TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Redes de distribución de contenidos basadas en redes definidas por software

Autor
Roberto Jiménez Sánchez

Directores
Juan Manuel López Soler
Jorge Navarro Ortiz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Septiembre de 2016

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Redes de distribución de contenidos basadas en redes definidas por software

Autor
Roberto Jiménez Sánchez

Directores
Juan Manuel López Soler
Jorge Navarro Ortiz

Departamento de Teoría de la Señal, Telemática y Comunicaciones

—

Granada, Septiembre de 2016

Redes de distribución de contenido basadas en redes definidas por software

Roberto Jiménez Sánchez

Palabras clave: Software Defined Networking, OpenDaylight, Mininet, Content Delivery Network, Apache Traffic Server, Balanceo de carga, Controlador, Cluster, Java.

Resumen

El uso de nuevas tecnologías, y la aparición de nuevas necesidades en la sociedad, ha traído consigo un incremento en el número de dispositivos conectados a Internet. Este incremento, ha provocado un aumento considerable del tráfico de red. La gestión de tal cantidad de tráfico ha dado lugar a que los proveedores de servicios e *Internet Service Provider* (ISP) busquen mecanismos para mejorar los servicios ofrecidos, así como reducir el tráfico en la red troncal. Es por ello que han surgido nuevos métodos de distribución de contenidos, que tratan de paliar los efectos que se producen al tener que ofrecer contenidos a una gran cantidad de usuarios simultáneamente.

Content Delivery Network (CDN), es un sistema que, mediante la replicación de los contenidos, normalmente en localizaciones más cercanas al usuario, permite al cliente acceder de forma más rápida al contenido. Por lo general, las CDNs reducen el tráfico total en la red, beneficiando tanto a proveedores de servicios como proveedores de infraestructuras.

Actualmente, la mayor parte del tráfico de Internet se cursa por las CDN, situándose como un elemento fundamental, ya no solo en la distribución de contenidos, sino en el comportamiento general de Internet. Akamai es el mayor proveedor de CDN del mundo, sin embargo, otras empresas como Amazon o Level 3, son de gran relevancia en el mercado [18].

Con todo ello, a pesar del uso de las CDN, las redes de telecomunicación actuales, cuentan con una serie de limitaciones, razón por la cual nace el concepto de redes definidas por *software* (*Software-Defined Networking* (SDN)).

Las redes definidas por *software*, suponen una solución a una serie de problemas existentes en las redes actuales, que es la dificultad de gestionarlas, la falta de flexibilidad en la implementación de servicios, y la necesidad de disminuir costes. El modelo de SDN propone la separación del plano de control y el plano de datos. Esto se consigue mediante la centralización de los recursos de control en dispositivos llamados controladores. El controlador, tiene una visión general de la red, permitiendo gestionar con mayor versatilidad y facilidad las redes. Además, permite aplicar con mayor facilidad nuevas funcionalidades que anteriormente eran demasiado complejas de implementar. Este modelo rompe con el tipo de redes actuales, en la cual la

lógica de control y de datos se encuentra descentralizada en los dispositivos de red, dificultando la gestión de las redes y haciéndola más rígida.

A partir de esta innovación tecnológica, surge la idea de llevar a cabo el presente Trabajo de Fin de Máster (TFM). La idea es implementar un sistema de distribución de contenidos usando la versatilidad que ofrecen las redes definidas por *software*. Así pues, se hará uso del simulador de redes Mininet [11], que permitirá crear una topología de red con dispositivos de conmutación compatibles con la tecnología SDN. Con el *software* Apache Traffic Server [33] se usará un *proxy* que realice el almacenamiento temporal de la información (o cacheo). Haciendo uso de OpenDaylight [13], se dispondrá de un controlador que se comunicará con los elementos de la topología SDN. En el controlador se implementará la lógica encargada de aplicar los mecanismos de distribución de contenidos en los dispositivos de red. Finalmente, se comprobará la viabilidad de esta implementación, los aspectos positivos y negativos, y, posibles variaciones en función de las circunstancias del escenario, haciendo uso de la versatilidad que proveen las redes SDN.

Content Delivery Network using a Software-Defined Networking

Roberto Jiménez Sánchez

Keywords: Software Defined Networking, OpenDaylight, Mininet, Content Delivery Network, Apache Traffic Server, Load balancing, Controller, Cluster, Java.

Abstract

The use of new technologies, and the new needs in society, has brought an increase in the number of Internet-connected devices. This increment has resulted in an increase of network traffic. Managing such amount of traffic has led to service providers and ISPs to seek mechanisms to improve services, as well as reduce traffic on the backbone network. For this reason, has emerged new methods of content distribution, trying to reduce adverse side-effects when you have to deliver content to a large number of users simultaneously.

A Content Delivery Network, is a system that by replication of contents (usually closer to user's location), allows customers faster access to the contents. In general, CDNs reduce total network traffic, benefiting service providers and infrastructure providers.

Currently, most Internet traffic is served by CDNs, positioning itself as a key element, not only in content distribution, but in the general behavior of Internet. Akamai is the largest provider of CDN of the world, however, other companies like Amazon or Level 3 are of great importance.

Despite the use of the CDN, the present telecommunications network has a number of limitations, for this reason it is born the concept of Software-Defined Networking (SDN).

Software-Defined Networking, is the solution to a number of problems existing in current networks, such as the difficulty of managing them, the lack of flexibility and the need to reduce costs. SDN model proposes a separation of the control layer and the data layer. This is achieved by centralizing control resources in devices called controllers. The controller has an overview of the network, thus permitting an easier and more versatile network management.

From this technological innovation, it arises the idea of carrying out this project. The objective is to implement a content distribution network using the versatility of Software-Defined Networking. In order to do this, the network emulator Mininet will be used to create a network topology with SDN compatible devices. The software Apache Traffic Server will be used

as a proxy for temporary storage of information (or caching). Using OpenDaylight (ODL), there will be a controller that will communicate with the elements of the SDN topology. In ODL it will be implemented mechanisms for carry out content distribution in network devices. Finally, it will be verified the viability of this implementation, positive and negative aspects, and possible variations depending on the circumstances of the scenario, using the versatility that SDN provides.

Yo, **Roberto Jiménez Sánchez**, alumno del **Máster en Ingeniería de Telecomunicación** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Roberto Jiménez Sánchez

Granada a 16 de septiembre de 2016.

D. **Juan Manuel López Soler**, Profesor del Área de Telecomunicaciones del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

D. **Jorge Navarro Ortiz**, Profesor del Área de Telecomunicaciones del Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Redes de distribución de contenidos basadas en redes definidas por software***, ha sido realizado bajo su supervisión por **Roberto Jiménez Sánchez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 16 de Septiembre de 2016.

Los directores:

Juan Manuel López Soler **Jorge Navarro Ortiz**

Yo, **Roberto Jiménez Sánchez**, alumno del **Máster en Ingeniería de Telecomunicación** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77138105S, declaro explícitamente que el trabajo presentado es original, entendido en el sentido que no he utilizado ninguna fuente sin citarla debidamente.

Fdo: Roberto Jiménez Sánchez

Granada a 16 de septiembre de 2016.

Agradecimientos

Quiero agradecer a mi familia por apoyarme en todos los sentidos para llevar a cabo mi etapa en la Universidad y en la realización de este proyecto.

Y a mis tutores, Juanma y Jorge por darme la oportunidad de realizar este trabajo, y haber tenido la paciencia y compromiso para ayudarme en lo que necesitara.

Índice general

Índice de figuras

Índice de tablas

Acrónimos

AD-SAL API-*Driven* SAL

AJAX *Asynchronous JavaScript And XML*

ALTO *Application Layer Traffic Optimization*

ARP *Address Resolution Protocol*

ASF Apache Software Foundation

ATS Apache *Traffic Server*

API *Application Programming Interface*

CAPEX *Capital Expenditures*

CDN *Content Delivery Network*

COTS *Commercial off-the-shelf*

CPE *Customer-Premises Equipment*

DHCP *Dynamic Host Configuration Protocol*

DHT *Distributed Hash Tables*

DNS *Domain Name System*

DSA *Dynamic Site Acceleration*

DSCP *Differentiated services Code Point*

EPL Eclipse Public License

FOSS *Free and Open Source Software*

HTML *HyperText Markup Language*

HTTP *Hypertext Transfer Protocol*

IP *Internet Protocol*

ISP *Internet Service Provider*

MAC *Media Access Control*

MD-SAL *Model-Driven* SAL

NAT *Network Address Translation*

NETCONF *Network Configuration Protocol*

NFV *Network Functions Virtualization*

NFV-MANO *Network functions virtualization management and orchestration architectural framework*

NFVI *Network Function Virtualization Infraestructure*

ODL OpenDaylight

ONF *Open Networking Foundation*

ONOS *Open Network Operating System*

OPEX *Operating Expense*

OSGi *Open Services Gateway Imitative*

OVS Open vSwitch

PC *Personal Computer*

P2P *Peer to Peer*

PoP *Point of Presence*

QoS *Quality of Service*

REST *Representational State Transfer*

RTT *Round-Trip delay Time*

RPC *Remote Procedure Call*

TLD *Top Level Domain*

SAL *Service Abstraction Layer*

SDN *Software-Defined Networking*

SPOF *Single Point of Failure*

SSD *Solid-State Disk*

TCO *Total Cost of Ownership*

TCP *Transmission Control Protocol*

TFM Trabajo de Fin de Máster

TSTC Teoría de la Señal, Telemática y Comunicaciones

URI *Universal Resource Identifiers*

URL *Universal Resource Locator*

VNF *Virtualize Network Function*

XML *Extensible Markup Language*

YANG *Yet Another Next Generation*

Capítulo 1

Introducción

En los últimos años la complejidad de las redes de telecomunicación ha ido incrementando. Se necesitan más elementos de red para soportar los servicios requeridos, y a su vez cada elemento de red necesita integrar más funcionalidades. Tradicionalmente los equipos de red, como *routers* o *switches*, solían ser cerrados, con sistemas específicos (a menudo propietarios), por tanto, la correcta gestión de la red se ha convertido en un objetivo muy importante en un entorno donde crear configuraciones individualizadas en multitud de elementos crea un incremento de gasto de recursos no deseado. La programación de redes aborda dicho problema simplificando en gran medida la gestión de la red, de forma que se facilita la programación de servicios extremo a extremo sin tener que entrar tan a fondo en configuraciones individuales. Con las Redes Definidas por *Software* (SDN) precisamente se consigue ese objetivo: ser capaces de modificar el comportamiento de la red de manera controlada, haciéndola más flexible, escalable y reprogramable [70]. Mediante el uso de APIs (Interfaces de programación de aplicaciones) se puede configurar aplicaciones que modifiquen de forma dinámica los servicios, maximizando la fiabilidad y rendimiento de cada servicio o usuario, generando nuevas oportunidades de negocio y optimizando el CAPEX y OPEX.

Además de la complejidad de la red, el tráfico de red ha sufrido un enorme crecimiento [69], es por ello que surge la necesidad de utilizar elementos que ayuden a aumentar la eficiencia en la red, como las *Content Delivery Network* (CDN) [48]. Se tratan arquitecturas de red o *framework* que soportan el alojamiento y distribución de contenido, permitiendo al proveedor de contenido replicar y servir sus contenidos más populares en una serie de puntos alojados en diferentes localizaciones, dando una gestión de contenidos descentralizada. Actualmente juegan un papel fundamental en las redes, dada la gran cantidad de tráfico que gestionan, actualmente el 50 por ciento del tráfico consumido en Internet, es servido por CDNs [22].

El objetivo de este proyecto es el de implementar el escenario de una CDN haciendo uso de una red SDN, así como estudiar sus aspectos positivos y negativos. Para implementar la CDN se hará uso de ATS, y para SDN se utilizará el emulador Mininet [11] y ODL [13], una implementación en código abierto de un controlador. En dicho escenario, ATS se encargará de actualizar y sincronizar la caché, y el controlador SDN implementará la funcionalidad de balancear la carga entre distintos equipos ATS.

1.1. Iniciativa y conceptos

El diseño de infraestructuras de redes ha ido evolucionando paulatinamente. Inicialmente las infraestructuras se modelaban de forma específica según el tipo de aplicación, el ámbito o los requisitos de seguridad deseados. Con el paso del tiempo, mediante la estandarización de protocolos y servicios, el diseño ha debido ajustarse a un tipo de modelo en el que las empresas deben trabajar, haciendo menos competitivo y desafiante el diseño de redes por parte de las empresas de infraestructuras [49].

El uso de SDN facilita la inclusión de nuevos servicios, haciendo más atractivo el diseño de redes, y una buena oportunidad para nuevas empresas en el sector.

1.1.1. Software-Defined Networking

La *Open Networking Foundation* (ONF) describe las redes SDN en este artículo [38] de la siguiente forma: "En la arquitectura SDN, el plano de control y el plano de datos están desacoplados, la inteligencia y los estados de la red están centralizadas de forma lógica, y la infraestructura de red subyacente es abstracta para las aplicaciones". Como resultado se obtiene una mejor programabilidad, automatización y control en la red, permitiendo construir una red mucho más flexible, escalable. Todo esto aporta a las empresas una mayor capacidad de incorporar nuevos servicios y aplicaciones.

La ONF describe la necesidad de crear un nuevo tipo de arquitectura de red, debido a la masiva introducción de dispositivos móviles, la virtualización de servidores y la inclusión de servicios *cloud* (de la nube). Muchas redes tradicionales tienen una arquitectura jerárquica, sin embargo, dicho modelo tan estático está tendiendo a uno con mayor presencia de computación y almacenamiento dinámico.

Las principales limitaciones en las redes actuales son:

- Complejidad: Por un lado, existe un enorme número de protocolos, que tratan de forma individualizada problemas específicos.

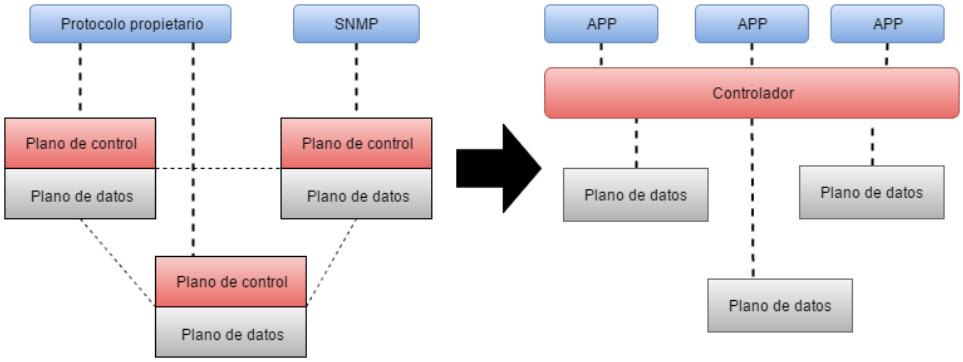


Figura 1.1: Abstracción de SDN. Modelo tradicional (izq.) y modelo SDN (der.)

- Políticas complejas: Implementar políticas en una red de cierto tamaño requiere la configuración de multitud de dispositivos y mecanismos. Incluir una sola máquina virtual nueva puede llevar hasta horas y días de incluir en la red.
- Problemas de escalabilidad: Las redes se han vuelto muy complejas debido a la gran cantidad de dispositivos de red que deben ser configurados y gestionados.
- Dependencia con los vendedores: Dicha dependencia intentará ser paliada con el uso de la arquitectura SDN, siendo desarrollada asociada a un estándar.

Las SDN intentan, por tanto, paliar los problemas asociados a las redes tradicionales enumerados anteriormente.

Las redes definidas por *software* o SDN suponen un enfoque de la red que permite abstraer de las funcionalidades de más bajo nivel a los administradores de red en su tarea de gestión. La forma en la que lo consigue es desacoplando el plano de control y el plano de datos, como se describe en el Apartado 1.1.1. De esta forma, SDN se establece para paliar problemas de las redes tradicionales de rigidez y escalabilidad. En la Figura 1.1 se puede apreciar el concepto de separación de plano de control y plano de datos de forma visual.

Arquitectura SDN

En la arquitectura SDN descrita por la ONF, los principios básicos son tres [39]:

- Separación del plano de control del plano de datos. Es decir, tener un

controlador (el sistema que toma la decisión de dónde es enviado el tráfico) separado del plano de datos (equipos por los que el tráfico es enviado).

- Centralización lógica del control. Tener un controlador que tenga una visión global de la red, para poder tomar mejores decisiones de control.
- Realizar una abstracción de los recursos de red y sus estados a aplicaciones externas. Para así tener una mejor interacción con las aplicaciones externas.

En la Figura 1.2 se muestra de forma visual estos elementos [62].

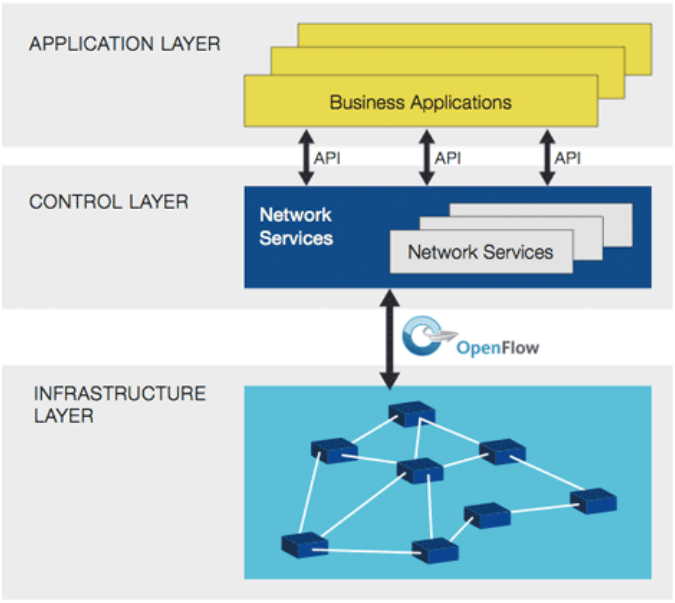


Figura 1.2: Arquitectura SDN. Elementos de aplicaciones (arriba), elementos de control (centro) y elementos de la infraestructura (abajo)

Como se puede apreciar, la arquitectura contempla principalmente tres capas:

- Infraestructura (capa de datos): Se trata de la infraestructura de red, es decir el conjunto de elementos de red, como *switches*, *routers* o *hosts*.
- Controlador (Plano de control): Mantienen una visión global de la red. De forma que para las aplicaciones se muestra como si fuera un único *switch* lógico, sirviendo de abstracción a éstas. El controlador usa la información de la infraestructura de red para gestionarla y controlar los

flujos. La comunicación entre los planos de control y de datos se realiza mediante la *Southbound API* [63], dado el apoyo que está teniendo, el protocolo OpenFlow [16] se está convirtiendo en el estándar *de facto* (descrito más a fondo en el apartado 5.2).

- Aplicaciones: El controlador aporta abstracción a las aplicaciones de los detalles de bajo nivel de la infraestructura de red, facilitando su programación, permitiendo usar lenguajes más comunes sin depender de tecnologías específicas, al contrario de lo que ocurre con las aplicaciones de redes actuales. Se comunica con el controlador mediante la *NorthBound API*.

En la Figura 1.3 se muestra el uso de algunos protocolos y herramientas usadas típicamente en la tecnología SDN como el protocolo *de facto* OpenFlow, controladores como POX o FloodLight y tecnología usada en la *Northbound API* como REST.

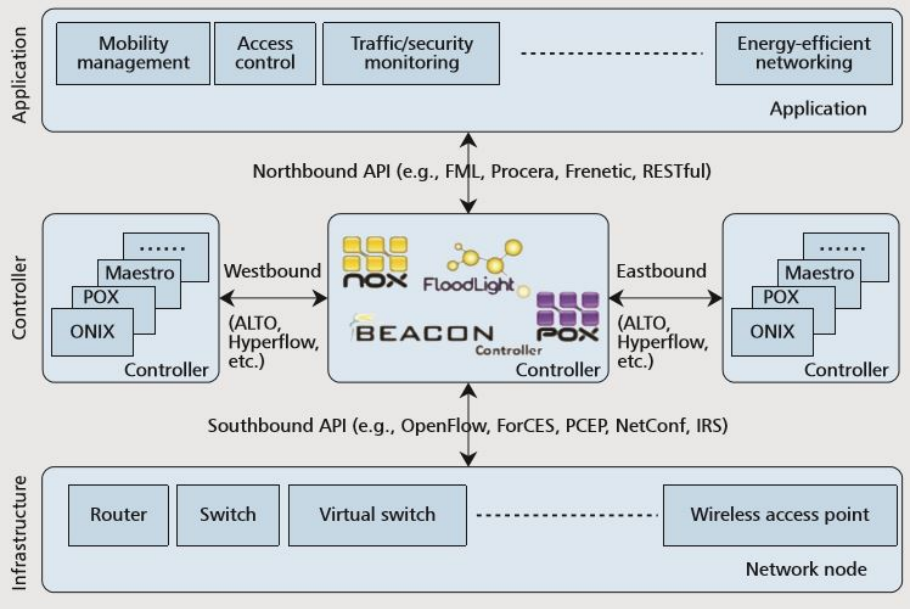


Figura 1.3: Arquitectura SDN detallada. [62]

SDN vs NFV

En algunas ocasiones se habla de forma casi indistinta de SDN y *Network Functions Virtualization* (NFV). Conviene aclarar que SDN y NFV son tecnologías diferentes. Mientras que SDN supone la separación del plano de control y el de datos. NFV consiste en la virtualización de las funciones

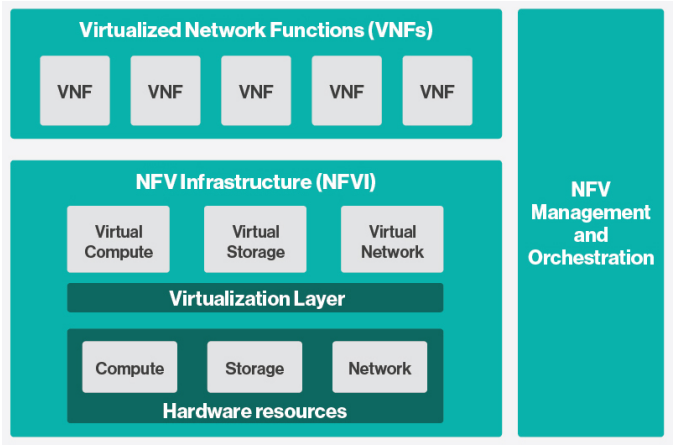


Figura 1.4: NFV *Framework*. [19]

de red, agrupando *hardware* de múltiples proveedores que operan en la red de sus clientes. NFV puede ser desplegado sin SDN y viceversa. Aunque el objetivo de ambos es controlar la red lógicamente, mediante software, y minimizar el trabajo directo con el *hardware* [60]. NFV puede ser aplicado tanto al plano de control como al de datos. Aprovechando tecnologías de virtualización (*cloud computing*), como la virtualización de *hardware* usando *hypervisors* y *switches* virtuales como Open vSwitch, para conectar tráfico entre máquinas virtuales e interfaces físicas.

NFV Framework

El *framework* NFV consiste básicamente de tres elementos [17]:

- VNF: *Virtualize Network Function*. Son implementaciones *software* de las funciones de red que pueden ser desplegadas en una *Network Function Virtualization Infraestructure* (NFVI).
- NFVI: *Network Function Virtualization Infraestructure*. Son el conjunto de componentes, tanto *hardware* como *software*, que constituyen el escenario donde las *Virtualize Network Function* (VNF) son desplegadas.
- NFV-MANO: *Network functions virtualization management and orchestration architectural framework*. Es el conjunto de bloques funcionales, repositorios de datos y puntos de referencia e interfaces por los cuales se intercambia información con el objetivo de administrar las NFVI y VNFs.

Conceptos de SDN

- Con las SDN, la infraestructura no está tan íntimamente relacionada con las funcionalidades [71], es decir, las implementaciones llevadas a cabo no dependen tanto de los componentes que conforman la infraestructura, sino que se tiene una mayor abstracción e independencia del *software* sobre el *hardware*, dotándolas de mayor versatilidad para implementar y gestionar servicios de forma más sencilla.

- Gracias a la virtualización se consigue:

  - Evitar el sobredimensionado, optimizando la repartición de tareas entre diferentes sistemas, de forma que se pueden utilizar todas las máquinas a un nivel de uso razonable.
  - Actualizar la infraestructura más fácilmente y reusar equipos, evitando depender de tener que actualizar componentes de equipos propietarios.
  - Tener una menor dependencia del *hardware*, teniendo una mayor presencia las actualizaciones de *software*.
  - Se puede especializar la red a un coste inferior.

- Los dispositivos usados tienen una menor inteligencia, pues son controlados desde el controlador, ya no a nivel de conmutación sino de red.

- La transición a IPv6 será más sencilla porque se pueden tener varios usuarios usando la tecnología IPv4/IPv6 entre sí.

- SDN será un pilar fundamental en la tecnología 5G [25].

