Universidad de Alcalá

Escuela Politécnica Superior

Máster en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Desarrollo de un Switch híbrido Open Flow/All Path

Autor: Joaquín Álvarez Horcajo

Tutor: Isaías Martínez Yelmo

2017

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Máster en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Desarrollo de un Switch híbrido Open Flow/All Path

Autor: Joaquín Álvarez Horcajo

Tutor/es: Isaías Martínez Yelmo

TRIBUNAL:

Presidente: Guillermo Agustín Ibáñez Fernández

Vocal 1º: Lucas Cuadra Rodríguez

Vocal 2º: Isaías Martínez Yelmo

CALIFICACIÓN: _____

FECHA: _____

Contenido

Índice de Ilustraciones

Índice de Tablas

Dedicatoria

Me gustaría de dedicar este libro a mi futura mujer María ya que sin su apoyo y ayuda no hubiera podido sacarlo adelante.

A mi familia sin la cual no hubiera llegado tan lejos.

A mi tutor oficial Isaías ya que sin el este proyecto no hubiera sido lo que es.

A mi tutor extra-oficial Juan Antonio puesto que sin sus consejos y sabiduría no hubiera llegado tan lejos.

A todo el grupo NetServ por haber confiado a mí ya hace más de dos años.

Por último me gustaría agradecer al proyecto TIGER-5 la posibilidad de realizar este proyecto puesto que gracias a él se ha podido llevar a cabo este proyecto.

Resumen

El presente proyecto tiene como objetivo el diseño e implementación de un switch Híbrido desarrollado en lenguaje C capaz de funcionar tanto en las redes tradicionales como en las redes basadas en la arquitectura de SDN.

El sistema llevará implementado, además de la lógica necesaria para la interconexión con las redes SDN, un protocolo de encaminamiento autónomo (ARP-Path) el cual dotará de la "inteligencia" necesaria para la generación de caminos de mínima latencia.

Así mismo este switch dispondrá también de dos sistemas de recuperación diferentes: El primero totalmente centralizado apoyado en controlador, el segundo totalmente distribuido apoyado únicamente en los switches de la red.

Palabras Clave: SDN, Switch Híbrido, Recuperación, Controlador, OpenFlow, Arquitectura de Red, Latencia, ARP-Path.

Abstract

This project aims at the design and implementation of a hybrid switch developed in C language capable of interconnecting with both traditional networks and the networks based on the SDN architecture.

This system has implemented the logic for interconnect with SDN networks, also the system will implement an autonomous routing protocol (ARP-Path) which will provide the necessary "intelligence" for the generation of minimum latency paths.

Likewise this switch will also have two different recovery systems, the first fully centralized controller supported, the second fully distributed.

Keywords: SDN, Hybrid Switch, Recovery, Controller, OpenFlow, Network architecture, Delay, ARP-Path.

Resumen Extendido

Las redes de datos han ido evolucionando a lo largo de los últimos años. Está evolución ha elevado, no solo las prestaciones de las redes, sino también ha llevado consigo un cambio en el planteamiento, arquitectura y diseño de la mismas.

En esta línea de cambio han surgido múltiples propuestas, en este proyecto se va a prestar especial atención a la arquitectura SDN. Dicha arquitectura establece como primera modificación una división en tres capas funcionales distintas:

- La primera (capa de infraestructura) estará compuesta por todos los dispositivos físico o virtuales de la red, definiendo como dispositivos los switches multicapa capaces de interactuar con la capa inmediatamente superior.
- La segunda (capa de controlador) estará compuesta por un "*middleware"* encargado de interconectar la capa de infraestructura con la capa de aplicación.
- La tercera (capa de aplicación) estará compuesta de las diferentes lógicas diseñadas por el administrador de red.

Esta división lleva asociado un cambio en la lógica de los dispositivos (capa 1), es decir, los nuevos "nodos" no dispondrán de ningún mecanismo autónomo para el reenvió o gestión de los paquetes, sino que deberán consultar las acciones a realizar al controlador SDN (capa 2).

Como se puede intuir esta arquitectura presenta ciertas ventajas claras, como son la facilidad de configuración y gestión de la red, o la versatilidad que presenta ante cualquiera alteración o modificación de la red. Por otro lado, se puede observar también las carencias que presenta: problemas de escalabilidad, sobre carga del controlador o la necesidad de crear una segunda red paralela para la comunicación con el controlador de forma eficiente.

Así mismo también se puede observar un cambio claro en los dispositivos de la red, puesto que ya no existe la posibilidad de utilizar los switches actuales, sino que se deberán usar unos nuevos switches multicapa.

Para llevar a cabo este proyecto, se va a realizar un pequeño estudio sobre dos switches SDN diferentes, Open vSwitch (OvS) y OfSoftSwitch13(CPqD), dicho estudio tendrá como objetivo conocer la estructura interna de ambos así como su implementación, para de esta forma, poder elegir cuál de los dos será el sistema donde implementar el switch híbrido.

Ambos dispositivos se implementan en lenguaje "C" de programación, pero su implementación distan mucho uno del otro, es decir, en el caso de OvS, se trata de una

implementación mixta, donde una parte se implementa en espacio de usuario mientras que la otra parte se implementa en espacio de Core. Por el contrario, OfSoftSwitch13 posee una implementación completamente en espacio de usuario. Otra gran diferencia es la complejidad de dicha implementación, en el caso de OvS es mucho más complicada ya que se implementa a través de CallBacks mientras que OfSoftSwitch13 es una programación lineal clásica. Estas y otras razones han llevado a concluir que el sistema donde se va a realizar este proyecto será OfSoftSwitch13.

Una vez seleccionado dicho switch, se ha de implementar un sistema autónomo de generación de caminos, para ello se ha decido utilizar ARP-Path, un protocolo generado en el grupo de investigación GIST de la UAH. Este protocolo pretende dotar al switch de la capacidad de generar caminos de forma autónoma sin necesidad de consultar al controlador, reduciendo así la carga sobre este y aumentando también la escalabilidad del sistema.

Por otro lado, se ha tenido en cuenta la posibilidad de fallo en las redes, para ello se ha diseñado e implementado dos mecanismos de recuperación:

- El primero, se trata de un sistema que intenta aprovechar la capacidad que posee el controlador para conocer la red y por lo tanto de generar caminos alternativos.
- El segundo, es sistema que trata de generar un camino alternativo de forma distribuida utilizado la propia red como base para generar el mecanismo de recuperación.

Para concluir, se ha llevado a cabo una serie de pruebas, que intentan comprobar no solo el correcto funcionamiento de la implementación realizada, sino también el impacto que tiene este desarrollo sobre los sistemas basados en SDN. Para comprobar dicho impacto, se han realizado pruebas que tienen como objetivo medir una serie de parámetros tales como: el tiempo de establecimiento, el “throughput”, el tiempo de recuperación o la diversidad de caminos.

Estas medidas han sido realizadas sobre la plataforma MiniNet, la cual permite emular una red completa en un sistema físico. Esta emulación muestra una mejora de rendimiento cuando se usa un sistema híbrido, también se puede observar cómo se reduce el tiempo de establecimiento, así como también, se observa una reducción en el procesamiento necesario por parte del controlador. Por lo tanto se puede concluir como la utilización de sistemas híbridos puede dar respuesta a alguna de las carencias propias de la arquitectura SDN.

1 Introducción

La evolución actual en las redes de datos ha llevado consigo una modificación en la infraestructura de red. Dicho replanteamiento de las redes no solo ha conseguido una sustancial mejora de la misma, sino que también ha aumentado el nivel de complejidad en la configuración y mantenimiento de las mismas. Esto ha llevado a proponer arquitecturas que faciliten a los administradores la operación y mantenimiento de las redes.

Para mitigar dicha complejidad han surgido diferentes propuestas que intentan hacer más sencilla la gestión de la red, como por ejemplo la arquitectura SDN[1] ("Redes definidas por software").

SDN, como se ha mencionado anteriormente, intenta subsanar el problema de la gestión de las redes centralizando la inteligencia de la red en un controlador (único o distribuido). Esta arquitectura propone una división de la estructura de la red en tres capas diferentes, como se muestra en la Ilustración 1: Arquitectura establecida en Arquitectura SDN[2]:

- *Capa de infraestructura*, en la que reside el hardware.
- *Capa de control*: Aquí se encontrarían los controladores, quienes mantienen la inteligencia de la red. Los controladores se comunicarían con el hardware de la capa de infraestructura con el protocolo OpenFlow[2], y brindarían una API para que puedan ser programados desde la capa de aplicación.
- *Capa de aplicación*: Es la de más alto nivel y desde la cual se controla el comportamiento de la red. En ella se encuentran todas las aplicaciones generadas por los administradores de red para el control, gestión, administración, etc...

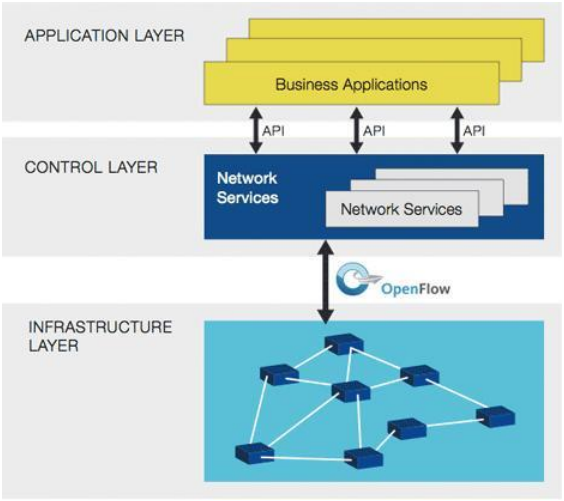


Ilustración 1: Arquitectura establecida en Arquitectura SDN[2]

Con esta separación se pretende que la inteligencia y el estado de la red sean gestionados de forma independiente (y a su vez, originalmente, de forma central) abstrayendo esta tarea de la complejidad que pueda suponer la red física. Esta gestión independiente la lleva a cabo el denominado Controlador SDN[3], que constituye el punto estratégico que retransmite información a los switches/routers de la red que hay en la capa de infraestructura, y también a las aplicaciones y la lógica de negocio que hay en la capa de aplicación.

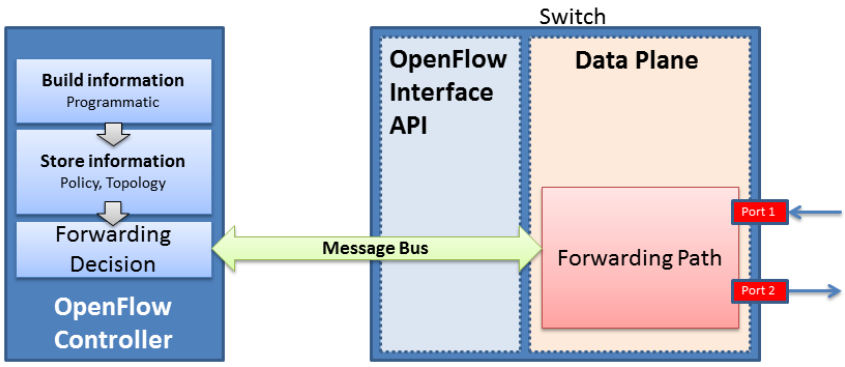


Ilustración 2: Separación plano de control de plano de datos[4]

Un controlador[3] SDN es típicamente un conjunto de módulos insertables dinámicamente que realizan diversas tareas de red como, por ejemplo, realizar un inventario de todos los equipos de red existentes en ésta, recolectar sus capacidades, agrupar estadísticas de red, etc.

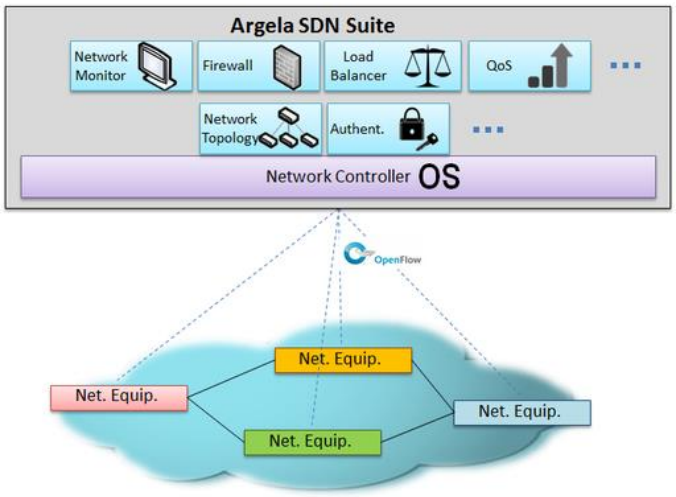


Ilustración 3: Ejemplo de controlador de alto nivel[5]

Sin embargo esta vuelta a los sistemas centralizados, donde todas las decisiones deben pasar por una entidad, trae consigo un aumento en el número de paquetes en la red, ya que para que el sistema en su conjunto funcione, cada nodo de la red debe solicitar a su controlador las reglas a ejecutar.

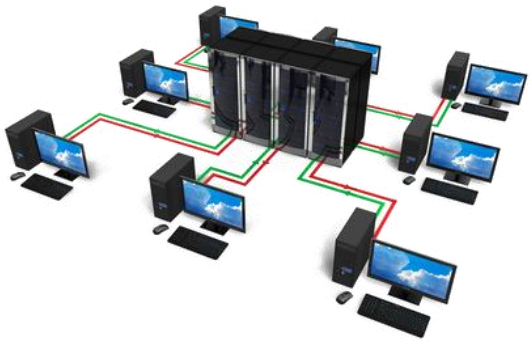


Ilustración 4: Ejemplo sistema centralizado[6]

El trabajo que aquí se presenta se ha definido en el entorno de las redes de capa dos, es decir, en redes compuestas por switches.

Un Switch es un dispositivo para la interconexión de redes que operan en la capa dos (nivel de enlace de datos) del modelo OSI. Su función es interconectar dos o más segmentos de red reenviando los paquetes de datos de un segmento a otro, siempre teniendo en cuenta las direcciones MAC destino.

Ilustración 5: Switch

El motivo por lo que se ha elegido este tipo de redes es el requisito implantado por el protocolo ARP-Path[7], protocolo elegido para la generación de caminos autónomos por parte del switch híbrido, es decir, como su propio nombre indica, este protocolo hace uso del protocolo ARP para generar caminos de mínima latencia en las redes.

ARP es un protocolo de nivel dos que permite, gracias a un intercambio de mensajes dentro de la misma red, descubrir la dirección MAC de los equipos implicados en una comunicación.

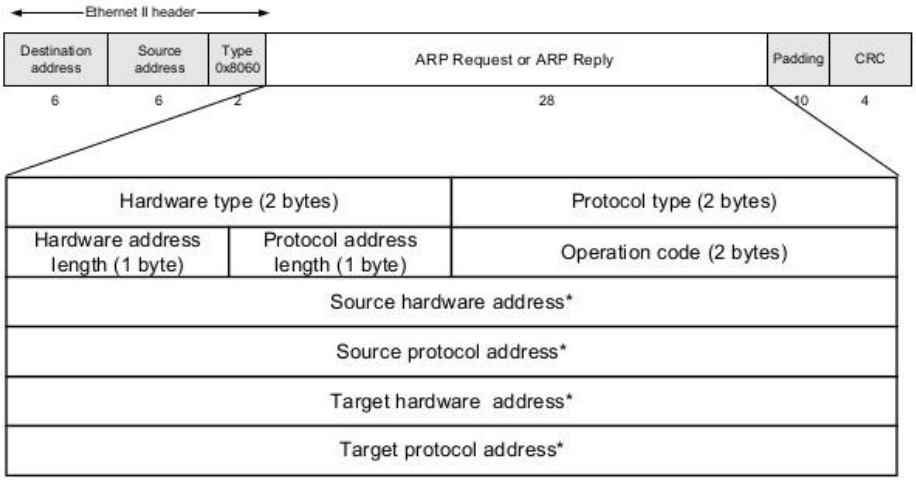


Ilustración 6: Paquete ARP

Para ello se hace uso de los paquetes ARP-Request, encargados de preguntar por la MAC del host destino, y ARP-Reply, encargados de transportar la respuesta del host destino.

Aprovechando este mecanismo de obtención de MAC utilizado por los host, ARP-Path establece caminos de mínima latencia entre ellos.

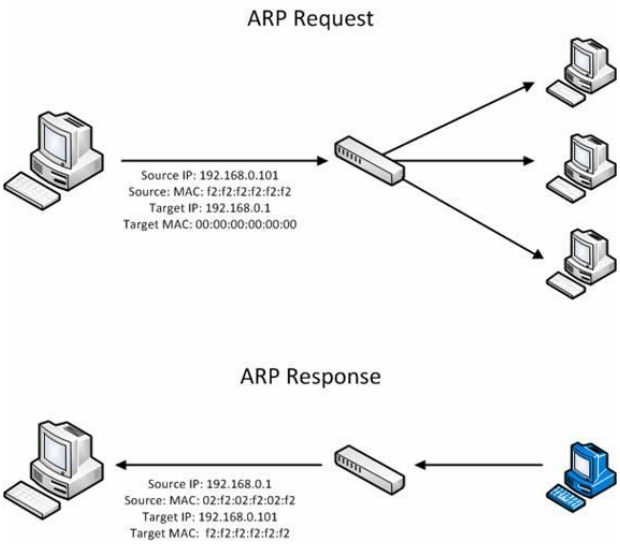


Ilustración 7: Funcionamiento ARP[8]

Utilizando el paquete broadcast ARP-Request establece el camino de vuelta entre el host A y el host B, mientras que con el paquete ARP-Replay establece el camino de ida entre ellos.

La propuesta que se desea realizar en este trabajo fin de Máster, es la de implementar en un switch híbrido SDN los protocolos estudiados por el grupo de investigación Gist-NetServ (ARP-Path[7]), los cuales establecen mecanismos autónomos para el establecimiento de caminos dentro de las redes, de tal manera que se intenta liberar de carga al controlador y bajar el número de paquetes intercambiados entre la capa de control y los diferentes nodos de la red.

1.1 Objetivos

Tras el planteamiento visto en la introducción de este proyecto, se puede establecer como objetivo general la implementación de un switch híbrido (AOSS), switch que hibrida los sistemas SDN centralizados con los protocolos de descubrimiento de caminos estudiados por el grupo de investigación Gist-NetServ.

Como se ha dicho, éste es el objetivo final, pero para llegar a él este proyecto va a establecer una serie de objetivos más precisos.

El primer paso a realizar es el estudio de la arquitectura SDN, ya que para la implementación de un sistema híbrido compatible con SDN primero debe conocerse el funcionamiento, arquitecturas, capas, protocolos, etc. que lo componen.

Una vez estudiado y entendido dicho apartado, el siguiente paso será la elección de un controlador para la implementación de toda la lógica SDN necesaria para realizar las diferentes pruebas o escenarios.

Por otro lado, no se debe perder de vista que el proyecto se va a desarrollar bajo un sistema de emulación, por lo tanto para poder continuar se debe primero entender la plataforma de emulación, en este caso MiniNet. En ella se deberá estudiar sus funcionalidades, limitaciones, automatizaciones, así como su API.

Así mismo, también, se ha determinado que el objetivo último de este proyecto es la implementación de un switch híbrido SDN, por lo tanto se requiere, como parte de este proyecto, del estudio de diferentes switches virtuales, compatibles con MiniNet, determinando tras dicho estudio cuál de ellos se tomará como referencia para la implementación del switch híbrido(AOSS).

Una vez se tenga toda la información anteriormente citada como objetivos, se tendrá únicamente la mitad de la información necesaria para la realización de este switch híbrido, es decir, se conoce toda la parte SDN del sistema. Por lo tanto, el siguiente objetivo será el estudio y compresión del protocolo para la creación de caminos autónomos (ARP-Path).

Con toda la información anterior ya se está en disposición de pasar a diseñar funcionalmente el switch, pasando al siguiente objetivo, el diseño funcional del sistema híbrido, por lo que se deberá obtener una serie de diagramas de flujos funcionales.

Una vez se esté en posesión de los diagramas funcionales, será el momento de pasar a realizar la implementación propiamente dicha. Para dar por correcto este objetivo se deberá obtener la implementación de un switch virtual totalmente funcional.

Una vez terminado este apartado se pasará al punto de recuperaciones. Para ello, como primer objetivo, se deberá obtener un esquema funcional de los dos tipos de recuperaciones objeto de este estudio: Una recuperación centralizada, donde todo el proceso se realiza en el controlador; así como una segunda recuperación distribuida, donde todo el proceso de recuperación se realiza en la propia red.

Como último objetivo de implementación, una vez obtenidos los diseños para las recuperaciones se deberá pasar a implementar todas las funciones necesarias para realizar correctamente ambas recuperaciones. Para el caso de la recuperación centralizada también se deberá implementar, en el controlador seleccionado, todas las funcionalidades necesarias para llevar a cabo dicha recuperación.

Para concluir con el proyecto, se propone como objetivo una fase de pruebas donde se desea comprobar varios apartados:

1. Retro compatibilidad con la nueva arquitectura. Al tratarse de un switch híbrido debe ser capaz de trabajar correctamente bajo las órdenes de un controlador SDN, por lo tanto deberá poder interactuar tanto con los paquetes OpenFlow como con sus tablas, así como bajo un sistema clásico, es decir, de forma totalmente autónoma.
2. Correcta ejecución, por parte del switch, de los diferentes protocolos All-Path, centrando el desarrollo en el protocolo ARP-Path, al ser el resto de protocolos modificaciones de éste.
3. Estudio del impacto que tiene en el controlador SDN la dotación de automatismos a la capa de infraestructura.
4. Comparación de los retardos producidos en el establecimiento de los caminos, Anchos de banda, Flow Completion Time (FCT) que existen entre los protocolos

autónomos implementados en el switch y los protocolos generados en el controlador SDN

5. Mejora de la escalabilidad de las redes SDN disminuyendo el impacto que tiene el controlador, que puede ser un cuello de botella, sobre las redes.

1.2 Estructura y contenidos de la memoria

Tras establecer los objetivos y el marco de trabajo para este proyecto, en este apartado se muestra, a modo resumen, la estructura lógica que se va a seguir en la memoria.

• Capítulo 1: Introducción al tema que se va a tratar en este proyecto así como, una exposición de los objetivos a realizar por este proyecto.

• Capítulo 2: Muestra una visión general del estado actual de las diferentes tecnologías de estudio de este proyecto.

• Capítulo 3: Explica y describe los diferentes diseños que se van a realizar en este proyecto.

• Capítulo 4: Explica y describe las diferentes implementaciones que se van a tratar de estudiar.

• Capítulo 5: Describe las pruebas de diferentes experimentos realizados mediante los diferentes entornos implementados en este proyecto. Así mismo también se describen y analizan los resultados obtenidos en cada uno de los experimentos.

• Capítulo 6: Se valoran los resultados obtenidos para argumentar la conclusión obtenida en él proyecto. Se realiza una pequeña reflexión sobre los diferentes caminos y trabajos futuros que se abren tras la realización de este proyecto.

• Anexos:

  – Anexo A: Descripción de la planificación temporal del proyecto.

  – Anexo B: Presentación del presupuesto de ejecución del proyecto.

2 Estado del Arte

2.1 SDN

Las redes definidas por software (SDN)[9], establecen una arquitectura de red cuyo objetivo es facilitar a los administradores la gestión de los servicios de red a través de la abstracción de las funcionalidades de bajo nivel.

Por lo tanto, podría decirse que el objetivo de SDN es facilitar el desarrollo e implementación de servicios de red abstrayendo al administrador de gestionar dichos servicios a bajo nivel.

2.1.1 Infraestructura

SDN[10] elimina la "inteligencia" establecida en los nodos de las redes tradicionales delegando la capacidad de toma de decisiones en el controlador, estableciendo tres capas desacopladas: capa de aplicación, capa de control y capa de infraestructura.

Una representación visual de la arquitectura SDN se muestra en la Ilustración 8: Arquitectura SDN.

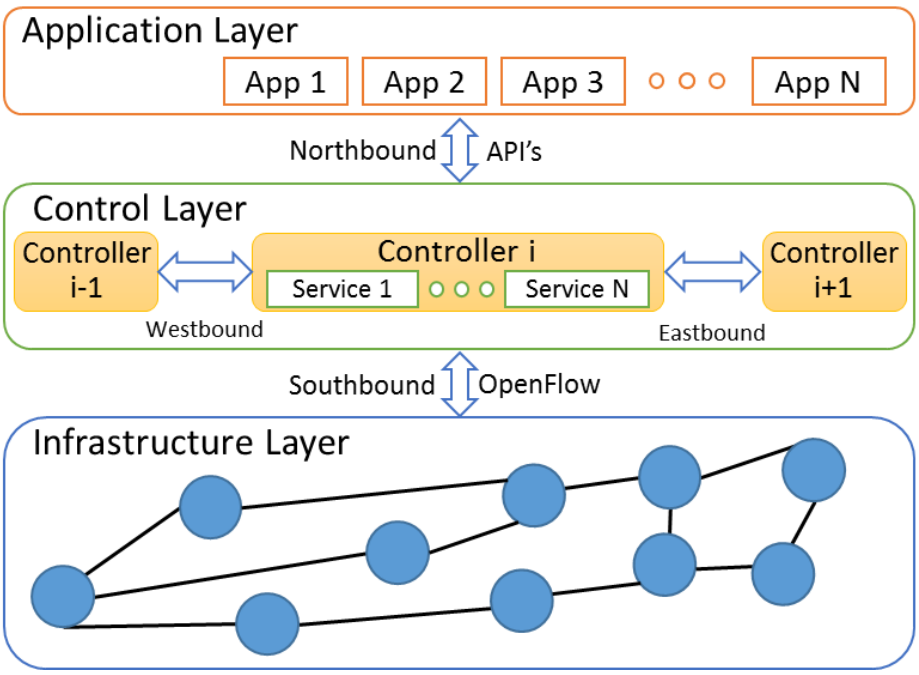


Ilustración 8: Arquitectura SDN

2.1.1.1 Capa de aplicación

La capa de aplicación[1] es la capa de más alto nivel y desde la cual el programador o administrador de la red estable las diferentes reglas o comportamientos de la red. Esta capa no es capaz de comunicarse directamente con la capa de infraestructura. Para ello, hace uso de un "middleware" denominado controlador, el cual se encuentra en la capa de control. La arquitectura SDN estable una interfaz denominada "Northbound API" para comunicar las aplicaciones con la capa de control.

