad de ficheros de propiedades en diferentes formatos. Para lograr esta simplificación, Spring usa la inversión de control y la inyección de dependencias, con esto se consigue que actúe como un contenedor encargado de generar todos los componentes necesarios para el desarrollo de una aplicación (clases, instancias, dependencias, etc.).

Uno de los aspectos principales de Spring es que proporciona la integración con distintos tipos de servicios situados en múltiples capas de una aplicación. Estos módulos se pueden aprovechar para construir diferentes aplicaciones escalables y flexibles. Adicionalmente, Spring ofrece una serie de proyectos dependiendo de las necesidades de la infraestructura de una aplicación (configuración de seguridad, aplicaciones, big data, etc.). En la Figura 3.12 se presentan los principales proyectos Spring [S13WEB].

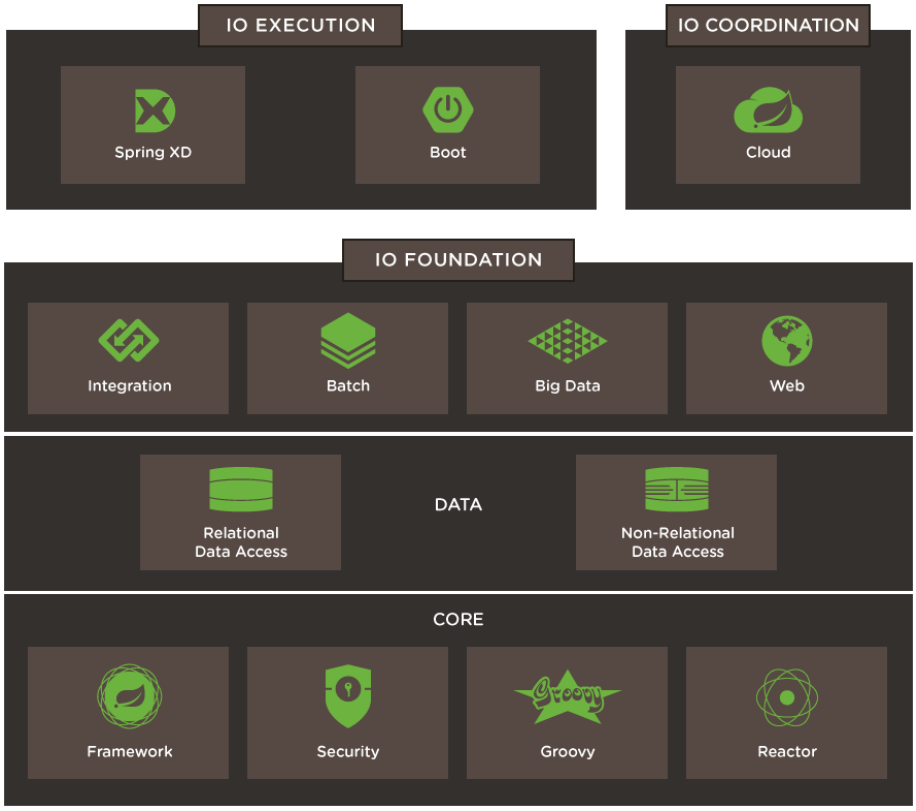


Figura 3.12: Principales proyectos Spring.

Spring Framework

Rod Johnson en el año 2000, mientras escribía el libro Expert One-on-one J2EE Design And Development (Programmer to programmer), amplió su código para sintetizar su visión acerca de cómo las aplicaciones que trabajan con varias partes de la plataforma J2EE podían llegar a ser más simples y más consistentes. A partir de esto nace Spring framework [SFWEB]. Spring framework proporciona una programación entendible y un modelo de configuración para las aplicaciones modernas basadas en Java, sobre cualquier clase de plataforma de desarrollo. En la Figura 3.13, se muestran los componentes que conforman el framework Spring.

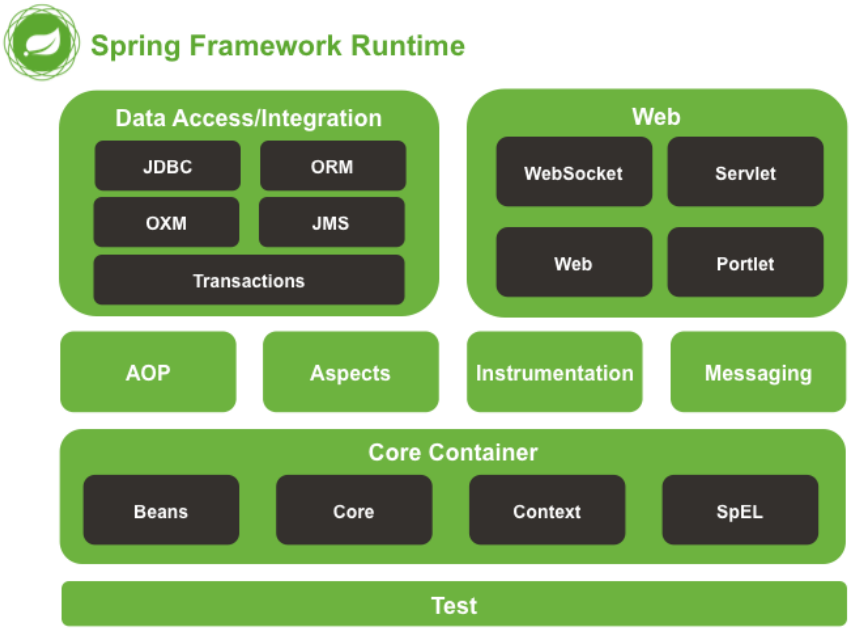


Figura 3.13: Arquitectura Framework Spring.

- **Spring Core:** Provee la funcionalidad esencial del framework, está compuesta por el BeanFactory, el cual utiliza el patrón de Inversión de Control (IoC) y configura los objetos a través de Inyección de dependencias.
- **Spring Context:** "El módulo BeanFactory del núcleo de Spring es lo que lo hace un contenedor, y el módulo de contexto es lo que lo hace un framework". Proporciona métodos de acceso a los objetos del modo tradicional en otros frameworks. Permite internacionalización, propagación de eventos, recarga de recurso y la creación transparente de contextos.
- **Spring DAO:** Proporciona una capa abstracción de JDBC, que elimina la necesidad de manejar código asociado a JDBC y con los errores específicos de cada base de datos.
- **Spring ORM:** Proporciona una capa de integración con framework de mapeo objeto-relación como Hibernate, JPA, JDO, etc.
- **Spring AOP:** Proporciona una implementación de programación orientada a aspectos, compatible con la alianza AOP, para definir puntos de acceso, interceptores, proxis dinámicos, etc.

- **Spring Web:** Proporciona funcionalidades básicas de integración web como upload de ficheros en varias partes, inicialización del contenedor IoC mediante servlet listeners y un contexto de aplicación web.
- **Spring MVC:** Proporciona una implementación del patrón Modelo - Vista - Controlador, para el desarrollo de aplicaciones web.

Spring Boot

Spring Boot [SBSSWEB] tiene como objetivo facilitar la creación de aplicaciones y servicios. Es recomendado para aquellos usuarios que no hayan tenido una experiencia con Spring. En Spring Boot el principal beneficio es la configuración de recursos en función de la etiqueta que encuentra en una clase. Por otro lado, también permite empaquetar en un archivo JAR un servidor completo Tomcat y finalmente la facilidad para la instalación y despliegue de una aplicación o servicio.

Spring Data

Spring Data [SDWEB] es un módulo de Spring que permite el acceso a varios marcos de acceso a datos en la construcción de aplicaciones, tanto para bases de datos relacionales y no relacionales. Este es un proyecto global que contienen muchos subproyectos que son específicas de una base de datos determinada. En la actualidad la gama de nuevas tecnologías de acceso de datos tales como NoSQL y Hadoop, son relativamente fáciles de construir con este módulo.

Spring Actuator

Spring Actuator es un subproyecto de Spring Boot que permite obtener las características de la aplicación puesta en producción.

3.2.4. WEB

El diseño de sitios web, enfoca su campo de actuación principalmente en garantizar la experiencia que tiene un usuario que interactúa con interfaces web. El diseño web, se divide en tres etapas:

1. Visualización de la información que se desea mostrar.
2. Organización y niveles de navegación.
3. Posición en búsqueda.

Una vez definidas las etapas del diseño web, las principales tendencias y herramientas que se utilizan son:

- **Bootstrap:** Se trata de un framework CSS de diseño responsive desarrollado por Twitter y liberado bajo una licencia MIT en el año 2011. Tiene como características principales:
    - Diseño por columnas.
    - Soporte casi completo para HTML5 y CSS3.
    - Compatibilidad con la gran mayoría de navegadores.

- **ASP.Net:** Se trata de un entorno de desarrollo web que reúne y agrupa todo lo necesario para la creación de aplicaciones web. Pertenece a Microsoft y utiliza el sistema de "code behind" para ayudar a los desarrolladores a separar la interfaz visual del código de control y favorecer la abstracción.

- **JavaScript:** Se trata de un lenguaje de programación dinámico incluido en los diseños web actuales. Se implementa en el lado cliente para aumentar la experiencia del usuario y permite hacer y recibir peticiones gracias a que se puede combinar con otras tecnologías como AJAX.

- **AJAX:** Se trata de un mecanismo utilizado en el desarrollo web para conseguir que el navegador trabaje de una forma más eficiente y rápida gracias a que la carga de los datos necesarios para mostrar e interactuar con el usuario se realizan en segundo plano, eliminando la necesidad de la recarga de la página.

- **JSON:** Se utiliza para el intercambio de datos, y, su ámbito de actuación son los distintos navegadores actuales. Gracias a las continuas mejoras en su implementación presenta una estructura ligera y sencilla.

Por otro lado, existen herramientas capaces de presentar una gran cantidad de información de forma gráfica en el ámbito web (mapas, tablas, calendarios, etc.). Para ello se utilizan librerías, normalmente desarrolladas en JavaScript, que interpretan peticiones JSON y las adaptan a la visualización. Las herramientas más representativas de las mencionadas son: HighCharts y D3JS.

3.3. GESTIÓN DE REDES DEFINIDAS POR SOFTWARE

Tras mencionar las numerosas tecnologías que existen en la actualidad sobre los distintos campos enfocados tanto en la monitorización tradicional como en la monitorización de redes SDN, se concluye que no existen arquitecturas desarrolladas que cubran las necesidades de los administradores de red.

Por ejemplo, una implementación que se aproxima a cubrir las necesidades de monitorización se detalla en el punto 3.2 con el Manager Interactivo de Monitorización para redes SDN. Esta implementación, tiene como limitación la falta de un servicio dedicado a la petición y respuesta que realiza un cliente (normalmente web). Es por ello que no proporciona la posibilidad de tener múltiples tecnologías para la implementación del lado cliente.

Por este motivo, en este proyecto se desarrolla una arquitectura modular, que pretende cubrir no solo esta necesidad sino también ofrecer flexibilidad en la incorporación de los múltiples gestores de bases de datos existentes. La propuesta de arquitectura se detalla en el capítulo que se describe a continuación.

4. ARQUITECTURA PARA LA MONITORIZACIÓN DEL ESTADO DE LOS ENLACES EN REDES SDN/OPENFLOW

4.1. INTRODUCCIÓN

Con el fin de cubrir las necesidades que se presentan en la actualidad en el campo de la monitorización de las redes SDN, se presenta en este proyecto una arquitectura capaz de capturar, almacenar, visualizar y analizar las distintas métricas que se obtienen de los componentes que forman una red SDN. Así mismo, la arquitectura busca flexibilidad y alto nivel de abstracción capaz de adaptarse a nuevas tecnologías de monitorización.

La arquitectura propuesta se describe en la Figura 4.1. En la misma se utilizan diferentes módulos funcionales organizados en diferentes capas: capa de monitorización, capa de transferencia y capa de visualización. En el presente capítulo se analizará cada una de los diferentes módulos de la arquitectura.

4.2. DESCRIPCIÓN DE LA ARQUITECTURA

El presente proyecto permite a los administradores de red visualizar y analizar el comportamiento del tráfico y el estado de los enlaces de los elementos dentro de una red SDN. Con este objetivo, se ofrece una herramienta de visualización con componentes gráficos y en tiempo real. Así mismo, gracias a la abstracción de la arquitectura, los desarrolladores tienen la posibilidad de instaurar sus propias herramientas con los datos que el API les proporciona, ya que los datos tienen un formato que son admitidos en cualquier tipo de tecnología. Toda esta arquitectura se basa en diferentes capas, entendiendo capas como un módulo o conjunto de módulos que ejecutan una tarea específica como se muestra en la Figura 4.1.

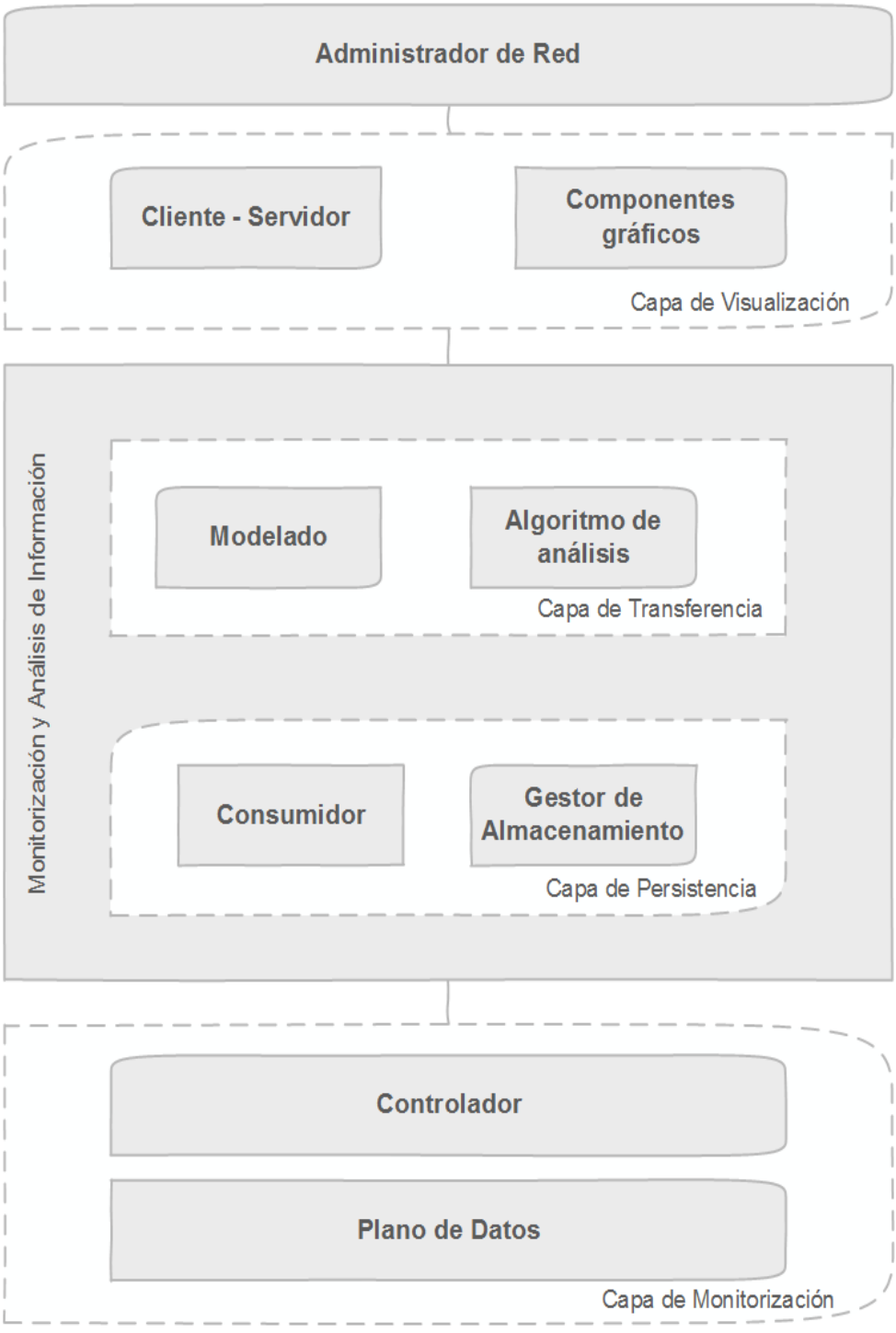


Figura 4.1: Arquitectura propuesta.

4.2.1 CAPA DE MONITORIZACIÓN

La capa de monitorización es la encargada de obtener las métricas de los elementos de red y el estado de los enlaces de la infraestructura. Así mismo, la información recibida es organizada en función del componente de red SDN, las métricas obtenidas y el instante de tiempo concreto. Esta información es enviada a capas superiores para su posterior procesamiento y análisis. La capa de monitorización se divide en dos módulos:

1. **Plano de Datos:** Son todos los elementos físicos y componentes de software básicos encargados del movimiento de información a través de la red. Su función se centra en recibir los paquetes de información por un puerto de entrada, reconocer su cabecera, buscar la correspondiente entrada en la tabla de flujos y re-enviarlos por un puerto de salida. En redes SDN, el procesamiento y búsqueda del camino entre fuente y destino es tarea del plano de control o controlador SDN. El medio de comunicación entre el plano de datos y plano de control es el protocolo OpenFlow.

2. **Controlador:** El controlador cumple múltiples funciones. En primer lugar, recibe los mensajes del plano de datos y abstrae la topología de la red. Esta información incluye el número de dispositivos (switch A, switch B), número de enlaces (switch A-puerto 1) y las conexiones entre las mismas (Switch A - puerto 1 - Switch B - puerto 2). Así mismo, el controlador establece los diferentes caminos entre fuente y destino de los paquetes ya que mantiene las diferentes tablas de flujo de los elementos de la red. Por último, en el presente proyecto se ha ampliado las funciones del controlador para que pueda obtener métricas del estado de los enlaces en la red SDN. Las principales métricas que calcula el módulo de monitorización son: tasa de transmisión, tasa de pérdidas y retardo.

- **Tasa de transmisión:** Es un valor que se calcula mediante el envío de peticiones de puertos a los switches y lee las respuestas de los mismos. Las peticiones son enviadas a través del controlador, el cual envía el mensaje "OFPT_STATS_REQUEST" a los switches durante un período de tiempo de monitorización. El switch responde con un mensaje de tipo "OFPT_STATS_REPLY". El controlador guarda la información del switch y el puerto que le envía la respuesta. La diferencia de datos leídos entre los instantes de tiempo inicial y final de monitorización es la tasa de datos enviada.