  - Plan Horizon 2020. Se trata del mayor programa de investigación e innovación de fondos públicos europeos, con 80.000 millones de euros disponibles a lo largo de 2014-2020, además de la inversión privada atraída. Dicho plan comprende un programa de ayuda para implementar 5G-PPP (*5G-Public Private Partnership*) con un presupuesto de 700 millones para investigación, desarrollo e innovación [1].

- Proyección económica. Según IDC (*International Data Corporation*) el mercado de SDN, que comprende infraestructuras físicas, virtualización y control del *software*, aplicaciones y servicios profesionales, tendrá un valor cercano a los 12.500 millones de dólares para el año 2020. Con un crecimiento anual del 54 por ciento desde 2014 [23].

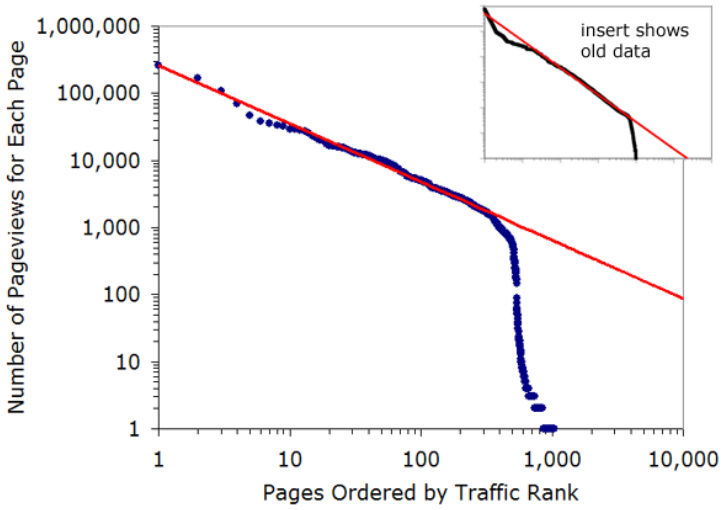


Figura 1.5: Distribución de tráfico, según la popularidad de la página web.

1.1.2. Content Delivery Network

Una *Content Delivery Network* o CDN permite llevar a cabo la distribución de contenido de forma descentralizada. Así que alguien puede conectarse al nodo más cercano, o al menos utilizado (u otros criterios, según la configuración) para que sea más eficiente el acceso a dicho contenido. Dada las demandas de tráfico, tanto en el presente como para el futuro, el uso de CDN se convierte en un pilar fundamental en las redes de telecomunicaciones.

Qué es una CDN

Las popularidad de un sitio web, el número de visitantes referidos a cada sitio, y el tráfico sobre sus consultas de búsqueda siguen una distribución zipf [55], esto se traduce en que hay multitud sitios con poco tráfico, pero unos pocos que concentran una gran cantidad de tráfico, tal y como se muestra en la Figura 1.5. Este tipo de distribución da lugar a que se concentre mucho tráfico web en unos pocos sitios, dando lugar a problemas de saturación, disponibilidad o retardos. Una forma de paliar este tipo de problema es haciendo uso de CDNs. Una CDN provee servicios que mejoran el rendimiento de la red y el uso de ancho de banda, mejorando la accesibilidad y manteniendo la exactitud del contenido mediante su replicación por toda la infraestructura [57]. Las CDNs constan de una combinación de:

- *Content-delivery*: Consiste en una serie de servidores perimetrales (también llamados *surrogates*) que sirven copias del contenido al usuario final.

- *Request-routing infrastructure*: Es responsable de direccionar la petición del cliente al *surrogate* apropiado. También, actualiza el contenido de las cachés.

- *Distribution infrastructure*: Mueve el contenido del servidor origen a los servidores CDN y se encarga de asegurar la consistencia del contenido en dichas cachés.

- *Accounting infrastructure*: Mantiene *logs* de los acceso de los clientes y almacena el uso que hace cada uno.

Las CDNs son servicios que suelen ofrecer los proveedores de CDNs, como Akamai o Cloudfront (comentado en la sección 2.2.1). Estos proveedores se encargan de instalar multitud de servidores CDN por Internet, bien en grandes *datacenters*, o bien cerca del usuario, de forma que con la replicación de contenido se consigue una mejor distribución de la información a la que se quiere acceder. De este modo, se consiguen una serie de ventajas [57]:

- Evitar congestiones y altas latencias. Por la utilización de servidores más cercanos al usuario, disminuyendo el tráfico de la red troncal.

- Fiabilidad. Debido al alto grado de redundancia y a la posibilidad de paliar el efecto de ataques DoS.

- Buena escalabilidad cuando es desplegado correctamente.

Por qué usar una CDN

La importancia de las CDNs se puede representar con algunos datos:

- El incremento de uso de las CDNs. En 2014 más del 39 por ciento del tráfico de Internet era soportado por CDNs. Para 2019 se estima que será más del 60 por ciento del tráfico.

- Se estima que, en 2019, el total de tráfico IP mensual que se alcanzará será de 139 exabytes. Más del triple que en 2014 [69].

- El tráfico de las CDNs también se va a incrementar en proporción al tráfico total. Como se puede ver en la Figura 1.6.

- La tendencia a usar *Big Data* [50], como nueva forma para administrar y tomar decisiones en las empresas. Estudiar el tráfico y tomar decisiones en base a las evidencias que este aporta, hace necesario gestionar eficientemente estos recursos.

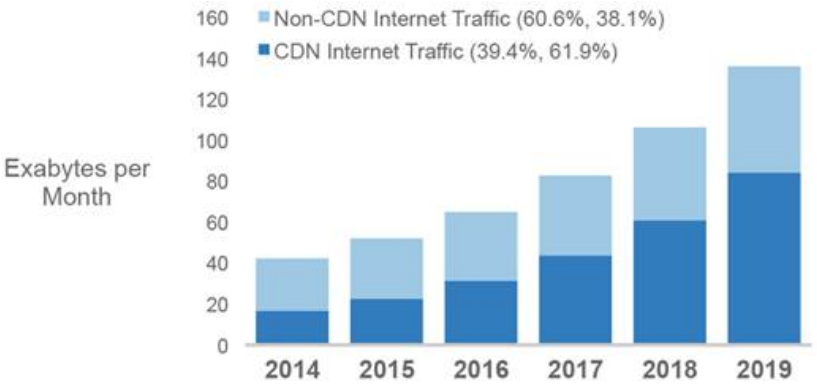


Figura 1.6: Tráfico de CDNs de 2014 a 2019. [69]

- Actualmente la interacción entre CDN e ISP no está completamente desarrollada, hay información "oculta" que impide un despliegue más eficientemente. La razón es que ambas partes deben revelar secretos de información, algo que puede ser crítico en sus planes de negocio. El ISP debe mostrar detalles de la topología y el estado de la CDN, pudiendo exponer debilidades de la infraestructura. Por otra parte, el proveedor CDN debe revelar donde están sus *surrogates* y sus políticas de asignación. Con SDN se puede llevar a cabo una mejor ocultación de este tipo de información [74].

1.1.3. Motivación empresarial

Uno de los principales incentivos en la utilización de NFV/SDN es la búsqueda de una disminución de costes respecto a la tecnología tradicional [45].

Tanto NFV como SDN suponen un cambio significativo, especialmente en el ciclo de innovación que deben aportar los proveedores, además de simplificar la red y adaptarla para tener una mayor eficiencia en el uso de recursos. De modo que repercutirá de forma positiva aumentando los beneficios y reduciendo el coste total de propiedad o *Total Cost of Ownership* (TCO). El TCO consiste en:

- *Capital expense* (CAPEX): la adquisición de nuevos activos.
- *Operating expense* (OPEX): crear, hacer funcionar y distribuir servicios a los usuarios.

Por tanto, la utilización de NFV/SDN en una red hace atractiva su implantación en un medio-largo plazo, reduciendo la complejidad y coste de

Categoría OPEX	**Área**	**Impacto esperado**
Operaciones de red/IT	Despliegue y mejora	50-90 % ahorro
	Administración de capacidad	60-80 % ahorro
	Garantía (NOC remoto)	Hasta 5-25 % ahorro
	Garantía (reparación en campo)	80-95 % ahorro
Operaciones de red	Transporte	10-25 % ahorro
	Administración de proveedores	Hasta 40 % incremento
Op. medioambientales	Potencia	20-70 % ahorro
	Inmuebles	Hasta 30-50 % ahorro
Operaciones de servicios	Garantía	10-20 % ahorro
	Servicio *Onboarding*	5-10 % ahorro

Tabla 1.1: Resumen de las mejoras esperadas en OPEX. Basado en el modelo de Laboratorios Bell.

los equipos de los clientes. Cuando se alcance un servicio totalmente NFV/SDN los costes internos de la estructura caerán significativamente. Los Laboratorios Bell han modelado[1] diversos casos de estudio SDN/NFV como puntos de referencia para la repercusión en el OPEX [45]. La tabla 1.1 recoge un resumen de las estimaciones obtenidas con dicho modelo.

A continuación se exponen más detalladamente dichos conceptos:

- Despliegue y la mejora: Ya no depende de una entidad, al no ser un sistema tan cerrado, se convierte en un entorno más dinámico.

- Administración de capacidad: Monitorizar el uso, predecir y planear el despliegue de capacidades adicionales depende actualmente de los vendedores, con los cuales se dimensiona el producto. Con NFV/SDN la administración de esta capacidad se realizará de forma más ágil gestionándolo por los VNF y NFVI; componentes de la estructura *framework* NFV (sección 1.1.1).

- Costes y garantía: La virtualización reduce los costes, puesto que la uniformidad del *hardware* de la infraestructura ayuda a disminuir los costes de aprendizaje y conocimiento. Debido a la disminución de *hardware* es más fácil encontrar los fallos. En el caso de virtualizar funciones de red como los *Customer-Premises Equipment* (CPE), pueden reducirse las intervenciones de campo sustancialmente.

- Operaciones de transporte de red: Capacidad y aprovisionamiento de actividades para elementos de transporte de red como *routers*, dispo-

[1]El modelo utilizado son un conjunto de proyecciones de futuros estados, basados en los evaluación de la tecnología y una extrapolación desde los puntos de operación de datos actuales.

sitivos ópticos o nodos de acceso. Con NFV/SDN podemos habilitar dinámicamente, situar y conectar funciones en la red para modelar el tráfico y mejorar el rendimiento.

- Gestión con proveedores: La uniformidad e independencia del *hardware* y *software* simplificará las actividades de la cadena de suministros (como el seguimiento de pedidos o mantener contratos de *hardware* y *software*). Sin embargo, los contratos de gestión de proveedores incrementarán el OPEX inicial, debido al proceso de virtualización.

- Garantía de servicio: Se centra en la disponibilidad, fiabilidad y calidad del servicio, desde la perspectiva del cliente. Una mayor capacidad de análisis de red, automatización y uniformidad de *hardware*, debería resultar en una reducción del número de incidentes, minimizando el OPEX por servicio.

- Servicios *Onboarding*: Consiste en la integración del servicio con los sistemas asociados (tarificación, pedido, administración...) y los procesos que permiten ser ofrecido junto a un soporte. La adopción de APIs abiertas y estándares, facilitará una integración más ágil de los servicios en las plataformas de administración.

- Operaciones ambientales: El mantenimiento de los costes de energía e inmuebles son los principales costes de las "operaciones ambientales". Hay un ahorro en el número de equipos.

1.2. Alcance de la memoria

En este Trabajo de Fin de Máster (TFM) se va a llevar a cabo una prueba de concepto de distribución de contenido sobre una red que hace uso de la tecnología SDN. Dicha prueba de concepto consistirá en un escenario en el que se utilizará el emulador de redes Mininet para emular la topología SDN. Adicionalmente, se utilizarán equipos conectados a esta topología que contarán con el *software* ATS para realizar la función de cacheo, de forma que se simule el comportamiento de una CDN a pequeña escala. El objetivo de este escenario es mostrar la viabilidad y utilidad de utilizar la tecnología SDN para llevar a cabo distribución de contenido. Dicho sistema puede implementarse posteriormente a una escala mayor. El uso de la tecnología SDN beneficiará en gran medida a las empresas a reducir costes, tanto en el desarrollo de servicios y como en el mantenimiento de la red.

En la memoria se detallará conceptos básicos, la razón de utilizar un *software* u otro, una planificación de los costes del desarrollo del TFM y finalmente unas conclusiones sobre la utilización de dicha tecnología y del

proyecto llevado a cabo. La presente memoria se ha estructurado en los siguientes capítulos:

- **Capítulo 1. Introducción.** En este capítulo se hará una introducción de aspectos más importantes relacionados con la motivación de la realización del proyecto y el estudio de conceptos claves.
- **Capítulo 2. Estado del arte.** Se detallan proyectos e investigaciones relacionadas con el proyecto. Adicionalmente, se estudiarán herramientas tanto comerciales como académicas de CDN.
- **Capítulo 3. Planificación de costes.** En este capítulo se lleva a cabo una planificación de recursos y costes que conllevan la realización del proyecto. Además, se aportará un presupuesto con los costes totales.
- **Capítulo 4. Análisis de objetivos y metodología.** Se realiza un visionado general de los objetivos y el alcance del proyecto realizado. Se estudiar los requisitos necesarios y la metodología seguida. Finalmente se evaluará el cumplimiento de los objetivos propuestos.
- **Capítulo 5. Tecnologías y herramientas utilizadas.** En este capítulo se estudian a fondo los conceptos relacionados con la distribución de contenidos y con la tecnología CDN. Posteriormente se estudian elementos relacionados con la tecnología SDN, como su arquitectura, componentes y el protocolo OpenFlow. Finalmente se hace un estudio de la elección del *software*, como máquinas virtuales, programas o sistemas operativos. Y se detallará el uso del *software* más importante; Apache Traffic Server (ATS), OpenDayLight (ODL) y Mininet.
- **Capítulo 6. Desarrollo práctico del escenario SDN y el balanceador de carga. Resultados**. En este capítulo se lleva a cabo un estudio del escenario finalmente implementado. En él se describe tanto la implementación de ATS como de ODL y Mininet. Así como el *cluster* de caché (Traffic Server Cluster), y la consistencia de la información, es decir, el refresco de la caché. Finalmente se lleva a cabo una comprobación de los resultados, con una serie de casos de ejemplo en los que se estudian distintas configuraciones y las variables obtenidas del funcionamiento directo de la red.
- **Capítulo 7. Conclusiones y trabajos futuros**. Cumplimiento de objetivos tanto académicos como profesionales y líneas futuras.

Capítulo 2

Estado del arte

En el anterior capítulo se ha repasado la importancia que están adquiriendo las redes SDN, así como su proyección. Del mismo modo, se ha visto la gran importancia que actualmente tienen las CDN, tanto para los proveedores de servicios como para el usuario. Dado que las redes SDN son una tecnología aún en fase de desarrollo, no existen una gran cantidad de proyectos en los que se combinen el uso de SDN y la distribución de contenidos, sin embargo, se ha llevado a cabo recapitulación de información relacionada en el ámbito de investigación.

Adicionalmente, se estudiarán utilidades y herramientas actuales, utilizadas para la implementación de una CDN.

2.1. Proyectos e investigación

Existen una serie de proyectos donde se pueden ver puntos en común con la materia tratada en este trabajo.

En las referencias [47], [40] y [61] se propone la creación de un sistema de balanceo de carga *Round Robin* sobre una infraestructura de red SDN. En ellos hacen uso del controlador SDN POX, y de la herramienta de emulación de redes Mininet. Estos proyectos son similares a lo que se llevará a cabo en este proyecto, sin embargo, en el proyecto que nos concierne, se llevará a cabo además un cacheo de la información en diferentes equipos, haciendo uso del *proxy* de cacheo web Apache Traffic Server, y adicionalmente, el balanceo de carga que se llevará a cabo no será simplemente Round Robin, sino que se aprovechará la capacidad que tienen las redes SDN para elaborar uno que haga uso de las estadísticas aportadas por los dispositivos de red. Por último, el controlador que se usará será OpenDaylight, que es programado en Java, frente a POX (utilizado en las referencias) que hace uso de Python.

En la universidad de Seúl, estudian la posibilidad de distribuir contenido a través de diferentes proveedores de servicio (ISP) usando OpenFlow [28], (explicado OpenFlow en la Sección 5.2. En la memoria se propone un sistema donde las peticiones y paquetes de contenidos se reenvían mediante una dirección privada en cada ISP. El controlador de cada ISP intercambia información de señalización para llevar a cabo la entrega de contenido.

En el trabajo [67] desarrollado en la Universidad Aristóteles de Salónica y en la Universidad de Chipre, se lleva a cabo el desarrollo de una herramienta llamada *CDN utility*. Esta herramienta captura el tráfico aplicando diferentes parámetros en la red, para así estudiar la utilidad de cada servidor CDN (*surrogate*) desplegado.

Telefónica y NEC Corporation ganaron en 2014 el premio Telecoms.com al proyecto *best fixed network innovation* por su CPE virtualizado (vCPE), que ayudaba a reducir los costes de ancho de banda de acceso de la red mediante la virtualización de componentes. El proyecto de prueba se llevó a cabo en Brasil, donde Telefónica opera bajo la marca Vivo. Telefónica y NEC fueron las primeras compañías en el mundo en comenzar la virtualización de *consumers' broadband gateways* en octubre de 2013. Actualmente proveen conectividad básica, actuando como punto de acceso, *switch* y módem. El enrutamiento, configuración de *Dynamic Host Configuration Protocol* (DHCP) y las funcionalidades de *Network Address Translation* (NAT) se proveen remotamente a través de la nube, corriendo funciones de red virtualizadas (VNF) en servidores *commercial off-the-shelf* (COTS). Haciendo uso de su vCPE, al tratarse de una infraestructura enteramente virtual se llevó a cabo un ahorro considerable en la contaminación, dada la disminución del número de traslados en vehículo necesarios para solventar problemas en el lugar producido, puesto que podía hacerse a distancia. La virtualización además de aportar más agilidad para solventar problemas, también permite añadir nuevos servicios rápidamente [21].

Gurbani, Scharf, Lashman y Hilt, en [42], comentan la posibilidad de usar un protocolo que provea a las aplicaciones una capa de abstracción con una vista global de la red, sin exponer detalles o políticas de los proveedores. Para ello describen que el protocolo *Application Layer Traffic Optimization* (ALTO), aporta una limpia, madura y estandarizada abstracción que puede ser usada por redes SDN para obtener información de red. Las CDNs son un ejemplo de tecnología que une redes con la funcionalidad de aplicaciones como pretende SDN. La abstracción de ALTO en SDN puede ser usada por una CDN para mejorar la selección del servidor de caché, o del servidor origen. Además, ALTO ayuda en el proceso de actualizar la caché de los servidores. Todo ello partiendo de la idea de que las peticiones de *routing* de CDN puedan hacer uso de la interfaz *Northbound*, para obtener la topología, y su información sobre distancias o tipo de contenido, para así distribuir

mejor el contenido.

En el *paper* de Wichtlhuber, Reinecke y Haysheer presentan un nuevo enfoque promoviendo la colaboración entre el ISP y el proveedor de CDN, con un despliegue sobre *switches* SDN en una red ISP [74]. En este caso el desarrollo de aplicaciones se realiza sobre el controlador RYU y Mininet.

2.2. Herramientas y proveedores de CDNs

En esta sección se identifican los principales proveedores de tecnologías CDN, y, además, se incluyen las principales herramientas de ámbito académico que actualmente pueden ser utilizadas para llevar a cabo la implantación y el estudio de redes que hagan uso de tecnología CDN.

2.2.1. CDNs comerciales

Akamai es la CDN dominante en el mercado. Con una tasa aproximada del 53 por ciento en el mercado en 2014 y con unos ingresos anuales de unos 2.200 millones de dólares (2015). Tiene una amplia cartera de clientes como Adobe, IBM o Fiat. Concretamente 1 de cada 3 empresas de la lista Global 500 ®[2]. Akamai es responsable de servir entre el 15 y 30 por ciento del tráfico de todo el contenido web [20]. Cuenta con más de 216 mil servidores en más de 120 países.

Existen otros sistemas de CDN con gran presencia en el mercado, se tratan de Level 3, CloudFront de Amazon, edgeCast y Limelight Networks. La suma de los cuatro ronda unos ingresos aproximados de 800-1.000 millones de dólares (2014) [18]

- CloudFront: Es la apuesta de Amazon en servicios CDN. Permite compaginar el uso haciendo uso de un servidor de origen y otros Servicios Amazon Web (AWS) como S3 o EC2, que son de almacenamiento y computo en la nube, respectivamente. Una de sus ventajas es el pago por uso de GB, y no en contratos o mensualmente [3].

- Level 3 Communications: Es una gran compañía, con años de experiencia y posee una red Tier 1 (cobertura internacional) y sobre 5.6Tbps de capacidad de *peering* [9].

- EdgeCast: La CDN actualmente es propiedad de Verizon Communications. EdgeCast cuenta con clientes importantes como Wordpress o Twitter. EdgeCast localiza puntos estratégicos para colocar sus servidores y construir puntos de presencia (PoP) [4].

- Limelight Networks: Provee vídeo, música, juegos y descargas bajo demanda o en vivo. Cuenta con más de 18 mil servidores, con presencia en todos los continentes. Tiene clientes reconocidos como Carrefour o MTV [10].

También existen otras CDNs importantes, pero con una menor presencia en el mercado como CDNetworks, ChinaCache, ChinaNetCenter, entre otras [18].

2.2.2. CDNs Académicas

Las CDNs son sistemas en los que se invierte mucho dinero y constituyen una fuente de rentabilidad para numerosas empresas, por lo que las CDNs comerciales tienen una mayor presencia y soporte que las CDNs académicas. No obstante, existen algunos proyectos académicos aún en uso [27]:

- CoDeeN: Es una CDN académica para bancos de pruebas, construida sobre PlanetLab (en glosarioB) por el grupo Network Systems Group, de la Universidad de Princeton. Consiste en una red de servidores de alto rendimiento. Sobre CoDeeN hay numerosos proyectos como CoBlitz, que es un servicio de distribución de archivos de gran tamaño, o CoDNS que es un servicio de *lookup* DNS [5].
- CoralCDN: Es una CDN basada en tecnología peer-to-peer. Opera sobre PlanetLab en la fase de *testbed* [6].
- Globule (globule.org): Es una CDN colaborativa Open Source. Provee replicación de contenido, monitorización de servidores y redirección de peticiones [8].
- COMODIN: Es una CDN de *streaming*, desplegada sobre una *testbed* [7].

Capítulo 3

Planificación y estimación de costes

En este capítulo se abordan los aspectos relacionados con la planificación y el coste asociado a la implementación del proyecto, así como la elaboración de la documentación.

En primer lugar, se llevará a cabo un estudio de la planificación a realizar, para tener una referencia temporal del trabajo realizado.

En segundo lugar, se estudiarán los recursos necesarios, tanto materiales como humanos.

Finalmente, se realizará una estimación de los costes asociados a todo el proyecto en función de los recursos necesitados y del tiempo empleado.

3.1. Planificación

En esta sección se va a llevar a cabo de forma resumida una planificación del proceso de desarrollo del proyecto y su memoria. Para ello se establecerán una serie de tiempos en los que se llevará a cabo.

Para representar de forma gráfica esta planificación se adjunta un diagrama de Gantt que condensa los periodos de tiempo que conforman la planificación de TFM, tal y como vemos en la Figura 3.1.

3.1.1. Revisión bibliográfica

Estudio del estado del arte y estudio teórico. Por un lado, se adquirirán los conocimientos necesarios relacionados tanto con las redes *software* como con las redes de distribución de contenidos. La obtención de una bue-

Figura 3.1: Diagrama de Gantt de la planificación temporal del proyecto.

na bibliografía es fundamental para tener una base para poder seleccionar correctamente las características que tendrá el escenario creado, así como el *software* utilizado que nos permitirá trabajar con él. Adicionalmente se llevará a cabo un visionado de los proyectos e investigaciones que tengan características similares a las de nuestro proyecto. Finalmente se estudiarán proveedores comerciales que actualmente ofrecen servicios de CDN y *software* académico que nos permite trabajar con esta tecnología.

3.1.2. Planteamiento del diseño

Se estudiarán las diferentes alternativas sobre el diseño y el funcionamiento de los posibles escenarios creados. En función de las posibilidades y de las posibilidades brindadas por el diferentes *software*, se elegirá un diseño u otro. Finalmente veremos las posibles limitaciones y los puntos fuertes del escenario planteado, para finalmente empezar a trabajar en él.

3.1.3. Programación de equipos

Instalación y configuración de máquinas virtuales, interfaces de red y *software* necesario. Puesta a punto de los archivos de configuración necesarios para el desarrollo de las aplicaciones principales; ATS, Mininet y ODL.

Desarrollo en Apache Traffic Server

Para llevar a cabo el desarrollo en Apache Traffic Server es necesario conocer todos sus módulos, y saber qué tipo de funcionalidades se pueden utilizar. Tras aprender a realizar las configuraciones básicas de sus módulos y archivos de configuración, se podrá llevar a cabo la implantación de las características que se quieren llevar a cabo. Se creará un *reverse proxy*,

de forma que traducirá direcciones *Universal Resource Locator* (URL) y cacheará información en sus servidores.