2.1.1.2 Capa de control

Esta capa se puede definir como un middleware, ya que facilita la interconexión entre la capa de aplicación y la capa de infraestructura. A esta capa se le conoce comúnmente como controlador, y gracias a los protocolos de comunicación (principalmente OpenFlow), permite configurar, gestionar y orquestar a la capa de infraestructura según ha sido indicado por el administrador de la red.

2.1.1.3 Capa de Infraestructura

Es la capa propiamente de hardware. Está compuesta por los diferentes nodos multicapa SDN[9]. Es importante[10] destacar que los componentes de esta capa son únicamente los diferentes dispositivos de red o switches, es decir, únicamente se consideran elementos de esta capa todos aquellos dispositivos físicos capaces de interactuar con el protocolo OpenFlow[2].

Otra propiedad importante de esta capa es la heterogeneidad del hardware, es decir, no es necesario que los nodos o hardware de la red sean de un único fabricante, tan solo cada nodo deberá contar con el soporte del protocolo de comunicación que les conecta con el controlador.

2.1.1.4 Interfaces de conexión entre capas

2.1.1.4.1 Southbound API

Se trata de la interfaz de interconexión entre la capa de control y la capa de infraestructura. Dicha interfaz se encarga de traducir a un lenguaje inteligible para los switches SDN los diferentes mensajes definidos por el administrador en la capa de aplicación.

Dichos mensajes no solo son información necesaria para el reenvío de los paquetes, sino también puede tratarse de información propia de los switches solicitada por alguna aplicación activa en el controlador.

2.1.1.4.2 Northbound API

Estas interfaces son las más críticas en las redes SDN[10], debido a que soportan una gran variedad de servicios y aplicaciones por encima, su principal función es la de interconectar la capa de aplicación con la capa de controlador.

2.1.1.4.3 Westbound and Eastbound API

Son interfaces de uso exclusivo para el controlador, no son propias del estándar SDN, pero si del estándar SDNi[11], el cual establece la posibilidad de poder comunicarse los controladores entre sí, para así poder solucionar el problema de escalabilidad en SDN.

2.2 Limitaciones de las arquitecturas SDN actuales.

La arquitectura SDN (Software-Defined Networking) ofrece numerosas ventajas, como el control de flujo con granularidad fina, la configuración múltiple de los switches (evitando así la necesidad de configurar cada switch por separado), la heterogeneidad del hardware en la red, la coexistencia entre redes productivas y experimentales o la programación de la red. Este tipo de arquitectura se ha extendido a redes de alto rendimiento, como redes de centros de datos (DevoFlow[12], Hedera[13] y PAST[14]).

Por el contrario, este tipo de propuestas SDN suponen un gasto excesivo, puesto que la necesidad de control total por parte del controlador genera una sobrecarga en el controlador, provocando cuellos de botella en la red.

Como solución, Kandoo[15] propone distribuir la carga entre múltiples controladores para resolver los cuellos de botella en el lado del controlador. Otras propuestas distribuyen parte del estado de la red entre los conmutadores, como en DevoFlow[12], OpenState [16], o Difane[17].

Otra limitación viene producida por el gran número de flujos que es capaz de recibir un switch. Este gran número de flujos genera en los switches un grandísimo número de entradas en sus tablas. Estos tamaños de tablas son imposibles de introducir en los TCAM actuales. Por lo tanto, el uso de SRAMs, que no tienen las restricciones de TCAMS, se ha propuesto para mejorar la escalabilidad de este tipo de redes, como en PAST[14].

Por último, otra limitación de las redes puramente SDN es el problema que genera el uso del mecanismo de LLDP[18] para la supervisión de los enlaces de una red. Se ha comprobado[19] que dicho mecanismo, en primer lugar, no escala correctamente, debido a la sobrecarga producida en el controlador. Esto provoca que las recuperaciones en este tipo de redes sean más lentas de lo realmente deseable.

OpenFlow[20] es un protocolo de comunicaciones entre la capa de control y la capa de infraestructura en la arquitectura SDN, el cual permite el acceso, manipulación y gestión de las tablas internas de los dispositivos de red.

2.2.1 Funcionamiento OpenFlow

OpenFlow[21], como se ha comentado anteriormente, permite comunicar de forma sencilla la capa de control con la capa de infraestructura a través de la interfaz Southbound API. Esta comunicación se hace a través de un canal seguro.

El switch y el controlador pueden comunicarse a través de una conexión TLS. Dicha conexión se inicia desde el switch OpenFlow, estableciéndose por defecto a través del puerto TCP 6633, aunque puede ser modificado sin mayor problema.

2.2.2 Switch OpenFlow

Uno de los conceptos principales en el protocolo de OpenFlow es "Datapath" o camino de datos. Este concepto consiste simplemente en una tabla de flujo y una acción asociada con cada entrada de flujo.

Una vez comprendido ese concepto se puede pasar a definir un switch OpenFlow. Este concepto está compuesto por 3 partes:

- La primera, una Tabla de Flujos, donde se encuentra una acción asociada para cada entrada, la cual le indica al switch como procesar cada paquete;
- La segunda, un Canal Seguro que conecta al switch con un proceso remoto que recibe el nombre de "controlador".
- La tercera, una Tabla de Grupos, donde se almacena información, así como parámetros y acciones asignadas a los diferentes grupos.

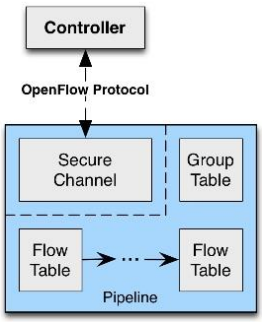
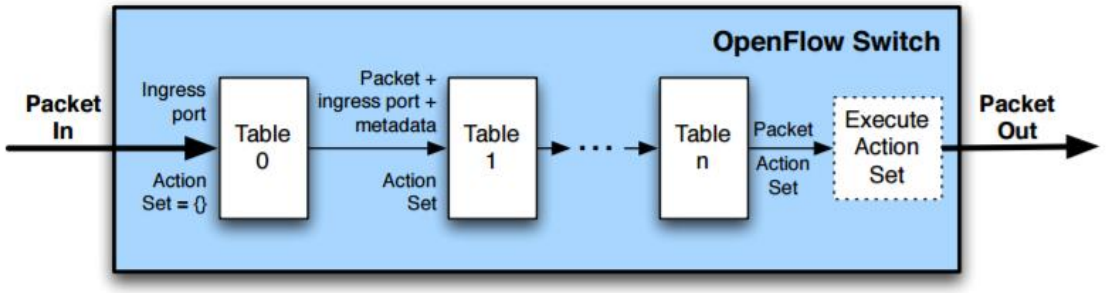


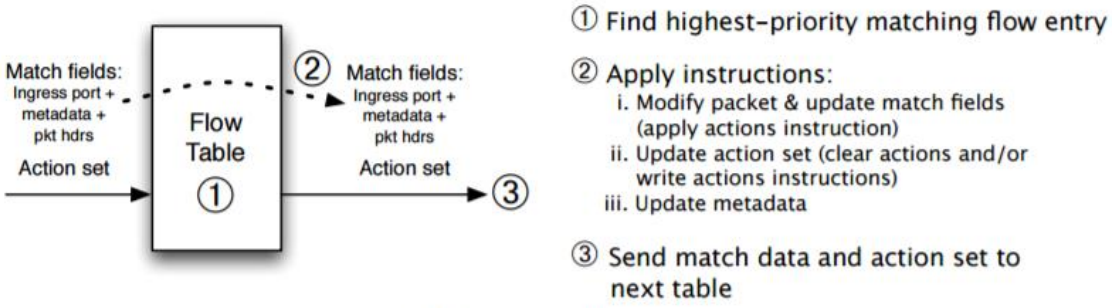
Ilustración 9: Funcionamiento del canal OpenFlow[2]

2.2.3 OpenFlow Pipeline

Un punto fundamental de los switches OpenFlow es su Pipeline, el cual establece el mecanismo de procesamiento que va a recibir cada uno de los paquetes que recibe el switch. Cada Pipeline posee un conjunto de tablas de flujos que a su vez poseen múltiples entradas que definen las acciones para cada flujo. Esto se muestra en la Ilustración 10: OpenFlow pipeline[2]



(a) Packets are matched against multiple tables in the pipeline



(b) Per-table packet processing

Ilustración 10: OpenFlow pipeline[2]

Por lo tanto, como se ha podido observar, el procesamiento establecido se puede dividir en pasos que se repiten de forma iterativa, es decir, primero se realiza una comparación entre el paquete y la primera tabla. Si existe algún registro se obtiene la acción asociada y se almacena, y posteriormente se pasa a la siguiente tabla para realizar el mismo procedimiento. Este proceso se realiza con todas las tablas restantes. Una vez se tienen todas las acciones se pasa al módulo de ejecución de acciones, para de esta manera despachar dicho paquete.

2.2.4 Tablas OpenFlow

El protocolo OpenFlow, establece la utilización de tablas para el almacenaje de los diferentes datos. Todas las tablas definidas por OpenFlow se pueden resumir de la siguiente manera.

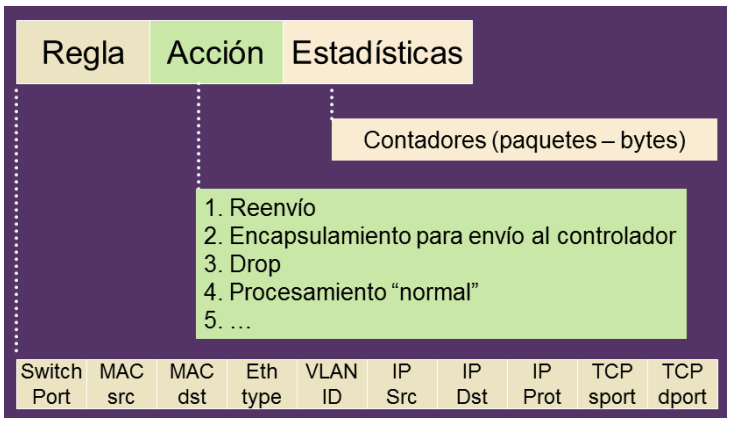


Ilustración 11: Resumen general de tablas OpenFlow[10]

Estas tablas pueden ser agrupadas en tres grandes grupos:

- *Tabla de Flujos:* En estas tablas se almacena toda la información relacionada con el procesamiento de los diferentes paquetes o flujos que pasan por el switch.
-

Tabla 1: Definición de Tabla de Flujos OpenFlow[2]

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

- *Tabla de Grupos:* En esta tabla se establece los diferentes conjuntos de elementos que pueden realizar la misma acción. Son de gran utilidad a la hora de implementar reglas para tráfico multicast, aunque también se pueden usar para tráfico unicast con características similares.
-

Tabla 2: Definición tabla de Grupos OpenFlow[2]

Group Identifier	Group Type	Counters	Action Buckets

- *Tabla de Métricas:* En esta tabla se obtienen los datos necesarios para el control de paquetes. Su función principal es facilitar la creación, control, configuración de los diferentes servicios de red que permiten generar calidad de servicio, como por ejemplo QoS.
-

Tabla 3: Definición de tabla "Meter"[2]

Meter Identifier	Meter Bands	Counters

2.2.4.1 Tablas de Flujos

El paradigma de la computación de paquetes se basa en la capacidad de un elemento para transmitir los diferentes paquetes que llegan. Para ello, OpenFlow hace uso de un recurso encargado de almacenar los parámetros, contadores, campos de representación e incluso las diferentes instrucciones que se deben aplicar a los diferentes paquetes.

Tabla 4: Ejemplo tabla de Flujos OpenFlow[22]

Campos de la cabecera	Contadores	Acciones	Priority
If ingress port == 2		Drop packet	32768
if IP_addr == 129.79.1.1		re-write to 10.0.1.1, forward port 3	32768
if Eth Addr == 00:45:23		add VLAN id 110, forward port 2	32768
if ingress port == 4		forward port 5, 6	32768
if Eth Type == ARP		forward CONTROLLER	32768
If ingress port == 2 && Eth Type == ARP		forward NORMAL	40000

Para esta función, OpenFlow define que cada tabla de flujo debe contener los siguientes parámetros:

- *Match Fields o Campos de coincidencias*: Estos parámetros son los encargados de almacenar parámetros representativos de los paquetes, tales como el puerto de entrada, cabeceras y, opcionalmente, metadatos, los cuales deben seguir un determinado patrón definido por el protocolo OpenFlow.

- *Contadores*: Son un parámetro de medida estadística referente al cómputo de paquetes.
- *Instrucciones*: Establece las acciones o procesamientos permitidos para cada paquete.

2.2.4.2 Tabla de Grupo:

La agrupación de elementos permite a la mayoría de los sistemas ahorrar memoria y recursos. OpenFlow, siguiendo este modelo, establece un conjunto de tablas donde se puede almacenar las características de los diferentes grupos, así como los diferentes parámetros que definen dichos grupos.

Tabla 5: Ejemplo tabla de grupos OpenFlow[20]

Group Identifier	Group Type	Counters	Action Buckets
5			Out port x,y
15			Out port a
6			Group ID 5
7			Out port m, Group ID 15
9			Drop Packet

Cada entrada de grupo contiene lo siguiente:

- *Identificador de Grupo*: Se trata de un identificador numérico que determina de forma unívoca a cada grupo.
- *Tipo de Grupo*: Determina el Tipo del grupo. Dichos Tipos pueden ser:
    - Todos: Ejecutan todas las acciones dentro del grupo. Este grupo se utiliza para transmisión multicast o broadcast.
    - Seleccionado: Ejecuta una acción del grupo.
    - Indirecto: Ejecuta una acción definida en este grupo.
    - Conmutación por error rápida: Ejecuta el primer cubo "vivo" (activo).
- *Contadores*: Al igual que en las tablas anteriores, son datos estadísticos.
- *Acciones asociadas:* Como su propio nombre indica son el conjunto de listas de acciones y parámetros definidos para dicho grupo.

2.2.4.3 Ejemplo de datos en tablas

Como se ha explicado anteriormente, para el reenvío de los diferentes paquetes por las redes SDN, los switches deben tener en sus tablas algún registro que establezca la acción a realizar. En la siguiente imagen se muestran varios ejemplos de dichas reglas.

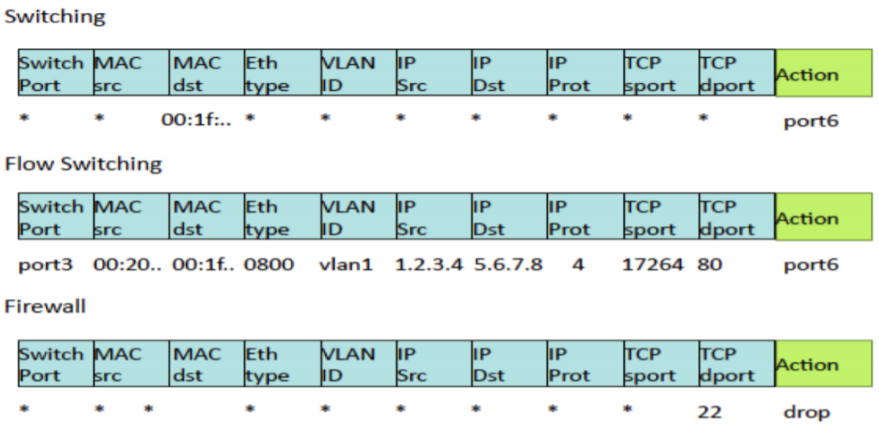


Ilustración 12: Ejemplo de reglas en tablas OpenFlow según el tipo de switch implementado en controlador[22]

2.2.5 Tipos de Mensajes

2.2.5.1 Mensajes Controlador a Switch

Son mensajes iniciados por el controlador. Este tipo de mensajes tienen como principal función la verificación del estado de la configuración del switch. Pueden ser o no del tipo petición-respuesta, es decir, puede tener o no respuesta por parte del nodo de red.

Existen diferentes tipos[23]

- Features: Este tipo de mensajes tienen la función de establecer la conexión entre el switch y el controlador. Estos mensajes tienen como función básica la obtención de las características hardware del nodo de red. Son mensajes del tipo petición-respuesta.
- Configuration: Este tipo de mensajes son para configurar o consultar determinados parámetros hardware del nodo. Al igual que el caso anterior, también son del tipo petición-respuesta.
- Modify-State: Son mensajes cuya principal función es insertar, eliminar o modificar los flujos en las tablas OpenFlow del nodo.

- Read-State: Se trata de mensajes de obtención de datos y estadísticas. Dichos datos contienen información acerca de flujos, puertos, entradas de tabla, etc.
- Send-Packet: Es usado por el controlador para indicarle al switch el puerto de salida que ha de utilizar para reenviar el paquete asociado a la comunicación entre ambos.
- Barrier: Mensaje de comprobación por parte del controlador, cuya función principal es asegurarse que las dependencias de un mensaje han sido cumplidas.
- Role-Request: Mensaje de establecimiento de canal OpenFlow.

2.2.5.2 Mensajes Asíncronos

Este tipo de mensajes, a diferencia del caso anterior, es iniciado por el switch. Su principal función es la generación de eventos de la red, es decir, notificar cambios en la red, cambios de estado del switch, errores, etc...

Los mensajes principales de este tipo son [23]:

- Packet-in: Se trata de un paquete cuya funcionalidad es la de informar al controlador de la entrada de un paquete desconocido en el nodo.
- Flow-Removed: Se trata de un paquete que informa al controlador sobre la inactividad de un flujo, o vencimiento del TTL del mismo. Por otra parte citar que este tipo de paquete puede ser enviado por el controlador o por el Switch indistintamente.
- Port-Status: Se trata de un mensaje que se envía en el momento en que el nodo de red detecta un cambio en la configuración de uno de sus puertos.
- Error: Mensaje de notificación de cualquier error distintos al resto de mensajes anteriores.

Un ejemplo de intercambio de mensajes asimétricos se muestra en la Ilustración 13: Intercambio mensajes asimétricos

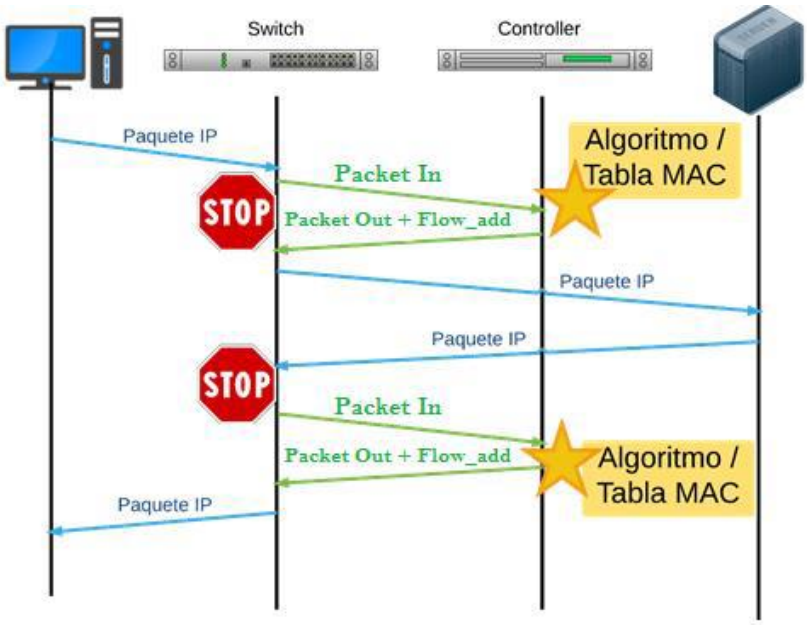


Ilustración 13: Intercambio mensajes asimétricos

2.2.5.3 Mensajes Simétricos

Al igual que los mensajes anteriores, iniciados por el switch, además en este caso también pueden ser iniciados por el controlador. Se trata principalmente de mensajes utilizados al iniciar un switch en una nueva red. Su principal función es indicar al controlador las características funcionales del nuevo dispositivo.

Los mensajes principales de este tipo son[23]:

- Hello: Son mensajes de inicio de conexión entre controlador y switch. Su principal función es la negociación de la versión del protocolo OpenFlow.
- Echo: Es un mensaje del tipo petición-respuesta cuya función es la de verificar parámetros tales como ancho de banda, latencia, así como la de determinar el estado de actividad del nodo.
- Vendor: Permite dar funcionalidades extra al Switch. Este mensaje se usará principalmente en futuras versiones de OpenFlow.

Un ejemplo de intercambio de mensajes simétrico puede ser el representado por la Ilustración 14: Intercambio de mensajes simétricos

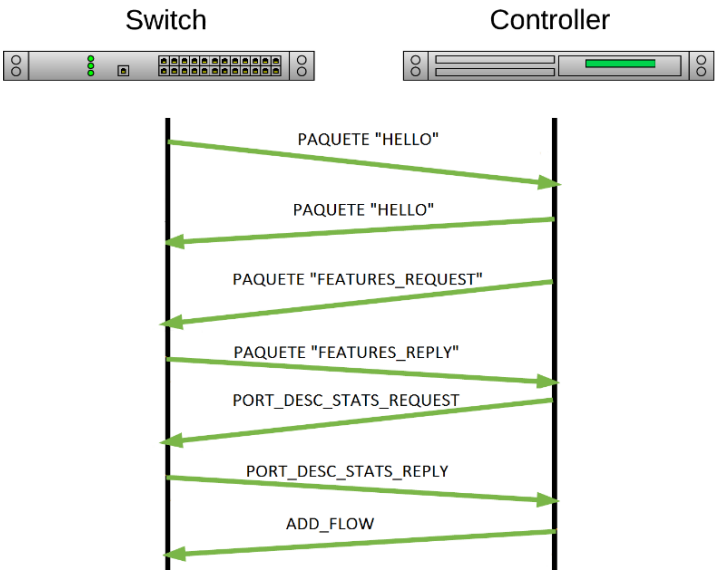


Ilustración 14: Intercambio de mensajes simétricos

2.3 Controlador

El controlador se puede considerar un middleware, que permite al administrador de la red abstraerse de la dificultad propia de la red. Este software se sitúa en la capa de control, el cual centraliza el control de la red. Para ello utiliza el protocolo OpenFlow[2] como lenguaje de comunicación con los diferentes nodos de la red.

Como se ha explicado anteriormente, el controlador se encarga de introducir, gestionar y controlar los nodos de la red a través de reglas. Dichas reglas se pueden introducir siguiendo dos pautas diferentes[24] :

- Proactiva o Preinstalación de reglas: El controlador instala las reglas necesarias antes de que un paquete llegue al switch, consiguiendo reducir los tiempos de retardo, pero como contraprestación el controlador debe conocer el tipo de tráfico que circula por la red a priori.
- Reactiva: Cuando switch OpenFlow recibe un paquete que no conoce, éste lo envía al controlador mediante un mensaje del tipo "Packet-in".
  El controlador es quien, aplicando las diferentes reglas definidas en su capa de aplicación, decide cómo se debe procesar y encaminar dicho paquete. Posteriormente dicha decisión se comunica mediante los diferentes paquetes OpenFlow (FlowMod/Packet-Out) al switch para que el nodo la realice.

2.3.1 Tipos de controlador

Existen múltiples controladores en el mercado actual. En la siguiente tabla se muestra algunas posibilidades

Tabla 6: Tabla de Controladores[24]

	NOX	POX	RYU	ODL	ONOS
Soporte OpenFlow	v1.0	v1.0	v1.0, v1.2, v1.3	v1.0, v1.2, v1.3	v1.0, v1.2, v1.3
Lenguaje	C++	Python	Python	Java	Java
Plataforma	Linux	Linux, Mac Os, Windows	Linux	Linux	Linux
Soporte OpenStack	No	No	Sí	Sí	Sí
Multiproceso	Sí	No	No	Sí	Sí
Código abierto	Sí	Sí	Sí	Sí	Sí
API REST	No	No	Sí	Sí	Sí

2.3.2 Ryu

Ryu[25] se trata de un controlador diseñado como una aplicación monoproceso bajo el lenguaje Python. Este diseño monolítico trae consigo un problema de rendimiento, pues al no disponer de una arquitectura multiproceso no permite de forma natural la generación aplicaciones paralelas, lo que provoca que su ejecución sea lineal. Este tipo de arquitectura, por el contrario, simplifica mucho la implementación de aplicaciones y prototipos que no requieran un alto rendimiento.

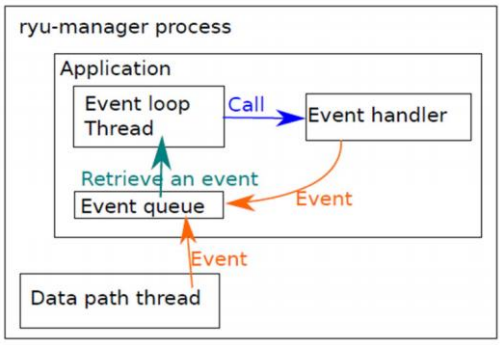


Ilustración 15: Manejador de procesos Ryu[26]

Ryu[25] implementa varias funcionalidades, como por ejemplo manejadores de procesos, visores de topología e incluso generadores de paquetes.

