Trabajo de Fin de Grado

Redes Definidas por Software en Centros de Datos

Software Defined Networks in Data Centers

Akshay Chatani Chatani

La Laguna, 8 de junio de 2020

D. **Jonás Philipp Luke**, con N.I.E. X0581666-L, profesor contratado doctor adscrito al Departamento de Ingeniería Telemática de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

”*Redes Definidas por Software en Centros de Datos*”

ha sido realizada bajo su dirección por D. **Akshay Chatani Chatani**, con N.I.F. 79235229E

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de junio de 2020

Agradecimientos

Deseo expresar mi profundo y sincero agradecimiento a mi tutor de este proyecto, Dr. Jonás Philipp Lüke, por la dedicación y apoyo que ha brindado a este trabajo, por el respeto a mis sugerencias e ideas y por la dirección y el rigor que ha facilitado a las mismas. Gracias por la confianza ofrecida desde que llegué a esta facultad.

Un trabajo de investigación es siempre fruto de ideas, proyectos y esfuerzos previos que corresponden a otras personas. En este caso, a mi tutor. Gracias por su amabilidad y por facilitarme todos los recursos necesarios para poder realizar este proyecto, por su tiempo y por todas sus ideas.

Por su orientación y atención a mis consultas interminables sobre el controlador Ryu y Mininet.

Muchas gracias.

Licencia

© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional.

Resumen

Este proyecto consiste en estudiar las redes definidas por software y desarollar una simulación explicando esta tecnología, conociendo el protocolo Openflow y explicando los casos de uso basados en este paradigma. Estos se centrarán en las problemáticas inherentes a las arquitecturas de red en centros de datos (Google FatTree, DCell), por lo que para ello se simularán algunas topologías de red propias de centros de datos y se evaluarán comparativamente los resultados obtenidos.

Palabras clave: Redes definidas por software, Ryu, Mininet, Centro de datos, Google FatTree, DCell

Abstract

The goal of this project is the study of software defined networks , its development and demonstration explaining this technology, learning more about the Openflow protocol and explaining use cases based on this paradigm. These will focus on the issues faced in data center network architectures (Google FatTree, DCell), and for this purpose some data centre network topologies will be simulated and benchmarked making a comparison based on the results.

Keywords: Software defined networks, Ryu, Mininet, Data centers, Google FatTree, DCell

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En los últimos años se ha producido un enorme crecimiento de Internet, su uso y en general de las aplicaciones de las redes de computadores. Se mueven ingentes cantidades de datos a través de Internet y de las redes privadas de empresas e instituciones. Esto hace que el control y la configuración de los dispositivos de red sean cada vez más complejos.

La gestión de las redes informáticas modernas se ha convertido en un reto complejo que es difícil de manejar, y que lucha por escalar a las necesidades cada vez más demandantes los entornos que conocemos hasta ahora. Las Redes Denidas por Software (RDS) representan un nuevo enfoque que intenta abordar estas debilidades del paradigma actual.

En este proyecto se estudia la aplicación de este nuevo paradigma aplicado a los centros de datos. Se obtiene en primer lugar una visión general sobre las aplicaciones de RDS en centros de datos. Se hace una revisión de las arquitecturas de centros de datos y la posibilidad de basarlas en RDS. A partir de ahí, se han seleccionado dos de ellas y se simulan implementando algunas partes del control basado en RDS. Además, se compara el rendimiento de ambas implementaciones y se estudian sus limitaciones.

1.1. Objetivos

El objetivo de este proyecto es aproximarse a las redes definidas por software y al protocolo Openflow, así como su uso en centros de datos, desarrollando casos de uso basados en este paradigma. Estos se centrarán en las problemáticas inherentes a las arquitecturas de red en centros de datos, por lo que además se simularán algunas topologías de red en centros de datos y se evaluarán comparativamente estas soluciones.

Los objetivos específicos son los siguientes:

- Desarrollar una topología FatTree simulada y parametrizable.
- Desarrollar una topología DCell simulada y parametrizable.
- Desarrollar un programa de control para la topología de FatTree basado en direcciones IP.

- Desarrollar un programa de control para la topología de FatTree basado en direcciones MAC.
- Desarrollar un programa de control para la topología de DCell basado en direcciones MAC.

1.2. Perspectiva histórica

Las principales redes de comunicaciones en todo el mundo en la primera mitad del siglo XX fueron redes de telefonía, en su gran mayoría centralizadas, con grandes volúmenes de servidores y usuarios conectados a grandes centros de conmutación. Como se explica en [1], Paul Baran, un inmigrante polaco que se convirtió en investigador trabajando en Rand Corporation en los Estados Unidos en la década de 1960, argumentó que en caso de ataque enemigo, las redes, como por ejemplo, la red telefónica, eran muy fáciles de dañar. La solución propuesta por Baran era transmitir las señales de voz de las conversaciones telefónicas en paquetes de datos que podrían viajar de forma autónoma a través de la red, encontrando su propio camino hacia su destino. Este concepto se basaba en que si parte del camino que se usa para una conversación dada fuera destruido por un ataque o fallo, la comunicación no se vería interrumpida al tener la capacidad de redirigir automáticamente a un camino alternativo que llevara al mismo destino. Nace así la conmutación de paquetes.

Según expone un artículo de la web [8], a medida que avanzaban los años, comenzaron a surgir los primeros centros de datos. Computadoras como la ENIAC, la Harvard Mark I o la Manchester Small-Scale Experimental Machine eran de gran importancia en la historia de la computación y su nombre está vinculado a estos primeros años en los que se comenzaron a desarrollar los primeros computadores. A los centros que utilizaban este tipo de ordenadores los denominaban centros de cómputo.

Desde estos primeros años, la gestión de las salas técnicas en las que se construían, operaban y se programaban estos primeros computadores supuso también un gran reto tecnológico para estos ingenieros que comenzaban a enfrentarse a problemas operacionales debidos al calor disipado por estos computadores y la necesidad de garantizar el suministro eléctrico para hacerlos funcionar (retos a los que hoy en día nos seguimos enfrentando también). Evidentemente el factor de escala no era el mismo y, durante casi cuatro décadas, una computadora o un mainframe era sinónimo de un gigante de hierro, que era capaz de ocupar una gran habitación.

El IBM S/360 fue el primer ordenador en usar microprogramación, y creó el concepto de microarquitectura. La familia del 360 consistió en 6 ordenadores que podían hacer uso del mismo software y los mismos periféricos. El sistema también hizo popular la computación remota, con terminales conectados a un servidor, por medio de una línea telefónica.

El concepto de estas salas cambió de centro de cómputo a área de informática, destacándola a nivel de dirección dentro de cualquier organización. Surgieron algunos profesionales en informática y con ello la construcción de centros de cómputo con personal más capacitado. Además, se minimizaron costos al usar computadoras más pequeñas

a diferencia de las grandes que requerían de instalaciones costosas y especiales, pero aun así la presencia de los mainframe era ya ineludible en prácticamente todas las esferas de control gubernamental, militar y de la gran industria. Gracias a [8] se sabe que las enormes computadoras de las series CDC, CRAY, Hitachi o IBM por ejemplo, eran capaces de atender a varios cientos de millones de operaciones por segundo.

La nueva arquitectura actual en los centros de datos modernos utiliza una plataforma de hardware que consiste en estaciones de trabajo, microcomputadoras, minicomputadoras y macrocomputadoras. Estos equipos pueden ser actualizados y empleados constantemente para mejorar la calidad de los servicios o implementar nuevos. Internet se basaba en las redes de computadoras como nunca antes en la historia de la tecnología. Por otro lado, durante este período de cambio, la red informática mundial dio lugar a centros de datos cada vez mayores que alojaban sitios web cada vez más complejos, es decir, con mayor suscripción de servicios. Debido a la gran cantidad de estos servidores, se organizaron físicamente en filas de bastidores de servidores con un grado de organización muy alto. Las torres de servidores de cómputo estaban ordenadas jerárquicamente de tal manera que los conmutadores de la parte superior del bastidor (ToR) proporcionaban la red dentro del propio bastidor y entre otros bastidores

Según explica [9], “La mayoría de los conmutadores en estos centros de datos eran ordenadores Unix de uso general que ejecutaban un software que inspeccionaba un paquete que entraba por una interfaz y buscaba la dirección IP de destino en una estructura de datos de búsqueda eficiente, la tabla de enrutamiento. Según la entrada encontrada durante esta búsqueda, el paquete se transmitirá por la interfaz de salida indicada. Los paquetes de control se desviarían a los procesos de control apropiados en el sistema Unix en lugar de procesarse a través de esa tabla de enrutamiento“.

En un principio, para poder aumentar el rendimiento de estos dispositivos, se logró distribuyendo el procesamiento en implementaciones cada vez más paralelas que involucran múltiples hilos, cada uno de los cuales ejecuta microprocesadores de última generación con acceso independiente a una tabla de reenvío distribuido. Sin embargo, a medida que avanzaban los años, la velocidad de las interfaces llegó a un punto en el que la inspección de encabezado y la búsqueda de la tabla de enrutamiento en el software no podían mantenerse eficaces.

La solución fue impulsar una mayor inteligencia en niveles inferiores en la tabla de reenvío de modo que la mayor parte del procesamiento de paquetes posible se pueda realizar a velocidades de línea en el motor de reenvío de hardware.

Los primeros enrutadores solo tenían que realizar modificaciones de campo de encabezado de paquetes IP limitadas. Esto incluyó disminuir el campo Time-to-Live (TTL) e intercambiar el encabezado de MAC a los encabezados de origen-destino de MAC para el próximo salto. En el caso del soporte de multidifusión, tuvieron que replicar el paquete para la transmisión de múltiples puertos de salida. A medida que las capacidades del conmutador aumentaron para admitir características avanzadas como redes de área local virtuales (VLAN) y conmutación de etiquetas multiprotocolo (MPLS), más campos de encabezado de paquetes necesitaron una manipulación cada vez más compleja. Con este fin, surgió la idea de reglas programables, por lo que parte de este procesamiento más

complejo podría codificarse en reglas y llevarse a cabo directamente en los motores de reenvío a velocidad de hardware. Esta misma programabilidad del hardware fue una de las primeras semillas que dio vida al concepto de RDS.

Por otro lado, en cuanto a la implementación a nivel de software se siguen utilizando los protocolos de control habituales (como por ejemplo RIP,BGP o STP), y son sólo una pequeña parte de una larga lista de protocolos que deben implementarse en un conmutador moderno de capa dos o capa tres para que sea un dispositivo completamente funcional para su uso en internet. En las redes de centros de datos a gran escala, según [1], el 30 % de la capacidad del plano de control del conmutador se gasta actualmente en el seguimiento de la topología de la red.

Hay que destacar que en los centros de datos prácticamente no existe tiempo de inactividad que no esté programado previamente. Aunque hay enlaces y nodos en el centro de datos que pueden fallar ocasionalmente, la topología base se aprovisiona centralmente. Los servidores físicos y las máquinas virtuales no aparecen ni se mueven espontáneamente en el centro de datos, sino que más bien, cambian sólo cuando el software de gestión central ordena que lo hagan. A pesar de que la topología del centro de datos es estática y los enlaces son estables, las dificultades del conmutador tradicional se multiplican a escala dependiendo del centro de datos y su tamaño. De hecho, dado que estos protocolos de control son en realidad protocolos distribuidos, cuando algo se cambia intencionalmente, como la adición de un servidor o un enlace de comunicaciones fuera de servicio, estos protocolos distribuidos requieren tiempo para converger en un conjunto de tablas de reenvío consistentes, causando retardos temporales dentro del centro de datos. Por lo tanto, se tiene una situación en la que la mayoría de los cambios de topología se realizan de manera programada e intencional, reduciendo el beneficio de los protocolos autónomos y distribuidos. Por otro lado, el tamaño y complejidad de las redes es mucho mayor de lo que estos protocolos pueden soportar, lo que hace que los tiempos de convergencia sean exageradamente largos.

1.2.1. Planos de datos, control y gestión

Se procede a definir los planos de control, gestión y datos de los conmutadores mencionados anteriormente y de los que se utilizan actualmente, ya que serán mencionados en las siguientes secciones además de aparecer también en la Figura 1.1, con el fin de facilitar la comprensión del proyecto.

- Plano de datos : La mayoría de los paquetes manejados por el conmutador sólo son tocados por el plano de datos. El plano de datos consta de los diversos puertos que se utilizan para la recepción y transmisión de paquetes y una tabla de reenvío con su lógica. Este plano asume la responsabilidad del almacenamiento en búfer de paquetes, la programación de paquetes, el encabezado, la modificación y reenvío. Si la información del encabezado de un paquete de datos que llega se encuentra en la tabla de reenvío, puede estar sujeta a algunas modificaciones en el campo del encabezado y luego se reenviará sin ninguna intervención de los otros dos planos.

- Plano de control : Hay que tener en cuenta que no todos los paquetes pueden

manejarse en el plano de datos, por ejemplo cuando su información aún no se ha ingresado en la tabla o porque pertenecen a un protocolo de control. Todo esto debe ser procesado por el plano de control. Este plano, como se muestra en la Figura 1.1, y que es el encargado de tomar las deciciones de control en la red, está involucrado en muchas otras actividades, como pueden ser, las relacionadas con las estadísticas de la red o su estado.

- Plano de gestión : El tercer plano representado es el plano de gestión. Los administradores de red deben configurar y monitorear el dispositivo a través de este plano, que a su vez extrae información o modifica datos en los planos de control y datos según corresponda. Además, utilizan un sistema de gestión de red para comunicarse con este plano en un dispositivo.

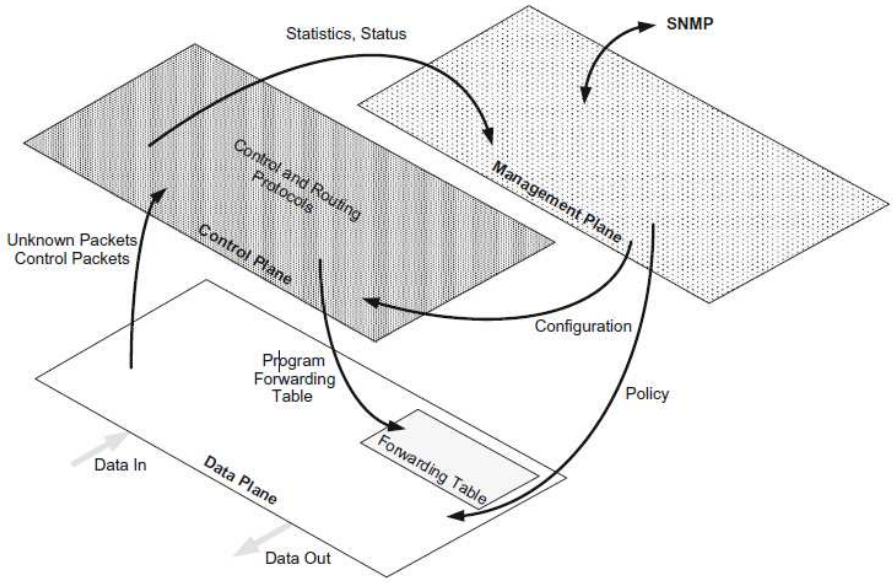


Figura 1.1: Relación entre los planos de datos, de control y de gestión de una red (figura tomada de [1]).