Desarrollo en Mininet

Para utilizar Mininet es necesario habituarse con la programación en Python y conocer sus funcionalidades más sencillas. La página de SDN Hub [15] cuenta con un tutorial con las características esenciales de Mininet, que ayudarán a implementar el escenario de red SDN. También será necesario saber cómo funcionan las tablas de grupo contenidas en el *switch* virtual de la topología. Dicho *switch* es compatible con el protocolo estándar *de facto* de SDN; Openflow.

Desarrollo en OpenDaylight

OpenDaylight es un controlador SDN bastante completo, pero aún en maduración, por lo que familiarizarse con todos los módulos implicados en el desarrollo de una aplicación puede ser tedioso, dada la gran cantidad de información que en algunos casos queda desactualizada por la continua evolución de ODL. En él se implementará la lógica del balanceador de carga, mediante el uso del módulo Restconf se llevará a cabo la modificación de tablas de grupo contenidas en el *switch*.

Fase de pruebas

Una vez montado el sistema se llevará a cabo una serie de escenarios para comprobar el funcionamiento del sistema implementado, y se comprobará que todo funciona según lo estipulado. Finalmente, haciendo uso de peticiones Restconf, desde el controlador se tomarán datos de las variables del *switch*, para comprobar que el comportamiento del escenario, y de la lógica implementada.

Memoria técnica

En último lugar, la memoria contará con el conjunto de conocimientos tanto prácticos como teóricos vistos a lo largo del desarrollo del proyecto. Partiendo de las referencias bibliográficas, hasta el desarrollo a fondo de todo el escenario implementado con tecnología SDN.

3.2. Recursos

3.2.1. Humanos

- D. Juan Manuel López Soler y D. Jorge Navarro Ortiz, profesores de la Universidad de Granada en el Departamento de Teoría de Señal, Telemática y Comunicaciones, en calidad de tutores del proyecto.
- Roberto Jiménez Sánchez, alumno del Máster Universitario de Ingeniería de Telecomunicación de la Universidad de Granada, autor del proyecto.

3.2.2. Hardware

- Ordenador portátil Asus GL552VW-DM141 con procesador Intel Core i7-6700HQ, memoria RAM de 8GB y disco duro de 1TB.

3.2.3. Software

- Sistema Operativo Windows 10. Usado en el equipo principal.
- VirtualBox. En dicho entorno se ejecutará los servidores Apache Traffic Server y la red SDN.
- Sistema Operativo Ubuntu 12.04 LTS. Utilizado en las máquinas VirtualBox.
- OpenDaylight. Controlador SDN.
- Open vSwitch 2.3.0.
- Mininet.
- Wireshark 1.12.1 (con soporte para analizar paquetes OpenFlow)
- JDK 1.8
- Eclipse Luna (programación Java)
- Maven 3.3.3
- Procesador de texto *online* Overleaf. Con él se editará la documentación del proyecto.

Fase	**Tiempo**
Revisión bibliográfica	60 horas
Planteamiento del diseño	5 horas
Programación de equipos	20 horas
Programación de ATS	60 horas
Programación de Mininet	40 horas
Programación de ODL	60 horas
Fase de pruebas	20 horas
Memoria técnica	110 horas
Trabajo de tutorías	25 horas

Tabla 3.1: Estimación de tiempos de recursos humanos.

3.3. Estimación de costes

Humanos

- Las horas de trabajo autónomo para un máster en ingeniería de telecomunicación se situará en 25 euros / hora.
- El sueldo para los tutores se fijará en 50 euros / hora.

3.3.1. Hardware y software

Recurso	Vida media	Coste unitario
Ordenador portátil	48 meses	850 euros
Sistema operativo	-	0 euros
Entorno de desarrollo	-	0 euros
Software adicional	-	0 euros

Tabla 3.2: Costes de recursos *hardware* y *software* del proyecto.

3.4. Presupuesto

Concepto	Coste
1 desarrollador x 400 horas x 25 euros/hora	10000.00 euros
2 tutores x 25 horas x 50 euros/hora	1250 euros
Hardware (Portátil x 850 euros x 10 meses / 48 meses)	177.08 euros
Recursos *software*	0 euros
Total	11250 euros

Tabla 3.3: Presupuesto total del proyecto.

Capítulo 4

Análisis de objetivos y metodología

En este capítulo se tratan algunas cuestiones importantes antes de comenzar el desarrollo del proyecto.

En primer lugar, se analizarán los objetivos principales en el planteamiento del desarrollo del proyecto, que se ajuste a los requisitos temporales que equivalen a los 12 créditos ECTS pertinentes.

En segundo lugar, en base a los objetivos previamente planteados, se estudiarán los requisitos necesarios para llevar a cabo la implementación práctica del proyecto.

Se analizará la planificación fijada para la realización del proyecto, y cómo han sido llevadas a cabo en cada una de sus fases.

Por último, se estudiará si se han cumplido los objetivos pertinentes una vez finalizado el trabajo, en base a los inicialmente planteados.

4.1. Objetivos y alcance

El principal objetivo de este proyecto es implementar un sistema de distribución de contenido o CDN sobre una red SDN. Para ello se utilizará el *software* Apache Traffic Server, que hará la función de cacheo para poder implementar la CDN, y para la parte de SDN se utilizará una máquina virtual del foro SDN llamado SDN Hub [15]. Dicha máquina virtual contiene el *software* necesario para la implementación y estudio de la red SDN (como OpenDaylight, Mininet o Wireshark). Haciendo uso del controlador SDN, OpendayLight, se llevará a cabo un balanceador de carga que repartirá el tráfico entre las máquinas con el *software* ATS.

La elaboración de un escenario simple que haga uso de estos componentes citados, nos ayudará a estudiar de una manera básica el comportamiento de los protocolos implicados.

En última instancia se llevará a cabo un estudio de diferentes escenarios, esto nos permitirá ver las posibilidades que nos brinda la tecnología SDN y la versatilidad que nos aporta tener una visión global de la red desde el controlador.

4.2. Requisitos

4.2.1. Requisitos funcionales

Los requisitos funcionales son aquellos que definen el comportamiento de un sistema. Es decir, define la función de un sistema o sus componentes.

- **Interconexión de equipos**: En primer lugar, es necesario tener un diseño de red para interconectar los equipos, haciendo uso de máquinas virtuales, es necesario configurar correctamente las interfaces.
- **Cachear y reenviar**: Es necesario configurar los equipos con el *software* ATS para cachear y redirigir contenido del servidor cuando se realice la petición. Y configurar la página web en el servidor que enviará la información a los equipos con ATS.
- **Escenario SDN**: La implementación debe llevarse a cabo en un escenario SDN. Por lo que es necesario programar en el emulador de red Mininet (Python) una red a la que conectar los equipos anteriormente descrito.
- **Balanceo de carga**: Se necesita programar el controlador OpenDayLight. De esta forma, sobre el diseño SDN anteriormente implementado, se podrá gestionar el reenvío de paquetes y añadir las funcionalidades de balanceo de carga.

4.2.2. Requisitos no funcionales

Los requisitos no funcionales en vez de estudiar el comportamiento, se centra en las características del funcionamiento.

- Compatibilidad: El *software* en términos de compatibilidad está correctamente contemplado. Por un lado, ATS es compatible con Linux, en nuestro caso es implementado en Ubuntu. Por otro lado, Mininet está basado en Python que es uno de los lenguajes ampliamente

soportado por los sistemas SDN [73]. Por último, el controlador OpenDayLight es uno de los más utilizados y es programado en Java.

- Costes y licencias: En términos de *software* el proyecto no tiene coste alguno. Opendaylight es Open Source bajo la licencia Eclipse Public License (EPL) 1.0. Mininet: también es Open Source bajo una licencia BSD. Finalmente ATS utiliza la licencia Apache license v2.

- Escalabilidad: el proyecto parte de un escenario sencillo, sobre todo debido a la falta de recursos cuando son implementados en un mismo equipo mediante máquinas virtuales. Sin embargo, en equipos separados y utilizando equipos específicos SDN (no emulados) el proyecto puede ser implementado en una red de mayor tamaño.

- Documentación: el *software* usado cuenta con una gran cantidad de documentación, y una comunidad dinámica. Sin embargo, las redes SDN, al tratarse de tecnología relativamente novedosa, está aún en desarrollo, y hay bastante información desactualizada (de versiones anteriores).

4.3. Metodología

Para el cumplimiento de los objetivos la planificación llevada a cabo será:

1. Tras formalizar la adjudicación del proyecto, se llevará a cabo el estudio teórico de las tecnologías predominantes implicadas en el proyecto, concretamente sobre el uso de SDN y de las CDNs.

2. Posteriormente se realizarán ejemplos sencillos para familiarizarse con las herramientas que serán utilizadas en el proyecto.

3. Se configurarán las máquinas virtuales haciendo uso de VirtualBox, para crear una topología sencilla, sin hacer uso de SDN.

4. Sobre la topología anterior se implementará ATS. Para usarlo correctamente es necesario estudiar su funcionamiento, Apache tiene bastante documentación en su página y con él se llevará a cabo el redireccionamiento y cacheo de contenido, simulando de forma mínima el comportamiento de una CDN.

5. Una vez realizada una topología "tradicional", se llevará a cabo un escenario de mínimos, sobre SDN, haciendo uso de Mininet. Se estudiará el comportamiento de la red SDN, haciendo uso de herramientas como Wireshark, de forma que ayude a paliar futuros errores.

6. Se llevará a cabo el uso del controlador OpenDayLight. Para ello será necesario aprender los conceptos básicos de utilización. El entorno de OpenDayLight es programado con Java, de forma que hace falta saber manipular la gestión y construcción de proyectos, mediante el *software* Maven. Tras tener los conceptos claros, se llevará a cabo la instalación de módulos fundamentales para llevar a cabo la comunicación entre equipos (llamado *learning switch*), para así poder establecer un "ping" entre ellos.

7. Tras estudiar de forma práctica el funcionamiento de los protocolos implicados en SDN, así como del uso de las herramientas relacionadas, se añadirán equipos que hacen uso de ATS. De esta forma se creará la topología final, que implicará el uso de SDN y ATS.

8. Una vez montada la topología se añadirá la funcionalidad de balanceo de carga. Para ello, se hará uso del controlador OpenDayLight. Mediante peticiones al *switch* haciendo uso de *Representational State Transfer* (REST), se obtendrá información del tráfico enviado, y en función de éste se priorizará el tráfico entre un equipo u otro.

9. Finalmente se llevará a cabo la implementación de varios escenarios. Con ellos se estudiarán diferentes posibilidades de configuración del módulo de balanceo de carga, en función de las necesidades de la red. Adicionalmente, se estudiará un conjunto de parámetros que permitirán comprender mejor el funcionamiento de la red y del balanceador de carga.

Capítulo 5

Tecnologías y Herramientas

En este capítulo se estudiarán las tecnologías involucradas para llevar a cabo la distribución de contenidos en una red SDN. Así, se podrá entender de forma más clara el diseño llevado a cabo, y el funcionamiento de la red implementada.

Por un lado, se verán conceptos relacionados con las CDN, para entender mejor su funcionamiento. Por otro lado, se estudiarán fundamentos relacionados con las redes SDN, y la razón de su utilización.

Finalmente se estudiará el *software* utilizado en la realización del proyecto. Por un lado, veremos sus características y su funcionamiento, y además, se justificará su utilización frente a otras alternativas.

5.1. Distribución de contenidos

Como se ha visto en la introducción, actualmente, hasta el 50 por ciento del tráfico en Internet es servido por CDN, y, además, con previsión de seguir en aumento [69]. Para entender la necesidad de utilizar este tipo de arquitectura es importante conocer la evolución que han tenido, así como los problemas y las soluciones que han ido surgiendo, en la distribución de contenidos.

5.1.1. Modelos de distribución de contenidos

Cliente-Servidor

El modelo de servicio tradicional es el del paradigma cliente - servidor. En este escenario tenemos un servidor y un cliente conectados punto a punto.

Este tipo de modelo lleva consigo una serie de problemas [52]:

Figura 5.1: Modelo cliente servidor.

- Por un lado no puede escalarse correctamente, pues el servidor puede verse saturado por el número de peticiones.
- Los usuarios pueden estar lejos del servidor, por lo que los retardos pueden ser muy palpables.
- La existencia de un SPOF (*single point of failure*) o punto único de fallo, da lugar a que si el servidor tiene un fallo, el sistema global fallaría, pues los clientes no podrían acceder a la información.
- El efecto *flash crowd*. Se produce cuando existe un aumento brusco en el número de clientes que intentan acceder de forma simultánea al servidor web.

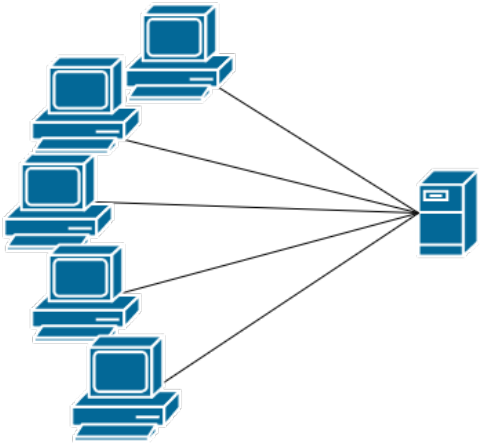


Figura 5.2: Modelo cliente servidor, puede conllevar problemas de escalado o SPOF.

Por ello que surge la necesidad de distribuir el contenido y así paliar:

- Conexiones de red y servidores saturados. Son evitados con la repartición del contenido.
- Retardos originados por la distancia de los servidores. Son evitados con el acercamiento del contenido.

Para paliar estos problemas surgen alternativas de distribución de contenidos que consisten en llevar la duplicación del contenido a un gran número de servidores, y posteriormente proporcionar a los clientes un medio para elegir los servidores que les entregan el contenido más velozmente. Las posibles soluciones son utilizar una granja de servidores, usar un modelo cliente-caché-servidor, el paradigma *peer-to-peer*, y, por último, una red de distribución de contenidos (CDN).

5.1.2. Web caching

El modelo cliente-caché-servidor es utilizado para responder a los clientes sin necesidad de preguntar de nuevo al servidor, sino directamente haciendo uso de la caché. De forma que las peticiones son enviadas primero a la caché [46]. Su funcionamiento es:

1. Las peticiones son enviadas a la caché. Posteriormente, se comprueba si el objeto solicitado está en la caché. Existen otros criterios a tener en cuenta, como que esté actualizado el contenido.
2. Si el objeto está en la caché, se devuelve al cliente.
3. Si el objeto no está en la caché, la caché lo solicita al servidor original. Finalmente se devuelve el contenido al cliente.

Como vemos, el contenido se actualiza bajo demanda, por cada petición que se recibe de los usuarios.

Hay diferentes tipos de modelos, según donde se sitúe el *proxy*. Independientemente del tipo de *proxy* que sea utilizado, ambos duplican el contenido de forma reactiva y reducen los costes en el servidor origen.

Proxy delantero

También llamado *forward proxy*. El *proxy* es situado cerca del usuario, reduciendo los retardos. Además, se disminuye el coste en el ISP (proveedor de servicios de Internet), haciendo que curse menos tráfico a la red.

Proxy trasero

También llamado *reverse proxy*. Dicho *proxy* se sitúa más cerca del servidor. De esta forma el tráfico se puede balancear según el servidor con el que vaya a ser servido. Sin embargo, no reduce tan sustancialmente los retardos, pues el contenido se sirve desde una mayor distancia que con un *proxy* delantero.

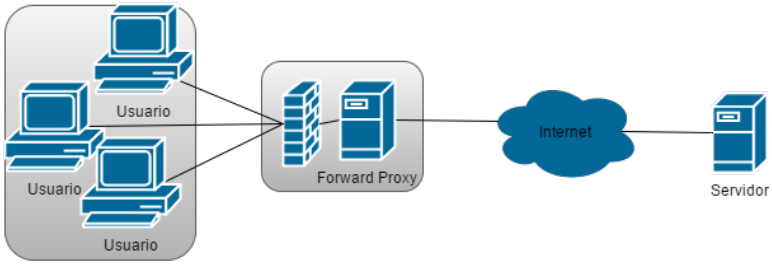


Figura 5.3: *Forward proxy.*

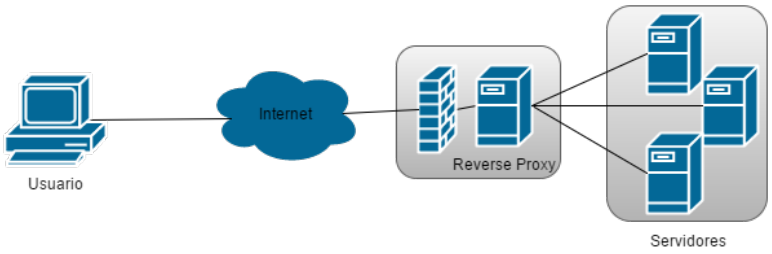


Figura 5.4: *Reverse proxy.*

Existe también otros tipos de mecanismos, como el *proxy* de interceptación (en un punto intermedio entre el servidor y el cliente), el cacheo cooperativo o las granjas de caché.

5.1.3. Peer to peer

Las redes *Peer to Peer* (P2P) no utilizan el paradigma cliente-servidor, sino que los pares (*peers*) actúan tanto como de clientes como de servidores. Las redes P2P son sistemas distribuidos en los cuales hay un intercambio directo de servicios o datos entre ordenadores. Implícitamente, en dicha definición, se parte de la base que los pares deben ser iguales [46].

En estas redes son los propios clientes, los que se encuentran en los bordes de Internet, los que llevan a cabo la compartición de recursos. Además de compartición de contenidos, se puede llevar a cabo compartición de CPU, ancho de banda o almacenamiento, entre otros parámetros.

P2P permite tener una alta escalabilidad dado que se aleja del paradigma cliente-servidor, y nos aporta una aproximación descentralizada. Una red P2P pura (o descentralizada) puede funcionar sin ningún sistema centralizado de control, o una organización jerárquica. Sin embargo, existen sistemas híbridos que hacen uso o de algún elemento centralizado, o estructuras jerárquicas.

A continuación, se verá una comparativa con las redes convencionales.

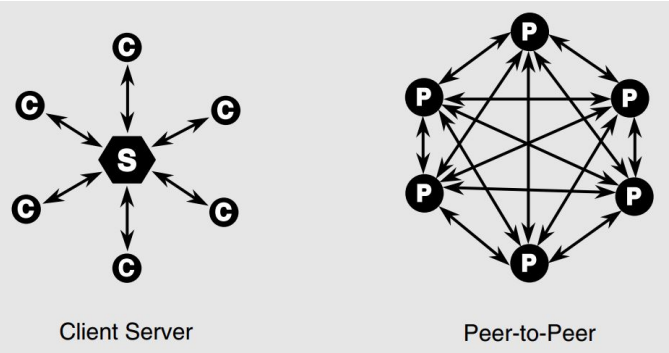


Figura 5.5: Modelo cliente servidor (izq.). Modelo P2P (der.). [46]

Redes convencionales	Redes P2P
Se basan en infraestructuras específicas, ofrecen servicios de transporte.	Estructuras superpuestas. Enfocado en el almacenamiento y distribución del contenido.
Se trata de una arquitectura centralizada. El contenido es almacenado y proporcionado por un servidor, o varios.	Se trata de una arquitectura descentralizada, donde cada uno de los pares almacena la información y realiza la búsqueda de información entre el resto de pares.
Arquitectura estática. Pueden introducir nuevos servidores, depende del proveedor.	Continuamente cambia. Pares que entran y salen de la red.

Estructura P2P

Las estructuras de P2P pueden ser de distintos tipos:

- Sin estructura: Centralizadas, puras e híbridas.
- Estructuradas: tabla de Hash distribuida.

P2P centralizadas

Las redes P2P centralizadas consisten en un servidor al cual el cliente le pide el objeto deseado (Figura 5.6). Dicho servidor devuelve una lista con los pares que tienen dicho objeto. En base a esta lista, y basándose en el tiempo de descarga estimado, el cliente selecciona el otro par del que quiere recibir el objeto.

Tiene como principal problema que al tratarse de una estructura parcialmente descentralizada (tráfico descentralizado, pero gestión centralizada), el

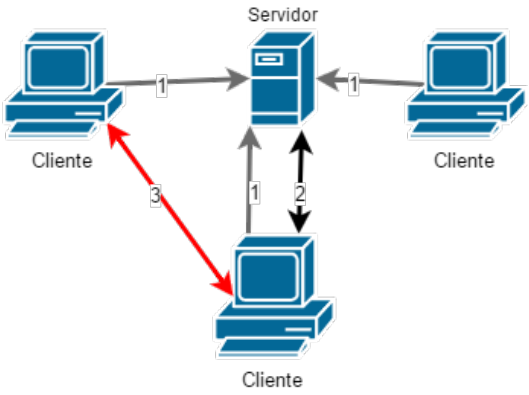


Figura 5.6: Modelo P2P centralizado. (1) publicación de la IP y datos compartidos, (2) petición del objeto, (3) selección del par.

directorio central hace de SPOF, por lo que en caso de fallo puede caer toda la red. Otro problema son los cuellos de botella por la cantidad de peticiones recibidas en el servidor central.

P2P puras

Las redes descentralizadas (o puras) no cuentan con una entidad central, sino que la localización del contenido se lleva a cabo mediante los propios pares. De esta forma, no existe un único punto de fallo, ni se generan cuellos de botella con tanta facilidad. Como contrapartida, uno de sus principales problemas es la generación masiva de peticiones, debido al descubrimiento de nuevos pares y a la búsqueda de contenidos por la red. Para el descubrimiento de nuevos pares, en la red P2P de Gnutella, se envían peticiones (*ping*) que son respondidas por los pares activos (*pong*), como vemos en la Figura 5.7.

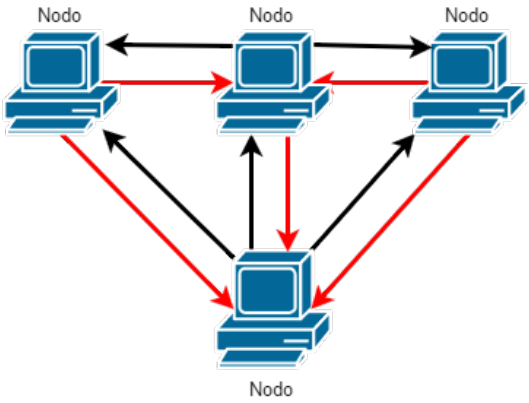


Figura 5.7: Modelo p2p descentralizado. En color negro ping. En color rojo pong.

Como alternativa para paliar los problemas de eficiencia de las redes descentralizadas, surgen las P2P híbridas. Haciendo uso de unos "supernodos", que son los que constituyen una red troncal en la red P2P para gestionar mejor el acceso al contenido ofrecido por nodos ordinarios.

Redes P2P con estructura

Una red P2P con estructura se puede conseguir haciendo uso de Tablas de Hash Distribuidas (*Distributed Hash Tables* (DHT)). Con este tipo de redes la localización del contenido se distribuye por el conjunto de nodos, siguiendo unos patrones concretos. De esta forma se puede evitar el uso de recursos debido a la inundación de mensajes en las P2P puras, y también se evitan los problemas de escalado en las P2P centralizadas.

Cada nodo gestiona una serie de referencias (el nombre del nodo y la clave del contenido) a otros nodos, siguiendo una distribución logarítmica:

$$O(log_2(N))$$

Donde N es el número de nodos. De forma que puede ser escalado con facilidad. Y se obtiene una mejor respuesta de búsqueda. En caso de haber un cambio en el conjunto de nodos de la red, se lleva a cabo una interrupción muy pequeña debido a la actualización las tablas de mapeo de los nodos. Se pueden manejar grandes volúmenes de entradas y salidas en la red.

5.1.4. Content Delivery Network

Las redes de distribución de contenido, CDN, surgieron a finales de los años 90 con el objetivo de disminuir los tiempos de espera en los usuarios, y, por lo tanto, reducir la latencia de red. La forma en la que se lleva a cabo es haciendo uso de múltiples servidores, distribuyéndolos en la cercanía de los clientes. De esta forma, se intenta reducir dicha latencia intentando evitar rutas más congestionadas. Estos servidores son denominados *surrogates* o réplicas. Actualmente, algunas CDNs como Akamai, son usadas por multitud de sitios web dada su capacidad de balancear la carga y reducir la latencia [52].

Frente al cacheo web, en las CDNs son los proveedores de servicios CDN los encargados de actualizar el contenido en los diferentes servidores (proactivo). Su alto grado de redundancia, los hace más fiables y permite mitigar ataques de DoS, evitar congestiones al usar los *surrogates* más cercanos o con menor congestión y evitar altas latencias dado que el tráfico queda en los bordes de internet, reduciendo la carga de la red troncal.