Un detalle importante que se debe tener en cuenta a la hora de diseñar cualquier tipo de aplicación para este controlador es la manera en que Ryu, a través de su API[27], gestiona el tratamiento de los diferentes mensajes OpenFlow. Cada aplicación[27] en Ryu recibe eventos a través de una cola FIFO, preservando de este modo el orden de llegada de los diferentes mensajes. Esta manera de gestionar los mensajes Ryu los define como Manejadores de Eventos

Existen diferentes tipos[27]:

2.3.2.1 Observador de Eventos

Ryu puede escuchar, registrar y tratar eventos externos mediante la librería "ryu.controller.handler.set_ev_cls[27]"

2.3.2.2 Generador de Eventos

Una aplicación Ryu[27] puede generar eventos llamando a los métodos apropiados de ryu.base.app_manager.RyuApp como send_event o send_event_to_observers.

2.3.2.3 Tipos de Eventos OpenFlow

El módulo "ryu.controller.ofp_event"[27] maneja los eventos que producen las diferentes recepciones de mensajes OpenFlow. Los diferentes manejadores de eventos se denominan "ryu.controller.ofp_event.EventOFPxxxx"[27] donde "xxxx" es el nombre del mensaje OpenFlow correspondiente. Por ejemplo, "EventOFPPacketIn" sería el manejador del paquete PacketIn enviado por el nodo de red.

El software de OpenFlow implementado en Ryu descodifica los mensajes de OpenFlow recibidos y los envía por medio de la interfaz correspondiente (Northbound) a las

diferentes aplicaciones que en su código posean el manejador adecuado indicado mediante la sentencia "ryu.controller.handler.set_ev_cls[27]".

2.3.2.4 Web Topology

Una de las propiedades más versátiles de los controladores es la gestión y visualización de la topología de la red. Como es de esperar RYU también lo contempla.

Para hacer uso de esta propiedad, se debe realizar la siguiente llamada:

```
PYTHONPATH=. ./bin/ryu run --observe-links ryu/app/gui_topology/gui_topology.py
```

Esto le indica al sistema dos funciones principales:

- La primera, que levante RYU ejecutando la aplicación que el diseñador desee.
- La segunda, que comience a emitir los diferentes mensajes LLDP para el posterior descubrimiento de la red.

Para acceder a la visualización, el usuario únicamente deberá introducir en el navegador:

```
http://<ip address of ryu host>:8080
```

El resultado de este comando se puede mostrar en la siguiente imagen:

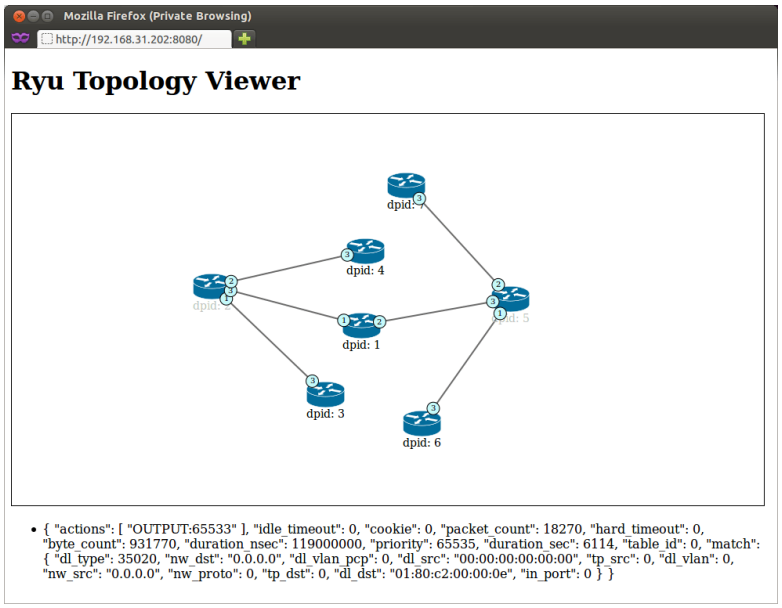


Ilustración 16: Representación de la topología por parte del controlador Ryu

2.4 MiniNet

MiniNet[28] es un emulador de red que ejecuta una colección de dispositivos finales (switches, routers y hosts) y enlaces. Para la emulación de la red, MiniNet utiliza virtualización ligera mediante CGroups[29] de Linux.

Esta virtualización permite generar una red completa en un solo sistema físico. Para ello, MiniNet ejecuta procesos independientes que emularán a los switches, generando también las interfaces de red virtuales e interconectando todo mediante enlaces virtuales generados en el sistema operativo de la máquina contenedora. Es decir, MiniNet genera todos los dispositivos que componen una red utilizando software en lugar de hardware, obteniendo en su mayor parte un comportamiento similar a los elementos reales.

Para la realización de pruebas en MiniNet existen dos maneras:

- La primera forma es a través de su interfaz, la cual se muestra en la Ilustración 17: Interfaz de usuario de MiniNet.
- La segunda manera de generar redes y pruebas en MiniNet es a través de una script en Python.

2.4.1 Interfaz

Como se puede ver en la Ilustración 17: Interfaz de usuario de MiniNet, se trata de una interfaz por línea de comando gracias a la cual se generan todos los elementos necesarios para la emulación de la red.

Ilustración 17: Interfaz de usuario de MiniNet

2.4.1.1 Comandos de la Interfaz

Para ejecutar MiniNet se deberá introducir el siguiente comando:

```
$sudo mn
```

Lo que por defecto creará una red con un switch conectado a dos hosts y al controlador de referencia de MiniNet.

MiniNet nos permite crear así mismo otros tipos de topologías predefinidas gracias al subcomando topo. Su sintaxis se muestra a continuación:

```
$sudo mn --topo=[Topologia]
```

Una vez creada la topología se podrá, a través de diferentes comandos, interactuar con el sistema. Alguno de esos comandos son los siguientes:

Para solicitar ayuda:

```
mininet> help
```

Para mostrar los nodos, equipos finales y controladores:

```
mininet> nodes
```

```
mininet> nodes
available nodes are:
c0 h1 h2 s1
```

Ilustración 18: Comando nodes

Para mostrar una descripción de la red:

```
mininet> net
```

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Ilustración 19: Comando net

```
mininet> dump
```

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=31276>
<Host h2: h2-eth0:10.0.0.2 pid=31277>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=31282>
<OVSController c0: 127.0.0.1:6633 pid=31268>
mininet>
```

Ilustración 20: Comando dump

MiniNet también permite ejecutar algunos comandos sobre un elemento en particular. Por ejemplo, si se desea obtener la información sobre la interfaz de red de un equipo final se deberá introducir el siguiente comando:

```
mininet> h1 ifconfig -a
```

Si se desea realizar un ping desde el equipo final h1 al equipo final h2, se lanzará:

```
mininet> h1 ping h2
```

Así mismo, MiniNet posee la capacidad de lanzar un ping entre todos los equipos finales de la red. Para ello se deberá introducir la siguiente sentencia:

```
mininet> pingall
```

Por otro lado, este emulador permite comprobar el rendimiento de una red. Por ejemplo, si se desea medir el ancho de banda entre dos equipos finales se deberá introducir el siguiente comando:

```
mininet> iperf h1 h2
```

Así mismo, si se desea abrir las consolas de los equipos finales, se deberá ejecutar el comando:

```
mininet> xterm h1
```

Lo que abrirá un terminal para cada nodo.

Finalmente, para cerrar la red emulada por MiniNet se deberá introducir