1.2.2. Evoluvión del hardware de red

En los comienzos, casi todo lo que no pertenecía a la capa física (capa uno) se implementó en el software. Sin importar si los dispositivos fueran puentes, conmutadores o enrutadores, el software se usaba ampliamente dentro de estos dispositivos para realizar incluso las tareas más simples, como las decisiones de reenvío a nivel de MAC. Esto se mantuvo incluso durante los primeros días de Internet, que se comercializó a principios de los años noventa.

Con el tiempo estas funciones se movieron del software al hardware. Actualmente vemos la mayoría de las decisiones de reenvío y filtrado implementadas completamente en hardware, como muestra la figura 1.2. Estas decisiones son respuestas que vienen de las tablas configuradas y establecidas por el plano de control anterior. Este movimiento hacia el hardware de la toma de decisiones de nivel inferior ha producido enormes beneficios

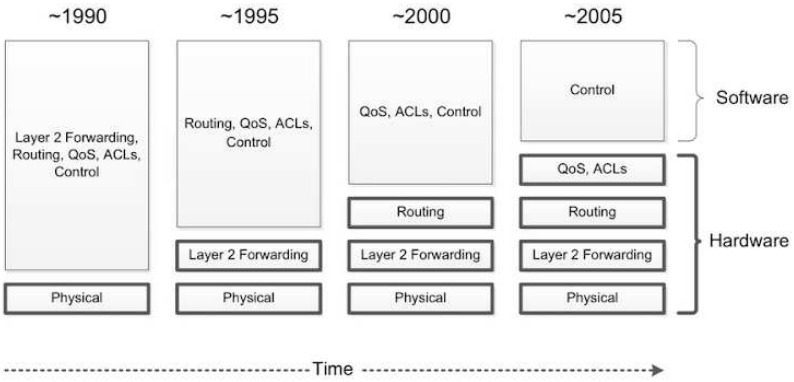


Figura 1.2: Funcionalidad de red que migra al hardware(figura tomada de [1]).

en términos de reducir los costos del dispositivo y aumentar el rendimiento.

Hoy en día, los dispositivos de conmutación se componen normalmente de hardware, como circuitos integrados de aplicación específica (ASIC), dispositivos de puertas lógicas programables (FPGA) y memorias ternales direccionables por contenido (TCAM). La potencia combinada de estos circuitos integrados permite que las decisiones de reenvío se tomen completamente en el hardware a velocidad de línea. Este rendimiento se ha vuelto más crítico a medida que las velocidades de red han aumentado de 1 Gbps a 10 Gbps o 40 Gbps y más. El hardware ahora es capaz de manejar todas las decisiones de reenvío, enrutamiento, lista de control de acceso (ACL) y lo relacionado con QoS. Las funciones de control de nivel superior, responsables de la colaboración de toda la red con otros dispositivos, se implementan en el software.

Dado que el software actual del plano de control en los dispositivos de red carece de la capacidad de distribuir información de políticas sobre temas como seguridad, QoS y ACL, estas características como pueden ser la existencia de un software en el dispositivo que proporcione la funcionalidad del plano de control o la necesidad de aplicar una política en la red, aún deben ser tratadas a través de interfaces de administración.

En definitiva, se presenta una oportunidad para simplificar todos los dispositivos de red y avanzar a la próxima generación de redes y administración.

1.3. Redes definidas por software

Las redes definidas por software son un enfoque arquitectónico de la red que permite a ésta ser controlada de manera inteligente y central, o programada, utilizando aplicaciones de software. Separa la administración de red de la infraestructura de red subyacente, que permite a los administradores ajustar de forma dinámica el flujo de tráfico en toda la red para satisfacer las necesidades cambiantes.

Las RDS buscan reducir la complejidad de redes definidas estadísticamente, automatizar las funciones de red, acelerar la implementación de aplicaciones y servicios y simplificar la implementación y administración de recursos de la red.

Una de las principales características de la RDS es la separación del plano de control explicado anteriormente, que desaparece de los conmutadores, y la automatización y virtualización de la red, que son los temas tratados en las siguientes secciones. Además, también se expondrá la problemática de los dispositivos de conmutación actuales, y las principales limitaciones que tiene la RDS. Y por último, para finalizar el capítulo, se explica el estándar OpenFlow, su funcionamiento y la manera en la que los dispositivos de conmutación hacen uso de este protocolo.

1.3.1. Separación de planos

La primera característica fundamental de RDS es la separación de los planos de reenvío y control. La funcionalidad de reenvío, incluida la lógica y las tablas para elegir cómo tratar los paquetes entrantes en función de características como la dirección MAC, la dirección IP y la ID de VLAN, reside en el plano de reenvío. Las acciones fundamentales realizadas por el plano de reenvío destacan por la forma en que prescinde de los paquetes que llegan. Puede reenviar, descartar, consumir o replicar un paquete entrante. Para el reenvío básico, el dispositivo determina el puerto de salida correcto realizando una búsqueda en la tabla de direcciones. Se puede descartar un paquete debido a condiciones de desbordamiento del búfer o debido a un filtrado específico resultante de una función de limitación de velocidad de QoS, por poner un ejemplo.

La separación entre el plano de datos y de control se consigue por medio de una interfaz de programación de aplicaciones (API) que permite a los controladores RDS ejercer control directo sobre los elementos del planos de datos. El ejemplo más conocido de éste tipo de API es OpenFlow. Los conmutadores que trabajan con OpenFlow tienen una o más tablas, conocidas como tablas de flujo, donde se definen todas las reglas de encaminamiento del dispositivo. Cada una de estas reglas define por un lado a un subconjunto del tráfico y por el otro establece las acciones que deben aplicarse a éste y que se definen en las siguientes secciones.

Esta separación del plano de control permite a los administradores de red aplicar políticas dinámicas desde una perspectiva más alejada y global, y sin la complicación de tener que configurar cada dispositivo de red de forma individual, sino que es el propio controlador quien se encarga de instalar las reglas necesarias en cada uno de éstos de forma automática y dinámica. Se acelera también la capacidad de innovación, ya que el control lógico no está atado al hardware, y aumenta la flexibilidad de la red, dado que permite introducir nuevos servicios más fácilmente.

En definitiva, se produce una simplificación de los dispositivos que son controlados por el controlador, que es un sistema centralizado que ejecuta el software de gestión y control. En lugar de cientos de miles de líneas de software del plano de control que se ejecutan en el dispositivo y permiten que éste se comporte de forma autónoma, ese software se elimina del dispositivo y se coloca en el controlador. Este controlador gestiona la red utilizando políticas de nivel superior. Además, proporciona instrucciones básicas a los dispositivos simplificados cuando es debido para permitirles tomar decisiones rápidas sobre cómo manejar los paquetes entrantes sin tener que pasar otra vez por el controlador.

1.3.2. Automatización y virtualización de la red

La abstracción del reenvío de paquetes permite al programador especificar los comportamientos de reenvío necesarios sin ningún conocimiento del hardware específico del proveedor. Trabajar con la red a través de esta abstracción de configuración es realmente la virtualización de la red en su nivel más básico. Este tipo de virtualización se encuentra en el corazón de la definición de RDS.

El controlador centralizado basado en software en RDS proporciona una interfaz abierta en el controlador para permitir el control automatizado de la red. El controlador también ofrece una API relacionada con las aplicaciones, que permite que se conecten al controlador y, por lo tanto, da la posibilidad para que el software proporcione los algoritmos y protocolos que pueden ejecutar la red de manera eficiente. Estas aplicaciones pueden realizar cambios de red de forma rápida y dinámica a medida que surja la necesidad.

1.3.3. Problemática de dispositivos de conmutación actuales

Los dispositivos de red actuales deben almacenar, administrar y ejecutar el software del plano de control que discutimos en la sección anterior, y que tiene que ser implementado en cada uno de ellos. El resultado de estas demandas en el dispositivo se manifiesta en un mayor costo por dispositivo debido a la potencia de procesamiento requerida para ejecutar ese software avanzado para su funcionamiento, así como la capacidad de almacenamiento para mantenerlo. El problema es que una gran cantidad de software debe ejecutarse en todos y cada uno de los dispositivos de red, con lo cual hay una sobrecarga adicional que proviene del requisito de admitir múltiples versiones de protocolos heredados, así como mantenerse al día con los últimos protocolos definidos por los organismos de estándares.

No obstante, la RDS logra solucionar este problema ya que no es necesario implementar todo ese software en cada uno de los dispositivos. Gracias a las características de este paradigma sólo sería necesario incluir dicho software en el controlador, y éste es el que se encarga de distribuir las reglas necesarias en cada uno de los dispositivos cuando lo requieran.

Por otro lado, a menudo se agregan mejoras a las implementaciones estándar de los conmutadores, que intentan permitir que el producto de un proveedor supere a su competencia. Con muchos proveedores agregando estas mejoras, el resultado final es que cada producto del proveedor tendrá dificultades para interactuar con los productos de otro proveedor. Además los problemas de interoperabilidad y administración a menudo superan con creces las ventajas que se pueden obtener al elegir otro proveedor. Como resultado, los clientes con frecuencia se adhieren con un proveedor que eligieron años o incluso décadas antes.

La manera en que la RDS soluciona esto consiste simplemente en una de las principales características de este paradigma que es la completa programabilidad de la red, y al ser OpenFlow un estándar común para todos, no habría conflicto alguno en cuanto a las mejoras que se vayan incluyendo.

1.3.4. Limitaciones de RDS

- Escalabilidad en el controlador :

  Tener un controlador centralizado significa que la responsabilidad de la organización topológica de la red, la decisión de las rutas óptimas y las respuestas a los cambios deben ser manejadas por el controlador.

  Sin embargo, a medida que se agregan más y más dispositivos de red a la red, surgen preguntas sobre la escalabilidad y la capacidad de un sólo controlador para manejar todos esos dispositivos. Es difícil saber el grado de adecuación de un sistema centralizado en cuanto al manejo de cientos, miles o millones de dispositivos de red y saber exáctamente cuál es la solución cuando la cantidad de dispositivos de red supera la capacidad del controlador para manejarlos. Si intentamos escalar agregando más controladores, ¿cómo se tienen que comunicar y quién organiza la coordinación entre estos controladores?.

- Alta disponibilidad :

  El controlador centralizado no debe ser el único punto débil para la red. Esto implica la necesidad de esquemas de redundancia, como no podía ser de otro manera. Primero, debe haber controladores redundantes de modo que la potencia de procesamiento esté siempre disponible en caso de error de un solo controlador. En segundo lugar, los datos reales utilizados por el conjunto de controladores deben reflejarse de manera que los mismos puedan programar los dispositivos de red de manera coherente. En tercer lugar, las rutas de comunicación a los diversos controladores deben ser redundantes para garantizar que siempre haya una ruta de comunicación funcional entre un conmutador y al menos un controlador.

- Seguridad :

  Tener un controlador centralizado significa que los ataques de seguridad pueden centrarse en ese único punto importante y, por lo tanto, existe la posibilidad de que este tipo de solución sea más vulnerable a los ataques externos o internos que un sistema distribuido. Es importante establecer de antemano qué pasos adicionales se deben tomar para proteger tanto el controlador centralizado como los canales de comunicación entre este y los dispositivos de red.

1.4. Descripción del estándar OpenFlow

OpenFlow es un protocolo que permite a un servidor decirle a los conmutadores de red a dónde enviar paquetes. Se considera uno de los primeros estándares de RDS. Originalmente definió el protocolo de comunicación en entornos RDS que permite que el controlador interactúe directamente con el plano de reenvío de dispositivos de red como conmutadores y enrutadores, tanto físicos como virtuales, para que pueda adaptarse mejor a los cambiantes requisitos empresariales. La especificación OpenFlow delinea tanto el protocolo que se utilizará entre el controlador y el conmutador, así como el comportamiento esperado del conmutador.

La adopción de OpenFlow por parte de las comunidades de investigación y los proveedores de redes marcó un cambio radical en las redes. La especificación OpenFlow llevó a

los proveedores a implementar y habilitar OpenFlow en sus productos de conmutación.

En [7] se indica que OpenFlow comenzó con la publicación de la propuesta original en 2008. Para 2011, OpenFlow había reunido el impulso suficiente para que la responsabilidad del estándar en sí se trasladara a la Open Networking Foundation (ONF). La ONF fue establecida en 2011 por Deutsche Telekom, Facebook, Google, Microsoft, Verizon y Yahoo.

Un aspecto novedoso e interesante de la ONF es que su junta directiva está compuesta por los principales operadores de redes, y no por los vendedores. Por otro lado, la dirección de ONF está protagonizada por directores de tecnología (CTO), directores técnicos y miembros de compañías como Google, Facebook, Deutsche Telekom, Verizon, Microsoft y NTT, entre otros. Esto ayuda a evitar que la ONF se preocupe por los intereses de un importante proveedor de redes dejando de lado a los otros. También ayuda a proporcionar una perspectiva del mundo real sobre lo que debe investigarse y estandarizarse.

1.4.1. Elementos de OpenFlow

A continuación se describen los elementos básicos (extraídos de [10]) que están presentes en una solución OpenFlow:

- El controlador llena los dispositivos conectados a él con entradas de la tabla de flujo. Puede ser estático o dinámico. En el caso de ser estático las entradas son insertadas en el momento en que se crea la topología. Por otro lado, si es dinámico, éstas son insertadas a medida que se generen paquetes. Este proyecto simula ambos ejemplos.

- El conmutador evalúa los paquetes entrantes y encuentra un flujo coincidente, y procede a realizar la acción asociada. Si no se encuentra ninguna coincidencia, el conmutador reenvía el paquete al controlador para obtener instrucciones sobre cómo manejarlo.

  Por lo general, el controlador actualiza el conmutador con nuevas entradas de flujo a medida que se reciban nuevos patrones de paquetes, de modo que el conmutador pueda tratarlos localmente. También es posible que el controlador programe reglas comodín que podrán dirigir muchos flujos a la vez.

  Al conjunto de tablas de flujos y a un canal hacia un controlador lo denominaremos switch OpenFlow. Consta de tres partes:

- Una tabla de flujos que indica cómo debe procesar el switch los flujos que le lleguen.

- Un canal necesario para comunicar el switch con el controlador. Esta comunicación se efectúa por medio del protocolo OpenFlow.

- El protocolo OpenFlow. Consiste en un estándar abierto mediante el cuál se especifican las entradas en la tabla de flujos. OpenFlow es un protocolo de comunicación entre la capa de control y la capa de infraestructura de red.