- **Tasa de pérdidas:** De una forma parecida a la tasa de datos, se calcula la tasa de error. Con la misma información guardada del mensaje "OFPT_STATS_REPLY" de respuesta. Se consulta en el controlador el enlace que corresponde a la clave switch origen - puerto origen - switch destino - puerto destino, se consulta la tasa de datos enviados y la tasa de datos recibidos. La diferencia sería la tasa de error que se ha dado en el mencionado enlace.

- **Retardo:** Retardo en milisegundos asociado al tráfico de un enlace. El controlador encapsula el instante de tiempo en un paquete que se envía a través del enlace y le pide a los switches que respondan al paquete. Cuando el controlador recibe el paquete de respuesta, compara el instante de tiempo que se encapsuló, con el instante de tiempo actual. La diferencia de ambos instantes de tiempo es el retardo que se ha producido en el mencionado enlace.

Las métricas calculadas ayudan a establecer con claridad la situación actual de la red y el estado de los enlaces, para lo cual necesitan ser guardadas y analizadas a lo largo del tiempo.

4.2.2 CAPA DE PERSISTENCIA

En la capa de persistencia, las diferentes métricas enviadas por el controlador SDN son almacenados, previo un proceso de comprobación de integridad. Con este fin, este nivel se divide en los siguientes módulos:

1. **Consumidor:** Captura e instancia los datos proporcionados por la capa de monitorización. Para ello se dividen los datos dependiendo de los distintos enlaces que forman la red SDN y sus métricas, de tal forma que cada tupla forma un objeto. La tupla se define de la siguiente manera:

   (Switch origen, puerto origen, switch destino, puerto destino,
   tasa de datos, tasa de errores, retardo, instante de tiempo)

   De esta forma quedan identificados todos los datos que se obtienen de cada enlace e instanciados en un único objeto que se envía al sistema gestor de bases de datos.

2. **Gestor de almacenamiento:** Las instancias obtenidas se almacenan en un sistema gestor de almacenamiento, de forma que cobra importancia el instante de tiempo en el que fueron generadas. Es una característica que garantiza la consistencia de los datos porque se evita la duplicidad de los mismos. Como se ha comentado en el punto anterior, cada fila de la base de datos va a contener una tupla dividida en columnas, que correspondan a la información de un enlace en un instante de tiempo determinado.

Al abstraer la capa de persistencia en la arquitectura, se consiguen una serie de características que hacen que el almacenamiento sea más estable y de mayor rendimiento:

- Permite que la base de datos se encuentre en otra máquina física, de tal forma que se le pueden asignar los recursos necesarios a misma.
- Reduce la carga del consumidor, provocando un mayor rendimiento tanto en la base de datos como en la capacidad de petición de datos del mismo.
- Permite que se pueda establecer cualquier sistema gestor de bases de datos, tanto relacionales como no relacionales, simplemente cambiando la máquina que contiene al gestor. Esto es posible gracias a que el consumidor también puede conectar con cualquier sistema gestor de bases de datos.

4.2.3 CAPA DE TRANSFERENCIA

En la capa de transferencia, los módulos que se presentan sirven de base para garantizar la disponibilidad de los datos en la capa de visualización.

Con la información de la abstracción de una red almacenada por la capa de persistencia, es una necesidad recoger estos datos para su estudio. En situaciones normales, bastaría con realizar consultas directamente a la base de datos, pero esto supone un peligro para la integridad de los datos almacenados. Además, esta capa permite acceder al sistema de una forma conocida, uniforme e independiente. Con esto se indica que todas las aplicaciones se integran con la capa de persistencia de la misma manera. Tener un API permite integrar una capa más de seguridad, el cliente no tiene acceso directamente a la base de datos, sino a lo que la configuración del API le permita.

Ejemplo de esta estructura es la revolución IoT, el cual sienta sus bases en la comunicación HTTP a través de APIs, siendo esto considerado una gran ventaja para el producto final.

La capa de transferencia se divide en:

1. **Modelado:** Se encarga de tratar los datos recibidos de la capa de persistencia. Su función principal es la elaboración de un nuevo esquema que satisface las peticiones realizadas por un cliente (API). Estas peticiones se filtran por un conjunto de parámetros facilitados por el cliente, como, por ejemplo: enlaces, switches o puertos.

   Esta capa cumple con una de las bases fundamentales en la gestión y transferencia de la información, es decir, aunque no exista persistencia se mantiene la integridad de la información a nivel de aplicación.

   Por otra parte, tener una API permite establecer la disponibilidad de los datos de la capa de persistencia. Otro factor importante es cumplir con las tendencias actuales en el desarrollo de aplicaciones, ya que no siempre será una aplicación web, por lo cual el diseño de esta capa está pensado para no generar problemas en el caso que el cliente desee desarrollar aplicaciones móviles.

La independencia que presenta la arquitectura cliente - servidor, da oportunidad a los clientes (o equipo de desarrollo) a mostrar la información como mejor le parezca, independientemente de la tecnología a utilizar. Mientras, en el lado del servidor, éste debe seguir enviando los datos requeridos por los clientes sin importar las migraciones de lenguajes o actualizaciones que se puedan realizar, cumpliendo así con "el acuerdo" establecido cuando el cliente decidió utilizar los datos que el API pone a su disposición.

2. **Análisis:** El conjunto de datos que se puede obtener de la capa de persistencia se utiliza para el análisis de las métricas de la red. Este módulo facilita a los administradores el uso de diferentes algoritmos y herramientas de análisis de información. Dependiendo de las necesidades del operador se pueden incluir herramientas como: Weka, MapReduce, Orange, RapidMiner, JHepWork, etc. Como prueba de concepto, en este proyecto se ha desarrollado un algoritmo simple que compara los datos que se van almacenando con la media y máximos históricos del conjunto de datos.

   El uso de herramientas como las mencionadas anteriormente, proveería de una fortaleza aún mayor a la arquitectura. El análisis de datos es cada vez más importante para lograr los objetivos de negocio de las organizaciones. Además, estas herramientas ofrecen un rendimiento en tiempo real, lo cual lleva a poder tomar decisiones al instante. En definitiva, se hace necesario realizar un estudio de los datos que se disponen para poder incrementar su valor, al mismo tiempo que se pueden descubrir claves nuevas buscando correlaciones entre ellos.

4.2.4. CAPA DE VISUALIZACIÓN

En la capa de visualización se facilita a los administradores, la comprensión de la información del estado de la red por medio de una interfaz gráfica y amigable. En otras palabras, se presentan en forma de gráficos, las distintas métricas monitorizadas en la red en un instante de tiempo determinado. Es necesario destacar, que, gracias a la universalidad utilizada para el diseño y representación de esta capa, es posible que cualquier cliente web forme parte de las misma, tanto de forma individual como colectiva.

La capa de visualización se divide en los siguientes módulos:

1. **Cliente-Servidor:** El cliente se encarga de realizar continuas peticiones “GET” con los parámetros deseados a la capa de transferencia.
2. **Componentes gráficos:** Una vez obtenida la información solicitada en un esquema, se trata según las necesidades de los siguientes perfiles de cliente:
    - **Administrador:** Como se ha comentado anteriormente se proporciona una interfaz web compuesta por los siguientes elementos:
        - Gráfico en tiempo real: Se presentan las métricas deseadas en función del tiempo.
        - Gráficas estáticas: Dependiendo de un número establecido de datos se muestra el estado de las distintas métricas.
    - **Desarrollador:** Como la información se presenta en un esquema JSON unificado, se favorece la independencia del desarrollador en el uso de las tecnologías.

4.3. IMPLEMENTACIÓN DE LA ARQUITECTURA

En este capítulo se describe la implementación de la arquitectura propuesta en la sección anterior. La implementación sigue los principios de independencia y modularidad.

Para alcanzar la funcionalidad deseada se parte de las siguientes tecnologías para los diferentes módulos en las capas de la arquitectura:

1. Mininet (Plano de Datos)
2. Python (Algoritmo de análisis de la información)
3. Floodlight (Plano de Control)
4. Spring Boot (Servicios REST)
5. MySQL (Sistema Gestor de Base de Datos)
6. Desarrollo Web: Bootstrap, JavaScript, AJAX, JSON
7. Herramienta de visualización: HighCharts

Cabe destacar que para obtener estadísticas de monitorización fiables de una red SDN, se parte de una implementación realizada por el grupo GASS de la UCM [ACETV14] y probada del Plano de Datos y del Plano de Control. Esta implementación contiene modificaciones que le han sido añadidas para completar su funcionalidad, como es el hecho de obtener el instante de tiempo en el que se obtienen los datos monitorizados.

Finalmente, los objetivos perseguidos con la implementación de esta arquitectura son:

- Consumir los recursos proporcionados por el API REST de Floodlight, de una topología generada por Mininet.
- Insertar en una base de datos, los recursos que proporciona la implementación de la que se parte.
- Implementación de un API REST que se encargue de proveer los recursos necesarios para las solicitudes que realizan los clientes.
- Desarrollar una aplicación web que permita visualizar el estado en el que se encuentra la red.

Para la implementación de la capa de persistencia y la capa de transferencia se considera a Spring Boot como una buena opción, debido a la facilidad con la que permite desarrollar en Java. Por otro lado, interesa de Spring la flexibilidad de las librerías que proporciona, así como la escalabilidad a niveles de BigData o despliegue de micro servicios en la nube, entre otros. El flujo de trabajo para lograr los objetivos mencionados anteriormente se muestra en la Figura 4.2.

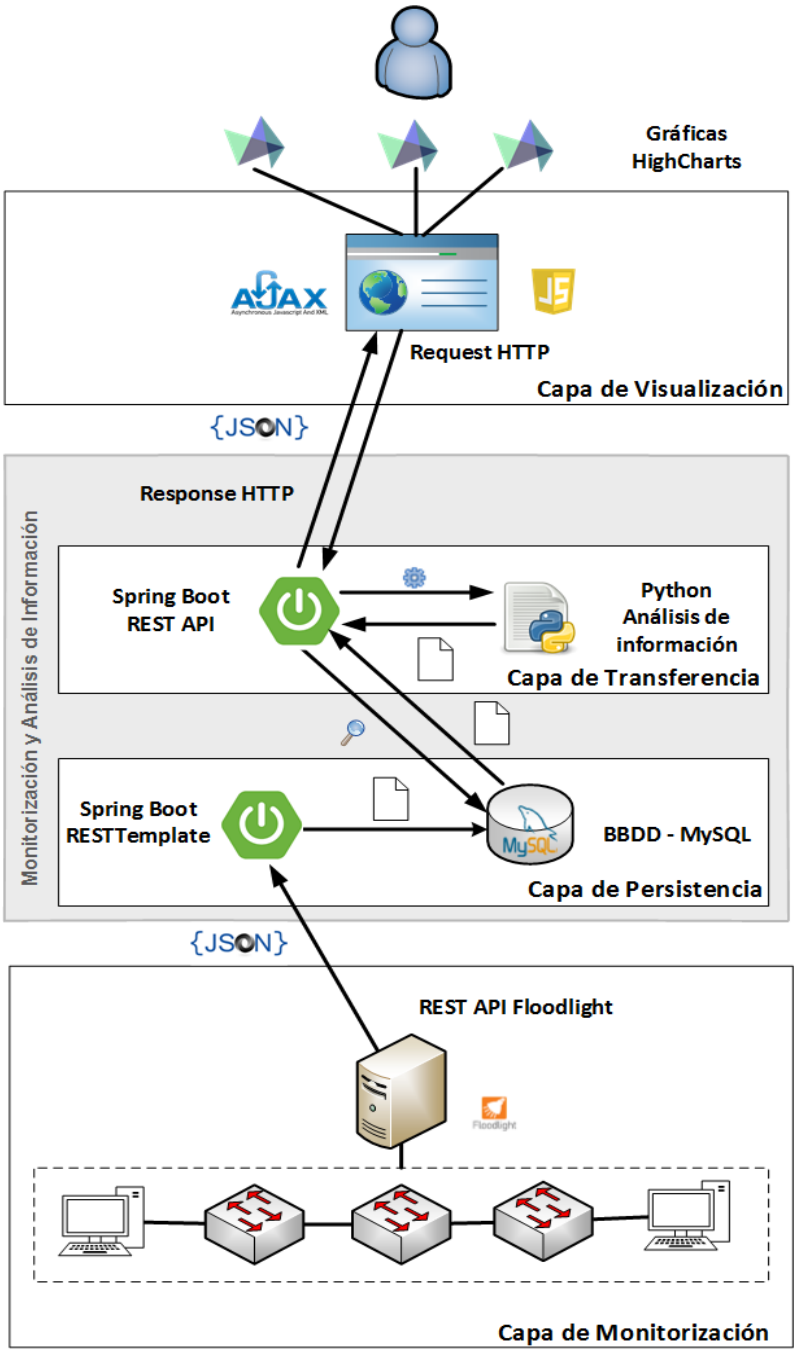


Figura 4.2: Implementación de arquitectura propuesta.

4.3.1. CAPA DE MONITORIZACIÓN

La Capa de Monitorización, se encarga tanto de la creación como de la abstracción de la red. Su objetivo principal es conocer las características de los enlaces que la componen, mediante la monitorización. Las métricas que se analizan en este proyecto son: *retardo, tasa de pérdida y tasa de transmisión.*

Para la implementación se utiliza como sistema base el desarrollado para el Algoritmo de Calidad de Experiencia para Transmisiones de Video en Redes Definidas por Software [ACETV14]. Este sistema se compone de los siguientes módulos:

Mininet

Se utiliza para la creación de las topologías que se van a utilizar y analizar en el desarrollo de esta arquitectura. Gracias a las ventajas de virtualización, Mininet es capaz de emular en un único host, toda una topología de comportamiento similar a una topología de red físico. De igual manera, Mininet es capaz de generar retardos y cambios en el ancho de banda de uno o varios enlaces. Para emular una red y comprobar la funcionalidad de la arquitectura, se utiliza Mininet para crear la topología y se modifica un host virtual de forma que se envía un archivo de vídeo desde un host (servidor) a otro (cliente) usando el protocolo RTP. El objetivo es difundir una carga de tráfico elevada, para determinar la variación que experimentan las métricas analizadas.

Floodlight

La arquitectura que se describe en este proyecto, utiliza como controlador a Floodlight debido a la simplicidad y polivalencia que ofrece en su implementación. Floodlight está desarrollado en Java y contiene la librería OpenFlow por lo que se permite el uso de todas las características de estos estándares.

Floodlight permite conocer: los detalles de los elementos que componen la red (direcciones IP, máquinas, números de puerto, conexiones, etc.), el número de conmutadores OpenFlow y el camino más corto que sigue en un paquete desde un origen a un destino.

La presente implementación contiene componentes exclusivos que favorecen la captura de las métricas a estudiar. Esta implementación consiste en la captura individual del flujo de datos que circula por la red para extraer las siguientes métricas: retardo, tasa de errores y tasa de datos. Una vez extraídos se realiza la composición de un mensaje en formato clave-valor para tener una visión única de cada uno de ellos.

Debido a las características de la implementación, se detecta que el número de mensajes que genera el controlador contiene duplicados. Por este motivo, surge la necesidad de eliminarlos para conseguir una representación real del tráfico que genera la topología creada por Mininet.

Tras el estudio del algoritmo de monitorización implementado en el controlador, se concluye que es necesario añadir en la creación del mensaje, la marca de tiempo (timestamp) exacta en la que el controlador va a generar el par clave-valor que compone el mensaje. Con el timestamp se garantiza que cada mensaje será creado una única vez al realizar la petición al controlador y no por otro hecho que ocurra en la red.

Una vez libre de duplicados, la Capa de Monitorización, a través del API REST del controlador Floodlight, se encarga de traspasar el mensaje (en formato JSON) a la Capa de Persistencia para que sea instanciado y almacenado en el sistema gestor de base de datos. Finalmente, el mensaje está compuesto por los siguientes elementos: delay, data rate, error rate y timestamp.

4.3.2. CAPA DE PERSISTENCIA

La capa de persistencia se encarga de instanciar el mensaje JSON que viene producido por la capa de monitorización. El mensaje se recibe con la estructura como se muestra en la Figura 4.3.


```
{"test":
   {"1":[
         {"src-switch":1,
          "src-port":2,
          "dst-switch":2,
          "dst-port":2,
          "lk-dataRate":1648,
          "lk-errorRate":0,
          "lk-delay":0,
          "timestamp":12345678901}
        ],
    "2":[
         {"src-switch":2,
          "src-port":2,
          "dst-switch":1,
          "dst-port":2,
          "lk-dataRate":1648,
          "lk-errorRate":0,
          "lk-delay":0,
          "timestamp":12345678901}
         ]
   }
}
```

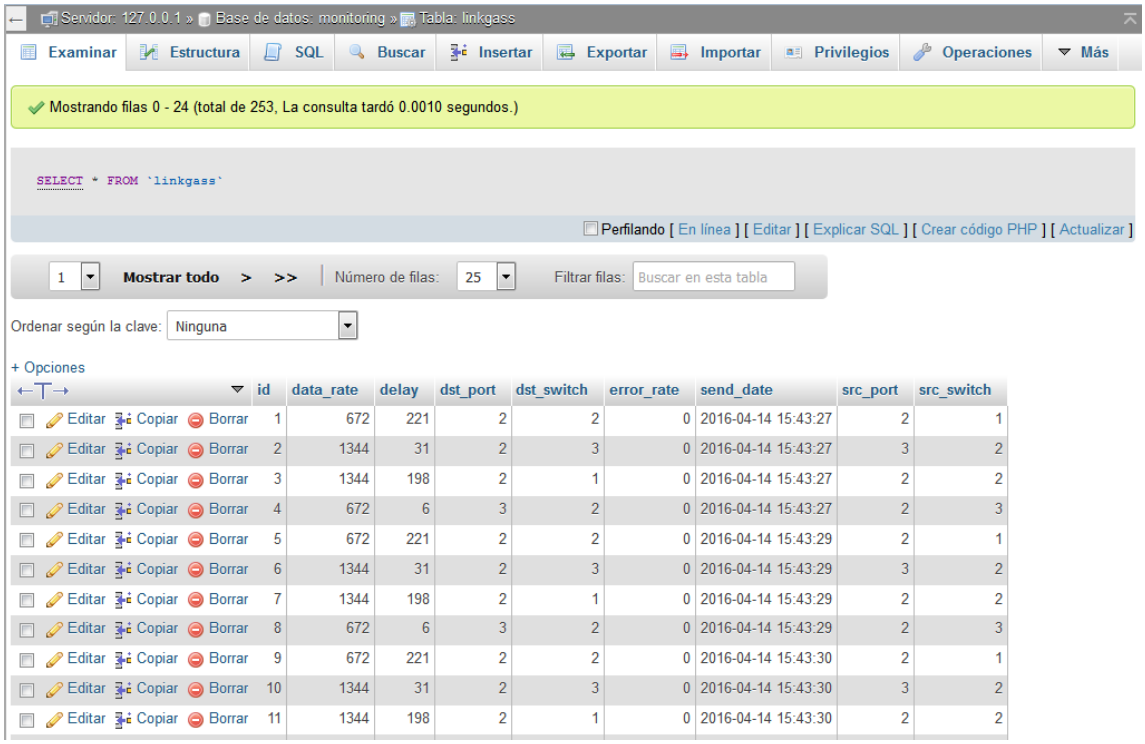

Figura 4.3: Estructura de esquema JSON.

Se muestra una lista de enlaces que forman la topología de la red, con sus respectivas características: dispositivos origen y destino, puertos origen y destino, tasa de datos, tasa de errores y retardo, en el preciso instante (timestamp) en el que se realiza la solicitud.

El módulo encargado de tratar esta información para convertirlo en un objeto, es el llamado "consumidor". Éste módulo se encuentra desarrollado con Spring Boot y gracias a las funcionalidades que ofrece REST Template (como consumidor REST), se captura el mensaje solicitado y se guarda un objeto que contiene la lista de enlaces y su estado en distintas variables. De esta forma, se tiene un objeto completo que puede ser almacenado en el sistema gestor de bases de datos. Además, se añadió el módulo Spring Data JPA para hacer uso de los métodos de la interfaz CrudRepository que ésta facilita, cabe indicar que JPA es una capa que aísla la persistencia del ORM, por lo que, si en algún momento se decide cambiar de ORM, bastaría con cambiar la configuración. Como ORM en este proyecto, se utiliza Hibernate.

El siguiente módulo es la base de datos MySQL y la conexión se realiza mediante JDBC. Con el objeto formado en el apartado anterior se puede guardar toda la información que lo compone por columnas. Así, se tiene por filas la información completa de cada enlace en cada instante de tiempo. Un enlace concreto viene determinado por un dispositivo más un puerto origen y un dispositivo más un puerto destino.

Si se observa la Figura 4.4, se puede apreciar la estructura de la tabla diseñada en la base de datos para almacenar, por columnas, la información y estado de un enlace. La base de datos será utilizada por las capas superiores para mostrar de forma gráfica, la información que ésta contiene.



	id	data_rate	delay	dst_port	dst_switch	error_rate	send_date	src_port	src_switch
Editar Copiar Borrar	1	672	221	2	2	0	2016-04-14 15:43:27	2	1
Editar Copiar Borrar	2	1344	31	2	3	0	2016-04-14 15:43:27	3	2
Editar Copiar Borrar	3	1344	198	2	1	0	2016-04-14 15:43:27	2	2
Editar Copiar Borrar	4	672	6	3	2	0	2016-04-14 15:43:27	2	3
Editar Copiar Borrar	5	672	221	2	2	0	2016-04-14 15:43:29	2	1
Editar Copiar Borrar	6	1344	31	2	3	0	2016-04-14 15:43:29	3	2
Editar Copiar Borrar	7	1344	198	2	1	0	2016-04-14 15:43:29	2	2
Editar Copiar Borrar	8	672	6	3	2	0	2016-04-14 15:43:29	2	3
Editar Copiar Borrar	9	672	221	2	2	0	2016-04-14 15:43:30	2	1
Editar Copiar Borrar	10	1344	31	2	3	0	2016-04-14 15:43:30	3	2
Editar Copiar Borrar	11	1344	198	2	1	0	2016-04-14 15:43:30	2	2

Figura 4.4: Tabla de base de datos,

4.3.3. CAPA DE TRANSFERENCIA

La capa de transferencia se encarga de recibir las peticiones del cliente, tratarlas y responder con la información solicitada (API REST), por otro lado, se encuentra el módulo de análisis de la información el cual genera un fichero de alertas.

API REST

Como se muestra en la Figura 4.1 el flujo de trabajo que persigue este módulo se centra principalmente en la acción que el cliente desea realizar, en este caso obtener información de la Capa de Persistencia, la cual se consulta mediante una URI. El formato elegido para entregar información desde el API REST es JSON.

Para la versión de este proyecto, la función principal del API REST es responder a las operaciones de aplicaciones. Se apoya en el método HTTP GET para mostrar información. En la Figura 4.5 se muestra el esquema de funcionamiento de una aplicación cliente realizando peticiones al API REST.

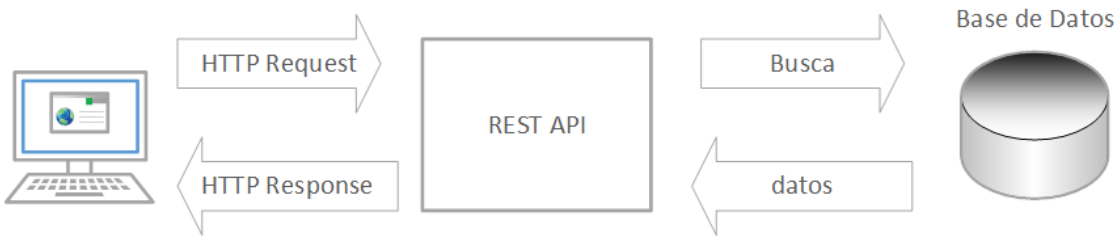


Figura 4.5: Esquema de petición al API REST.

Las peticiones al API REST se realizan siguiendo los siguientes pasos:

1. El cliente realiza una petición mediante AJAX donde se establece HTTP Request (GET) y la URI a consultar.
2. El API REST recibe la petición.
3. El API REST genera un esquema JSON con los datos obtenidos de la base de datos o el fichero generado del análisis de la información.
4. Se envía un HTTP Response con el JSON generado.

El response que se genera mediante un esquema JSON son de dos tipos, como se muestra en la Tabla 4.1.

LinkGass	Alerta
● id: Identificador único, ● srcSwitch: switch origen, ● srcPort: puerto origen. ● dstSwitch: switch destino. ● dstPort: puerto destino. ● dataRate: métrica data rate. ● errorRate: métrica error rate. ● delay: métrica delay. ● sendDate: timestamp.	● Nombre: nombre de fichero. ● Datos: bytes de ficheros.

Tabla 4.1: Información contenida en esquema JSON.

Los modelos jerárquicos que componen una URI para que el API REST los considere como correcta se muestra en la Tabla 4.2.

	Jerarquía	**{order}**
Root	/ws/monitoring	
Buscar por fecha	/linkgass/date/{order}	asc, desc
Buscar según opciones	/linkgass/options/{order}	asc, desc
Alerta	/alerta	

Tabla 4.2: Modelos jerárquicos de URI.

Por lo tanto, las consultas que el cliente puede realizar mediante una URI se muestran en la Tabla 4.3.

Petición	URI
Buscar todos los enlaces	/ws/monitoring/linkgass
Cantidad de filas almacenadas	/ws/monitoring/linkgass/count
Buscar enlaces por fecha de manera ascendente	/ws/monitoring/linkgass/date/asc
Buscar enlaces por fecha de manera descendente	/ws/monitoring/linkgass/date/desc
Buscar los Y primeros enlaces por fecha de manera ascendente, donde Y puede ser 1,10,100 y 1000.	/ws/monitoring/linkgass/date/asc?load=Y
Buscar los Y primeros enlaces por fecha de manera descendente, donde Y puede ser 1,10,100 y 1000.	/ws/monitoring/linkgass/date/desc?load=Y
Buscar los enlaces según opciones de manera ascendente, donde S = switch origen; D = switch destino; SP = puerto origen; DP = puerto destino.	/ws/monitoring/linkgass/options/asc/ src_w=S&dst_sw=D&src_port=SP& dst_port=DP
Buscar los enlaces según opciones de manera descendente, donde S = switch origen; D = switch destino; SP = puerto origen; DP = puerto destino.	/ws/monitoring/linkgass/options/desc/ src_sw=S&dst_sw=D&src_port=SP& dst_port=DP
Buscar los Y primeros enlaces según opciones de manera ascendente, donde S = switch origen; D = switch destino; SP = puerto origen; DP = puerto destino; donde Y puede ser 1, 10, 100 y 1000.	/ws/monitoring/linkgass/options/asc/ src_sw=S&dst_sw=D&src_port=SP& dst_port=DP?load=Y
Buscar los Y primeros enlaces según opciones de manera descendente, donde S = switch origen; D = switch destino; SP = puerto origen; DP = puerto destino; donde Y puede ser 1, 10, 100 y 1000.	/ws/monitoring/linkgass/options/desc/ src_sw=S&dst_sw=D&src_port=SP& dst_port=DP?load=Y
Devolver informe con alertas	/ws/monitoring/alerta

Tabla 4.3: URIs

La implementación se basa en Spring Boot, y se añaden los módulos Spring Data y Spring Actuator.

Spring Data permite utilizar la interfaz CrudRepository el cual facilita las consultas a realizar como, por ejemplo: findAll, insert, delete, count, etc. Sin embargo, para este proyecto se utilizarán aquellas búsquedas que empiecen con la palabra clave "find", que ofrece este módulo sin la necesidad de programar la consulta. Por ejemplo: "findTopByOrderBySendDateAsc" proporciona las primeras filas ordenadas de forma ascendente de la columna sendDate de la base de datos.

Se añade también el módulo Spring Actuator el cual permitirá saber el estado del API REST en el entorno de producción (en nuestro caso en el servidor de pruebas). El acceso a éstas métricas se realiza mediante una URI.

A continuación, se especifica de forma general los aspectos técnicos utilizados para la implementación del API REST:

- **@RestController**

  Esta etiqueta indica que una clase es un controlador el cual devolverá un objeto de dominio/POJO en vez de código que represente una vista. Este objeto que se envía es instanciado en un formato que los clientes puedan comprender. Los formatos que se pueden adoptar son XML y JSON. Para el desarrollo de este proyecto se presentan dos controladores: LinkGass y Alarma.

- **CORS** [CORSWEB]

  Es un concepto importante de seguridad que es implementado en los navegadores de internet para evitar que código JavaScript realice peticiones desde un dominio a otro. Por ejemplo, si un cliente realiza una petición de un recurso, éste debe llevar una cabecera de origen del código del cliente.

  El servidor tendrá en cuenta el origen de la solicitud y responderá con el recurso solicitado y una cabecera Access-Control-Allow-Origin en la respuesta. Suponiendo que el Access-Control-Allow-Origin coincide con el origen de la petición, entonces el navegador accederá a la petición. En otro caso, el navegador denegará la solicitud. En el desarrollo de este proyecto existe un bean de configuración el cual realiza el filtrado CORS.

Análisis de la información

Se ha definido un sistema de registro de alertas en un fichero log, mediante el análisis de los datos almacenados en la capa de persistencia. Estos registros se obtienen tras la ejecución de scripts desarrollados en Python- El sistema se encarga de recoger la información obtenida de la base de datos y de realizar un análisis estadístico sobre los datos. Los módulos utilizados para la implementación son proporcionados por Python, los cuales son:

- import pandas as pd: para poder almacenar los datos consultados de la capa de persistencia en un Dataframe.
- import pymysql as mysql: para realizar la conexión con la base de datos MySQL
- import numpy as np: librería de python que ofrece distintas funciones matemáticas y estadísticas.

En la definición de las alertas, se ha optado por usar los valores estadísticos media y máximo, pero esto es una opción totalmente personalizable, de tal forma que se puede definir un límite numérico, o utilizar cualquier otra medida estadística que se adapte a lo que se quiera guardar en el fichero.

4.3.4. CAPA VISUALIZACIÓN

La capa de visualización se encarga de presentar en un ámbito web las métricas que se han obtenido, en forma de esquema, de la capa de transferencia. Para ello se sigue la arquitectura cliente-servidor.

La arquitectura cliente-servidor, se basa en la petición y respuesta de los múltiples servicios que conforman el sistema. El flujo seguido por la arquitectura es el siguiente: uno o varios clientes realizan peticiones al servidor quien es el encargado de responder dando la información solicitada.

Para la representación, se han utilizado las siguientes herramientas:

- Bootstrap: Se utiliza para favorecer el modelo responsive y la compatibilidad con prácticamente todos los navegadores modernos.

- HighCharts: Se utiliza gracias que tiene un carácter Open-source y a su amplio abanico de gráficas. Es una librería escrita en JavaScript que al igual que Bootstrap es soportada por la mayoría de navegadores (incluso en Internet Explorer 6).

Herramientas

Tras la respuesta a la petición realizada por el cliente, se obtiene un objeto de datos en formato JSON, con estos datos se inicia la implementación de la capa de visualización utilizando las siguientes herramientas:

- **Grafana**[10]: Se trata de una librería open source que permite la implementación de gráficos dentro del ámbito de aplicaciones web. Permite la representación y análisis de cualquier información que desee un cliente. Por ejemplo: estadísticas industriales, automatizaciones, mecanismos de control, etc. Ofrece también dashboards predefinidos para utilizarlos de manera directa. Sin embargo, tras estudiar ejemplos de arquitecturas que ya utilizaban Grafana, se observa que su implementación tiene una complejidad elevada. Este factor hace que la integración con este proyecto represente un reto difícil de afrontar.

  Además, cabe destacar que Grafana utiliza como sistema gestor de base de datos a Influx. El uso de Influx supone otro inconveniente para su uso porque para el almacenamiento de las métricas presenta un nivel bajo de rendimiento.

- **ASP.NET:** Esta solución ofrece la posibilidad de componer el esquema obtenido de la capa de transferencia, de acuerdo con la representación soportada por HighCharts en cada una de sus implementaciones gráficas. La primera aproximación consiste en el uso de ASP.NET debido a la simplicidad que presenta en el acceso entre la interfaz visual y el código de control (code behind). Sin embargo, aunque esta aproximación funciona, se observa que, para satisfacer el objetivo principal del sistema en la representación real de los datos, basta con la realización de peticiones AJAX cada vez que un dato sea almacenado en el sistema. Es por esto, que, aunque el desarrollo en esta herramienta es viable, no es lo más adecuado en nuestro caso.

[10] "Grafana - Beautiful Metrics, Analytics, dashboards and monitoring!." 2014. 5 Jun. 2016 <http://grafana.org/>

- **Dashboard V1:** Esta implementación permite presentar los gráficos de Highcharts de forma conjunta. En primer lugar, el usuario es capaz de elegir la métrica que desea ver y estudiar, y, en segundo lugar, una vez elegidas se presentan a pantalla completa para poder analizarlos y/o exportarlos ya sea en formato imagen o formato PDF. Gracias a la simplicidad en el diseño, la fusión con el desarrollo es prácticamente inmediata. El prototipo de este dashboard se muestra en la Figura 4.6.

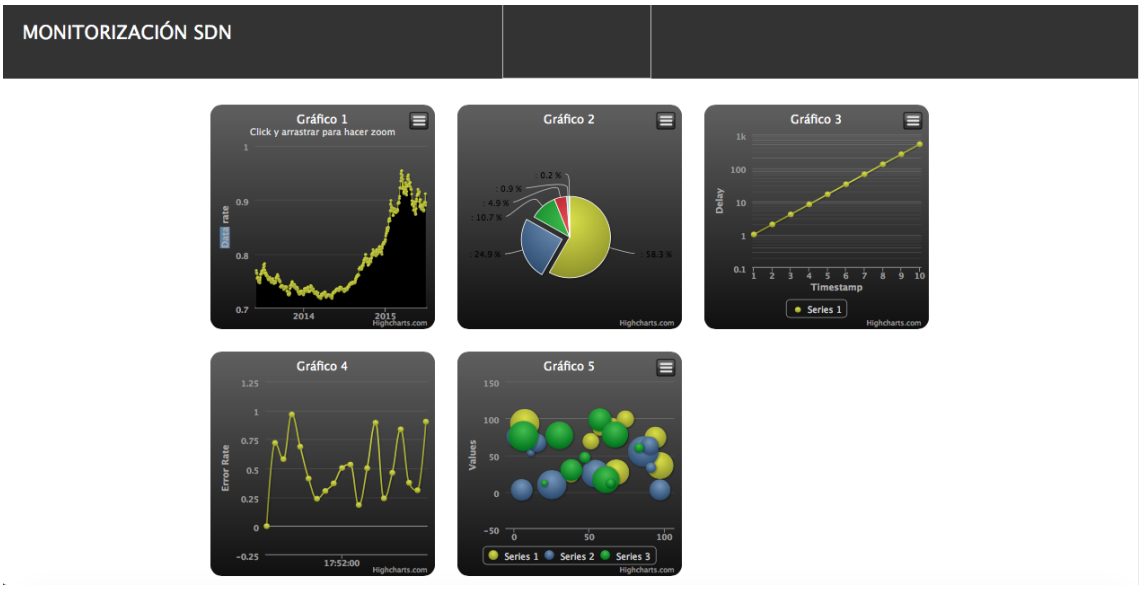


Figura 4.6: Dashboard V1.

Sin embargo, la realización de múltiples pruebas para ajustar los gráficos, arroja como resultado que el template utilizado contiene tanto estilos como scripts propios que no son compatibles con la visualización utilizada en este proyecto y también se observa que esta implementación no se adapta al diseño responsive buscado.

Con los inconvenientes que aparecen en la primera versión del dashboard, se pone en manifiesto que además existe la limitación de espacio para mostrar el sistema de alertas sobre las distintas métricas que se analizan en este proyecto. Así pues, se adecua a nuestra idea un nuevo template que cumpla con las características requeridas. Se elige uno proporcionado por Bootstrap Taste [BTT16] ya que proporciona un core gratuito que permite la modificación y adaptación según las necesidades de desarrollo.

- Bootstrap Taste: Este template se desarrolla utilizando el framework de diseño Bootstrap. Tiene como características principales la compatibilidad con el diseño responsive y la división del mismo en forma de Dashboard de trabajo.

La presente implementación utiliza el framework de Bootstrap Taste porque:
En primer lugar, posee un mecanismo de barra lateral que proporciona la posibilidad de establecer un enlace directo al sistema de alertas y a cada uno de los gráficos. En segundo lugar, una barra en el margen superior, donde se encuentra el título del sistema de administración y que se registra bajo el nombre de SDN Monitoring, que hace la función de acceso directo al inicio desde cualquier lugar de aplicación. También en esta barra superior, se proporciona un acceso directo a cada gráfico de forma individual, gracias a la implementación de un menú desplegable que contiene un enlace directo a los gráficos mencionados. Finalmente, en el panel central se implementan los gráficos del sistema en formato reducido, de forma que el administrador de red puede tener una visión completa y detallada del estado del sistema en ese instante en concreto. Cabe destacar que gracias a la versatilidad de HighCharts, se han implementado dos formas de representación.

La primera contiene un gráfico que se va actualizando en tiempo real de acuerdo con el último dato que se ha almacenado en el sistema. Su funcionamiento se constata gracias a las peticiones AJAX que realiza la aplicación. Estas peticiones constan en primer lugar de una primera petición de los últimos datos introducidos para mostrar una aproximación del estado del sistema, y, en segundo lugar, se van realizando peticiones sucesivas cada vez que un dato se almacena en el sistema con el fin de generar una representación en tiempo real del estado mencionado. Por otro lado, la segunda implementación consta de una serie de gráficos estáticos que proporcionan información de las diferentes métricas:

- El primero de ellos es una representación en forma circular que muestra el porcentaje de datos monitorizados y almacenados en el sistema de todas las métricas.

- El segundo de ellos es una representación unívoca de todas las métricas representadas como áreas. Cabe destacar que para la realización de este gráfico se presenta el tiempo como formato común del eje x.

La característica principal de este gráfico es la capacidad de ver el valor de todas las métricas en un instante de tiempo determinado gracias a la característica común del eje x mencionada.

- El tercer gráfico es una representación en forma de área máxima de la tasa de error. En él, destaca la posibilidad de seleccionar un área concreta para expandir y observar de forma más concreta la tasa de error en un instante concreto. Este hecho es favorecido gracias a la opción drag and drop que posee el gráfico y que hace que el gráfico experimente un zoom instantáneo en el área seleccionada.

A modo de ejemplo en la Figura 4.7 y la Figura 4.8, se ilustra lo anteriormente descrito.

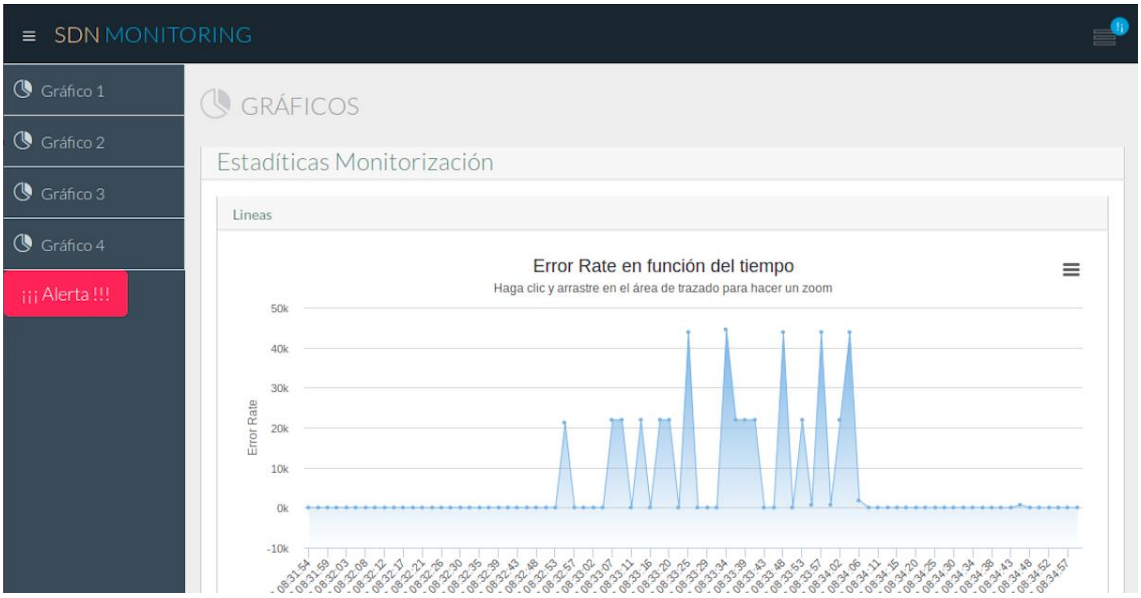


Figura 4.7: Gráfico 3 Área máxima Tasa de Error

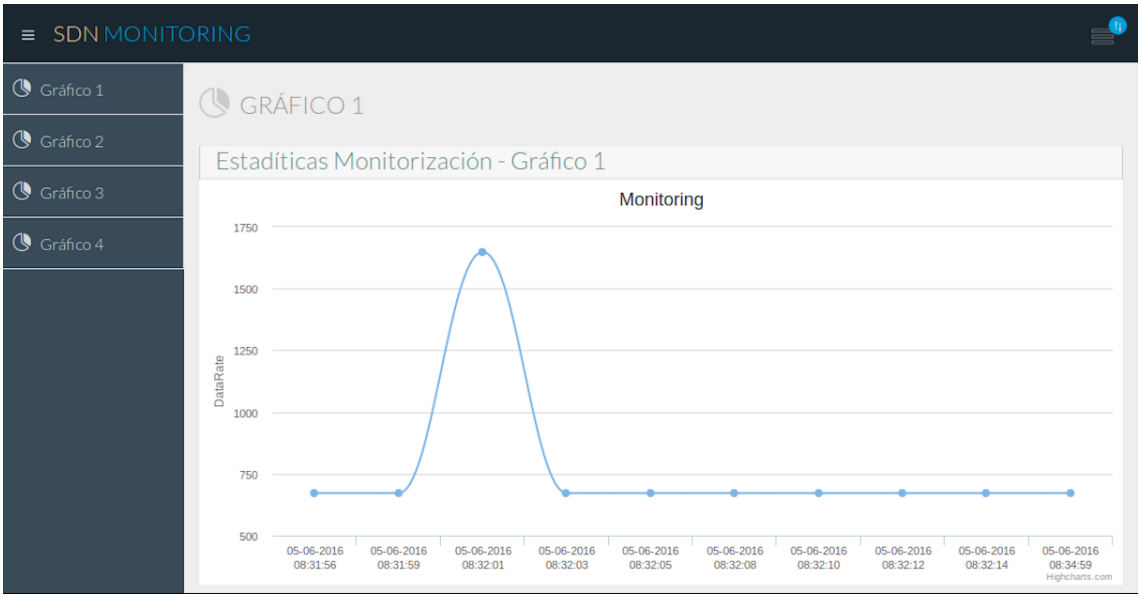


Figura 4.8 Gráfico 1 Tasa de Datos en Tiempo Real

Junto con todas estas funcionalidades, la aplicación ofrece la posibilidad de ver cada gráfico a tamaño completo para que el administrador de red sea capaz de: Observar, analizar, examinar, estudiar y comparar la información de las métricas monitorizadas en la red SDN. Es destacable que la aplicación se encuentra implementada de forma fluida y estable, y, se favorece el aprendizaje de los usuarios en un tiempo ínfimo gracias a la simplicidad del diseño desarrollado.

5. EXPERIMENTOS Y RESULTADOS

Tras definir como se ha implementado la arquitectura del proyecto en el apartado anterior, se procede a mostrar la validez del mismo. Con este fin, se enfoca como contexto de trabajo un sistema compuesto por seis máquinas virtuales y un servidor:

5.1. COMPONENTES SOFTWARE Y ENTORNOS DE EJECUCIÓN

Los componentes del entorno de ejecución de pruebas están compuestos por varias máquinas virtuales localizados en el servidor del grupo GASS-UCM, tal como se describe en la Figura 5.1.

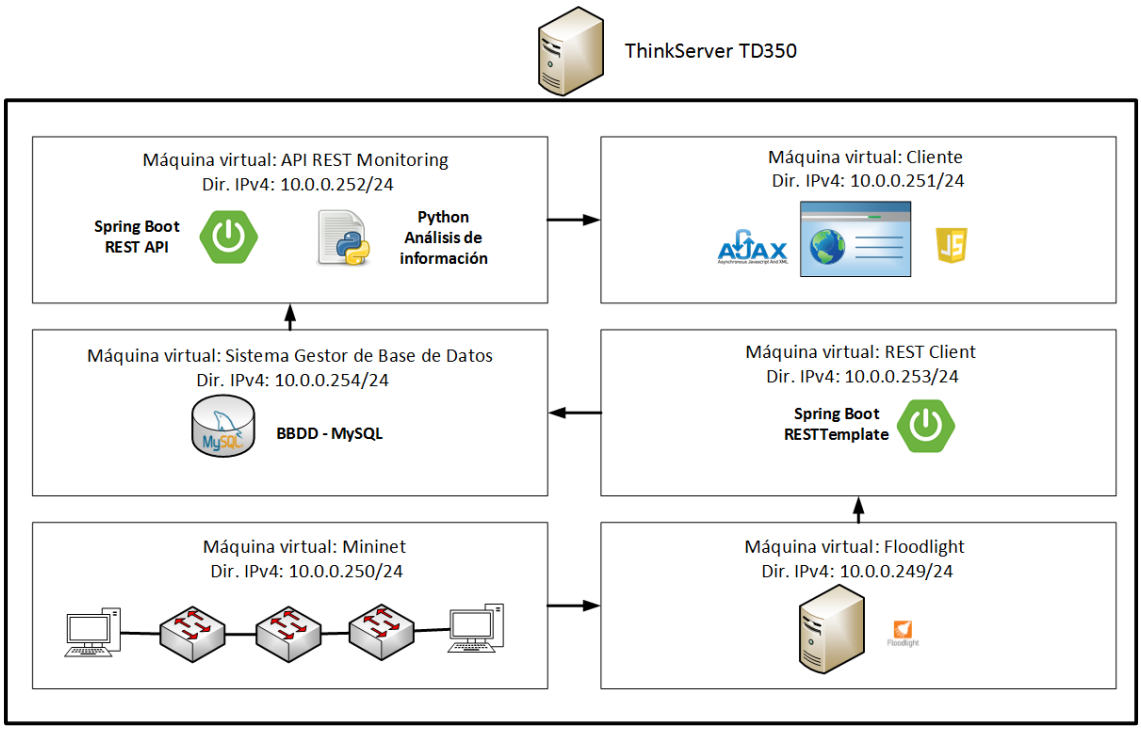


Figura 5.1: Componente software en entorno de ejecución.

1. **Servidor:**

   El servidor utilizado, aloja todas las máquinas virtuales del apartado anterior. Aquí se encuentran asociadas para formar la arquitectura propuesta. Con esto se consigue formar un sistema homogéneo para monitorizar, analizar y gestionar una red SDN.

Servidor	Características
ThinkServer TD350	Intel® Xeon® Serie E5-2600 v3 de 16 núcleos
	DDR4 de 512 GB a 2133 MHz. 16 ranuras (RDIMM/LRDIMM)
	Adaptador ThinkServer RAID 720i AnyRAID (0/1/10/5/50/6/60)
	Dimensiones: 251 mm x 459 mm x 686 mm (9,9" x 18,1" x 27,0")

Tabla 5.2: Características de servidor.

2. **Máquinas virtuales:**
   a. **Mininet:** máquina que se encarga de generar la topología de la red SDN a analizar, junto con el tipo y volumen de tráfico de la misma.
   b. **Floodlight:** máquina que monitoriza el tráfico que circula por la red SDN creada en la máquina Mininet.
   c. **Rest-Client:** máquina que se encarga ejecutar un JAR que captura el esquema JSON que genera Floodlight, lo instancia y guarda el objeto en la base de datos MySQL.
   d. **Sistema Gestor de Base de Datos:** máquina que tiene almacenados los datos obtenidos de la máquina Rest-Client. Destaca su independencia gracias a encontrarse como un módulo de almacenamiento externo, por lo que se favorece la introducción de cualquier otro sistema gestor de base de datos.
   e. **API REST- Monitoring:** su función principal es el tratamiento de los datos almacenados en la capa de persistencia para satisfacer las necesidades de un cliente. Para completar esta funcionalidad, se ejecuta dentro de la misma un JAR encargado de formar el esquema descrito.

f. **Cliente:** máquina que tiene la función de mostrar la información del estado de la red por medio de gráficos, para que el administrador tenga una visión real e instantánea del estado de la misma. Para ello, realiza peticiones AJAX a la capa de transferencia. La dotación de acceso a Internet de esta máquina, surge de la necesidad de utilizar las múltiples librerías necesarias para una aplicación web.

Las máquinas virtuales de Mininet y Floodlight presentan el mismo software y las mismas características de configuración. El resto de las máquinas virtuales, albergan los diferentes módulos que se presentan en la propuesta de arquitectura, también con software y configuración idéntica. Para preparar el entorno de ejecución, se ha utilizado el software que se describe en la Tabla 5.1.

	Características	**Software instalado**	**Direcciones IP en servidor**
Mininet	Linux Ubuntu 64 bits, 4096 memoria base, 2 CPU.	VLC Player. Python 2.7.4.	IP: 10.0.0.250/24 Gw: 10.0.0.1
Floodlight	Linux Ubuntu 64 bits, 4096 memoria base, 2 CPU.	Java (Entorno Eclipse):Controlador Floodlight con implementaciones propias.	IP: 10.0.0.249/24 Gw: 10.0.0.1
REST-Client	Linux Ubuntu 64 bits, 4096 memoria base, 2 CPU, 8 GB disco duro.	REST-JAR: Implementación propia que captura la monitorización de Floodlight.	IP: 10.0.0.253/24 Gw: 10.0.0.1
Sistema Gestor de Base de Datos	Linux Ubuntu 64 bits, 4096 memoria base, 2 CPU, , 8 GB disco duro.	LAMPP: Servidor local para distribuciones Linux.	IP: 10.0.0.254/124 Gw: 10.0.0.1
API REST Monitoring	Linux Ubuntu 64 bits, 4096 memoria base, 2 CPU, 8 GB disco duro.	REST Monitoring-JAR: Implementación propia que acepta y responde peticiones AJAX.	IP: 10.0.0.252/24 Gw: 10.0.0.1
Cliente	Linux Ubuntu 64 bits, 4096 memoria base, 2 CPU, 8 GB disco duro.	LAMPP: Servidor local para distribuciones Linux. Aplicación Web	IP: 10.0.0.251/24 Gw: 10.0.0.1

Tabla 5.1: Características del entorno de ejecución.

Las máquinas virtuales con las características que se muestran en la Tabla 5.1, se encuentran en un servidor proporcionado por GASS. Éstas se encuentran conectadas mediante una red interna. El acceso a las máquinas virtuales desde el exterior se logra mediante la conexión VPN que proporciona la UCM.

5.2. EXPERIMENTOS

El objetivo principal que se busca es demostrar de forma real y clara el estado en el que se encuentra una red de acuerdo con las distintas métricas que se analizan en este proyecto.

Para ello se han definido distintos escenarios con y sin errores, para poder observar el comportamiento de las gráficas una vez que se ha transmitido el vídeo a través de ellos. Las topologías usadas en los experimentos tienen las siguientes características:

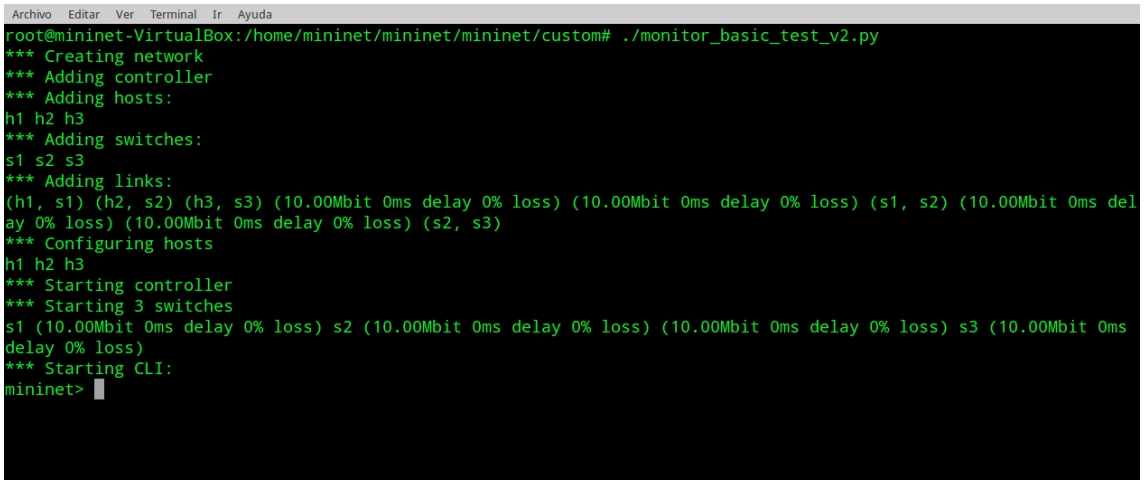
1. La primera topología es sencilla y está compuesta por 3 switches y 3 hosts. El vídeo se transmite desde el host conectado al switch 1 hasta el host conectado al switch 3.
2. La segunda topología es más grande, está compuesta por 10 switches y 10 hosts y el vídeo se transmitirá desde el host en el switch 1 hasta el host en el switch 10. Así se podrá comprobar cómo varían las métricas de la monitorización cuando el camino tiene más saltos.

Para cada una de las topologías, se han definido dos escenarios, uno con errores y otro sin ellos, de esta forma:

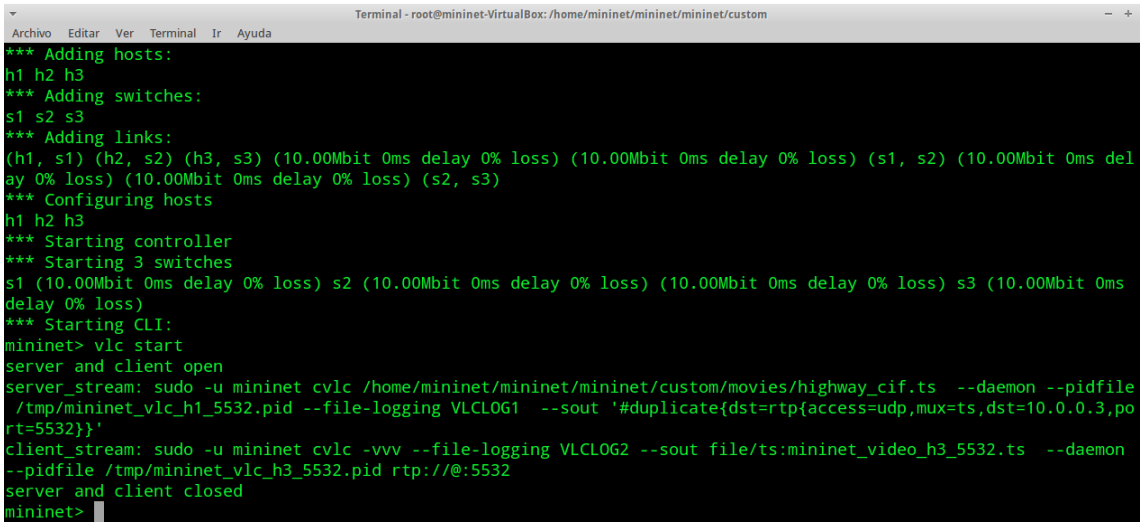
1. Escenarios sin errores: los enlaces entre los switches no tienen ningún error añadido, es decir, el retardo se ha definido a 0 milisegundos y la tasa de pérdida de paquetes al 0%.
2. Escenarios con errores: se han añadido errores a los enlaces entre los switches, tanto en el retardo como en la tasa de pérdida de paquetes.

Para el desarrollo de las pruebas, se ha transmitido el vídeo highway_cif.ts (en formato MPEG2 y cuyo tamaño es de 2,5 MB) de un host a otro de la topología. Las pruebas realizadas constan de los siguientes pasos:

1. Ejecutar el archivo monitor_basic_test_v2.py o el archivo linear_basic_text_v1.py que contienen los scripts para la creación de las topologías, en la máquina virtual Mininet. El controlador de la red se encuentra en la máquina Floodlight cuya dirección IP es: 10.0.0.249, con número de puerto 6633.
2. Una vez iniciada la topología, dentro del prompt se introduce la instrucción: *vlc start*. Este comando realiza la labor de envío del vídeo desde el host-servidor al host-cliente (Figura 5.2).



(a)



(b)

Figura 5.2: (a) Arranque y (b) envío de vídeo.

La información del flujo de datos que se da en el envío del vídeo de prueba, circula por todas las capas descritas en la arquitectura hasta llegar a la capa de visualización donde se presentan los resultados obtenidos.

5.2.1. TOPOLOGÍA 1

La primera topología, denominada Monitor basic test V2, consta de tres switches y tres hosts. Su distribución se especifica en la Figura 5.3.

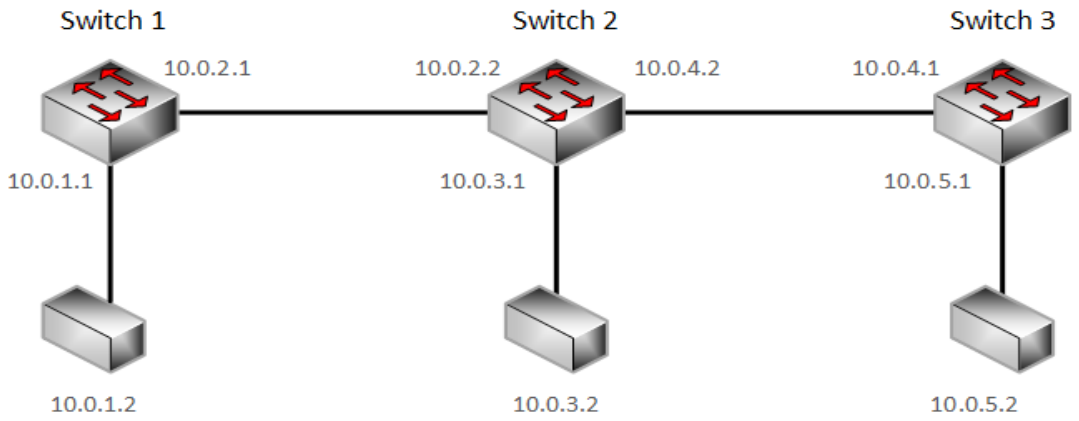


Figura 5.3: Monitor basic test V2.

El enlace que se ha monitorizado es el que une el switch 2 con el switch 3, que es donde se encuentra el host destino conectado.

5.2.1.1. Escenario 1

En este escenario se busca una topología sin errores, sin retardos y con un ancho de banda adecuado para que no se produzcan cuellos de botella ni saturaciones en los enlaces de la red. En el script de Python que crea la topología se detallan las características de los enlaces para satisfacer este escenario. Estas características se muestran en la Tabla 5.3.

Enlaces	**Ancho de banda (bw)**	**Retardo (delay)**	**Tasa de Pérdida (loss)**	**(use_htb)**
linkopts1	10 Mbps	0 ms	0 %	true
linkopts2	10 Mbps	0 ms	0 %	true

Tabla 5.3: Características de enlaces en escenario 1 de topología 1.

5.2.1.2. Escenario 2

En este escenario se busca tener una aproximación de lo que ocurre con el tráfico si circula por una red que no es estable, es decir, presenta errores en su implementación como: ruido que existe en la red, tiempo que tarda en estabilizarse, etc. Los parámetros de error se definen dentro del fichero de topología de red con las características que se muestran en la Tabla 5.4.

Enlaces	**Ancho de banda (bw)**	**Retardo (delay)**	**Tasa de Pérdida (loss)**	**(use_htb)**
linkopts1	10 Mbps	5 ms	5 %	true
linkopts2	10 Mbps	2 ms	2 %	true

Tabla 5.4: Características de enlaces en escenario 2 de topología 1.

5.2.2. TOPOLOGÍA 2

La segunda topología, denominada Linear basic test V1, consta de diez switches y diez hosts. Con el fin de conseguir que los valores de las métricas analizadas en esta arquitectura tengan un valor añadido y se encuentren dentro del ámbito de una topología más grande, se ha implementado ésta. Su distribución se especifica en la Figura 5.4.

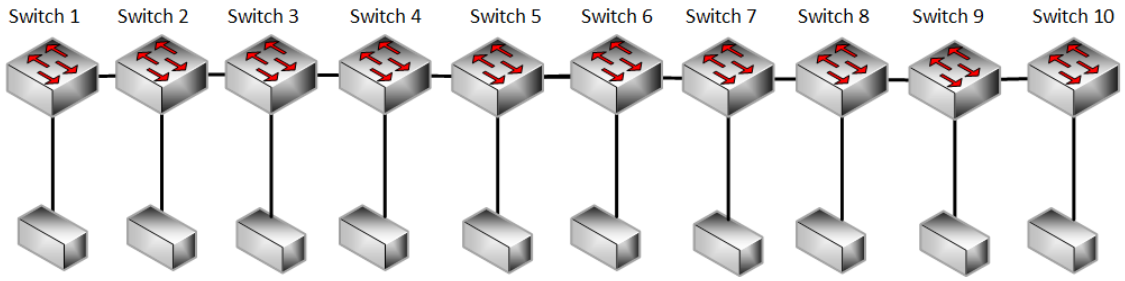


Figura 5.4: Linear basic test V1.

En este caso, el enlace que se ha monitorizado es el más alejado del origen de la transmisión, es decir, el que une el switch 9 con el switch 10, que es donde está conectado el host destino.

5.2.2.1. Escenario 1

Al igual que en el Escenario 1 de la Topología 1, todos los enlaces se han creado sin errores añadidos, como se muestra en la Tabla 5.5.

Enlaces	**Ancho de banda (bw)**	**Retardo (delay)**	**Tasa de Pérdida (loss)**	**(use_htb)**
linkopts	10 Mbps	0 ms	0 %	true

Tabla 5.5: Características de enlaces en escenario 1 de topología 2.

5.2.2.2. Escenario 2

En este escenario se ha establecido una configuración agresiva de los enlaces, para poder ver las diferencias mostradas gráficamente en el cliente, con respecto a los escenarios anteriores. Al igual que en el escenario 1, todos los enlaces están dotados de las mismas características.

Los parámetros de error se definen dentro del fichero de topología de red con las características que se muestran en la Tabla 5.6.

Enlaces	**Ancho de banda (bw)**	**Retardo (delay)**	**Tasa de Pérdida (loss)**	**(use_htb)**
linkopts	10 Mbps	10 ms	5 %	true

Tabla 5.6: Características de enlaces en escenario 2 de topología 2.

5.2.3. ALERTAS

En el dashboard del cliente se ha colocado un enlace llamado “Alertas”. Es el encargado de llamar a los scripts de python y traer los ficheros de eventos que se hayan podido generar hasta ese momento y mostrarlos por pantalla. En este caso, para el cálculo de las alertas, se han utilizado los valores estadísticos media y máximo, de la siguiente forma:

- Al analizar los valores de los distintos enlaces de la red, se calcula la media de todas las métricas de tasa de datos. De esta forma, cada vez que llega un dato mayor que la media y prolongado en el tiempo durante más de 15 segundos, se graba una alerta.
- Lo mismo ocurre para la métrica de tasa de errores, pero comparándolo con el máximo que había llegado hasta ese momento. Se graba la alerta una vez que hay una tasa de errores mayor que las que han llegado hasta ese momento.

5.3. RESULTADOS

Una vez se han definido los escenarios que se van a utilizar para los experimentos (Apartado 5.2), se han ido ejecutando uno por uno de tal forma que se han obtenido los distintos resultados. Al igual que en el apartado anterior, se van a mostrar los resultados usando el orden en el que se han ido ejecutando, empezando por los dos escenarios de la topología 1 y terminando por los dos escenarios de la topología 2. Para cada uno de ellos, se adjuntan las gráficas correspondientes a las métricas obtenidas, tasa de datos, tasa de errores y retardo.

Al final de los resultados, se adjuntan las comparativas de los diferentes valores obtenidos al ejecutar los experimentos. Se pueden ver de la Tabla 5.7 a la Tabla 5.10.

5.3.1. TOPOLOGÍA 1

5.3.1.1. Escenario 1

En el escenario 1 se ha enviado el vídeo del host origen al destino a través de Mininet y mediante el uso de una topología sin errores.

Si se analizan las métricas observadas en el dashboard de gráficas del cliente, se puede observar que se han producido muy pocos errores (Figura 5.6) y que la transferencia de datos se ha mantenido constante a lo largo del envío del flujo de datos (Figura 5.5 y Figura 5.7). Ya que la topología se había definido libre de errores, se puede concluir que los errores que aparecen son introducidos por los elementos de la red.

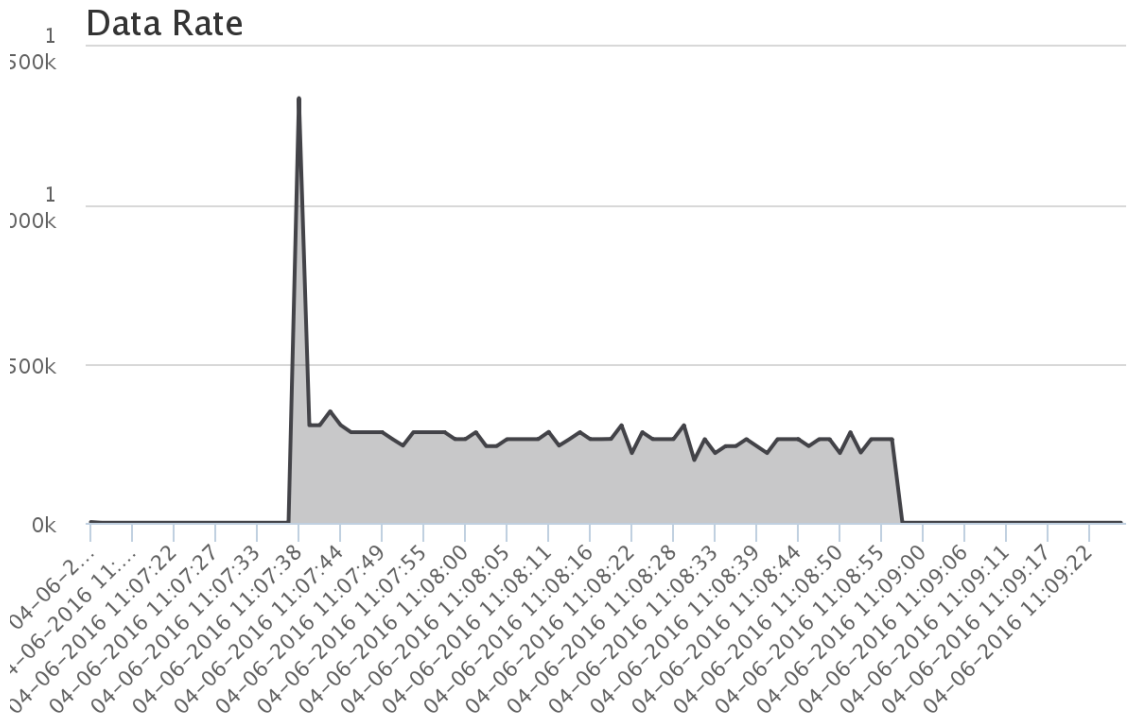


Figura 5.5: Métrica data rate de escenario 1 y topología 1.

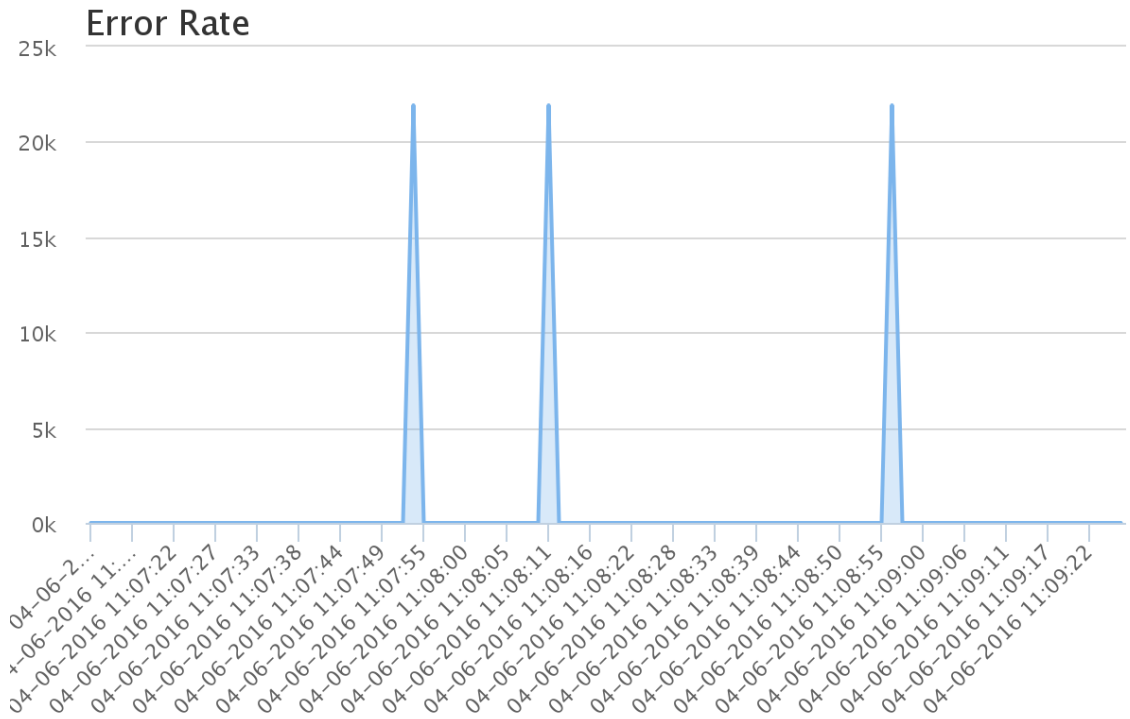


Figura 5.6: Métrica error rate de escenario 1 y topología 1.

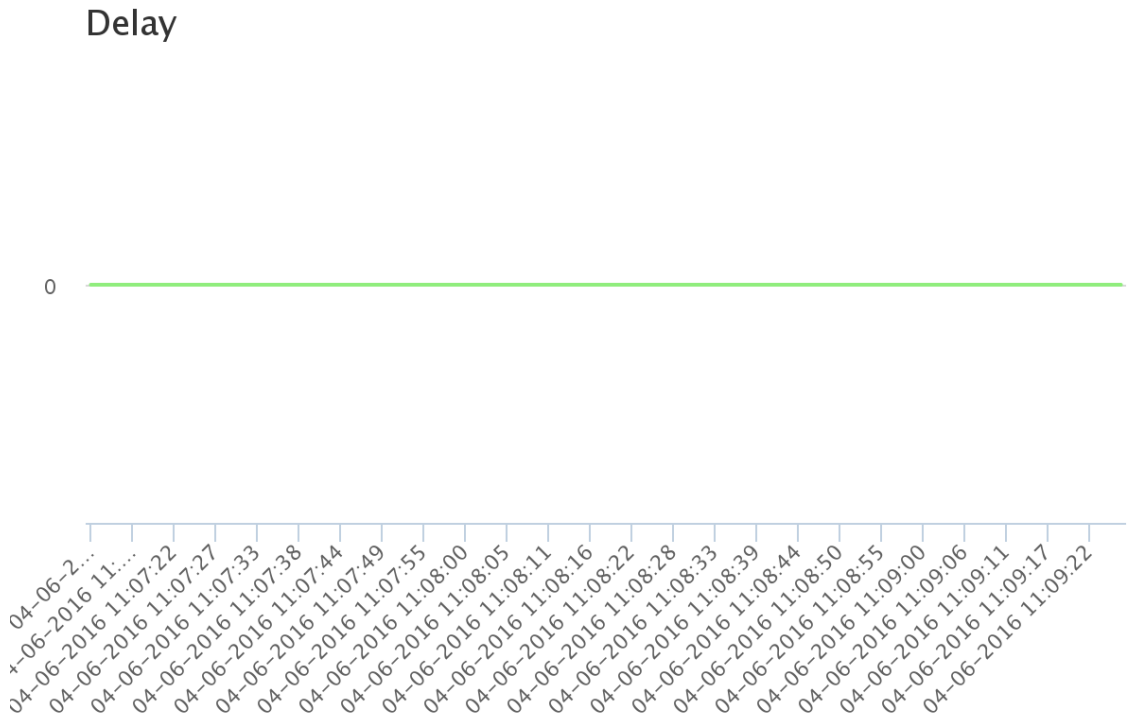


Figura 5.7: Métrica delay de escenario 1 y topología 1.

Los picos encontrados en la tasa de errores son casi despreciables, teniendo en cuenta la tasa de datos que se estaba transmitiendo en ese preciso instante (Tabla 5.7). El hecho de que se hayan producido pocos errores tiene sus consecuencias en el tamaño y la calidad del vídeo, los cuales se han reducido ligeramente (Tabla 5.8 y 5.9).

5.3.1.2. Escenario 2

Una vez que se ha enviado el vídeo del host origen al destino a través de Mininet y en el escenario de una topología con los errores mencionados anteriormente, se puede observar en las Figuras 5.8 y Figura 5.9, la variabilidad que se ha producido en la tasa de datos y de errores, con respecto al escenario anterior. Por otro lado, el retardo se mantiene constante (Figura 5.10).

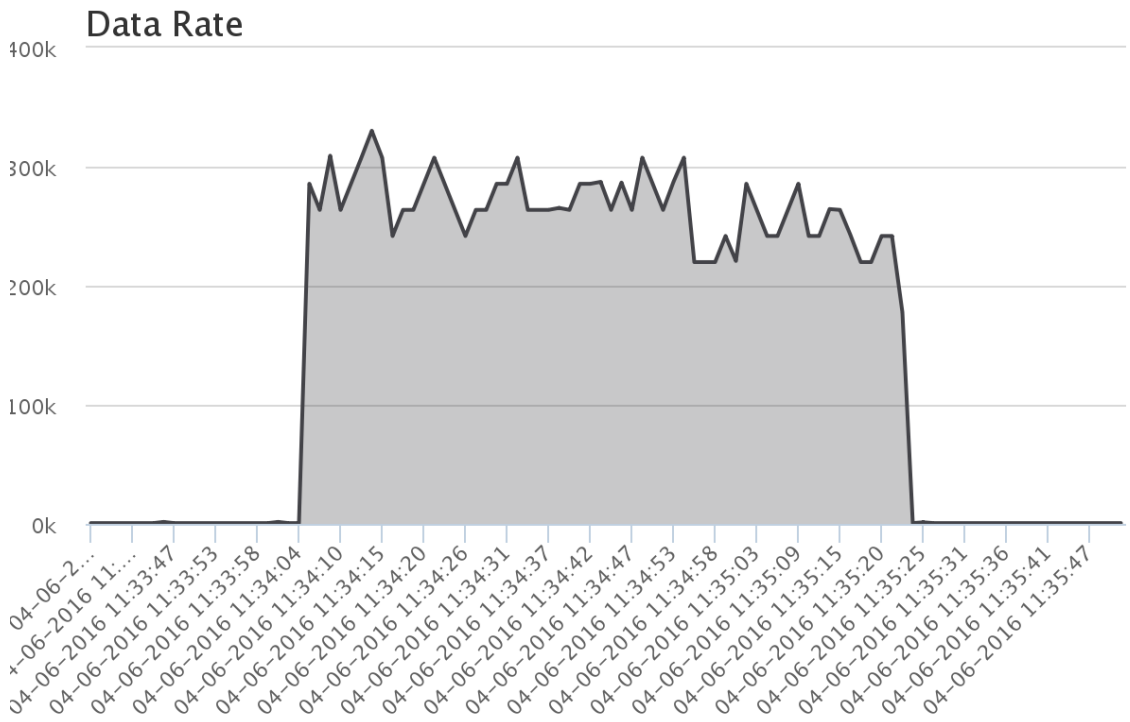


Figura 5.8: Métrica data rate de escenario 2 y topología 1.

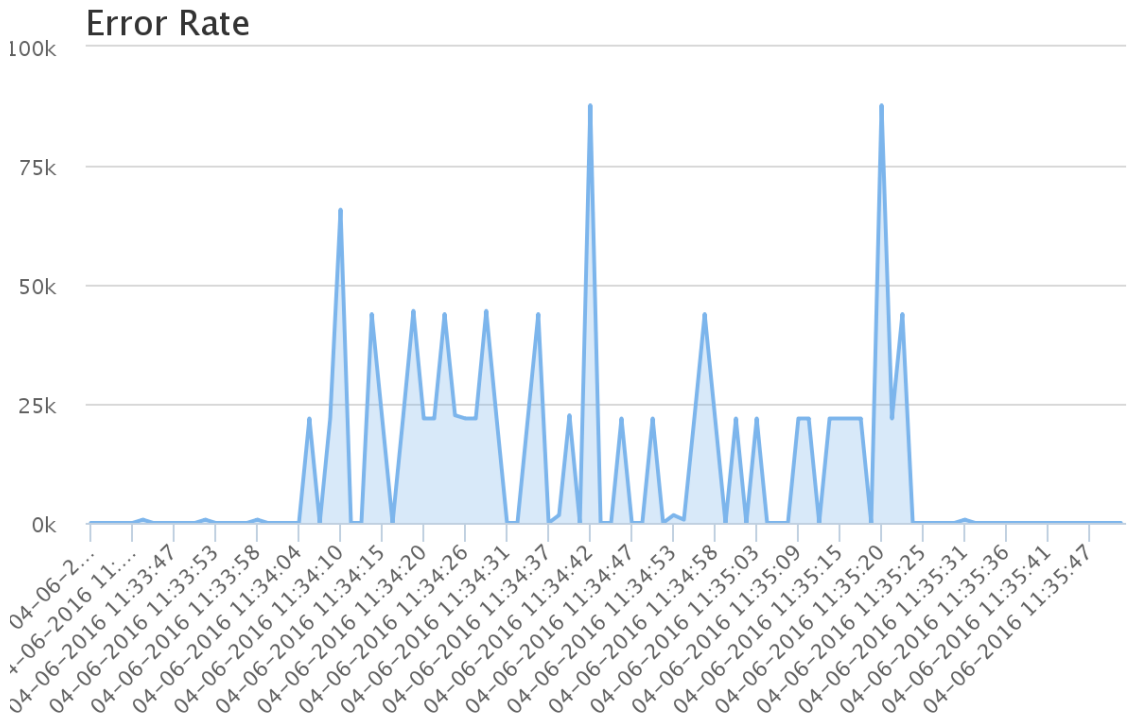


Figura 5.9: Métrica error rate de escenario 2 y topología 1.

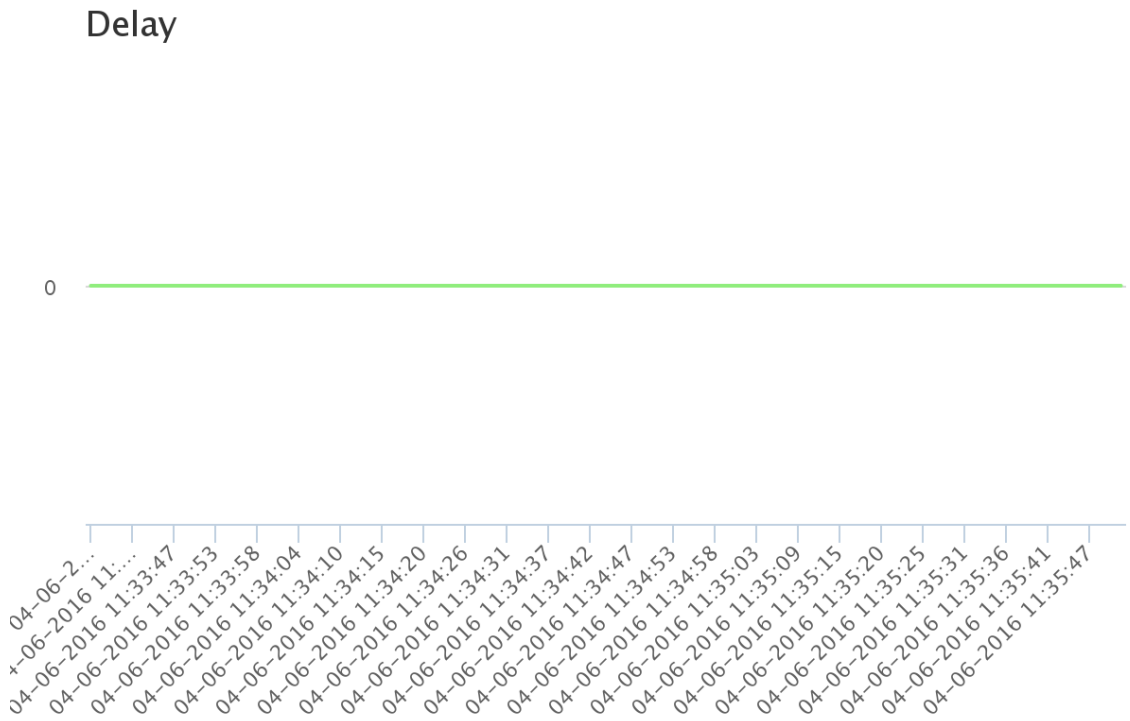


Figura 5.10: Métrica delay de escenario 2 y topología 1.

De acuerdo con los resultados obtenidos, se deduce que, en una topología con errores, se producen mayores fluctuaciones tanto en la tasa de datos enviados como en la tasa de errores de los mismos. El hecho de que la pérdida de paquetes sea una actividad constante mientras circule tráfico, hace que el archivo llegue dañado y que se visualice pixelado en ciertos instantes.
Se pueden observar los datos medidos y sus resultados a través de la Tabla 5.7. Además, se puede comprobar como el vídeo ha reducido su tamaño después de la transmisión (Tabla 5.8).

5.3.2. TOPOLOGÍA 2

5.3.2.1. Escenario 1

Se ha utilizado el mismo archivo de vídeo que en la topología 1, por tanto, tiene el mismo tamaño inicial. Al enviar el vídeo del host origen al destino a través de Mininet y en el escenario de una topología linear de 10 switches sin errores añadidos, se puede observar en la Tabla 5.8, que el tamaño del vídeo se ha reducido ligeramente. Cabe recordar que el enlace que se está monitorizando es el más alejado del host origen, es decir, el enlace entre el switch 9 y el 10.

Al analizar las métricas observadas en el dashboard de gráficas del cliente, se puede observar que se han producido muy pocos picos de errores, aunque la constancia de los mismos no es tan evidente como lo era en la topología 1 (Figura 5.12). Esto es debido a que se está monitorizando el último enlace en el camino y que la información ha pasado primero por 8 enlaces (9 switches).

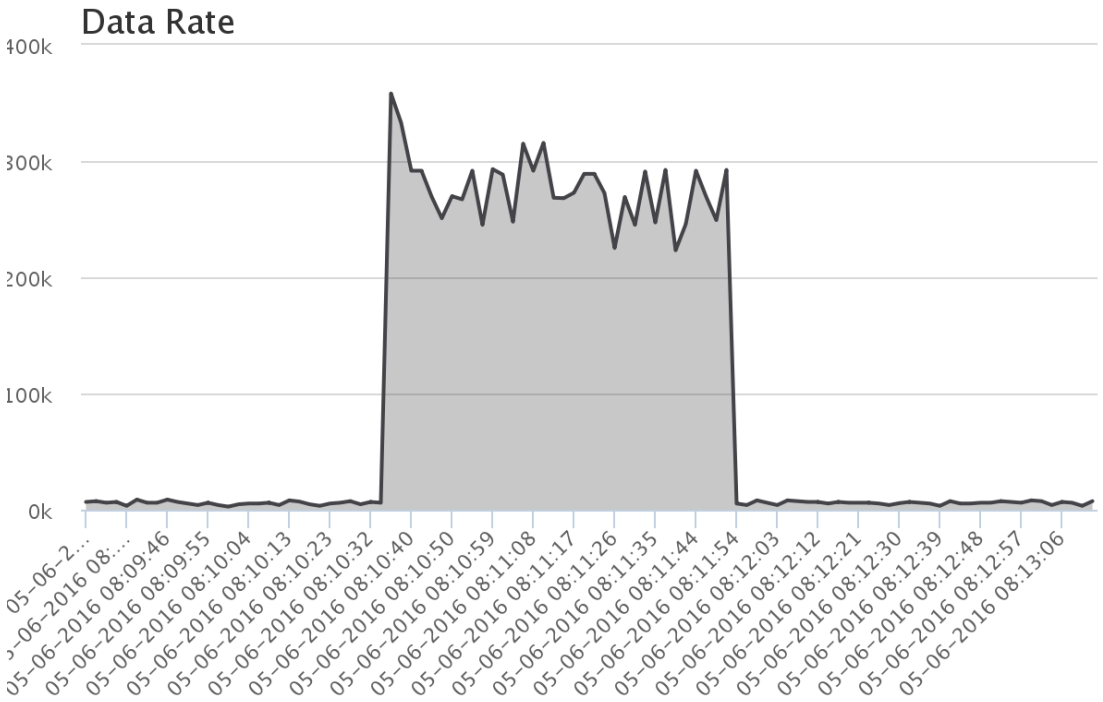


Figura 5.11: Métrica data rate de escenario 1 y topología 2.

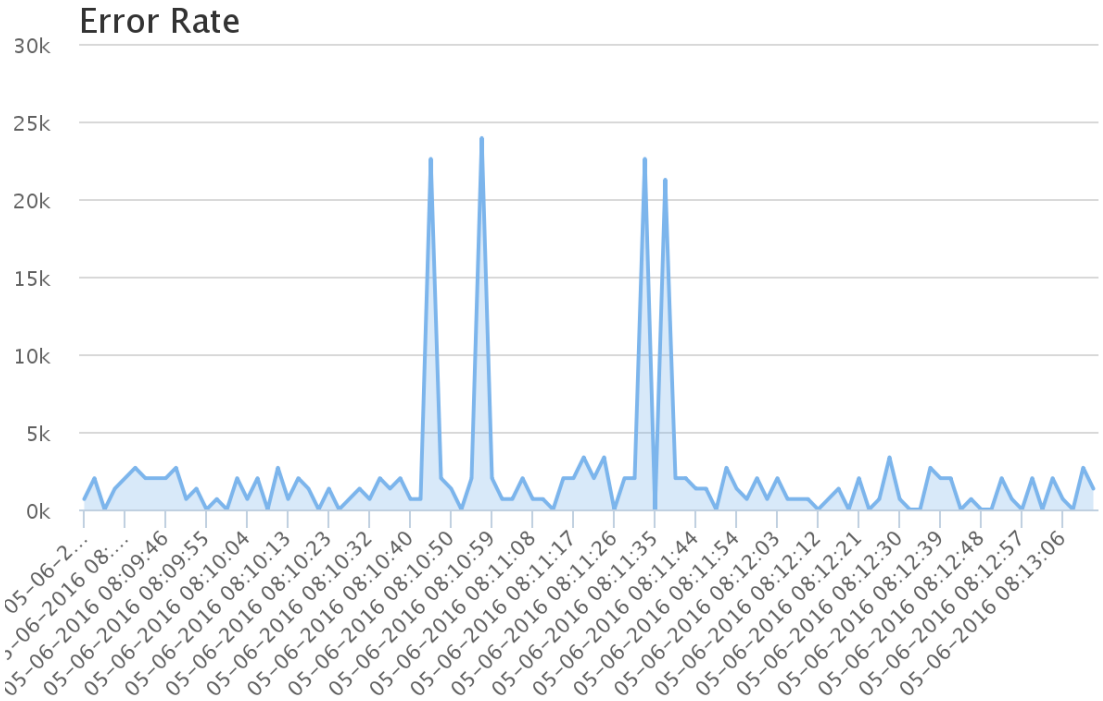


Figura 5.12: Métrica error rate de escenario 1 y topología 2.

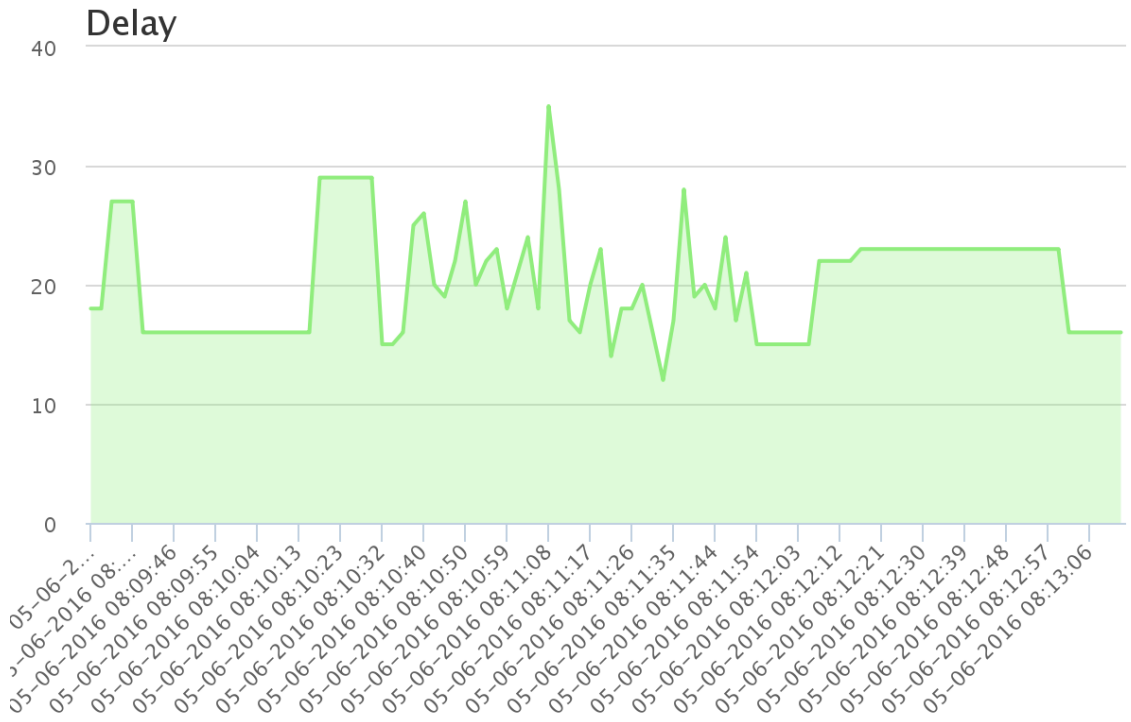


Figura 5.13: Métrica delay de escenario 1 y topología 2.

De los datos obtenidos de dichas mediciones, se puede concluir que, a pesar de ser un camino más largo, la pérdida no ha sido muy alta, similar al escenario 1 de la topología 1. Se puede observar este hecho en la Tabla 5.7.

Lo que sí se ha producido a tenor de la Figura 5.13, es un retraso en el tiempo de transmisión desde el host origen al destino. Como se ha comentado antes, la ruta de la información es más larga que en la Topología 1.

5.3.2.2. Escenario 2

Se ha comentado anteriormente que las pruebas realizadas en el escenario 2 son más agresivas, para ver el rendimiento ofrecido por la red en este caso. El tamaño del vídeo se ha reducido considerablemente (Tabla 5.8), hasta el punto de concluir que el archivo de vídeo no se visualizará correctamente (ha perdido la mitad de su tamaño) como se muestra en la Figura 5.14.

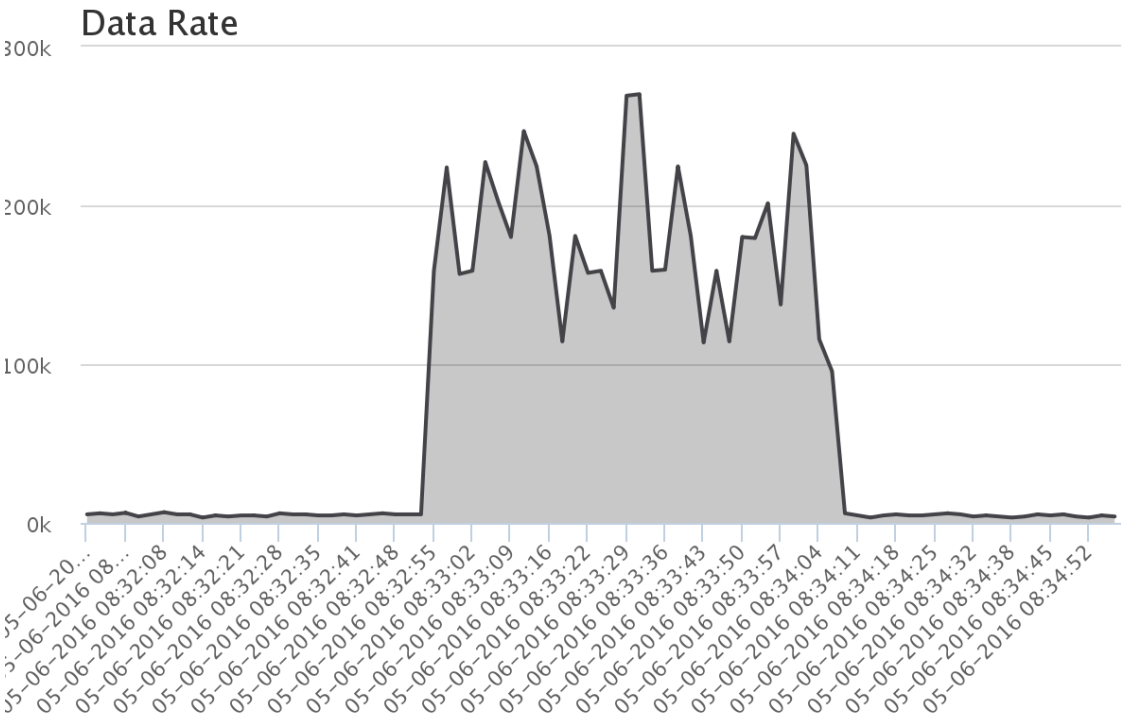


Figura 5.14: Métrica data rate de escenario 2 y topología 2.

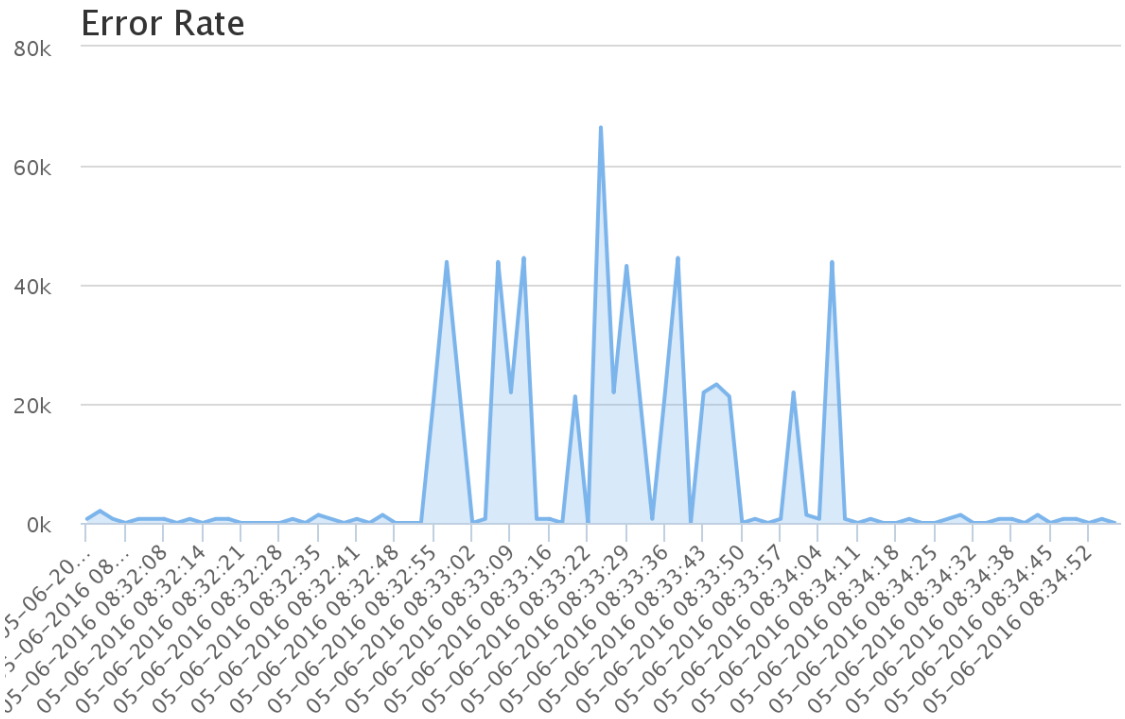


Figura 5.15: Métrica error rate de escenario 2 y topología 2.

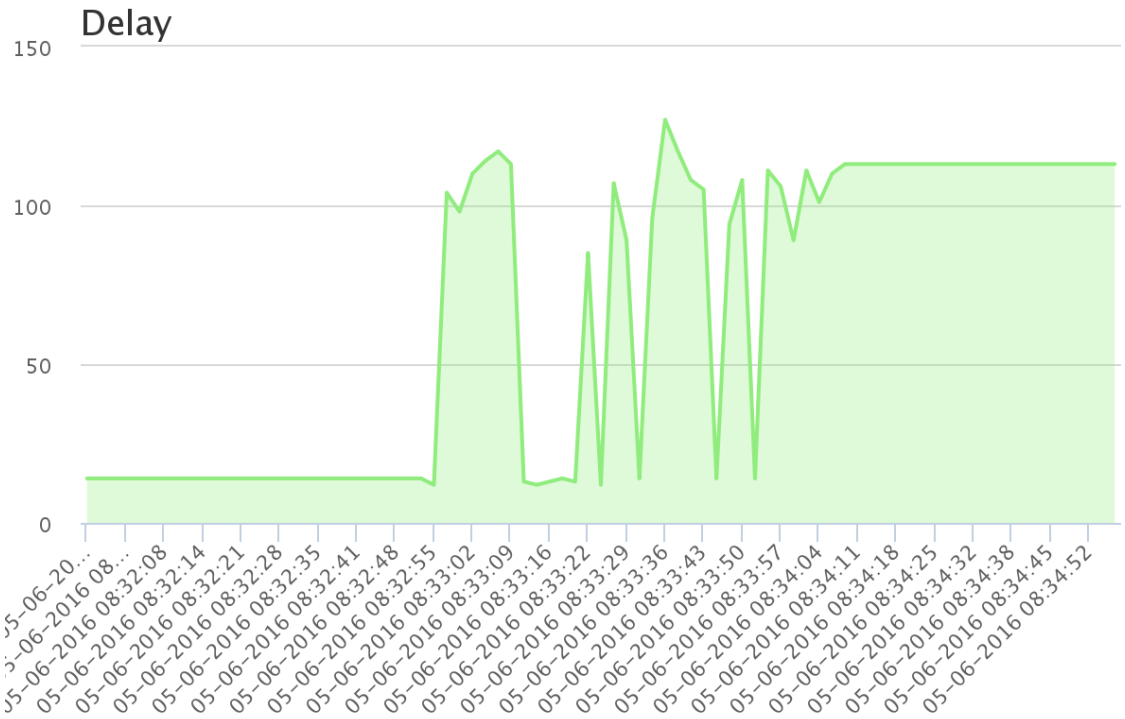


Figura 5.16: Métrica delay de escenario 2 y topología 2.

Al analizar las métricas obtenidas, se puede observar cómo se ha reducido la tasa de datos transmitidos (debido al porcentaje de pérdida de paquetes introducido) y el aumento en la tasa de errores constantes en el tiempo. De ahí que el vídeo haya perdido casi la mitad de su tamaño (Tabla 5.8).

Una vez realizadas todas las mediciones de los distintos escenarios, se puede ver en la Tabla 5.7 la tasa de errores y la tasa del flujo de datos obtenidos. De esta forma, se puede diferenciar el porcentaje de errores que se ha producido durante el envío del vídeo del host origen al destino en las topologías probadas. Para ello, se ha calculado el porcentaje de error de la siguiente manera:

$$\%\, Error \;=\; (Tasa\ de\ Errores\ Máxima \;\div\; Tasa\ de\ Datos\ en\ ese\ instante) \times 100$$

	Tasa de Errores Máxima	Tasa de Datos en ese instante	Porcentaje de Error
Topología 1 Escenario 1	21920 Kbit	300000 Kbit	7,3 %
Topología 1 Escenario 2	87680 Kbit	300000 Kbit	29,2 %
Topología 2 Escenario 1	23936 Kbit	245152 Kbit	9,7 %
Topología 2 Escenario 2	66432 Kbit	158816 Kbit	41,8 %

Tabla 5.7: Mediciones características

Como conclusión, a la vista de las Tablas 5.8 y 5.9, se puede observar cómo han variado las características del vídeo durante su transmisión, tanto en tamaño, como en duración y formato. En la Tabla 5.8 se muestran las características del vídeo original, mientras que en la Tabla 5.9, se muestran sólo los datos que han cambiado durante la transmisión.

	Tamaño del vídeo	Duración	Resolución	Tasa de Datos	Códec	Protocolo de Transmisión
Vídeo Original	2.572.968 bytes	79 segundos	352x288	257 kbps	MPEG-2	RTP

Tabla 5.8: Características del vídeo original

	Topología 1 Escenario 1	Topología 1 Escenario 2	Topología 2 Escenario 1	Topología 2 Escenario 2
Tamaño del vídeo	2.514.876 bytes	2.227.800 bytes	2.425.764 bytes	1.279.904 bytes
Duración	77 segundos	77 segundos	77 segundos	71 segundos
Tasa de Datos	250 kbps	228 kbps	250 kbps	142 kbps

Tabla 5.9: Características del vídeo transmitido

Además, se adjunta en la Tabla 5.10 la calidad de los distintos fotogramas tomados del vídeo final, una vez que ha sido transmitido a través de las distintas redes. Se puede ver claramente como en el Escenario 1 de ambas Topologías, la calidad de vídeo es similar.
En el Escenario 2 de la Topología 1, la calidad del vídeo se ha reducido ligeramente, mientras que en la Topología 2, es prácticamente nula, dado que se ve pixelado a lo largo de la totalidad del mismo.

Visualización	Escenario 1	Escenario 2
Topología 1		
Topología 2		

Tabla 5.10: Visualización de la calidad del vídeo transmitido

5.3.3. ALERTAS

A continuación, se muestra en las Figuras 5.16, 5.17 y 5.18 las distintas alertas registradas durante un período de pruebas determinado, simplemente para mostrar cómo se ha formado la estructura de las mismas:

```
Alerta
-------------
Fecha de Inicio: 2016-04-14 15:54:09 a Fecha de Fin: 2016-04-14 15:54:54
Data Rate supera el umbral de 1141.12 en el link:
Source Switch: 2 Destination Switch: 3
Source Port: 3 Destination Port: 2

Alerta
-------------
Fecha de Inicio: 2016-04-14 15:54:09 a Fecha de Fin: 2016-04-14 15:54:54
Data Rate supera el umbral de 1141.12 en el link:
Source Switch: 2 Destination Switch: 3
Source Port: 3 Destination Port: 2

Alerta
-------------
Fecha de Inicio: 2016-04-14 15:54:09 a Fecha de Fin: 2016-04-14 15:54:54
Data Rate supera el umbral de 1141.12 en el link:
Source Switch: 2 Destination Switch: 3
Source Port: 3 Destination Port: 2
```

Figura 5.17: Alertas en la Tasa de Datos.

```
Alerta
-------------
Fecha de Inicio: 2016-04-14 15:54:54 a Fecha de Fin: 2016-04-14 15:54:54
Delay supera el umbral de 74.64 en el link:
Source Switch: 1 Destination Switch: 2
Source Port: 2 Destination Port: 2

Alerta
-------------
Fecha de Inicio: 2016-05-12 02:00:00 a Fecha de Fin: 2016-05-12 02:00:00
Delay supera el umbral de 74.64 en el link:
Source Switch: 1 Destination Switch: 2
Source Port: 2 Destination Port: 2

Alerta
-------------
Fecha de Inicio: 2016-04-14 15:54:09 a Fecha de Fin: 2016-04-14 15:54:54
Delay supera el umbral de 8.96 en el link:
Source Switch: 2 Destination Switch: 3
Source Port: 3 Destination Port: 2

Alerta
-------------
Fecha de Inicio: 2016-04-14 15:54:09 a Fecha de Fin: 2016-04-14 15:54:54
Delay supera el umbral de 81.06 en el link:
Source Switch: 2 Destination Switch: 1
Source Port: 2 Destination Port: 2
```

Figura 5.18: Alertas en la Tasa de Errores.

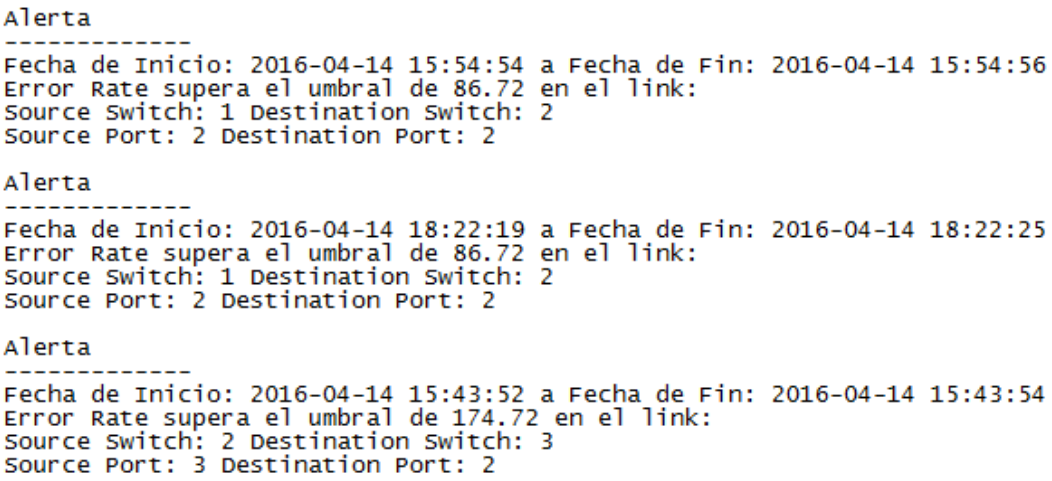


Figura 5.19: Alertas en el retardo.

6. CONCLUSIONES Y TRABAJO FUTURO

6.1. CONCLUSIONES

Tras la implementación de la arquitectura que propone este proyecto, se consigue constatar que, para mejorar la gestión de una red SDN, es necesario tener herramientas capaces de monitorizar y analizar los distintos sucesos que ocurren en un instante de tiempo determinado. Con esta implementación se ha conseguido proporcionar un mecanismo gráfico que da una visión detallada de lo que está ocurriendo en la red.

Gracias a la división de la arquitectura en módulos independientes, se ha comprobado que cada capa que la compone realiza su labor de forma correcta. En primer lugar, se consigue que la capa de monitorización genere las topologías a monitorizar y capture las distintas métricas a analizar. En segundo lugar, la capa de persistencia es capaz de almacenar los datos relacionados con los resultados de la capa de monitorización de forma íntegra, de tal manera que siempre se encuentren disponibles. Finalmente, la capa de transferencia realiza peticiones y captura las respuestas según el administrador de la red las requiera en la capa de visualización. Gracias a la flexibilidad ofrecida en el diseño responsive de la aplicación se muestran los resultados de las métricas analizadas de una forma sencilla, completa e intuitiva al administrador, para que tenga en todo momento una visión real del sistema.

Se puede concluir, por tanto, que, a través del diseño y la implementación de la arquitectura del presente proyecto, se ha cubierto una de las necesidades más imperativas de las redes definidas por software: tener una herramienta capaz de enseñar de forma intuitiva el estado de una determinada red, permitiendo a los desarrolladores añadir funcionalidades nuevas deseadas, o cambiar la tecnología de cualquiera de las capas, gracias al diseño modular de las mismas.

6.2. TRABAJO FUTURO

En el marco de la monitorización de las redes SDN existe un número limitado de aplicaciones que subsanen esta necesidad. Es por esto que, aunque la arquitectura de este proyecto proporciona una visión bastante completa de los eventos que ocurren en la red, se establece un análisis limitado de las métricas.

Así, en esta implementación se ha cubierto el análisis de las métricas de pérdidas, tasa de datos y retardo. Sin embargo, existen otros tipos de escenarios y de métricas que proporcionan información adicional de lo que está ocurriendo en la red. Por lo cual, para ofrecer mecanismos de análisis más completos, se podría añadir más parámetros para su análisis en las distintas capas que componen la arquitectura.

Además, para gestionar los datos de la arquitectura se ha utilizado un sistema gestor de base de datos relacional. En este sentido, para conseguir mayor universalidad y elevar la cantidad de datos a analizar, sería conveniente sustituir el módulo de almacenamiento por un sistema gestor de base de datos no relacional como MongoDB, HBase o inclusive realizar pruebas con Impala. De esta forma se puede procesar una mayor cantidad de información y conseguir un mayor rendimiento a la hora de llevar a cabo el tratamiento de los datos. Esto traería como consecuencia, cambiar la versión del API.

También es conveniente comparar los resultados que arroja la arquitectura con implementaciones de topologías más complejas, como por ejemplo una topología de tipo Data Center en la que los dispositivos se organizan en forma de árbol, y cada uno de ellos lleva a cabo una labor específica.

Finalmente, otra de las consideraciones a tener en cuenta, es añadir más mecanismos de alerta que complementen los ya desarrollados en este proyecto. Este hecho, beneficiaría a los administradores porque disminuiría el tiempo de reacción en el caso de que ocurriese un error inesperado.

6.3. CONCLUSIONS

After the implementation of the architecture proposed in this project, it is necessary to improve tools that can monitor and analyze the different events that occur in an instant of time during the management of a SDN network.

With this implementation it has been obtained a graphical mechanism that gives a detailed view of what is happening in the network. Thanks to the division of architecture into separate modules, it was found that each composing layer does its work correctly.

First of all, it is achieved that the monitoring layer generates topologies to capture the different metrics to analyze. Then, the persistence layer is capable of storing integrally related data with results from the monitoring layer, and make it always available.

Finally, the transfer layer requests and captures the information as a response to the required petition of the network administrator in the display layer. Thanks to the flexibility represented in the responsive application design, the results are analyzed in a simple, complete and intuitive way so the administrator has at all times a real vision of the system.

Therefore, it can be concluded, that through the design and implementation of the architecture of this project, one of the most imperative needs of software defined networking has covered: having a capable tool to show status intuitively, allowing developers to add new functionality desired or changing any of the layers. It is possible thanks to the modularized design.

7. DIVISIÓN DE TRABAJO

En este apartado se procede a describir las tareas realizadas tanto conjunta como individualmente por parte de los integrantes del equipo de desarrollo de la arquitectura. La tónica general seguida para el desarrollo e implementación del trabajo ha sido una división sistemática y equitativa del trabajo. Para conseguirlo se han repartido las tareas entre todos los integrantes del grupo por igual, sin dividir las distintas fases del desarrollo de forma específica. Además, la gran mayoría del desarrollo se ha realizado de forma conjunta estando presentes los tres miembros del equipo en todo momento. Gracias a esta metodología de trabajo, se ha conseguido un alto nivel de comunicación y debate tanto para las decisiones relacionadas con la implementación como para la planificación a seguir. Finalmente, la colaboración entre los tres miembros ha enriquecido a cada uno, no solo en el aspecto técnico sino en el personal y ha favorecido la toma de contacto con un sistema de desarrollo real enmarcado dentro de la entrega de un producto final.

7.1. FEDERICO IBÁÑEZ MORUNO

En el proyecto "Diseño e implementación de una herramienta de visualización para análisis en tiempo real de redes SDN/Openflow" se definieron diferentes etapas a seguir para cumplir con los objetivos planteados en las primeras reuniones con GASS.

Federico, en la primera etapa de proyecto, realizó un curso de SDN ofertado por Coursera con el fin de obtener los conocimientos tantos teóricos como prácticos de SDN. Centró su interés en las siguientes temáticas: el plano de datos y la herramienta Mininet. Además, realizó la labor de investigación de papers y trabajos oficiales relacionados con las funcionalidades de las redes definidas por software. El proceso de la investigación dio a Federico una idea general de todo lo relacionado con esta novedosa tecnología.

Durante la segunda etapa del proyecto, y una vez adquiridas las bases de las redes definidas por software, Federico se dedicó a investigar papers relacionados con la tecnología y representativos del proyecto que se iba a realizar. En concreto, revisó los siguientes:

- Mininet: una herramienta versátil para emulación y prototipado de Redes Definidas por Software. [Mi1]
- Obtener estadísticas del tráfico de dispositivos de red en entornos SDN usando OpenFlow. [HYTNOR]

Una de las primeras tareas a realizar para comenzar con la arquitectura del proyecto, era definir los elementos característicos del plano de datos. Federico se encargó de estudiar las distintas topologías de red a utilizar, a través de la herramienta Mininet. También se encargó de estudiar el funcionamiento del controlador Floodlight, así como el módulo de monitorización de estadísticas creado previamente por trabajos anteriores: "gass_mon_leo". Federico cambió la configuración del mismo para incluir un parámetro importante en la visualización de los datos: el instante de tiempo o timestamp.

Llegados a este punto, los miembros del equipo del proyecto, desarrollaron un cliente de monitorización con el entorno Spring y REST Template, de tal forma que se podían instanciar los datos proporcionados por Floodlight, incluso desde un equipo distinto del controlador. A partir de aquí, Federico se encargó de adaptar los datos recibidos a una base de datos, para su persistencia.

En la fase final del proyecto, Federico ha colaborado con sus compañeros en la correcta creación y adaptación de un API REST para consultar la información de la base de datos y proveerla a un cliente, de tal forma que pueda ser representada gráficamente. Al mismo tiempo, se dedicó a desarrollar un sistema de alertas que fuera capaz de analizar los datos y guardar en un fichero de eventos las posibles alertas que se puedan definir.

Para la labor de configuración del servidor encargado de alojar las distintas máquinas virtuales que contienen los distintos módulos de cada capa de la arquitectura, todos los integrantes del equipo colaboraron de forma conjunta y dinámica para la puesta en marcha.

Durante la escritura de la memoria, Federico ha sido el encargado de realizar las pruebas con las distintas máquinas virtuales, para obtener los distintos pantallazos que han dado lugar a los resultados finales. Cabe destacar que en la localización de recursos científicos y oficiales que apoyen los datos y tecnologías mencionados en este documento, el equipo se ha implicado en un grado superlativo de lectura y análisis para conseguir fundamentar de forma veraz todo lo mencionado en este documento.

7.2. JESÚS ALEJANDRO LÉVANO LÉVANO

En el proyecto “Diseño e implementación de una herramienta de visualización para análisis en tiempo real de redes SDN/Openflow” se definieron diferentes etapas a seguir para cumplir con los objetivos planteados en las primeras reuniones con GASS.

Jesús, en la primera etapa de proyecto, realizó un curso de SDN ofertado por Coursera con el fin de obtener conocimientos tantos teóricos como prácticos de SDN. Centró su interés en las siguientes temáticas: Plano de control de arquitectura SDN y NFV. Además, realizó la labor de investigación de papers y trabajos oficiales relacionados con las funcionalidades de las redes definidas por software. Los procesos de la investigación dieron a Jesús una idea general de todo lo relacionado con esta novedosa tecnología.

Durante la segunda etapa del proyecto, y una vez adquiridas las bases de las redes definidas por software, Jesús se dedicó a investigar papers relacionados con la tecnología y representativos del proyecto que se iba a realizar. En concreto, revisó los siguientes: Payless [RBAB14] y FlowSense [YLZSJV13].

Tras la aportación de Federico en el análisis del plano de datos y plano de control, se obtuvieron resultados favorables, permitiendo continuar con la etapa de planificación y planteamiento de las características del proyecto. Todos los miembros del equipo participaron con el aporte y la propuesta de ideas y tecnologías que puedan cubrir la funcionalidad REST. Se acordó realizar una reunión en la que se llevó a cabo una “Lluvia de ideas”.

Finalmente, entre las diferentes ideas, se eligió la propuesta por Jesús. Ésta idea supuso el prototipado de la primera versión de un módulo de la capa de monitorización que se desarrolló en base a Spring Boot y las librerías que éste ofrecía. Esto se concretaría en el módulo “consumidor” estable de la capa de persistencia.

Para definir los siguientes módulos que conforman la arquitectura, todos los miembros del equipo buscaron el mecanismo más efectivo a seguir para su implementación. Éstos deberían favorecer la flexibilidad y adaptación a nuevas tecnologías. Por ello se acordó continuar con el desarrollo en Spring Boot y la explotación de sus módulos. Jesús organizó y propuso la arquitectura que se presenta en este proyecto, a partir de esto ha colaborado con las versiones del desarrollo del API REST, aquí ha desarrollado la estructura JSON y los métodos que permitieron constatar que el proyecto iba encaminado, también del mantenimiento de versiones estables, y de la implementación para recuperar la información de la base de datos.

En la parte dedicada a la visualización, todos los miembros del equipo realizaron una labor de estudio y aprendizaje de forma activa de las tecnologías asociadas a los entornos web. Concretamente todos los miembros del equipo contribuyeron con el desarrollo de los gráficos, implementados con Highcharts, que componen la herramienta que se muestra a los administradores de la red.

Jesús es quien lleva el control de versiones, eligió que, durante el desarrollo de este proyecto, el repositorio se mantendría como privado hasta la presentación de este proyecto. Por otro lado, se ha encargado de crear la máquina virtual instalando todo el software necesario para la ejecución en el servidor.

Finalmente es destacable que en la redacción de la memoria y en la localización de recursos científicos y oficiales que apoyen los datos y tecnologías mencionados en este documento, el equipo se ha implicado en un grado superlativo de lectura y análisis para conseguir fundamentar de forma veraz todo lo mencionado en te documento.

7.3. ERIK STEVE NIETO MALDONADO

Para el desarrollo del presente proyecto, Diseño e Implementación de una Herramienta de Visualización para el Análisis en Tiempo Real de Redes SDN/Openflow, se fijaron, en varias reuniones las fases que serían necesarias para conseguir los objetivos y propósitos de este proyecto.

Durante la primera fase del proyecto, Erik realizó un curso de SDN ofertado por Coursera para adquirir las competencias teóricas y prácticas necesarias para entender el marco en el

que se enfoca SDN. Centró su interés en las siguientes temáticas: problemas y retos a los que se enfrenta, componentes y lenguajes de programación de SDN. Además, realizó la labor de investigación de papers y trabajos oficiales relacionados con las funcionalidades de SDN.

En la segunda fase ya teniendo una idea general y completa de las redes definidas por software, dedicó sus esfuerzos al estudio de distintos trabajos de investigación para tener una visión más detallada de los conceptos relacionados con las redes definidas por software. Estos trabajos son: Funciones de virtualización de la red para la implementación en la nueva generación de redes móviles. [CINGNM14] y Monitorización de la latencia que se produce en redes OpenFlow. [PB13]

En la tercera etapa, se planifica qué características debe tener el proyecto. En este sentido, destaca que todos los miembros, a través de una lluvia de ideas, presentan las posibles tecnologías para cubrir las necesidades que presenta REST. Una vez estudiadas, se elige la propuesta de Spring Boot, ya que es la que más encaja con el desarrollo de las capas de persistencia y transferencia. Ésta idea supuso el prototipado de la primera versión. En la capa de persistencia, destaca la labor realizada por Erik, en la creación de un esquema relacional del sistema gestor de base de datos. Más tarde y junto a sus compañeros, se decidió el esquema final al que se debe ajustar el diseño. Además, ha participado en el desarrollo de la base de la arquitectura API REST propuesta.

En una cuarta etapa, Erik se ha encargado de realizar un análisis funcional de los requisitos necesarios que debe contener la capa de transferencia para producir en el formato JSON de Highcharts adecuado los datos que necesita el sistema de visualización en la siguiente capa. También se ha encargado del desarrollo de los templates que contienen las representaciones de los datos analizados en forma de gráfica. Para ello, en primer lugar, desarrolló una implementación completa de un Dashboard con gráficas de Highcharts con la tecnología ASP.Net, para más tarde realizar una versión definitiva utilizando Bootstrap como Framework de diseño responsive, de forma que, se adapte a las necesidades del proyecto. Junto con sus compañeros ha contribuido con el desarrollo de los gráficos, implementados con Highcharts, que componen la herramienta que se muestra a los administradores de la red.

Para la labor de configuración del servidor encargado de alojar las distintas máquinas virtuales, todos los integrantes del equipo colaboraron de forma conjunta y dinámica para la puesta en marcha.

Finalmente, Erik ha participado de manera proactiva en la redacción de la memoria, ya que además de la labor de escritura conjunta con sus compañeros, ha sido, el encargado de traducir al inglés, los elementos requeridos. También es destacable que ha participado junto con el equipo en la localización de recursos científicos y oficiales que apoyen con veracidad los datos y tecnologías mencionados en este documento.

BIBLIOGRAFÍA

[ACETV14] Proyecto Fin de Máster en Sistemas Inteligentes. Algoritmo de Calidad de Experiencia para Transmisiones de Video en Redes Definidas por Software. Jesús Antonio Puente Fernández. 2014

[AMCSM15] Trabajo de Fin de Máster en Ingeniería Informática. Arquitectura para la Mejora de la Calidad de Servicios Multimedia en Redes Definidas por Software. Marco Antonio Sotelo Monge. 2014-2015

[BGOPWEB] Beginners Guide Overview Python. <https://wiki.python.org/moin/BeginnersGuide/Overview>

[BRAB13] Md. Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba - David R. Cheriton School of Computer Science, University of Waterloo. "PolicyCop: an autonomic QoS policy enforcement framework for software defined networks." *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for* 11 Nov. 2013: 1-7.

[BTT16] Free Bootstrap Themes and website templates 2016. <bootstraptaste.com >

[CCT14] Gabrielsson, Jan; Hubertsson, Ola; Más, Ignacio; Skog, Robert. Cloud computing in telecomunications, January 2014

[CFVMET] Ángel Leonardo Valdivieso Caraguay, Jesús Antonio Puente Fernández, Luis Javier García Villalba. “An Optimization Framework for Monitoring of SDN/OpenFlow Networks”.

[CINGNM14] IEEE Network Mag., November 2014. NFV: State of the Art,Challenges and Implementation in Next Generation Mobile Networks (vEPC)

[CORSWEB] Cross-Origin Resource Sharing. <https://spring.io/understanding/CORS>

[CRS15] Talita De Paula Cypriano De Souza ; Ericsson Res., Indaiatuba, Brazil ; Christian Esteve Rothenberg ; Mateus Augusto Silva Santos ; Luciano Bernardes De Paula. "Towards Semantic Network Models via Graph Databases for SDN Applications." *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on* 30 Sep. 2015: 49-54.

[CSIS14] MonSamp: A distributed SDN application for QoS monitoring Computer Science and Information Systems (FedCSIS). 14.

[FOMR15] Framework for optimized multimedia routing over software defined networks Ángel Leonardo Valdivieso Caraguay, Jesús Antonio Puente Fernández, Luis Javier García Villalba. 2015.

[HABRZ15] Pedro Heleno Isolani, Juliano Araujo Wickboldt, Cristiano Bonato Both, Juergen Rochol, and Lisandro Zambenedetti Granville - Federal University of Rio Grande do Sul, Porto Alegre, Brasil; University of Santa Cruz do Sul, Santa Cruz do Sul, Brasil. "SDN interactive manager: An OpenFlow-based SDN manager." *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on* 11 May. 2015: 1157-1158.

[HGJL15] Bo Han, Vijay Gopalakrishnan, Lusheng Ji and Seungjoon Lee - AT&T Labs Research Bedminster, NJ 07921, USA. "Network function virtualization: Challenges and opportunities for innovations." *Communications Magazine, IEEE* 53.2 (2015): 90-97.

[HYTNOR] Diyar Jamal Hamad, Khirota Gorgees Yalda, and Ibrahim Tanner Okumus. "Getting traffic statistics from network devices in an SDN environment using OpenFlow".

[ITMWEB] Introduction to Mininet. <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#what>

[LPL14] A Lightweight Portability Layer for the SDN OS. Minlan Yu. USC. Andreas Wundsam. Big Switch Networks. Muruganantham Raju. 14.

[LLA14] *Jun Liu, Feng Liu, and Nirwan Ansar.* "Monitoring and analyzing big traffic data of a large-scale cellular network with Hadoop." *Network, IEEE* 28.4 (2014): 32-39.

[Mi1] B. Valencia, S. Santacruz, L.Y. Becerra y J.J. Padilla. "Mininet: a versatile tool for emulation and prototyping of Software Defined Networking". Entre Ciencia e Ingeniería, ISSN 1909-8367. Año 9 No. 17 - Primer Semestre de 2015, página 62 - 70

[MCSS13] Modular Composable Security Services for Software-Defined Networks. 10 Feb - NDSS 2013 Paper - FRESCO.

[MDB1] MongoDB <https://www.mongodb.com/es>

[Mi1] B. Valencia, S. Santacruz, L.Y. Becerra y J.J. Padilla. “Mininet: a versatile tool for emulation and prototyping of Software Defined Networking”. Entre Ciencia e Ingeniería, ISSN 1909-8367. Año 9 No. 17 - Primer Semestre de 2015, página 62 - 70

[MSQL1] MySQL <https://www.mysql.com/>

[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.

[NSMOFC12] IEEE International Conference on Network Protocols (ICNP). 12

[ODL16] The OpenDaylight Platform <https://www.opendaylight.org/>

[Op1] Open Networking Foundation <https://www.opennetworking.org/>

[Ope12] Open Networking Foundation Software-Defined Networking: The new Norm for Networks, April 13,2012

[PB13] Kevin Phemius and Mathieu Bouet - Thales Communications & Security, Paris, France. "Monitoring latency with openflow." Network and Service Management (CNSM), 2013 9th International Conference on 14 Oct. 2013: 122-125.

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

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

[QDGBV14] Zhijing Qin, Grit Denker, Carlo Giannelli, Paolo Bellavista, Nalini Venkatasubramanian. "A software defined networking architecture for the internet-of-things." *Network Operations and Management Symposium (NOMS), 2014 IEEE* 5 May. 2014: 1-9.

[RBAB14] Shihabur Rahman Chowdhury, Md. Faizul Bari, Reaz Ahmed, and Raouf Boutaba - David R. Cheriton School of Computer Science, University of Waterloo. "Payless: A low cost network monitoring framework for software defined networks." *Network Operations and Management Symposium (NOMS), 2014 IEEE* 5 May. 2014: 1-9.

[SAO12] SDN Security - An Oxymoron. 12 Feb - SDN Central Interview regarding our NDSS FRESCO

[SBSSWEB] Spring Boot, Simplifying Spring for everyone.
<https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone>

[SDWEB] Spring Data <http://projects.spring.io/spring-data/>

[SOI15] SevOne. Solutions for Software Defined Network Monitoring.
<ww.sevone.com>

[SFWEB] Spring Framework <https://projects.spring.io/spring-framework/>

[SSHC+13] S. Sezer, S. Scott-Hayward, P. 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, 51(7):36-43, July 2013.

[S13WEB] "Spring." 2013. <https://spring.io/>

[W3CWEB] W3C. <http://www.w3c.es/>

[YLZSJV13] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V. Madhyastha. "Flowsense: Monitoring network utilization with zero measurement cost." *Passive and Active Measurement* 18 Mar. 2013: 31-41.