5.1.5. Selección de las réplicas y distribución del contenido

En primer lugar, vamos a ver un ejemplo de cómo se lleva a cabo la redirección de una página, en el proceso de elegir la réplica. Posteriormente veremos más a fondo el funcionamiento de la replicación del contenido.

Resolución DNS

El procedimiento es el siguiente [29]:

1. En primer lugar se realiza una petición a la página web a la que se quiere acceder. El registro A (*Address*) indica la dirección IP a la que se debe traducir.
2. Mediante el registro NS (*Name Server*) se pregunta a los servidores *Domain Name System* (DNS) por dicho dominio. En este caso al DNS local (primario/secundario).
3. En caso de no encontrarse en el servidor local, realiza una petición al servidor raíz (*root*) de Internet. El servidor DNS raíz aporta datos del servidor raíz del *Top Level Domain* (TLD).
4. El servidor TLD hace referencia a dominios como ".com", ".gov" o ".net". Una vez recibida la respuesta de los servidores que controlan la zona TLD, se obtiene la dirección del dominio, como por ejemplo microsoft.com.
5. Del servidor "autoritativo" (*authorithative*) se obtiene el nombre canónico (CNAME). En este caso hace referencia a un dominio equivalente al de Microsoft, alojado en uno de los servidores de Akamai.
6. Se vuelve a repetir el mismo procedimiento anteriormente llevado a cabo, pero en este caso para localizar el dominio Akamai.
7. Finalmente se recibe una respuesta del servidor Akamai que ha sido elegido para servir la información a nuestro equipo (en base a unos criterios fijados por Akamai).

Una de las cuestiones fundamentales para los proveedores de CDNs es, dónde poner el sustituto. El objetivo fundamental es aportar la mayor calidad de servicio posible a los clientes, a la par que se intenta minimizar la inversión. Para ello las estrategias llevadas a cabo son el de balancear la carga entre *surrogates*, analizar el tráfico del acceso a la red para situar en la mejor localización los sustitutos, y tener conexión directa al mayor número de redes ISPs posible para minimizar el tráfico que cursan hacia éstas.

Por otro lado, está la elección de qué servidor responde una petición, como se ha visto en el caso anterior con la página de Microsoft. Los criterios de selección pueden ser los siguientes:

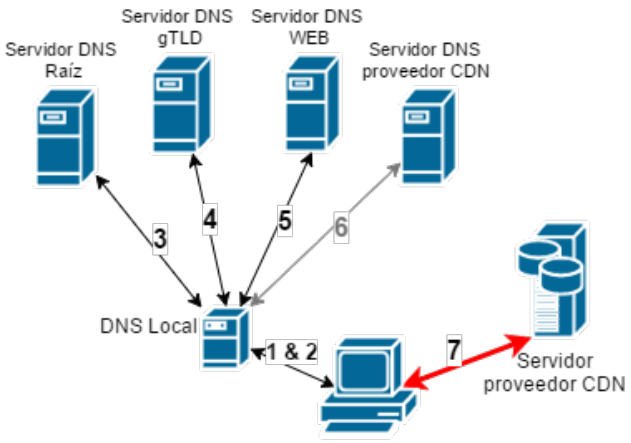


Figura 5.8: Resolución de DNS de una página web con contenido en una CDN.

- *Round-Trip delay Time* (RTT). Es el tiempo que tarda un paquete en ir desde un emisor, y en volver a dicho emisor, pasando por el destino (un equipo receptor)
- La carga y disponibilidad en el servidor sustituto
- La distancia física a dicho sustituto
- El rendimiento de la red para llegar al sustituto
- Aspectos económicos
- El tipo de objeto requerido. Puede haber servidores especializados en tipos de datos concretos.
- Selección anteriormente calculada mediante la tabla de encaminamiento

Otro aspecto importante es el reenvío de la petición al servidor sustituto. Normalmente, cada proveedor CDN tiene su algoritmo para llevar a cabo la elección del sustituto. Dicha elección es importante, pues determina en gran parte la calidad del servicio al usuario final. Las técnicas llevadas a cabo son las siguientes:

- Modificación del contenido: En el "index.html" descargado directamente del servidor primario, se tiene la dirección del contenido que se quiere almacenar en la CDN, apuntando directamente hacia ella, por tanto, el cliente simplemente debe resolver el direccionamiento a la CDN.

- Redirección HTTP: Hay un control más exacto de la *Internet Protocol* (IP) que se selecciona en función del cliente. Sin embargo, necesita un mayor número de conexiones, pues es necesario conectar con un intermediario para redirigir al servidor indicado.

- DNS: Se lleva a cabo una consulta en el DNS local, por lo que no se tiene en cuenta directamente al cliente, por lo que puede haber carga escondida al no saber cuántos usuarios hay detrás de una petición. Sin embargo, evita retrasos para establecer la conexión *Transmission Control Protocol* (TCP).

- *Anycast*: Directamente puede hacer la petición al servidor más cercano. Haciendo uso de *anycast*, se elige un único receptor dentro de un grupo de potenciales receptores, todos identificados con la misma dirección de destino. El sistema de *routing* es el que decide qué nodo es usado para cada petición, basándose en el diseño de la topología, y desde dónde es originada la petición [24].

- *Front-end*: Con un *front-end* se puede llevar a cabo un balanceo de carga, sin embargo, existen problemas de escalabilidad si solo existe un *front-end*, dada la capacidad que debe soportar.

Distribución de contenido

Cada proveedor CDN tiene un sistema para llevar a cabo la distribución de contenidos por la red. Cada uno aporta una serie de ventajas y desventajas, dependiendo de la red que se gestione. Las principales características son las siguientes:

- Distribución bajo demanda: Si está basada directamente en peticiones, el contenido solamente se actualiza en los servidores sustitutos cuando el cliente hace una petición al contenido.

- Distribución proactiva: El contenido se actualiza automáticamente cuando se actualiza el contenido en el servidor original.

- Cooperación entre sustitutos: Se puede realizar cooperación entre sustitutos para llevar a cabo la actualización del contenido, y para comprobar la coherencia del contenido (en nuestro modelo existirá la cooperación entre equipos gracias al uso del *software* ATS). Como contrapartida puede aumentar el tráfico de la red considerablemente.

Distribución estática y dinámica

Normalmente una página web suele contar con un conjunto de contenidos tanto estáticos como dinámicos. Los contenidos estáticos pueden ser

imágenes, vídeos, hojas de estilo... Es decir, es contenido que no cambia muy frecuentemente, y generalmente cambia predeciblemente.

Sin embargo, existe contenido que es dinámico que no puede ser cacheado. Los objetos dinámicos son únicos cada vez y nunca son el mismo dos veces. Por ejemplo, las llamadas *Asynchronous JavaScript And XML* (AJAX) usualmente son dinámicas [26]. Contenidos personalizados como el *login* de usuarios o el de transacciones de tarjetas de crédito entran en esta categoría. Las CDNs, y en general cualquier tipo de caché, no llevan a cabo el cacheo de contenido dinámico porque nunca es el mismo dos veces. En su lugar, las CDNs simplemente se encargan de distribuir lo más rápidamente posible este tipo de contenido desde el origen lo más rápida y eficientemente posible. Por lo que usar las CDNs en este caso no tienen como función cachear este tipo de objetos, sino el de entregarlo en diferentes localizaciones lo más rápidamente posible. Dicha técnica se denomina *Dynamic Site Acceleration* (DSA).

El contenido *event-driven* (contenido impulsado por eventos) a menudo es confundido con el contenido dinámico [53]. Se trata de contenido estático durante un tiempo determinado, el problema es que no se sabe cada cuanto tiempo cambiará. A menudo se le considera contenido dinámico, pero en el sentido más estricto no lo es, pues a diferencia del contenido dinámico, éste sí es posible cachearlo. Aunque sea impredecible su comportamiento, las CDNs actuales cuentan con mecanismos para llevar a cabo su cacheo. Un ejemplo de esto son las páginas de tipo Wikipedia, marcadores deportivos, comentarios, etc.

Cuando se lleva a cabo la distribución del contenido es necesario asegurar la consistencia de los datos, es decir, en una CDN debe existir coherencia del contenido en toda la red. Para ello la forma de asegurarla son mediante notificaciones.

- Sincronización periódica: Pueden existir periodos de tiempo con una inconsistencia de datos, dado el periodo de tiempo que está sin sincronizar. Se reduce el tráfico en la actualización de contenidos.
- Sincronización con notificación de cambios: Al notificarse directamente los cambios se reduce el tiempo de inconsistencia de datos. Sin embargo, se genera un gran tráfico en redes que suelen cambiar su contenido, al tener que notificar individualmente cada cambio.

El tipo de intercambio de datos puede ser mediante las siguientes conexiones [30]:

- *Unicast*(unidireccionales): *Unicast* es el término utilizado para describir aquellas comunicaciones donde la información es enviada de un punto a otro caso, con un único emisor y receptor involucrados.

- *Multicast* [43]: Con la distribución *one-to-many* o *many-to-many* (uno-a-muchos o muchos a muchos). Son comunicaciones donde los datos son enviados desde uno o varios puntos a una serie de puntos destino. Sus paquetes IPv4 se pueden identificar en que utilizan la clase D (224.0.0.0 - 239.255.255.255).

- *Broadcast* (difusión): Son las comunicaciones *one-to-many* (uno a muchos) donde la información es enviada de un emisor a todos los receptores conectados en ese dominio de difusión.

5.2. Openflow

Tal y como la define la ONF, el protocolo OpenFlow es el primer estándar encargado de la comunicación de interfaces *southbound* entre el plano de control y de datos en una arquitectura SDN [16]. Por tanto, dicho protocolo es necesario para llevar a cabo la separación del plano de control y del plano de datos. OpenFlow permite acceder y manipular directamente equipos de la capa de datos como *switches* o *routers*. El *software* externo puede acceder a primitivas básicas especificadas por OpenFlow para así llevar a cabo la modificación de las tablas de flujo (explicadas en el Apartado 5.2.2), con las que se puede controlar el tráfico de la red. Según el RFC 6437 [59], un flujo (*flow*) es "una secuencia de paquetes enviados de una fuente a un/os receptor/es mediante *unicast*, *anycast* o *multicast*".

OpenFlow, en concreto, utiliza flujos para identificar el tráfico de red, basándose en reglas predefinidas que pueden ser programadas por la capa de control de la SDN. De esta forma los gestores de red pueden definir la forma en la que el tráfico se comportará, basándose en parámetros como los patrones de uso o los recursos disponibles. Aunque OpenFlow permite programar flujo a flujo, la arquitectura SDN cuenta con un gran nivel de granularidad, permitiendo programar a nivel de usuario, sesión o aplicación, todo en tiempo real.

Recapitulando, la estructura de OpenFlow (tal y como se aprecia en la Figura 5.9) quedaría de la siguiente forma [51]:

- Tenemos un controlador, que será el encargado de comunicarse con los *switches*.

- Tenemos el *hardware* (plano de datos), es decir, *switches* con la capacidad de usar el protocolo OpenFlow.

- La comunicación entre el controlador y los *switches* se lleva a cabo mediante un canal seguro, típicamente se usará SSL o TLS.

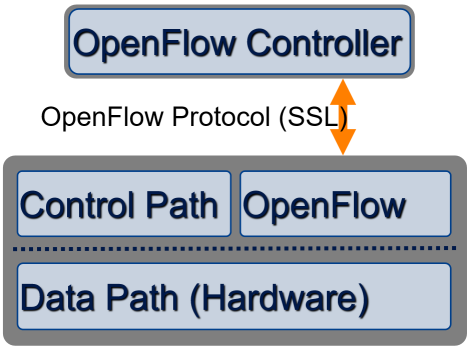


Figura 5.9: Ejemplo de comunicación segura en OpenFlow. [51]

5.2.1. Switch en OpenFlow

El objetivo principal del *switch* en OpenFlow es tomar los paquetes que llegan al puerto y reenviarlo por otro puerto, haciendo modificaciones en el paquete si fuera necesario [41]. Se pueden apreciar los caminos que pueden tomar los paquetes en la Figura 5.10. Primero se tiene un paquete que llega al puerto 2 (Caso X). Este paquete pasa por la función de *matching* (encuentra una coincidencia) de paquetes, y a continuación por la tabla de flujos. Tras este proceso el paquete es enviado a la casilla de acciones. Donde existen tres acciones que pueden ser tomadas:

- Caso A: Reenviar el paquete a un puerto de salida local, con posibilidad de modificar algunas cabeceras del paquete.
- Caso B: Descartar el paquete.
- Caso C: Enviar el paquete al controlador. El paquete es enviado mediante un canal seguro. Como respuesta, utilizando los mensajes "PACKET OUT" (Caso Y), el controlador puede reenviar el paquete bien hacia un puerto específico, o bien hacia la lógica de *matching*.

La introducción de colas en los puertos físicos de los *switches* facilita la implementación de algoritmos para dar *Quality of Service* (QoS). OpenFlow permite también mapear flujos a colas ya definidas. De forma que se tiene la capacidad de elegir qué tipo de cola queremos usar, dependiendo del flujo asociado a nuestro paquete.

5.2.2. Tabla de flujos

La tabla de flujos es un elemento fundamental en los *switches* OpenFlow. Una tabla de flujos consiste en una serie de "entradas de flujo" (*flow entries*).

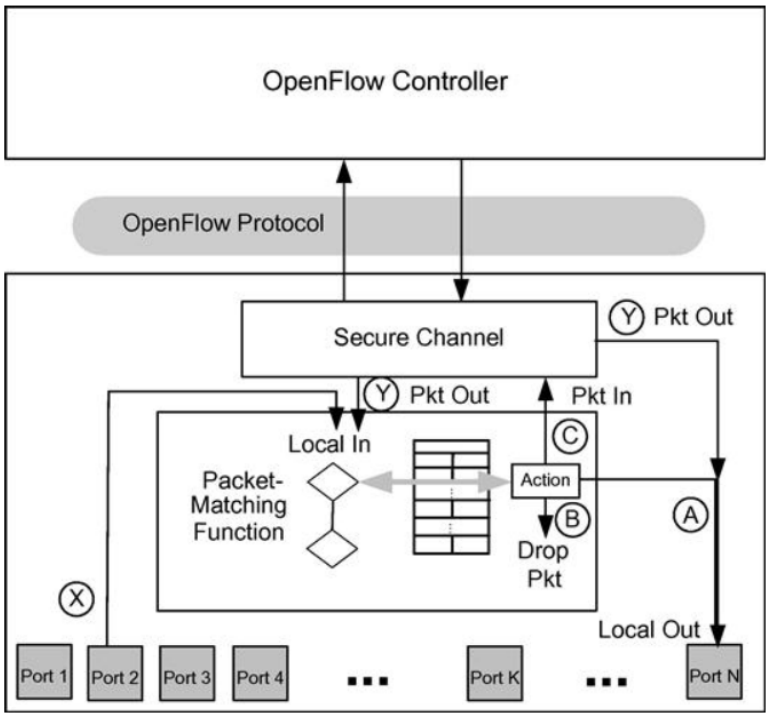


Figura 5.10: Ejemplo de tabla de flujo. [41]

Las entradas de flujo tienen un conjunto de campos con los cuales se puede llevar a cabo el *matching* de los paquetes que recibe, para posteriormente aplicar las instrucciones pertinentes. Según la especificación de OpenFlow [64], los principales campos son los siguientes:

- ***Match Fields***: Este campo es utilizado para comprobar la coincidencia de los paquetes. Consiste en el puerto de entrada y en las cabeceras de los paquetes. Opcionalmente puede contar con campos de tablas anteriores (cuando se realiza *pipeline*).
- ***Counters***: Contadores que se actualizan cuando se encuentra un paquete coincidente.
- ***Instructions***: Modifica la acción.

Sin embargo, el estándar ha ido evolucionando, y la versión actual [66], cuenta con los siguientes campos:

- ***Priority***: En caso de coincidencias en las acciones para un paquete, da prioridad.
- ***Timeouts***: Tiempo máximo en el que el flujo será borrado del *switch*.

- ***Cookie***: Dato asignado elegido por el controlador. Puede ser usado para filtrar flujos afectados por estadísticas o modificaciones.
- ***Flags***: Alteran la forma en la que los flujos de entrada son gestionados.

En las especificaciones más actuales se muestra la existencia de una tabla llamada *egress table*, sin embargo, al tratarse de un componente opcional, se va a obviar su explicación. Cada *switch* OpenFlow tiene múltiples tablas de flujo. El OpenFlow *pipeline* es el proceso que realizan los paquetes interactuando con esas tablas [64]. La Figura 5.11 muestra este proceso de *pipeline*.

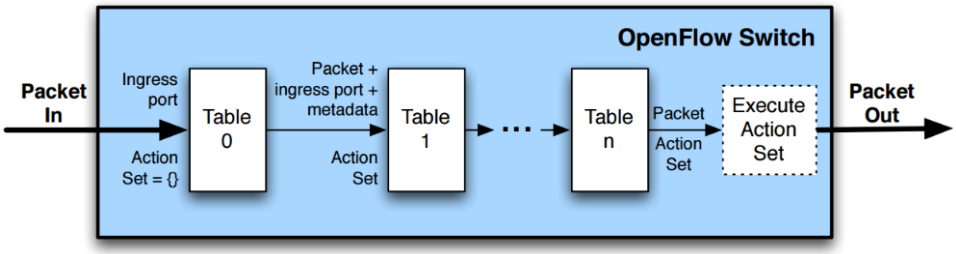


Figura 5.11: Paquetes procesados mediante multiples tablas (*pipeline*). [64]

Podemos comprobar que las tablas de flujos están numeradas en OpenFlow, empezando de 0 hasta n. Los paquetes cuando llegan al *switch* son comparados con la tabla 0, y, posteriormente, cada paquete puede usar o no las siguientes tablas, en función de las coincidencias que se den.

Podemos ver en la Figura 5.12 a nivel de tabla los campos que son comparados para comprobar si existe un *match* (coincidencia).

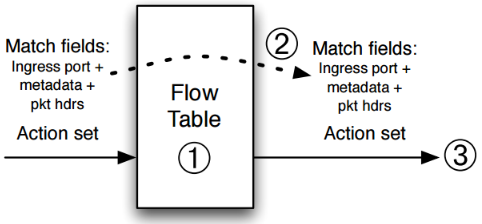


Figura 5.12: Paquetes procesados a nivel de tabla. [64]

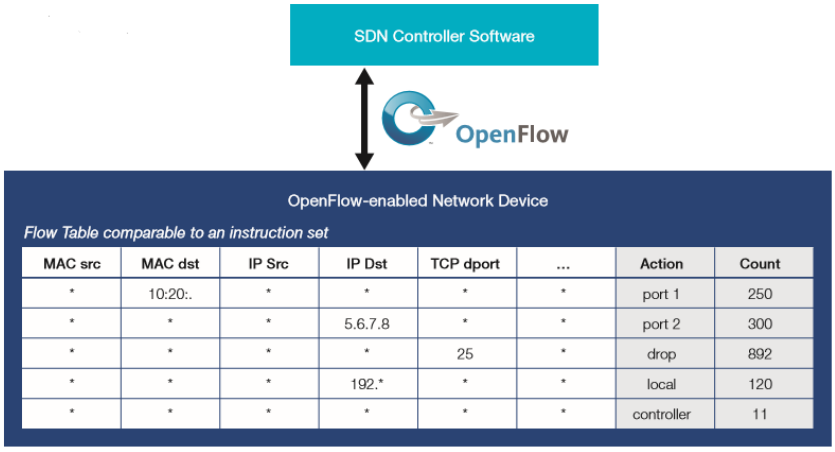
1. Cuando se recibe un paquete, el *switch* de OpenFlow lleva a cabo una búsqueda (*lookup*) en la primera tabla de flujo. En caso de existir coincidencias se aplica la de mayor prioridad (pueden existir varios *matching* coincidentes). En caso de no existir coincidencias, se produce

un *table miss*, por defecto el paquete es enviado al controlador (vía mensaje *packet in*), pero también podría ser descartado el paquete o ser procesado por la siguiente tabla (secuencialmente numerada).

2. En caso de existir coincidencias, se ejecutan las instrucciones pertinentes (actualizar las acciones, el paquete o los metadatos).

3. Si la tabla contiene una instrucción Goto, el paquete se envía a otra tabla de flujo. Si no cuenta con esa instrucción, el proceso de *pipeline* termina, por tanto, el paquete es procesado con sus acciones asociadas, y normalmente, reenviado.

En la Figura 5.13 tenemos un ejemplo de una tabla de flujo, con los campos de los paquetes entrantes que son usados para llevar a cabo el *match*. Aunque en la Figura no están detallados todos los parámetros con los que



MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figura 5.13: Ejemplo de tabla de flujo. [38]

puede compararse el tráfico entrante, existen además otros campos de Metadatos, EtherType, id de VLAN, etiqueta MPLS...

5.2.3. Tabla de grupos

Las tablas de grupo, contienen una serie de acciones aplicadas a un conjunto de flujos. De forma que, si quiere aplicarse acciones a una serie de flujos con unas características concretas, todos apuntarían a la misma tabla de grupo. Las tablas de grupo contienen una serie de parámetros tal y como vemos en el cuadro 5.1.

- **Identificador de grupo**: 32 bits que identifican al grupo.

Group Identifier	Group Type	Counters	Action Buckets

Tabla 5.1: Parámetros de la tabla de grupos. [64]

- **Tipo de grupo**: Puede ser de tipo *all*, *select*, *indirect* y *fast failover*. Descritos a continuación.
- **Contador**: El contador es actualizado cuando se procesan paquetes por el grupo.
- ***Buckets* de acciones**: Conjunto de acciones, cada *bucket* contiene una serie de acciones a ejecutar.

Según el tipo de tabla de grupo, OpenFlow permite usar diferentes métodos de reenvío:

- **All**: Ejecuta todos los *buckets* del grupo. Puede usarse para *broadcast* y *multicast*, puesto que lleva a cabo la clonación del paquete, por cada *bucket* enviado.
- **Select**: Ejecuta sólo un *bucket* del grupo, la forma de elegir el *bucket* por el que será enviado puede especificarse, una técnica simple sería mediante *round robin*. También pueden asignarse pesos (mayor peso es mayor prioridad).
- **Indirect**: Se utiliza para ejecutar un único *bucket* en el grupo. Es idéntico a usar un grupo *all* con un único *bucket*.
- **Fast failover**: Ejecuta el primer *bucket* vivo. Si un *bucket* no estuviera disponible, ejecutaría el siguiente.

5.2.4. Contadores

Existen una serie de contadores en OpenFlow, por cada tabla, flujo, puerto, cola, grupo, *bucket*, métrica y grupo de métrica [66].

En la tabla 5.2 están recogidos los contadores que existen por puerto. Se puede ver que solo algunos de estos contadores deben ser obligatoriamente implementados en el *switch*.

Estos contadores pueden dar información valiosa al controlador, para supervisar anomalías o estudiar el comportamiento del tráfico en la red en tiempo real.

Counter	Bits	
Per Port		
Received Packets	64	Required
Transmitted Packets	64	Required
Received Bytes	64	Optional
Transmitted Bytes	64	Optional
Receive Drops	64	Optional
Transmit Drops	64	Optional
Receive Errors	64	Optional
Transmit Errors	64	Optional
Receive Frame Alignment Errors	64	Optional
Receive Overrun Errors	64	Optional
Receive CRC Errors	64	Optional
Collisions	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional

Tabla 5.2: Lista de contadores por puerto. [66]

5.2.5. Tabla de métricas

Las tablas de métricas son una característica añadida en la especificación de OpenFlow en la versión 1.3 [65]. Esta tabla permite a OpenFlow implementar por flujo un limitador de tráfico. De la misma forma, aporta a OpenFlow la capacidad de llevar a cabo políticas QoS más complejas, como *Differentiated services Code Point* (DSCP), que permite clasificar paquetes por flujos en múltiples categorías basándose en su ratio de tráfico.

La métrica mide el ratio de los paquetes y permite controlar su tráfico, los componentes principales son los que se aprecian en la tabla 5.3:

Meter Identifier	Meter Bands	Counters

Tabla 5.3: Parámetros de la tabla de métricas. [65]

- **Meter identifier**: Son 32 bits que identifican inequívocamente la métrica.
- **Meter bands**: Son el conjunto de *meter bands*. Cada una especifica el ratio de la banda y la forma en la que se procesa el paquete.
- **Counters**: Contadores que se actualizan cuando un paquete es procesado por la métrica (*meter*).

5.3. Estudio técnico del *Software*.

Una vez vistos los conceptos básicos relacionados con las redes de distribución de contenidos y las redes definidas por *software*, se va a llevar a cabo una descripción más a fondo del *software* que será utilizado en el proyecto.

El *software* más importante utilizado en este proyecto son tres:

- Apache Traffic Server: Se usará como servidor *proxy* de cacheo.
- Mininet: Un emulador de red en el que se implementará la red SDN.
- OpenDaylight: Será el controlador de la red SDN.

5.3.1. Apache Traffic Server

Apache Traffic Server (ATS) es un *proxy* de cacheo que permite mejorar la eficiencia y el rendimiento de la red. ATS está diseñado para mejorar la distribución de contenidos para empresas, ISP, proveedores y grandes *intranets* [31]. Principalmente ATS suele usarse para implementar un cacheo de la información frecuentemente accedida, especialmente en los bordes de la red, de esta forma se consigue acercar el contenido a los usuarios, reduciendo el uso de ancho de banda total de la red y consiguiendo una distribución del contenido más rápida.

Cuando se decidió utilizar ATS se barajaron diferentes alternativas, sin embargo, los principales motivos por los que finalmente se decantó por este *software* son los siguientes:

- Es un proyecto del grupo Apache Software Foundation (ASF). ASF es una fundación que da soporte a multitud de proyectos *software Free and Open Source Software* (FOSS) (*software* libre y de código abierto). Entre sus principales patrocinadores se encuentran Google, Microsoft o Facebook [32].
- Buena documentación. Existe una gran cantidad de documentación es su página web, y se actualiza tras cada versión lanzada [34].
- ATS es utilizado por una gran cantidad de empresas por todo el mundo, desde pequeñas hasta grandes como Yahoo, Akamai o Comcast [31].
- Es un *software* muy completo, pero también permite realizar configuraciones sencillas con relativa facilidad, especialmente gracias al soporte de la comunidad y a su documentación.

Cuando se quiere llevar a cabo una CDN, hay múltiples opciones (incluidas *open source*) que permiten la distribución de contenido con rapidez y fiabilidad. Sin embargo, cada una tiene sus propias ventajas y desventajas.

- nginx (engine x) [54]: Es también un servidor que puede actuar como *reverse proxy*. Cuenta con importantes clientes como Netflix, Wordpress o Yandex. Los servidores de nginx son los segundos más usados (tras los de Apache), por lo que es un *proxy* bien asentado en el mercado. Cuenta con buena documentación y soporte comercial. Permite realizar balanceo de carga, cosa que también permite ATS (pero haciendo uso de un *plugin* experimental).
- Varnish [72]: Varnish es también un *reverse proxy*. En el caso de Varnish utiliza un *pool* de hebras para las conexiones (por cada conexión una hebra). Sin embargo, no es tan utilizado como ATS o nginx.

Archivos de configuración

Para llevar a cabo las configuraciones más básicas en ATS, se recurren a los siguientes archivos, ubicados en el directorio donde se lleva a cabo la instalación:

- records.config: Es el archivo de configuración. Contiene una lista de variables, usadas por ATS. En él hay variables de configuración relacionadas con los DNS, el control de la caché, el tipo de *proxy* usado o la configuración de un *cluster*, entre otras cosas. El archivo puede ser modificado directamente, pero para aplicar los cambios en el Traffic Server, debe reiniciarse, o bien ejecutar el comando `traffic_ctl config reload`.

  El formato que sigue cada variable es el siguiente:

  `SCOPE variable_name DATATYPE variable_value`

  - `SCOPE`: Define el nivel al que se aplica. Puede ser de tipo `CONFIG` aplicándose a todos los miembros del *cluster*. O `LOCAL`, lo cual se aplica solamente a la máquina local.
  - `DATATYPE`: Puede ser `FLOAT, INT o STRING` según el tipo de variables usadas.

- remap.config: En este archivo se encuentran las reglas de asignación (*mapping*) que Traffic Server usa para llevar a cabo:

  - Asignación de una URL (*Map URL*) a un servidor de origen, cuando ATS actúa como *reverse proxy*.

  - Redirección HTTP.
  - Asignación inversa (*Reverse-map*). Se lleva a cabo cuando el servidor origen responde a la petición y el cliente es dirigido a otra localización.

- cache.config: Define cómo ATS cachea los objetos web. En él se puede especificar:
  - No cachear objetos de una IP específica.
  - Cuánto tiempo mantener objetos en la caché.
  - Ignorar directivas de no-caché.
  - Cuánto tiempo considerar objetos como actualizados (*fresh*).
- plugin.config: Controla la ejecución y configuración de *plugins* disponibles para Traffic Server. Existen *plugins* útiles para este proyecto como *Stats Over HTTP* que permite enviar estadísticas por la red. También existe un balanceador de carga en fase experimental.

Configuración de un reverse proxy

ATS puede funcionar como *forward proxy* y como *reverse proxy* (términos vistos en el Apartado 5.1.2). Concretamente, en la implementación realizada, será necesario usarlo a modo de *reverse proxy*.

Un *reverse proxy* sencillo requiere cambios en los archivos de configuración que vienen por defecto. Para ello, en el archivo records.config se debe modificar los siguientes parámetros de la siguiente forma:

- `CONFIG proxy.config.http.cache.http INT 1`. Esta configuración permite actuar como *proxy* de las peticiones HTTP.
- `CONFIG proxy.config.reverse_proxy.enabled INT 1`. Habilita el *reverse proxy*.
- `CONFIG proxy.config.url_remap.remap_required INT 1`. Obliga a que una regla remap exista antes de funcionar como *proxy* ante una petición.
- `CONFIG proxy.config.url_remap.pristine_host_hdr INT 1`. Hace que Traffic Server mantenga la petición del cliente intacta en caso de que el servidor origen esté tomando acciones que dependan de dicha cabecera.
- `CONFIG proxy.config.http.server_ports STRING 8080`. Traffic Server es vinculado (*bind*) al puerto 8080.

Ya llevada a cabo la habilitación del servicio de *reverse proxy* se debe escribir las reglas de mapeo (*map*). Esto se lleva a cabo en el archivo *remap.config* descrito anteriormente. Se usará de ejemplo una página con el nombre "www.servidorats.com", como dirección del equipo ATS. Y en el dominio privado tendremos un equipo con la dirección 192.168.1.1. La petición para redirigir quedaría de la siguiente forma:

```
map http://www.servidorats.com 192.168.1.1
```

De esta forma, tenemos lo siguiente, tal y como vemos en la Figura 5.14:

1. "Usuario" realiza una petición a www.servidorats.com (el equipo ATS).
2. Son redirigidas al equipo 192.168.1.1 en caso de no estar en caché.
3. El equipo 192.168.1.1 envía la información al equipo ATS
4. Traffic Server puede guardar la información en caché (según nuestro criterio) y reenviar al usuario.

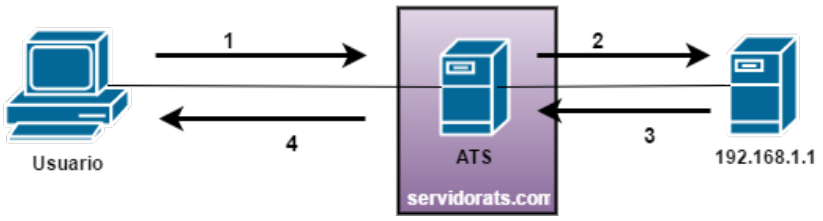


Figura 5.14: Funcionamiento de un *reverse proxy* haciendo uso de ATS.

En caso de llevarse a cabo otra petición a servidorats.com y éste tenga ya la información del equipo 192.168.1.1 guardada en caché, no es necesario ser actualizada, siempre bajo los criterios que determinemos. De esta forma:

1. "Usuario" realiza una petición a www.servidorats.com
2. Al tenerlo en caché ATS se contesta directamente al cliente, sin tener que contactar con el equipo 192.168.1.1

Por defecto, Traffic Server tiene una configuración de 256 MB para guardar información en la caché. Se puede aumentar ese valor en el archivo de configuración `storage.config`.

5.3.2. OpenDaylight

OpenDaylight es otro de los pilares fundamentales del proyecto. ODL actúa como controlador SDN en nuestro escenario. Desde su página afirman

que es considerado por muchos el estándar *de facto* en la industria. Cuentan con el apoyo de la Linux Foundation [37], con el objetivo de avanzar en el avance de tecnologías SDN y NFV colaborando en una plataforma común *open source* como es OpenDaylight. Otros miembros encargados de apoyar ODL son Cisco, Ericsson e Intel, entre otros. Por lo que OpenDaylight tiene un gran apoyo por parte de la industria [13].

Se ha optado por trabajar con OpenDaylight principalmente por el gran apoyo que tiene actualmente, además de ser el controlador usado en el departamento de Teoría de la Señal, Telemática y Comunicaciones (TSTC). Sin embargo, existen controladores que tienen un peso importante en las redes definidas por *software*. En el *paper* [68] se lleva a cabo una comparación entre los controladores: POX, Ryu, ONOS y OpenDaylight, haciendo uso del emulador de redes Mininet.

- NOX se trata del primer controlador SDN que soportó OpenFlow. Fue escrito en C++, y con el propósito de funcionar sobre Linux.
- POX es similar a NOX, solo que es desarrollado en Pyhon, de esta forma, aprender a programar en POX es más rápido y es la principal razón por la que es usado para investigación, demostraciones o experimentación, sin embargo, al ser programado en Python, el rendimiento es inferior al de controladores que usan otros lenguajes de programación como Java o C++.
- Ryu es un controlador SDN que también se programa en Python, cuenta con la ventaja sobre POX de que soporta una gran cantidad de versiones de OpenFlow, y otros protocolos de la interfaz *Southbound*. De la misma forma que POX, tiene un menor rendimiento comparado con otros controladores.
- *Open Network Operating System* (ONOS) [56] es un controlador escrito en Java, es más complejo de aprender que los basados en Python (como POX y Ryu). Tiene un buen soporte de OpenFlow y otros protocolos, como Netconf. Tiene un buen rendimiento.
- OpenDaylight como ya se ha comentado anteriormente tiene un gran soporte en la industria, además de ser robusto y tener un rendimiento elevado. Sin embargo, es complejo de usar y toma cierto tiempo aprender a desarrollar aplicaciones.
- Otro controlador a tener en cuenta es Floodlight. Éste cuenta con licencia Apache y se programa en Java.

La fundación OpenDaylight (ODL *foundation*) [13] fue fundada en 2013, promueve el desarrollo, distribución y adopción de ODL en un entorno neutral, para ayudar a la adopción de un ecosistema SDN abierto.

La comunidad de ODL trabaja en torno a ciclos de lanzamientos cada 6 meses. En el mes de septiembre de 2016 se lleva a cabo el lanzamiento oficial de la quinta versión de OpenDaylight, Boron. La versión finalmente utilizada en el proyecto es la versión Beryllium. A continuación, en el cuadro 5.4 se muestran los lanzamientos llevados a cabo.

Versión	Lanzamiento oficial
Hydrogen	Septiembre 2013
Helium	Septiembre 2014
Lithium	Junio 2015
Beryllium	Febrero 2016
Boron	Septiembre 2016
Carbon	No definido

Tabla 5.4: Versiones de OpenDaylight y sus fechas de salida. [36]

Estructura OpenDaylight

Para utilizar ODL es necesario tener unas nociones de su arquitectura. ODL está desarrollado en base a *Open Services Gateway Imitative* (OSGi), que es un *framework* Java para desarrollar *software* y librerías de forma modular. De forma que cada módulo se puede construir de forma independiente. Tal y como se muestra en la Figura 5.15, la arquitectura ODL está dividida en capas [44]:

1. Capa de aplicaciones (superior)
2. Capa de control (centro)
3. Capa de elementos de red (inferior). No forma parte de ODL, pero se necesita establecer conexión con esta capa.

El elemento más importante de ODL es la capa de control, se encuentra más detallado en la Figura 5.16 que contiene:

- Funciones básicas de la red como topologías, estadísticas, reenvío de servicios...
- Funciones que incluyen módulos para tareas específicas de *networking*.
- Funciona como capa de abstracción de servicios, entre la capa inferior y la superior. Además, lleva a cabo la comunicación entre ambas. De forma que ayuda a los desarrolladores a centrarse en el desarrollo de aplicaciones en vez de la comunicación entre capas.

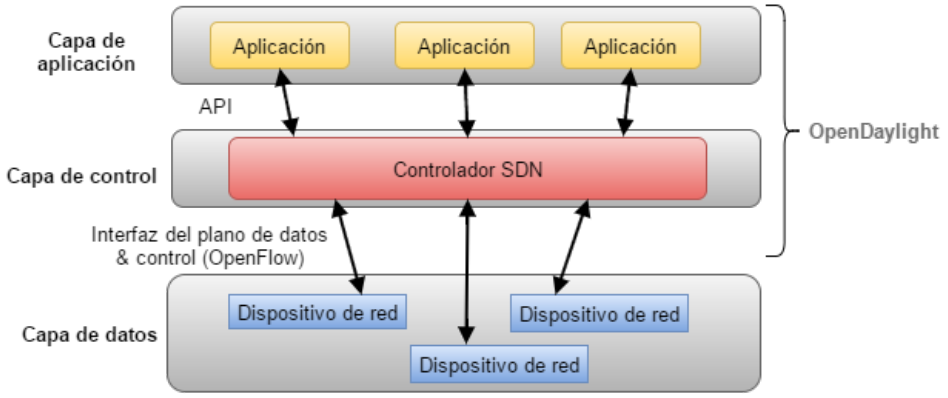


Figura 5.15: Estructura de *Software Defined Networking*.

En el *Southbound* puede soportar múltiples protocolos (y *plugins*) como OpenFlow, BGP, NETCONF... Estos módulos son vinculados dinámicamente a la capa de abstracción de servicios *Service Abstraction Layer* (SAL), en los cuales se exponen los servicios. SAL permite a ODL llevar a cabo el servicio, independientemente del protocolo subyacente usado entre el controlador y los dispositivos de red.

Las APIs del *Northbound* del controlador son usadas por las aplicaciones. ODL soporta el *framework* OSGi y REST bidireccional. El *framework* OSGi es usado por aplicaciones que correrán en el mismo espacio que el controlador, mientras que la API REST en aplicaciones que no corre en el mismo espacio de direcciones.

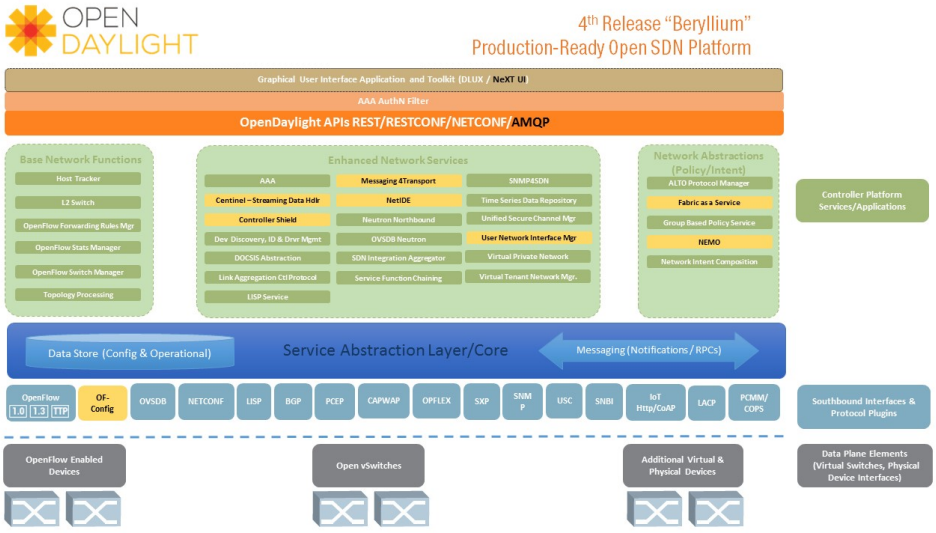


Figura 5.16: Arquitectura OpenDaylight (Beryllium).

Concretamente, las versiones actuales de ODL hacen uso de *Model-Driven* SAL (MD-SAL), se trata de un conjunto de guías para estructurar un modelo, con el objetivo de proveer soporte a las aplicaciones y *plugins* desarrollados [35]. MD-SAL viene a sustituir el modelo API-*Driven* SAL (AD-SAL), que actualmente se encuentra obsoleto en las versiones más actuales de ODL.

Hay una serie de conceptos importantes cuando se trabaja en el entorno de OpenDaylight, y que es importante conocer:

- El protocolo *Network Configuration Protocol* (NETCONF) es usado para gestionar y configurar elementos de red, así como mantener sus configuraciones en *data stores* y realizar operaciones de bajo nivel. NETCONF soporta *Remote Procedure Call* (RPC), notificaciones, comparar configuraciones...
- En ODL, YANG es un lenguaje de modelado de datos usado para desarrollar configuraciones, notificaciones y RPC para el protocolo NETCONF.
- RESTCONF es una interfaz de aplicación que permite acceder a operaciones sobre NETCONF haciendo uso de *Hypertext Transfer Protocol* (HTTP), que son definidas como RPC en *Yet Another Next Generation* (YANG). A los datos de configuración se puede acceder y modificar mediante la *Universal Resource Identifiers* (URI).

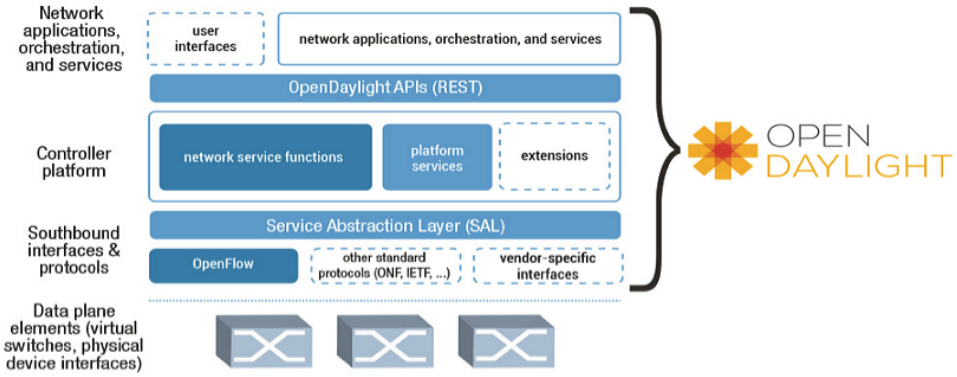


Figura 5.17: Estructura general de ODL.

Máquina usada con OpenDaylight

Para llevar a cabo la ejecución de OpenDaylight se hará uso de la máquina virtual de SDNHub. Esta máquina virtual contiene todos los programas

necesarios para la correcta emulación del escenario SDN (incluyendo el programa Mininet descrito en el Apartado 5.3.3). Centrándonos en los componentes usados por ODL en esta máquina, la página de SDNHub describe los siguientes [15]:

- Interfaces Java: Las interfaces de java son usadas en la escucha de eventos y especificaciones.
- Maven: facilita y automatiza la construcción. Hace uso de un pom.xml (*Project Object Model*) para la carga de dependencias entre paquetes.
- OSGi: Este *framework* es el *backend* de OpenDaylight. Permite cargar dinámicamente *bundles* y .jar para llevar a cabo el intercambio de información.
- Karaf: Es un pequeño *runtime* OSGi, que provee un contenedor para cargar diferentes módulos.

Para llevar a cabo la ejecución del controlador se debe inicializar el Karaf, con ello, todos los *bundles* de Java instalados en OSGi se ejecutan. Desde la terminal Karaf se manejan todas las aplicaciones y *bundles* (`feature:list` por ejemplo, muestra las funcionalidades que hay accesibles en el OSGi), a través del cual se pueden instalar o desinstalar funcionalidades, tal y como se aprecia en la Figura 5.18.

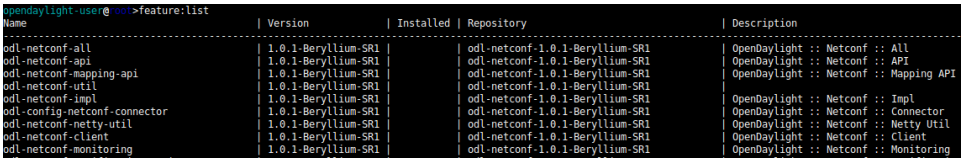


Figura 5.18: Visualización de algunos paquetes disponibles desde la terminal Karaf.

5.3.3. Mininet

Mininet, tal y como definen en su página oficial [11], es un emulador de redes capaz de crear redes virtuales de *hosts*, *switches*, controladores y enlaces. Los *hosts* virtuales de Mininet ejecutan un Linux estándar, y sus *switches* soportan OpenFlow, de manera que puede crearse una topología SDN con los equipos emulados en Mininet.

Además, Mininet permite experimentar con topologías y llevar a cabo *debugging*, trabajar simultáneamente varios desarrolladores en la misma topología, ejecutar aplicaciones OpenFlow sobre una red *testbed*... De forma

que todo el código escrito en Mininet, ya sea para el controlador OpenFlow como para *switches*, puede ser portado a un sistema real con pocos cambios.

Mininet es una utilidad muy útil, pero al ser ejecutado en un PC cuenta como principal limitación la CPU o el ancho de banda disponible en el *host* de alojamiento, aunque en un equipo potente puede llegar a arrancar hasta 4096 *hosts*.

Una de las principales razones por las que se ha utilizado Mininet es su gran cantidad de documentación, ya no solo en su web, sino en información académica como *papers*, además de contar con una buena comunidad con la que se pueden contrastar y revisar problemas con una mayor facilidad que otras alternativas.

La versión que usaremos de Mininet, al igual que ODL, es la que viene en la máquina virtual de SDNHub [15].

Comandos sencillos

Para familiarizarse con el uso de Mininet se verán unos comandos sencillos, Mininet cuenta con multitud de configuraciones, se mostrarán solo algunos ejemplos. Para crear una red sencilla, tal y como muestra la imagen 5.19 se puede hacer simplemente con una línea de comando:

```
sudo mn --switch ovs --controller ref --topo
tree,depth=3,fanout=2 --test pingall
```

Incluso con menos comandos podría generarse la topología, sin embargo, es conveniente conocer alguno de los utilizados:

- `mn`: Es el programa encargado de ejecutar Mininet
- `--switch ovs`: Especifica el tipo de *switch* utilizado. En este caso Open vSwitch (OVS), un *switch* multicapa [58] que soporta OpenFlow, y otros protocolos como Netflow, filtrado de tráfico, VLAN...
- `--controller ref`: con este comando se elige el controlador que queremos utilizar. En este caso es elegido el de referencia para Mininet.
- `--topo tree,depth=3,fanout=2`: Este comando específica una topología del tipo árbol, con una profundidad de 3, y con un total de 2 equipos conectados descendentemente a cada equipo.
- `--test pingall`: Este comando ejecuta un *ping* a todos los equipos, es útil puesto que podemos comprobar el alcance que tienen entre ellos antes de comenzar a operar con la topología.

En las topologías creadas por Mininet los *host* están nombrados como "hx", los *switch* como "sx" y los controladores como "cx". Desde cualquiera

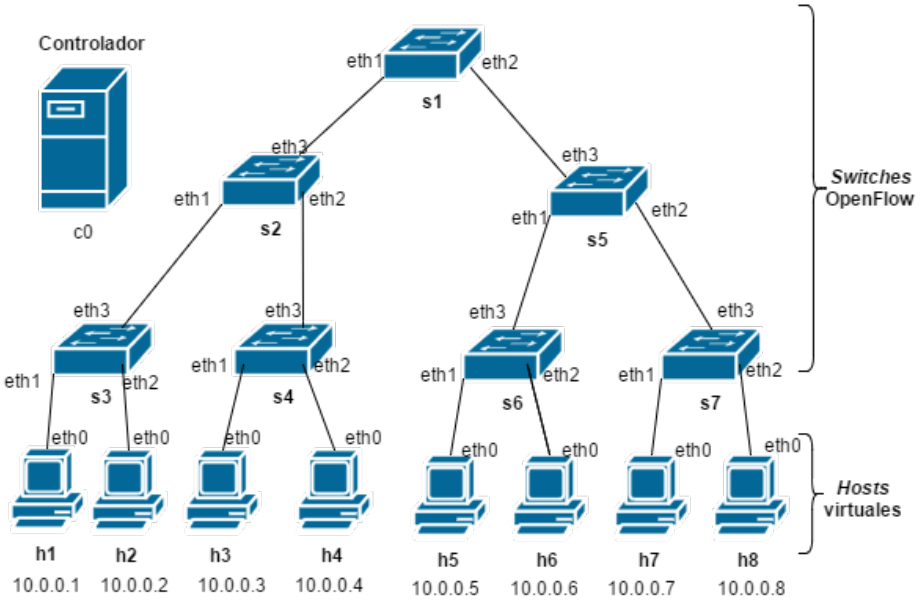


Figura 5.19: Topología creada por Mininet.

de estos equipos podemos realizar acciones, simplemente indicándolo desde la terminal de Mininet, indicando en primer lugar el equipo que queremos que ejecute la acción:

```
mininet>h1 ping h2
```

Con este comando, h1 realiza un *ping* a h2. Se puede realizar cualquier acción que nos permita ejecutar la máquina Linux que contiene cada equipo. La máquina que ejecuta Mininet (en nuestro caso la máquina virtual de SDNHub) comparte disco duro con los equipos de la topología de Mininet, por lo que, si tenemos un programa en la máquina anfitrión, estos serán accesibles desde por ejemplo `h1` en nuestra topología. Esto es muy útil, pues se podría ejecutar un servidor web, Apache Traffic Server o Wireshark, sobre los *hosts* de la topología.

Hay otra serie de comandos interesantes, como son:

- `mininet>help`: Muestra los comandos de la interfaz de línea de Mininet
- `mininet>nodes`: Muestra los nodos que se encuentra en la topología actual (controladores, *switches* y *host*).
- `mininet>xterm`: Abre una terminal del equipo que especifiquemos.
- `mininet>net`: Muestra información de los enlaces que existen en la topología actual.

- `mininet>dump`: Muestra información de todos los nodos de la topología, como podemos ver en la Figura 5.20

```
mininet> dump
<Host h1: h1-eth0:192.168.1.1 pid=10165>
<OVSSwitch s1: lo:127.0.0.1,eth1:None,eth2:None,eth3:None,s1-eth4:None pid=10150>
<RemoteController c0: 127.0.0.1:6633 pid=10143>
```

Figura 5.20: Muestra de los datos obtenidos con el comando *dump* en Mininet.

Si fuera necesaria la elaboración de una red más compleja, Mininet permite crear *scripts* para topologías de red. Para ello, la API de Mininet hace uso de Python.

5.3.4. Otro software

A lo largo de este proyecto también se han utilizado otras aplicaciones. Algunas de ellas son:

- VirtualBox para cargar todas las imágenes de máquinas virtuales utilizadas; la que contiene la infraestructura SDN con ODL y Mininet (obtenida de SDN Hub). Y otras 3 imágenes con equipos que formarán parte de la topología global.
- Sistemas operativos: Ubuntu 14.04 de 64 bits en la imagen de SDN Hub, y Ubuntu 12.04 de 64 bits en las otras 3 imágenes.
- Open vSwitch 2.3.90. Se trata de un *switch* virtual multicapa bajo la licencia *open source* de Apache 2.0. Actualmente forma parte de Linux Foundation (desde agosto de 2016) [58]. Soporta hasta la versión de OpenFlow 1.3, y las versiones 1.4 y 1.5, pero con ausencia de funciones [12].
- Wireshark 1.12.1 con soporte para analizar (*parse*) el contenido de OpenFlow. De esta forma se pueden estudiar las trazas de tráfico generadas en la topología.
- Eclipse Luna. Para llevar a cabo la parte de programación en Java del controlador ODL, y su gestión de eventos.
- Apache Maven 3.3.3. Es una herramienta usada para la gestión de proyectos, usado para gestionar la construcción del proyecto Java.

Capítulo 6

Desarrollo práctico del escenario SDN y el balanceador de carga. Resultados

En este capítulo se va a detallar todo el proceso relacionado con la creación del escenario SDN, con el objetivo de implementar un balanceador de carga.

- Primero, se llevará a cabo la creación de las máquinas virtuales. Una contendrá un servidor Apache, otras dos harán de caché con el programa Apache Traffic Server y en otra tendremos implementada la lógica SDN, con el programa Mininet y el controlador OpenDaylight.
- Posteriormente, se llevará a cabo una topología en Mininet a la que se conectarán el servidor, los equipos con ATS y un *host* virtual.
- Finalmente se aplicará la lógica a un módulo del controlador ODL, con el cual se llevará a cabo el balanceo de carga.

6.1. Escenario implementado

6.1.1. Creación de la topología

La topología que quiere llevarse a cabo está compuesta por un total de 4 máquinas virtuales en VirtualBox. Tal y como se muestra en la Figura 6.1, cada máquina virtual está representada con un color. Cada máquina virtual estará compuesta por:

- Máquina con Mininet y OpenDaylight (en color verde). Será la encargada de implementar la topología SDN y el controlador que implementará la aplicación de balanceo de carga.
- Dos máquinas con ATS. Serán las encargadas de realizar el almacenamiento temporal o cacheo del contenido. Al recibir las peticiones, en caso de no tener la información en caché, lo consultan al servidor web. Forman un *cluster* que mantiene actualizada la información entre ambas cachés.
- Máquina con servidor web. Contiene la información a la que se quiere acceder. Implementa Apache con una sencilla página web.

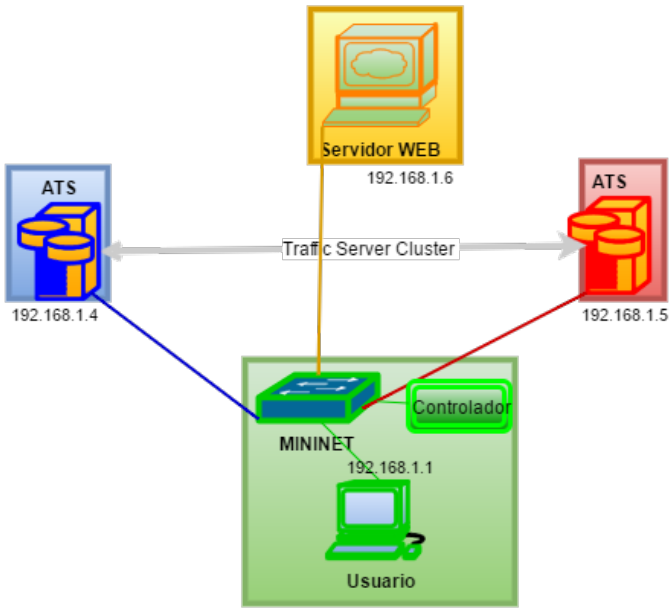


Figura 6.1: Escenario SDN a implementar.

Para la configuración de la topología, primero se comenzará con la configuración de las interfaces de red de las máquinas virtuales. Para la configuración de los equipos ATS y del servidor web, se realizará un proceso similar:

1. Habilitar en VirtualBox un adaptador de red de tipo “Red Interna”, el nombre de dicho adaptador debe ser el mismo que el del otro equipo al que estará directamente conectado.
2. Configurar la IP en el archivo “/etc/network/interfaces”, de los equipos. Asignando las IP 192.168.1.4 hasta 192.168.1.6 para los equipos con Apache Traffic Server y el servidor web.

Para llevar a cabo la configuración de la topología en la máquina virtual que contiene Mininet y OpenDaylight se llevará a cabo:

1. Habilitación en VirtualBox de tres adaptadores de red ("Red Interna") para cada uno de los equipos conectados, es decir, los dos equipos de cacheo (ATS) y el servidor.

2. Creación de una topología en Mininet (explicado en la Sección 6.1.3) con un *switch* Open vSwitch compatible con OpenFlow, y un equipo virtual (llamado "Usuario" en la imagen), con una IP 192.168.1.1.

3. A este *switch* se le conectarán los otros tres equipos de las máquinas virtuales. El *switch* Mininet además estará comunicado con el controlador OpenDaylight, que será el que se encargará de notificar sobre la actualización de las tablas de flujo que comunican los equipos entre sí en la topología SDN. Adicionalmente, el controlador se encargará de modificar la tabla de grupo para balancear la carga entre los dos equipos caché.

4. Para realizar correctamente la comunicación con los equipos externos a la topología SDN es necesario habilitar el modo promiscuo en todas las interfaces de red involucradas.

6.1.2. Configuración de Apache Traffic Server

Para llevar a cabo el cacheo de la información se instalará el *software* Apache Traffic Server en los equipos con IP 192.168.1.4 y 192.168.1.5.

Las instalación de ATS, en las versiones actuales de Ubuntu, puede llevarse a cabo directamente desde el repositorio con `apt-get install`. Sin embargo, dado que las máquinas sobre las que se han instalado son Ubuntu 12.04, ha tenido que hacerse manualmente, descargando los archivos directamente desde la página de Apache Traffic Server [33].

Los equipos con ATS instalado harán de *proxy* y cachearán la información. Para llevar a cabo este procedimiento será necesario modificar ciertos valores en el archivo de asignación de URL (*remap.config*). El objetivo es que las peticiones que vayan a los equipos 192.168.1.4 y 192.168.1.5 sean redirigidas al equipo 192.168.1.6, para ello en el equipo 192.168.1.4 se añadirá la siguiente línea en el archivo *remap.config*:

```
map 192.168.1.4:80 192.168.1.6:80
```

De la misma forma se hará en el equipo 192.168.1.5. El puerto 80 será donde se encuentre ejecutado el servidor Apache en el servidor.

Apache Traffic Server se ejecutará por defecto en el puerto 8080 en los equipos que harán de *proxy*, por lo que se usará una regla en el cortafuegos

Iptables para redirigir las peticiones del puerto 80 (HTTP) al 8080. ATS por defecto lleva a cabo el cacheo de cierta información, pero, Traffic Server permite modificar ciertos parámetros para cachear una mayor cantidad de información. Tal y como se aprecia en la Figura 6.2, cuando se hace una petición al equipo ATS con la dirección 192.168.1.4, y aún no tiene guardado en caché el contenido del servidor, es necesario primero enviar una petición al servidor de origen (192.168.1.6), para poder entregar la petición web al equipo 192.168.1.1.

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.1	192.168.1.4	TCP	74	57902 > http [SYN]
2	0.000053	192.168.1.4	192.168.1.1	TCP	74	http > 57902 [SYN,
3	0.039287	192.168.1.1	192.168.1.4	TCP	66	57902 > http [ACK]
4	0.043235	192.168.1.1	192.168.1.4	HTTP	141	GET / HTTP/1.1
5	0.043274	192.168.1.4	192.168.1.1	TCP	66	http > 57902 [ACK]
6	0.067115	192.168.1.4	192.168.1.6	TCP	74	33520 > http [SYN]
7	1.065585	192.168.1.4	192.168.1.6	TCP	74	33520 > http [SYN]

Figura 6.2: Trazas de Wireshark cuando el contenido no está en caché.

Sin embargo, al volver a realizar la petición, al encontrarse el contenido en caché, no es necesario que el equipo ATS realice la petición al servidor origen, de esta forma se reduce el número de peticiones, y en general, el tiempo total dedicado a obtener la información. Tal y como muestra la Figura 6.3.

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.1	192.168.1.4	TCP	74	57911 > http [SYN] Seq=
2	0.000035	192.168.1.4	192.168.1.1	TCP	74	http > 57911 [SYN, ACK]
3	0.058406	192.168.1.1	192.168.1.4	TCP	66	57911 > http [ACK] Seq=
4	0.064721	192.168.1.1	192.168.1.4	HTTP	141	GET / HTTP/1.1
5	0.064788	192.168.1.4	192.168.1.1	TCP	66	http > 57911 [ACK] Seq=

Figura 6.3: Trazas cuando el contenido se encuentra en caché.

Típicamente, en una red CDN, el equipo ATS se situaría relativamente cerca del cliente (con respecto al servidor origen), por lo que la reducción de tiempos sería sustancial.

Por último, haciendo uso de ATS se implementará un *cluster*, por el cual se realizará una comunicación continua entre los equipos Traffic Server para llevar a cabo la actualización de las cachés. Para implementar el *cluster* se debe indica en el archivo `records.config`. Las principales variables a definir para crear el *cluster* son el nombre del *cluster*, los puertos, la interfaz de red y dirección *multicast* de la que hará uso para compartir la información, en nuestro caso será la dirección *multicast* 224.0.1.37 (para más información ver el Apéndice A.4).

6.1.3. Configuración de Mininet

En la máquina virtual de SDN Hub, como ya se vio en la Sección 5.3.2, están instaladas todas las herramientas que permitirán implementar una red SDN. Entre ellos se encuentra el programa Mininet, con este emulador se creará una sencilla topología en Python (código completo en A.2), usando un *host* (con IP 192.168.1.1), un *switch* y un controlador. A esta topología se le conectarán los dos equipos con ATS.

Se comenzará añadiendo un controlador, llamado "c0" con OpenFlow 1.3.

```
c0 = net.addController( name='c0', controller=RemoteController,
protocols='OpenFlow13', ip='127.0.0.1', port=6633)
```

Posteriormente, se agregará un *switch* con la MAC '00:00:00:00:00:10':

```
s1 = net.addSwitch('s1', cls=OVSSwitch,
mac='00:00:00:00:00:10', protocols='OpenFlow13')
```

Este *switch* es creado con 3 interfaces para así poder conectar los dos equipos ATS y el servidor web. De forma similar a la siguiente:

```
intfName= 'eth1'
Intf( intfName, node=s1 )
```

Finalmente se añadirá el *host* con una MAC "00:00:00:00:00:01" y una IP "192.168.1.1". Éste actuará de cliente, para llevar a cabo las peticiones al servidor.

```
h1 = net.addHost('h1', mac='00:00:00:00:00:01',
ip='192.168.1.1/24', defaultRoute='via 192.168.1.10')
```

6.1.4. Configuración del controlador

Para realizar correctamente la comunicación entre los equipos de la topología es necesario configurar el controlador, de forma que éste pueda procesar las peticiones y actualizar las tablas de flujo, para que de esta forma el *switch* pueda enviar la información al destino correcto.

Para iniciar el controlador se ejecuta el *runtime* Karaf, obteniendo lo que se aprecia en la Figura 6.4.

Desde esta interfaz se puede acceder a los diferentes módulos disponibles en el controlador. De esta forma, si es necesario añadir alguna funcionalidad al controlador, se puede añadir instalando algún paquete en concreto.

En esta implementación, se necesita que el controlador sea capaz de modificar la tabla de flujo del *switch* para que realice correctamente el proceso de *routing* cuando se realiza una petición de un equipo a otro. Instalando el paquete de "learning switch", se consigue esta capacidad. Se habilitará me-

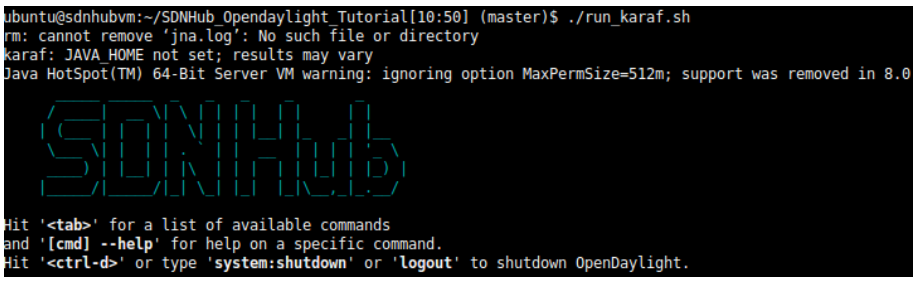


Figura 6.4: Interfaz de Karaf.

diante el comando `feature:install sdnhub-tutorial-learning-switch`.

6.2. Balanceador de carga

En este apartado, se va a implementar en el controlador la lógica relacionada con el balanceador de carga, para que las peticiones que reciba el *switch* sean repartidas entre los dos equipos ATS en base a los criterios que serán detallados posteriormente.

6.2.1. IP Virtual

Configuración en el *host*

Para el balanceador de carga también será necesario hacer uso de una IP y una *Media Access Control* (MAC) virtual, de forma que a esta dirección IP será a la que el cliente haga la petición HTTP. Posteriormente, el *switch* envía el paquete al controlador, y será éste el que decida a qué dirección IP se redirigirá la petición enviada a la dirección virtual. Para que se lleve a cabo correctamente el envío de la petición a la dirección virtual, es necesario establecer una entrada *Address Resolution Protocol* (ARP) estática en el cliente para determinar el destino de la dirección IP virtual, en este caso será así: `h1 arp -s 192.168.1.10 00:00:00:00:00:10`.

Configuración en el *switch*

En el *switch* se va a especificar una regla que hará que las peticiones que van destinadas a la IP virtual sean redirigidas a una tabla de grupo de tipo *select* descrita en la siguiente Sección 6.2.2. La regla es la siguiente:

```
sudo ovs-ofctl --protocols=OpenFlow13 add-flow s1
ip,nw_dst=192.168.1.10,priority=32769,actions=group:1
```

En ella se especifica el uso del protocolo Openflow 1.3, y se añade un flujo (*add-flow*) de forma que las peticiones enviadas al *switch* (s1) con dirección destino la IP virtual (192.168.1.10), serán dirigidas al grupo que se ha implementado con el identificador 1 (*actions=group:1*).

6.2.2. Lógica en el controlador

Para llevar a cabo el balanceador de carga se hará uso de una tabla de grupo de tipo *select*. Dado que las tablas de tipo *select* envían los paquetes solamente por uno de los *bucket* (los *bucket* se explicaron en la Sección 5.2.3). La forma en la que funciona es la siguiente:

1. Se tiene una tabla de tipo *select* para implementar en el *switch*, la cual contará con dos *buckets* (para ver el código ir al Apéndice A.1).

2. Cada *bucket* contiene la dirección de destino de uno de los equipos ATS. De forma que en un *bucket* se tiene la dirección de destino 192.168.1.4 y en otra la 192.168.1.5.

3. Cada uno de los *buckets* tiene una prioridad o peso (*weight*). A mayor prioridad, mayor es la probabilidad de enviar flujos por el *bucket*.

4. Se realiza el balanceo de carga modificando el valor del peso, asignando un mayor peso hacia donde queremos un mayor tráfico de datos.

5. Mediante el uso del módulo Restconf, el controlador hace una petición "GET" al *switch*, para así obtener los datos de los contadores de los puertos (tal y como vimos en la Sección 5.2.4) que van a los dos equipos de ATS.

6. Una vez obtenidos los datos de ambos puertos, se filtra el dato de "paquetes transmitidos", que será usado para llevar a cabo el balanceo de carga. De forma que se compararán los paquetes transmitidos de ambos puertos.

7. En base a un umbral fijado entre el tráfico enviado a ambos equipos (según el escenario), se incrementará la prioridad de tráfico hacia un equipo Traffic Server u otro.

8. La modificación de la prioridad en la tabla de grupo se llevará a cabo haciendo uso también del módulo Restconf, en este caso mediante un "PUT" al *switch*.

Petición Restconf

Tal y como hemos visto en los puntos 5 y 8 anteriores, es necesario hacer uso del módulo Restconf para llevar a cabo las peticiones "GET" y "PUT". Restconf permite operar con el banco de datos definido en YANG. El módulo Restconf se encuentra implementado en el artefacto `sal-rest-connector`, que se encuentra en los *bundles/features* de Karaf, para instalarlo fácilmente. Por tanto, tras ejecutar Karaf, se puede llevar a cabo la instalación de este módulo ejecutando simplemente `feature:install odl-restconf-all`. Este módulo de Restconf escucha en el puerto 8181 las peticiones de tipo HTTP (en versiones antiguas de ODL era el 8080).

Haciendo uso de Restconf se puede acceder a los datos almacenados en el controlador. El tipo de datos que son almacenados pueden ser:

- *Config*: Contienen datos que son insertados a través del controlador. Es decir, aquellos datos de configuración que los usuarios introducen.
- *Operational*: Contiene datos que como tal no son configurados por el usuario. Es decir, información operacional que viene del sistema y que normalmente se lee para comprobar el estado del sistema.

En nuestro caso, el dato que se quiere obtener concretamente es la información del tráfico de los puertos, se trata de información tipo *operational*. La dirección Restconf al puerto 1 (el que conecta el *switch* con el equipo 192.168.1.4) quedaría de la siguiente manera en la URI de petición:

```
http://localhost:8181/restconf/operational/opendaylight-
inventory:nodes/node/openflow:1/node-connector/openflow:1:1
```

- La petición se realiza a `localhost:8181` que es donde está escuchando el módulo Restconf.
- En la dirección, hay un apartado `/restconf/operational/`, que es el caso de los datos que se quieren tomar. También podría tratarse de datos tipo *config*.
- `opendaylight-inventory:nodes/node/openflow:1`, significa que la petición se hace a uno de los nodos (equipos) del inventario, concretamente con `openflow:1` se hace referencia al *switch*.
- `node-connector/openflow:1:1` hace referencia al enlace (`connector`) del equipo conectado al puerto 1 del *switch* (`openflow:1:1`), en este caso el equipo 192.168.1.4.

Lo anteriormente visto es para llevar a cabo el "GET", para obtener la información de un puerto concreto. Posteriormente con la información

obtenida (como se aprecia en la Figura 6.5), se puede filtrar el contenido necesitado, en este caso la información que se usará será la de los paquetes transmitidos (*packets transmitted*).

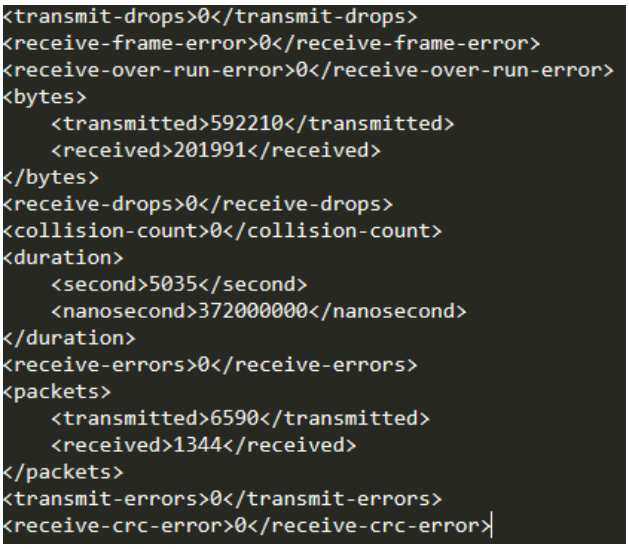


Figura 6.5: Datos obtenidos en el controlador tras realizar la petición "GET".

Para introducir los datos de grupo, y de esa forma modificar la prioridad del grupo tal y como se vio en el punto 8 del Apartado 6.2.2, se realizará de forma similar. En este caso, en vez de llevar a cabo una petición "GET" se realiza un "PUT", y la URI a la que se llevará a cabo la petición es de la siguiente forma:

`http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/group:1`

Ahora, en vez de indicarse un puerto, se indica el grupo con el id 1 que hay en el *switch*, es decir, `openflow:1/group:1`. Dado que mediante un "PUT" se lleva a cabo una modificación de un recurso, es necesario introducir qué parámetros se van a modificar. En este caso en el grupo 1 queremos modificar la prioridad de los *buckets* del grupo *select*. El código que llevará a cabo esta acción se encuentra detallado en el Apéndice A.3. En él se indica lo comentado anteriormente en los puntos 1 y 2, aunque adaptado para ser enviado mediante Restconf, para que el controlador pueda procesar la información y enviarla al *switch*.

En la Figura 6.6, se aprecia parte del código implementado en la lógica del módulo de OpenDaylight. En él se lleva a cabo la petición de tipo REST encargada de modificar los parámetros de la tabla de grupo del *switch*. La lógica de OpenDaylight se ha llevado a cabo haciendo uso de Eclipse (Java).

```
//////////////////////////////////////
//Actualización del grupo Select (PUT)
//////////////////////////////////////
// URL = base URL + grupo
URL url = new URL(baseURL + "/" + numeroGrupo);

// Autenticación
String authStr = user + ":" + password;
//Base64
String encodedAuthStr = Base64.getEncoder().encodeToString(authStr.getBytes());

// Conexión Http
HttpURLConnection connection = (HttpURLConnection) url.openConnection();

// Propiedades de conexión
connection.setRequestMethod("PUT");
connection.setRequestProperty("Authorization", "Basic " + encodedAuthStr);
connection.setRequestProperty("Accept", "application/xml");
//Para introducir contenido PUT
connection.setRequestProperty("Content-Type", "application/xml" );
connection.setDoOutput(true);

OutputStreamWriter out = new OutputStreamWriter(connection.getOutputStream());
```

Figura 6.6: Código de la petición tipo "PUT", desarrollado en Java.

6.2.3. Escenario final

Una vez configurados todos los componentes necesarios llevaremos a cabo la ejecución.

1. "Usuario" lleva a cabo una petición HTTP a una IP virtual, la IP 192.168.1.10.

2. La petición llega al *switch*, el cual tiene la regla que dirige las peticiones a la IP virtual a la tabla de tipo *select*.

3. La llegada de un paquete al controlador activa la aplicación encargada de comprobar los paquetes transmitidos por cada puerto, mediante una petición "GET" al *switch*. En caso de superar el umbral del puerto actualmente utilizado, la tabla será modificada mediante un "PUT" a la tabla de grupo del *switch*, de forma que asignará mayor prioridad al otro puerto. En caso de no superar el umbral, las prioridades de la tabla no serán modificadas.

4. La petición es enviada al equipo correspondiente del *bucket* elegido de la tabla de grupo.

5. El equipo con Traffic Server sirve la información en caso de tenerla en caché. En caso de no tenerla lo consulta al servidor web, y posteriormente la guardará en caché y lo enviará al cliente.

6. El usuario recibe el contenido solicitado.

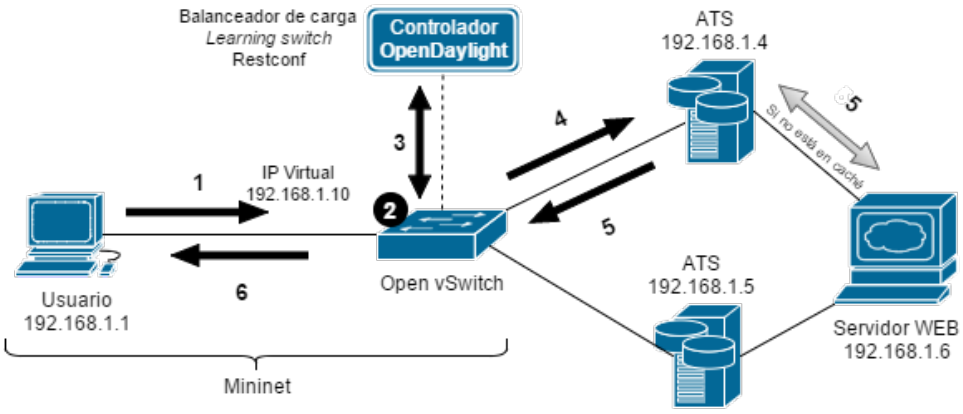


Figura 6.7: Escenario SDN final.

6.3. Evaluación

En esta sección vamos a comprobar los resultados obtenidos en la ejecución del escenario anteriormente visto. Adicionalmente, se implementarán otros escenarios, bajo una serie de supuestos, de forma que podamos comprobar la versatilidad, ante diferentes situaciones, de la aplicación que hemos desarrollado en este proyecto.

6.3.1. Escenario principal

Escenario 1: Balanceo de carga proporcional (alternando buckets)

En este escenario vamos a llevar a cabo la función planteada en el Apartado anterior (6.2.3). Es decir, se hará uso de las estadísticas de los puertos del *switch* por los que se enviarán las peticiones a los equipos ATS, de forma que se comprobará el tráfico que circula en los dos puertos, y en función de un umbral, la información será enviada por un *bucket* u otro.

El umbral del que se hará uso será de un 5 por ciento. De forma que los paquetes serán enviados por un puerto, hasta que el número de paquetes totales enviados por dicho puerto sea un 5 por ciento superior al número de paquetes enviados por el otro. En ese momento se cambiaría el *bucket* para que la información fuera enviada por el otro puerto, hasta que nuevamente se superara dicho umbral del 5 por ciento. De forma que se realizará un balanceo de carga en función del número total de paquetes enviados.

Para comprobar el funcionamiento de este sistema se realizará una petición a la dirección IP Virtual, para comprobar que se pueden realizar peticiones entre el equipo “Usuario” y los equipos ATS.

Haciendo una petición *curl* al equipo 192.168.1.10 (IP Virtual), se puede comprobar que responde correctamente tal y como muestra en la Figura 6.8.

```
mininet> h1 curl 192.168.1.10
<html><body><h1>Este es el SERVIDOR 3!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
<img src="3.jpg" alt="3" height="242" width="242">
</body></html>
mininet>
```

Figura 6.8: Petición a un *proxy*.

Haciendo uso de ODL, con el comando `log:tail` en la terminal de ODL se puede obtener un *log* en tiempo real del programa que implementa la lógica del balanceador. Esto será usado para comprobar el *bucket* actual seleccionado. Tal y como vemos en la imagen 6.9, el *bucket* actualmente en uso es el número 2, es decir, las peticiones son enviadas al equipo 192.168.1.5.

```
RJS - Paquetes enviados por el puerto 1: 10
RJS - Bucket actual: 2
RJS - Cociente entre paquetes (p1/p2): 1.0
RJS -
```

Figura 6.9: *Bucket* actual en uso.

Si se inspeccionan los paquetes con Wireshark en el equipo 192.168.1.5. Se puede apreciar en la Figura 6.10, que efectivamente la petición se ha llevado a cabo al equipo 192.168.1.5. Concretamente, el equipo 192.168.1.5 no ha necesitado contactar con el servidor origen porque tenía en caché la información.

83	1.574920	192.168.1.5	192.168.1.4	TCP	66	d-s-n > 38579 [ACK]
84	1.576831	192.168.1.1	192.168.1.5	TCP	74	38756 > http [SYN]
85	1.576867	192.168.1.5	192.168.1.1	TCP	74	http > 38756 [SYN,
86	1.577291	192.168.1.1	192.168.1.5	TCP	66	38756 > http [ACK]
87	1.577500	192.168.1.1	192.168.1.5	HTTP	141	GET / HTTP/1.1
88	1.577519	192.168.1.5	192.168.1.1	TCP	66	http > 38756 [ACK]

Figura 6.10: Traza Wireshark de la petición.

En caso de que el número de paquetes enviados al equipo 192.168.1.5 superara el umbral de 5 por ciento, tal y como se aprecia en la Figura 6.11 se llevaría el cambio de *bucket* al número uno, haciendo que las peticiones fueran enviadas al equipo 192.168.1.4

Cuando se envía la información por el puerto del equipo ATS 192.168.1.4, los parámetros de la tabla de grupo son la siguiente forma:

```
RJS - Bucket actual: 2
RJS - Cociente entre paquetes (port1/port2): 0.9475016
RJS - Cociente entre bytes (port1/port2): 0.86614054
RJS - Asignamos bucket 1
```

Figura 6.11: Cambio de *bucket*.

```
sudo ovs-ofctl --protocols=OpenFlow13 add-group
s1 group_id=1,type=select, bucket=weight:100,
mod_dl_dst:08:00:27:82:87:4B,mod_nw_dst:192.168.1.4,
output:1, bucket=weight:1,mod_dl_dst:08:00:27:04:DF:46,
mod_nw_dst:192.168.1.5,output:2
```

El peso asignado al equipo 192.168.1.4 es mucho mayor que el de 192.168.1.5, por lo tanto, la inmensa mayoría del tráfico que entra en este grupo (las peticiones a la IP virtual) se enviará a 192.168.1.4. En caso de querer enviar la información al equipo 192.168.1.5, simplemente se debe revertir los pesos (detallado anteriormente en el Apartado 6.2.3).

Teóricamente, de forma ideal, el escenario debería funcionar como se aprecia en la Figura 6.12. Concretamente, las líneas verticales negras indican el cambio de los pesos de los *bucket* cuando el umbral es superado.

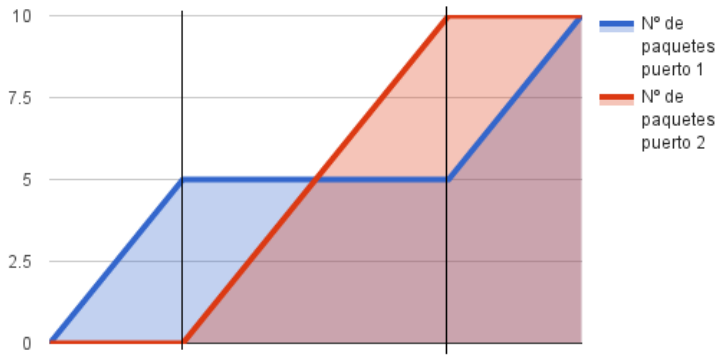


Figura 6.12: Evolución ideal del número de paquetes.

La lógica de la aplicación programada en el controlador, por tanto, funciona de tal forma que el número de paquetes enviado por el puerto 1 y 2 se va igualando cada vez que se supera el umbral. Una vez conocido el comportamiento ideal, se va a ejecutar en Mininet. En la Figura 6.13, se puede apreciar en qué momento el *bucket* de la tabla de grupo aumenta su prio-

ridad hacia un puerto u otro (marcado con líneas verticales). En el puerto del *bucket* que no está siendo usado en ese momento sigue existiendo cierto tráfico debido a otros paquetes, como el refresco de caché entre equipos ATS.

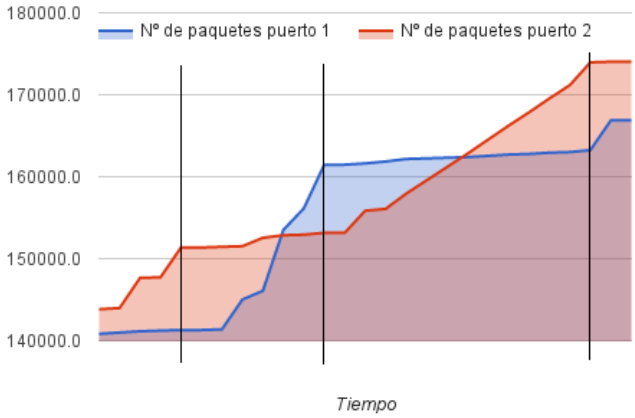


Figura 6.13: Evolución real del número de paquetes

Evolución del número de bytes

A continuación, se va a comprobar la evolución del número de bytes en el Escenario 1. Anteriormente se ha llevado a cabo un sistema en el que se intentaba igualar el número de paquetes en ambos puertos.

Como se puede apreciar en la Figura 6.13, el número de bytes no se iguala de la misma forma que los paquetes, dado que el tamaño de los paquetes puede ser distinto a los enviados por un puerto o por otro, en función de la información requerida. Sin embargo, en la Figura se puede apreciar cuándo se produce el cambio de *bucket*, concretamente, cuando se aprecia un aumento más prominente en el número de bytes en un puerto más que en el otro.

6.3.2. Escenarios alternativos

Escenario 2: Balanceo de carga proporcional (sin pesos)

En este escenario se parte de la base de que ambos equipos ATS tienen similares características, tanto computacionales como de velocidad de red, por lo que la carga de tráfico enviada a ambos debería ser similar para ambos.

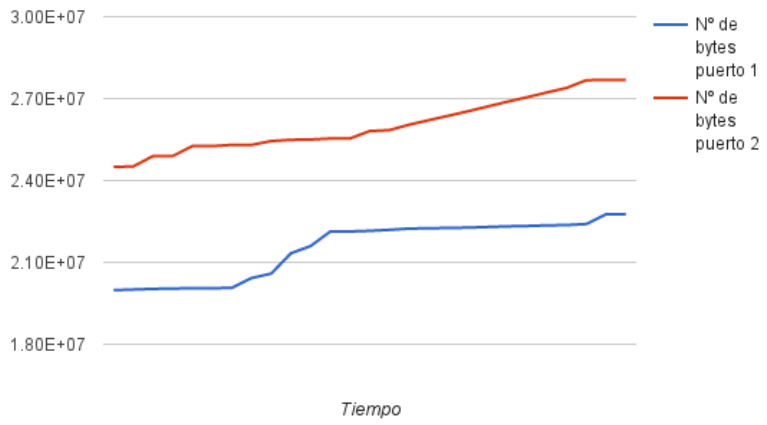


Figura 6.14: Evolución del número de bytes.

Sin embargo, en este caso, se va a llevar otra forma de repartir el tráfico. La forma de operar en este sistema es asignando los mismos pesos (*weight*) a ambos *buckets*, o lo que es lo mismo, no se asignará ningún peso. De forma que se llevará a cabo una distribución de tráfico entre los dos sistemas de forma equitativa, sin necesidad de cambiar la prioridad de los *bucket*.

Por tanto, la tabla de grupo de tipo *select* que deberá tener el *switch* será de la siguiente forma:

```
sudo ovs-ofctl --protocols=OpenFlow13 add-group s1
group_id=1,type=select,bucket=mod_dl_dst:08:00:27:82:87:4B,
mod_nw_dst:192.168.1.4,output:1,bucket=mod_dl_dst:
08:00:27:04:DF:46,mod_nw_dst:192.168.1.5,output:2
```

Es decir, no será necesario especificar el parámetro *weight*.

En la Figura 6.15 tenemos un ejemplo de cómo se debería comportar el sistema teóricamente. Es decir, deberían ser enviados el mismo número de paquetes por ambos puertos.

Tras llevar a cabo su ejecución en el emulador Mininet y obtener los datos de los paquetes enviados, se aprecia en la Figura 6.16 un comportamiento similar al caso ideal. Es decir, la tendencia (líneas negras) es enviar el mismo número de paquetes por los dos puertos, por eso el incremento de paquetes es similar en ambos puertos. Además de las peticiones a los equipos, existe un aumento de tráfico continuo debido a otro tipo de datos que circulan por el escenario.

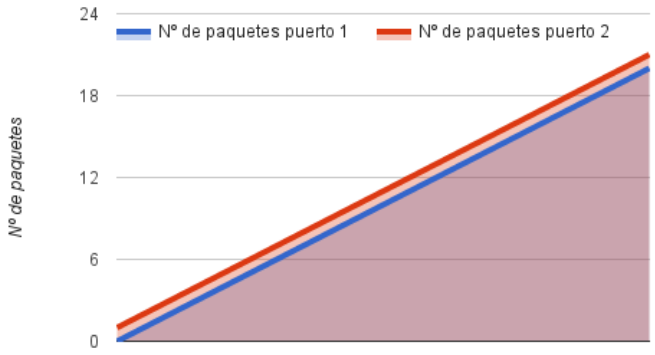


Figura 6.15: Evolución del tráfico ideal en una tabla de grupo *select* sin pesos.

Escenario 3: Enlace de menor capacidad que otro

En este escenario se tiene un enlace con una menor capacidad de ancho de banda, por lo que la velocidad máxima de transmisión del enlace será inferior a la del otro.

Esta característica puede ser simulada haciendo uso de Mininet. De forma que la interfaz de red del *switch* que va conectada al equipo ATS del puerto dos, es decir, el de dirección 192.168.1.5, se va a limitar a una velocidad del 30 por ciento la del enlace que va al equipo ATS con IP 192.168.1.4. Se define esta menor capacidad cuando se generan las interfaces de red del *switch*. En este caso, se hará uso de la función TCIntf, que es capaz de definir interfaces de red con control de tráfico (*Traffic Control*, TC):

```
TCIntf( intfName, node=s1, bw=0.3 )
```

En este escenario lo ideal es disminuir el flujo de datos que circula por el enlace que tiene una velocidad más limitada, para que sea el otro enlace el que soporte una mayor cantidad de peticiones. Así que se establecerá un peso (*weight*) fijo, que haga que el equipo con menor ancho de banda reciba una menor cantidad de tráfico (en torno a una tercera parte). En este caso, partiendo de la base de que el equipo 192.168.1.5 será el que tenga un menor ancho de banda, la tabla de tipo *select* quedará con un mayor *weight* en el *bucket* del equipo 192.168.1.4:

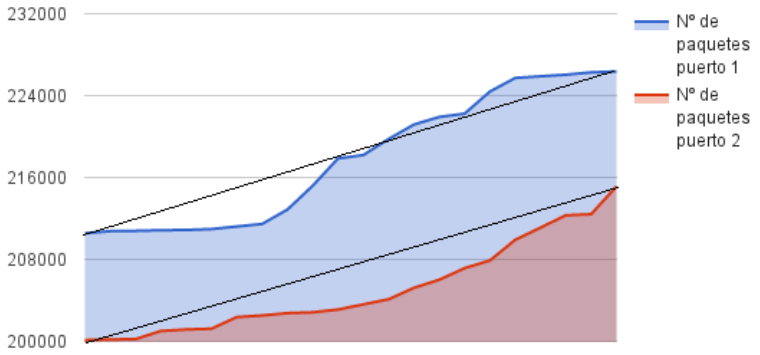


Figura 6.16: Evolución del tráfico usando una tabla de grupo *select* sin pesos.

```
sudo ovs-ofctl --protocols=OpenFlow13 add-group s1 group_id=1,
type=select, bucket=weight:3,mod_dl_dst:08:00:27:82:87:4B,
mod_nw_dst:192.168.1.4,output:1,bucket=weight:1,mod_dl_dst:
08:00:27:7D:8B:1F, mod_nw_dst:192.168.1.5,output:2
```

En la Figura 6.17, una vez más se puede ver el caso ideal en el número de paquetes que deberían ser enviados por cada uno de los puertos, siendo el puerto 2 el que envíe una menor cantidad de información. Tras la ejecución del Escenario 3, se aprecia en la Figura 6.18, que el envío de la información se corresponde con el patrón especificado en el modelo ideal.

6.3.3. Distribución de la caché entre equipos con Apache Traffic Server

Una vez configurado el *cluster* para la actualización de cachés de todos los equipos que forman parte, se realiza una comunicación continua entre estos equipos.

Como se aprecia en la Figura 6.19, existe un tráfico entre los dos equipos que forman parte del *cluster*, porque continuamente están comprobando el estado de disponibilidad y del contenido del otro equipo.

La forma en la que funciona el refresco de la caché es la siguiente:

- ATS1 (192.168.1.4) y ATS2 (192.168.1.5) tienen la información desactualizada, y se encuentran en el mismo *cluster*. Adicionalmente, otra condición es que estén conectados al mismo *switch* o en la misma

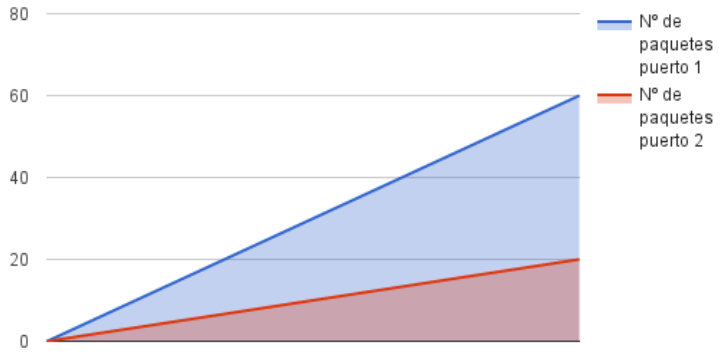


Figura 6.17: Evolución ideal del número de paquetes con un enlace de menor capacidad.

VLAN.

- Cuando uno de los dos equipos, por ejemplo, ATS1, recibe una petición del cliente, al tener la caché desactualizada, hace una petición al servidor de origen (192.168.1.6) para actualizarla.

- Mediante la comunicación entre ambos equipo, ATS2 es consciente de que debe actualizar la caché.

- La caché de ATS2 es actualizada mediante la comunicación con el equipo ATS1, sin necesidad de comunicarse con el servidor de origen.

Para comprobar que efectivamente funciona el cacheo correctamente, se llevará a cabo el borrado de ambas cachés, y se actualizará solo una, la de ATS1, mediante el acceso a la página, tal y como vemos en la Figura 6.20.

Posteriormente, se corta el acceso de ambos equipos al servidor origen. En caso de que no funcionara la actualización de caché, cuando se accediera al contenido de ATS2, no podría aportar la información, pues ATS2 no tiene acceso al servidor origen para obtener la información, sin embargo, sí es capaz de servir dicha información que anteriormente había cacheado el equipo ATS1, tal y como se aprecia en la Figura 6.21. El contenido *HyperText Markup Language* (HTML) es obtenido en su totalidad del otro equipo del *cluster*, aunque el contenido más pesado, como es el caso de las imágenes o vídeos, son aportados por el servidor de origen.

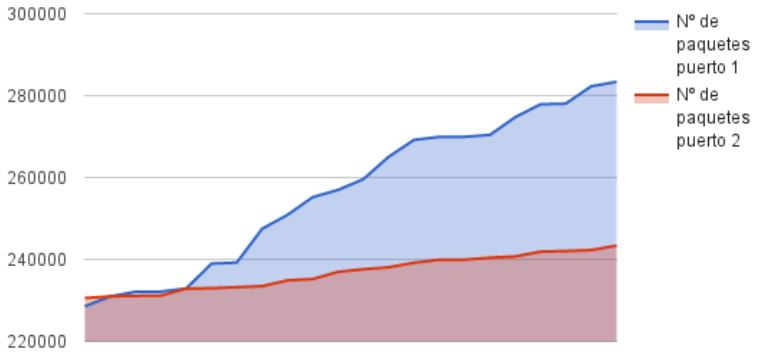


Figura 6.18: Evolución del número de paquetes, con un enlace de menor capacidad.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.4	192.168.1.5	TCP	138	40226 > d-s-n [PSH, ACK]
2	0.000162	192.168.1.5	192.168.1.4	TCP	138	d-s-n > 40226 [PSH, ACK]
3	0.000665	192.168.1.4	192.168.1.5	TCP	66	40226 > d-s-n [ACK] Seq=7
4	0.063332	192.168.1.5	192.168.1.4	TCP	138	d-s-n > 40226 [PSH, ACK]
5	0.064297	192.168.1.4	192.168.1.5	TCP	66	40226 > d-s-n [ACK] Seq=7
6	0.064320	192.168.1.4	192.168.1.5	TCP	138	40226 > d-s-n [PSH, ACK]
7	0.102670	192.168.1.5	192.168.1.4	TCP	66	d-s-n > 40226 [ACK] Seq=1
8	0.103567	192.168.1.4	192.168.1.5	TCP	138	40226 > d-s-n [PSH, ACK]
9	0.103670	192.168.1.5	192.168.1.4	TCP	66	d-s-n > 40226 [ACK] Seq=1
10	0.103780	192.168.1.5	192.168.1.4	TCP	138	d-s-n > 40226 [PSH, ACK]

Figura 6.19: Traza Wireshark.

6.3.4. Pros y contras de utilizar SDN

El uso de tecnología SDN aporta una serie de ventajas con respecto a la tecnología tradicional. A continuación, se identifican las más destacables.

- Facilidad de emulación y exportación en un escenario real.
- Gran cantidad de *software* gratuito, como el emulador Mininet o el controlador OpenDaylight.
- Comunidad colaborativa, aunque aún necesita una mayor base.
- Gran presencia en el ámbito académico y soporte de grandes empresas.
- Tecnología estandarizada, sin necesidad de recurrir a sistemas propietarios.

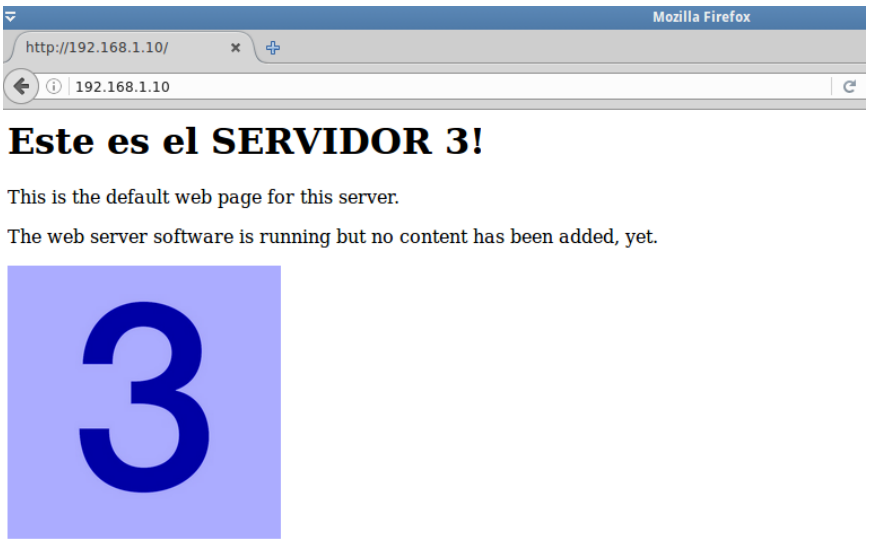


Figura 6.20: Obtención de la página.

- Uso de lenguajes de gran aceptación como Java o Python.
- Facilidad de implantación, una vez aprendidos los conceptos y el uso del *software*.

Sin embargo, los escenarios planteados, también cuenta con una serie de desventajas, especialmente, si los comparamos con una implementación "tradicional":

- SDN es aún tecnología aún en estado de madurez.
- Gran parte de los equipos usados actualmente no están optimizados para el uso de tecnología SDN.
- Difícil la curva de aprendizaje en el uso de *software* como OpenDaylight.
- El *software* tiene bastantes puntos de optimización de cara al desarrollo de funcionalidades. En el caso de ODL, los tiempos de compilación son excesivos (cuando solo se modifica una pequeña parte del código) y las funciones para llevar a cabo *debug* del código pueden ser algo tediosas.
- No todo el *software* tiene soporte actualizado del estándar. La versión del *switch* virtual OVS usado en Mininet no tiene soporte total de las últimas versiones de OpenFlow.

Figura 6.21: Obtención de la página en caché.

Capítulo 7

Conclusiones y Trabajos Futuros

En este capítulo final se van a exponer las conclusiones finales extraídas tras la realización del proyecto.

En primer lugar, se comentarán las conclusiones generales, así como la verificación de los objetivos planteados, y las posibles limitaciones y problemas tenidos en el desarrollo de éste.

Posteriormente, se trazará posibles líneas futuras en la mejora del desarrollo del proyecto, y en el desarrollo de nuevas funcionalidades a partir de los conocimientos adquiridos.

En último lugar, se llevará a cabo una valoración personal de la realización del proyecto.

7.1. Conclusiones

A rasgos generales, tras la conclusión del proyecto, se puede afirmar que se ha finalizado de forma satisfactoria, ya que los requisitos iniciales se han visto cumplidos. El objetivo principal consistía en emular un sistema de distribución de contenidos sobre una red SDN, y se ha visto cumplido llevando a cabo un sistema de cacheo y de balanceo de carga, sobre una red emulada de SDN, sin necesidad de desplegar un sistema costoso, sino sobre un solo *Personal Computer* (PC).

Esta implementación ha ayudado a estudiar multitud de conceptos, relacionados con ambas tecnologías:

- En la parte relacionada con las CDN, se ha estudiado la importancia que tienen actualmente los proveedores de CDN, dada la enorme can-

tidad de tráfico que manejan actualmente, y el aumento que se prevé en los próximos años. Además, se han visto diferentes modelos de distribución de contenido, y la importancia de contar con mecanismos de distribución eficientes que aumenten la calidad de servicio manteniendo unos costes aceptables. Finalmente, se ha usado e incorporado al escenario un *software* (ATS) que actualmente es usado por grandes empresas, y que, además de aportarnos una buena base práctica, se trata de un programa con una gran variedad de funcionalidades.

- En la parte de las redes definidas por *software*, se han estudiado los conceptos básicos para tener una visión amplia de esta tecnología. Tener una base teórica es fundamental para entender también las razones que llevan a la industria a realizar una transición hacia este tipo de redes. Por ello, se ha estudiado la proyección de las redes SDN de cara al futuro, y qué beneficios aportará su arquitectura, mediante la separación del plano de control y el plano de datos. Una vez entendidos los conceptos básicos, se ha podido trabajar haciendo uso del controlador OpenDaylight, lo que ha permitido familiarizarse con la arquitectura y el funcionamiento de las redes SDN de una forma práctica. Gracias al emulador Mininet se ha podido ejecutar una topología con un *switch* compatible con OpenFlow, de forma similar a si se estuviera usando en un equipo físico, por lo que los conocimientos vistos a lo largo del desarrollo del proyecto son especialmente útiles desde el punto de vista formativo.

- La cantidad de conceptos estudiados en el desarrollo del trabajo han sido especialmente amplios, por lo que en el aspecto de aprendizaje en materia relacionada con las redes SDN y las redes de distribución de contenido, han sido más que satisfactorias. Adicionalmente, los recursos prácticos han permitido tener una buena visión de trabajo con herramientas que actualmente son utilizadas para la distribución de contenidos y para la creación de redes que hacen uso de la tecnología SDN, haciendo que el trabajo desarrollado sea un proyecto funcional que podría ser implementado en una red existente (siempre y cuando se cuente con los recursos necesarios).

Es cierto que no toda la realización del proyecto ha sido sencilla, existían una serie de limitaciones y problemas que dificultaban el proceso de desarrollo:

- Por un lado, era necesaria una máquina potente que fuera capaz de ejecutar hasta 3 máquinas virtuales, una de ellas requerían una gran cantidad de RAM. Por lo que finalmente se ha tenido que incrementar la RAM del equipo a uno con 16 GB de RAM.

- Los tiempos de compilación del controlador hacía que el tiempo medio rondara en torno a los 6-7 minutos. Este se convertía en un proceso tedioso pues cualquier modificación de código en la lógica de nuestro controlador, necesitaba llevar a cabo la compilación. Los tiempos de compilación se vieron reducidos con las instalación de un *Solid-State Disk* (SSD), a un tiempo aproximado de 3-4 minutos.

7.2. Líneas futuras

La distribución de contenidos juega un papel fundamental en las redes actuales. Los operadores de infraestructuras y servicios necesitan buscar continuamente mecanismos de eficiencia que les ayude a mejorar sus prestaciones al dar un servicio. Precisamente, con el uso de SDN se ha podido comprobar que tener una visión global de la red aporta una gran versatilidad para llevar a cabo la gestión de la red, teniendo así una gran facilidad en la implementación de mecanismos que permitan realizar una distribución del contenido de una forma más eficiente. Por tanto, desde el punto de vista de desarrollo posterior:

- Es posible aplicar la lógica del controlador a una red SDN existente con relativamente pocas modificaciones en el código, para ayudar a proveedores de servicios a llevar a cabo un balanceo de carga entre sus equipos, en base a los parámetros usados.
- Se abre la posibilidad de incorporar un mayor número de posibilidades a la implementación de balanceo de carga. Se puede obtener un mayor número de parámetros (datos operacionales) de la red para así optimizar el proceso de balanceo. Estos parámetros podrían ser número de errores en un puerto, número de paquetes descartados, disponibilidad del enlace, etc. Un ejemplo de este caso, hallando el número de errores en el puerto, es comentado en la Sección 7.2.1.
- La topología SDN implementada en Mininet nos permite crear de manera sencilla un banco de pruebas para futuras implementaciones de módulos en el controlador SDN, por lo que no es necesario recurrir a un equipo físico para realizar una aplicación en el controlador en un futuro.

7.2.1. Balanceo de carga en caso de existir errores

Si de forma similar a los escenarios planteados en la implementación, se tuviera un enlace que presentara un mayor número de errores (en un principio, de forma no previsible), se podría hacer uso de la aplicación Restconf,

instalada en el controlador, para así obtener datos operacionales sobre los errores que se producen en dicho enlace. Es decir, en vez de obtener datos del número de paquetes enviados por los puertos, se comprobaría el número de errores (parámetro *transmit-errors*), y en función de estos, se podría enviar la información por un *bucket* u otro, de forma similar a la vista en el Apartado 6.3.1.

Con Mininet se puede emular el comportamiento de pérdidas, por ejemplo, del 50 por ciento, haciendo uso de interfaces con tráfico de control. De la siguiente forma:

```
TCIntf( intfName, node=s1, loss=50 )
```

7.2.2. Actualización de la caché haciendo uso del controlador

En el modelo ejecutado en la implementación, el refresco de la caché se lleva a cabo haciendo uso de un sistema de *multicast* que incorpora Apache Traffic Server, entre los equipos ATS que se encuentran dentro de un *cluster* configurado. Sin embargo, se podría haber modelado un sistema de actualización de la caché haciendo uso exclusivamente de tecnología SDN. De forma que no hiciera falta un programa tercero para actualizarla, sino haciendo directamente uso de tecnología de red.

Utilizando la versatilidad que aportan los controladores SDN, éste podría detectar (mediante el tipo de petición), cuándo un equipo ATS debe acceder al servidor de origen para refrescar su caché. En ese momento, es cuando el controlador duplicaría la petición, cambiando la dirección de origen, para que se realizara no solo desde el equipo que necesita la información, sino desde todos los del *cluster*. De esta forma, todos los equipos actualicen su información simultáneamente. Sin embargo, haciendo uso de este método hay que lidiar con los mecanismos de *handshake* que se realizan en las peticiones TCP y podría tener cierta dificultad de implementar.

7.2.3. Obtención de estadísticas haciendo uso de Apache Traffic Server

Para llevar a cabo la obtención de estadísticas, en la implementación se ha hecho uso del módulo Restconf, de forma que en base a los datos operacionales aportados por el *switch* se realizaba el balanceo de carga. Sin embargo, la aplicación ATS cuenta con un *plugin* que permite obtener información de los equipos ATS simplemente realizando sobre ellos una petición HTTP.

La información obtenida, es sobre datos locales de las máquinas ATS, datos como paquetes o bytes enviados, número de errores, peticiones, flujos, etc.

Por tanto, en vez de hacer uso de los datos operacionales ofrecidos por el *switch* con Openflow, se podría haber hecho uso de un programa de terceros (ATS) para modelar el tráfico en función de estos datos. Sin embargo, finalmente se optó por utilizar los datos que aportaba la red SDN, con el objetivo de reducir el número de peticiones.

7.3. Valoración personal

Finalmente se expondrán una serie de valoraciones personales una vez finalizada la realización del proyecto.

Primero, a nivel personal se puede afirmar que se han cumplido todos los objetivos inicialmente planteados, a pesar de las problemáticas surgidas, el resultado final ha dado un producto que es funcional y útil, especialmente si fuera planteado en un escenario de mayor escala. Aun así, como prueba de concepto, es un resultado más que satisfactorio.

Adicionalmente, la realización del TFM, ha dado lugar a la necesidad de llevar a cabo una planificación, y una constancia en el desarrollo. Esto ha permitido aprender la importancia de seguir con rigor una serie de pautas en la realización del proyecto, así como, saber medir, prever y abordar con agilidad los posibles problemas surgidos durante el desarrollo, pues los tiempos se pueden alargar más allá de lo previsto y afectar el conjunto del trabajo realizado. Al fin y al cabo, esto es una competencia necesaria que se debe tener también en el mundo laboral.

Una de las principales motivaciones personales para la realización de este proyecto, era la familiarización con tecnologías SDN, pues se ha comprobado que existe una gran oportunidad en el mercado, relacionada con la aportación de servicios de este tipo de redes. Así pues, en un plazo medio-largo jugarán un papel fundamental en el mundo de las telecomunicaciones. Por tanto, se ha intentado tener una gran cantidad de conocimientos durante la realización del trabajo, de manera que, a pesar de la gran variedad de términos y conceptos, se pudiera estudiar con suficiente profundidad.

La parte práctica ha sido la que ha aportado un buen punto de contacto en la consolidación de conceptos teóricos, y obviamente en el desarrollo de habilidades en la implementación y configuración de las redes SDN. Este es personalmente quizás el punto más motivante del desarrollo del proyecto, pues actualmente, la cantidad de profesionales que cuentan con conocimientos de redes SDN es reducida, lo que daría lugar a una gran cantidad de posibilidades laborales en los próximos años, pues se trataría de un perfil bastante cotizado.

En último lugar, se puede afirmar, por tanto, que existen un buen grado de satisfacción en un nivel personal con la realización del proyecto y

los conocimientos adquiridos, y también a nivel profesional, pues trabajar con tecnología actual (e incluso innovadora), supone un conocimiento muy interesante en ambas facetas.

Bibliografía

[1] 5G Research in HORIZON 2020. https://5g-ppp.eu/european-5g-actions/.

[2] Akamai. https://www.akamai.com/.

[3] Amazon Web Services. https://aws.amazon.com.

[4] CDN Planet. http://www.cdnplanet.com/.

[5] CoDeeN. http://codeen.cs.princeton.edu.

[6] CoDeeN. http://www.coralcdn.org.

[7] COMODIN. http://lisdip.deis.unical.it/index.html.

[8] Globule. http://www.globule.org.

[9] Level 3 Communications. http://www.level3.com.

[10] Limelight networks. https://www.limelight.com/.

[11] Mininet: An Instant Virtual Network on your Laptop (or other PC). http://mininet.org.

[12] Open vswitch faq. https://github.com/openvswitch/ovs/blob/master/FAQ.md. última visita: 28-08-2016.

[13] OpenDaylight: Open Source SDN Platform. https://www.opendaylight.org.

[14] PLANETLAB. https://www.planet-lab.org.

[15] SDN Hub. http://www.sdnhub.org.

[16] Open Networking Foundation. www.opennetworking.org, 2011.

[17] Network-Functions Virtualization (NFV) Proofs of Concept; Framework. GS NFV-PER 002 v1.1.1, 2013.

[18] CDN Market Size In 2015 And 2019. http://www.bizety.com, August 2015.

[19] NFV vs. VNF: What's the difference? http://searchsdn.techtarget.com/answer/NFV-vs-VNF-Whats-the-difference, July 2015.

[20] Strong dollar hurts Akamai's profit forecast, shares fall. http://www.reuters.com/, April 2015.

[21] Telefónica and NEC win innovation award for virtualized home gateway trials. http://www.gsma.com, January 2015.

[22] Global content delivery network (cdn) market insights, opportunity analysis, market shares and forecast 2016 - 2022. March 2016.

[23] Sdn market to experience strong growth over next several years, according to idc. *IDC*, Feb 2016.

[24] Joe Abley and Kurt Erik Lindqvist. Operation of anycast services. 2006.

[25] Pablo Ameigeiras, Juan J Ramos-Munoz, Laurent Schumacher, Jonathan Prados-Garzon, Jorge Navarro-Ortiz, and Juan M Lopez-Soler. Link-level access cloud architecture design based on sdn for 5g networks. *IEEE Network*, 29(2):24–31, 2015.

[26] Hooman Beheshti. The rise of event-driven content. https://www.fastly.com/blog/rise-event-driven-content-or-how-cache-more-edge.

[27] Rajkumar Buyya, Mukaddim Pathan, and Athena Vakali. *Content delivery networks*, volume 9. Springer Science & Business Media, 2008.

[28] Dukhyun Chang, Junho Suh, Hyogi Jung, Ted Taekyoung Kwon, and Yanghee Choi. How to realize cdn interconnection (cdni) over openflow? In *Proceedings of the 7th International Conference on Future Internet Technologies*, pages 29–30. ACM, 2012.

[29] Regis Donovan. How IT Works Domain Name System. https://technet.microsoft.com/en-us/magazine/2005.01.howitworksdns.aspx.

[30] Gorry Fairhurst. Unicast, Broadcast, Multicast and Ethernet MAC Addres. http://www.erg.abdn.ac.uk/users/gorry/course/intro-pages/uni-b-mcast.html.

[31] Apache Software Foundation. Apache software foundation. http://trafficserver.apache.org/users.html.

[32] Apache Software Foundation. Apache software foundation. `http://www.apache.org`.

[33] Apache Software Foundation. Apache traffic server. `http://trafficserver.apache.org`.

[34] Apache Software Foundation. Apache traffic server documentation v.7 .0. `https://docs.trafficserver.apache.org/en/latest/index.html`, 2015.

[35] OpenDaylight foundation. Opendaylight controller:md-sal. `https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL`. última visita: 18-08-2016.

[36] OpenDaylight foundation. Opendaylight: Release plan. `https://wiki.opendaylight.org/view/Release_Plan`.

[37] The Linux Foundation. Linux foundation. `https://www.linuxfoundation.org/`.

[38] Open Networking Fundation. Software-defined networking: The new norm for networks. *ONF White Paper*, 2012.

[39] Open Networking Fundation. Software-defined networking: Architecture overview. *ONF White Paper*, 2014.

[40] D. Gajjar, H. Kotak, and H. Joshi. Round robin load balancer using software defined networking (sdn). April 2016.

[41] Paul Goransson and Chuck Black. *Software Defined Networks: A Comprehensive Approach*. Elsevier, 2014.

[42] Vijay K Gurbani, Michael Scharf, TV Lakshman, Volker Hilt, and Enrico Marocco. Abstracting network state in software defined networks (sdn) for rendezvous services. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6627–6632. IEEE, 2012.

[43] Lawrence Harte. *Introduction to Data Multicasting, IP Multicast Streaming for Audio and Video Media Distribution*. Althos, 2008.

[44] Ahmad Hemid. Facilitation of the opendaylight architecture. 2014.

[45] Enrique Hernandez-Valencia, Steven Izzo, and Beth Polonsky. How will nfv/sdn transform service provider opex? *Network, IEEE*, 29(3):60–67, 2015.

[46] Markus Hofmann and Leland R Beaumont. *Content networking: architecture, protocols, and practice*. Elsevier, 2005.

[47] S. Kaur, K. Kumar, J. Singh, and Navtej Singh Ghumman. Round-robin based load balancing in software defined networking. In *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*, pages 2136–2139, 2015.

[48] F.T. Leighton and D.M. Lewin. Content delivery network using edge-of-network servers for providing content delivery to a set of participating content providers, April 22 2003. US Patent 6,553,413.

[49] Diego R. López. Redes basadas en software. qué son, cómo se aplican ahora y se aplicarán en el futuro. ETSIIT, 2016. Máster en Ingeniería de Telecomunicación.

[50] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data. *The management revolution. Harvard Bus Rev*, 90(10):61–67, 2012.

[51] Nick McKeown. Software-defined networking. *INFOCOM keynote talk*, 17(2):30–32, 2009.

[52] B Molina, CE Palau, and M Esteve. Estudio y modelado de una red de distribución de contenido. *IEEE América Latina*, 3:44–48, 2005.

[53] Amos Ndegwa. What is Micro-Caching? `https://www.maxcdn.com/one/visual-glossary/micro-caching/`.

[54] nginx. nginx website. `https://nginx.org/`.

[55] Jakob Nielsen. Traffic log patterns. July 2006.

[56] ON.Lab. Onos, open netowrk operating system. `http://onosproject.org/`.

[57] Al-Mukaddim Khan Pathan and Rajkumar Buyya. A taxonomy and survey of content delivery networks. *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, page 4, 2007.

[58] A Linux Foundation Collaborative Project. Open vswitch. `http://openvswitch.org`. última visita: 18-08-2016.

[59] Jarno Rajahalme, Shane Amante, Sheng Jiang, and Brian Carpenter. Ipv6 flow label specification. 2011.

[60] Shirin Esfandiari Ramón Jesús Millán Tejedor. Qué es... nfv (network functions virtualization). *bit*, (196):66–70, 2014.

[61] Ranjani S Senthil Ganesh N. Dynamic load balancing using software defined networks. 2015.

[62] Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, 2013.

[63] Myung-Ki Shin, Ki-Hyuk Nam, and Hyoung-Jun Kim. Software-defined networking (sdn): A reference architecture and open apis. In *2012 International Conference on ICT Convergence (ICTC)*, pages 360–361. IEEE, 2012.

[64] OpenFlow Specification. v1. 1.0.

[65] OpenFlow Specification. v1. 3.0.

[66] OpenFlow Specification. v1. 5.0.

[67] Konstantinos Stamos, George Pallis, Athena Vakali, and Marios D Dikaiakos. Evaluating the utility of content delivery networks. In *Proceedings of the 4th edition of the UPGRADE-CN workshop on Use of P2P, GRID and agents for the development of content networks*, pages 11–20. ACM, 2009.

[68] Alexandru L Stancu, Simona Halunga, Alexandru Vulpe, George Suciu, Octavian Fratu, and Eduard C Popovici. A comparison between several software defined networking controllers. In *Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS), 2015 12th International Conference on*, pages 223–226. IEEE, 2015.

[69] Cisco Systems. Cisco visual networking index. 2015.

[70] Attila Takacs, Elisa Bellagamba, and Joe Wilke. Software-defined networking: the service provider perspective. *Ericsson Review*, 2:2–8, 2013.

[71] Ramón Jesús Millán Tejedor. Qué es... sdn (software defined networking). *bit*, (202):81–84, 2 2016.

[72] Varnish. Varnish http cache. `https://varnish-cache.org`.

[73] Murrel W. Getting started: Why it's time to learn Python for SDN. `http://searchsdn.techtarget.com/tip/Getting-started-Why-its-time-to-learn-Python-for-SDN`.

[74] Matthias Wichtlhuber, Robert Reinecke, and David Hausheer. An sdn-based cdn/isp collaboration architecture for managing high-volume flows. *Network and Service Management, IEEE Transactions on*, 12(1):48–60, 2015.

Apéndice A

Código

A.1. Grupo select en el switch Mininet

Creación de grupo *select* con dos *bucket* con prioridades distintas. En este caso, el peso (*weight*) del primer *bucket* es de 100, frente a 1 del segundo, por lo que la inmensa mayoría de los paquetes serán enviados por el primer *bucket* (con IP 192.168.1.4).

```
sudo ovs-ofctl --protocols=OpenFlow13 add-group s1 group_id=1,type=
    select,bucket=weight:100,mod_dl_dst:08:00:27:82:87:4B,mod_nw_dst
    :192.168.1.4,output:1,bucket=weight:1,mod_dl_dst:08:00:27:04:DF
    :46,mod_nw_dst:192.168.1.5,output:2
```

A.2. Código de la topología de Mininet

```
#Initialization
        net=Mininet(controller=RemoteController, switch=
            OVSKernelSwitch, listenPort=6634)

#Add controller c0
        c0 = net.addController( name='c0', controller=RemoteController
            , protocols='OpenFlow13', ip='127.0.0.1', port=6633)

#Add switch s1
        s1 = net.addSwitch( 's1', cls=OVSSwitch, mac
            ='00:00:00:00:00:10', protocols='OpenFlow13' )

#Add interface eth1 (ATS1)
        intfName = 'eth1'
        checkIntf( intfName )
        Intf( intfName, node=s1 )

#Add interface eth2 (ATS2)
        intfName = 'eth2'
```

```
17          checkIntf( intfName )
18          Intf( intfName, node=s1 )
19
20  #Add interface eth3 (Server)
21          intfName = 'eth3'
22          checkIntf( intfName )
23          Intf( intfName, node=s1 )
24
25  #Add host h1
26          h1 = net.addHost( 'h1', mac='00:00:00:00:00:01', ip
                ='192.168.1.1/24', defaultRoute='via 192.168.1.10' )
27
28  #Add link between h1 y s1
29          net.addLink( s1, h1 )
30
31  #Starting network
32          net.build()
33          net.start()
34
35          for controller in net.controllers:
36                  controller.start()
37          net.get('s1').start([c0])
38          CLI(net)
39          net.stop()
40
41  if __name__ == '__main__':
42          setLogLevel( 'info' )
43          myNetwork()
```

A.3. Código del grupo select vía Restconf

Grupo de tipo *select*, introducido vía Restconf. Enviado en formato *Extensible Markup Language* (XML). En este caso para que el *bucket* 1 tenga una mayor prioridad (*weight* de 100) que el *bucket* 2 (*weight* de 1).

```
1  http://localhost:8181/restconf/config/opendaylight-inventory:nodes/
       node/openflow:1/group/1
2  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
3  <group xmlns="urn:opendaylight:flow:inventory">
4      <group-type>group-select</group-type>
5       <buckets>
6          <bucket>
7           <weight>100</weight>
8              <action>
9                  <set-field>
10                     <ethernet-match>
11                         <ethernet-destination><address>
                               08:00:27:82:87:4B</address></ethernet-
                               destination>
12                     </ethernet-match>
13                 </set-field>
14                 <order>1</order>
15            </action>
16            <action>
17                 <set-field>
18                    <ipv4-destination>192.168.1.4/32</ipv4-destination>
```

```
                </set-field>
                <order>2</order>
            </action>
            <action>
                <output-action>
                    <output-node-connector>1</output-node-connector>
                </output-action>
                <order>3</order>
            </action>
            <bucket-id>1</bucket-id>
        </bucket>
        <bucket>
          <weight>1</weight>
            <action>
                <set-field>
                <ethernet-match>
                    <ethernet-destination><address>08:00:27:04:DF:46</
                        address></ethernet-destination>
                </ethernet-match>
                </set-field>
                <order>1</order>
            </action>
            <action>
                <set-field>
                   <ipv4-destination>192.168.1.5/32</ipv4-destination>
                </set-field>
                <order>2</order>
            </action>
            <action>
                <output-action>
                    <output-node-connector>2</output-node-connector>
                </output-action>
                <order>3</order>
            </action>
            <bucket-id>2</bucket-id>
        </bucket>
    </buckets>
    <barrier>false</barrier>
    <group-name>SelectGroup</group-name>
    <group-id>1</group-id>
</group>
```

A.4. Configuración de un cluster en Apache Traffic Server

En el archivo records.config se lleva a cabo la configuración de un *cluster* mediante los siguientes comandos. Es necesario implementarlo en todos los equipos ATS para formar parte del *cluster*.

```
LOCAL proxy.local.cluster.type INT 1
CONFIG proxy.config.cluster.ethernet_interface STRING eth1
CONFIG proxy.config.cluster.cluster_port INT 8086
CONFIG proxy.config.cluster.rsport INT 8088
CONFIG proxy.config.cluster.mcport INT 8089
CONFIG proxy.config.cluster.mc_group_addr STRING 224.0.1.37
```

```
7 CONFIG proxy.config.proxy_name STRING MiCluster
8 CONFIG proxy.config.cluster.cluster_configuration STRING cluster.
      config
```

A.5. Asignación de URL en Apache Traffic Server

En el archivo remap.config, con el siguiente comando se puede redirigir de una dirección a otra una petición. En este caso de 192.168.1.4 a 192.168.1.6: `map http://192.168.1.4:80/ http://192.168.1.6:80/`

Apéndice B

Glosario

- ALTO: Application Layer Traffic Optimization
- API (Application Programming Interface): Interfaz de programación de aplicaciones
- OPEX (Operating Expense): Coste permanente para el funcionamiento de un producto, negocio o sistema. Puede traducirse como gasto de funcionamiento, gastos operativos, o gastos operacionales.
- CAPEX (Capital Expenditures): Inversiones de capital que crean beneficios. Un CAPEX se ejecuta cuando un negocio invierte en la compra de un activo fijo o para añadir valor a un activo existente con una vida útil que se extiende más allá del año imponible.
- TCO (Total Cost of Ownership): Coste total de propiedad. Determina los costes directos e indirectos, así como beneficios, sobre la compra de equipos o programas informáticos.
- COTS (Commercial off-the-shelf): Literalmente significa componente sacado del estante. Es un término formal para elementos comerciales, incluidos servicios, que están disponibles en forma comercial pero también pueden ser adquirido o comprado bajo contrato gubernamental. Un ejemplo es el software libre con apoyo comercial.
- Point of Presence (PoP): Punto de presencia. Son los servidores, *routers* y *switches*, donde una o más redes establecen un punto de conexión entre sí.
- Surrogates: También llamados servidores CDN o *edge servers*. Son los servidores que sirven copias del contenido a los usuarios finales.

- PlanetLab : Es una red global de investigación, que soporta el desarrollo de nuevos servicios de red. Actualmente cuenta con 1353 nodos en 717 sitios [14].