1.4.2. Funcionamiento de OpenFlow

Los dispositivos de OpenFlow contienen la funcionalidad de reenvío para poder decidir qué hacer con cada paquete entrante. También contienen los datos que impulsan esas decisiones de reenvío. Los datos en sí están representados por los flujos definidos por el controlador. Un flujo describe un conjunto de paquetes transferidos desde un punto origen de la red (o conjunto de puntos orígenes) a otro punto final (o conjunto de puntos finales). Los puntos finales pueden definirse como una dirección IP-puerto TCP / UDP.

Un conjunto de reglas puede describir las acciones de reenvío que el dispositivo debe realizar para todos los paquetes que pertenecen a ese flujo. Un flujo es unidireccional en el sentido de que los paquetes que fluyen entre los mismos dos puntos finales en la dirección opuesta podrían constituir cada uno un flujo separado. Esto es de utilidad, por ejemplo, para los servicios de QoS. Los flujos se representan en un dispositivo como una entrada en la tabla de flujo. Una tabla de flujo reside en el dispositivo de red y consiste en una serie de entradas de flujo y las acciones a realizar cuando un paquete que coincide con ese flujo llega a ese dispositivo. Cuando el dispositivo OpenFlow recibe un paquete, consulta sus tablas de flujo en busca de una coincidencia. Estas tablas de flujo se habían construido previamente cuando el controlador envió las reglas de flujo correspondientes al dispositivo de conmutación. Si el dispositivo encuentra una coincidencia, realiza la acción configurada que tiene memorizada, y que generalmente implica reenviar el paquete. Si no encuentra una coincidencia, el conmutador puede descartar el paquete o pasarlo al controlador, según la versión de OpenFlow y la configuración del conmutador.

```
mininet> sh ovs-ofctl dump-flows hs1
 cookie=0x0, duration=30.500s, table=0, n_packets=1, n_bytes=98, priority=4,dl_dst=20:0
0:00:00:00:01 actions=output:"hs1-eth1"
 cookie=0x0, duration=26.930s, table=0, n_packets=9, n_bytes=882, priority=3,dl_dst=20:
00:00:00:00:00 actions=output:"hs1-eth2"
 cookie=0x0, duration=26.767s, table=0, n_packets=0, n_bytes=0, priority=2,dl_dst=20:00
:00:02:00:00 actions=output:"hs1-eth3"
```

Figura 1.3: Ejemplo de una tabla de flujo.

A modo de ejemplo, en la figura 1.3 se muestra una tabla de flujo. En la segunda regla de esta tabla se puede observar que la entrada ha sido utilizada por 9 paquetes, ya que esos paquetes tenían como destino la MAC que aparece en el campo “dl-dst“ y cuya acción ha sido enviar los paquetes por el puerto 2 del host en el que se encuentra la tabla, que en este caso es el 1.

La aplicación RDS interactúa con el controlador, usándola para establecer flujos proactivos en los dispositivos y para recibir paquetes que se han enviado al controlador. Este tipo de flujo proactivo se conoce como flujo estático. Otro tipo de flujo proactivo podría ocurrir cuando el controlador decide modificar un flujo en función de la carga de tráfico que actualmente se observe a través de un dispositivo de red. Además de los flujos definidos de manera proactiva por la aplicación, algunos flujos se definen en respuesta a un paquete enviado al controlador. Al recibir los paquetes entrantes que se han reenviado al controlador, la aplicación RDS le indicará al controlador cómo responder al paquete y, si corresponde, establecerá nuevos flujos en el dispositivo para permitir que ese dispositivo responda localmente la próxima vez que vea un paquete perteneciente a ese flujo y no pierda tiempo volviendo a consultar al controlador para la misma acción (flujo reactivo).

Con lo mencionado anteriormente, ahora es posible escribir aplicaciones de software que implementen funciones de reenvío, enrutamiento, superposición, ruta múltiple y control de acceso, entre otras.

1.4.3. Dispositivos OpenFlow

Un switch OpenFlow puede tener diferentes tablas de flujos, en las cuales se realiza la búsqueda de coincidencia con los paquetes entrantes. Cada entrada en una tabla de flujos consta de tres campos :

- Cabecera : define el flujo.
- Contadores : contabilizan la coincidencia de paquetes. Se actualizan por flujo, por tabla, por puerto y por cola.
- Acciones : a realizar cuando se encuentra una coincidencia entre un paquete y una tabla de flujos.

Cuando un paquete entra en un switch OpenFlow, se examina comenzando por la primera tabla en busca de coincidencias. Si las hay, se añadirá la acción definida en la tabla en el conjunto de acciones, y se pasará a evaluar el paquete en la siguiente tabla. Cuando se evalúan todas las tablas, se ejecutará el contenido del conjunto de acciones. En caso de no hallar coincidencias en las tablas de flujo, el paquete será devuelto al controlador, que indicará al switch qué hacer con él. Cada entrada en la tabla de flujos tiene una acción asignada. Hay tres tipos de acciones básicas que todos los tipos de switches OpenFlow deben soportar. Estos tres tipos son:

- Reenviar el paquete a través de un puerto. Esta acción permite enviar los paquetes a través de la red.
- Encapsular y reenviar el paquete al controlador. Típicamente es la acción que se ejecutará al recibir un paquete de un nuevo flujo. De esta manera, el controlador puede decidir si instaurar una nueva entrada en una tabla de flujos y una acción asociada, o si procesar de alguna otra manera el paquete.
- Eliminar el paquete. De esta forma, el controlador puede actuar como un firewall, y bloquear paquetes sospechosos, por ejemplo.

A continuación se explican dos tipos diferentes de mensajes del protocolo OpenFlow:

Mensajes del controlador al switch: lo inicia el controlador, y su función es conocer y actuar sobre el estado del switch. Se envía una petición de características al switch, y éste debe responder con las capacidades de las que dispone. Dentro de este mensaje se pueden distinguir cuatro tipos:

- Modify-State: su principal propósito es añadir y eliminar entradas en las tablas de flujo y fijar las características de los puertos.
- Read-State: interroga al switch sobre estadísticas del tráfico (contadores de las tablas de flujo).

- Send-Packet: envía paquetes por un puerto específico en el switch.
- Barrier request o reply: usado para recibir notificaciones para operaciones completadas.

Mensajes asíncronos : los switches envían mensajes al controlador a la llegada de un paquete, tras un error o tras un cambio de estado. Existen tres tipos:

- Packet-in: enviado al controlador al recibir cualquier paquete que no tenga coincidencias en las tablas de flujo, o si la acción que corresponde a la entrada coincidente es reenviar el paquete al controlador.
- Flow-modify: se envía cuando una entrada es añadida a la tabla de flujos de un switch.
- Port-status: se envía al cambiar el estado de un puerto determinado.

Implementaciones de dispositivos OpenFlow existentes

Gracias a [11], se conoce que actualmente se encuentran disponibles varias implementaciones de dispositivos OpenFlow, tanto comerciales como de código abierto, como por ejemplo, Open vSwitch (OVS) o Big Switch. Los fabricantes de equipos de red existentes, como Cisco, HP, NEC, IBM, Juniper y Extreme, han agregado soporte OpenFlow a algunos de sus conmutadores de generaciones modernas.

En general, estos conmutadores pueden funcionar tanto en modo tradicional como en modo OpenFlow. También hay una nueva clase de dispositivos llamados dispositivos de caja blanca, que son minimalistas ya que están construidos principalmente a partir de chips de conmutación comerciales y una CPU y memoria de consumo. Estos dispositivos pueden estar habilitados para OpenFlow a un costo mucho menor.

1.5. Metodología y herramientas

La metodología utilizada consiste, en primer lugar, en analizar y adaptar cada topología de red a las limitaciones de la herramienta Mininet. En segundo lugar, ir desarrollando de menor a mayor nivel de escalabilidad la topología correspondiente. Y, por último, elaborar el programa de control haciendo uso del controlador Ryu para dicha topología, es decir, desarrollando primero las reglas que deben ir en las tablas de flujo comenzando por el menor nivel de escalabilidad hasta llegar al mayor.

Las principales herramientas utilizadas para el desarrollo han sido tres:

- Mininet : es un emulador de redes, capaz de configurar el entorno de nuestro sistema operativo para que sea prácticamente idéntico al de la topología que se le haya especificado. Esta tecnología permite crear máquinas virtuales muy ligeras dentro de nuestro sistema, ejecutando el mismo software que en la realidad y ofreciendo un aislamiento completo entre dispositivos, tanto a nivel de red como a nivel de procesos. Cada componente de nuestra topología creado por Mininet, dispone de su propio entorno de red como si de una máquina virtual se tratase y la interacción entre el software que se ejecutacon el sistema es exactamente la misma que en un entorno real.

- Controlador Ryu : es un software basado en componentes que define un entorno de red. Ryu ofrece componentes de software con una API que hacen que sea fácil para los desarrolladores crear nuevas aplicaciones de gestión y control de la red. Ryu es compatible con varios protocolos para la gestión de dispositivos de red. En nuestro caso el que nos interesa es OpenFlow. Todo el código está disponible gratuitamente bajo la licencia Apache 2.0.
- Python : es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad.

Capítulo 2

Desarrollo e implementación

En este capítulo se describen las arquitecturas de prueba utilizadas en este proyecto y las técnicas de enrutamiento basadas en OpenFlow que se implementaron para las mismas. Primero se hace un breve repaso por distintas arquitecturas de red para los centros de datos que se pueden encontrar en la literatura. Seguidamente se seleccionan dos de ellas (FatTree y DCell) y finalmente se explican las estrategias de enrutamiento que se han implementado.

2.1. Revisón de arquitecturas de red para data center

La elección de estas ideas para la implementación del proyecto surge de un análisis previo de las diferentes arquitecturas de red utilizadas por las compañías internacionales mencionadas anteriormente y sus casos de éxito, utilizando como base para ello las referencias en [7], [11] y [12].

A continuación se describen todas las arquitecturas que fueron, en un principio, candidatas a ser simuladas en Mininet:

- JellyFish : es una topología de red orientada a la flexibilidad y al alto ancho de banda que consta de n puertos en cada conmutador. Cada uno tiene puertos conectados a otros switches y el resto de los puertos conectados a los hosts. Los enlaces se agregan conectando aleatoriamente un par de conmutadores que aún no están conectados (es decir, no son vecinos) y que tienen al menos un puerto libre.

- BCube : esta arquitectura de red adopta un enfoque centrado en el servidor para producir un centro de datos modular utilizando conmutadores muy básicos. Funciona con miniconmutadores de gama baja. Hay dos tipos de dispositivos en BCube: servidores con múltiples puertos y conmutadores que se conectan a un número constante de servidores. Es una estructura definida recursivamente, simplemente siendo n servidores conectados a un conmutador de n puertos.

- Google FatTree : es una topología de red universal para una comunicación eficiente. Consiste en una estructura de árbol en la que cada rama tiene el mismo grosor, independientemente de su lugar en la jerarquía, y todas tienen poco ancho de banda. En un árbol grande, las ramas más cercanas a la parte superior de la jerarquía tienen más ancho de banda que las ramas más bajas de la jerarquía.

- Scafida : es una topología de red asimétrica para un centro de datos sin escala. Su especialidad es operar en distancia corta, tiene tolerancia alta a los errores y una

construcción incremental. Las redes sin escala tienen dos propiedades importantes: diámetro pequeño y alta resistencia a fallos aleatorios. Scafida consiste en un conjunto heterogéneo de conmutadores y hosts en términos de número de puertos / enlaces / interfaces.

- DCell : es una arquitectura híbrida centrada en el servidor, y en donde éste está conectado directamente a muchos otros servidores. Un servidor en un DCell está equipado con múltiples tarjetas de interfaz de red. La DCell sigue una jerarquía recursiva.

Las topologías JellyFish, BCube y Scafida mencionadas anteriormente se ilustran en la figura 2.1. Por su parte, las arquitecturas FatTree y DCell se van a describir más a fondo en la sección 2.2

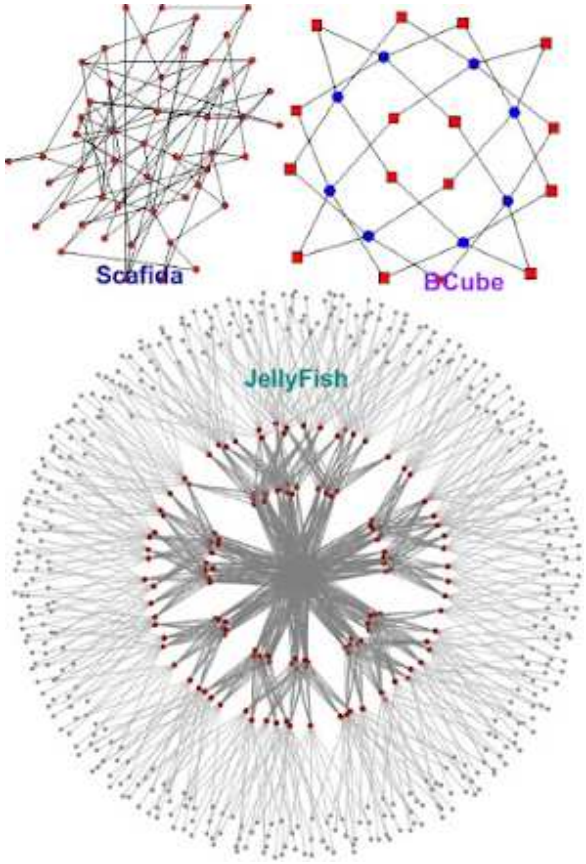


Figura 2.1: Otras arquitecturas analizadas(figura tomada de [2] y [3]).

2.2. Topologías de prueba

Teniendo en cuenta la dificultad para su simulación, la utilidad, escalabilidad y eficacia de cada una de ellas, se decidió finalmente optar por la DCell y Google FatTree.

Como la idea era implementar los programas de control para las mismas debemos ser capaces de ejecutar una serie de pruebas y evaluar empíricamente su rendimiento. Por ello, ambas arquitecturas se programaron en Mininet. Esta herramienta permite agregar de manera sencilla conmutadores y hosts para crear la topología de red deseada y a su

vez, fijar las direcciones IP e identificadores necesarios en cada uno de los dispositivos para su posterior manipulación. En este caso concreto se ha utilizado dicha herramienta y sus correspondientes librerías en el lenguaje de programación Python.

2.2.1. FatTree

Es una de las arquitecturas más utilizadas hoy en día y consiste en tres niveles de conmutadores. Un diseño de este tipo, como se muestra en la figura 2.2, tiene una serie de dispositivos de conmutación a nivel de core, aún más de los mismos a nivel de distribución, y por último, en mayor número en el nivel de acceso.

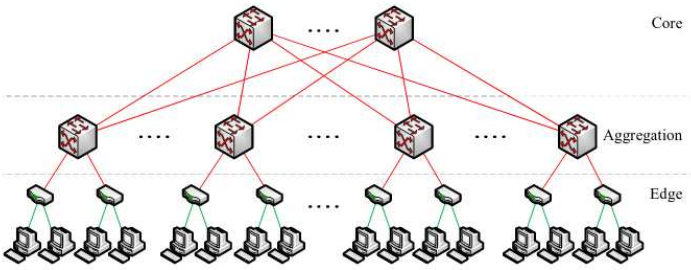


Figura 2.2: Topología convencional de un centro de datos(figura tomada de [4]).

Google implementó una modificación de la topología FatTree para interconectar conmutadores Ethernet básicos y poder producir grandes centros de datos escalables. La topología consiste en conmutadores de k puertos junto con nodos de computación básicos en las hojas del árbol como se muestra en la figura 2.3. El bloque de construcción básico del centro de datos se denomina pod. Google Fat Tree consta de k pods, cada uno con dos capas de $\frac{k}{2}$ conmutadores. Cada conmutador de k puertos en la capa inferior está directamente conectado a $\frac{k}{2}$ hosts. Cada uno de los puertos $\frac{k}{2}$ restantes está conectado a $\frac{k}{2}$ de los k puertos en la capa de agregación de la jerarquía. Hay $\frac{k}{2}$ conmutadores principales de $2k$ puertos. Cada conmutador de núcleo tiene un puerto conectado a cada uno de los k pods. Esta es también la configuración que se utiliza en este proyecto.

En el desarrollo de la topología simulada en Mininet, se ha creado esta arquitectura parametrizando los valores de cada uno de los niveles, así como de los hosts a poder conectar. Esto quiere decir que podemos escalar esta topología añadiendo el número de hosts que se desee en cada uno de los conmutadores del nivel de acceso. También es posible añadir más conmutadores de acceso, en vez de tener sólo los dos que aparecen en la figura 2.3. Por último también es posible aumentar los dispositivos del nivel de distribución y así poder aumentar el número total de dispositivos en la topología con el objetivo de acercarnos lo máximo posible a simular un centro de datos.

Se trabaja con tres tipos de conmutadores. El primero, utilizado en el nivel de acceso, es un conmutador con enlaces a 100 Mbps. Para niveles superiores como el de distribución, utilizamos conmutadores con enlaces a 500 Mbps. Y por último, para el nivel de

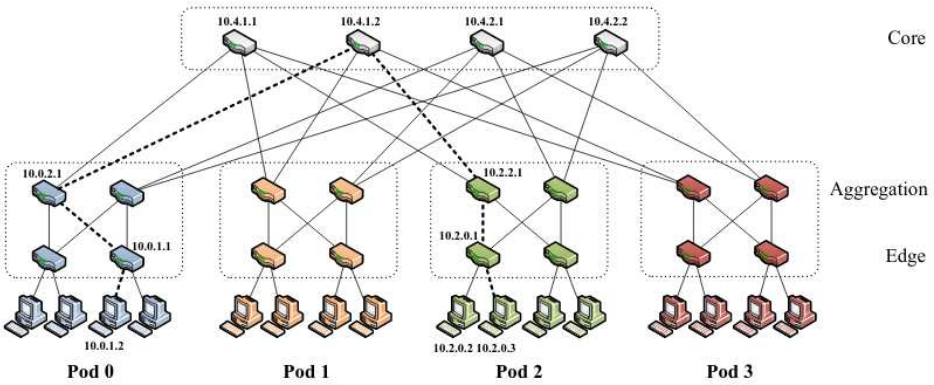


Figura 2.3: Topología FatTree de Google(figura tomada de [5]).

core, los mismos a 1 Gbps. [1]

Mininet permite crear las topologías de red mediante una API en Python que permite añadir los componentes de la misma: hosts, conmutadores y enlaces, además del controlador OpenFlow. De esta manera, se utiliza un programa en este lenguaje para generar la topología que sigue los siguientes pasos:

1. Se comienza creando los conmutadores para el nivel de distribución, identificando y asignándole a cada uno de ellos un ID único. Estos dispositivos se agregan a una lista de switches de su propio nivel.

2. A continuación se agregan los conmutadores necesarios en el nivel de enlace y se hace el mismo proceso que con los anteriores. Se añaden los ordenadores deseados para cada uno de estos conmutadores con la dirección IP correspondiente para poder conocer en todo momento la ubicación exacta del host que se está utilizando. El esquema de direccionamiento IP utilizado se muestra en la figura 2.4.

   Una alternativa a esta localización a través de direcciones IP es el uso de direcciones MAC virtuales (véase la sección 2.3), que consiste en asignar a cada host una MAC virtual que permite localizar el dispositivo dentro de la topología. Esta MAC es asignada temporalmente durante el viaje del paquete. Para facilitar para enrutarlo hacia su destino y una vez que llega al último switch de acceso se vuelve a cambiar la MAC virtual por la real del dispositivo. La figura 2.4 muestra el formato seguido para las direcciones MAC de esta segunda alternativa. A continuación se explican detalladamente estos dos formatos:

   - Direcciones IP : El primer byte es 10 para indicar que se trata de una red privada, el segundo indica el pod en el que se encuentra, el tercero el conmutador de nivel de enlace correspondiente, y el último la posición del host dentro de ese dispositivo. La máscara de red se establece en /8 para que no se necesite enrutar a otra red, es decir, consideramos la topología completa como una única red.
   - Direcciones MAC : Los 8 primeros bits de la MAC se dejan a 00. Los 8 siguientes son utilizados para conocer en qué pod se encuentra ese dispositivo. Los bits 24

[1] Mininet no permite velocidades superiores a 1 Gbps.

Figura 2.4: Esquema de direccionamiento para FatTree

al 32 nos dan la información del conmutador de nivel de acceso que lo conecta, y por último, los 8 últimos bits revelan la posición en la que se encuentra ese host dentro de ese conmutador.

3. Se comienza el cableado de red para los niveles de distribución y acceso. Esto se realiza recorriendo cada uno de los conmutadores del nivel de acceso y realizando una conexión con todos los conmutadores de nivel de distribución para ese dispositivo. Con esto se pretende conseguir un esquema de conexiones controlado y organizado.

4. Se utiliza una función que se ha desarrollado y que tiene como objetivo realizar las conexiones desde el nivel de distribución al nivel de core. En este caso, dependiendo con qué conmutador estemos tratando, se realiza la conexión de ese dispositivo con los dos conmutadores de core más próximos al mismo. Cabe destacar que el número de puertos 1 y 2 ha sido reservado de antemano para estas conexiones. De esta manera se tiene un control absoluto sobre la topología para poder realizar un enrutamiento correcto.

   Una iteración de todo lo mencionado anteriormente construye un pod de la topología FatTree.

5. El último paso ha sido la realización del conexionado entre los hosts y los conmutadores del nivel de acceso. Esto se deja para el final con el objetivo de tener todos los ordenadores o dispositivos en las últimas posiciones de los puertos de los conmutadores de ese nivel. Para realizar esta operación se recorren todos los conmutadores del nivel de acceso y se establece una conexión con el número de hosts deseados recorriendo a la par la lista que almacena a todos los hosts de la red.

2.2.2. DCell

En [7] se puede encontrar una descripción minunciosa de la arquitectura DCell. Se trata de una arquitectura híbrida en la cual el protagonista es el servidor, y donde éste mismo está conectado directamente a muchos otros servidores, en vez de a un conmutador que los interconecte. Un servidor en un DCell está fabricado con múltiples tarjetas de interfaz de red (NIC).

La DCell sigue una jerarquía de celdas de construcción recursiva como se muestra en la Figura 2.5. Una celda de nivel 0 es la unidad básica y el bloque de construcción de la topología de DCell, que está organizada en los niveles que se deseen, y en la cuál una celda de nivel superior contiene múltiples celdas del nivel justamente inferior a ese. Una celda de nivel 0 contiene n servidores y un conmutador de red básico. El conmutador de red solo se usa para conectar el servidor dentro de una misma celda de nivel 0, junto con otros servidores que lo acompañan.

Del mismo modo, una celda de nivel 1 contiene $k = n + 1$ celdas de nivel 0 y, de manera similar, una celda de nivel 2 contiene $k \times n + 1$ celdas de nivel 1. Un DCell puede construirse de forma recursiva obteniendo como resultado más de 3.26 millones de servidores. ($k = 3$, $n = 6$). La construcción en una DCell sigue un enfoque de divide y vencerás. Para que los paquetes lleguen de un host de origen al host de destino, deben pasar por la DCell de origen al ancestro común, un enlace que conecta los DCell de nivel anterior y, finalmente, al destino. En general, DCell es una topología altamente escalable y tolerante a fallos.

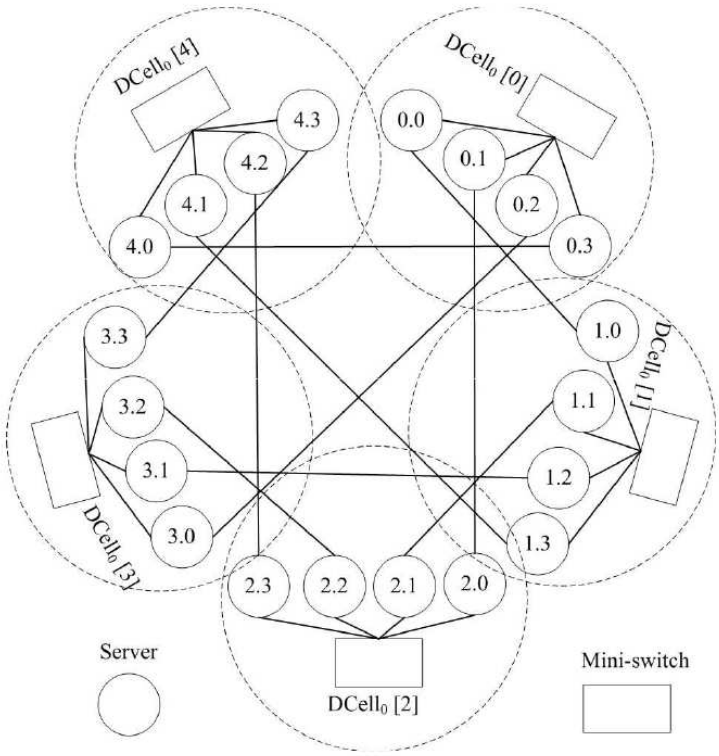


Figura 2.5: Topología DCell de nivel 1 con n=4(figura tomada de [6]).

En la figura 2.5 se muestra un ejemplo de una red DCell-1 cuando $n = 4$. Se compone de 5 redes DCell-0. Cuando consideramos cada DCell-0 como un nodo virtual, estos nodos virtuales forman un grafo completo. En la figura 2.2, cada servidor tiene dos enlaces en DCell-1. Uno se conecta a su mini conmutador, por lo tanto, a otros nodos dentro de su propio DCell0. El otro se conecta a un servidor en otro DCell-0. En una DCell-1, cada DCell-0, si se trata como un nodo virtual, está completamente conectado con todos los demás nodos virtuales para formar un grafo completo. Además, dado que cada DCell-0

tiene n enlaces entre DCell-0, un DCell-1 sólo puede tener $n+1$ DCell-0, como se ilustra en la figura.

Un DCell de alto nivel se construye a partir de un DCell del nivel justamente inferior, es decir usamos DCell-$(l-1)$ para construir una DCell-l, siendo $l > 0$. En la figura 2.6 se muestra el pseudocódigo para contruir una DCell-l. Cada DCell-l tiene t_{l-1} servidores y contiene g_l DCells de nivel inferior $(l-1)$. De esta manera, tenemos que:

$$g_l = t_{l-1} + 1$$
$$t_l = g_l \times t_{k-1}$$

```
/* pref is the network prefix of DCell_l
   l stands for the level of DCell_l
   n is the number of nodes in a DCell_0*/
BuildDCells(pref, n, l)
Part_I:
   if (l == 0) /*build DCell_0*/
      for (int i = 0; i < n; i++)
           connect node [pref, i] to its switch;
      return;
Part_II:
   for (int i = 0, i < g_l; i++) /*build the DCell_{l-1}s*/
      BuildDCells([pref, i], n, l - 1);
Part_III:
   for (int i = 0, i < t_{l-1}; i++) /*connect the DCell_{l-1}s*/
       for (int j = i + 1, j < g_l; j++)
           uid_1 = j - 1; uid_2 = i;
           n_1 = [pref, i, uid_1]; n_2 = [pref, j, uid_2];
           connect n_1 and n_2;
   return;
```

Figura 2.6: Pseudocódigo de construcción de una DCell de nivel k(figura tomada de [7]).

Este es el procedimeinto que se ha utilizado para construir una red DCell de nivel k en Mininet. Tiene tres partes:

- La parte I se aplica para construir un DCell-0. Simplemente conecta n nodos al conmutador correspondiente y se termina la recursion.
- La Parte II construye recursivamente el número g_l de DCell $l-1$.
- La Parte III interconecta estos DCell-$l-1$ donde dos DCell $l-1$ están conectados con un enlace.

Cada servidor en una red DCell-k tiene $k+1$ enlaces. El primer enlace, llamado enlace de nivel 0, se conecta a un conmutador que interconecta DCell-0. El segundo enlace, un enlace de nivel 1, se conecta a un nodo en el mismo DCell de nivel 1, pero en un DCell de nivel 0 diferente. Del mismo modo, el enlace de nivel l se conecta a un DCell-$l-1$

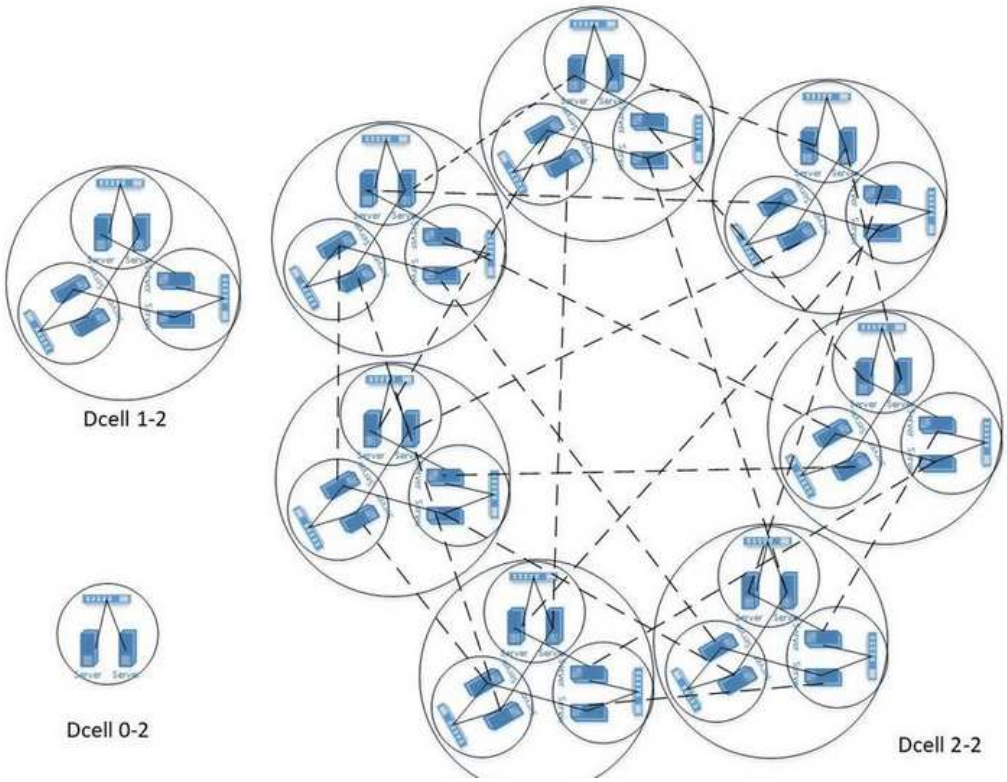


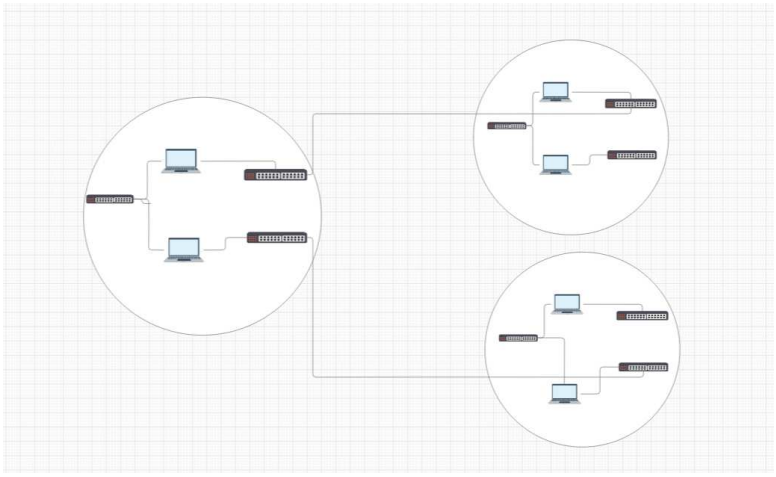
Figura 2.7: DCell con n=2 ; hasta el nivel 2(figura tomada de [6]).

diferente dentro del mismo DCell l.

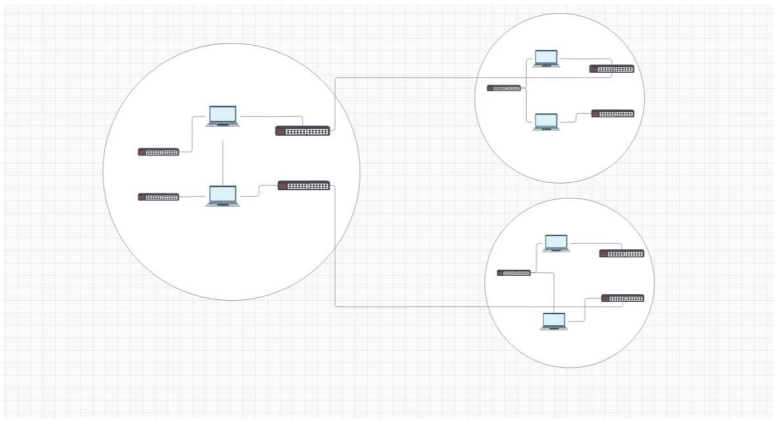
En la figura 2.7 se puede observar cómo sería la topología de una DCell de $n = 2$ y de los niveles 0 a 2. En cambio, para poder representar esta topología en Mininet y poder realizar el correspondiente programa en el controlador Ryu utilizando OpenFlow, se han considerado tres posibles soluciones que son mostradas en la figura 2.8 y explicadas a continuación:

- En la primera propuesta (figura 2.8a), los conmutadores de interconexión entre los hosts son los encargados de tener una conexión con cada uno de los hosts, y éste mismo las conexiones con las DCells del exterior.
- En la segunda propuesta (figura 2.8b), se intenta no tener más de una conexión con el host para evitar tener que disponer de varias tarjetas de red y por tanto, mayor número de direcciones IP, con lo cúal se decide interconectar los switches de una misma DCell manteniendo un conmutador y un host individual para cada uno.
- En la propuesta número tres (figura 2.8c) se intenta mejorar la anterior, aumentando el número de conmutadores, pero consiguiendo así una mayor integridad, lógica y separación de cada una de las celdas con el fin de poder aprovechar al máximo el protocolo OpenFlow y sus ventajas.

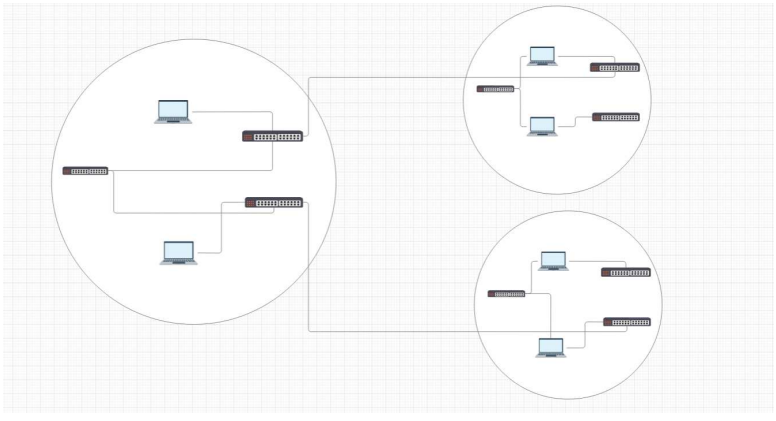
Todas las propuestas fueron diseñadas con el fin de poder introducir reglas en las tablas de flujo a través del controlador Ryu en cada uno de los conmutadores, y conseguir de esta manera, seguir enfocándo el proyecto a las redes definidas por software utilizando OpenFlow.



(a)



(b)



(c)

Figura 2.8: Propuestas para el desarrollo de la topología DCell. (a) Propuesta uno. (b) Propuesta dos. (c) Propuesta tres.

Finalmente se optó por la última opción (propuesta tres), mostrada en la Figura 2.8c por un factor muy importante, que es evitar tener que realizar el direccionamiento IP de cada uno de los hosts, de cada conmutador, y reducir el número de tarjetas de red en los mimos. Además de por motivos de coherencia y cohesión a la hora de realizar el enrutamiento con Ryu y para poder realizar pruebas más interesantes aprovechando todas las características de OpenFlow utilizando para ello los conmutadores vinculados a cada uno de los hosts.

Nuevamente, se optó por utilizar un enrutamiento basado en direcciones MAC virtuales (véase la sección 2.3). El esquema de direccionamiento que se ha utilizado para la DCell es el mostrado en la figura 2.9. Esto permite que se pueda tratar a la red como una red plana a nivel de IP.

Figura 2.9: Esquema de direccionamiento para la DCell

2.3. Direcciones MAC virtuales

En las secciones anteriores se ha indicado el uso de un enrutamiento basado en direcciones MAC virutales. Esto consiste en que el conmutador de acceso correspondiente sustituye la dirección MAC de origen de la trama por la dirección MAC virtual con el formato que se indicó anteriormente para cada caso. Esto es sencillo de realizar mediante una entrada en la tabla de flujos del conmutador OpenFlow. Por otra parte requiere la implementación de un mecanismo que permita tratar las consultas ARP para que las respuestas correspondan a la MAC virtual asociada al host correspondiente, ya que ésta codifica la localización de éste dentro de la topología. Esto último se resuelve mediante un programa de control adicional de manera que cuando un host hace una consulta ARP sea el controlador el que proporcione la respuesta a la misma indicando la MAC virtual correspondiente. Esto, además es una necesidad, puesto que en una red de las dimensiones de un centro de datos, es preciso mantener acotado el tráfico broadcast. Este aspecto no se ha abordado en este proyecto, para centrarlo en el enrutamiento y por falta de tiempo. En las implementaciones que se presentan aquí se dejó el trabajo de la resolución de ARP al simulador Mininet (flag `-arp`). En un caso real habría que hacer una implementación en un programa de control de Ryu.

2.4. Programas de control

En esta sección se van a describir las estrategias de enrutamiento que se han implementado utilizando el controlador Ryu para los tres casos que se han mencionado anteriormente:

- FatTree con enrutamiento basado en direcciones IP.

- FatTree con enrutamiento basado en direcciones MAC virtuales.
- DCell con enrutamiento basado en direcciones MAC virtuales.

2.4.1. Enrutamiento en FatTree basado en direcciones IP

Para esta arquitectura se realiza una inserción proactiva en las tablas de flujo de los distintos dispositivos en el momento de conectarse cada dispositivo por primera vez al controlador.

A continuación se explica la idea del código desarrollado para dicha implementación en el controlador Ryu para esta topología :

En primer lugar, se ha creado un diccionario con todos los IDs de todos los conmutadores, con la información correspondiente de su nivel (core, distribución o enlace) ya que cada uno de ellos tendrá unas reglas específicas para ese tipo de conmutador. Además tendrá la información correspondiente al pod en el que se encuentra, y su posición dentro del mismo.

Cuando entra un nuevo conmutador a la topología y es conocido por el controlador, se busca si su ID coincide con alguno de los que se tiene en el diccionario, y si es así, dependiendo de su nivel, se llama a la función correspondiente para que se inserten las reglas necesarias en ese dispositivo para que pueda procesar los paquetes entrantes y salientes. Esta reglas dependen del tipo de dispotitivo (acceso, agregación o core):

- En los conmutadores de acceso hay tres tipos de reglas:
  - Para enviar un paquete a un host directamente conectado. En este caso, el match se hace con la dirección IP completa y la acción corresponde a enviar el paquete por el puerto en el que se encuentra ese host.
  - Para enviar un paquete a un host dentro del mismo pod, pero en distinta subred, se inserta una regla que utiliza como patrón el prefijo de la subred (p.e. 10.pod.subred.0/24) y la acción consiste en enviar el paquete por el puerto correspondiente a la subred de destino.
  - Para enviar un paquete hacia otro pod distinto, se insertan reglas que hacen el match con el prefijo del pod (10.pod.0.0/16) y envían el paquete al conmutador del nivel de agregación.
- Si se trata de un conmutador de agregación, se insertan dos tipos de reglas:
  - Para los paquetes que van a otra subred dentro del mismo pod, se inserta una regla que hace match sobre el prefijo de subred (10.pod.subred.0/24) y lo envía al switch de acceso que corresponda.
  - Para los paquetes que van a otro pod se envía el paquete al switch de core en función de si la dirección IP es par o impar, con el fin de balancear la carga.
- En los conmutadores de core se inserta un único tipo de regla, que hace match en función de prefijo del pod (10.pod.0.0/16) y envían el paquete por el enlace hacia el pod correspondiente.

2.4.2. Enrutamiento en FatTree basado en direcciones MAC

Por otro lado, en el programa de enrutamiento a nivel de MAC, se ha creado un sistema de MAC virtual que, dependiendo del puerto por el que entra un paquete y la ID del switch, se cambia la MAC de origen temporalmente de ese paquete con la que debería de tener según su posición en la topología. Esto permite no tener que fijar las MAC manualmente de cada uno de los hosts, y da una visión más real que se asemeja a lo que ocurre en los centros de datos. Lo mismo ocurre con los paquetes de salida, pero esta vez se cambia la MAC de destino para que pueda llegar al host adecuado.

Como la MAC virtual contiene los mismos datos (pod, subred, host) que las IP en el caso del enrutamiento basado en IP, se procede de manera similar, pero haciendo los match parciales sobre las direcciones MAC de la trama en vez de realizarlo sobre las direcciones IP. Esto permite considerar la red como una red plana a nivel 3. Por este motivo, en los hosts se configuran las IP con una máscara /8.

2.4.3. Enrutamiento en DCell

El enrutamiento en una DCell no puede utilizar un esquema de enrutamiento basado en estado de enlace, ya que el objetivo de la estructura de DCell es interconectar hasta millones de servidores. El OSPF jerárquico tampoco sería una solución adecuada, ya que necesita un área troncal para interconectar todas las demás áreas. Esto crea un cuello de botella en el ancho de banda y un fallo en un solo punto crítico de la topología.

En [13], se expone un esquema enrutamiento básico y otro tolerante a fallos para una DCell. Es una solución de enrutamiento descentralizada y muy cercana a ser óptima, que explota efectivamente la estructura DCell y puede manejar de manera controlada y eficaz varios fallos potenciales (que puedan ocurrir debido al hardware, software o alimentación), u otro tipo de problemas que son comunes en los centros de datos, según explican en [14].

El esquema básico, consiste en un algoritmo de enrutamiento de ruta única simple y eficiente para unidifusión al explotar la estructura recursiva de DCell. Se denomina DCellRouting, y se muestra en la Figura 2.10. El diseño de DCellRouting sigue un enfoque de divide y vencerás. Se consideran dos nodos src y dst que están en el mismo DCell-k pero en dos DCell-$k-1$ diferentes. Al calcular la ruta de src a dst en un DCell-k, primero calculamos el enlace intermedio ($n1$, $n2$) que interconecta los dos DCell-$k-1$. El enrutamiento se divide en cómo encontrar las dos sub-rutas de src a $n1$ y de $n2$ a dst. La ruta final de DCellRouting es la combinación de las dos sub-rutas y ($n1$, $n2$).

En la Figura 2.10, la función `GetCommonPrefix` devuelve el prefijo común de `src` y `dst` y `GetLink` calcula el enlace que interconecta los dos sub-DCells. El enlace puede derivarse directamente de los índices de los dos sub-DCells. Sean s_{k-m} y d_{k-m}, los índices de los dos sub-DCells. Basándose en la función `BuildDCells` de la figura 2.5, el enlace que interconecta estos dos sub-DCells es $[sk-m, dk-m-1]$ y $[dk-m, sk-m]$.

Este pseudocódigo, descrito en el artículo mencionado anteriormente, permite saber a qué hosts se debe realizar el siguiente salto en la DCell para alcanzar el destino deseado. Ésta es la aproximación de ayuda que se ha utilizado para crear el programa de control en Ryu para dicha arquitectura, ya que lo que se necesita averiguar exactamente es,

```
/* src and dst are denoted using the (k + 1)-tuples
   src = [s_k, s_{k-1}, ..., s_{k-m+1}, s_{k-m}, ..., s_0]
   dst = [d_k, d_{k-1}, ..., d_{k-m+1}, d_{k-m}, ..., d_0]*/
DCellRouting(src, dst)
    pref =GetCommPrefix(src, dst);
    m = len(pref);
    if (m == k) /*in the same DCell_0*/
       return (src, dst);
    (n1, n2) = GetLink(pref, s_{k-m}, d_{k-m});
    path1 = DCellRouting(src, n1);
    path2 = DCellRouting(n2, dst);
    return path1 + (n1, n2) + path2;
```

Figura 2.10: Pseudocódigo para enrutamiento en una DCell-k(figura tomada de [6]).

dependiendo de la cabecera de cada paquete, a qué puerto de salida se debe dirigir éste en cada uno de los dispositivos.

Para solucionar este problema se ha desarrollado un ćodigo que se basa en la detecciones de patrones encontrados en las conexiones de los dispositivos de conmutación para poder fijar a qué puerto de salida debe dirigirse un paquete, teniendo en cuenta el origen y el destino.

Cabe destacar que, en este caso, se ha elaborado un mecanismo dinámico de inspección de paquetes y envío de reglas para las tablas de flujo, al contrario que en la topología anterior de Google FatTree, en la que se desarrolló de forma estática.

Esto significa que a medida que vayan apareciendo paquetes en la red, se inspeccionan sus cabeceras y se realiza una decisión en el controlador en ese mismo momento, es decir, las reglas no son insertadas al comienzo de la creación de la DCell, sino a medida que vaya siendo necesario.

Cada uno de los hosts tiene una dirección MAC correspondiente a su posición en la arquitectura. Para ello se ha utilizado el esquema de la figura 2.9. De esta manera se conoce exactamente la posición de cada uno de los servidores en la arquitectura. Dependiendo del caso y del tipo de dispositivos se analizan diferentes campos de la dirección MAC de destino y se realiza una decisión sobre a qué puerto de salida debe dirigirse ese paquete.

Por lo tanto, cuando un paquete entra en un dispositivo de conmutación y no sabe a dónde debe dirigirse, se envía al controlador y dependiendo de su destino y el origen, primero se averigua en qué dispositivo nos encontramos :

- Dispositivo de conmutación vinculado directamente a cada host : Para este tipo de conmutadores se insertan las reglas correspondientes según el prefijo común que se obtenga entre el host origen y el host destino. Si el prefijo común es 0, es decir, si el host destino se encuentra en otra DCell de nivel $k-1$, entonces se busca el conmutador que interconecte con la DCell destino de nivel $k-1$ correspondiente al host destino. Con lo cual, nos podemos encontrar con los siguientes escenarios :

- Si el conmutador que tiene el paquete es el que interconecta con la DCell de nivel $k-1$, entonces se envía el paquete por el puerto 4, y llegaría a un conmutador de la DCell del mismo nivel $k-1$ que el host destino.
- Si el conmutador que tiene el paquete está en la misma DCell de nivel $k-1$, pero distinto nivel $k-2$, entonces se envía el paquete por el puerto 2, y llega al dispositivo de interconexión de hosts.
- Si el conmutador que tiene el paquete es el que interconecta con la DCell destino de nivel $k-2$, entonces se envía el paquete por el puerto 3 y llega a un conmutador de la DCell del mismo nivel $k-2$ que el host destino.
- Si en cambio, no se está en ninguna de las situaciones mencionadas anteriormente, entonces significaría que se debe enviar el paquete al host vecino, con lo cual se utiliza el puerto de salida 2 y el paquete llegaría al conmutador de interconexión de hosts.

Si nos encontramos en la misma DCell de nivel $k-1$, pero distinta de nivel $k-2$, entonces el prefijo común sería 1 y estaríamos en una situación similar a la mencionada anteriormente, en la que se ha de buscar el host que tenga una conexión con la DCell destino de nivel $k-2$ en la misma DCell de nivel $k-1$ origen, y por lo tanto se analizarían los mismos casos relacionados con la DCell de nivel $k-2$ mencionados anteriormente.

Y por último, en el mejor de los casos, que nos encontremos en la misma DCell tanto de nivel $k-1$ como de nivel $k-2$. En este caso sólo haría falta envíar el paquete al dispositivo de interconexión de hosts, y éste se lo enviaría al host destino, que estaría directamente conectado a él.

- Dispositivo de conmutación de interconexión de hosts : Para este tipo de conmutadores también se insertan las reglas correspondientes según el prefijo común que se obtenga entre el host origen y el host destino. Nos podemos encontrar con los siguientes casos :

  - Si el paquete no se encuentra en la misma DCell destino de nivel $k-1$, entonces se envía por el puerto correspondiente al número de la DCell destino de nivel $k-1$, ya que el cableado se ha realizado de manera que el puerto n de este tipo de conmutadores conecte directamente con la DCell n de nivel $k-1$.
  - Si el paquete ya se encuentra en la DCell del nivel $k-1$ adecuada, entonces se calcula si el host destino es par o impar. Si es par se envía el paquete por el puerto 1 ya que es el que interconectaría con la DCell destino de nivel $k-2$. Si en cambio es impar se envía por el puerto 2.
  - Por último, si el paquete se encuentra en la misma DCell de nivel $k-1$ así como de nivel $k-2$, entonces se envía el paquete por el puerto correspondiente al número del host destino.

Capítulo 3

Resultados

3.1. Metodología de pruebas

Los escenarios de prueba que se han considerado son los siguientes:

- Topología FatTree con enrutamiento basado en direcciones IP y sin retardo.
- Topología FatTree con enrutamiento basado en direcciones IP con un retardo de enlaces de 2 ms.
- Topología FatTree con enrutamiento basado en direcciones MAC virtuales y sin retardo de enlaces.
- Topología FatTree con enrutamiento basado en direcciones MAC virtuales con un retardo de enlaces de 2 ms.
- Topología DCell con enrutamiento basado en direcciones MAC y sin retardo..
- Topología DCell con enrutamiento basado en direcciones MAC con un retardo de enlaces de 2 ms.

La topología FatTree se parametrizó con 6 pods, 4 cores, 4 subredes, y 4 hosts en cada uno de ellos. En total son 96 hosts. Por otro lado, la topología DCell se parametrizó con 2 hosts y $n = 2$. En total son 42 hosts, 7 DCells de nivel 1 con 3 DCells de nivel 0 en cada uno de ellos.

A cada uno de estos escenarios se le aplican una serie de pruebas para medir distintos aspectos relacionados con el rendimiento, que se explican en la siguiente sección.

A cada escenario se le aplicaron tres tipos distintos de pruebas:

- **Prueba 1**: Ping de uno de los hosts (se ha utilizado el host `h0`) con el host más alejado en base al número de saltos dentro de la topología. Esto permite tener una idea de los mayores retardos promedio que cabe esperar en cada caso y las posibles fluctuaciones.
- **Prueba 2**: Prueba de conectividad y medida de los tiempos de ida y vuelta de los paquetes enviados desde cada uno de los hosts que componen la topología, hasta el resto de los hosts destinos. Para realizar esta prueba se ha utilizado el comando `pingallfull` que proporciona Mininet.

- **Prueba 3**: Prueba de velocidad media de un flujo de paquetes entre hosts cercanos y llegando hasta los más lejanos. Para realizar esta prueba se ha utilizado el comando `iperf` disponible en Mininet.

Cabe esperar que los retardos de propagación de los enlaces internos de un centro de datos sean muy pequeños por lo que se testeó cada escenario sin tener en cuenta los retardos y luego se introdujo un retardo de propagación de 2 ms para comprobar el posible efecto de éste sobre la simulación.

Todas las pruebas han sido realizadas en un ordenador MSI A10M con procesador Intel®Core™i5-10210U a 1.6 GHz. En cada una de ellas no se ha ejecutado ninguna otra tarea en paralelo con el fin de obtener los resultados más reales posibles, teniendo en cuenta en todo momento que es una simulación. Además, se ha utilizado el máximo número de dispositivos posibles soportado por el ordenador en cuanto a la escalabilidad de las topologías.

3.2. Prueba 1: `ping`

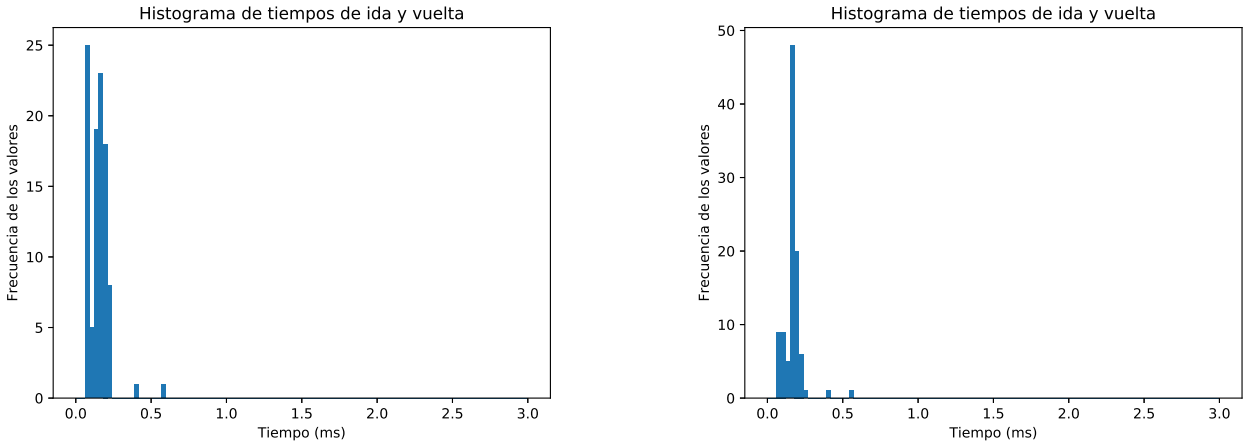
Esta prueba consiste en realizar un `ping` entre los dos host más alejados de la topología para determinar su efecto. Cada `ping` se repitió 100 veces con el fin de observar las fluctuaciones de los tiempos de ida y vuelta medidos.

En la figura 3.1, se muestran los tiempos de ida y vuelta medidos para el FatTree con enrutamiento basado en direcciones IP. En la fila superior, se pueden observar los resultados para el escenario sin retardos y en la fila inferior se muestran los mismos datos para el escenario con retardos de 2 ms. La columna de la izquierda corresponde a una primera ejecución y la de la derecha a una segunda.

Por otra parte, en la figura 3.2, se muestran los tiempos de ida y vuelta medidos para el FatTree con enrutamiento basado en direcciones MAC. Al igual que para el escenario anterior, en la fila superior, se pueden observar los resultados para el escenario sin retardos y en la fila inferior se muestran los mismos datos para el escenario con retardos de 2 ms. La columna de la izquierda corresponde a una primera ejecución y la de la derecha a una segunda.

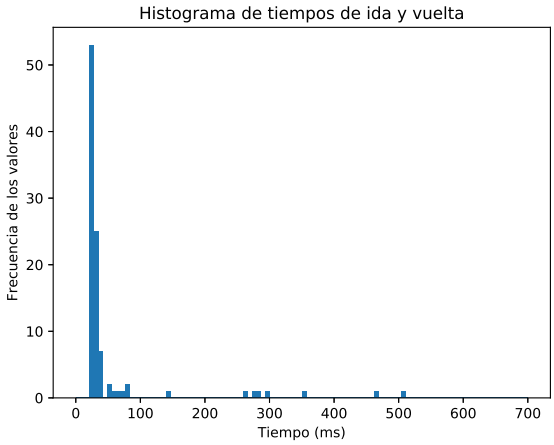
Finalmente, en la figura 3.3, se muestran los tiempos de ida y vuelta medidos para la topología DCell. Igual que antes, la fila superior muestra los resultados para el escenario sin retardos y en la fila inferior se muestran los mismos datos para el escenario con retardos de 2 ms. La columna de la izquierda corresponde al escenario con las tablas de flujo vacías, mientras que la de la derecha corresponde a un escenario con las tablas de flujo pobladas.

En la tabla 3.1, se puede encontrar un resumen de los histogramas recogidos en las figuras 3.1a, 3.1c, 3.2a, 3.2c, 3.3a y 3.3c, es decir, para las pruebas con las tablas de flujo vacías. La tabla muestra el valor mínimo, el valor máximo, el promedio y la desviación típica de los tiempos de ida y vuelta de los 100 pings ejecutados en cada uno de los escenarios.

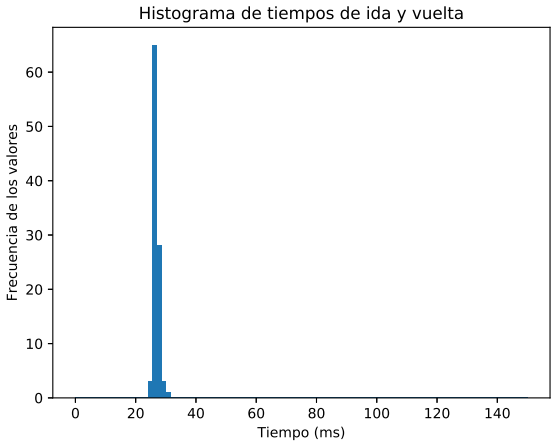


(a)

(b)



(c)



(d)

Figura 3.1: Histogramas de tiempos de ida y vuelta para FatTree - IP. (a) Sin retardo (primera ejecución). (b) Sin retardo (segunda ejecución). (c) Delay de enlaces 2 ms (primera ejecución). (d) Delay de enlaces 2ms (segunda ejecución).

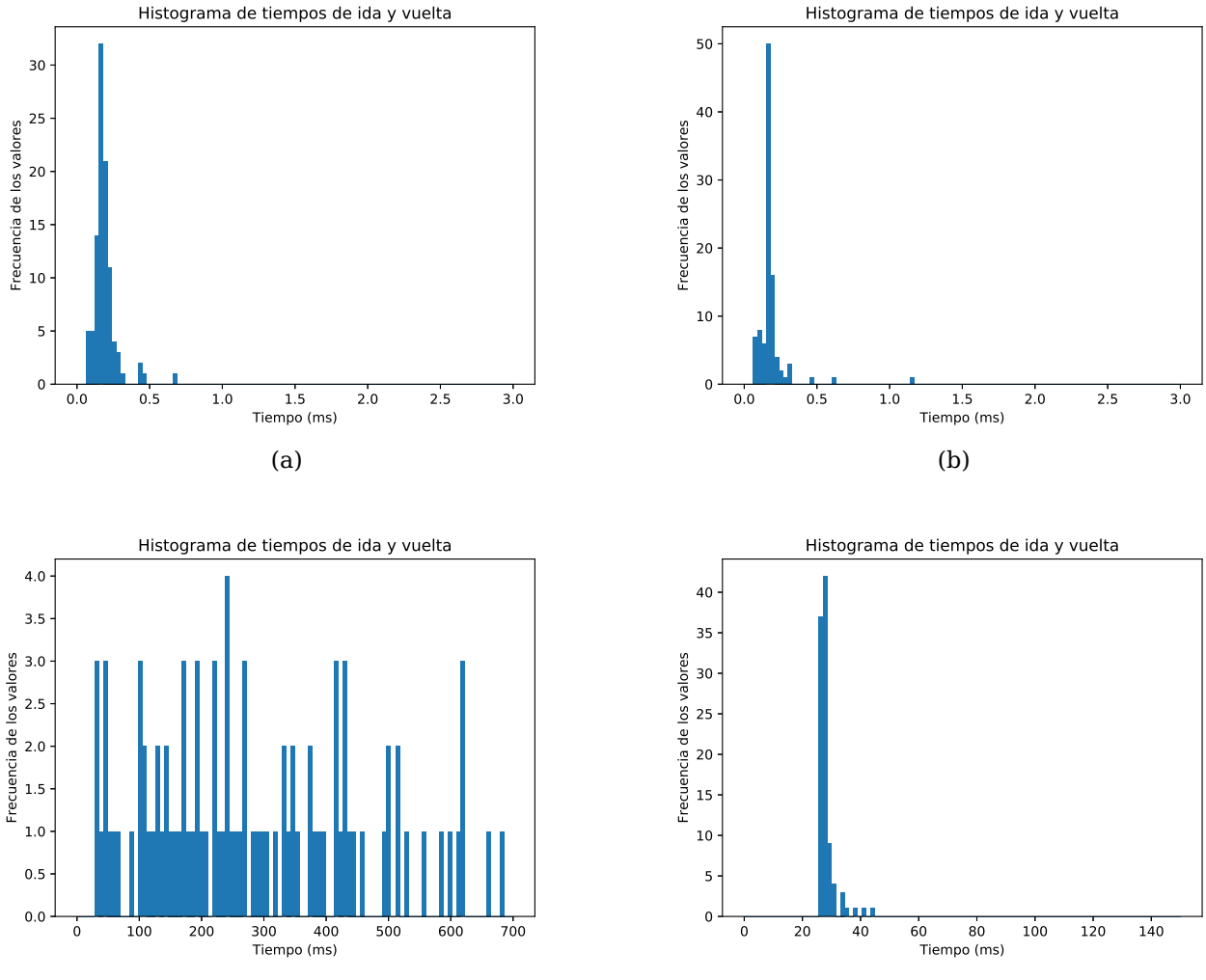
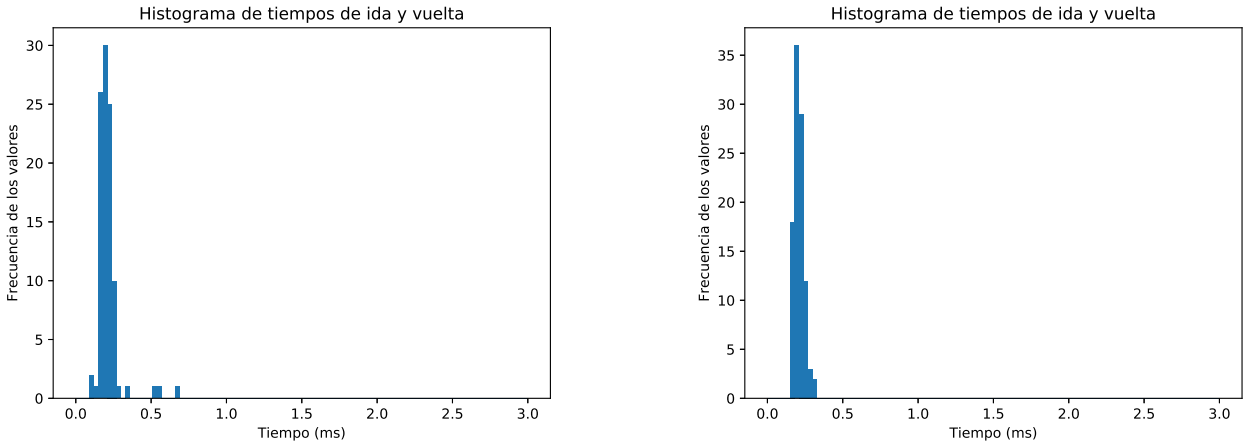


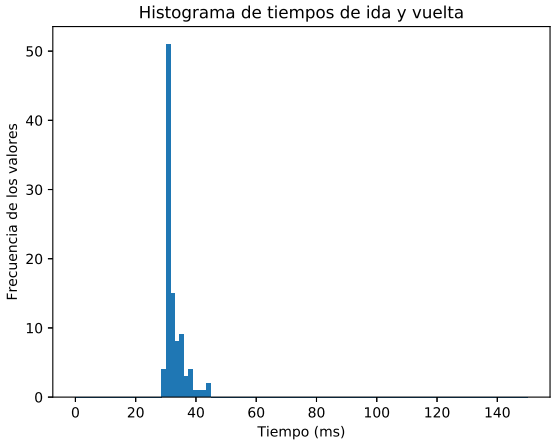
Figura 3.2: Histogramas de tiempos de ida y vuelta para FatTree - MAC. (a) Sin retardo (primera ejecución). (b) Sin retardo (segunda ejecución). (c) Delay de enlaces 2ms (primera ejecución). (d) Delay de enlaces 2 ms (segunda ejecución).



(a)

(b)

(c)



(d)

Figura 3.3: Histogramas de tiempos de ida y vuelta para DCell. (a) Sin retardo (con tabla de flujo vacía). (b) Sin retardo (con tabla de flujo poblada). (a) Delay de enlaces 2 ms (con tabla de flujo vacía). (b) Delay de enlaces 2 ms (con tabla de flujo poblada).

	FatTree – IP (sin retardo)	FatTree – IP (2 ms)	FatTree – MAC (sin retardo)	FatTree – MAC (2 ms)	DCell (sin retardo)	DCell (2 ms)
Minimo	0,06	26,10	0,07	31,90	0,1	29,10
Máximo	0,57	158,30	0,67	746,00	191	120,00
Medio	0,15	54,80	0,18	330,00	2,12	32,28
Desviación típica	0,07	85,55	0,08	223,48	18,98	10,81

Tabla 3.1: Resumen de tiempos de ida y vuelta medidos en milisegundos para los distintos escenarios de prueba.

3.3. Prueba 2: `pingallfull`

Esta prueba consistió en realizar un ping desde cada uno de los hosts origen hacia el resto de hosts en la topología. En la figura 3.4 se muestra el resultado para los tres escenarios poniendo a 0 ms el retardo de los enlaces. La columna de la izquierda muestra los histogramas de tiempos de ida y vuelta para los distintos escenarios y la columna de la derecha muestra los tiempos codificados en un mapa de color para cada pareja de hosts posibles. En este caso, el eje vertical corresponde al emisor del ping y el eje horizontal corresponde al destinatario. El mismo experimento se repitió suponiendo retardos de 2 ms. Los resultados obtenidos se pueden observar en la figura 3.5.

3.4. Prueba 3: Velocidad de transmisión

La velocidad de transmisión promedio es otro parámetro crítico en el centro de datos. Por ello, se midió la velocidad entre los dos hosts cercanos, dos hosts a distancia media y dos hosts a distancia lejana en número de saltos. Los resultados obtenidos de estas pruebas se muestran en la tabla 3.2. Además, en la figura 3.6 se pueden observar los datos de la tabla de manera gráfica.

3.5. Discusión

Para cada topología se han considerado dos experimentos, el que no contempla retados y el que contempla unos retardos de propagación, quizás exagerados, de 2 ms por enlace, con el fin de comprobar el efecto del mismo sobre el retardo global. Hay que tener en cuenta que los resultados que se han mostrado se han obtenido a través de dipositivos simulados en Mininet, lo que podría haber condicionado las medidas realizadas. Las cifras obtenidas deben interprestarse siempre con la debida cautela, ya que en una infraestruc-

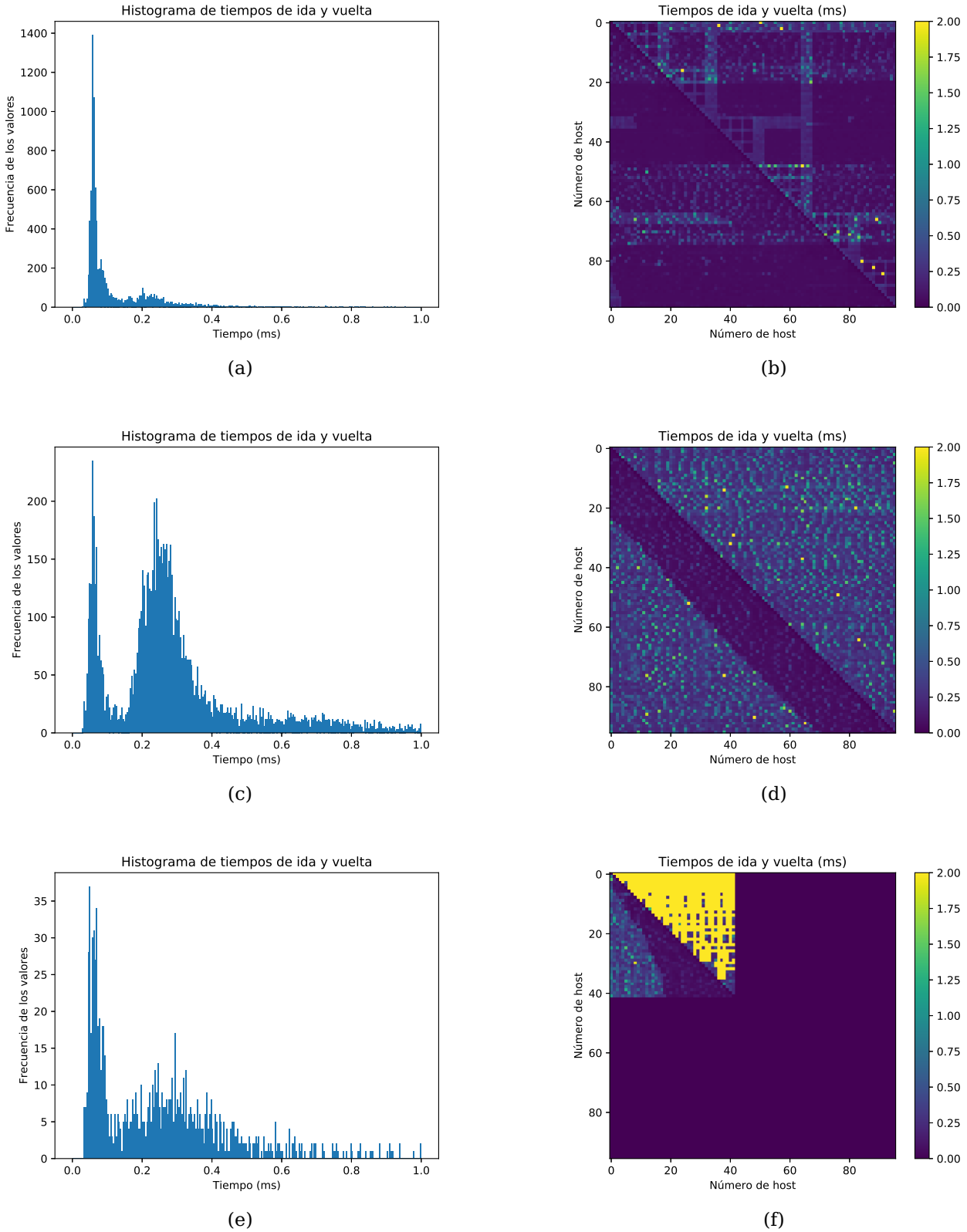
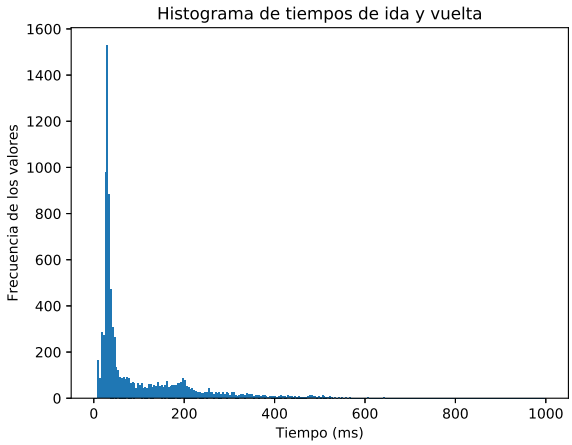
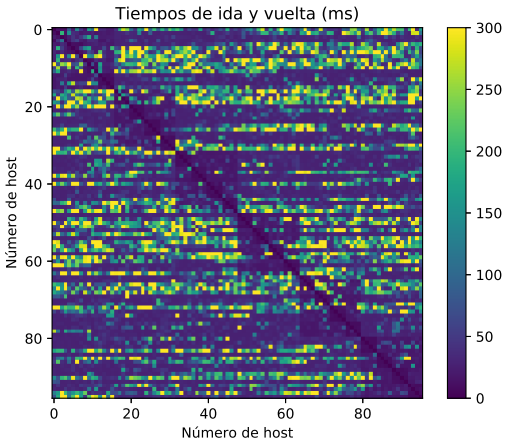


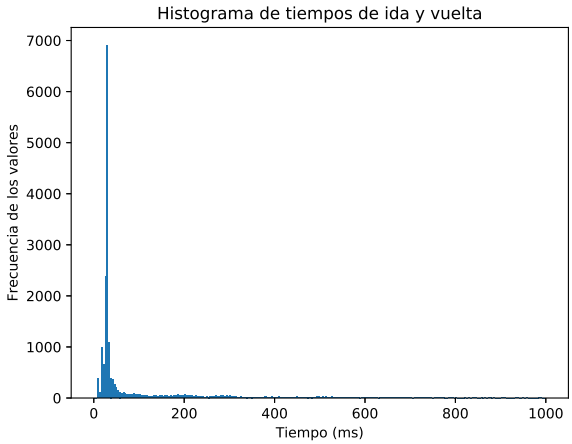
Figura 3.4: Tiempos de ida y vuelta todos con todos (sin retardo de enlace). (a) Histograma Fat Tree - IP. (b) Tiempos de ida y vuelta Fat Tree - IP. (c) Histograma Fat Tree - MAC. (d) Tiempos de ida y vuelta Fat Tree - MAC. (e) Histograma DCell. (f) Tiempos de ida y vuelta DCell.



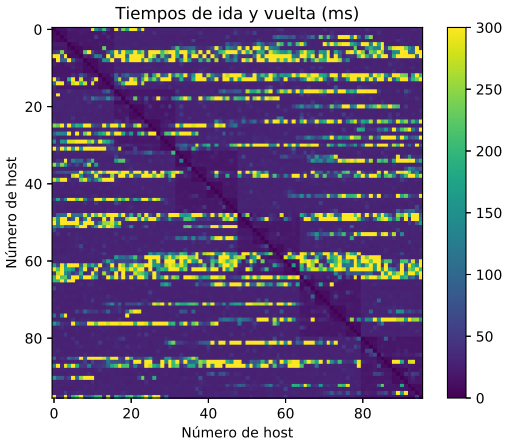
(a)



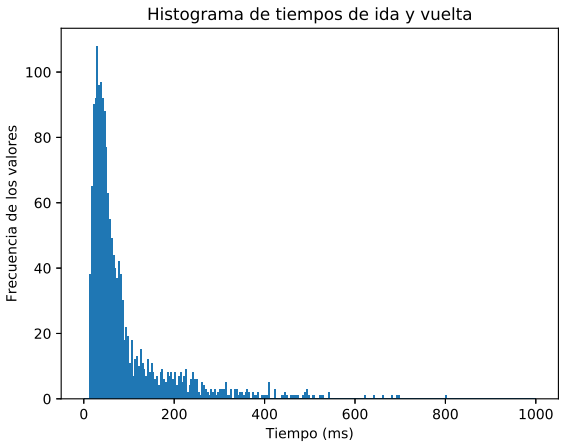
(b)



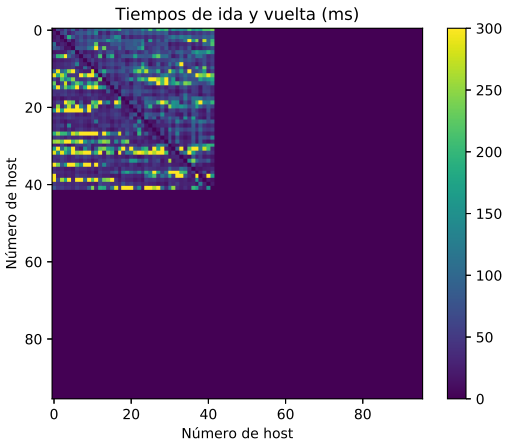
(c)



(d)



(e)



(f)

Figura 3.5: Tiempos de ida y vuelta todos con todos (retardo de enlace 2 ms). (a) Histograma Fat Tree - IP. (b) Tiempos de ida y vuelta Fat Tree - IP. (c) Histograma Fat Tree - MAC. (d) Tiempos de ida y vuelta Fat Tree - MAC. (e) Histograma DCell. (f) Tiempos de ida y vuelta DCell.

	FatTree – IP (sin retardo)	FatTree – IP (2 ms)	FatTree – MAC (sin retardo)	FatTree – MAC (2 ms)	DCell (sin retardo)	DCell (2 ms)
Cercano	89,2	62,6	88,4	82,4	83,7	89,8
Mediana	85,3	60,2	89,3	61,4	78,8	67,3
Lejana	79,4	56,2	84,8	60,5	68,4	57,1

Tabla 3.2: Velocidades de transmisión promedio.

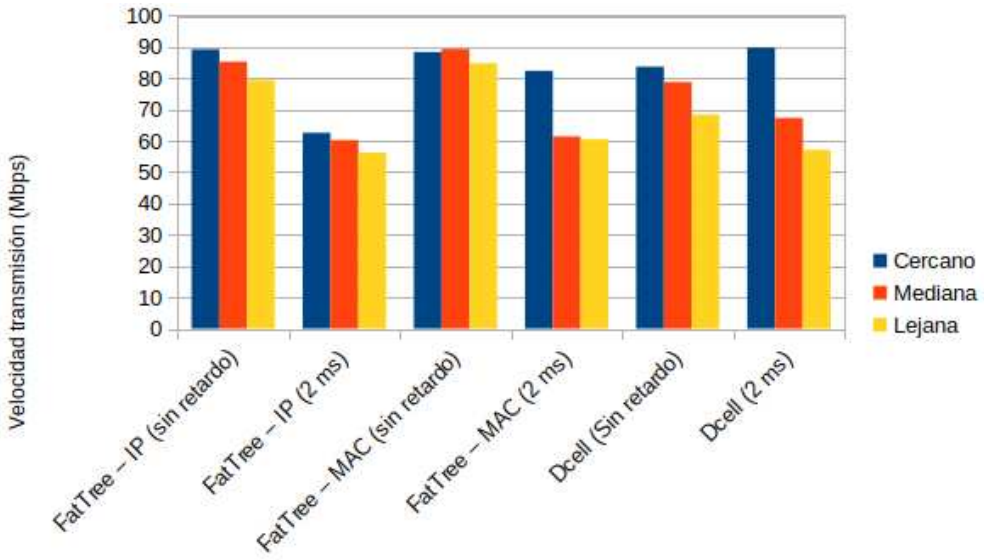


Figura 3.6: Velocidades de transmisión promedio medidas en Mbps.

tura real los datos cambiarían, al menos desde un punto de vista cuantitativo.

En la figura 3.1, se muestran los tiempos de ida y vuelta medidos para un “ping“ entre los hosts más alejados de la toplogía FatTree con enrutamiento basado en direcciones IP. Se hacen dos ejecuciones para cada caso (sin retardo y con un retado de enlace de 2 ms). Se observa cómo el comportamiento es similar en ambos casos, ya que en esta arquitectura las tablas de flujo se rellenan de manera proactiva al comienzo de la ejecución del dispositivo, por lo que no cabe esperar retardos debido a la accción del controlador. Cabe destacar que se observan algunos retardos más altos (de hasta 520 ms) en la figura 3.1c, que podrían deberse al uso de la CPU por parte de otros procesos durante la simulación, ya que este efecto no se observa en la figura 3.1d.

Se repitió el mismo experimento con la topología FatTree con direccionamiento basado en direcciones MAC. Como muestran los resultados de la figura 3.2, para el caso sin retardo ambas ejecuciones tienen un comportamiento similar tal y como se esperaba, observándose algunas medidas de mayor retardo. Para el experimento con retardo de enlace de 2 ms, se observa un comporatamiento inestable con unos retardos muy elevados (figura 3.2c), que luego en la segunda ejecución desaparece (figura 3.2d). No se ha podido determinar el motivo de este comportamiento.

En la figura 3.3, se muestran los resultados para el caso de la topología DCell. Aquí se pueden observar algunos retardos elevados cuando las tablas de flujo no están pobladas y el controlador debe insertar las reglas para el primer paquete. Además, como cabe esperar, éstas desaparecen en la segunda ejecución de la prueba cuando las tablas de flujo ya están pobladas.

Los comporatmientos descritos anteriormente también se pueden observar en la tabla 3.1, donde resulta especialmente llamativo observar los valores máximos de algunos casos, como es el caso del DCell sin retardo. Estos serían atribuibles a que los primeros paquetes deben enviarse al controlador. Sin embargo, este fenómeno no se observa para las dos variantes de la arquitectura FatTree sin retardo, puesto que en éstas se rellena la tabla de flujo al conectarse cada conmutador al controlador la primera vez.

La mera comprobación de conectividad entre dos hosts no permite verificar el correcto funcionamiento del esquema de enrutamiento implementado para cada una de las topologías. Por ello, se realizó una prueba de conectivad de todos con todos. Los resultados para el caso sin retardo se reflejan en la figura 3.4. Se muestran tanto los histogrmas de tiempos de ida y vuelta, como una imagen codificada en color con los RTT entre cada pareja de hosts de la topología. Esto permite observar que los esquemas de enrutamiento implementados permiten la conectividad entre todos los hosts.

En el caso de la topología DCell hay que destacar que tiene sólo 42 hosts, de ahí que la imagen no ocupe todo el recuadro. Comparando las figuras 3.4a y 3.4c, se puede observar un comportamiento similar en el que presentan 2 picos, que se pueden relacionar con las distancias de los hosts que están en el mismo pod y los hosts de otro pod, aunque en la figura 3.4c el pico de mayor distancia es mucho mayor. Para el caso de la topología DCell, se observa algo similar, pero en este caso es atribuible al hecho de usar una inserción de reglas reactiva (se insertan cuando aparezca un paquete para ese flujo). Esto se puede

contrastar en la figura 3.4f, donde se observan unos retardos mayores para las parejas de hosts que aún no han intercambiado datos (recordemos que el “pingall“ ejecuta las pruebas en orden por número de host), mientras que para el recorrido inverso estos retardos no son tan altos porque ya se han preparado las reglas correspondientes con el paquete de ida.

En cuanto a las pruebas de velocidad, la tabla 3.2 y la figura 3.6 permiten observar cómo éstas siempre están por debajo de los 100 Mbps que es el enlace de menor ancho de banda. La prueba se ha realizado entre parejas de hosts a distintas distancias dentro de la topología. Se observa el comporatamiento esperado en la mayoría de los casos, que es una disminución de la velocidad de transferencia media ya que a mayor número de saltos, mayor número de dispositivos de reenvío son implicados. Por otra parte, el retardo de enlace no parece tener una influencia significativa, como también es de esperar. Se vuelve a detectar la anomalía, vista anteriormente, en el caso del FatTree enrutado con direcciones MAC.

Capítulo 4

Conclusiones y líneas futuras

Las conclusiones de este trabajo de fin de grado son las siguientes:

- Se han simulado dos topologías de datacenter (FatTree y DCell) en Mininet. Esto ha permitido probar distintos programas de control sobre las mismas.
- En cuanto a las herramientas, Mininet, a pesar de no disponer de un entorno gráfico, se presenta como una de las herramientas más potentes y ágiles en cuanto a velocidad de creación y puesta en marcha de topologías RDS. Desde que se tenga lista su implementación y una vez terminado el despliegue tarda pocos segundos en ponerse en marcha.
- Se han implementado dos alternativas para el programa de control de la topología FatTree. Igualmente, se ha implementado un programa de control para la topología DCell, utilizando para ambos casos el controlador Ryu.
- Se ha realizado una serie de experimentos sobre las distintas topologías e implementaciones de programas de control obteniendo medidas de velocidad y de retardo, que han permitido comprobar el funcionamiento de las mismas, en cuanto a conectividad y correcto funcionamiento de los esquemas de enrutamiento planteados.
- En términos generales, una topología FatTree parece más fácil de implementar que una DCell debido a su sencillez en la arquitectura y una mayor facilidad a la hora de tener que detectar el origen de un fallo.
- Si el objetivo principal es la escalabilidad, la mejor opción es sin duda una topología de DCell, ya que el número de hosts crece exponencialmente a medida que se sube de nivel.
- El uso de direcciones MAC virtuales que permiten realizar un enrutamiento a nivel de enlace es un recurso interesante dentro de un centro de datos, ya que permite trabajar luego a nivel de IP sin atender a dónde está exactamente un host dentro de la topología.
- Una de las limitaciones del desarrollo realizado consiste básicamente en la seguridad de la topología y la escalabilidad en cuanto al número de controladores, ya que en este caso particular, sólo se ha desarrollado uno que controle a toda la red.

En cuanto a los trabajos futuros cabe destacar lo siguiente :

- Abordar la implementación del programa de control para la resolución ARP dentro de las topologías implementadas.
- Someter las simulaciones realizadas a pruebas con perfiles de tráfico real, para comprobar su rendimiento en estas situaciones. Asimismo, tomar medidas del tráfico broadcast en base a las aplicaciones y estudiar posibles formas de limitarlo.
- Sería interesante poder tomar medidas con dispositivos físicos para contrastar las cifras obtenidas en la simulación, aunque esto supone el acceso a un centro de datos, o al menos a parte del mismo para realizar el cableado con las topologías estudiadas, por lo que se considera difícil su desarrollo.
- Desarrollar esquemas de enrutamiento dinámicos y tolerantes a fallos.
- Estudiar la posibilidad de utilizar varios controladores y el manejo de su redundancia.

Capítulo 5

Summary and Conclusions

5.1. Final conclusions and future ideas

The conclusions of this end-of-degree project are as follows:

- Two datacenter topologies (FatTree and DCell) have been simulated on Mininet. This has made it possible to test different control programs on them.
- Regarding the tools, Mininet, despite not having a graphical environment, is presented as one of the most powerful and agile tools in terms of speed of creation and implementation of SDN topologies. Once your implementation is ready and the deployment is finished, it just takes a few seconds to get started.
- Two alternatives for the FatTree topology control program have been implemented. Likewise, a control program for the DCell topology has been implemented, using for both cases the Ryu controller.
- A series of experiments have been carried out on the various topologies and implementations of the control programs, obtaining speed and delay measurements, which have made it possible to check their performance, in terms of connectivity and correct functioning of the proposed routing schemes.
- In general terms, a FatTree topology seems easier to implement than a DCell due to its simplicity in the architecture and a greater ease in detecting the origin of a fault.
- If the main goal is the scalability, the best option is certainly a DCell topology, as the number of hosts grows up exponentially as you level up.
- The use of virtual MAC addresses that allow to perform a routing at link level is an interesting resource within a data center, since it allows to work at IP level without considering where exactly a host is within the topology.
- One of the limitations of the development is basically the safety of the topology and the scalability in terms of the number of controllers, since in this particular case, only one is controlling the whole network.

As for future work, the following should be noted:

- Address the implementation of the control program for ARP resolution within the implemented topologies.

- Test simulations with real traffic profiles to check their performance in these situations. Also to take measures of broadcast traffic based on applications and explore possible ways to limit it.
- It would be interesting to be able to take measures with physical devices to contrast the figures obtained in simulation, although this presupposes access to a data center, or at least to part of it to perform the wiring with the studied topologies, but that it is considered difficult to achieve.
- Develop dynamic and fault-tolerant routing schemes.
- Study the possibility of using multiple controllers and managing their redundancy.

Capítulo 6

Presupuesto

Los recursos hardware necesarios para el desarrollo del proyecto han sido un portátil de prestaciones medias de alrededor de 600€. Se ha calculado el coste de amoritzación del equipo, suponiendo una vida útil de 5 años. En cuanto a los recursos de software necesarios, contamos con Mininet, Ryu, y Python, pero éstos no suponen coste alguno ya que son gratuitos. Por último, los recursos humanos que serían necesarios consisten en un ingeniero informático durante 150 horas (26€ por ahora).

Tipo de recurso	**Precio (en Euros)**
Amortización de equipos	30
Licencias de software	0
Salarios	3.900
Subtotal	3.930

Tabla 6.1: Presupuesto

Bibliografía

[1] Paul Goransson, Chuck Black, and Timothy Culver. *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[2] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. pages 225–238, 2012.

[3] Gyarmati and Tuan Anh Trinh. Scafida: A scale-free network inspired data center architecture. *CCR*, 2010.

[4] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

[5] Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6. Ieee, 2014.

[6] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 75–86, 2008.

[7] Eric Jo, Deng Pan, Jason Liu, and Linda Butler. A simulation and emulation study of sdn-based multipath routing for fat-tree data center networks. In *Proceedings of the Winter Simulation Conference 2014*, pages 3072–3083. IEEE, 2014.

[8] JJ. Velasco. Historia de la tecnología. *https://hipertextual.com/2013/06/historia-de-la-tecnologia-manchester-baby*, 2013.

[9] Patricia A Morreale and James M Anderson. *Software defined networking: Design and deployment*. CRC Press, 2014.

[10] The OpenÑetworking Foundation. Openflow switch specification. 2012.

[11] Amit Nayyer, Aman Kumar Sharma, and Lalit K Awasthi. Issues in software-defined networking. In *Proceedings of 2nd International Conference on Communication, Computing and Networking*, pages 989–997. Springer, 2019.

[12] Idris Zoher Bholebawa, Rakesh Kumar Jha, and Upena D Dalal. Performance analysis of proposed openflow-based network architecture using mininet. *Wireless Personal Communications*, 86(2):943–958, 2016.

[13] Yong Li and Min Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[14] Albert Rego, Laura Garcia, Sandra Sendra, and Jaime Lloret. Software defined network-based control system for an efficient traffic management for emergency situations in smart cities. *Future Generation Computer Systems*, 88:243–253, 2018.