```
mininet> exit
```

Si MiniNet fallara o se finalizara la emulación con Ctrl+C, se deberá ejecutar el comando `sudo mn -c` para limpiar el entorno virtualizado.

2.4.2 Script Python

La segunda manera de generar redes y pruebas en MiniNet es a través de una script en Python. En ella se automatizan todos los procesos necesarios para la emulación de la red y su posterior puesta a prueba.

Dicha Script se debe implementar mediante los comandos facilitados por la API de MiniNet[30]. Algunos de los comandos más importantes se citan a continuación:

Tabla 7: Comando básicos API MiniNet Python

Comando	Funcionalidad
CLI	Permite la generación de una interfaz de usuario al iniciar la script
Intf	Permite caracterizar una interfaz de forma genérica
TCIntf	Permite caracterizar un TC en la interfaz
Link	Genera un enlace entre dos nodos
TCLink	Genera un TC en el enlace entre dos nodos cualesquiera
TCULink	Genera un TC optimizado para enlaces entre nodos del tipo User-space
MiniNet	Clase creadora de la red emulada
Host	Genera un host como nodo de red
UserSwitch	Genera un switch del tipo User-space
OVSSwitch	Genera un switch del tipo OVS
RemoteController	Permite indicar a MiniNet la existencia de un controlador externo al entorno de emulación

2.4.3 Clúster

MiniNet[31] es capaz de emular en una sola máquina física una red formada por cientos de nodos (OvS) o de decenas de nodos (OfSoftSwitch13), es decir, redes de un tamaño mediano. Sin embargo, si se desea emular redes de mayor tamaño o con unas tasas de tráfico muy elevadas, MiniNet por sí solo no dispone de los recursos necesarios para hacerlo.

Por el contrario, si se utiliza una pequeña red de servidores físicos, donde distribuir nodos y enlaces de MiniNet (aumentando así el número de recursos totales), es posible escalar

fácilmente el sistema global para soportar así redes más grandes. De esta idea nace "MiniNet Cluster".

2.4.3.1 Funcionamiento de MiniNet Cluster

Existen principalmente dos grandes diferencias entre MiniNet y MiniNet Cluster. Estas diferencias son la comunicación entre los nodos y la creación de los mismos.

- *Comunicación entre nodos*[31]*:* En MiniNet estándar no hay nodos remotos, entendiendo como remotos aquellos nodos situados en máquinas físicas diferentes. Cada enlace se crea entre dos Ethernet virtuales dentro del mismo sistema. Sin embargo, en MiniNet Cluster, sí que existen nodos remotos, por lo que el mecanismo de generación de enlaces es diferente. En este caso, los enlaces se generan gracias a túneles ssh. En el futuro, MiniNet propone agregar otros tipos de túneles adicionales, como túneles GRE.
- *Creación de nodos*[31]*:* Al iniciar la emulación, MiniNet Cluster genera una única instancia MiniNet en una de las máquinas del clúster. La máquina seleccionada es la encargada de gestionar la creación de todos los nodos de la red (nodos locales y nodos remotos). Para ello, crea los nodos locales mediante procesos en su sistema, mientras que los nodos remotos son creados a través de conexiones ssh. Los comandos que se ejecutan en cada máquina se asemejan a los mismos comandos que se ejecutan durante el inicio del nodo local, pero se ejecutan a través de una conexión ssh al servidor remoto. Esto significa que no hay demonios de nodo remoto y sólo una instancia de MiniNet que se ejecute en el clúster.

2.4.3.2 Algoritmos de emplazamiento y distribución de nodos

MiniNet Cluster posee una opción que permite al desarrollador colocar de forma automática los diferentes nodos entre los servidores físicos. Esta funcionalidad dispone de dos algoritmos diferentes:

- *Algoritmo SwitchBinPlacer*: Su principal característica es que intenta colocar todos los host unidos a un mismo nodo en la misma máquina física. El problema viene cuando las máquinas del clúster no son iguales, y al no tener todas las máquinas la misma potencia de cálculo, puede generar descompensación en las diferentes pruebas.
- *Algoritmo RandomPlacer:* Coloca de forma aleatoria los distintos nodos y host de la red. Esto lleva, en muchos casos, a la generación de túneles ssh innecesarios.

Por último se encuentra la opción de "Emplazamiento Manual", es decir, el usuario puede situar de manera arbitraria todos los elementos que componen la red y distribuirlos entre los servidores físicos.

2.5 Virtual Switch

Un virtual Switch[32] o switch virtual es un sistema software capaz de conmutar tráfico dentro de una máquina física. Este sistema es muy usado en centros de datos, puesto que en cada servidor físico se pueden encontrar cientos de máquinas virtuales, las cuales deben conectarse de alguna manera al exterior. Dicha conexión se realiza a través de este tipo de software al cual se le asignan las interfaces físicas por un extremo y por el otro se le conectan de forma virtual las diferentes máquinas contenidas en el sistema hardware. Un esquema de representación se puede observar le la Ilustración 21: Esquema de utilización de Virtual Switch

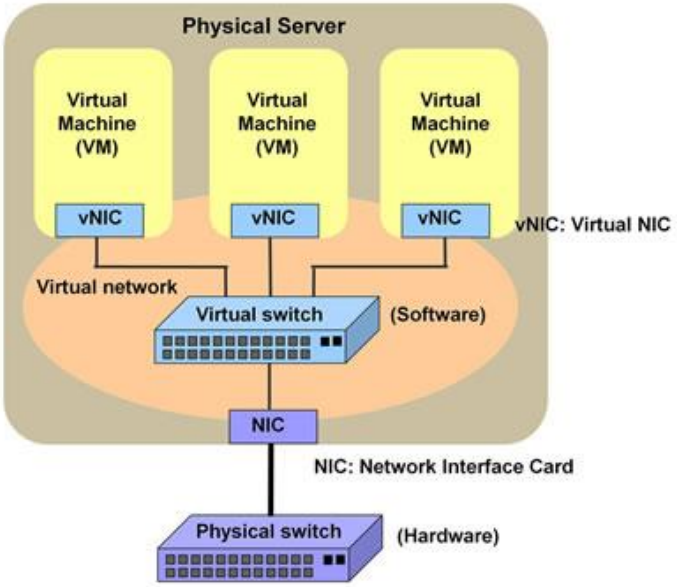


Ilustración 21: Esquema de utilización de Virtual Switch[32]

Como se puede observar dentro de una misma maquina física se encuentra N máquinas virtuales con N tarjetas de red virtuales, las cuales son conectadas al switch virtual y éste se conecta a su vez a la tarjeta de red real del servidor para dar salida a las máquinas internas.

2.5.1 Open vSwitch (OvS)

Open vSwitch[33], a partir de ahora OvS, es un switch virtual multicapa bajo licencia Open Source Apache 2.0. Este switch está diseñado para permitir la conmutación masiva de paquetes en la red a través de un sistema software, al tiempo que sigue soportando interfaces y protocolos estándar de administración (por ejemplo, NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). Está diseñado para soportar la distribución a través de varios servidores físicos similares a la vwNetwork de VMware o Cisco Nexus 1000V.

Este switch a diferencia de otros se encuentra programado en dos segmentos diferentes del sistema operativo:

- El primero se desarrolla en la sección de usuario, donde se hace la gestión del switch.
- El segundo se implementa en espacio de Kernel, donde verdaderamente se realiza la conmutación de los paquetes.

Esta forma de implementación lo convierte en un switch de altas prestaciones con una programación compleja a base de callbacks en lenguaje C.

Una representación de la estructura de programación de OvS se muestra en la Ilustración 22: Funcionamiento interno OvS.

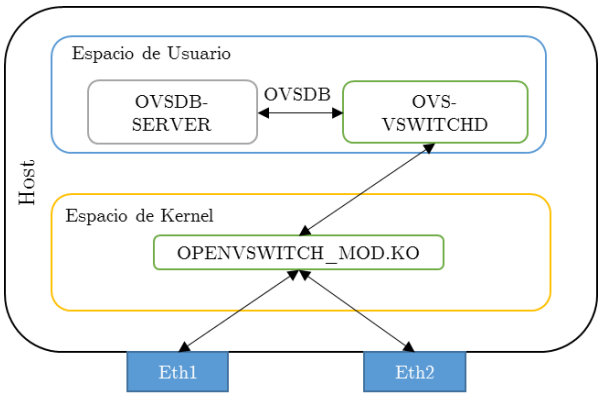


Ilustración 22: Funcionamiento interno OvS

2.5.1.1 Gestión de OvS

OvS permite su gestión mediante dos protocolos. En ambos casos, este switch necesita de una entidad que centralice la gestión de los flujos:

- OpenFlow[2]: Como se ha explicado anteriormente, es un protocolo de comunicaciones entre la capa de control y la capa de infraestructura en la arquitectura SDN.
- OVSDB (Open vSwitch Database Management Protocol)[34]: Se trata de un protocolo de gestión implementado sobre un componente propio de Open vSwitch. Dicho protocolo dispone de una licencia de código abierto y está diseñado para administrar los diferentes componentes que forman OVS, así como para introducir las reglas necesarias para la conmutación de los paquetes.

2.5.2 OfSoftSwitch

OfSoftSwitch[35] también conocido como switch CPqD por su desarrollo en este centro de investigación, se trata de un virtual switch implementado totalmente en espacio de usuario. Este switch se ha programado en lenguaje C y está confeccionado a partir de las características de un switch OpenFlow1.3.

Estas características se pueden observar claramente en la Ilustración 23: Arquitectura OfSoftSwitch basada en OpenFlow 1.3[36]

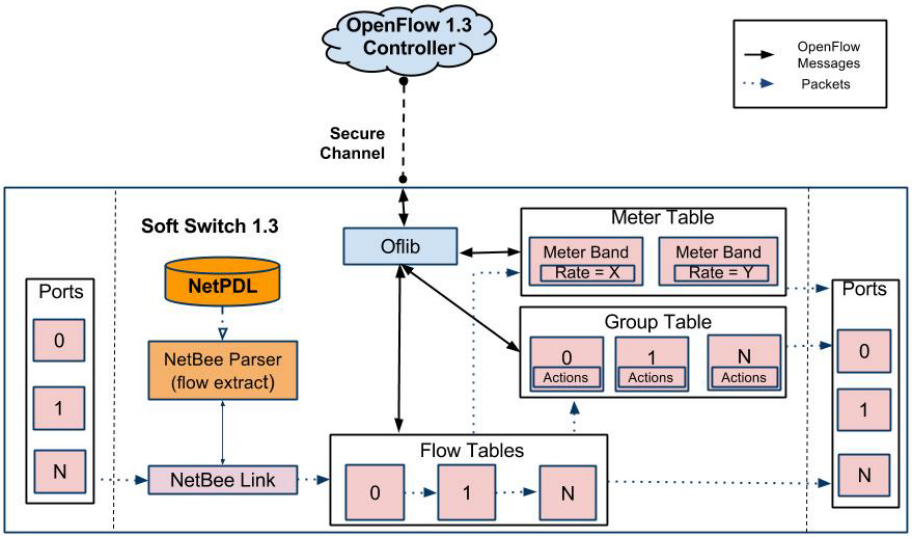


Ilustración 23: Arquitectura OfSoftSwitch basada en OpenFlow 1.3[36]

Como se ha explicado en el apartado 2.2.2, un switch OpenFlow está compuesto principalmente por tres partes: Tabla de Flujos, Canal Seguro y Tabla de Grupos.

Como se puede ver en la imagen, estas tres partes no son partes independientes entre ellas, es decir, cada una de ellas posee un enlace con las demás.

Si se analiza la figura se puede observar como el switch OfSoftSwitch13 dispone de una estructura totalmente lineal, es decir, un paquete es interceptado por uno de los puertos (representados por los puertos 0,1,..., N, parte izquierda). A continuación dicho paquete se envía a la librería NetBee, la cual tiene la función de "desmigar" el paquete, es decir, cada uno de los paquetes que se envían a esta librería son analizados para poder así rellenar las tablas de datos del OfSoftSwitch13.

A continuación, como se ha explicado en el apartado 2.2.4.1, dichos paquetes se hacen pasar por una función de comprobación, la cual deberá obtener una correlación entre el paquete y la información guardada en las tablas de flujos. De no ser así, el paquete será enviado a través del canal seguro al controlador para su análisis y posterior procesamiento.

Otra función importante es OfLib, la cual define dentro del switch todas las relaciones, tablas, acciones y apartados definidos en OpenFlow, y como se puede ver es la unidad de gestión de las diferentes tablas definidas en dicho protocolo.

Por último, en caso de encontrar una relación en las tablas de flujos, el paquete se envía junto a las diferentes acciones asociadas a los mismos hacia los puertos (Port, parte derecha de la imagen), para así, posteriormente, enviarse a través de ellos.

En este proyecto se ha escogido el OfSotfSwitch13 para la implementación del switch híbrido gracias a su fácil compresión y a su implementación lineal, la cual permite una rápida curva de aprendizaje y manipulación sencilla.

2.6 Topologías

El término topología[37] se refiere a la forma en que se enlazan los nodos de una red. Existen diferentes tipos de topologías, como son la topología en bus, en árbol, en estrella, en anillo, mallada, etc...

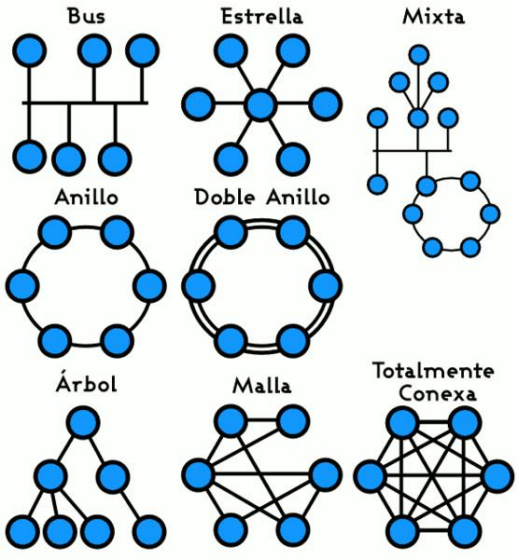


Ilustración 24: Ejemplos de Topologías

A su vez, cada una de ella se puede mezclar junto con otra para formar una topología híbrida.

Para la realización de este proyecto se han utilizado tres topologías diferentes, las cuales se explican a continuación:

2.6.1 Topología Mallada

La topología mallada[37] es un tipo de topología física que utiliza conexiones punto a punto en donde los nodos se encuentran conectados entre sí por medio de cables separados, como se muestra a continuación.

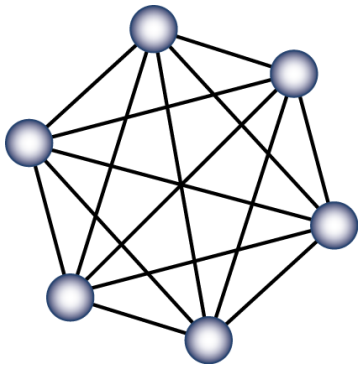


Ilustración 25: Topología Mallada completamente conectada

La topología mallada completamente conectada, al necesitar de mucho más cableado que otro tipo de topologías, resulta ser más costosa, por lo que su uso es inferior al de resto de topologías.

La implementación en este proyecto de la topología mallada será el mostrado por la Ilustración 26: Topología mallada para experimentos, la cual no conecta interconecta todos los nodos de la red entre sí.

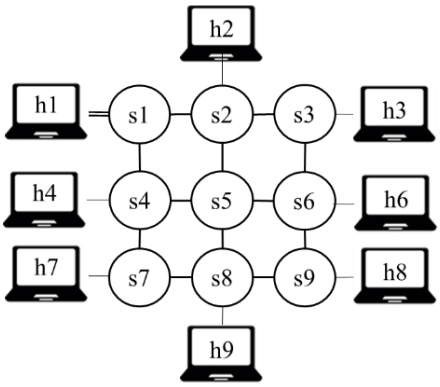


Ilustración 26: Topología mallada para experimentos

2.6.2 Topología Circular o Anillo

Topología de red[38] en la que cada estación está conectada a la siguiente y la última está conectada a la primera. Para esta topología es necesario que todos los nodos que componen la red tengan al menos dos interfaces.

Para los experimentos realizados en este proyecto se utilizará una topología circular como la representada por la Ilustración 27: Topología circular.

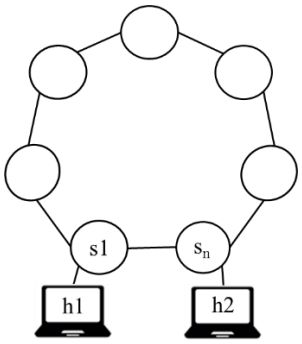


Ilustración 27: Topología circular

2.6.3 VL2

VL2[39, p. 2] se trata de un topología regular compuesta por tres capas, como se muestra en la Ilustración 28: Diseño Topología VL2[39, p. 2]

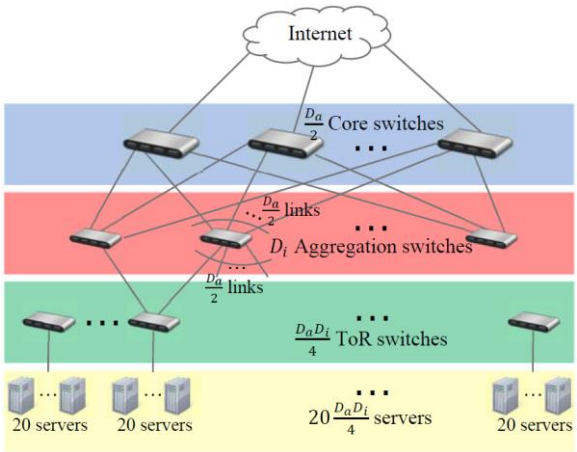


Ilustración 28: Diseño Topología VL2[39, p. 2]

Esta topología de red se caracteriza por tener diferentes velocidades en los enlaces, es decir, la velocidad en los enlaces que unen los distintos switches es 10 veces superior a la velocidad en los enlaces pertenecientes a la red de acceso.

Principalmente esta red es utilizada en centros de datos, donde suele utilizarse de dos maneras diferentes:

- La primera se propone sin sobresuscripción, es decir, se introducen 20 hosts por TOR (Top of Rack, switch situado en el superior de un armario de servidores).
- La segunda se propone con una sobresuscripción de 2 a 1, es decir, se introduce 40 hosts por TOR.

Otra propiedad de esta topología es la posibilidad de calcular el número de switches necesarios para constituirla. Si denominamos D_c al número de puertos que posee cada uno de los switches destinados para la capa "núcleo", y D_a para el número de puertos que posee cada uno de los switches destinados para la capa de agregación, se obtienen las siguientes expresiones gracias a las cuales se obtendrá el número de switches necesarios para generar la topología:

Número de switches para la capa de núcleo:

$$Num_{Cores} = \frac{D_c}{2}$$

Número de switches para la capa de agregación:

$$Num_{Agg} = D_a$$

Número de switches destinados a la capa de acceso:

$$Num_{Tor} = \frac{(D_a * D_c)}{4}$$

Para los experimentos realizados en este proyecto, se hará uso de la topología vl2 representada en la Ilustración 29: Topología VL2.

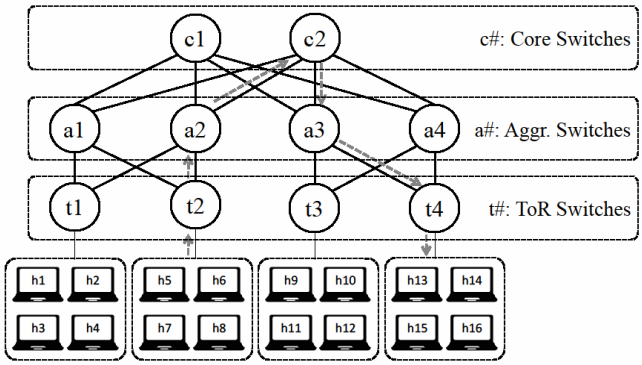


Ilustración 29: Topología VL2

2.7 Protocolos de encaminamiento de nivel 2

2.7.1 Spanning Tree Protocol (STP)

El protocolo STP[40] (Spanning Tree Protocol, protocolo del árbol de expansión) es un protocolo de capa dos que se ejecuta en bridges y switches. La especificación para STP se denomina IEEE 802.1D, cuyo objetivo es la generación de un árbol para la difusión de las tramas que impida la creación de bucles en redes con caminos redundantes.

2.7.1.1 Funcionamiento

Para comenzar se debe seleccionar un switch raíz[40]. Este proceso comienza al iniciar el sistema, los switches que componen la red, seleccionan el switch raíz. Para ello, se realiza el envío de un BPDU[41] generado por cada switch y que envía a los switches vecinos. Estos, una vez recibidas las BPDU, comparan dicha BPDU con la BPDU que ellos han enviado. El switch raíz es aquel cuya ID es la más baja y se determina para cada VLAN[42]

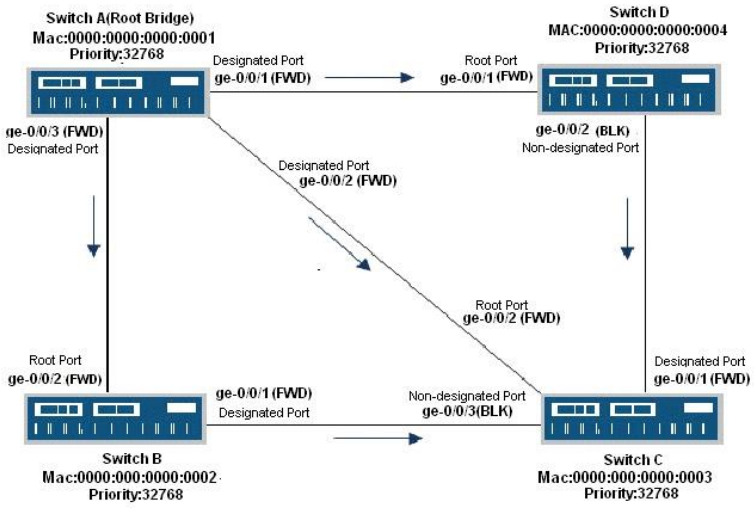


Ilustración 30: Funcionamiento STP[43]

Posteriormente, una vez elegido el switch raíz, cada switch elige el mejor camino para llegar a él. Para esta decisión los switches utilizan las BPDU[41] que han ido recibiendo por cada puerto. Seleccionan, para llegar al switch raíz, el puerto de salida que posea menor cantidad de información en la BPDU[41] recibida. En este momento el switch establece ese puerto en modo reenvío y el resto en modo bloqueo. Este proceso se repite hasta generar el árbol.

2.7.1.2 Versiones

Así mismo, se pueden encontrar distintas versiones de STP. Como resumen, se puede observar la siguiente tabla, donde además se añaden algunas de sus principales características:

Tabla 8: Tabla resumen de distintas versiones de STP[44]

Propiedad de Cisco	PVST[45] - Utiliza el protocolo de enlace Troncal ISL (Propiedad de Cisco) - Cada VLAN cuenta con una instancia de Spanning-Tree. - Capacidad de balancear la carga de tráfico en Capa 2. - Incluye las extensiones BackboneFast[46], UplinkFast[47] y PortFast[48].
	PVST[49] - Admite ISL y enlace troncal IEEE 802.1Q. - Admite las extensiones de STP propietarias de Cisco. - Agrega mejoras en la protección de BPDU en la protección de raíz.
	PVST+ Rápido - Basado en el estándar IEEE 802.1w. - Posee convergencia más veloz que 802.1D.
Estándar IEEE	RSTP[50] - Brinda una convergencia más veloz que STP. - Implementa versiones genéricas de las extensiones de STP. - IEEE incorporó RSTP dentro de 802.1D.
	MSTP[51] - Pueden asignarse varias VLAN a una misma instancia de Spanning-Tree. - Inspirado en el protocolo Spanning-Tree de múltiples instancias (MISTP). - IEEE 802.1Q-2003 ahora incluye MSTP.

2.7.2 Shortest Path Bridging (SPB)

SPB[52][53] permite establecer un sistema de caminos para transportar tramas a nivel de enlace libre de bucles con total independencia de la topología física. En dicha red SPB también utiliza un protocolo de enrutamiento de tipo Estado de Enlaces[54] para el enrutamiento de los paquetes.

SPB[52][53], por lo tanto, une dos elementos importantes dentro de las redes. Por un lado, el encapsulamiento MAC-in-MAC de los paquetes de usuario (sistemas de nivel 2), y por otro, una variante del protocolo de enrutamiento IS-IS[55](sistema de nivel 3).

Las tramas que entran en la red SPB[52], se encapsulan con las direcciones MAC del switch SPB origen y la del switch SPB[52] destino. Mientras que el protocolo de encaminamiento IS-IS, por su parte, es el encargado de decidir por qué camino óptimo entre switches debe encaminarse dicho flujo.

2.7.2.1 Características

Las características más destacables son las siguientes[56]:

- Escalabilidad.
- Encaminamiento multicamino determinista.
- Aprendizaje de MAC's solamente en el borde de la red.
- Convergencia rápida (rango de 100 msec).
- Soporta otros estándares 802.1.
- Connectivity Fault Management (802.1ag CFM)[57].
- Edge Virtual Bridging (802.1Qbg EVB).
- Compatible regiones MSTP[51], RSTP, VLANs 802.1Q.
- Soporta servicios Metro Ethernet Forum (MEF).

2.7.2.2 Desventajas

- Alta complejidad para su implantación
- Altísimo número de entradas en memoria para los switches debido al alto número de árboles que genera para la propagación de paquetes.

2.7.3 TRILL: Rbridges

TRansparent Interconnection of Lots of Links (TRILL)[58], [59] permite gestionar los posibles bucles de la red de una manera diferentes a STP, ya que TRILL maximiza el uso de todos los enlaces, provocando de manera innata un balanceo de carga, permitiendo, además usar características propias de los algoritmos de routing, lo que produce una hibridación en la forma de encaminar los paquetes entre los niveles dos y tres.

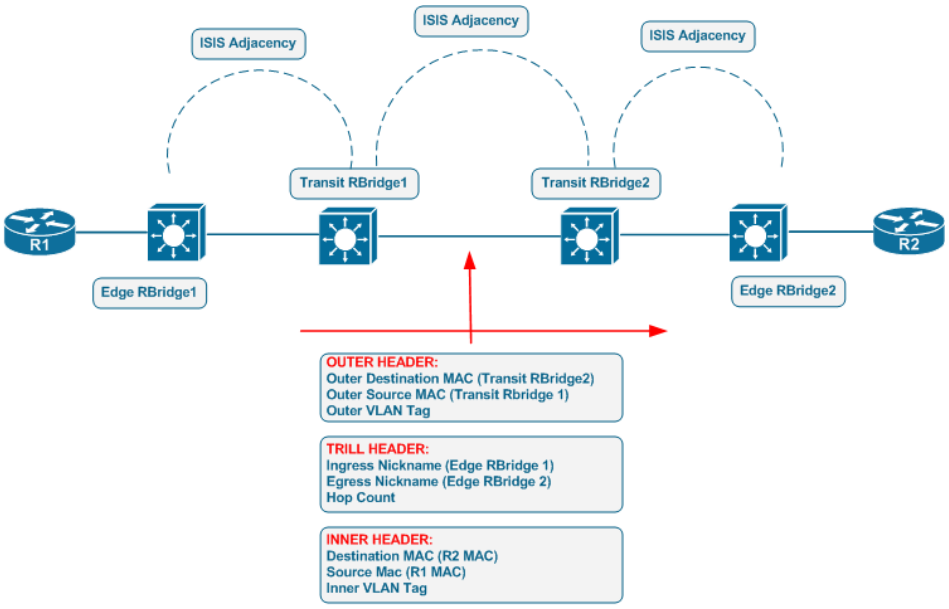


Ilustración 31: Representación red TRILL[43]

Por lo tanto se puede decir que el punto fuerte de TRILL es que mezcla las características de los niveles dos y tres.

TRILL utiliza una modificación del protocolo IS-IS[55] (enrutamiento de nivel tres), para realizar el encaminamiento de paquetes a nivel dos en la red. Cada uno de los equipos que se encuentra dentro de la red TRILL recibe el nombre de RBridge (Routing Bridge) y el mismo protocolo permite incluir enlaces de diferentes niveles físicos dentro de su red, tales como: Ethernet, Wireless Ethernet o Power Line Communications (PLC)[60], haciéndolos transparentes a los dispositivos externos a dicha red.

2.7.4 ARP-Path

ARP-Path[4][37] es una evolución natural de los "learning switches[61]". Estos switches se caracterizan por no necesitar de un protocolo de encaminamiento de nivel dos, como los anteriormente explicados, para establecer las rutas entre host. Este protocolo (ARP-Path) aprovecha la necesidad de los host finales de usar el protocolo ARP para conocer a sus vecinos, es decir, establece un camino de mínima latencia entre dos host al mismo tiempo que se envía el paquete estándar ARP-Request. Posteriormente, este protocolo "confirma" el camino marcado utilizando el paquete ARP-Reply de respuesta del host destino.

2.7.4.1 Generación de caminos

La generación de caminos de este protocolo se puede explicar en dos fases diferentes:

La primera se encuentra en el proceso de propagación de los paquetes ARP-Request. En esta fase los switches aprenden la MAC del host origen asignándoles en su tabla ARP-Table el puerto de entrada de dicho paquete y una marca de tiempo. Posteriormente, el switch ARP-Path reenvía el paquete por todos sus puertos menos por el puerto de entrada.

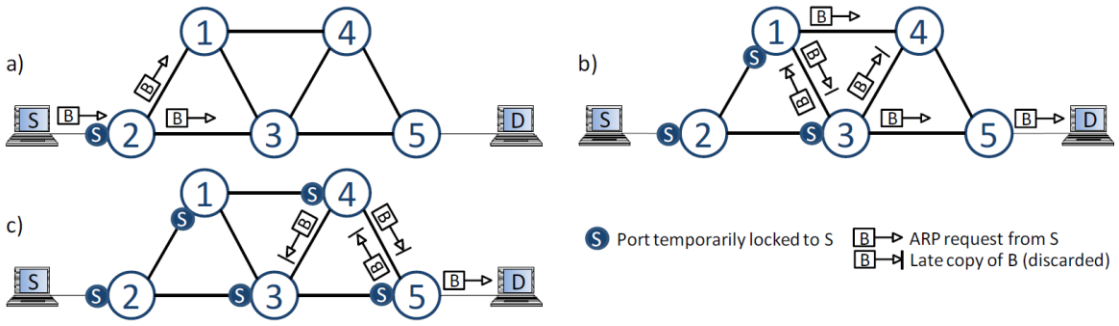


Ilustración 32: Propagación del paquete ARP-Request para la generación de caminos en ARP-Path[53]

La segunda fase se encuentra en el proceso de propagación de los ARP-Replay. Una vez el switch ARP-Path recibe uno de estos paquetes debe realizar dos pasos: el primero, es guardar en su tabla ARP-Table la MAC del host origen asociada al puerto de entrada y una marca temporal. En segundo lugar, buscar en dicha tabla la MAC destino para así obtener el puerto que debe utilizar para reenviar el paquete. Una vez localizado actualiza la marca de tiempo y reenvía el paquete.

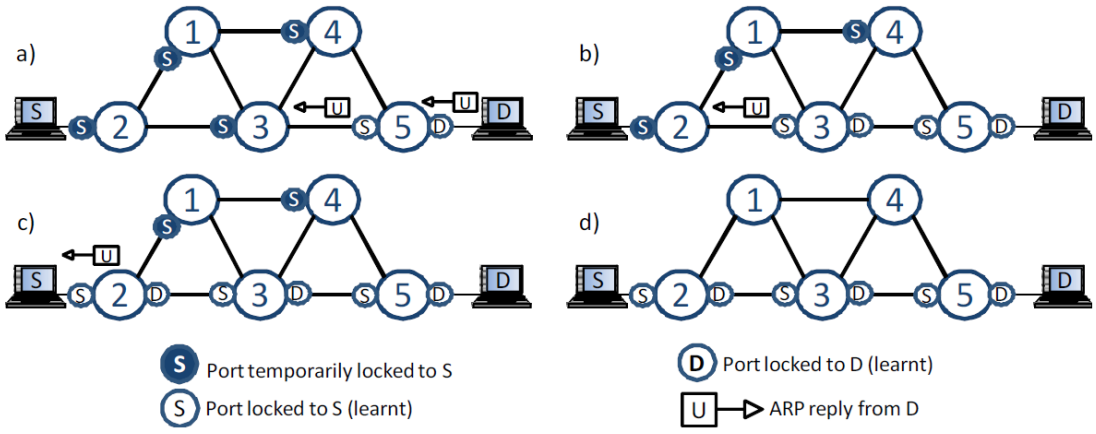


Ilustración 33: Propagación del paquete ARP-Replay para la "confirmación" de caminos en ARP-Path[53]

2.7.4.2 Funcionamiento con paquetes Unicast

Una vez los caminos han sido establecidos, el switch ARP-Path siempre realizará la misma acción, buscará la MAC del equipo destino dentro de su ARP-Table para obtener así el puerto de salida. Posteriormente, reenviará el paquete por dicho puerto de salida y actualizará la marca temporal de dicha entrada en la ARP-Table.

2.7.4.3 Mecanismo para evitar bucles

Como se ha comentado anteriormente, un punto peligroso de las redes es la generación de bucles. Este protocolo posee un mecanismo para evitarlas, el cual consiste en bloquear la propagación de los paquetes broadcast y Multicast mediante un temporizador de bloqueo (BTime), es decir, un paquete broadcast o multicast solo será reenviado por un switch ARP-Path en tres casos:

- En primer lugar, el switch ARP-Path no debe poseer en su tabla ARP-Table la MAC del equipo de origen del paquete.
- En segundo lugar, el switch debe poseer la MAC del equipo origen dentro de su tabla, pero el puerto de entrada del paquete debe ser el mismo que el puerto anotado en dicha tabla para esa MAC.
- En último caso, al igual que en el caso anterior, el switch contendrá en su tabla la MAC destino, pero en este caso la marca de tiempo de bloqueo se encuentra expirada.

2.7.4.4 Diagrama completo de funcionamiento de ARP-Path

Tras ver todos los casos posibles de ARP-Path, a continuación, se muestra en un diagrama de flujo todos los puntos explicados anteriormente, así como las interconexiones entre los mismos para obtener un protocolo funcional.

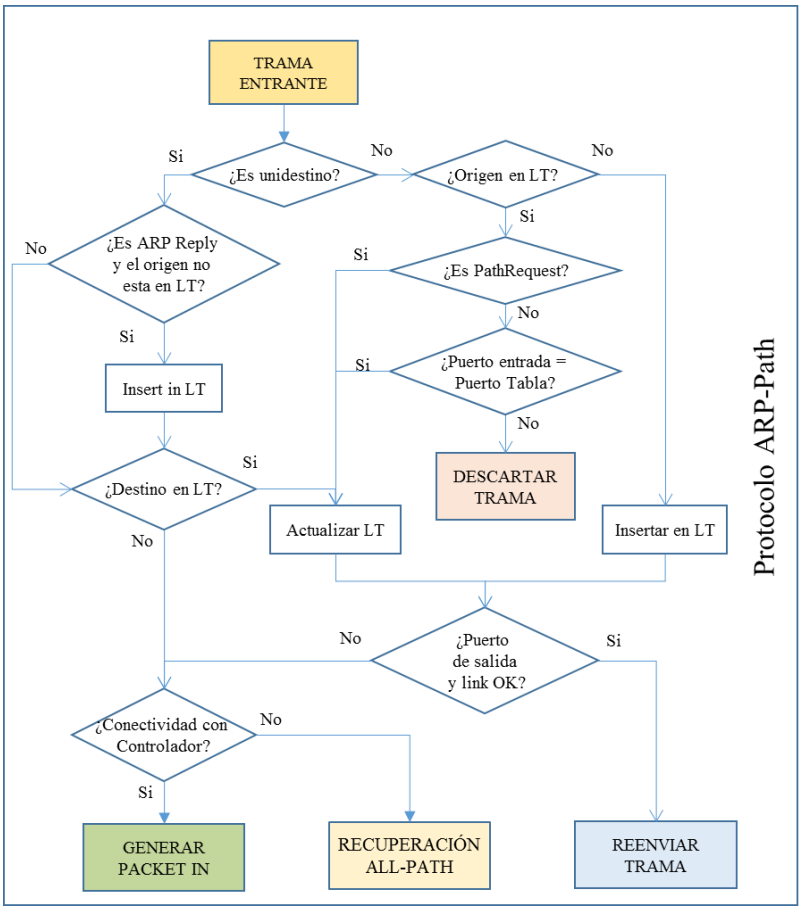


Ilustración 34: Diagrama de Flujo Protocolo ARP-Path

2.8 Sistemas de Recuperación

Uno de los puntos más problemáticos en los sistemas actuales son los mecanismos de recuperación. Durante la realización de este proyecto se han desarrollado dos modelos diferentes de recuperación.

2.8.1 Recuperación Centralizada

Al tratarse de un sistema híbrido, puede hacerse uso del controlador como si de un sistema nativo SDN se tratase. Esto quiere decir que se puede utilizar el controlador para generar rutas alternativas. Para entender mejor este tipo de controlador, se va a suponer una red como la representada en la Ilustración 35: Topología básica con controlador

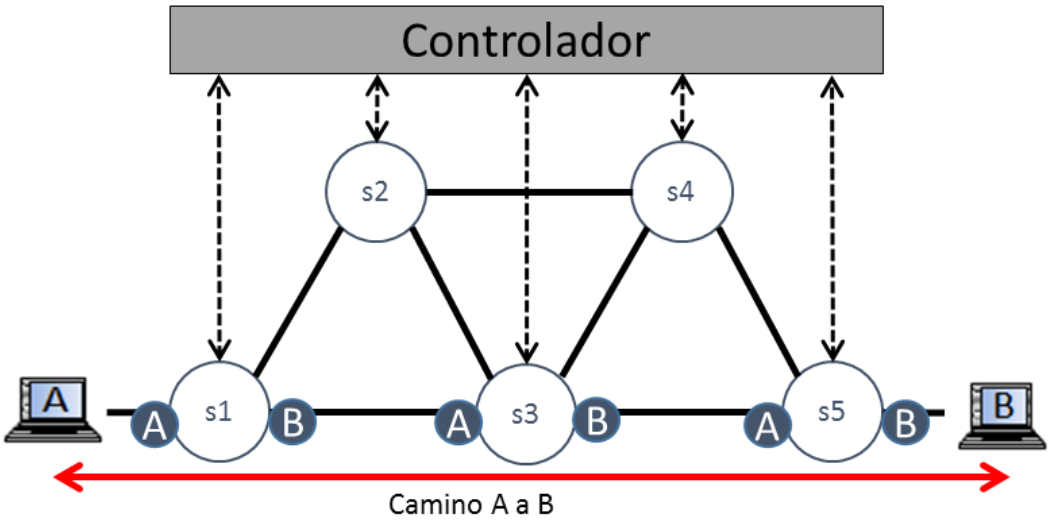


Ilustración 35: Topología básica con controlador

El sistema una vez que ha detectado la caída de un enlace encapsulará el paquete recibido en un PACKET IN[62] para, posteriormente, enviarlo al controlador. Su representación en la Ilustración 36: PacketIn al controlador.

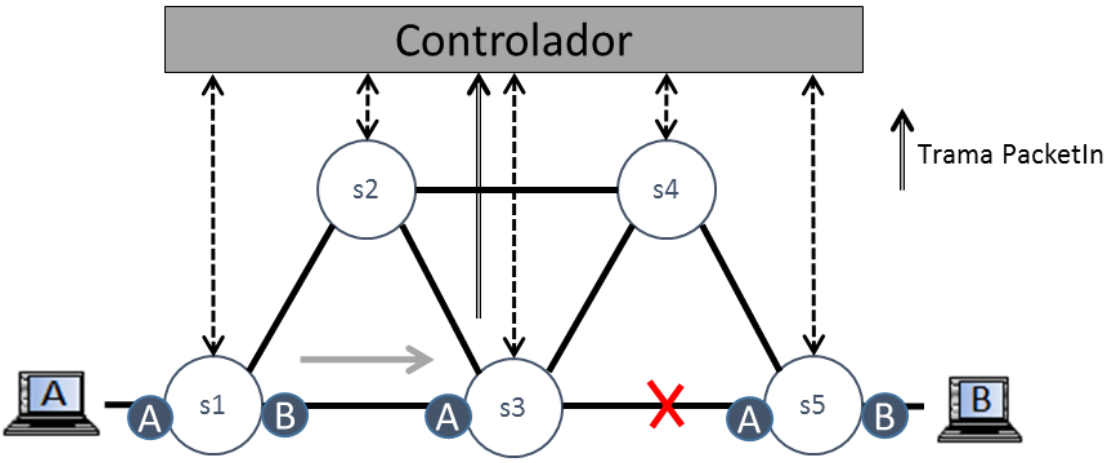


Ilustración 36: PacketIn al controlador

El Controlador, una vez ha recibido dicho paquete, calculará una nueva ruta. Dicha ruta se generará mediante el algoritmo de Dijkstra[63] aplicado al nuevo grafo resultante de la red.

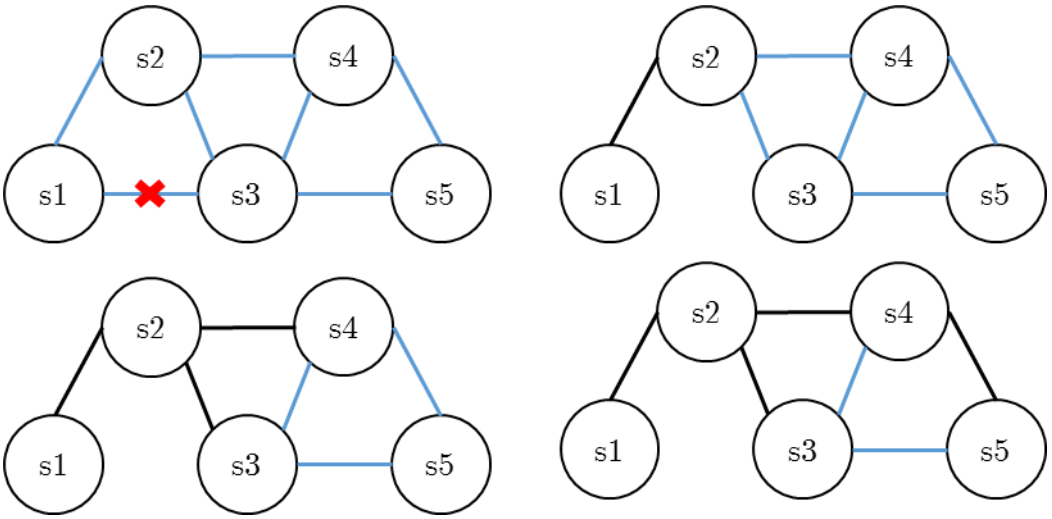


Ilustración 37: Aplicación Algoritmo Dijkstra a topología por parte del controlador

Por último, el Controlador generará tantos FLOWMOD[64] como switches existan en dicha ruta, definiendo ruta como la rama del árbol más corta desde origen a destino, para así instalar todas las reglas necesarias para establecer el nuevo camino. La instalación de las reglas se hará siempre desde el segundo switch hasta el último.

Esto se debe al tiempo que tardan los switches en instalar las reglas, es decir, el tiempo de procesamiento en los switches para la instalación de las nuevas reglas en muchos casos es mayor que el tiempo que tarda el controlador en enviar los paquetes de modificación más el paquete para restablecer el flujo, lo que puede llevar a su vez a una nueva recuperación sin ésta ser necesaria.

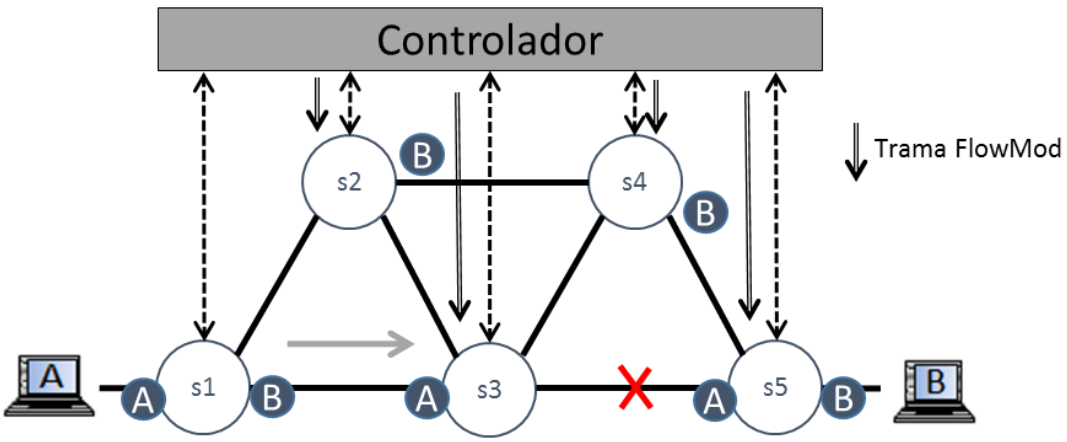


Ilustración 38: Instalación de nuevas reglas mediante FlowMod

Por último, enviará un PACKET OUT más FLOWMOD [64] al switch que comunicó el fallo para restaurar el flujo.

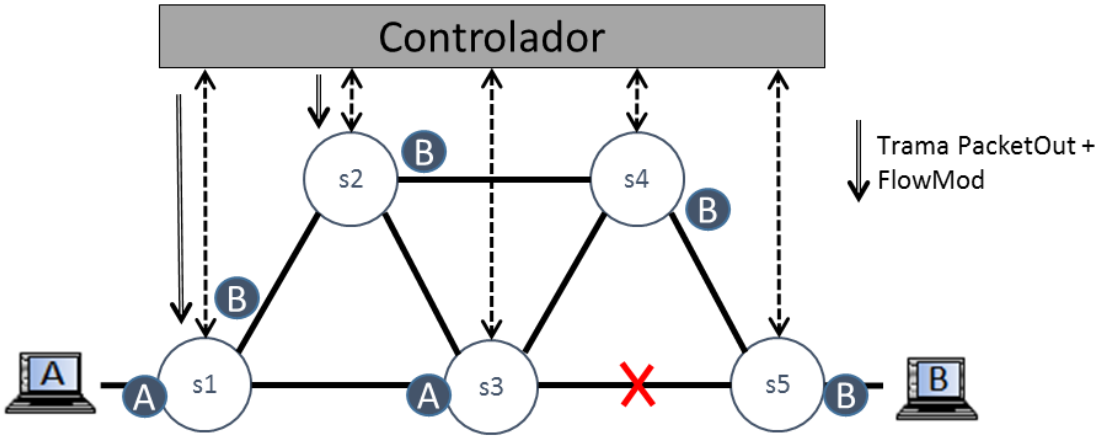


Ilustración 39: Envío de FlowMod más PacketOut al switch notificador del error

2.8.2 Recuperación Distribuida

El último tipo de recuperación implementada es una recuperación totalmente distribuida, es decir, no hace uso del controlador. Para esta implementación se ha diseñado la siguiente lógica en los switches.

Se parte de una situación como la representada en la Ilustración 35: Topología básica con controlador, donde se encuentra una red en la cual existe un flujo activo. En un instante de tiempo se rompe un enlace, como se representa en la siguiente figura:

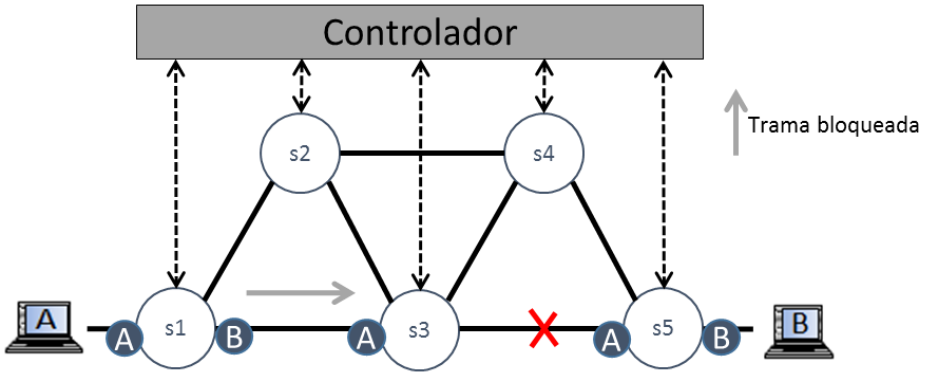


Ilustración 40: Caída de un enlace en la red

Una vez un switch detecta que no es capaz de reenviar el paquete por ningún puerto válido, éste crea un paquete especial llamado "Path Recovery", del tipo "Request", cuyo destino es el destino del paquete original. Dicho paquete se difunde por la red utilizando como mecanismo de bloque el mismo que utiliza ARP-Path, es decir, se comprueba si el paquete ha sido tratado anteriormente. En caso de ser afirmativo se rechaza el paquete; En caso contrario, se trata y reenvía.

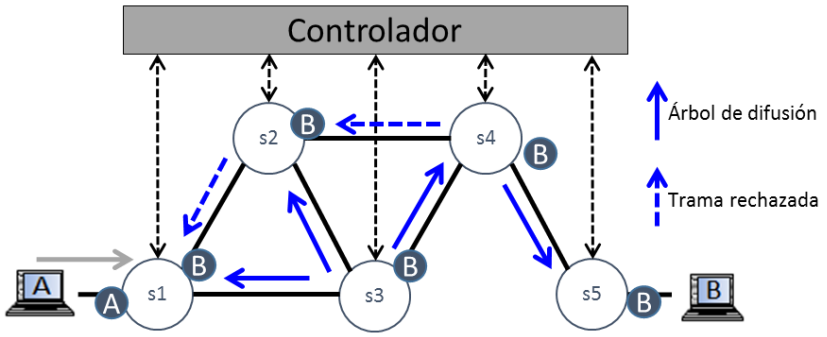


Ilustración 41: Difusión trama Path Recovery Request

Una vez la trama difundida es recibida por el switch frontera destino, este contesta con una nueva trama multicast "Path Recovery", del tipo "Replay", cuyo destino en este caso es el host origen del paquete original. En este proceso también se aplica bloque para evitar bucles en la red.

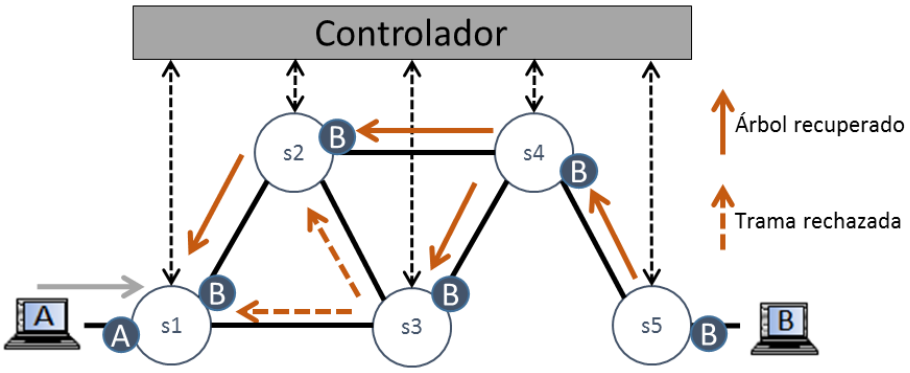


Ilustración 42: Difusión trama Path Recovery Replay

Al tratarse de un mecanismo de recuperación, todos los switches de la red actualizan sus tablas con la nueva situación del host destino.

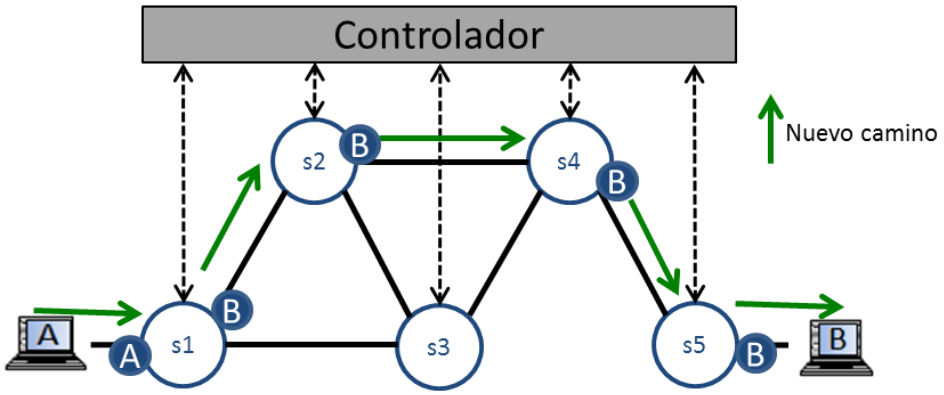


Ilustración 43: Recuperación del camino mediante recuperación distribuida

Este tipo de recuperación que puede producir movimientos en los caminos de los flujos vivos de la red, pero a su vez ahorra un gran número de nuevas recuperaciones ya que con una única recuperación se genera un nuevo árbol de distribución con raíz el switch frontera del host destino que puede ser utilizado por todos los host de la red.

2.8.2.1 Paquete Path Recovery

Se trata de un paquete especial que se genera el controlador y se propaga por la red para la recepción de la misma. La estructura de dicho paquete se muestra en Ilustración 44: Estructura Path Recovery, la cual se muestra a continuación.

ETH DESTINO	ETH ORIGEN	ETH TYPE	PATH TYPE
Dirección MAC del Grupo Multicast.	Dirección MAC del Equipo Final Destino de la recuperación	Identificador de protocolo	16 bits tipo de paquete 1: Path Request 2: Path Reply

Ilustración 44: Estructura Path Recovery

Este paquete pose dos tipos diferentes, Path Request y Path Reply, ambas tramas son multicast.

2.8.3 Lógica ARP-Path con recuperación distribuida

Como primera aproximación a un sistema con recuperación activa, se ha implementado una variación en la lógica propia de ARP-Path.

Dicha modificación consiste en introducir una nueva comprobación previa, es decir, al recibir cualquier paquete el sistema determina si es o no un paquete de recuperación "Path Recovery". En caso de no ser un paquete de recuperación se aplica directamente el protocolo ARP-Path, en caso contrario, se aplica la recuperación distribuida, explicada anteriormente en el apartado 2.8.2: Recuperación Distribuida.

Dicha implementación queda reflejada en el siguiente diagrama de flujo (Ilustración 45: Diagrama de Flujo para recuperación Distribuida).

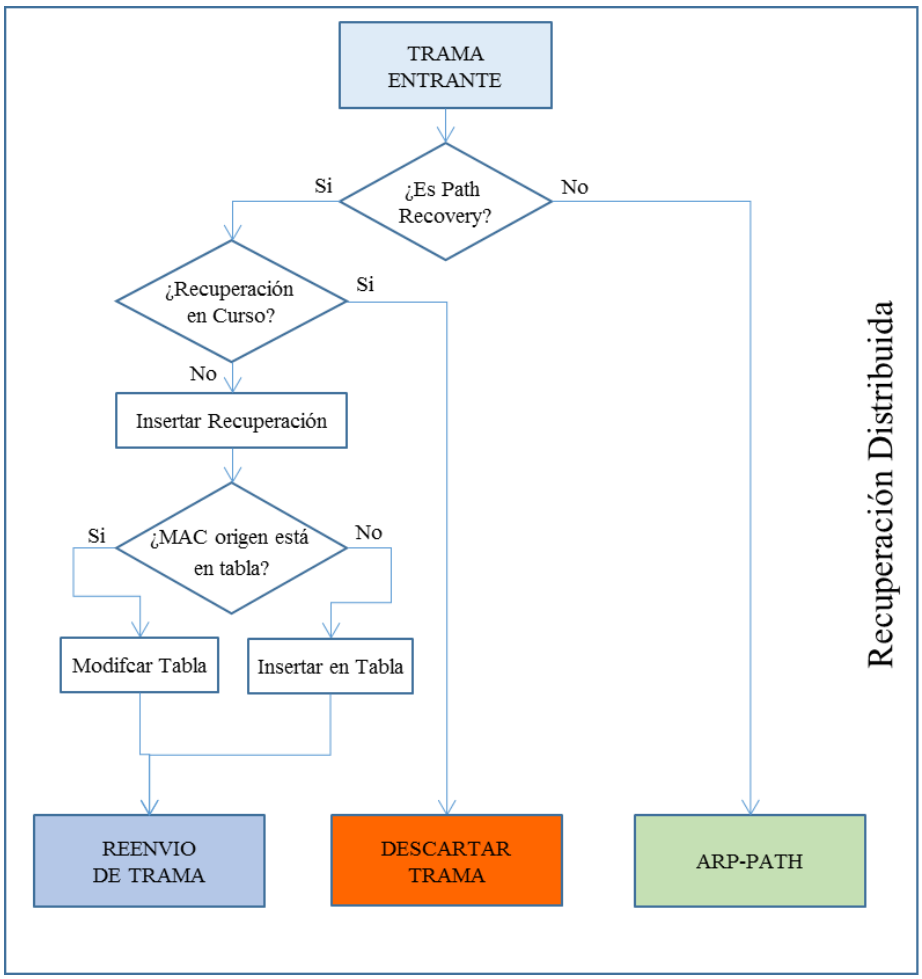


Ilustración 45: Diagrama de Flujo para recuperación Distribuida

3 Diseño del Sistema híbrido

Para desarrollar el switch híbrido (AOSS) se ha realizado un estudio previo donde se pretende entender el flujo de las funciones necesarias para propagación de los paquetes en el switch original (Ofsoftswitch13, CPqD[65]).

3.1 Switch OpenFlow

Si se observa el protocolo OpenFlow en su versión 1.3[2] se puede observar el siguiente diagrama de flujo:

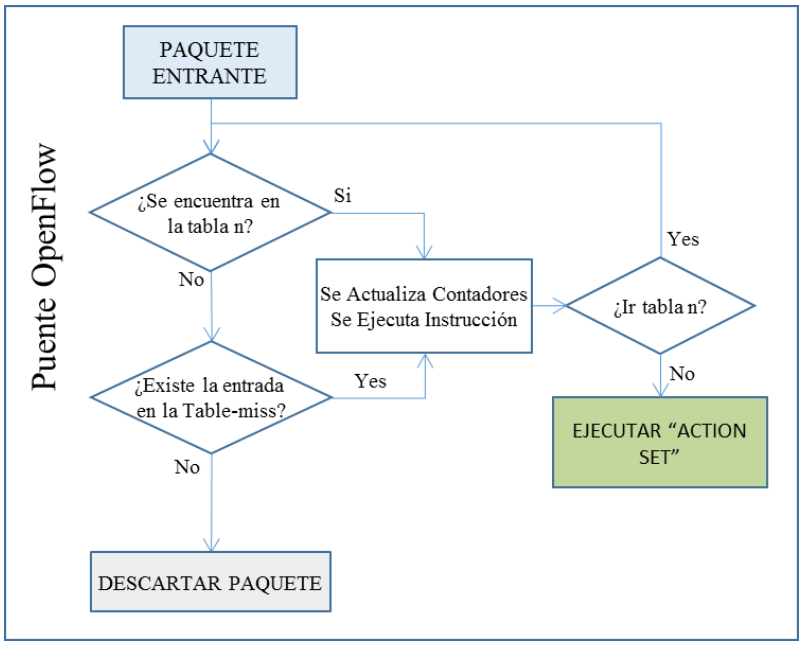


Ilustración 46: Diagrama de Flujo para switch OpenFlow

El funcionamiento que se puede obtener de dicho diagrama es el siguiente: cada paquete entrante es analizado para obtener los diferentes valores de la cabeceras (Ethernet, IP, TCP, etc...) que switch SDN es capaz de procesar.

Una vez se han obtenido dichos datos, se pasa a comprobar si existe alguna entrada en las tablas OpenFlow para dicho paquete. Para ello, se comprueba tabla a tabla si existe o no una coincidencia. De existir, se realiza la acción asociada a dicho registro: Envío al controlador, rechazo de paquete o, en caso contrario, se apunta la acción y se comprueba la siguiente tabla. Al finalizar la comprobación de todas las tablas, el sistema realiza el conjunto de acciones almacenadas según los registros consultados.

Por el contrario, si no existe ningún registro en las tablas consultadas, se consulta la tabla denominada "Miss-Table". En esta tabla el controlador al iniciar el switch suele insertar una acción por defecto, en la mayoría de los casos enviárselo a sí mismo.

Si por el contrario tampoco existe una acción asociada, el sistema rechaza el paquete.

Para una explicación más visual se puede consultar la Ilustración 47: Procesado interno de un paquete en un switch SDN.

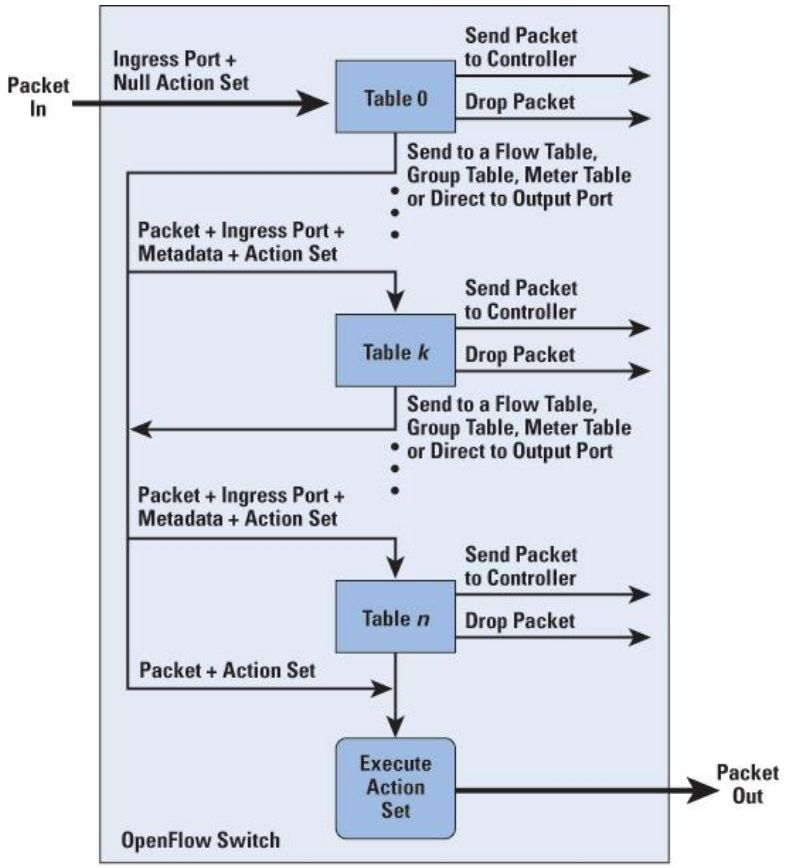


Ilustración 47: Procesado interno de un paquete en un switch SDN[20, p. 1]

3.2 Switch Híbrido Sin Recuperación

Una vez estudiado el switch original, se ha diseñado a partir de él un nuevo diagrama de flujo para insertar la parte distribuida.

La modificación parte del final del diagrama representado en la Ilustración 46: Diagrama de Flujo para switch OpenFlow. Al final de dicho diagrama se estable que si no existe una entrada en ninguna tabla OpenFlow se debe rechazar el flujo. En su lugar, se estable

una nueva manera de proceder, la cual consiste en realizar el protocolo ARP-Path[7] anteriormente explicando en el apartado 2.7.4: ARP-Path.

El diagrama de flujo, por lo tanto, sería como el mostrado en la Ilustración 48: Diagrama de Flujo switch híbrido sin recuperación, mostrada a continuación:

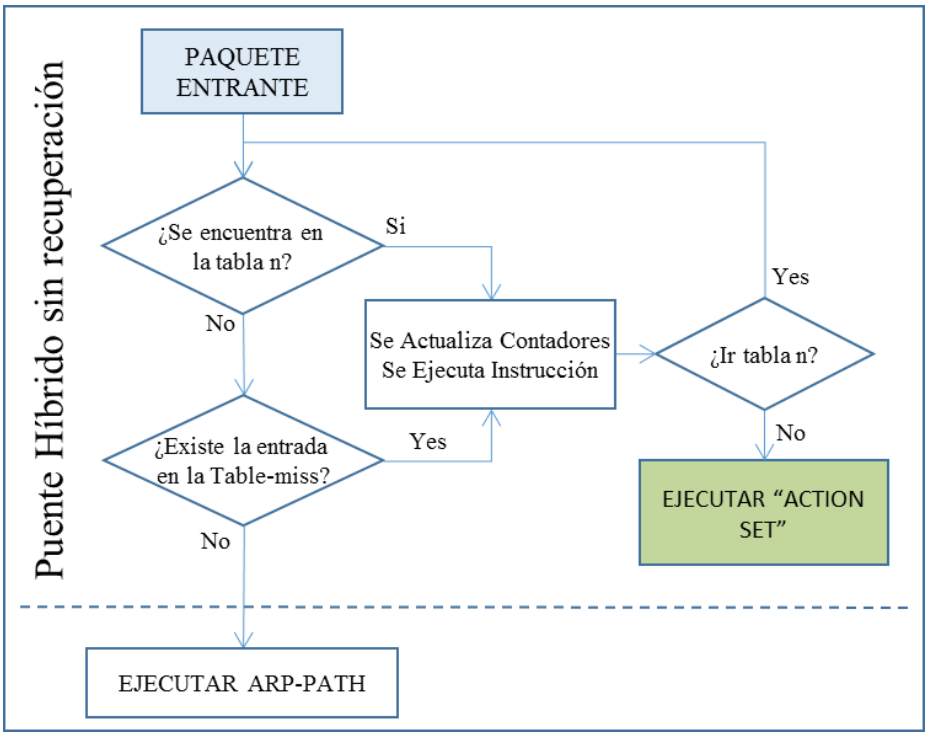


Ilustración 48: Diagrama de Flujo switch híbrido sin recuperación

3.3 Diseño en controlador de recuperación centralizada

Al tratarse de una recuperación centralizada, la implementación no se realizará sobre el propio switch, sino que esta vez se realizará en el controlador.

El controlador, al tratarse de una entidad superior, es capaz de conocer en todo momento la topología de la red, por lo tanto, aunque esta cambie, el controlador podrá recomponer el grafo que describe la red.

Para este proceso de recomposición el controlador hará uso del protocolo LLDP[18], es decir, el controlador de forma continua envía tramas LLDP a cada uno de los switches. Estos, una vez recibida dicha trama, se la envían a sus vecinos para que éstos se la devuelvan al controlador. De esta manera, el controlador conoce todas las conexiones que existen entre los switches.

Aprovechando este conocimiento de la red por parte del controlador, y una pequeña función introducida en el switch, gracias a la cual éste comunica sus equipos finales al controlador, se puede implementar esta recuperación.

La lógica para llevarla a cabo es la siguiente: El controlador una vez recibe un Packet-In, procedente de un switch, comprueba que el host destino está dentro de los host existentes en su red, los cuales ha podido almacenar por la función anteriormente explicada. Si es así, genera un camino óptimo en número de saltos para llegar hasta él. Por último, establece las reglas pertinentes en los diferentes switches afectados. El diagrama de flujo que representa a esta lógica se puede observar en la Ilustración 49: Lógica de implementación en controlador para recuperación centralizada

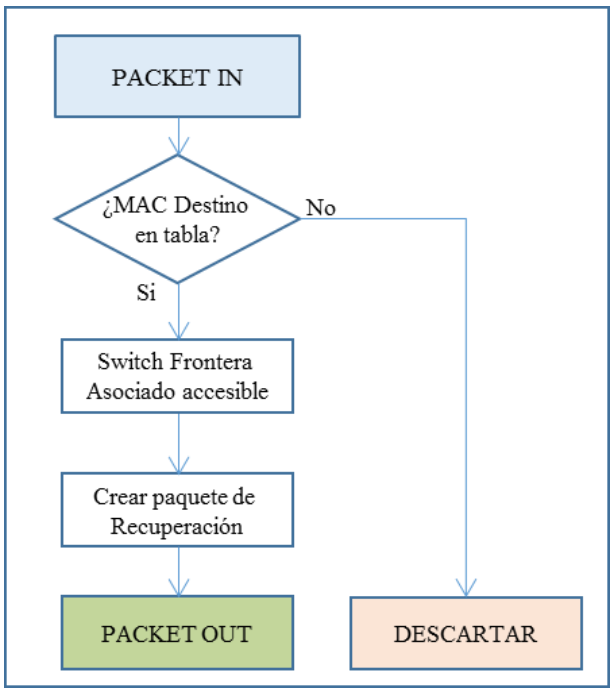


Ilustración 49: Lógica de implementación en controlador para recuperación centralizada

3.4 Switch Modificado con Recuperación distribuida

Por último, se propone generar un diseño que además de la recuperación anterior también sea compatible con la recuperación totalmente distribuida. Para ello, es necesario una restructuración del diagrama funcional representado en la Ilustración 48: Diagrama de Flujo switch híbrido sin recuperación.

En esta ocasión, además se va a tener en cuenta para este diseño, la velocidad de ejecución, por lo tanto se propone anticipar los procesos de detección de paquetes de

recuperación a la búsqueda en las tablas de OpenFlow. Es decir, como se ha podido observar en los dos casos anteriores, el paquete entraba en el flujo del sistema y éste lo primero que realizaba era comprobar si existía en el conjunto de tablas de OpenFlow alguna regla para tratar dicho paquete, y solo en caso de no poseer ninguna regla para dicho paquete se ejecutaba todos los procesos siguientes.

En este caso se introduce una comprobación previa al sistema de OpenFlow, la cual establece si el paquete recibido es o no un paquete de recuperación, y en caso afirmativo ejecuta directamente la Recuperación Distribuida.

Dicha modificación queda explicada en la Ilustración 50: Diagrama de Flujo Switch Modificado con Recuperación distribuida, la cual se muestra a continuación:

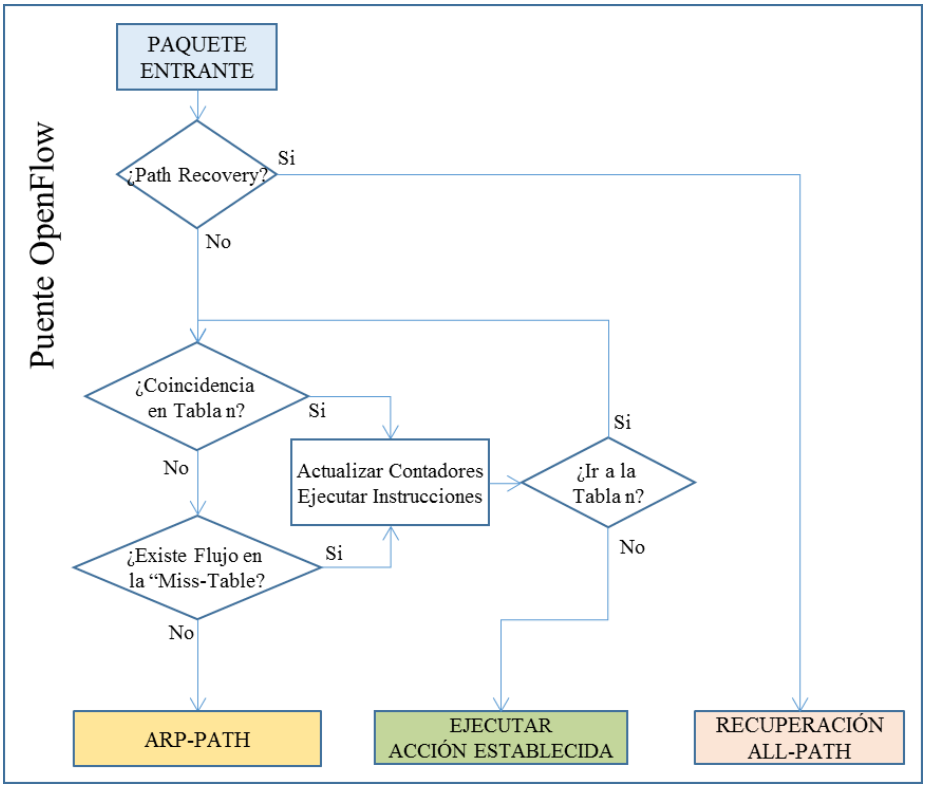


Ilustración 50: Diagrama de Flujo Switch Modificado con Recuperación distribuida

3.5 Diseño Funcional

3.5.1 Procesado de Flujo Inicial

Como se ha comentado en apartados anteriores, para tener un concepto claro de las funciones a implementar se debe primero tener claro que funciones existen y cuáles de ellas se han de modificar.

Para ello, se ha realizado un estudio del flujo de funciones que recorren los paquetes una vez se encuentran dentro del área de acción del switch OpenFlow Ofsoftswitch13.

Como se puede observar en la Ilustración 51: Diagrama Funcional de OfSoftSwitch, se puede ver como el switch está constituido por un bucle de funciones las cuales tratan el paquete en diferentes puntos.

El sistema comienza por un proceso denominado Run_DataPath, el cual estable todos los procesos necesarios para el correcto funcionamiento del datapath, desde la lectura de los puertos hasta la limpieza de procesos y manejadores.

Dentro del proceso anterior, el sistema lee cada uno de los puertos que tiene asociado a través de la función Dp_Run_Port. Ésta función es la encargada de obtener de los diferentes puertos los paquetes encolados y enviarlos a procesar.

Dicho proceso de paquetes lo realiza la función Process_Packet. Ésta función es la encargada de determinar si existe o no alguna regla dentro de las tablas de OpenFlow. La implementación de esta función se ha realizado de la forma más fiel a lo establecido en el diagrama de flujo representado por la Ilustración 46: Diagrama de Flujo para switch OpenFlow

Por último, y según el resultado obtenido por la función Process_Packet, el switch reenvía el paquete utilizando la regla obtenida por la función anterior o por el contrario descarta el paquete según lo establecido en el protocolo OpenFlow en su versión 1.3.

Como resumen visual se puede observar todo lo explicado anteriormente en la Ilustración 51: Diagrama Funcional de OfSoftSwitch, la cual se muestra a continuación:

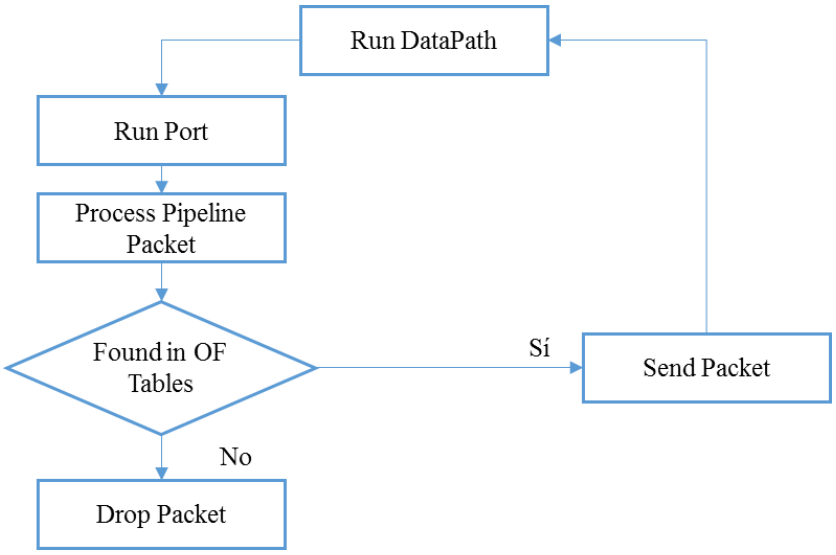


Ilustración 51: Diagrama Funcional de OfSoftSwitch

3.5.2 Procesado de Flujo modificado

Tras analizar el ciclo funcional anterior, se puede observar que existe un camino crítico por donde todos los paquetes no OpenFlow deben pasar. Dicho camino se representa en la Ilustración 51: Diagrama Funcional de OfSoftSwitch.

Por lo tanto, para poder desarrollar correctamente el switch híbrido (AOSS) se deben modificar algunas de esas funciones.

El primer cambio se puede observar en la generación de una nueva función Process_Pipeline_Packet_UAH, la cual sustituye (que no elimina) a la función Process_Pipeline_Packet, ya que debe mantenerse esta función para los diferentes manejadores OpenFlow incluidos en el switch.

Esta nueva función establece antes de la lectura de las tablas si el paquete recibido es o no un paquete recuperación propio de este desarrollo. En dicho caso, realiza la recuperación como se ha explicado en el apartado 3.4. En caso de determinar que no se trata de un paquete de recuperación se pasa a comprobar las diferentes tablas OpenFlow.

Otro de los cambios trata la eliminación de los paquetes en el caso de no existir una regla en las tablas OpenFlow y pasa a realizar el protocolo ARP-Path[7] explicado en el apartado: 2.7.4.

Por último se añaden dos nuevos procesos:

- El primero se trata de un proceso de limpiado de tablas para evitar tener así registros obsoletos y poder crear así un pequeño balanceo de carga.

- El segundo se trata de un proceso de envío de paquetes "Hellos" propios del protocolo ARP-Path[7].

Para una mayor claridad se ha diseñado la Ilustración 52: Diagrama secuencial de funciones del switch, la cual se muestra a continuación y donde se muestra un diagrama funcional del resultado producido por los cambios anteriormente descritos.

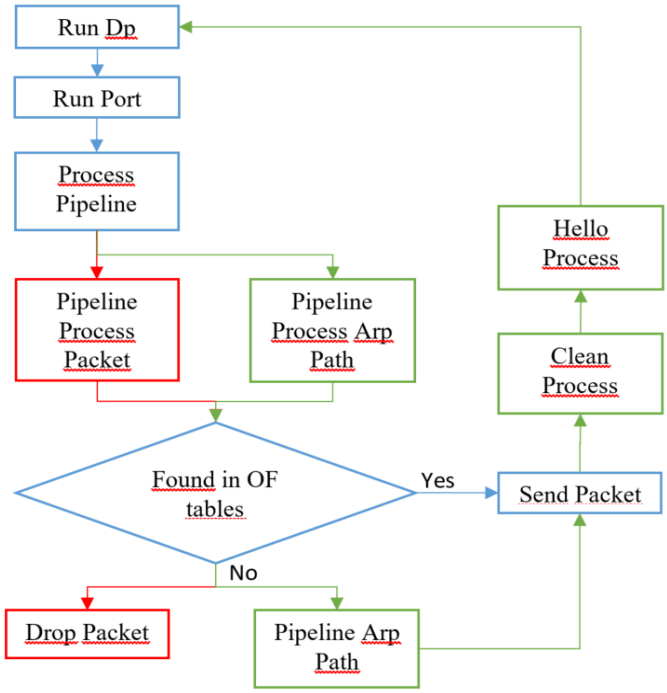


Ilustración 52: Diagrama secuencial de funciones del switch

4 Desarrollo del Sistema híbrido

Una vez explicado los diferentes análisis lógicos que se ha realizado en el proyecto, en esta sección se va a explicar los diferentes cambios que han sido implementados para llevar acabo la hibridación.

4.1 Switch híbrido

4.1.1 Modificación de la Función de Pipeline-Process

Como se ha explicado en el apartado 2.5.2, el switch elegido es un switch OpenFlow, por lo que se debe modificar la función de procesado general para hacer "hueco" e insertar la parte híbrida. Para esta sección se decido generar una nueva función de procesamiento de Pipelines. Esto se debe a que la función original para el procesamiento de Pipelines, denominada:

```
void pipeline_process_packet(struct pipeline *pl, struct packet *pkt);
```

Es utilizada por los diferentes manejadores de los paquetes propios del protocolo OpenFlow, lo que imposibilita la modificación directa de dicha función.

Por lo tanto, se decide hacer una modificación general del esquema lógico de funcionamiento, insertando en él dos nuevas funciones que solo afectan al flujo principal de paquetes de red y no al de los paquetes de OpenFlow. Dichas funciones son:

- *Pipeline Process UAH:* Se trata de una función de unificación de funcionalidades, es decir, es la encargada de gestionar la entrada en juego de cada una de las diferentes partes que conforman la hibridación, así como la detección de los sistemas de recuperación. Su definición es la siguiente:

```
void pipeline_process_Uah(struct pipeline *pl, struct packet *pkt,
struct mac_to_port *mac_port, struct mac_to_port *neighbor_table,
struct mac_to_port *recovery_table,  struct table_tcp * tcp_table,
uint8_t *puerto_no_disponible, struct timeval * t_ini_recuperacion);
```

- *Pipeline_arp_path:* Es propiamente la función que implementa el protocolo autónomo, en este caso ARP-Path. Su definición es la siguiente:

```
void pipeline_arp_path(struct pipeline *pl, struct packet *pkt, struct
mac_to_port *mac_port, struct mac_to_port *neighbor_table, struct
mac_to_port *recovery_table, int TIME_RECOVERY, uint8_t
*puerto_no_disponible, struct timeval * t_ini_recuperacion);
```

4.1.2 Implementación de ARP-Path

Al tratarse de un switch híbrido, se debe tener al menos dos elementos unidos, como se ha explicado anteriormente. Uno de los objetivos de este proyecto es otorgar al switch la capacidad de generar caminos sin necesidad de consultar al controlador. Para ello, se debe implementar en él el protocolo explicado anteriormente en el apartado 2.7.4.

Dicho protocolo dispone de dos tablas donde el sistema almacena las diferentes MAC, así como los puertos asociados a ellas.

4.1.2.1 Tabla de Bloque (BT)

En esta tabla el protocolo inserta todas las MAC de los paquetes broadcast. Su función principal es la eliminación de bucles dentro de la red, ya que si una MAC existe dentro de esta tabla, el sistema la rechaza siempre que no entre por el mismo puerto asociado.

4.1.2.2 Tabla de Aprendizaje (LT)

En esta tabla el protocolo inserta todas las MAC de los paquetes ARP-Reply. Se trata de la tabla de encaminamiento propiamente dicha, ya que es de ésta de la que sale el puerto de salida del paquete.

En este punto se encuentra la primera diferencia entre la teoría y la implementación. Al tratarse de sistemas emulados, la memoria y capacidad de procesos son limitados y contra menos memoria ocupada exista menor tiempo de proceso se tendrá, ergo mayor realismo en las pruebas se obtendrá. Para esto se ha tomado la decisión de crear una única tabla.

4.1.2.3 Tabla de Encaminamiento

En esta tabla el switch introduce todas las "MAC origen" procedentes de los paquetes broadcast, así como las MAC origen de los paquetes ARP Reply. Así mismo, se ha incluido una lógica adicional en ella. Dicha lógica sigue el siguiente pseudocódigo:

```
Si No existe MAC en TABLA:
      Insertar MAC
Si Existe:
      Si Port In = Port Tabla
            Actualiza MAC
      Sino
            Rechazar paquete
```

De esta manera se obtiene la misma funcionalidad pero utilizando una cantidad menor de memoria. Por el contrario, la velocidad de búsqueda es menor, es decir, al tratarse de una lista simplemente enlazada, el sistema tiene que recorrer todos los registros hasta encontrar el que coincide. Para agilizar este proceso se ha generado una función que se

encarga de reordenar los registros de la tabla para que en la parte superior esté siempre el último buscado. Esto se debe a que la probabilidad de que el siguiente paquete tenga relación con el actual es muy alta. Se ha implementado el siguiente pseudocódigo:

```
Buscar_registro:
      Desde inicio hasta fin:
            Si Existe coincidencia:
                  Se Coloca al inicio de la lista el registro
                  Se devuelve Puerto de salida
```

4.1.3 Implementación de los Mensajes Hello

Una de las funcionalidades necesarias para la posterior generación de la recuperación distribuida es la generación de los paquetes Hello. Estos paquetes son los encargados de indicar a los diferentes nodos de la red que vecinos tienen, es decir, cada uno de los nodos que componen la red envían a todos sus vecinos mensaje indicando que el vecino de ese puerto son ellos.

Para implementar este tipo de paquetes se ha generado una función cuya cabecera es la siguiente:

```
struct packet * packet_hello_create(struct datapath *dp, uint32_t
in_port, bool packet_out);
```

Esta función es la encargada de crear los paquetes Hellos. En el comienzo del proyecto, esta función era llamada periódicamente para que el switch creara el paquete y lo enviara. A continuación se decidió que era más eficiente llamarla una única vez para crear el paquete. Posteriormente dicho paquete se almacena en memoria para su posterior envío.

La segunda función que ha sido implementada es:

```
void packet_hello_send(void);
```

Esta función es la encargada de reenviar el paquete por todos los puertos de switch. Esta función es llamada de forma periódica para refrescar la tabla de vecinos almacenada en cada switch. Esta tabla sigue la misma estructura física que la tabla utilizada en la implementación de ARP-Path. Esto se ha decido así para poder reutilizar todas las funciones generadas para la correcta utilización de las tablas del protocolo anterior.

Por último, tras la inserción de esta tabla, ha sido necesario modificar la función *Pipeline Process UAH,* ya que existía la necesidad de tratar estos mensajes de forma diferente al

resto de paquetes. Estos paquetes son propios del protocolo ARP-Path, necesarios para que los switches conozcan a sus vecinos y así poder aplicar las diferentes recuperaciones.

La modificación consiste en insertar al comienzo una desviación lógica de estos paquetes, es decir, el switch al detectar estos paquetes deberá sacarlos del flujo general, para, posteriormente, anotar o actualizar la tabla de vecinos la MAC del switch vecino y descartar el paquete.

4.1.4 Implementación del Mecanismo de Recuperación Distribuida

Como se ha explicado en el apartado 2.8.2, esta recuperación se realiza de forma distribuida, es decir, no se hace uso del controlador para llevarla a cabo.

Para la implementación de esta recuperación es necesario realizar anteriormente otra implementación, la encargada de crear los paquetes que recorrerán la red creando los nuevos árboles de distribución.

4.1.4.1 Creación paquetes de recuperación

Para esta implementación se han realizado dos funciones:

- *packet_recovery_create*: Función encargada de crear el paquete de recuperación propiamente dicho. Para la implementación de la misma se ha definido la siguiente cabecera de función:

```
struct packet * packet_recovery_create(struct datapath *dp, uint32_t
in_port, bool packet_out, struct packet * pkt_original, uint8_t
opcion);
```

- send_packet_recovery: Función encargada de reenviar los diferentes paquetes de recuperación. Para la implementación de la misma se ha definido la siguiente cabecera de función:

```
void send_packet_recovery(struct datapath *dp, struct packet *
pkt_original, uint8_t opcion);
```

A diferencia del caso de la generación de los Hellos, estos paquetes si necesitan de una creación especial dentro de las funciones de maquetación y creación de los paquetes físicos. Para ello ha sido necesario modificar tanto la librería NetBee como el archivo XML de generación de paquetes.

4.1.4.2 Modificación XML para la generación de nuevos paquetes

Para la generación de paquetes ha sido necesario modificar el archivo XML encargado de generar los paquetes físicos. Se ha introducido un nuevo tipo denominado "ARPPath_repair" cuya definición es la siguiente:

```
<protocol name="arppath_repair" longname="RECOVERY ARP PATH" comment="Recuperacion
protocolo ArpPath." showsumtemplate="arppath_repair">
	<execute-code>
		<after when="$linklayer == 1">
			<assign-variable name="$packetlength" value="$currentoffset"/>
		</after>
	</execute-code>
	<format>
		<fields>
			<field type="fixed" name="op" longname="{0x8000 60}" size="2"
showtemplate="arppath_repair.code"/>
			<field type="fixed" name="sHwAddr" longname="{0x8000 61}"
size="6" showtemplate="MACaddressEth"/>
			<field type="fixed" name="dHwAddr" longname="{0x8000 62}"
size="6" showtemplate="MACaddressEth"/>
		</fields>
	</format>
</protocol>
```

4.1.4.3 Modificación NetBeen para la generación de nuevos paquetes

Por otro lado, para completar la correcta creación de estos paquetes, se ha modificado la librería NetBeen, más concretamente nbee_link.cpp. Se ha incluido un nuevo caso de uso, donde se ha indicado que se debe insertar en el paquete los datos solicitados por el XML. El código introducido es el siguiente:

```
else if (protocol_Name.compare("arppath_repair") == 0 && pkt_proto-
>arppath_repair == NULL)
{
	pkt_proto->arppath_repair = (struct arppath_repair_header *)
((uint8_t*) pktin->data + proto->Position);
	PDMLReader->GetPDMLField(proto->Name, (char*) "op", proto->FirstField,
&field);
	nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_ARP_REP_OP);
	PDMLReader->GetPDMLField(proto->Name, (char*) "sHwAddr", proto-
>FirstField, &field);
	nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_ARP_REP_SCR);
	PDMLReader->GetPDMLField(proto->Name, (char*) "dHwAddr", proto-
>FirstField, &field);
	nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_ARP_REP_DST);
}
```

4.1.5 Tablas de Datos

Dicha tabla se ha implementado como una lista simplemente enlazada formada por la siguiente estructura de datos:

```
struct mac_port_time{
      uint8_t  Mac[ETH_ADDR_LEN];
      uint16_t port_in;
      uint8_t vecino;
      uint64_t time_entry;
      struct mac_port_time *next;
};
struct mac_to_port{
      struct mac_port_time *inicio;
      struct mac_port_time *fin;
      int num_element;
};
```

4.1.6 Detector de caminos ARP-Path

Por último, se han implementado las funciones necesarias para observar de forma clara la variabilidad de los caminos generados por el protocolo ARP-Path. Para ello se ha llevado a cabo distintas implementaciones.

4.1.6.1 Generación del paquete de seguimiento

Al igual que en el caso anterior, se ha implementado un paquete especial en el que va almacenada la Id de cada switch por donde pasa el paquete hasta llegar a su destino. Para ello se ha implementado una función que genera un paquete que encapsula en su interior el paquete ARP-Request, pero además va apuntado los switches por donde pasa.

En este caso también se ha modificado, al igual que en el caso anterior, la librería NetBeen y el archivo XML asociado. Dicha modificación se asemeja mucho a la explicada anteriormente, pero añadiendo los nuevos campos necesarios para el almacenaje.

4.1.6.2 Desencapsulado del paquete de seguimiento

Para obtener de nuevo el paquete original (ARP-Request), se debe primero consultar la tabla de encaminamiento, en la cual al hacer la inserción de las diferentes MAC se ha indicado si dicha MAC es o no vecina del switch en cuestión. En el caso de que esta comprobación sea afirmativa, es decir dicha MAC sea vecina, el switch realizará la desencapsulación del paquete en cuestión y anotará en un log el camino realizado por el paquete para su posterior análisis.

4.2 Controlador

Al tratarse de un sistema híbrido, se debe comprobar y aprovechar la potencia que proporciona la entidad centralizada SDN para la realización de cálculos y generación de caminos.

4.2.1 Implementación recuperación centralizada

Aprovechando que existe una entidad "superior" que conoce toda la red, se puede generar un sistema de recuperación centralizado.

Para llevarlo a cabo se ha realizado las siguientes implementaciones:

4.2.1.1 Lector de Topologías

Para realizar correctamente la recuperación, el sistema debe conocer primero la topología de la red. Para ello se ha implementado una función. Esta función genera paquetes LLDP en el controlador. Una vez generados son enviados mediante Packet-out (no son almacenados en el switch) para que éste los reenvíe a sus vecinos. Una vez estos nuevos switches reciben dicho paquete, lo envían al controlador mediante un Packet-In.

Una vez el manejador del Packet-in del controlador detecta este tipo de paquetes llama a esta función, la cual comprueba varias cosas. El puerto de entrada al switch que envió el paquete, así como el origen de este paquete. Estos datos son insertados en un diccionario que posee la siguiente estructura:

```
{[Switch Inicial, Switch Final, Puerto entrada]}
```

Este tipo de estructura se repite tantas veces como enlaces se tenga, de tal forma que se obtiene un grafo direccional que define la topología de la red.

Está función es capaz de detectar de forma muy rápida cualquier cambio en la red, puesto que dichos paquetes se envían de forma periódica para actualizar el grafo.

4.2.1.2 Envío de los hosts vecinos al controlador:

Ya que la generación de caminos se realiza de forma autónoma, el sistema debe indicar al controlador la situación de los diferentes hosts. Para ello se ha realizado una modificación en la función Pipeline_process_uah.

Dicha modificación consiste en generar un paquete paralelo al flujo real del sistema, donde se le indica al sistema que un host es vecino de un nodo. Para ello, el sistema crea un paquete igual al que posteriormente se reenvía por la red, siendo el destino de este nuevo paquete el controlador.

En el controlador se esperan paquetes que le indiquen esta información. Es por ello que se ha activado un manejador de Packet-In encargado de obtener la dirección MAC origen del paquete para su posterior inserción en la tabla de vecinos propia del controlador.

4.2.1.3 Manejador para la recuperación de flujos

Al tratarse de una función de recuperación, el controlador debe detectar los flujos que le llegan mediante los paquetes Packet-in. Una vez obtenida la información necesaria, debe dirigirse a la tabla de vecinos para localizar los switches origen y destino del flujo.

Una vez conocido estos dos switches, este manejador deberá obtener un nuevo camino. Para ello se ha hecho uso de una librería que implementa la función K-ShortestPath.

Para hacer uso de ella, el sistema deberá pasarle un grafo, representado por un diccionario, donde queda descrita la arquitectura de la red, así como la pareja de switches origen-destino del flujo. Esta función devolverá otro diccionario con la rama recuperada.

Posteriormente esta función instalará las nuevas reglas sobre los switches afectados.

4.2.1.4 Función generadora de caminos K-Shortest Path

Para la implementación de esta función se ha hecho uso de una librería que implementa en su interior el algoritmo de Yen[66].

Dicha implementación ha sido realizada siguiendo el siguiente pseudocódigo:

```
function YenKSP(Graph, source, sink, K):
   Determinar el shortest path del flujo haciendo uso de Dijktra y se
   almacena dicho camino
   Desde 1 hasta numero de caminos distintos:
       Desde el primer nodo del camino almacenado hasta el ultimo
           Se obtiene las rutas mas cortas para cada nodo del camino
           La secuencia de nodos desde el orgien hasta el nodo en
           cuestion se define como la ruta corta
           Posteriormente se eliminar las rutas no validas de cada
           Nodo
Al final se devuelve los K caminos seleccionados
```

Como se puede observar el algoritmo es complejo, por lo que se ha decido utilizar una librería que lo implemente correctamente.

5 Experimentación y Resultados

5.1 Escenario de Pruebas

Para la realización de estos experimentos se ha utilizado un sistema distribuido de máquinas formado por cuatro máquinas con procesador Core i7, cuya función es el soporte propio de la emulación de la red (ejecución de procesos de switches, generación de flujos y virtualización ligera de host) junto a un servidor con procesador Core i5, el cual tiene la función de Controlador Ryu. Todo ello interconectado mediante líneas GbEthernet a través de un switch.

Todo el montaje se puede observar en la Ilustración 53: Escenario de Pruebas

Ilustración 53: Escenario de Pruebas

5.2 Estudio de la sobrecarga del controlador

Como se ha expuesto anteriormente, la arquitectura utilizada en SDN[67] establece un sistema centralizado mediante controlador. Dicho sistema presenta un problema de escalabilidad, ya que, como se ha explicado anteriormente, SDN utiliza una máquina para calcular y establecer todas las reglas de la red. Esto provoca que si dicha red es lo suficientemente grande o posee un volumen de tráfico muy elevado, genera un procesamiento excesivo en el controlador, lo que puede llevar a un cuello de botella en el controlador, provocando un retardo en el cálculo de rutas de la red.

En este experimento se desea observar el uso de procesador por parte del controlador, así como el intercambio de paquetes entre el switch y el propio controlador, para poder establecer con datos esa falta de escalabilidad citada anteriormente.

Para ello se ha hecho uso de una topología VL2[39, p. 2], constituida por 2 Cores de 4 puertos cada uno, 4 Agregados de 4 puertos cada uno, 4 ToRes de 18 puertos cada uno y 16 equipos por ToR, como se muestra en la Ilustración 29: Topología VL2.

Sobre dicha topología se ha lanzado un total de 120 flujos distribuidos de la siguiente manera: El host 1 envía flujos a todos los demás hosts menos así mismo; El host2 envía flujos a todos los demás host menos así mismo y al host 1; El resto de host repite la misma acción, es decir, cada host envía flujos a todos los demás hosts menos a sus predecesores y así mismo. Los flujos se envían de forma secuencial y con una pequeña separación temporal entre cada uno de ellos.

Tras la realización de la prueba se puede observar los siguientes resultados:

5.2.1 Comparación intercambio de paquetes

Si se toma como objeto de estudio el intercambio de paquetes necesario entre el controlador, utilizando como aplicación un "*Learning Switch*" y un switch OpenFlow (OfSoftSwitch[65]) o el switch híbrido implementado (AOSS), se observa una diferencia en el intercambio de paquetes realizado entre OfSoftSwitch[65] y AOSS. Dicha diferencia queda reflejada en la siguiente gráfica, donde se puede observar una disminución en el intercambio de paquetes entre el switch AOSS y el controlador.

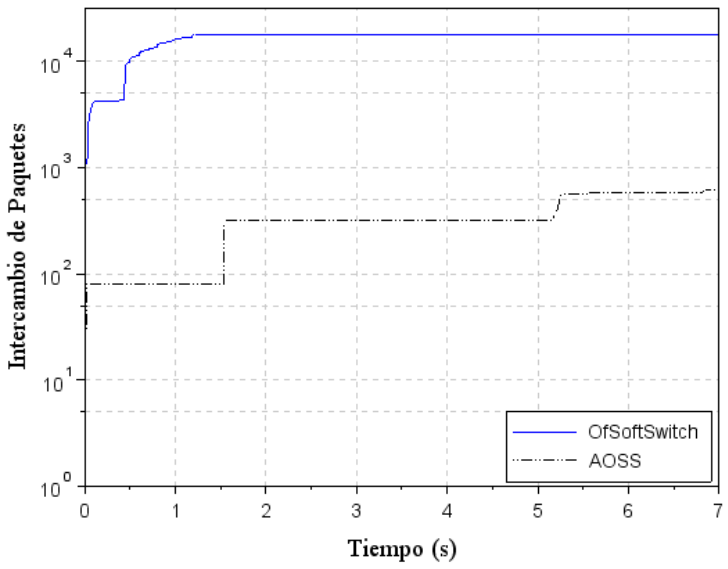


Ilustración 54: Gráfica comparativa intercambio de paquetes

5.2.2 Comparación procesamiento del controlador

Si por el contrario se utiliza como elemento de estudio el porcentaje de uso del procesador necesario para cada uno de los dos casos, se puede observar que el uso de AOSS provoca, en el controlador, una disminución en el uso de su procesador. Así mismo dicha diferencia queda reflejada en la siguiente gráfica:

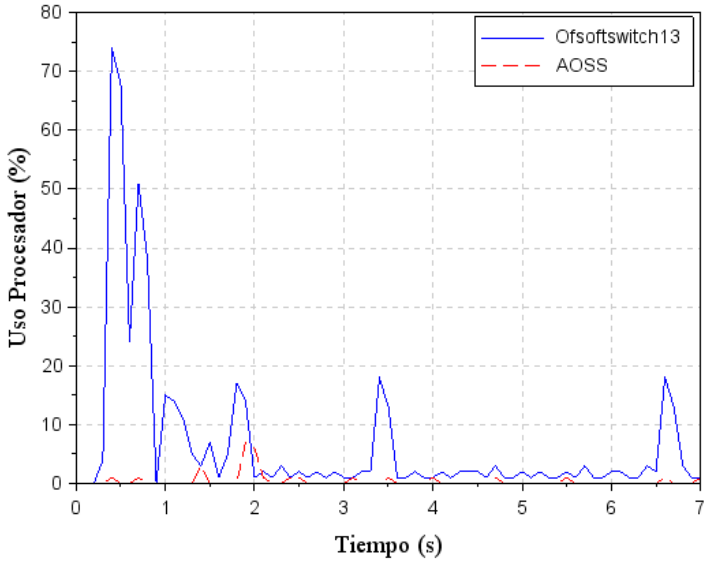


Ilustración 55: Gráfico comparativa sobre el uso de procesador en el controlador

Como se puede observar en dicha ilustración, AOSS baja el uso del controlador hasta un 90%.

5.3 Medición de Retardos

Como segundo parámetro de medida, se ha estudiado el tiempo de establecimiento de los flujos. La medición de este parámetro es muy importante para el protocolo ARP-Path, puesto que se encarga del establecimiento de caminos mediante la exploración de la red para, posteriormente, seleccionar los de mínima latencia. Es decir, necesita conocer el retardo de cada una de las ramas para así escoger en todas ellas el camino con el menor tiempo de transmisión. Por tanto, es de esperar que los retardos obtenidos por ARP-Path sean mejores que el de otras soluciones más clásicas.

En este caso, se ha decidido comparar una aplicación en controlador que genera caminos disjuntos, una aplicación en controlador que genera caminos congruentes y el switch

implementado en este proyecto. Dichos controladores calcularán dos tipos de rutas diferentes:

- El primero generará rutas lo más disjuntas posibles, generando así una red lo más balanceada posible,
- El segundo generará rutas congruentes. El balanceo alcanzado por este tipo de rutas es menor al anterior.

Así mismo, dichos controladores serán ejecutados de diferentes formas:

- La primera de ellas, el controlador instalará todas las reglas a priori en los switches (pre-instalación de reglas)
- En el segundo caso, el controlador realizará la instalación de reglas bajo demanda.

Para la realización de este experimento se ha diseñado una topología del tipo VL2 con las mismas características que el experimento anterior.

Tras realizar el experimento se puede observar los siguientes resultados:

5.3.1 Retardos Medios

Como se puede observar en la siguiente gráfica:

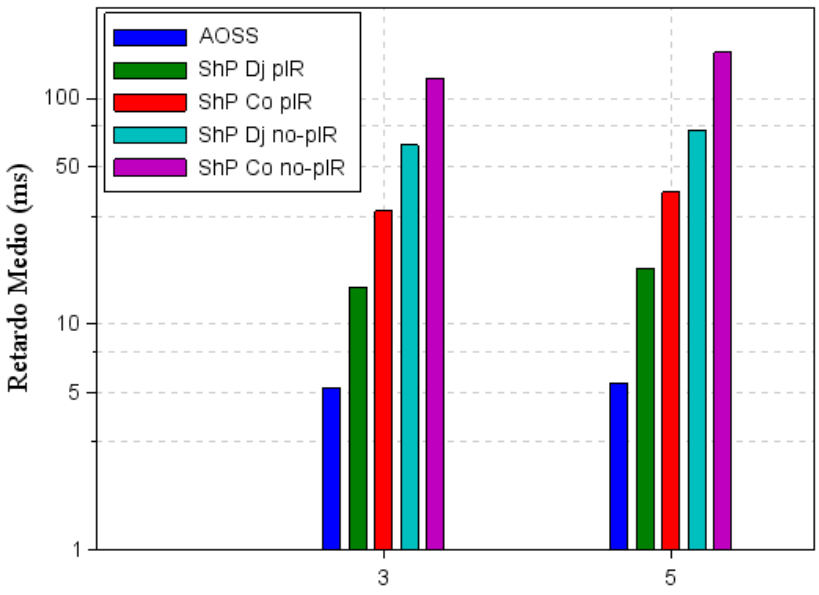


Ilustración 56: Gráfica de retardos medios en el establecimiento de caminos según el número de saltos en la red

El sistema implementado (AOSS) obtiene unos tiempos de establecimiento medios más pequeños que en el resto de casos, ya que aprovecha la versatilidad del protocolo ARP-Path para generar caminos mediante la difusión de los ARP's.

Por el contrario, los caminos establecidos con el controlador que no pre-instala reglas obtienen los peores resultados, ya que en cada uno de los switches el paquete inicial del flujo debe consultar el siguiente salto al controlador lo que introduce un retardo en el establecimiento.

Por último, se puede observar como el controlador que calcula caminos disjuntos para los flujos, obtiene mejores resultados que el controlador con caminos congruentes, ya que aprovecha la multiplicidad de caminos existentes en la red para realizar el balanceo de carga y así reducir los tiempos de establecimiento.

5.3.2 Funciones de distribución acumulada

Otra medida realizada en este experimento es la evolución del establecimiento de los caminos. Para observar este comportamiento, se han generado las siguientes gráficas, las cuales representan las funciones de distribución acumulada.

En la primera gráfica, se puede observar la evolución para un controlador cuyo comportamiento no permite la pre-instalación de reglas.

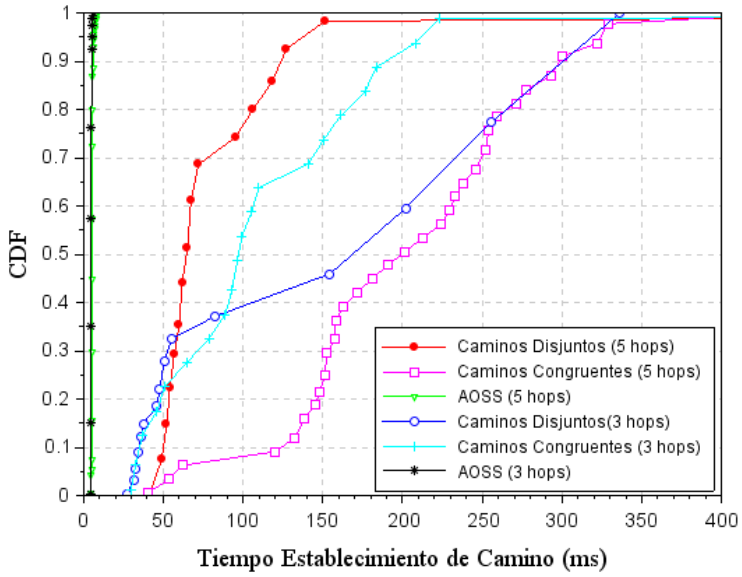


Ilustración 57: Gráfica función de distribución acumulada sobre establecimiento de caminos sin regalas OpenFlow instaladas

En la segunda se puede observar la evolución para un controlador cuyo comportamiento pre-instala todas las reglas necesarias para conectar los diferentes flujos.

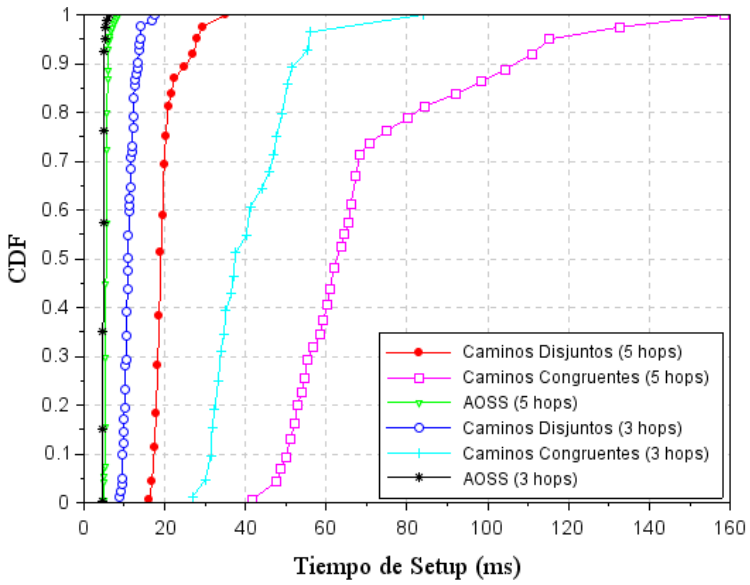


Ilustración 58: Gráfica función distribución acumulada tiempo de establecimiento de caminos con reglas OpenFlow instaladas

En ambos casos se puede observar como el switch AOSS posee un mejor comportamiento a la hora de establecer los flujos disminuyendo el tiempo de establecimiento entre un 50 % y un 95% dependiendo del tipo de controlador utilizado. Esta mejora se debe a que la búsqueda en sus tablas es mucho más rápida gracias a su sencillez, y como segundo motivo, no debe consultar a nadie el reenvió de una trama desconocida.

5.4 Multiplicidad de Caminos

El uso del protocolo ARP-Path [7] permite, como se ha explicado anteriormente en el apartado 2.7.4, la obtención de caminos diferentes. Al ser un protocolo que selecciona los caminos según los retardos de cada uno de ellos, puede generar diferentes caminos según el estado de la red en el momento de la generación del mismo. Es decir, como se ha explicado anteriormente, ARP-Path establece como parámetro de creación de camino el tiempo de latencia que tiene cada rama, por lo tanto, cada vez que se realiza una prueba en una red, estos caminos han podido sufrir modificaciones y, por lo tanto, la latencia de cada uno de ellos es diferente. Esto lleva a que ARP-Path genere caminos diferentes según el momento de creación del flujo.

En este experimento se desea comprobar esta propiedad. Para ello, se ha diseñado la siguiente prueba. Utilizando una topología VL2[39, p. 2] semejante a la representada en

la Ilustración 29: Topología VL2, se generan flujos UDP siguiendo el mismo modelo que en las pruebas anteriores.

Tras esta prueba se han observado los siguientes resultados:

Tabla 9: Tabla de multicaminos para flujos

Flujo	Repetición	Caminos	Flujo	Repetición	Caminos
h6 - h10	1	t2-a2-c2-a3-t3	h6 - h14	1	t2-a2-c2-a3-t4
	2	t2-a2-c1-a4-t3		2	t2-a2-c1-a4-t4
	3	t2-a2-c1-a3-t3		3	t2-a2-c2-a4-t4
	4	t2-a2-c1-a3-t3		4	t2-a2-c1-a3-t4
	5	t2-a1-c2-a3-t3		5	t2-a2-c1-a3-t4
h11 – h3	1	t3-a3-c1-a2-t1	h11 – h7	1	t3-a3-c1-a1-t2
	2	t3-a4-c1-a1-t1		2	t3-a4-c1-a1-t2
	3	t3-a3-c2-a1-t1		3	t3-a3-c1-a2-t2
	4	t3-a3-c1-a1-t1		4	t3-a3-c1-a1-t2
	5	t3-a3-c2-a2-t1		5	t3-a3-c2-a2-t2
h16 – h4	1	t4-a4-c2-a1-t1	h16 – h8	1	t4-a3-c1-a2-t2
	2	t4-a4-c2-a2-t1		2	t4-a4-c1-a1-t2
	3	t4-a3-c1-a2-t1		3	t4-a4-c1-a1-t2
	4	t4-a3-c2-a1-t1		4	t4-a4-c1-a1-t2
	5	t4-a4-c1-a1-t1		5	t4-a4-c1-a1-t2

Como se puede observar en el fragmento de tabla anterior, el protocolo es capaz de generar caminos diferentes para cada flujo, lo que provoca en la red un aumento en el balanceo de la carga. Esto conlleva, a su vez, un aumento y mejora en el throughput asociado a cada flujo. Esto último se verá más claramente en el siguiente experimento.

5.5 Throughput

Para realización de este experimento se ha generado una topología Mallada como la indicada en la Ilustración 26: Topología mallada.

En dicha topología se generarán nueve flujos UDP iguales, donde el origen de cada uno de ellos serán los Host comprendidos entre el Host2 y Host9, mientras que el destino de todos ellos será el mismo (Host1).

En esta prueba se ha implementado en el controlador dos protocolos diferentes de encaminamiento de flujos. El primero STP, explicado en el apartado Spanning Tree Protocol (STP) y ShP[68]. Dichas implementaciones se han realizado sobre el controlador Ryu[25].

Tras la prueba se puede observar los siguientes resultados, los cuales han sido representados sobre la siguiente gráfica:

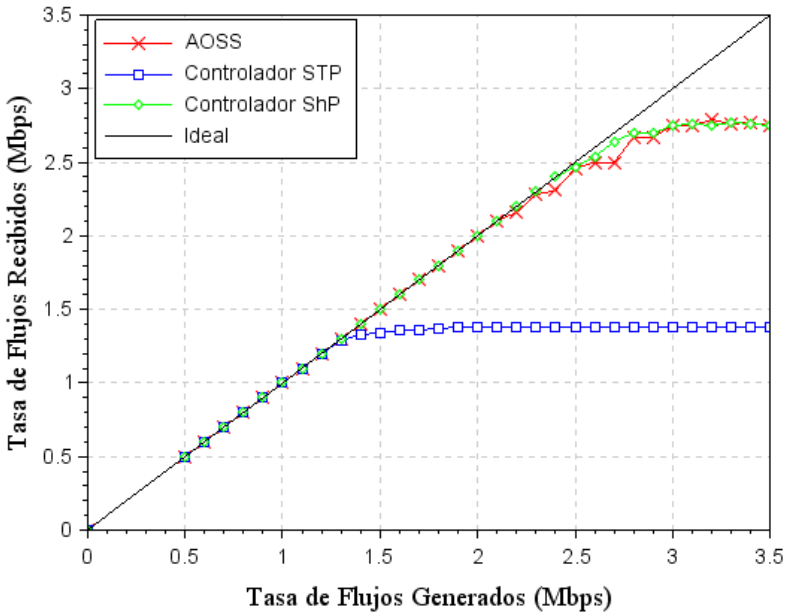


Ilustración 59: Gráfica comparativa sobre el rendimiento en la red

Como se puede observar, AOSS es capaz de generar caminos cuyo rendimiento es prácticamente igual a los caminos generados mediante un controlador con ShP y hasta un 93% más rápidos en el caso de utilizar un controlador con STP, además como se ha visto en los experimentos anteriores, dicho rendimiento se obtiene sin necesidad de generar una sobrecarga en el controlador.

Esto es debido a que al establecer como parámetro de creación el tiempo genera un balanceo de carga innato en la red. Es decir, cada instante de la red provoca que la latencia de cada rama entre dos puntos varíe, lo que provoca a su vez que existan ramas más o menos cargadas dependiendo del momento de generación del flujo.

ARP-Path, aprovechando la necesidad que tienen los host de la red de generar ARP's para conocer las diferentes MAC, genera de manera sencilla un camino de mínima latencia entre dos puntos, lo que lleva a que el throughput de los flujos aumente, puesto que se genera un balanceo de carga en la red de forma natural.

5.6 Medición tiempo de Recuperación

Uno de los puntos importantes de todas las redes es su capacidad de reacción ante un fallo. Para comprobar el correcto funcionamiento del switch AOSS ante los posibles fallos de la red, se ha diseñado la siguiente prueba, la cual hace uso de una topología circular

como la explicada en la Ilustración 27: Topología circular. En dicha topología se han conectado dos host situados en dos switches lo más próximos posibles entre ellos, ya que de esta manera se consigue el peor de los escenarios para la recuperación puesto que el camino que se va a generar es el más extenso posible.

En este escenario se genera un único flujo entre dichos host. Dicho flujo se genera en un rango de tasas, que van desde 10Mbps hasta 80Mbps. En un momento aleatorio de la prueba se genera un corte en el enlace que une los switches que soportan a los host.

En esta prueba se han medido dos elementos diferentes.

- El primero es el tiempo de recuperación propiamente dicho
- El segundo elemento es el tiempo de procesamiento necesario por parte del controlador para cada una de las recuperaciones explicadas anteriormente.

Dichos datos se representan a continuación:

5.6.1 Tiempo de recuperación

Tras realizar el experimento, se puede observar que el tiempo de recuperación del flujo depende, por un lado, del tipo de recuperación que se haya activa en el momento de la caída del enlace y, por otro lado, del grado de congestión o saturación de la red. Esto se puede observar en la siguiente gráfica comparativa:

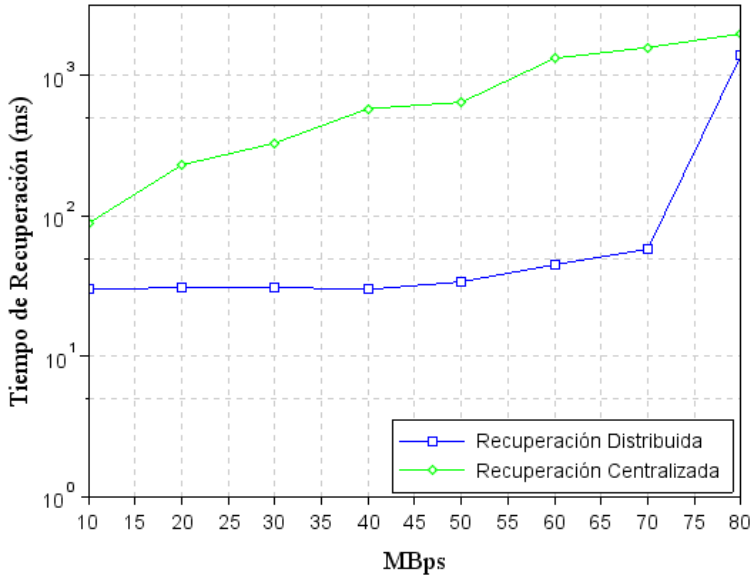


Ilustración 60: Gráfica comparativa sobre el tiempo de recuperación

Si se analizan los resultados, se puede observar que en una red con carga media baja, la recuperación centralizada necesita de un mayor tiempo para la recuperación del sistema, ya que en este caso el controlador tiene un tiempo de procesamiento no despreciable. Por el contrario, a tasas altas donde la red se encuentra congestionada, los paquetes de recuperación necesarios para la recuperación distribuida van encontrando colas de espera a medida que avanzan por la red, lo que provoca que el tiempo de recuperación se aproxime más al tiempo de recuperación del sistema centralizado, que al hacer uso de una red de control "outband", es decir una red de control paralela donde no existe dicha congestión provoca que sus tiempos se mantenga mucho más constantes independientemente del estado de la red de datos.

5.6.2 Tiempo en controlador

Al tratarse de una arquitectura apoyada en controlador, los sistemas SDN, además de tener un tiempo de recuperación propiamente dicho de la red, poseen otro tiempo mayoritario. Dicho tiempo es el procesamiento que el controlador tarda en gestionar la petición. Este tiempo se ha medido y comparado entre ellos, llegando así a la siguiente gráfica:

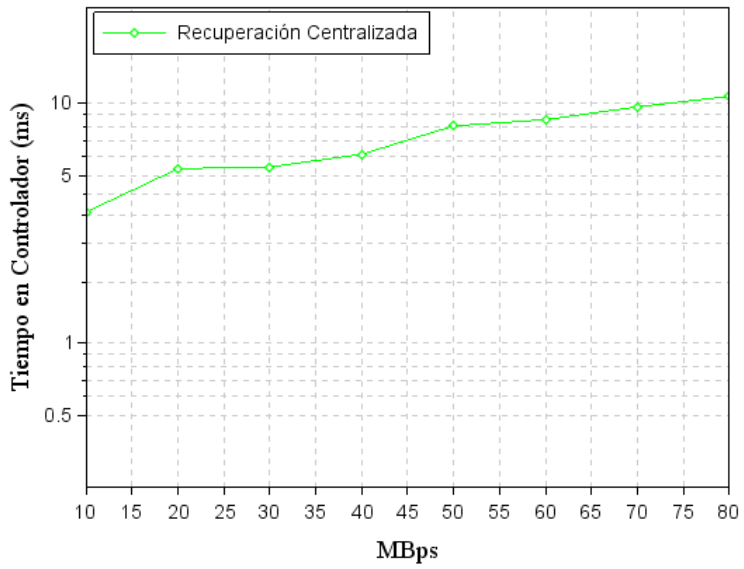


Ilustración 61: Gráfica comparativa sobre el tiempo de procesamiento en controlador para recuperación

Como se puede observar el tiempo de procesamiento en controlador para la recuperación basada en controlador aumenta según aumente la afluencia de paquetes recibidos por este, mientras que en el caso de la recuperación distribuida no existe procesamiento alguno en el controlador.

6 Conclusiones y trabajos futuros

6.1 Conclusiones

Al final de este proyecto se puede obtener varias conclusiones:

Se ha realizado un estudio sobre la nueva propuesta arquitectónica "SDN". En ella se observa una división en tres capas totalmente separadas, tanto estructural como funcionalmente. Se observa también que esta arquitectura centraliza la "inteligencia" de las redes actuales en una capa de controlador, donde se realiza todo el procesado de los paquetes de datos. Además se observa que gracias a esta estructura se facilita mucho el control de la red por parte de los usuarios. Por último, citar que esta arquitectura muestra un problema de escalabilidad importante, ya que en redes grandes o con grandes tasas de tráfico necesita de un controlador muy potente y costoso.

Como se ha podido observar a lo largo de este trabajo, la arquitectura SDN necesita de una entidad "inteligente" denominada controlador. En la actualidad, el mercado ofrece muchos controladores tales como, ONOS, ODL o RYU. Este último ha sido el elegido para la realización de este proyecto, debido a su corta curva de aprendizaje y a su capacidad de adaptación para procesos de prototipado.

Al tratarse de sistema software, ha sido necesario el estudio del sistema bajo el que se va a ejecutar, en este caso MiniNet. Tras el estudio se ha podido comprobar que, aunque se trata un sistema que requiere de mucha potencia de cálculo y memoria, es un sistema robusto donde realizar las pruebas. Por otro lado, también se ha podido comprobar la versatilidad que ofrece MiniNet al disponer también de un sistema clúster.

El objetivo final de este proyecto trata de la implementación completa de un switch híbrido. Para ello se ha tomado la decisión de modificar un switch SDN ya implementado. Se ha realizado el estudio de dos de los principales switches existentes en el mercado, OvS y OfSoftSwitch, eligiendo como base de programación este último. La gran diferencia y razón por la que se ha elegido OfSoftSwitch es por la sencillez de su programación, ya que a diferencia de OvS se trata de un switch monoproceso que se encuentra programado en C y sobre espacio de usuario.

Como se comentado durante todo el proyecto, se ha de implementar un switch híbrido, es decir, unión de dos elementos diferentes. Como se ha podido ver hasta ahora, solo se ha comentado uno de ellos, SDN. El siguiente objetivo se trataba de estudiar el protocolo ARP-Path, el cual es el protocolo de generación de caminos autónomos del switch. Como se ha podido demostrar, se trata de un protocolo cuyo comportamiento se realiza en dos

fases diferentes: la primera, la propagación del paquete ARP-Request, gracias al cual se estable el camino de vuelta del flujo; y una segunda fase, que se lleva a cabo durante la propagación del paquete ARP-Reply, y gracias a la cual se obtiene el camino de ida del flujo.

Se ha requerido el diseño funcional del switch híbrido. Como se ha podido observar en el apartado 3, se ha establecido todos los diseños necesarios para abordar de la mejor manera posible la implementación de este proyecto.

Tras la creación de los diseños funcionales, se ha realizado la implementación de los mismos en el lenguaje del switch elegido para esta tarea. Como se ha podido comprobar en el capítulo 4, quedan reflejadas las funciones más importantes implementadas para el correcto funcionamiento del switch.

Se ha diseñado e implementado dos tipos de recuperaciones. Una basada en controlador, que en el caso de este proyecto se ha realizado según lo explicado en el apartado 2.8.1, para su posterior implementación a través de todas las funcionalidades sobre el controlador RYU, tal y como ha sido expuesto en el apartado 4.2. La segunda recuperación implementada es, a diferencia de la anterior, totalmente distribuida. Esta recuperación ha sido diseñada según lo explicado en el apartado 2.8.2 e implementada en el apartado 4.2.1

Para concluir con el proyecto, se propone como objetivo una fase de pruebas donde se desea comprobar varios apartados:

1. Tras los diferentes test a los que el sistema ha sido sometido, se puede llegar a la conclusión de que la implantación de un switch funcional basado en un protocolo ALL-Path ha sido conseguido.
2. Tras las pruebas realizadas con la recuperación centralizada, se puede concluir que el objetivo de retrocompatibilidad con las redes SDN queda conseguido, ya que dicha recuperación ha sido implementada como una aplicación de controlador. Por lo tanto, esta implementación debe cumplir todas las características de una aplicación SDN-OpenFlow.
3. Como se ha podido observar en los experimentos, la arquitectura SDN tiene un problema de escalabilidad. Si se analizan los resultados, se puede ver como cuando en la red aumenta el número de flujos, el procesamiento en el controlador, así como la cantidad de paquetes intercambiados, aumenta de manera exponencial generando un cuello de botella en ese punto. Por lo tanto, se puede obtener como conclusión que insertar un sistema autónomo que libere al controlador del cálculo de cientos o miles de rutas en una red, consigue aumentar la escalabilidad de dicha red. Esto queda demostrado en la Ilustración 55: Gráfico comparativa sobre

el uso de procesador en el controlador donde se puede observar que le procesamiento del controlador utilizando un sistema híbrido es un 90% menor que si se utiliza un sistema SDN.

4. También cabe citar que insertar un protocolo autónomo en la capa de infraestructura, sin insertar otro medio de aviso al controlador, impacta negativamente sobre éste. Es decir, SDN basa todo su planteamiento en un conocimiento total por parte del controlador, al hacer que este conocimiento no sea total sino parcial, provoca que el poder de dicho controlador se vea muy mermado y, por lo tanto, negativamente afectado.
5. Tras las pruebas realizadas, se pude comprobar como AOSS disminuye los retardos de las redes entre un 50% y 95%, pasando de 110ms en el peor caso a unos escasos 5ms gracias a la utilización de ARP-Path como protocolo de generación de caminos. También se puede observar, en la figura Ilustración 59: Gráfica comparativa sobre el rendimiento en la red, como AOSS es capaz de obtener un throughput similar a un protocolo mucho más complejo como los implementados en este trabajo sobre el controlador ShP e incluso mejor como en el caso del protocolo STP, en este caso la mejora obtenida es del 93%.

Tras comprobar la consecución de los diferentes objetivos marcados al principio del proyecto, se puede concluir que se ha cumplido el objetivo general del proyecto: "Implementación de AOSS, switch que interconecta los sistemas SDN centralizados con los protocolos de la familia ALL-Path estudiados por la universidad de Alcalá de Henares".

Como reflexión final cabe citar que el uso de sistemas híbridos debe ser el siguiente paso, ya que aprovechan mucho mejor las propiedades de los sistemas centralizados y de los sistemas distribuidos.

6.2 Trabajos Futuros

Las redes SDN actuales tienen varios problemas como se ha podido comprobar. El primer problema es de escalabilidad. Es decir, para redes grandes se necesita ampliar la potencia del controlador incluyendo máquinas cada vez más potentes y costosas. El segundo es la necesidad de incluir una segunda red paralela para que las comunicaciones entre switch y controlador no interfieran en las comunicaciones entre equipos, en resumen, las redes SDN actuales requieres una inversión económica considerablemente alta.

Por otro lado, existen actualmente estudios de nuevos protocolos como es TCP-Path, que permiten balancear el tráfico de una red aplicando sencillas reglas en los switches, por lo

tanto, se tiene aquí otra vía de estudio muy interesante para obtener un mejor rendimiento en las redes, balanceando de manera más óptima la carga en ellas.

Este trabajo ha intentado demostrar que existe una línea de investigación muy interesante, la hibridación o unión de las redes SDN centralizadas con las redes tradicionales. De esta manera se puede aprovechar de forma más óptima las mejores características de las dos redes, es decir, tener un sistema que es capaz de conocer la red de forma precisa es muy interesante para ciertos tipos de trabajo, pero tener otro sistema que es capaz de balancear la carga de trabajo de forma distribuida también lo es. Si se asignan las tareas pertinentes a cada una de las partes, se puede obtener una mejora sustancial en el rendimiento global.

Por último se puede generar en el controlador un sistema basado en Machine Learning, para así obtener una mejor respuesta del controlador. Esta respuesta se podría mejorar gracias a las predicciones del controlador.

En definitiva, siguiendo estas líneas de investigación, se pueden obtener mejoras de rendimiento en las redes.

.

7 Bibliografía

[1] Luis Arizmendi, «Ideas básicas sobre Software Defined Network (SDN)», 19-nov-2013. [En línea]. Disponible en: http://luisarizmendi.blogspot.com.es/2013/11/software-defined-network-sdn.html.

[2] O. N. Foundation, *OpenFlow Switch Specification 1.3.2*. Open Networking Foundation, 2013.

[3] A. García Centeno, C. M. Rodríguez Vergel, C. Anías Calderón, y F. C. Casmartiño Bondarenko, «Controladores SDN, elementos para su selección y evaluación», *Revista Telem@tica*, vol. 13, pp. 10-20, sep-2014.

[4] «On data center scale, OpenFlow, and SDN». [En línea]. Disponible en: http://bradhedlund.com/2011/04/21/data-center-scale-openflow-sdn/. [Accedido: 05-mar-2017].

[5] «SDN Controller Comparison Part 1: SDN Controller Vendors». [En línea]. Disponible en: https://www.sdxcentral.com/sdn/definitions/sdn-controllers/sdn-controllers-comprehensive-list/. [Accedido: 05-mar-2017].

[6] «INTERNET/TELECOM/NETWORKING». [En línea]. Disponible en: http://www.bryternet.com.ng/. [Accedido: 05-mar-2017].

[7] G. Ibanez, J. A. Carral, J. M. Arco, D. Rivera, y A. Montalvo, «ARP-Path: ARP-Based, Shortest Path Bridges», *Commun. Lett. IEEE*, vol. 15, n.º 7, pp. 770-772, jul. 2011.

[8] «Definición de ARP». [En línea]. Disponible en: https://www.mastermagazine.info/termino/3908.php. [Accedido: 05-mar-2017].

[9] Francisco José Ibáñez García, «Estudio de las tecnologías SDN y NFV», Proyecto Final de Grado, 2016.

[10] Norberto Figuerola, «SDN – Redes definidas por Software». [En línea]. Disponible en: https://articulosit.files.wordpress.com/2013/10/sdn.pdf.

[11] Rafat Jahan, «Inter SDN Controller Communication (SDNi)».

[12] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, y S. Banerjee, «DevoFlow: Scaling Flow Management for High-performance Networks», en *Proceedings of the ACM SIGCOMM 2011 Conference*, New York, NY, USA, 2011, pp. 254–265.

[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, y A. Vahdat, «Hedera: Dynamic Flow Scheduling for Data Center Networks», en *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, Berkeley, CA, USA, 2010, pp. 19–19.

[14] B. Stephens, A. Cox, W. Felter, C. Dixon, y J. Carter, «PAST: Scalable Ethernet for Data Centers», en *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, New York, NY, USA, 2012, pp. 49–60.

[15] S. Hassas Yeganeh y Y. Ganjali, «Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications», en *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, New York, NY, USA, 2012, pp. 19–24.

[16] G. Bianchi, M. Bonola, A. Capone, y C. Cascone, «OpenState: programming platform-independent stateful openflow applications inside the switch», *SIGCOMM Comput Commun Rev*, vol. 44, n.º 2, pp. 44-51, 2014.

[17] M. Yu, J. Rexford, M. J. Freedman, y J. Wang, «Scalable flow-based networking with DIFANE», *SIGCOMM Comput Commun Rev*, vol. 41, n.º 4, p. , ago. 2010.

[18] *802.1AB: Link Layer Discovery Protocol (LLDP)*. .

[19] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, y P. Skoldstrom, «Scalable fault management for OpenFlow», en *Communications (ICC), 2012 IEEE International Conference on*, 2012, pp. 6606-6610.

[20] William Stallings, «Software-Defined Networks and OpenFlow - The Internet Protocol Journal, Volume 16, No. 1», *Software-Defined Networks and OpenFlow - The Internet Protocol Journal, Volume 16, No. 1*. [En línea]. Disponible en: http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html.

[21] «FUNCIONAMIENTO DE OPENFLOW», 22-feb-2015. [En línea]. Disponible en: http://electivaredesavanzadas.blogspot.com.es/.

[22] «Introduccion a SDN OpenFLow».

[23] Carlos Alberto Contreras Pardo, «Implementación De Un Openflow Controller Para El Manejo De Openflow Switches», Pontificia Universidad Javeriana, Bogotá, 2014.

[24] Pablo Yagües Fernádez, «Programación de redes SDN mediante el controlador POX», Universidad Politécnica De Cartagena, Cartagena, 2015.

[25] «Getting Started — Ryu 3.27 documentation». [En línea]. Disponible en: http://ryu.readthedocs.org/en/latest/getting_started.html. [Accedido: 24-nov-2015].

[26] «RYU SDN Framework».

[27] «Ryu application API». [En línea]. Disponible en: http://ryu.readthedocs.io/en/latest/ryu_app_api.html.

[28] Washington A. Velásquez Vargas, «Emulación de una red definida por software utilizando MiniNet», Escuela Técnica Superior de Ingenieros en Telecomunicaciones (ETSIT - UPM).

[29] «Introducción a los grupos de control (cgroups) de Linux», *Introducción a los grupos de control (cgroups) de Linux*. [En línea]. Disponible en: https://elpuig.xeill.net/Members/vcarceler/articulos/introduccion-a-los-grupos-de-control-cgroups-de-linux. [Accedido: 05-mar-2017].

[30] «Mininet Python API Reference Manual». [En línea]. Disponible en: http://mininet.org/api/annotated.html.

[31] lantz edited, «Cluster Edition Prototype», 06-nov-2014. [En línea]. Disponible en: https://github.com/mininet/mininet/wiki/Cluster-Edition-Prototype. [Accedido: 05-mar-2017].

[32] «Fujitsu Develops Technology Employing 10 Gbps Virtual Switch to Substitute for On-Server Virtual Switch Functions», 11-jun-2010. [En línea]. Disponible en: https://phys.org/news/2010-06-fujitsu-technology-gbps-virtual-substitute.html.

[33] «Open vSwitch». [En línea]. Disponible en: http://openvswitch.org/. [Accedido: 27-mar-2015].

[34] *The Open vSwitch Database Management Protocol*. .

[35] E. Leão Fernandes y C. Esteve Rothenberg, «OpenFlow 1.3 Software Switch», presentado en In Salão de Ferramentas XXXII Simpósio Brasileiro de Redes de Computadores - SBRC'2014, Florianópolis, 2014.

[36] E. Leão Fernandes y C. Esteve Rothenberg, «OpenFlow 1.3 Software Switch», presentado en In Salão de Ferramentas XXXII Simpósio Brasileiro de Redes de Computadores - SBRC'2014, Florianópolis, 2014.

[37] Topología de malla, «Topologias», 25-mar-2009. [En línea]. Disponible en: http://topo-malla.blogspot.com.es/.

[38] María Angélica Casillas Gallegos y Ricardo Domínguez Ruíz., «Topología de redes», *Blog*. [En línea]. Disponible en: http://redestipostopologias.blogspot.com.es/2009/03/topologia-de-redes.html. [Accedido: 05-mar-2017].

[39] A. Greenberg *et al.*, «VL2: a scalable and flexible data center network», *SIGCOMM Comput Commun Rev*, vol. 39, n.º 4, pp. 51-62, 2009.

[40] «Introducción y configuración del Spanning Tree Protocol (STP) en los switches Catalyst», *Introducción y configuración del Spanning Tree Protocol (STP) en los switches Catalyst*, Abril-2008. [En línea]. Disponible en: http://www.cisco.com/cisco/web/support/LA/7/73/73037_5.html.

[41] techopedia, «Bridge Protocol Data Unit (BPDU)». [En línea]. Disponible en: https://www.techopedia.com/definition/793/bridge-protocol-data-unit-bpdu. [Accedido: 05-mar-2017].

[42] smerlin Oscar p.s, «Que es una Vlan y su función», 18-jul-2015. [En línea]. Disponible en: http://redesconfiguracion.blogspot.com.es/2015/07/que-es-una-vlan-y-su-funcion.html. [Accedido: 05-mar-2017].

[43] «Posts Tagged 'Spanning Tree Protocol'», 03-jul-2016. [En línea]. Disponible en: https://niktips.wordpress.com/tag/spanning-tree-protocol/. [Accedido: 05-mar-2017].

[44] «STP Protocolo». [En línea]. Disponible en: http://todosobreredesdedatos.blogspot.com.es/p/stp_5.html. [Accedido: 05-mar-2017].

[45] «Per VLAN Spanning Tree (PVST)». [En línea]. Disponible en: http://www.cisco.com/en/US/tech/tk389/tk621/tk846/tsd_technology_support_sub-protocol_home.html. [Accedido: 05-mar-2017].

[46] Jose R. Torrico, «Backbone Fast de Cisco: Teoría y comandos de configuración», 07-feb-2016. [En línea]. Disponible en: https://supportforums.cisco.com/es/blog/12882841. [Accedido: 05-mar-2017].

[47] Cisco, «Understanding and Configuring the Cisco UplinkFast Feature». [En línea]. Disponible en: http://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/10575-51.html. [Accedido: 05-mar-2017].

[48] «Configuring Spanning Tree PortFast, BPDU Guard, BPDU Filter, UplinkFast, BackboneFast, and Loop Guard», 06-may-2007. [En línea]. Disponible en: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst4000/8-2glx/configuration/guide/stp_enha.html. [Accedido: 05-mar-2017].

[49] «Per VLAN Spanning Tree Plus (PVST+)». [En línea]. Disponible en: http://www.cisco.com/c/en/us/tech/lan-switching/per-vlan-spanning-tree-plus-pvst/index.html. [Accedido: 05-mar-2017].

[50] «Introducción al Rapid Spanning Tree Protocol [Spanning Tree Protocol rápida] (802.1w)». [En línea]. Disponible en: http://www.cisco.com/cisco/web/support/LA/7/75/75983_146.html. [Accedido: 05-mar-2017].

[51] «Understanding Multiple Spanning Tree Protocol (802.1s)». [En línea]. Disponible en: http://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/24248-147.html. [Accedido: 05-mar-2017].

[52] «IEEE 802.1aq™. Draft Standard for Local and Metropolitan Area Networks---- Virtual Bridged Local Area Networks---- Amendment 4: Shortest Path Bridges. IEEE». .

[53] Juan Antonio Carral Pelayo, «Contribución al Diseño de Conmutadores Transparentes Avanzados Basados en Tecnología Ethernet», Universidad de Alcalá, Alcalá, 2013.

[54] Eduardo Collado, «Protocolos Estado del Enlace», 04-may-2009. [En línea]. Disponible en: http://eduangi.org/node123.html. [Accedido: 05-mar-2017].

[55] Cisco, «Intermediate System-to-Intermediate System Protocol». [En línea]. Disponible en: http://www.cisco.com/en/US/products/ps6599/products_white_paper09186a00800a3e6f.shtml. [Accedido: 05-mar-2017].

[56] Guillermo Agustín Ibáñez Fernández, «Ethernet Avanzado (Bridging)», Universidad de Alcala.

[57] R. Harel y R. Solomon, *Connectivity fault management (CFM) in networks with link aggregation group connections*. Google Patents, 2010.

[58] «Transparent Interconnection of Lots of Links (trill) -». [En línea]. Disponible en: https://datatracker.ietf.org/wg/trill/charter/. [Accedido: 02-may-2015].

[59] Xavier Ramón, «TRILL: La revolución de Spanning Tree Protocol». [En línea]. Disponible en: http://blogs.salleurl.edu/networking-and-internet-technologies/trill-la-revolucion-de-spanning-tree-protocol/. [Accedido: 05-mar-2017].

[60] «Power Line Communications». [En línea]. Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/salvador_s_g/capitulo2.pdf. [Accedido: 05-mar-2017].

[61] TelcoNotes, «Switch Learning and Forwarding», MAR 9. [En línea]. Disponible en: https://telconotes.wordpress.com/2013/03/09/how-a-switch-works/. [Accedido: 05-mar-2017].

[62] Flowgrammable, «PacketIn», *PacketIn*. [En línea]. Disponible en: http://flowgrammable.org/sdn/openflow/message-layer/packetin/#PacketIn_1.3.

[63] flowgrammable, «PacketOut», *PacketOut*. [En línea]. Disponible en: http://flowgrammable.org/sdn/openflow/message-layer/packetout/#tab_ofp_1_3_3.

[64] flowgrammable, «FlowMod», *FlowMod*. [En línea]. Disponible en: http://flowgrammable.org/sdn/openflow/message-layer/packetout/#tab_ofp_1_3_3.

[65] «CPqD/ofsoftswitch13», *GitHub*. [En línea]. Disponible en: https://github.com/CPqD/ofsoftswitch13. [Accedido: 27-mar-2015].

[66] «Yen's K-Shortest Path Algorithm», *Yen's K-Shortest Path Algorithm*. [En línea]. Disponible en: https://github.com/Pent00/YenKSP. [Accedido: 05-mar-2017].

[67] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, y T. Gayraud, «Software-Defined Networking: Challenges and research opportunities for Future Internet», *Comput. Netw.*, vol. 75, Part A, pp. 453-471, 2014.

[68] P. Hansen y M. Zheng, «Shortest shortest path trees of a network», *Discrete Appl. Math.*, vol. 65, n.º 1, pp. 275-284, 1996.

8 Anexos

8.1 Anexo A: Planificación Temporal

8.1.1 Esquema Temporal

Tabla 10: Esquema Temporal

Proyecto	Fecha inicio prevista	Dias trabajados	Fecha final prevista	Situación	Dias para el final
Anílisis Arquitectura SDN	23-feb.-16	7	1-mar.-16	Terminado	0
Estudio y configuración de máquina mininet	1-mar.-16	7	8-mar.-16	Terminado	0
Análisis del Protocolo OpenFlow	8-mar.-16	7	15-mar.-16	Terminado	0
Análisis Estructura Switch	15-mar.-16	7	22-mar.-16	Terminado	0
Estudio y Compresión Protocolos All-Path	22-mar.-16	7	29-mar.-16	Terminado	0
Diseño de funciones Arp-Path	29-mar.-16	14	12-abr.-16	Terminado	0
Implementacion Funciones Arp-Path	12-abr.-16	90	11-jul.-16	Terminado	0
Implementacion Funciones TCP-Path	11-jul.-16	14	25-jul.-16	Terminado	0
Implementación Sistemas de Recuperacion	25-jul.-16	14	8-ago.-16	Terminado	0
Pruebas Switch Arp-Path	8-ago.-16	45	22-sep.-16	Terminado	0
Pruebas Switch TCP-Path	22-sep.-16	14	1-nov.-16	En curso	26
Implementacion Funciones TFE-Path	1-nov.-16	-26	8-nov.-16	En curso	33
Pruebas Switch TFE-Path	8-nov.-16	-33	15-nov.-16	En curso	40

8.1.2 Diagrama de Gantt

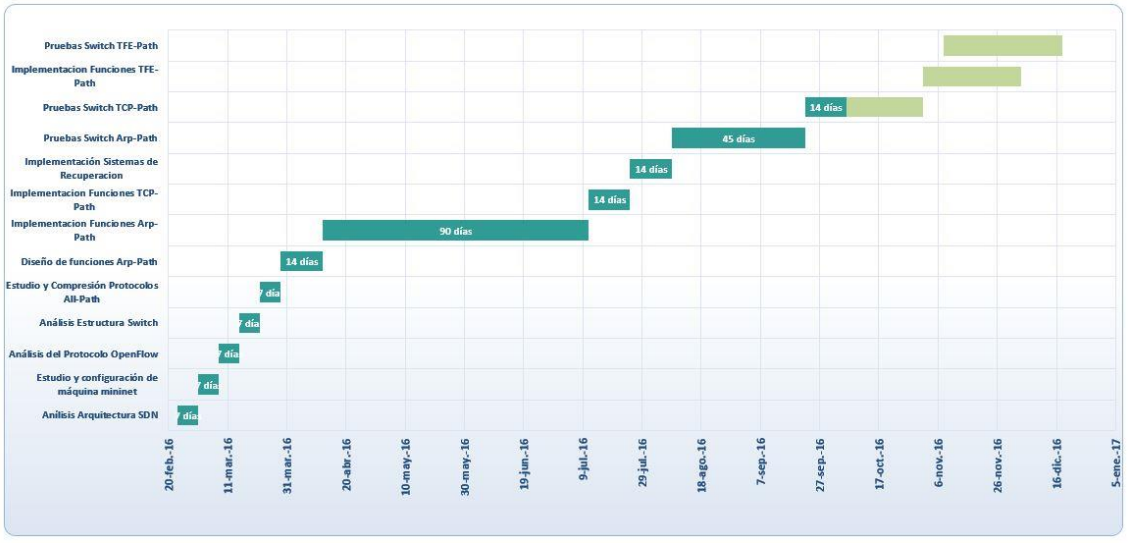


Ilustración 62: Diagrama de Gantt

8.2 Anexo B: Presupuesto

Este presupuesto se ha realizado suponiendo que es un proyecto de I+D (investigación y desarrollo), por tanto no lleva visado del colegio de ingenieros, no es un proyecto clásico ni un proyecto industrial. No lleva honorarios de realización del proyecto.

El presupuesto le vamos a dividir en tres sub apartados:

- Coste total de materiales para construir el entorno de pruebas
- Coste total horas de ingeniería

8.2.1 Coste total de materiales

Tabla 11 - Coste total de materiales para construir el entorno de pruebas

Material	Cantidad	€/U	Total
Equipo Core i7 MSI	4	780€	3.120 €
Equipo Core i5	1	500€	500€
Switch GbEthernet	1	120€	120€
Cableado	3	60€	180€
SAIS	2	100€	200€
		Total	4.800€

8.2.2 Coste total de horas de ingeniería

Tabla 12 - Coste total de horas de ingeniería

Material	Cantidad	€/U	Total
Horas ingeniería	300	50€	15.000€
		Total	15.000€

8.2.3 Coste total

Tabla 13: Coste total

		Total	19.040€

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR