CONTRIBUCIONES EN ARQUITECTURAS DE REDES DE CONMUTADORES TRANSPARENTES ETHERNET DE ALTAS PRESTACIONES

Tesis Doctoral

Elisa Rojas Sánchez

Departamento de Automática
Escuela Politécnica Superior
Universidad de Alcalá

Alcalá de Henares, abril de 2013

Universidad de Alcalá

Escuela Politécnica Superior

Doctorado en Tecnologías de la Información y las Comunicaciones

Tesis Doctoral

CONTRIBUCIONES EN ARQUITECTURAS DE REDES DE CONMUTADORES TRANSPARENTES ETHERNET DE ALTAS PRESTACIONES

Autor: Elisa Rojas Sánchez
Ingeniero de Telecomunicación

Director: Guillermo Ibáñez Fernández
Doctor Ingeniero de Telecomunicación

Abril de 2013

Elisa Rojas Sánchez
Área de Ingeniería Telemática (Departamento de Automática).
Escuela Politécnica. Universidad de Alcalá.
Campus Universitario. Ctra. Madrid-Barcelona, Km.33.600
Alcalá de Henares (Madrid)
e-mail: elisa.rojas@uah.es

A mi familia,
amigos
y, en especial, a Álvaro
(para ver si se anima “con lo suyo” ☺)

Agradecimientos

En estos momentos en los que una está terminando de escribir el documento de su propia Tesis, en los que tiene más en la cabeza los tiempos y plazos burocráticos, es complicado ponerse de repente a escribir un apartado de “Agradecimientos”, pero no por ello quiero saltármelo, es que sí, en realidad quiero dar las gracias.

Quiero dar las gracias primero de todo un poco al “azar”, o no, de haber terminado donde estoy. De haber decidido estudiar para ser ingeniera de telecomunicaciones, de que me gustara y se me diera relativamente bien, de que me ofrecieran la beca Erasmus y trabajar en Telefónica I+D para terminar el proyecto de fin de carrera algo más tarde de lo que siempre me hubiese gustado. Esa “casualidad” que hizo que comenzara el máster en 2009 y que en 2010 me ofrecieran trabajar en el grupo de trabajo GIST-NETSERV de la Universidad de Alcalá del que formo parte ahora y gracias al que he podido realizar mi Tesis.

Pero en realidad no es ninguna suerte, pues todos los eventos anteriores tienen nombre y apellidos. Mis padres, por darme la oportunidad de hacer siempre un poco lo que he querido y guiarme allá donde fuera, por su confianza en mí. Mi hermano, Fernando Rojas, por hacerme aprender más que nunca al ser mi hermano pequeño, estudiar mi misma carrera, y requerir a veces mi ayuda en ciertas asignaturas, pero especialmente por descubrir así mi vocación por la enseñanza y hacerme así medio partícipe de su gran talento. Todos mis compañeros de colegio, instituto y universidad, que me motivaron para seguir delante de un modo u otro, en los que siempre encontré apoyo y que ahora algunos son grandes amigos. Especialmente quiero agradecer a Guillermo Ibáñez la gran oportunidad que me dio al ofrecerme trabajar en la Universidad de Alcalá, quizás él buscaba un compañero más del grupo de investigación, quizás cursé la asignatura correcta en el momento correcto y fue algo muy trivial, pero lo que está claro es que me dio una oportunidad de oro y en un momento en el que ya tenía medio asumido que trabajar y realizar la Tesis en la Universidad de Alcalá era algo muy complicado para mí. Finalmente, dar las gracias a todos mis compañeros de trabajo y estudiantes realizando el proyecto de fin de carrera, en especial a Diego Rivera, pues sin su esfuerzo y colaboración, quizás no estaría escribiendo estas líneas ahora.

Y seguro que me dejo a más de uno ☺ Gracias a todos.

Este trabajo ha sido parcialmente financiado por la Comunidad de Madrid y la Comunidad de Castilla-La Mancha a través de los proyectos MEDIANET-CM (s-2009/TIC-1468) y EMARECE (PII1I09-0204-4319)

Resumen

Las redes campus y de centros de datos requieren hoy en día un alto rendimiento y gestionabilidad, todo ello dentro de un coste razonable, especialmente en redes de centros de datos, en las que se tiende a utilizar un mayor número de dispositivos genéricos (*commodity*) en lugar de alternativas más complejas y de mayor precio por tanto. Las redes de conmutadores transparentes Ethernet se presentan como la primera opción en este ámbito. Sin embargo, el empleo del protocolo de árbol de expansión (*Spanning Tree Protocol*, STP) como paradigma para propagar los mensajes a través de la red y evitar bucles es una evidente limitación del desempeño y el tamaño de las redes Ethernet. Mientras que los dos recientes y principales estándares *Shortest Path Bridging* (SPB) y *Routing Bridges* (RBridges, TRILL) permiten la utilización de los todos los enlaces de la infraestructura para obtener caminos mínimos, pero utilizan un protocolo de estado de enlace, operando en capa dos, y no cumplen con el principio básico de simplicidad de los conmutadores transparentes puros de capa dos.

Esta tesis presenta diversas contribuciones en conmutadores transparentes Ethernet del tipo genéricamente denominado por puentes de caminos mínimos (*shortest path bridges*) para resolver las grandes restricciones que impone el protocolo de árbol de expansión, pero que se caracterizan por evitar el uso de algoritmos de encaminamiento, enfoque actualmente predominante en las propuestas tanto estándar como propietarias. Se evita así la complejidad de utilizar un protocolo de estado de enlace (como IS-IS en SPB y TRILL), mientras que a la vez se aprovecha el concepto básico de "difundir para aprender" de los conmutadores transparentes puros para explorar todos los caminos en la red usando todos los enlaces de la topología. Con este principio, introducimos la familia All-Path, una familia de protocolos de puentes transparentes que ofrece caminos de mínima latencia con reparto de carga automático y que se adapta a diferentes requirimientos de escalabilidad y balanceo de carga de las topologías. También presentamos Torii-HLMAC, un protocolo distribuido, tolerante a fallos, sin configuración, con direccionamiento y encaminamiento basado en árboles múltiples y reparación sobre la marcha, específico para redes de centros de datos, principalmente los llamados *fat trees*.

Palabras clave: protocolos, redes de computadores, conmutadores transparentes, direccionamiento jerárquico, caminos de mínima latencia, multicaminos, reparto de carga.

Abstract

At present, campus and data center networks need high performance and manageability, within a reasonable price, especially for data center networks in which the tendency is to use more commodity devices instead of more complex and expensive alternatives. Ethernet switched networks turn up to be the first option in this field. Nevertheless, the adoption of the *Spanning Tree Protocol* (STP) as the paradigm to propagate messages through the network and avoid loops is a manifest limitation of the performance and size of Ethernet networks. While the two main recent standards, *Shortest Path Bridging* (SPB) and *Routing Bridges* (RBridges, TRILL) allow utilization of all infrastructure links to obtain shortest paths, but these standards use a link-state routing protocol (IS-IS), operation at layer two, and fail to follow the inherent simplicity of pure layer two transparent bridges.

This thesis presents several contributions on Ethernet transparent bridges from the group generically denominated shortest path bridges to solve the big limitations that the spanning tree impose, but characterized to avoid the use of any routing algorithm, main approach followed by standards and proprietary proposals. In this way, the complexity of using a link-state routing protocol (like IS-IS in SPB and TRILL) is avoided, while taking advantage of the broadcasting-to-learn basic conception of pure transparent bridges in order to explore all paths in the network by using all the possible links of the topology. With this principle, we introduce the All-Path family, a family of Ethernet transparent bridging protocols that offers minimum latency paths with automatic load balancing, which can be adjusted to different topology requirements in scalability and load balancing. We also present Torii-HLMAC, a distributed, faul-tolerant, zero configuration protocol with multiple tree-based addressing and forwarding and on-the-fly path repair, specific for data center networks, primarily the so-called *fat trees*.

Keywords: protocols, computer networks, transparent bridges, hierarchical addressing, shortest path bridging, multipath, load balancing.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Introducción

En este primer capítulo se pretende realizar una breve presentación de la Tesis, pasando por mostrar el contexto de la misma, así como los problemas relacionados, que nos harán enumerar los objetivos propuestos, para finalmente pasar a describir la estructura del documento completo.

Esta Tesis se enmarca en la tecnología de conmutadores transparentes Ethernet tal y como indica el título de la misma. La tecnología de los conmutadores Ethernet ha demostrado gran versatilidad y capacidad de adaptación en distintos entornos, y con diferentes requisitos, convirtiéndose en una tecnología principal en entornos en los que anteriormente sería impensable utilizarla, puesto que su propósito inicial poseía una especificación concreta para cierto ámbitos y no la generalización que ha alcanzado. A continuación se presenta un resumen sobre la rápida evolución de la tecnología Ethernet en las últimas décadas.

1.2 La tecnología Ethernet

La tecnología Ethernet ha mostrado, a lo largo de sus ya 40 años de existencia una sorprendente capacidad de supervivencia. Ha pasado, gracias a sus altas prestaciones, su gran diversidad de velocidades que le permiten adaptarse a distintos entornos y necesidades y, sobre todo debido a sus economías de escala y a sus características de autoconfiguración y compatibilidad con sus versiones anteriores (*legacy*), de ser una tecnología de red local (*Local Area Network*, LAN) a ser la tecnología de interconexión y de transporte dominante en redes de mayor tamaño, sustituyendo sucesivamente a otras tecnologías más prometedoras o, incluso, ya asentadas en ciertos entornos como FDDI, ATM y SDH, y siendo por tanto aplicado en redes de mediano (*Metropolitan Area Network,* MAN) y gran tamaño (*Wide Area Network,* WAN), y hasta en redes de acceso y en redes del hogar de audio y video. En la base de esta evolución es fundamental el rol desempeñado por los conmutadores Ethernet, al asumir la función de conmutación de tramas, inicialmente residente en el medio físico de transmisión compartido, que limitaba fuertemente su capacidad.

Ethernet, inicialmente a 3 Mbps, ha evolucionado en capacidad desde 10 Mbps a 100 Gbps y de los simples puentes software que unían dos redes locales se ha pasado a los conmutadores de N*40 Gigabit, equipos con un potencial de conmutación equivalente al de los grandes enrutadores, proporcionando en todo momento familias de conmutadores con un coste por puerto y velocidad sin rival.

Este rápido desarrollo está planteando sustanciales cambios en la arquitectura de las redes locales, empresariales y, como caso específico, de los centros de procesamiento de datos (CPD), que pueden llegar a contener miles de ordenadores con requerimientos muy elevados de ancho de banda, con un estudio de ingeniería considerable, y que en el futuro, corresponderán a un elevado porcentaje de las comunicaciones en Internet. Los principales retos a los que se enfrenta la tecnología al ampliar no sólo el tamaño de las redes sino también las velocidades de trabajo y el entorno de servicios de aplicación se relacionan fundamentalmente con la necesidad de proporcionar garantías suficientes de escalabilidad, de rendimiento y de seguridad, manteniendo al mismo tiempo sus características de bajo coste, mínima gestión y mínima configuración.

Los conmutadores o puentes transparentes son un elemento clave en esta evolución, por ser el dispositivo base de conmutación en las redes basadas en la tecnología Ethernet, que permite alcanzar los destinos requeridos dentro de la topología, de una manera eficiente, sin necesidad de configurar tablas de encaminamiento ni protocolos de descubrimiento y/o cálculo de rutas (como ocurre en los enrutadores que operan en capa de red). Esta sencillez parte de la idea inicial de conmutador, como sistema simple de interconexión de segmentos, que aprendía en base a un sistema similar al de prueba y error: apuntado direcciones de origen en puertos de entrada y reenviando por el resto si el destino es desconocido, como si fuese un enlace único; en lugar de basarse en la predecibilidad de los enrutadores, que si desconocen un destino realizan precisamente la acción contraria: descartar la trama.

Así pues, la evolución de los puentes transparentes desde dichos sistemas sencillos a grandes sistemas, con cantidades muy elevadas de datos a reenviar, ha mostrado la gran versatilidad de este dispositivo, pero también se ha mostrado como un problema puesto que los puentes transparentes clásicos (802.1D y 802.1Q) no están previstos para satisfacer las nuevas necesidades de gestionabilidad, observabilidad, calidades de servicio controladas, caminos mínimos en la red, conmutación sin pérdidas impuestas por la aplicación de Ethernet en dichos campos. Algunos de los principales factores limitantes de la tecnología, que han obligado a sucesivos procesos evolutivos, y que tomaremos como referencia en el planteamiento de la Tesis, son los siguientes:

- El concepto de árbol de expansión como forma de difusión en la comunicación en la red sin que se produzcan bucles. Es importante recordar que los conmutadores propagan las tramas por todos sus puertos en caso desconocer cierto destino. El hecho de limitar la topología activa a un árbol sin bucles tiene consecuencias graves en el rendimiento de la misma como veremos en el capítulo dedicado al estado del arte.

- Las múltiples propuestas de puentes de caminos mínimos (*shortest path bridges*). A pesar de que el grupo de trabajo 802 del IEEE es el encargado de normalizar los puentes Ethernet, han surgido diversas propuestas de mejora tanto dentro del propio IEEE (actualmente con *Shortest Path Bridging*, SPB, como propuesta principal) como en otros organismos de estandarización como IETF (que presenta los RBridges del grupo de trabajo *Transparent Interconnection of Lots of Links*, TRILL) y en la comunidad científica en general. Estas propuestas normalmente combinan puentes y enrutadores por lo que presentan alta complejidad computacional y confusión de funciones con la capa de red y en la arquitectura de protocolos.

- La utilización de un sistema de direccionamiento plano, es decir, sin información topológica que facilite la localización del sistema destino. Las direcciones Ethernet o direcciones MAC (*Media Access Control*) vienen predefinidas por el fabricante según cierto esquema estandarizado y son únicas, lo que complica la localización de los dispositivos en la red al no existir correspondencia alguna entre la dirección Ethernet y la localización del mismo, al contrario que el caso de los enrutadores.

Por último, y no menos importante, en los últimos tres años los centros de datos han adquirido mucha importancia en Internet para alojar servicios y aplicaciones muy diversas (Google, Amazon EC2, Apple, etc), lo cual ha enriquecido esta problemática planteando requisitos muy exigentes de escalabilidad y altas prestaciones, con propuestas de topologías de red especializadas y protocolos muy focalizados para dichas redes.

En el capítulo dedicado al estado del arte se analizarán todas ellas en base a dos enfoques, el de las redes de encaminamiento empresarial y, como caso concreto de éste, el de las redes de centros de datos, que tiene su propia rama de investigación

en la comunidad científica. A continuación se resume el planteamiento y definición del problema, dando un giro de tuerca más y dejando a un lado dichas propuestas principales, SPB y TRILL, hacia un paradigma más sencillo que no dependa de protocolos de estado de enlace.

1.3 Planteamiento y contexto de la Tesis

Los objetivos que se plantean en el desarrollo de la Tesis consisten en la definición de nuevos paradigmas de conmutadores Ethernet que superen las limitaciones de los puentes transparentes actuales y satisfagan en lo posible los requisitos más exigentes en autoconfiguración, latencia y escalabilidad, manteniendo la estructura de trama Ethernet inalterada por razones de compatiblidad. Para ello el requisito de simplicidad se interpreta de forma radical con el fin de obtener protocolos muy escalables.

Respecto al protocolo de árbol expansión y su uso para frenar los bucles en la difusión, la presente Tesis pretende prescindir del mismo, relegándolo en todo lo posible a protocolo secundario (para compatibilidad y/o como protocolo de reserva en caso de fallo) con una función auxiliar, de uso opcional. Para ello se investiga la renovación del concepto y mecanismos de aprendizaje mediante difusión, recuperando sus orígenes y su principio en los puentes transparentes: la difusión como método para descubrir lo que aún no se conoce. Por lo tanto se evita el muy estudiado uso de protocolos de estado de enlace (como hacen SPB y TRILL), dejando atrás la tendencia dominante de hibridar los conmutadores con los enrutadores, aumentando por tanto su complejidad y enmarañando la arquitectura de comunicaciones, y volver a la esencia básica de los puentes transparentes, utilizando la difusión como una herramienta de descubrimiento y exploración de caminos de comunicación, y no como una lacra que se ha de evitar. Todo ello, por supuesto, sin perder las bondades ya mencionadas de la tecnología Ethernet.

Sobre estas bases nace en el grupo de investigación la familia All-Path: una familia de protocolos para conmutadores de altas prestaciones que tienen en común la exploración, aprovechando los mensajes difundidos en *broadcast*, y generación de caminos mínimos, de baja latencia, explorando todos los enlaces de la red. La familia All-Path comienza en 2009 con la concepción, en el marco de otra tesis doctoral [Car13], de un protocolo ARP-Path primitivo (denominado FastPath en su fase inicial), que utiliza la difusión de los mensajes ARP, que se intercambian de manera previa a toda comunicación, para explorar dichos caminos mínimos, de mínima latencia, sin necesidad de mensajes adicionales, ni configuración.

La presente tesis parte de la primitiva idea de ARP-Path (que a finales de 2009 incluía el protocolo Up/Down para evitar bucles de tramas) y explora en anchura y profundidad el nuevo espacio de diseño creado por el nuevo mecanismo de exploración, resolviendo problemas, encontrando simplificaciones, diseñando mecanismos adicionales resultado tanto del análisis teórico como de la experiencia práctica con las diversas implementaciones del protocolo, que lo harán evolucionar derivando en diferentes ramas o variantes, creando así una familia de protocolos,

aún en activa evolución, con diversidad de prestaciones y con posibilidades de aplicación en otras redes de conmutadores de muy reciente aparición como los *Audio Video Bridges* (AVB). Otros protocolos pertenecientes a la familia son Flow-Path (la versión basada en flujos de ARP-Path), ARP-Path* y ARP-Path multipath. Todos estos protocolos forman parte de dicha familia All-Path, propuesta de conmutadores transparentes para redes empresariales.

En el caso de la limitación del direccionamiento plano en Ethernet, que dificulta la localización de los destinos en la red, se plantea un segundo objetivo no menos ambicioso que aprovecha el caso concreto de las topologías específicas y conocidas de los centros de datos para plantear un direccionamiento jerárquico, que omita por tanto la necesidad siquiera de crear tablas de encaminamiento, pues el encaminamiento se realizará descodificando dichas direcciones jerárquicas. Una vez más, por supuesto, conservando las buenas características de la tecnología Ethernet y evitando que los dispositivos finales se vean afectados por el cambio.

De este segundo objetivo mencionado nace Torii-HLMAC, un protocolo surgido inicialmente como mejora directa de la propuesta de PortLand, protocolo específico para redes de centros de datos que se describe en el capítulo dedicado al estado del arte. Torii-HLMAC aprovecha la especificidad de las topologías utilizadas para redes de centros de datos, ya optimizadas y planificadas para un óptimo rendimiento, para crear una jerarquía que directamente hace que el uso de tablas de encaminamiento sea innecesario, pues el encaminamiento de tramas a través de la red se realiza con las HLMACs asociadas en los diferentes puentes (y estos a su vez, asignan HLMACs a los hosts a los que sirven) de la red. Torii-HLMAC es un protocolo que sólo requiere cierta baja configuración de cableado inicial y, a partir de ahí, posee encaminamiento sin necesidad de tablas, multicaminos para repartir tráfico entre los diversos caminos duplicados de la red y reparación de caminos automática sobre la marcha, eligiendo un camino alternativo en el mismo momento en el que se descubre el fallo, sin necesidad de ningún cálculo adicional.

1.4 Principales Contribuciones de la Tesis

Las principales contribuciones de la presente tesis pueden resumirse en:

- Concepción de variantes de protocolos de la familia All-Path tales como ARP-Path, Flow-Path (versión basada en flujos de ARP-Path), ARP-Path* y ARP-Path multipath.

- Análisis y simulación en profundidad del comportamiento dinámico del aprendizaje All-Path en sus diversas variante en diversos escenarios, concepción e implementación de mecanismos específicos de reparación de caminos.

- Análisis y evaluación del reparto de carga de los protocolos All-Path en sus diversas variantes mediante simuladores.

- Especificación, implementación y depuración de dichos protocolos en diversas plataformas, con la colaboración de expertos específicos en NetFPGA y OpenFlow.

- Exploración de protocolos focalizados en redes de centros de datos y particularmente en redes *fat tree*. Propuesta de protocolo distribuido Torii-HLMAC para mejora del protocolo PortLand y extensión a redes *fat tree* genéricas.

Así pues, las contribuciones que se exponen en la Tesis pueden agruparse en dos principales: la familia de protocolos All-Path y la arquitectura Torii-HLMAC. La familia All-Path posee buenas características para redes genéricas de conmutadores de pequeño y mediano tamaño, mientras que Torii-HLMAC se presenta como una propuesta para redes más específicas, en las que también se podrían presentar las bondades de All-Path, pero que en la práctica Torii-HLMAC incluso mejora al tener en cuenta el tipo de red.

1.5 Estructura de la Memoria

A fin de facilitar la lectura de la Tesis y dar una visión general de la misma, se describe a continuación su estructura y el contenido de los capítulos. En el capítulo actual, Capítulo 1, se han presentado las bases de la Tesis en un resumen que contiene brevemente el contexto de la misma, la definición del problema y un avance de las contribuciones a presentar. El Capítulo 2 es un capítulo exclusivamente dedicado al estado del arte y dividido en dos secciones principales: encaminamiento en redes empresariales y encaminamiento en redes de centros de datos. El Capítulo 3 presenta el planteamiento de la Tesis, con un detalle mayor sobre la definición del problema, análisis del espacio de soluciones, el proceso de diseño y las conclusiones. En los siguientes dos capítulos se desarrollan las diferentes contribuciones de la Tesis ordenadas de manera análoga al estado del arte: en el Capítulo 4 se exponen las contribuciones al encaminamiento en redes empresariales, presentando la familia All-Path y su evolución, y en el Capítulo 5 las contribuciones al encaminamiento en redes de centros de datos, describiendo el protocolo Torii-HLMAC. Finalmente, el Capítulo 6 está dedicado a las conclusiones y el trabajo futuro.

Nótese que los Capítulos 4 y 5 contienen no sólo la presentación teórica de las contribuciones, sino también la práctica, con diversas implementaciones y resultados, pues fue la combinación de teoría y práctica la que dio lugar a la clara evolución de los protocolos, sobre todo en el caso de la familia All-Path, y se consideró ésta como la manera natural de exponerlo por tanto.

Capítulo 2

Estado del arte

En este capítulo se revisa el estado del arte de las principales tecnologías de encaminamiento en redes empresariales y centros de datos, con especial énfasis en estos últimos por ser el entorno de red más exigente y de importancia cada vez mayor en Internet. En el caso de redes empresariales, la agrupación se realiza por tecnología, por semejanza en el tipo de soluciones propuestas y, dentro de las mismas, por orden cronológico. Para las redes de centros de datos, se realizan tres divisiones principales, de tres grupos de investigación de la comunidad científica que han realizado diversas propuestas, evolucionándolas a lo largo de los últimos cuatro años, la mayoría publicadas en la importante conferencia del ámbito de las comunicaciones ACM SIGCOMM.

Finalmente, en un último apartado se revisa el estado del arte de las diferentes herramientas utilizadas a lo largo del desarrollo de la presente Tesis, pues éstas han sido, junto al análisis teórico, un elemento clave en el desarrollo y evolución de las contribuciones que se exponen en la Tesis.

2.1 Encaminamiento en redes empresariales

En redes empresariales existe una creciente tendencia al uso de bridges autoconfigurables para construir redes de pequeños y mediano tamaño sin necesidad de grandes esfuerzos de configuración. Actualmente existen dos ramas principales de investigación, por un lado la que se considera por el IEEE la evolución lógica de estas redes y del árbol de expansión (tecnología predominante, pero ya obsoleta y limitada frente a las características que aportan muchas de las nuevas tecnologías en este ámbito), que es *Shortest Path Bridging* (SPB), y por otro, la visión del IETF que parte de jugar entre los ámbitos de las capas 2 y 3, creando bridges a medio camino entre los clásicos y los routers, que es *Transparent Interconnection of Lots of Links* (TRILL). Así pues, este apartado se divide en estas dos ramas principales, con sus respectivas evoluciones cronológicas.

2.1.1 Protocolos de Árbol de Expansión

En esta sección se introducen brevemente los protocolos de árbol de expansión, la cual fue la primera idea en torno al encaminamiento en base a bridges. El funcionamiento de los bridges transparentes consta de tres mecanismos básicos:

- Construcción de un árbol de expansión.
- Difusión de las tramas por el árbol de expansión.
- Aprendizaje de direcciones MAC asociadas a cada puerto.

La función del árbol de expansión es facilitar una estructura de difusión de las tramas por toda la red de forma que cada trama llegue a todos los nodos, conservándose el orden de las tramas y sin duplicación de las mismas ni formación de bucles. Esta estructura de difusión es el conocido Árbol de Expansión (*Spanning Tree,* STP). El problema de construir un Árbol Mínimo de Expansión es un problema clásico dentro de los algoritmos centralizados destacando los algoritmos de Prim [Pri57] y Kruskal [Kru56]. El algoritmo de Prim construye el árbol de mínima longitud partiendo de un nodo añadiendo sucesivamente el nodo de menor coste de unión al árbol. Es una derivación del algoritmo de Djikstra de caminos mínimos (SPF) y se diferencia en que Djikstra incorpora en cada paso el nodo con menor de menor coste total hasta el nodo origen. El algoritmo de Kruskal va seleccionando los enlaces de menor coste de la red y forma un bosque de árboles disjuntos que se terminan uniendo en un único árbol.

A lo largo de la sección veremos que el mecanismo estándar hasta 2004 para difusión en capa 2 en las redes Ethernet era el Protocolo de Árbol de Expansión (AE)[STP]. El AE minimiza la distancia desde el bridge raíz hasta cada nodo de la red, pero las distancias entre nodos a través del AE no son mínimas y además el tráfico se distribuye atravesando el bridge raíz, lo que crea cuellos de botella en sus

proximidades. Por último, STP está basado en temporizadores y tiene tiempos de convergencia superiores a 30 segundos, lo que es difícilmente aceptable. Tras STP, fue estandarizado el Protocolo Rápido de Árbol de Expansión (*Rapid Spanning Tree Protocol*, RSTP) [RSTP], que reduce el tiempo de convergencia ante cambios de topología al rango desde decenas de milisegundos hasta un segundo, haciendo uso de la circunstancia de que los enlaces Ethernet en las redes actuales son de forma predominante enlaces dedicados. Pero RSTP sigue utilizando un árbol de expansión como topología activa para la difusión, por lo que adolece de los restantes inconvenientes de STP mencionados más arriba, principalmente que los caminos no son mínimos y el tráfico se concentra alrededor del bridge raíz.

Tanto STP como RSTP se pueden considerar protocolos de árbol de expansión único, pues sólo se genera un árbol para todas las comunicaciones. Posteriormente aparecieron propuestas de árboles de expansión con caminos alternativos como DLS/GDLS (*Distributed Load Sharing*), su sucesor STAR (*Spanning Tree Alternate Routing Protocol*) y, finalmente, RSTAR (*Rapid Spanning Tree Alternate Routing Protocol*), que permiten utilizar algunos enlaces deshabilitados por el árbol de expansión, siempre que cumplan ciertos requisitos.

También surgen ideas sobre árboles de expansión múltiples, como es *Multitree* (MT; Sincoskie y Cotton [SC88]), que asigna cierto árbol según una función aleatoria entre MAC origen y destino, o los *Source Dependent Spanning Trees* (SDS; propuesta [RF91]), algo más sofisticado creando árboles de expansión generados dinámicamente en función del sistema final. Otro estándar relativamente reciente es el Protocolo de Árbol de Expansión Múltiple (*Multiple Spanning Tree Protocol*)[802.1s]. MSTP está basado en la configuración de instancias múltiples simultáneas de árboles de expansión en una determinada *región.* Se asignan mediante configuración diferentes árboles, prediseñados manualmente, a las distintas VLANs, lo que permite encaminar el tráfico separadamente por redes virtualmente independientes implementadas sobre una única red física. De esta forma se distribuye mejor el tráfico y mejora notablemente la utilización de la infraestructura de red porque los árboles múltiples habilitan y utilizan enlaces que serían cortados por el protocolo (caso de utilizar solamente STP o RSTP) para evitar la formación de bucles.

Pero el uso de MSTP presenta varios inconvenientes que restringen severamente su uso. El principal es la complejidad de diseño y configuración de los árboles múltiples, planificando sus bridges raíz respectivos y la asignación de VLANs a árboles, que debe ser configurada de forma idéntica en todos los bridges de la región MSTP. La configuración adecuada de VLANs en la red es delicada porque puede segmentarse la red en caso de fallo de un enlace al no poderse reencaminar alguna VLAN. Otro inconveniente es el diseño de dichas regiones MSTP dentro de las que operan estos árboles múltiples. Finalmente, los caminos utilizados en los árboles de expansión no son mínimos.

Así pues, tras presentar brevemente todos estos protocolos de árbol de expansión, la sección termina con un apartado detallado sobre SPB (*Shortest Path Bridging*), un último paso en esta evolución, finalmente estandarizado en marzo de 2012, que presenta una alternativa que sigue haciendo cierto uso del árbol de expansión, pero de manera diferente a los anteriores casos, creando siempre

caminos mínimos para todas las comunicaciones y, a su vez, manteniendo compatibilidad con los estándares anteriores.

2.1.1.1 Protocolo de Árbol de Expansión (STP)

El Protocolo de Árbol de Expansión (*Spanning Tree Protocol*, STP) se basa en el algoritmo creado por Radia Perlman [Per85] y fue incorporado al estándar IEEE 802.1D. El algoritmo minimiza el coste de cada nodo al nodo elegido como raíz. Se basa en el intercambio de unidades de protocolo de bridge (BPDUs) entre los bridges de una red. Cada bridge envía BPDUs a sus vecinos tras recibir una BPDU procedente del bridge raíz. El proceso de formación del árbol de expansión en una red consta de tres pasos: Elegir el bridge raíz, elegir el puerto raíz en cada bridge (puerto que conecta al árbol hasta el bridge raíz) y elegir los puertos designados (puertos que prolongarán el árbol hacia abajo a otros bridges). La identidad de bridge *(bridge ID)* se compone de un prefijo de prioridad configurable de 16 bits y un sufijo de 48 bits que contiene la dirección MAC del bridge. Los bridges eligen como bridge raíz el de menor identidad de bridge. En este proceso todos los bridges comienzan "autorreivindicándose" como bridge raíz mientras no reciben BPDUs de otros bridges con bridge ID menor que la suya. Tras un tiempo de intercambio de BPDUs entre los bridges, todos los bridges coinciden en la elección del bridge con menor bridge ID de la red. La elección de puerto raíz se hace con los criterios siguientes: cada bridge elige el de menor coste de camino hasta el bridge raíz y los posibles empates se resuelven eligiendo el de menor identidad de bridge designado de la BPDU y si hubiera empate, el de menor identidad de puerto designado.

El procesado de las BPDUs es como sigue: el bridge raíz envía periódicamente (cada *Hello_Time*, dos segundos por defecto) BPDUs de configuración del protocolo STP a los bridges vecinos. Estas BPDUs contienen la identidad del bridge raíz, la identidad del propio bridge, el coste del camino hasta el bridge raíz y la identidad del puerto por el que se envía la BPDU. Los bridges que las reciben generan con estos datos más los costes de sus enlaces una BPDU propia que se envía a su vez a los bridges vecinos. Este algoritmo no dispone de un mecanismo de detección de terminación o convergencia del mismo en la red sino que utiliza temporizadores que esperan el tiempo suficiente para asegurar que haya convergido. STP por ello está basado en temporizadores fijos de 15 segundos para cambiar de estado los puertos, por lo que adolece de lentitud en la reconfiguración del árbol. La fiabilidad y disponibilidad se reducen con un mayor número de nodos ya que la probabilidad de reconfiguración se incrementa al aumentar el número de nodos. En la Figura 1 se muestra el diagrama de estados de puerto en STP y las transiciones permitidas. Tras la inicialización los puertos pasan a estado de bloqueo. Los estados estables en operación son *Blocking* y *Forwarding*. Los estados transitorios que atraviesa un puerto desde el estado *Blocking* hasta *Forwarding* son *Listening* and *Learning*. Cuando el algoritmo de árbol de expansión deduce que un puerto debe pasar a estado *Forwarding* primeramente lo pasa a estado *Listening* en que el puerto continua bloqueando el reenvío de tramas, pero recibe las BPDUs y las pasa al dispositivo para su proceso, pero no aprende direcciones MAC ni transmite BPDUs. Tras vencer el temporizador *Forward_Delay* el puerto pasa a estado *Learning* en el que el puerto aprende direcciones MAC, pero sigue sin reenviar tramas de usuario. El vencimiento del temporizador *Forward_Delay* causa el cambio de estado a

Forwarding, en el puerto además de aprender direcciones MAC y recibir y transmitir BPDUs reenvía tramas de usuario. La caducidad de la BPDU recibida está controlada por el temporizador *Max_Age*, período de validez de cada BPDU recibida.

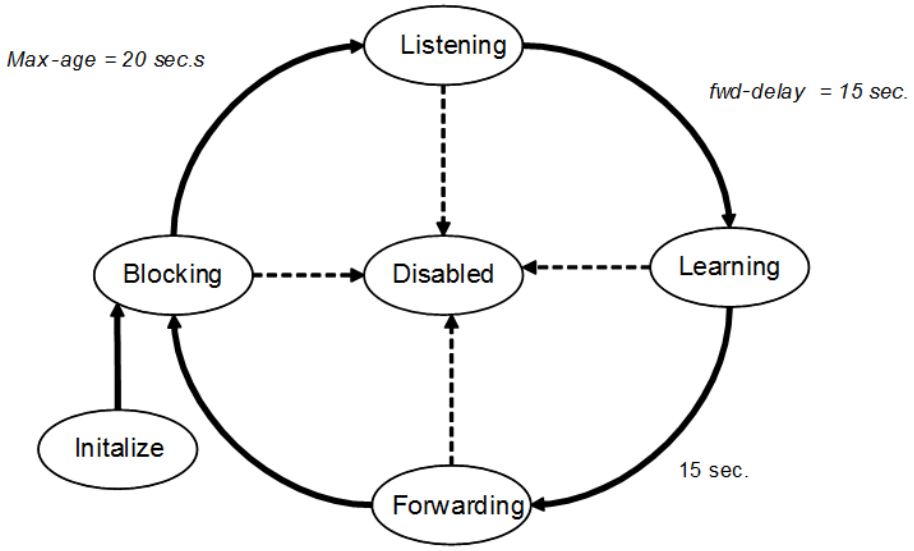


Figura 1. Estados de puerto en STP

2.1.1.2 Protocolo de Árbol Rápido de Expansión (RSTP)

El Protocolo Rápido de Árbol de Expansión (*Rapid Spanning Tree Protocol*, RSTP) fue estandarizado en 2001, inicialmente como 802.1w, enmienda al estándar IEEE 802.1D de 1998 y en 2004 fue incorporado al estándar 802.1D reemplazando a STP.

RSTP está implementado alrededor del vector de prioridad de puerto, vector que incluye la identidad de bridge raíz, identidad de bridge designado, coste al bridge raíz, prioridad de puerto e identidad de puerto, y que expresa su prioridad para ser elegido en el árbol de expansión. RSTP construye un árbol de expansión mediante anuncios de tipo vector distancia que contienen el coste al bridge raíz del árbol. En cada LAN, las BPDUs las transmite el puerto *Designado*. El puerto adquiere el rol de *Designado* al inicializarse o tras la elección de estado de puerto, para ello el puerto en cada extremo de la LAN compara su vector de prioridad con el recibido por la LAN por si debe pasar a ser designado. Se transmiten BPDUs siempre que cambia la información a transmitir, con una velocidad máxima configurable, además de periódicamente (*Hello_Time*). Cada bridge compara a nivel de puerto todos los vectores de prioridad recibidos de los vecinos y elige el mejor vector al bridge raíz. Se define como BPDU *superior* si el identificador (bridge ID) de bridge raíz es menor o el coste del camino al raíz es menor. RSTP reemplaza el mecanismo basado en temporizadores de 15 segundos empleado por STP para verificar que el algoritmo ha convergido con un mecanismo de propuesta-aprobación controlado localmente entre conmutadores vecinos para el cambio al estado *Forwarding* de los puertos en la dirección descendente del árbol de distribución. Este mecanismo requiere el uso de enlaces dedicados para funcionar correctamente sin crear bucles y se describe en la Figura 2. Mediante este mecanismo, el enlace entre el puerto raíz y su bridge designado se habilita (el puerto pasa a *Forwarding*) si previamente el bridge ha

bloqueado todos los puertos designados, con lo que se va ampliando la topología activa nivel a nivel, a la vez que un "corte" horizontal en la red se propaga desde el bridge raíz hacia abajo al bloquearse los puertos designados.

Otra característica propia de RSTP es que todos los conmutadores emiten autónomamente las BPDUs cada *Hello_Time*, en lugar de hacerlo tras la recepción de una BPDU del bridge raíz. Varios mecanismos adicionales contribuyen a obtener una reconfiguración rápida: cuando un conmutador recibe una BPDU con información menos preferente (es decir mayor identidad de switch raíz, Bridge ID, o coste de camino, métrica, al mismo), el switch receptor responde de inmediato al vecino con su propia BPDU para propagar la información superior de la que dispone. Con este mecanismo, un bridge que pierde su conexión con el bridge raíz recibirá rápidamente información de BPDU de los bridges vecinos que aún conservan su conexión al bridge raíz, seleccionando el que ofrezca la BPDU superior a efectos de elegir el nuevo puerto raíz. Aunque RSTP requiere enlaces punto a punto para el mecanismo de paso a *Forwarding*, no los requiere para la transición rápida de puerto raíz, entre el puerto que era raíz y el preseleccionado de reserva (rol de *Alternate*).

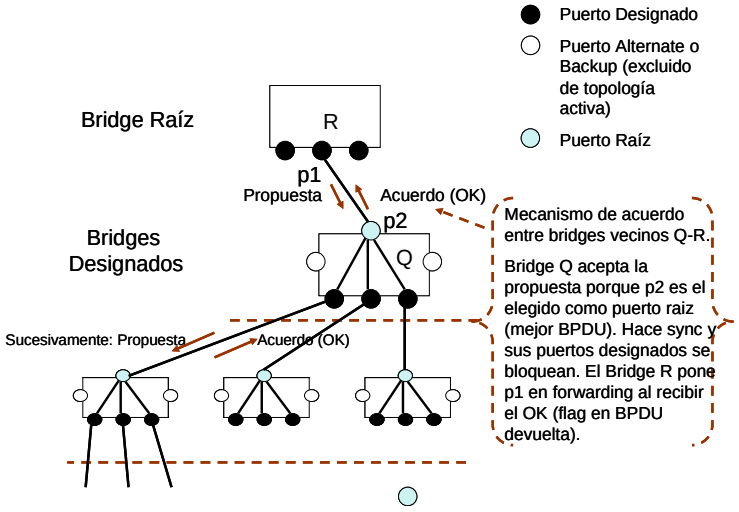


Figura 2. Mecanismo de RSTP para cambio de estado de puertos a *Designado*

Los estados de puerto, que eran cinco en STP, en RSTP quedan reducidos a tres: *Discarding*, *Learning* y *Forwarding.* El estado *Discarding* agrupa a los estados *Disabled, Blocking* y *Listening* de STP. En STP existía mezcla entre los conceptos de rol y el estado actual de puerto. En RSTP el rol de cada puerto es una variable del puerto, está vinculado al estado del puerto. Existen dos roles nuevos de puerto ligados al estado *Blocking* (en sentido STP) del puerto: *Alternate port* y *Backup port.* Estos estados de bloqueo del puerto se mantienen porque el puerto recibe mejores BPDUs de forma continuada por la LAN a la que está conectado, procedentes del mismo bridge (rol de *Backup port*) o de distinto bridge *(*rol de *Alternate port).* En caso de fallo del puerto raíz la elección del puerto Alternate permite una conectividad alternativa al bridge raíz inmediatamente disponible por estar precualificado para ser puerto raíz.

Este criterio de BPDU superior recibida es el elemento base de decisión en RSTP. Se comparan las BPDUs recibidas por diferentes puertos o con la información almacenada en el puerto. El puerto que recibe la BPDU superior asume el rol de puerto raíz (*root port).* Los puertos que pueden difundir la BPDU superior en el segmento de LAN al que están conectados asumen el rol de puertos designados.

RSTP envía BPDUs cada *Hello_Time* y las utiliza para hacer caducar la información recibida rápidamente (aparte de utilizar el temporizador *Max_Age*) y para detectar caídas en la comunicación entre bridges vecinos. Si un bridge no recibe ninguna BPDU en tres intervalos *Hello_Time* considera caída la comunicación con su vecino. La detección es más rápida y precisa que en STP dado que se detecta exactamente el enlace en fallo.

La Figura 3 muestra el formato estándar de BPDU para RSTP y el detalle del octeto de indicadores (flags). Estos indicadores se implementan con un bit de información cada uno y los bridges vecinos los emplean para comunicar y asentir los roles asignados a los puertos por el protocolo, estados de puerto y las transiciones de estado de los puertos.

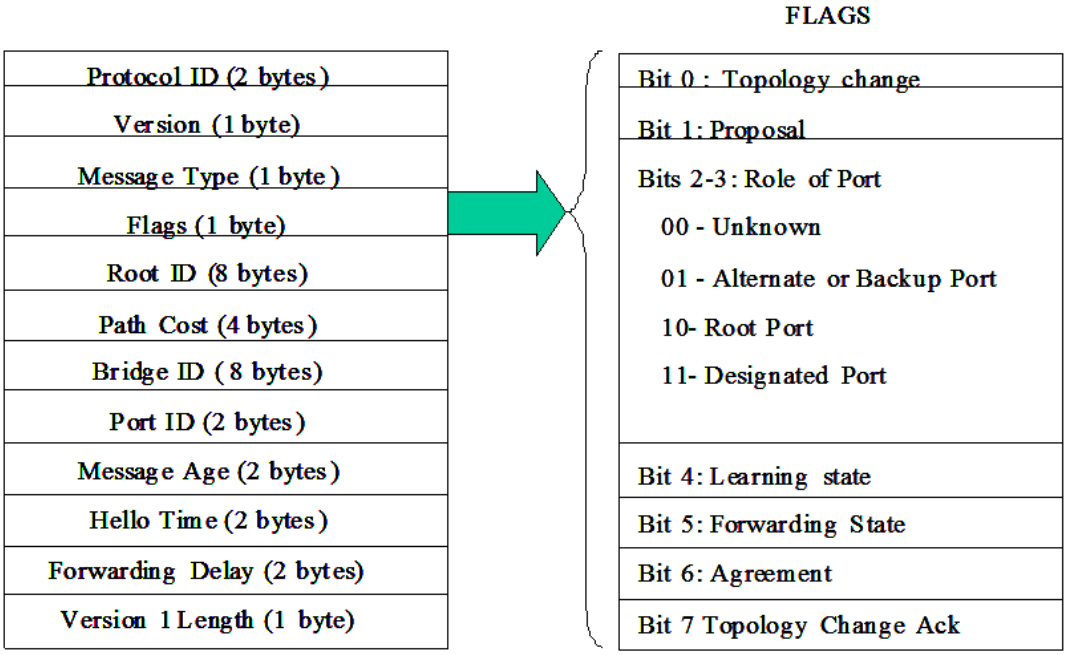


Figura 3. Formato de BPDU RSTP y detalle del octeto de indicadores (IEEE 802.1D)

Es destacable también un mecanismo proactivo para acelerar la reconfiguración consistente en que un puerto conectado a un puerto raíz del bridge contiguo, si recibe por el mismo una BPDU inferior que la suya, le contesta directamente con su información.

Los cambios de topología se comunican en la misma BPDU mediante el octeto de indicadores (flags) en vez de usar las BPDUs de cambio de topología (*Topology Change Notification*, TCN). Al enterarse de un cambio de topología el bridge lo propaga por todos los puertos excepto aquel por el que se ha recibido y se procede al borrado inmediato de las direcciones MAC aprendidas (por ese puerto).

El mecanismo de propagación de los cambios de topología es asimismo más rápido y completo que en STP, se propaga hacia abajo (además de continuar hacia arriba por el árbol de expansión) de inmediato desde cada nodo alcanzado al propagarse hacia arriba, en vez de propagarse primeramente hacia arriba hasta el bridge raíz y después hacia abajo por todo el árbol. RSTP es compatible con el protocolo de Agregación de Enlaces (enlaces paralelos en reparto de carga) ya incorporado en el estándar IEEE 802.3. Esto permite el reparto de carga entre varios enlaces paralelos sin que sean cortados por el protocolo y posibilitando el fallo de uno de ellos sin impacto adicional a la reducción de la capacidad total del grupo de enlaces agregados.

RSTP presenta tiempos de reconfiguración muy pequeños, si existe un puerto preseleccionado para suceder al puerto raíz solamente requiere el tiempo de detección de caída del enlace (tan corto como 10 mseg). Si no existe puerto preseleccionado, se requiere solamente la transmisión de una BPDU en cada sentido entre los puertos del nuevo enlace que va a ser raíz. No obstante, RSTP puede presentar un comportamiento de cuenta a infinito en caso de caída del bridge raíz, resultando en ese caso retardos de hasta varios segundos [MEZ04].

2.1.1.3 Protocolo de Árbol de Expansión Múltiple (MSTP)

Tras la definición de las redes locales virtuales (VLAN) y su estandarización en el estándar IEEE 802.1Q [VLAN] apareció la posibilidad de que estas nuevas redes virtuales montadas sobre una única red física, pudieran estar soportadas por diversos árboles de distribución, adecuados a su topología y aplicación. Para ello se desarrollaron diversos protocolos propietarios tales como *Per VLAN Spanning Tree* (PVST y PVST+) [Cisco] que permiten configurar diferentes árboles de distribución y asignarlos a diversas VLAN. Esta asignación se realiza por VLAN, por lo que puede consumir excesivos recursos para mantener en el conmutador la asignación de hasta 4094 instancias posibles de árbol, correspondientes a la asignación de un árbol por VLAN. La posterior estandarización del protocolo de árboles múltiples MSTP permitió que varias VLAN compartan un único árbol de expansión con el consiguiente ahorro de recursos.

El Protocolo de Árbol de Expansión Múltiple *(Multiple Spanning Tree Protocol*, MSTP) [802.1s] está basado en la configuración de instancias múltiples simultáneas de árboles de expansión en una determinada región. El protocolo MSTP provee de conectividad simple y completa a las tramas asignada a cualquier VLAN de una red local conmutada compuesta por bridges MSTP o RSTP. MSTP permite utilizar caminos separados a las tramas pertenecientes a diferentes VLAN, utilizando instancias independientes MST, instancias válidas dentro de cada región MST formada por redes LAN y bridges MST.

MSTP fue estandarizado en 2003 dentro del estándar para VLANs en bridges IEEE 802.1Q. Anteriormente existió como *IEEE 802.1s (draft) Multiple Spanning Tree*, como enmienda al estándar 802.1Q. MSTP es la extensión de RSTP para árboles de expansión múltiples. Esto permite una mejor utilización de la infraestructura y distribución de la carga, así como mejor resistencia a fallos por la multiplicidad de árboles construidos, al precio de la complejidad de configuración.

Prevé diversos dominios de administración con lo que las áreas MST pueden ser gestionadas por diferentes administradores. Existe un árbol común CST (*Common Spanning Tree*) y árboles por regiones, por lo que en caso de reconfiguración solo se afecta esa región y los tiempos de convergencia en redes grandes se reducen.

MSTP se apoya parcialmente en el protocolo RSTP, lo requiere y es compatible con él. Asimismo utiliza los mecanismos definidos para VLAN. Construye un conjunto de árboles de distribución independientes (instanciaciones o instancias) (*Multiple Spanning Tree Instance*, MSTI) en una región MST de la red. Dicha región se conecta a otras regiones MST, a efectos del control de protocolos STP, mediante un árbol de expansión común CST como se muestra en la Figura 4.

Dentro de una región, varias VLAN pueden asignarse a una sola instancia de árbol (*Spanning Tree Instance*, STI). Mediante el uso de múltiples instancias de árbol en la misma región, es posible un mejor aprovechamiento de los enlaces, utilizando enlaces que son bloqueados por el algoritmo de *Spanning Tree*. Dentro de cada región se mantiene su continuidad interna mediante un Árbol de Distribución Interno (*Internal Spanning Tree*, IST), identificado como instancia 0 (IST 0), que actúa como árbol básico de distribución dentro de la región. El CIST (*Common and Internal Spanning Tree*) o árbol total se compone del CST que conecta las regiones y del IST que realiza la conectividad dentro de cada región. Ello permite la gestión independiente de las regiones.

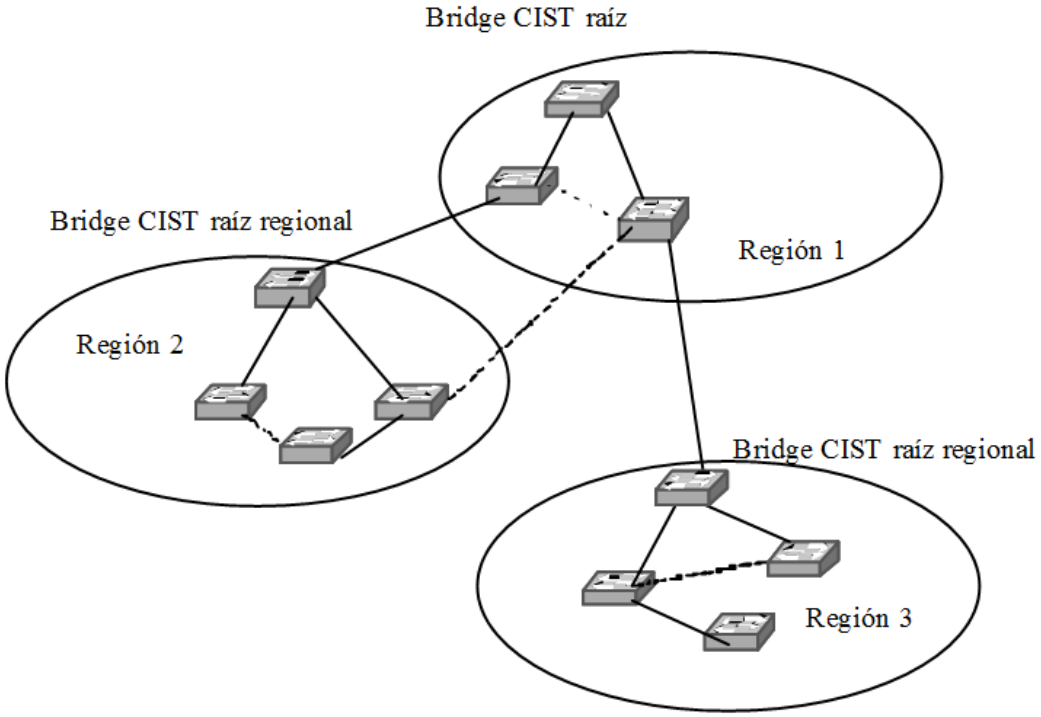


Figura 4. Regiones MST y Bridges Raíz Regionales en el árbol total CIST

La región aparece hacia el exterior como un único *superbridge* independiente. Es decir, toda la región se conecta al CST a través de un Puerto Raíz Regional y un cierto número de Puertos Designados, como si estuviera formada por un único bridge. Mediante mecanismos específicos se evita que el árbol IST salga de la región y vuelva a entrar en la misma. De esta forma los cambios externos de topología no provocan cambios internos. El bridge raíz del CST es la raíz de toda la red. Por cada

región MST hay un árbol IST maestro, el cual pertenece al CST. MSTP es adecuado para capa de distribución de redes campus, algo menos para troncales debido a la complejidad de configuración de las VLANs, aunque existen propuestas de sistemas de delimitación y configuración automática de regiones.

2.1.1.4 Shortest Path Bridging (SPB)

En marzo de 2005, bajo la denominación *Shortest Path Bridging* se presenta la propuesta de un nuevo proyecto que se acepta en septiembre de ese mismo año y que finalmente se aprueba como estándar a finales de marzo de 2012. Se trata del reemplazo de los antiguos protocolos de árbol expandido (IEEE 802.1D STP, IEEE 802.1w RSTP, IEEE 802.1s MSTP), que bloqueaban en todos los caminos excepto el usado, mientras que SPB permite a los caminos permanecer activos con múltiples caminos de igual coste. El estándar IEEE 802.1aq (*Shortest Path Bridging*, SPB) especifica encaminamiento mediante caminos mínimos de tramas unicast y multicast, incluyendo protocolos para calcular múltiples topologías activas que pueden compartir su información de localización aprendida y el uso de VLAN con múltiples identificadores (VIDs) por topología. Además permite que todos los enlaces estén activos con múltiples caminos de igual coste, así como el uso de topologías de capa 2 mucho mayores (hasta 16 millones en comparación con el límite de las 4096 VLANs), posee tiempos de convergencia más rápidos y mejora el uso de topologías en malla, incrementando el ancho de banda y la redundancia entre dispositivos. Todo esto manteniendo la compatibilidad con los estándares anteriores.

El funcionamiento del protocolo es el siguiente. En una topología de *N* nodos (bridges), STP/RSTP construiría un único árbol, mientras que en MSTP se podrían configurar múltiples árboles. SPB sin embargo, genera una instancia de árbol (STI o MSTI de la variante MSTP) por cada uno de los *N* nodos de la topología. Es evidente que si en una topología, se genera un árbol expandido con la raíz en el bridge A, todos los caminos hacia A serán mínimos (véase Figura 5).

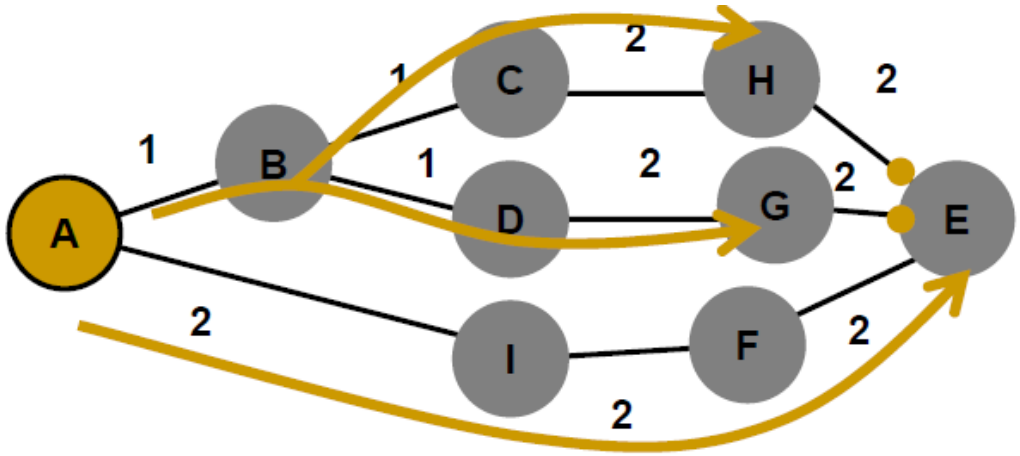


Figura 5. Generación de una instancia MSTI con raíz en el switch A en SPB

La tecnología de SPB facilita la construcción de redes Ethernet lógicas sobre infraestructuras Ethernet usando un protocolo de estado de enlace para anunciar tanto la topología, como la pertenencia a cierta red lógica. Los paquetes se

encapsulan en el switch frontera mediante MAC-in-MAC (IEEE 802.1ah) o tramas etiquetadas (IEEE 802.1Q/802.1ad) y son transportadas únicamente a los miembros de la red lógica. Soporta envío unicast y multicast y todo el encaminamiento se realiza en base a caminos mínimos simétricos, que pueden coexistir con otros caminos mínimos de igual coste. El plano de control se basa en IS-IS con un pequeño número de extensiones definidas en la RFC 6329 [Fed+12].

Para profundizar algo más en el funcionamiento de este protocolo, en los siguientes apartados se presenta poco a poco la concepción de SPB, sus aproximaciones iniciales [Fin05], con las diferentes necesidades que iban apareciendo y cómo se fueron solucionando, para finalmente introducir los conceptos de *Shortest Path Bridging-VID* (SPBV), *Shortest Path Bridging-MAC* (SPBM) y *Equal Cost Multi Tree* (ECMT), que fueron la solución final.

La necesidad de simetría en SPB

El problema inicial que aparece es que necesariamente los caminos de cualquier bridge a otro han de ser simétricos. Cojamos como ejemplo la Figura 6, en el que las VLANs enraízadas en los bridges A y E (es decir, los árboles construídos para cada bridge frontera A y E) están bloqueadas como muestra la figura. En ese caso, el camino desde E a A sería E-H-C-B-A, mientras que el camino de A a E sería A-I-F-E. Entonces imaginemos que A recibe una trama de un host con destino a otro host cuyo bridge frontera es E. Puesto que A desconoce la localización del host destino, propagará la trama a través de su propia MSTI A y alcanzará el switch E a través del enlace F-E, que la reenviará al host destino. Así pues, cuando el host destino responde, E ha aprendido que el camino de vuelta pasa por F y lo reenviará hacia él, para seguir el camino E-F-I-A. Sin embargo, en este caso se estará usando la MSTI E, que es diferente, y al llegar a A, este último switch descartará la trama porque el puerto se encuentra en estado bloqueado en esa instancia de árbol. Lo mismo sucedería en el caso de que un host origen en E quisiera comunicarse con otro destino en A, el puerto H-E del bridge E bloquearía la trama. Sin embargo, si los dos árboles expandidos son simétricos, por ejemplo árboles que usaran el camino A-B-C-H-E para la transmisión de tramas entre los dos bridges frontera y en las dos direcciones, el aprendizaje funciona perfectamente y no tendremos problemas.

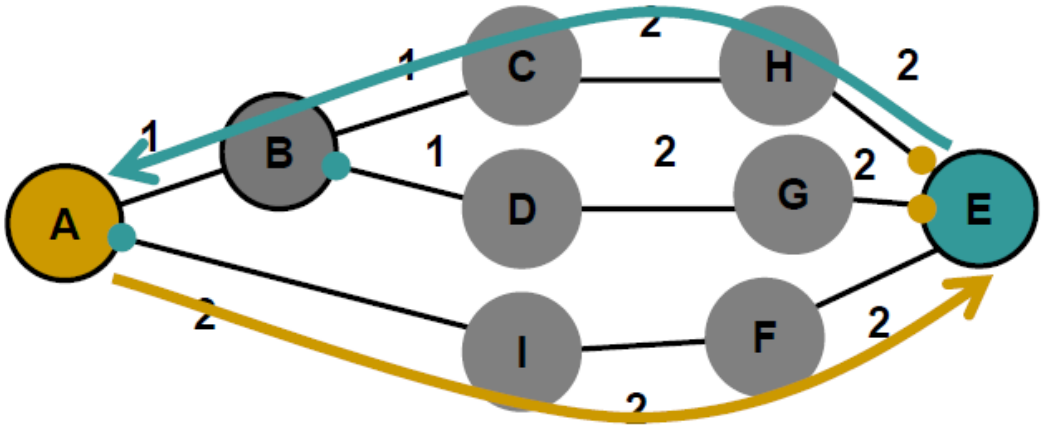


Figura 6. Ejemplo de caminos asimétricos imposibles en SPB

Una vez sabida la necesidad de simetría a la hora de construír los caminos en ambas direcciones con las respectivas MSTIs, llega otro problema: el de cómo configurar la red para que se consiga esa simetría de manera natural siempre.

Dado que los costes de los enlaces pueden ser configurados, así como calculados a partir de la velocidad del enlace, los dos bridges en los extremos de un enlace pueden no estar de acuerdo en el coste del enlace usado para el algoritmo de árbol expandido. Esto haría la simetrización imposible. De manera similar, las prioridades de los bridges pueden ser configuradas de manera diferente para diferentes instancias de árbol expandido, lo que también causa asimetría.

Por lo tanto, hay que esforzarse algo más para asegurar que ambos bridges usan el mismo coste de enlace, para eso:

- El bridge anuncia su coste de enlace configurado en las BPDUs.
- Todos los bridges de una LAN específica usan los costes anunciados por el puente designado del CSTI.
- El CSTI no sigue este esquema.

Así pues, como conclusión se requiere un parámetro de coste de enlace y además las prioridades de bridge deben ser las mismas en todos los STIs que han de ser simétricos.

Una vez planteados estos requerimientos, analicemos cuál es la clave del problema de las posibles asimetrías. Para eso tomamos la Figura 7. Si asumimos enlaces de mismo coste, una topología en forma de anillo con un número de nodos impar no tendría problemas de asimetría, mientras que si el número de nodos es par sí lo habría, dado que existen dos caminos iguales entre los extremos opuestos del anillo. Si añadimos diferentes costes a los enlaces, el problema no se soluciona puesto que seguirá sucediendo que mientras una topología con número de nodos impar no tenga problemas, la de número par sí los tendrá, o viceversa (según cómo sean los costes de los enlaces).

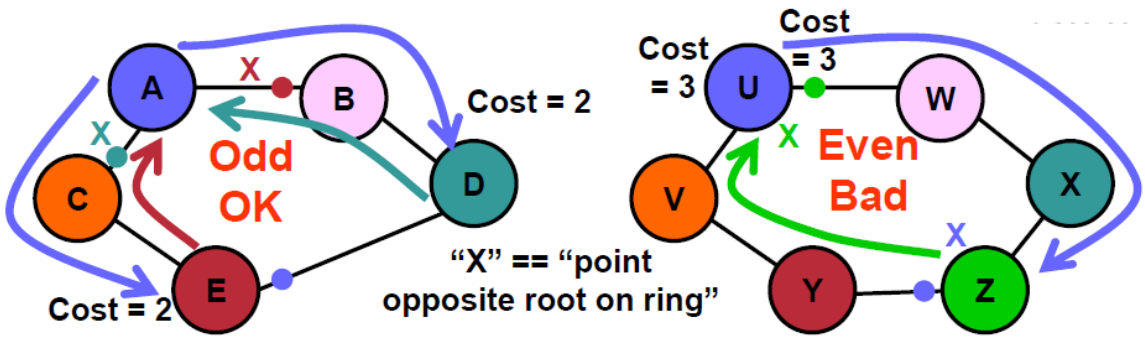


Figura 7. La clave del problema de posible asimetría al construir árboles expandidos

Puesto que este problema de asimetría siempre es visible desde ambos extremos de la comunicación, entonces bastaría con que, a la hora de tomar una decisión sobre qué árbol de expansión escoger en un caso de simetría, se escogiera siempre el mismo en ambos extremos. Normalmente esta decisión se hace de

manera individual, pero se sabe que cualquier otra decisión sería perfectamente válida y compatible con los algoritmos existentes de árbol de expansión, así que la idea es decidir de manera conjunta ahora y para eso se añade el Vector Camino (*Path Vector*).

El Vector Camino (Path Vector)

Si ambos extremos de la comunicación conocen el problema, deberían ser capaces de hacer algo al respecto. Por lo tanto, SPB añade en la BPDU, para cada instancia de árbol, un vector que contiene un bit de información sobre cada bridge en la red.

Para ver el funcionamiento de este vector, cojamos como ejemplo el de la Figura 8. Mirando dicha figura, se puede apreciar que existen dos caminos simétricos entre el bridge I y el A (uno que pasa por C y H, y otro que pasa por D y G) y que, al margen de eso e independientemente de dicha comunicación, los bridges B y E querrán bloquear uno de sus puertos para los diferentes árboles de I y A, y si B bloquea el de arriba y E el de abajo, habrán creado árboles asimétricos sin ser ni I, ni A, capaces de hacer nada al respecto. Sin embargo, en la práctica, cada bridge, como raíz de su propia MSTI, inicia un Vector Camino, *Path Vector* (PV), vacío. Cada bridge "posee" un bit diferente y según el PV se propaga a través de la red, cada bridge añade su propio bit al PV que se va generando con el resto de bridges por los que ha ido pasando. Así pues, en el caso en que el vector de la MSTI A alcanza el bridge B, éste tiene dos caminos de igual coste hacia el bridge raíz A y decidirá qué puerto será el raíz en base al vector, en vez de usar las IDs de puente (C y D), para lo que simplemente comparará los números binarios contenidos en los PVs recibidos (BCH y BDG en la figura) y decidirá en base a las diferencias (que son los bits CH y DG). De manera similar, al bridge E se le presentará la misma elección (dos números binarios CEH y DEG, que sólo difieren en los bits CH y DG). Si ambos bridges, B y E, realizan la misma decisión a partir de los vectores, entonces los puertos raíz coincidirán para las MSTIs I y A y los caminos serán simétricos (en la figura por ejemplo se escogen como raíz los puertos que vienen de los enlaces C y H, y se sigue construyendo el PV de manera común y reenviándose hacia I y A, quedando BCHE). Todo esto se consigue con una única propagación del vector a través de la red.

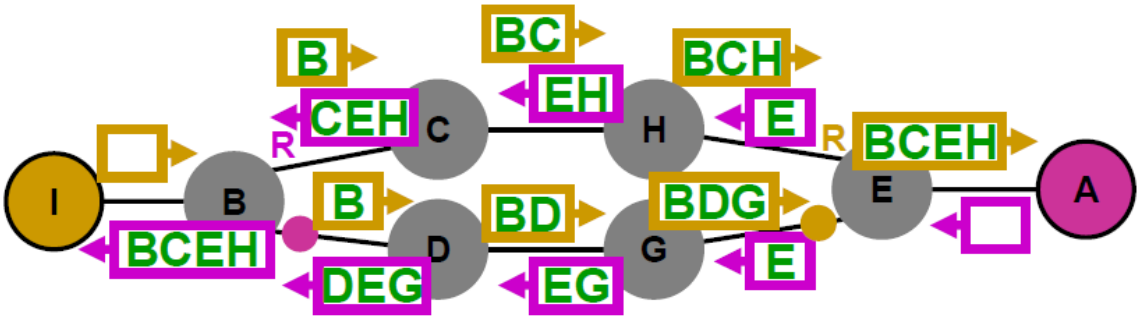


Figura 8. Propagación del Vector Camino (*Path Vector*)

En el caso del estándar IEEE 802.1Q-2003 (MSTP y VLANs, de donde proviene SPB y con los que se pretende guardar compatibilidad), el denominado puerto regional raíz (*Regional Root Port*) para cierta MSTI X es el puerto por el que se

recibe el mejor Vector Prioridad (*Priority Vector*) del puente designado de la MSTI X. Este vector posee los siguientes campos:

{Root ID, Root Path Cost, Bridge ID, Port ID}

donde *Root ID* es la identidad del bridge raíz, *Root Path Cost* es el coste hasta dicho puente raíz, y *Bridge ID* y *Port ID* proceden del bridge designado en la MSTI X. Para mantener la compatibilidad y, a la vez, incluír el Vector Camino que evita las posibles asimetrías, el Vector Prioridad se modifica cambiando el campo *Bridge ID* por el PV de la MSTI X, quedando:

{Root ID, Root Path Cost, Path Vector, Port ID}

El Vector Reflexión (Reflection Vector)

Como ya se conoce, a la hora de enviar una trama, la transmisión se realiza en base al árbol enraizado en el puente que origina dicha trama. Sin embargo, existe aún un problema en la transmisión multicast. Tomando como ejemplo la topología de la figura anterior, cómo sabe el bridge B sobre qué puertos distribuír cierta dirección de grupo multicast G en cierta MSTI A, conociendo que el bridge C quiere recibirla. Si B supiera si está dentro de la MSTI entre A y C (recordemos que son dos MSTIs, pero simétricas en la práctica), le bastaría para responder esta pregunta.

Si miramos el ejemplo de la Figura 9, el puente C no puede suponer, a partir de las dos entradas que recibe del PV, qué decisión tomarán los bridges B y E a la hora de escoger entre los dos caminos de igual coste entre I y A. Es decir, C no puede saber si estará o no dentro del camino entre los bridges I y A.

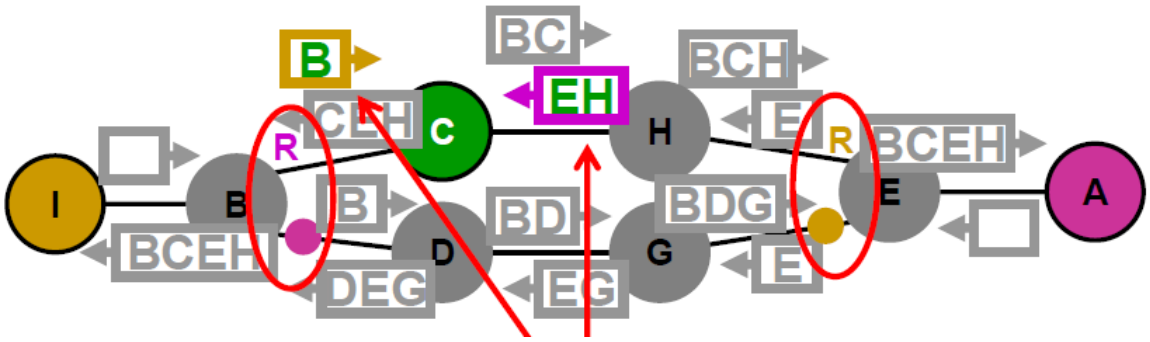


Figura 9. El problema de la transmisión multicast en SPB

Para solucionar este problema, se define el Vector Reflexión, *Reflection Vector* (RV). Cada puente raíz inicia un RV con un bit para cada una de las otras MSTI. Ese bit dice si el RV ha (R) o no ha (N) pasado sólo a través de puertos raíz de la instancia de árbol MSTI a la que hace referencia durante el camino del RV desde el puente raíz del RV. Gracias a este vector, ahora cualquier bridge puede ver si está en el camino entre dos bridges o no. Por ejemplo, en la Figura 10, tras el paso del RV cualquier bridge en la red podría determinar si está dentro del camino entre I y A. Es sencillo, si el bit del bridge A en el RV del bridge I es "R" y viceversa, entonces es posible afirmar que el bridge está en el camino, si es "N", entonces no. Además, debido a que el PV asegura que todas las MSTIs son simétricas, no es necesario tener ambos bits (en ambos sentidos) y con uno es suficiente para saber si el bridge

pertenece al camino o no. Eso deja el tamaño del RV en la mitad, lo que ayuda a que no aumente demasiado el tamaño de la BPDU.

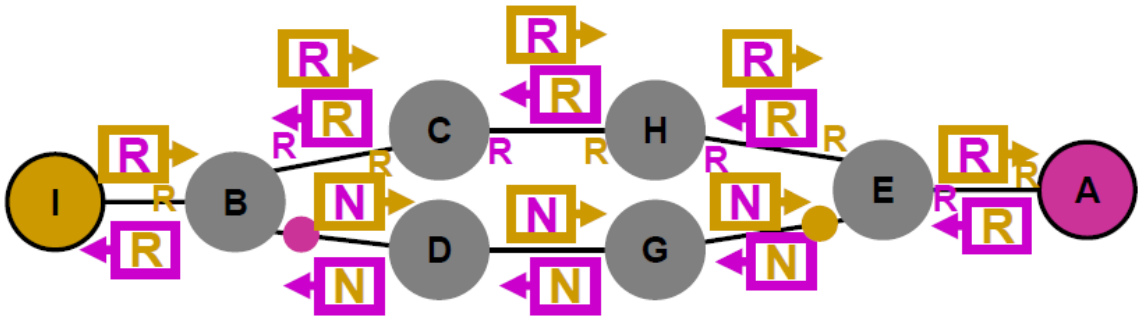


Figura 10. Propagación del Vector Reflexión (*Reflection Vector*)

Así pues el RV se genera de la siguiente forma. Cada bridge mantiene un RV para cada MSTI. El RV tiene un bit para cada otro bridge con ID de MSTI (MSTID) menor que el MSTID del puente que origina el vector. Un 1 indica "raíz en esta dirección" y un 0 indica "raíz no en esta dirección". El RV para cierta MSTI X estará todo a unos en el bridge raíz de dicha MSTI X.

Tiempos de convergencia en SPB

Tras los apartados anteriores, podemos observar que han aparecido una serie de modificaciones importantes respecto a MSTP, como son los vectores PV y RV. Así que se hace necesario conocer qué pasa con los tiempos de convergencia en SPB.

El peor caso de convergencia para MSTP es aquel en el que el puente raíz desaparece. Esto ya no puede suceder ahora, puesto que si un puente raíz desaparece, entonces su MSTI pasa a ser irrelevante. No sólo eso, sino que ahora el PV evita el problema de "la cuenta a infinito", común en algoritmos de vector distnacia como son MSTP y RIP (*Routing Information Protocol*). Si un mensaje MSTI recibido por un bridge X contiene un PV con el bit de dicho bridge X activado, se ignora, dado que el vector estará representando información antigua que está en bucle. Por consiguiente, el "diámetro de red" (*Max_Age*) no necesita ser configurado y elimina otra posible fuente de fallos de red: un "diámetro de red" configurado más pequeño de lo que es el tamaño real de la red.

Así pues, el PV converge normalmente en un único paso a través de la red y, ocasionalmente, en dos pasos (cuando información pasada hace bucle una vez). Mientras que el RV converge en, como mucho, un paso adicional más.

Etiquetado VLAN (VLAN Tagging)

El tag (etiqueta) VLAN de SPB puede ser dividido en tres partes (Figura 11):

- *Root Part* (R Part): que especifica qué bridge raíz se usa a la hora de encaminar la trama.
- *Multipath Part* (M Part): que especifica qué conjunto de parámetros de coste de enlace se usa a la hora de encaminar la trama.

- *Community Part* (C Part): que especifica el dominio broadcast (es decir, la subred IP).

Figura 11. Partes de la etiqueta VLAN en SPB

A continuación describiremos cómo funciona cada parte, tanto en solitario como de manera conjunta con las demás, para finalmente describir cómo se encuentran representadas físicamente en la práctica y algunos detalles puntuales sobre el plano de datos y qué sucede cuando algunas partes de la etiqueta faltan (por compatibilidad).

C Part

En el estándar IEEE 802.1Q de 1998, el estándar VLAN original, la etiqueta VLAN tiene esta única función, dado que había una única instancia de árbol expandido. Esta parte de la etiqueta identifica quién debería ver la trama. No se usa en el encaminamiento de tramas, excepto en el caso concreto de MAC duplicadas tal y como se establece en los protocolos HSRP y VRRP.

R Part y M Part

Juntas, seleccionan la instancia de árbol expandido que ha de ser usada a la hora de encaminar la trama. El número de instancias separadas lo define el número de partes R por el número de partes M.

M Part

Los árboles expandidos, como OSPF o IS-IS, se construyen minimizando la suma de "costes" de los enlaces por los que una trama podría pasar. Cada instancia de árbol con la misma parte M debe tener la misma estructura de coste de enlace. Las instancias de árbol con partes M diferentes, pueden tener diferentes costes de enlace. Esto permite especificar caminos alternativos a través de la red para diferentes flujos. La parte M etiqueta cada trama con el equivalente a un valor hash de canal Ethernet.

M Part y C Part

Juntas, determinan el ID Base de Filtrado de Datos, *Filtering Data Base ID* (FID), que ha de ser usado cuando se examina una dirección MAC. La tercera parte, la *Root Part*, nunca afecta la selección del FID. Diferentes partes C pueden, pero diferentes partes M no, estar mapeadas con el mismo FID (las VLANs privadas están mapeadas con el mismo FID).

R Part

Esta parte de la etiqueta selecciona qué bridge raíz de árbol expandido tiene que ser usado a la hora de encaminar la trama. Gracias al uso de esta parte, se puede conseguir encaminamiento perfecto tanto para tramas unicast como para multicast, evitando la queja común sobre los árboles expandidos.

Combinación V+M Part

El estándar IEEE 802.1s, *Multiple Spanning Trees* (MSTP), combina las partes M y C. Esta fusión de ambas partes es una típica queja contra el estándar 802.1s, MSTP sólo puede dividir los múltiples caminos de encaminamiento con el uso de límites de subred. Pero en todo caso, si este método de selección de camino se utiliza de manera satisfactoria, no se necesitan bits para la parte M mientras que se mantiene la compatibilidad con MSTP.

Representación Física

Si se dividen los 12 bits de la etiqueta VLAN, se mantiene la compatibilidad con el hardware, pero se deja poco espacio para la parte R.

Figura 12. Idea inicial de representación física de la etiqueta VLAN en SPB

Alternativamente, estas partes podrían situarse físicamente en una nueva, mucho mayor, etiqueta VLAN, tal y como se muestra en la figura siguiente. El estándar 802.1ah, *Provider Backbone Bridges* (PBB, también conocido como “MAC-in-MAC”) , puede manejar esto de manera adecuada.

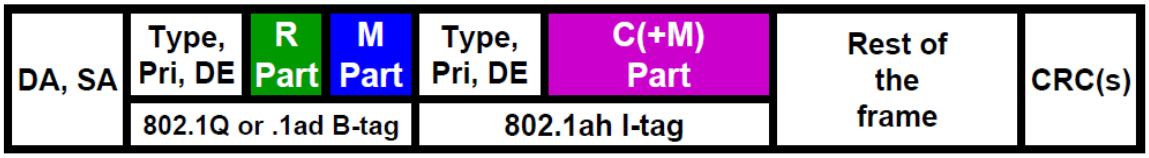


Figura 13. Representación física de la etiqueta VLAN en SPB basada en el estándar 802.1ah

Una última opción sería el mapeado de las diferentes partes, que permite mayor flexibilidad. Por ejemplo, si el bridge 6 nunca necesitará la parte C, entonces esa combinación de partes R y C no es necesario que existan. Esto puede aumentar el tamaño de la red a cambio de un coste proporcionado en flexibilidad.

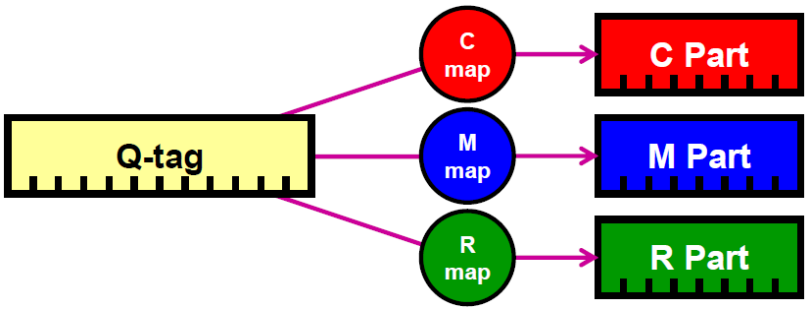


Figura 14. Representación física de la etiqueta VLAN mapeada en SPB

Detalles sobre el plano de datos

El aprendizaje de direcciones MAC y el reenvío de tramas de datos funciona exactamente igual que con el estándar 802.1Q. Las tramas que no tienen partes R y M (que especifica por qué MSTI se tiene que reenviar la trama) y se reciben en un puente, son asignadas a cierta MSTI (se les asigna parte R y M) por el puente que las recibe.

La parte R es constante para cada bridge. La parte M puede ser constante o puede ser una función de los contenidos de la trama, por ejemplo una función hash que incluya la dirección IP. Si la parte M se elige por función hash, esa función tiene que ser simétrica respecto a origen y destino.

La necesidad de Protocolos de Estado de Enlace (Link State Protocols)

Una vez conocido cómo se lleva a cabo la configuración automática y dado que no hay direcciones MAC prefijadas (es decir, jerárquicas), hay poco más que tenga que ser configurado. Sin embargo, queda una cosa pendiente, los bridges han de asignar las instancias de árbol, las MSTIs. Esto requiere un esfuerzo previo de configuración y complica el posible *plug-and-play* (el hecho de poder conectar algo y que se pueda usar directamente sin ningún ajuste extra).

Para quitar la necesidad de configuración de dichas instancias podría usarse información de estado de enlace para generar las MSTIs. El problema es que los bridges no son capaces de saber qué tienen conectado a su alrededor para anunciar así sus direcciones MAC, ni tampoco transmiten nunca múltiples copias de tramas multicast, ni tramas desordenadas, ni las tramas Ethernet tienen campo TTL (*Time To Live*) para evitar bucles temporales durante los cambios de topología. Para solucionar esto, se podrían adaptar los switches para que funcionasen de manera más parecida a un router, como es el enfoque de TRILL (protocolo del que hablaremos más adelante), pero esto no es lo que se busca. Así pues, lo que se hace en SPB es actualizar el software del bridge para calcular los múltiples árboles expandidos óptimos con el uso de información de estado de enlace, en vez de MSTP.

Es posible demostrar que la suma de todas las rutas hacia cierto puente A de una topología es idéntica al árbol expandido enraizado en A, véase Figura 15. Por lo tanto, si el bridge A aplica el algoritmo de Dijkstra con la información de estado de enlace, éste podría calcular el árbol expandido enraizado en A. Sin embargo, esto último es irrelevante, pues A ya conoce que es el raíz de su propio árbol y lo que

necesita conocer es cuál es su rol en el resto de árboles expandidos de los demás bridges. Para eso, se modifica ligeramente el algoritmo de manera que el cálculo quede simétrico, de tal forma que si hay múltiples caminos de igual coste, la decisión que se haga en todos los bridges sea la misma, entre otras cosas (como que debe aplicarse el algoritmo también para los vecinos), para así solucionar la configuración de la MSTI sólo con estado de enlace.

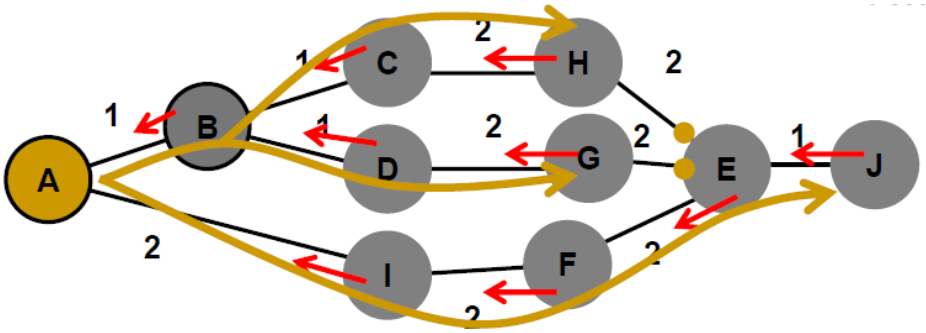


Figura 15. Equivalencia entre la información proporcionada por estado de enlace de rutas hacia A y el árbol enraizado en dicho bridge A

Queda eso sí, un último asunto que resolver: cómo solucionar los bucles temporales durante los cambios de topología, dado que las tramas Ethernet como sabemos no tienen campo TTL. Para evitar dichos posibles bucles se realizan bloqueos tal y como se hacen para MSTP y las tramas son descartadas hasta que se complete la actualización de la topología. Nótese que estos bloqueos sólo se utilizan cuando se necesitan para evitar bucles. En la práctica, estos mecanismos están basados en un acuerdo mutuo a nivel de cada enlace, cambiando de estado ambos puertos a la vez cuando se cumplen ciertas condiciones que eviten bucles, tal y como se hace en RSTP y MSTP.

Shortest Path Bridging-VID (SPBV) y Shortest Path Bridging-MAC (SPBM)

El estándar IEEE 802.1aq se puede considerar el plano de control Ethernet con estado de enlace para todas las VLANs del IEEE cubiertas en el estándar IEEE 802.1Q. *Shortest Path Bridging – VID* (SPBV) proporciona características que son compatibles hacia atrás con las tecnología de árbol expandido, mientras que *Shortest Path Bridging – MAC* (SPBM, conocido previamente como SPBB) proporciona valores adicionales para sacar provecho de las capacidades de *Provider Backbone Bridge* (PBB). SPB, el término genérico para ambos SPBV y SPBM, combina la construcción de un camino de comunicaciones Ethernet (ya sea siguiendo el IEEE 802.1Q en el caso de SPBV o PBB IEE 802.1ah en el caso de SPBM) con un protocolo de control de estado de enlace IS-IS corriendo entre los puentes SPB (en los enlaces de interconexión). El protocolo de estado de enlace se usa para descubrir y anunciar la topología, y para calcular los árboles de caminos mínimos entre todos los bridges de la región SPB.

En SPB, como con cualquier otro protocolo basado en estado de enlace, los cálculos necesarios se realizan de forma distribuida. Cada nodo computa el comportamiento de la red independientemente, de manera basada en una visión común normalmente sincronizada de la red (de escalas de alrededor de 1000 nodos

o menos). Las tablas de encaminamiento se generan localmente para implementar independientemente el comportamiento de reenvío de su propia porción de red.

En SPBM se distribuye tanto la dirección Backbone MAC (B-MAC) de los nodos participantes, como la información de "miembro" para las interfaces participantes y las que no. Con "participante", hablamos de las interfaces que están dentro de la región SPBM en la que la comunicación se realiza en base a la B-MAC. La información de topología es la base del mecanismo de cálculo que computa los árboles de caminos mínimos basados en coste mínimo desde cada nodo participante hacia el resto de nodos que también participan. Sin embargo, en SPBV, estos árboles proporcionan caminos mínimos donde las direcciones MAC individuales pueden aprenderse y la participación a cierta dirección de grupo multicast también puede ser distribuida. Las diferencias entre ambas versiones radican en que SPBM pretende el aislamiento de diversas instancias de LANs cliente, por lo que usa encapsulamiento completo, mientras que SPBV no pretende dicho aislamiento de las direcciones MAC de cliente.

A continuación se describen ambas versiones por separado, aunque la mayor parte de ambas es común y las diferencias se encuentran en el plano de datos.

Shortest Path Bridging – VID (SPBV)

Cronológicamente, esta versión es la que apareció primero. En SPBV, el mecanismo utilizado para identificar el árbol es el uso de diferentes identificadores VLAN (VID) para cada bridge origen. El uso de una topología IS-IS está destinado a situar las diferentes y únicas identidades SPBVs (*Shortest Path Bridging VID*), y habilitar el encaminamiento por caminos mínimos para direcciones individuales y de grupo. Originalmente estaba orientado a pequeñas redes, de poca necesidad de configuración, pero evolucionó con el tiempo a un proyecto mayor.

En SPB se definen diferentes regiones y se tienen en cuenta los bridges que son capaces de procesar SPB, incluyendo en la región los bridges SPB que tienen la misma VID base y MSTID. SPBV construye árboles de caminos mínimos y permite el aprendizaje de las direcciones MAC, pero puede distribuir direcciones multicast que pueden ser usadas para "recortar" el árbol de acuerdo con los nodos que son o no miembros de dicho grupo multicast, tanto mediante el uso de *Multicast MAC Registration Protocol* (MMRP) o directamente con la distribución multicast de IS-IS. SPBV construye dichos árboles, pero también trabaja con bridges *legacy* que siguen protocolos como RSTP o MSTP, para eso usa técnicas de las regiones MSTP para la interconexión con regiones que no siguen SPB, comportándose dichas regiones de manera externa como un bridge enorme. A pesar de que son varios árboles, SPBV también construye un árbol expandido, calculado con estado de enlace, que utiliza la VID base. Esto quiere decir que SPBV puede usar ese árbol tradicional para el cálculo del CIST, que es el árbol por defecto usado a la hora de interconectarse con otros bridges *legacy*, y además también sirve como un árbol expandido de apoyo en caso de que existan problemas de configuración con SPBV.

SPBV ha sido diseñado para manejar un número moderado de bridges. Se diferencia de SPBM en que las direcciones MAC se aprenden en todos los bridges que conforman el camino mínimo y se usa un aprendizaje de VLAN compartido,

puesto que direcciones MAC destino pueden ser asociadas a múltiples SPBVs. Nótese que SPBV aprende todas las MAC que se reenvían, incluso fuera de la región SPBV.

Shortest Path Bridging – MAC (SPBM)

La versión SPBM reutiliza el plano de datos de PBB, que no requiere que los *Backbone Core Bridges* (BCB), es decir, los bridges que conforman la parte interna o núcleo de la región PBB, aprendan las direcciones encapsuladas de los clientes, *Client-MAC* (C-MAC). Éstas, sólo necesitan ser aprendidas en la frontera de la red. A pesar de que SPBM es bastante similar a *Provider Link State Bridging (*PLSB), usando el mismo plano de control y datos, el formato y contenidos de los mensajes de control en PLSB no son compatibles.

El funcionamiento es el siguiente. Las tramas con MAC individual (es decir, tráfico unicast) que se reciben a través de un dispositivo Ethernet conectado a un bridge frontera de la red SPBM, son encapsuladas con una cabecera IEEE 802.1ah, PBB (MAC-in-MAC), y atraviesan la red IEEE 802.1aq, SPB, sin modificaciones hasta que llegan al bridge frontera del destino, donde se desencapsulan para recuperar su formato original. El aprendizaje de las direcciones destino Ethernet se realiza sobre la LAN lógica y son las *Backbone-MAC* (B-MAC) las que participan a la hora de realizar el encaminamiento hacia cierto destino. De esta forma, las MAC originales nuncan se buscan dentro del core de una red IEEE 802.1aq.

Comparando SPBM a PBB, el comportamiento es casi idéntico. PBB no especifica cómo se aprenden las direcciones B-MAC y PBB puede usar *Spanning Tree* para controlar las *Backbone-VLAN* (B-VLAN). Mientras que en SPBM la diferencia principal es que las direcciones B-MAC se distribuyen y computan en el plano de control, eliminando el aprendizaje de las mismas en PBB. Además SPBM garantiza que la ruta seguida al encaminar es un camino mínimo.

Recuperación de SPB ante fallo de enlace

La recuperación de SPB frente a un fallo (de enlace o de nodo) se realiza como en IS-IS se hace siempre, anunciado el enlace fallido y ejecutando los nuevos cálculos, determinando así nuevas tablas de reenvío (FDB tables). Se puede lograr una detección rápida de fallo de enlace usando los mensajes del IEEE 802.1ag, *Continuity Check Messages* (CCMs), que realizan un chequeo del estado del enlace e informan de fallos al protocolo IS-IS. Esto permite una detección de fallo mucho más rápida de la posible con el uso de los mecanismos de pérdidas de mensajes *Hello* en IS-IS.

Ambos, SPBV y SPBM, heredan la rápida convergencia de un plano de control basado en estado de enlace. Un atributo especial de SPBM es su capacidad de reconstruir árboles de comunicación multicast en un tiempo similar a la convergencia de unicast, porque sustituye cálculo por señalización. Debido a que el protocolo SPBM no anuncia, ni ha de conocer, direcciones Ethernet, no es necesario volver a aprender nada en los switches del núcleo de la red y las encapsulaciones aprendidas no se ven afectadas en nada por fallos de enlace o nodo.

Equal Cost Multi Tree (ECMT)

Inicialmente se definen dieciséis caminos ECMT, aunque podría haber más. En una red IEEE 802.1aq, ECMT es más predecible que IP o MPLS gracias a la simetría entre los caminos de ida y vuelta. Por lo tanto, la elección de qué camino ECMT utilizar se basará en cierta decisión asignada por un operador, mientras que en IP o MPLS es una decisión basada en datos locales o hashing.

A la hora de elegir entre dos caminos de igual coste de enlace, en un primer algoritmo de ECMT se sigue la lógica a continuación: primero, si un camino es más corto que otro en número de saltos, se escoge el más corto, si no, se elige el camino con el mínimo valor de identificador de puente, *Bridge Identifier* (que se compone del *Bridge Priority*, seguido del IS-IS SysID). Otros algoritmos de ECMT se crean usando permutaciones conocidas de los valores *Bridge Priority* y IS-IS SysID. Se definen catorce algoritmos ECMT más, con diferentes máscaras de bits aplicadas a los identificadores de puente. Dado que el identificador incluye un campo de prioridad, es posible ajustar el comportamiento de ECMT aumentando o disminuyendo el valor de *Bridge Priority*. Estos dieciséis algoritmos pueden ser extendidos y se espera que otros grupos de estandarización, o fabricantes, generen variantes de los algoritmos definidos actualmente, resultando en comportamientos de ECMT adaptados a diferentes estilos de red.

2.1.2 Bridges con encaminamiento

Se revisan en esta sección los diversos tipos de bridges que utilizan encaminamiento de diversas características. Se comienza por los bridges con encaminamiento en origen, continuando brevemente con las propuestas de Autonet y Smartbridge, para finalmente enfocarnos en la propuesta de TRILL.

2.1.2.1 Bridges con encaminamiento en origen

Los bridges con encaminamiento en origen (*source routing bridges*) presentaron desde su aparición una alternativa, basada en encaminamiento (aunque estandarizada como *bridging*), a los bridges transparentes [DP88] [PW87]. En el encaminamiento en origen, la ruta es establecida por el dispositivo origen e incluida en los paquetes enviados. Cada dispositivo intermedio observa su lugar en la ruta y encamina al siguiente dispositivo indicado en la ruta explícita. Los bridges con encaminamiento en origen más conocidos son los usados en redes de Paso de Testigo en Anillo (*Token Ring*), actualmente obsoletos.

Uno de los inconvenientes de este tipo de bridges es, a diferencia de la autoconfiguración de los puentes Ethernet, la necesidad de asignar la identidad de todos los puentes y redes locales. Dado que en el planteamiento del problema se ha establecido el requisito de no afectar a los sistemas finales, deberán ser siempre los bridges los que realicen el encaminamiento en origen, nunca los sistemas finales, para mantener la transparencia. El segundo gran inconveniente es la gran sobrecarga de mensajes producida por el descubrimiento de rutas. Aunque el encaminamiento en origen aparezca como obsoleto en redes fijas, es objeto de

estandarización en las redes ad-hoc como es el caso del protocolo *Dynamic Source Routing* (DSR) [DSR04].

2.1.2.2 Autonet

El antecedente más destacado como propuesta de evolución de las tecnologías LAN es *Autonet* [SHO+91], ya que constituye uno de los primeros antecedentes de red local autoconfigurable. *Autonet* utiliza todo el ancho de banda disponible en sus enlaces empleando encaminamiento arriba/abajo (up/down), detallado en el Apéndice A (Protocolos de Prohibición de Giros), soportado sobre un protocolo de árbol de distribución STP modificado. Las tramas *Autonet* se utilizan para encapsular las tramas Ethernet, y las direcciones empleadas son de 12 bits para simplificar el encaminamiento. La traducción de direcciones UID (direcciones MAC, 48 bits) a direcciones cortas Autonet (12 bits) se realiza mediante aprendizaje de las tramas recibidas. Las direcciones cortas de los hosts y puentes son asignadas por el bridge raíz, que asimismo distribuye la información de topología y de árbol de distribución, de forma piramidal hacia abajo. Finalmente, cada bridge calcula su tabla de encaminamiento con la información recibida.

Un inconveniente considerable de *Autonet* radica en que la compatibilidad entre los modos de trabajo Ethernet y *Autonet* se implementa en los sistemas finales, requiriendo su modificación mediante la incorporación del módulo *Localnet*, situado por encima de los manejadores SW (drivers Ethernet y *Autonet*). *Localnet* se requiere para poder aprender la correspondencia entre las parejas de direcciones de formatos EUI/*Autonet*. Los enlaces deben ser punto a punto, con un solo sistema final conectado en cada segmento, de LAN, lo que facilita la localización de los sistemas finales, una parte del problema de encaminamiento.

2.1.2.3 Smartbridge

Smartbridge [RTA00], al igual que *Autonet*, tiene como objetivo lograr bridges capaces de encaminar por caminos mínimos. El encaminamiento tiene dos partes: encaminar entre LANs y llegar a los sistemas finales. Encaminar entre LANs es relativamente fácil, pero es también necesario conocer a qué LAN está cada sistema final conectado, dado que los sistemas finales se pueden mover. *Smartbridge* utiliza el concepto de *source tree* (unión de las rutas que parten de una determinada LAN) para detectar los sistemas finales que se han movido y el de *destination tree* (unión de las rutas que llegan a una LAN) para implementar el reenvío (*forwarding*). Entre los requisitos de diseño de *Smartbridge*, es de notar que se especifica que la trama no debe ser alterada por el bridge. Por este motivo se justifica la no utilización de encaminamiento de estado de enlaces al no poder disponer de los mecanismos antibucle (como el campo TTL de los paquetes IP) necesarios mientras el protocolo converge.

Smartbridge, a diferencia de *Autonet*, admite conexiones con segmento compartido e incorpora, como *Autonet*, el concepto de *diffusing computation* [DS80]. Este mecanismo permite separar las partes de la red que están en reconfiguración del resto. En este mecanismo, el iniciador hace peticiones a sus vecinos, los cuales las realizan a los suyos y así sucesivamente. La respuesta se envía al peticionario

cuando el requerido ha contestado, por eso cuando el iniciador recibe la respuesta de todos sus vecinos conoce que se ha completado el proceso. Para asegurar la consistencia, los cálculos deben separarse, de forma que no se mezclen procesos con información antigua y nueva. Esta separación se denomina wavefront.

Los procesos básicos de *Smartbridge* son los de *elaboración de inventario* y *aprendizaje de la topología.* El primero consiste en la elección de puerto designado del segmento LAN encargado de enviar todas las tramas. El segundo *(topology acquisition)* es una *diffusing computation* que se propaga por todos los bridges recopilando un lista de todas las conexiones de bridge a segmento y distribuye esta lista (que es una descripción de la topología) a todos los bridges. Cada instancia de obtención de topología se distingue por la UID del bridge iniciador y una marca de tiempo. Existen varias instancias simultáneas de adquisición de topología que se propagan por los bridges compitiendo. La última que finaliza contiene la topología actual.

Smartbridge no es compatible con los bridges 802.1D, aunque puede utilizar hardware estándar de PCs y tarjetas de red implementándolo en software. Encamina por caminos mínimos y requiere mínima configuración. Entre los inconvenientes destacan el no incluir la velocidad del enlace como criterio de elección de caminos y el utilizar reinicialización global ante cambios de topología (*global reboot*). Parece deseable un mecanismo más local y de menor impacto para reconfigurar en caso de cambios de topología.

2.1.2.4 Transparent Interconnection of Lots of Links (TRILL) – Rbridges

Transparent Interconnection of Lots of Links (TRILL) es un estándar del *Internet Engineering Task Force* (IETF) que hace uso de las técnicas de encaminamiento de capa 3, para crear un conjunto grande de enlaces que se muestren frente a nodos IP como una única subred IP. Permite crear una nube de capa 2 de manera que los nodos se puedan mover dentro de ella sin cambiar sus direcciones IP, mientras que se utilizan todas las técnicas de encaminamiento de capa 3 que han ido evolucionando a lo largo de los años, incluyendo el uso de caminos mínimos y múltiples (*multipath*). Además, TRILL admite características de capa 2 como las VLANs, la habilidad de autoconfiguración (mientras que la configuración manual se puede realizar igualmente si se desea) y envío de tráfico multicast/broadcast sin necesidad de usar ningún protocolo adicional. El problema inicial y la definición de la utilidad de TRILL pueden encontrarse en la RFC 5556 [TP09].

Se podría decir que TRILL es una evolución en el sentido de que un despliegue Ethernet, donde los enlaces están conectados a través del uso de bridges, puede convertirse en una nube TRILL reemplazando cualquier subconjunto de los bridges con dispositivos que implementen TRILL. Estos dispositivos se conocen como *Routing Bridges* (RBridges) y se definen detalladamente en la RFC 6325 [Per+11]. Cuando los bridges son reemplazados, nada cambia para los nodos IP conectados a la nube, salvo que la nube quizás se convierte en una más estable y que aprovecha mejor el ancho de banda disponible [PE11].

TRILL, como podrá verse en este apartado, es un protocolo de capa 2 y ½: Reune los enlaces de manera que los nodos IP observen la nube que forman como un único enlace. Por lo que TRILL está por debajo de la capa 3, pero por encima de la capa 2 dado que deja de lado el concepto de las nubes Ethernet tradicionales, tal y como hacen los routers IP.

En la siguiente figura se muestran los conceptos básicos de TRILL a la hora de manejar un paquete unicast donde la localización del destino es conocida:

- Los RBridges ejecutan un protocolo de estado de enlace, que les proporciona a cada uno información sobre la topología que consiste en todos los RBridges y todos los enlaces entre RBridges en la misma. Usando este protocolo, cada RBridge calcula caminos mínimos desde él mismo hasta cada uno de los demás RBridges, así como árboles para el reparto de tráfico multidestino.

- Cuando un RBridge R1 recibe una trama Ethernet desde un nodo final S, con destino a la dirección Ethernet D, R1 encapsula la trama con una cabecera TRILL, haciendo que el paquete se dirija hacia el RBridge R2, que tiene a D directamente conectado. La cabecera TRILL contiene un campo para el "RBridge de entrada" ("*ingress RBridge*") R1, otro para el "RBridge de salida" ("*egress RBridge*") R2, y un campo contador de saltos (*hop count*).

- Cuando R2 recibe el paquete encapsulado, retira la cabecera TRILL y reenvía la trama Ethernet a D.

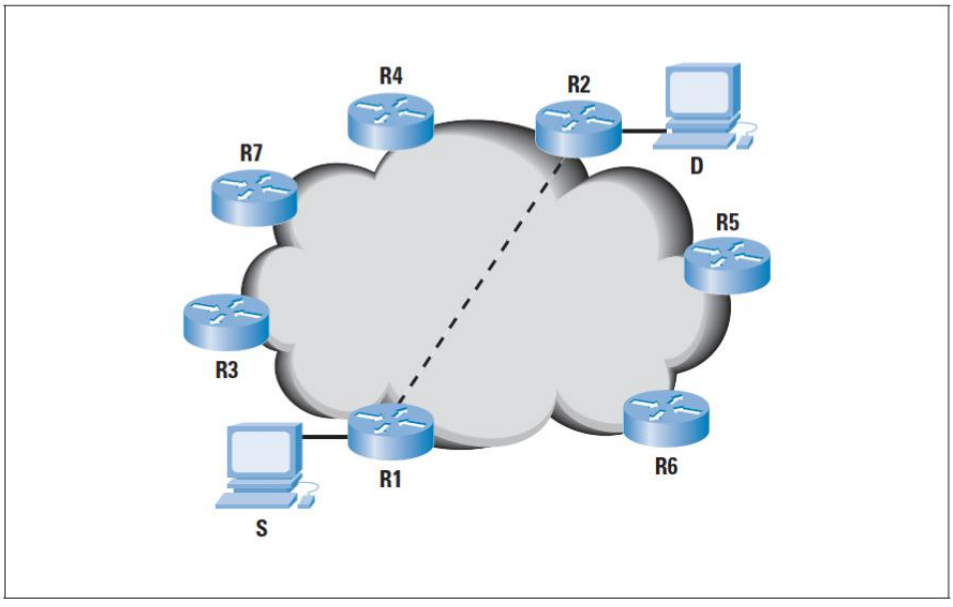


Figura 16. RBridging: El concepto de encaminamiento en las nubes formadas por RBridges

A continuación se describirá cómo es la cabecera TRILL, cómo R1 sabe que R2 es el "RBridge de salida" correcto y algunos de los conceptos del protocolo de estado

de enlace *Intermediate System-to-Intermediate System* (IS-IS). También se explicará cómo TRILL maneja las tramas multidestino, las VLANs, y el IP multicast.

La cabecera TRILL

Los campos principales de la cabecera TRILL son: *ingress RBridge nickname* (16 bits), *egress RBridge nickname* (16 bits), *hop count* (6 bits), y un flag para indicar si es multidestino (1 bit). Una cabecera típica de capa 3 tendría una fuente, un destino y un contador de saltos. Por lo tanto TRILL es básicamente una cabecera de encapsulamiento con direcciones planas de 16 bits. El hecho de cómo los RBridges obtienen los "*nicknames*" se describe más adelante. La cabecera completa se puede ver en la Figura 17, donde los campos son:

- TRILL Ethertype = Indica el tipo de *payload* (la parte de datos que envuelve la cabecera)
- V = Versión (2 bits)
- R = Reservado (2 bits)
- M = Multidestino (1 bit)
- OpLng = Longitud de las opciones de TRILL
- Hop = Contador de saltos, TTL
- Nicknames = Valores para designar el RBridge de entrada y de salida (16 bits)

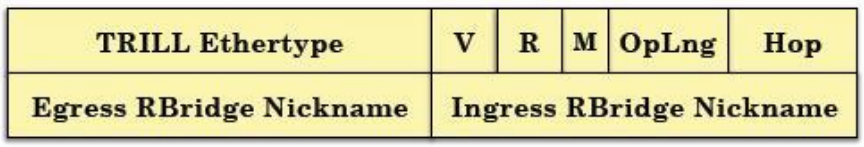


Figura 17. Formato de la cabecera TRILL

Esta cabecera es muy sencilla de reenviar para los core RBridges (es decir, los RBridges que forman el núcleo de una nube TRILL), en comparación con una cabecera IP o Ethernet. El campo del destino son sólo 16 bits, por lo que basta una simple búsqueda en una tabla para encontrar la entrada del puerto de salida, no como en el caso de destinos de 6 bytes de Ethernet por ejemplo, que normalmente requiere memoria que se pueda acceder de forma especial o mediante hashing, para que la búsqueda no sea excesivamente larga y costosa.

Tras el encapsulamiento de TRILL, la trama se codifica como una trama normal Ethernet con un valor de la dirección destino (*Destination Address*, DA) igual al siguiente RBridge de tránsito/salida y de dirección origen (*Source Address*, SA) igual a la MAC address del que mandó la trama, véase la Figura 18. La etiqueta VLAN exterior no es obligatoria y depende de la configuración del puerto. En el caso de que haya puentes que no sigan el protocolo TRILL en la topología, estos no se ven

afectados dado que el paquete TRILL es completamente transparente y podrán tratarlo como cualquier otro.

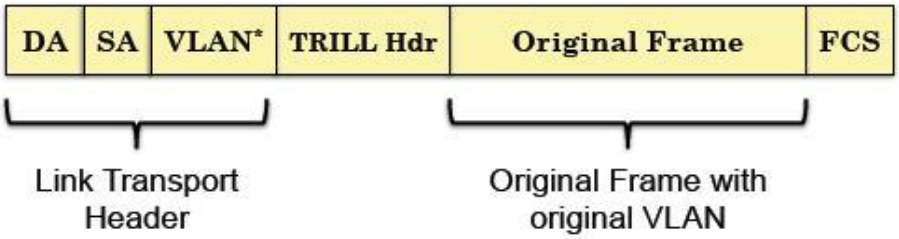


Figura 18. Formato del paquete TRILL

Aprendizaje de las localizaciones de los nodos destino

A estas alturas es possible que nos hayamos realizado una pregunta: cómo sabe R1 (siguiendo con el ejemplo de la Figura 16) que R2 es el RBridge de salida correcto para cierto destino D. El mecanismo por defecto es aprender la correspondencia entre la tupla (RBridge de entrada, dirección MAC origen) cuando el RBridge de salida desencapsula un paquete. Si R1 no conoce dónde se sitúa la MAC destino, R1 encapsula el paquete con una cabecera TRILL y el flag multidestino activo, indicando que deberá ser reenviado a todos los RBridges a través de un árbol.

Un mecanismo adicional, opcional, se conoce como *End-Station Address Distribution Information* (ESADI). ESADI permite a R1 anunciar parte o todos los nodos finales que están conectados directamente a R1. Tanto anunciar, como escuchar mensajes ESADI es algo opcional. Este mecanismo tiene ventajas sobre inundar la red y aprender de los paquetes de datos:

- Los paquetes ESADI pueden estar protegidos con criptografía.
- Quizás R1 tiene una razón más segura de saber que S está conectada a él que simplemente viendo un paquete con la dirección S en la cabecera. Por ejemplo, puede ser que R1 haya sido configurado para bloquear uno de sus puertos con la dirección MAC S. O quizás exista cierto protocolo de inscripción protegida cuando S se conecta a R1.
- Es posible que R1 sea capaz de verificar la ubicación de nodos finales locales de manera más precisa; por ejemplo, si son nodos IP, R1 podría ser capaz de hacerles *ping*.

También es posible disponer de un directorio que no sólo liste entradas del tipo (*RBridge nickname*, {conjunto de direcciones MAC de nodos finales conectados}), sino también las parejas {(direcciones IP de nodos finales, direcciones MAC de nodos finales)}. El primer RBridge, o incluso el proceso en sí del nodo final, podría lanzar peticiones al directorio con respecto al destino, y encapsular paquetes en lugar de inundar la red, y por lo tanto sería capaz de eludir el uso del protocolo de

resolución de direcciones de IPv4, *Address Resolution Protocol* (ARP), y de IPv6, *Neighbor Discovery Protocol* (NDP).

Los protocolos de estado de enlace en TRILL

Los protocolos de estado de enlace que habitualmente se despliegan en TRILL son *Intermediate System-to-Intermediate System* (IS-IS) [Ora90] (RFC 1142) y *Open Shortest Path First* (OSPF) [Moy98]. IS-IS, diseñado en los años 80 para encaminar DECnet, fue adoptado por la Organización Internacional de Normalización, *International Organization for Standardization* (ISO), puede encaminar tráfico IP y se usa por muchos proveedores de servicios de Internet, *Internet Service Provider* (ISP), para encaminar IP. IS-IS fue una elección natural para TRILL, dado que por su codificación permite fácilmente el uso de campos adicionales y además funciona directamente sobre capa 2, de manera que se autoconfigura, mientras que OSPF funciona por encima de IP y requiere que todos los routers tengan direcciones IP.

La Figura 19 muestra una pequeña red, en la parte superior, que consiste en 7 routers. En la mitad inferior de la misma, se muestra la base de datos de los paquetes de estado de enlace LSP (*Link State Packet*); todos los routers tienen la misma base de datos porque todos ellos reciben y almacenan los LSP generados más recientemente por cada uno de los otros routers. Dicha base de datos proporciona toda la información necesaria para calcular los caminos. También da información suficiente para que todos los routers calculen el mismo árbol, sin necesidad de utilizar un algoritmo de árbol expandido aparte. Como se puede ver, TRILL requiere un árbol (al menos uno) para distribuir los paquetes multidestino.

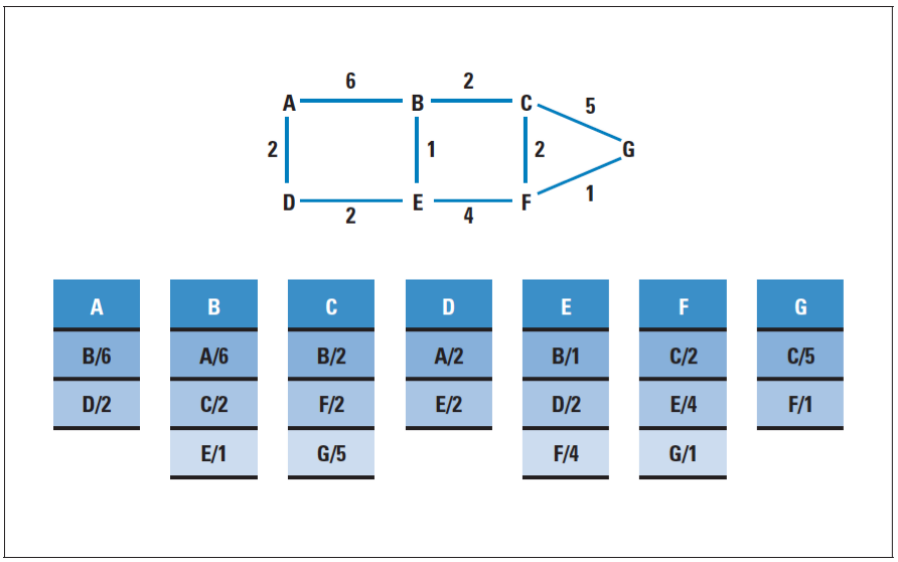


Figura 19. Red de routers y estado de enlace

Asignación de "nicknames" en TRILL

Dado que los paquetes de estado de enlace generados por cada RBridge son enviados a, y almacenados por, cada otro RBridge en la red, es posible difundir otra información con dichos paquetes, como un protocolo para asignar identidades, *nicknames*, únicas. Cada RBridge elige un valor de identidad al azar, evitando

escoger aquellos ya adquiridos por otros RBridges (conocidos al examinar la base de datos LSP). Si dos RBridges escogen los mismos valores, se desempata en base a un sistema de prioridad configurado y la ID del sistema de 6 bytes, manteniendo uno de los RBridges el valor y escogiendo el otro RBridge un valor que no aparezca como en uso.

Se da la posibilidad de poder configurar los valores de los *nicknames*, en cuyo caso un valor configurado tiene prioridad sobre cualquier otro que fuese escogido al azar. Y en el caso de una mala configuración, en la que dos RBridges sean asignados los mismos valores, se volvería a aplicar el sistema de prioridad anteriormente mencionado.

Los valores 0x0000 y aquellos entre 0xFFC0 y 0xFFFF están reservados y no deben usarse por los RBridges. Estos valores van en campos *Type-Length-Value* (TLV) de IS-IS junto con el campo de prioridad, que es un valor de 8 bits en el que el bit más significativo (0x80) indica que ha sido configurado y 7 bits restantes tienen un valor por defecto de 0x40. Normalmente cada RBridge tiene un único *nickname*, pero se pueden configurar múltiples. Por último, cada árbol multidestino posee su propio *nickname* también.

Interconexión de RBridges con otro tipo de bridges

TRILL está diseñado de manera que cualquier subconjunto de bridges en una red pueden ser sustituidos por RBridges. Un conjunto de enlaces conectados por puentes serán percibidos por RBridges como un único enlace compartido que conecta a todos los RBridge unidos a dicho enlace. Los puentes dentro de ese enlace se comportarán como puentes normales, formando un árbol expandido y reenviando paquetes a través de dicho árbol. La Figura 20 ilustra esta situación, en la que solo RB1, RB2, RB3 son RBridges en la topología Ethernet, el resto son puentes normales, y la línea de puntos se trata de un enlace que se ha seleccionado como *backup* por el árbol expandido.

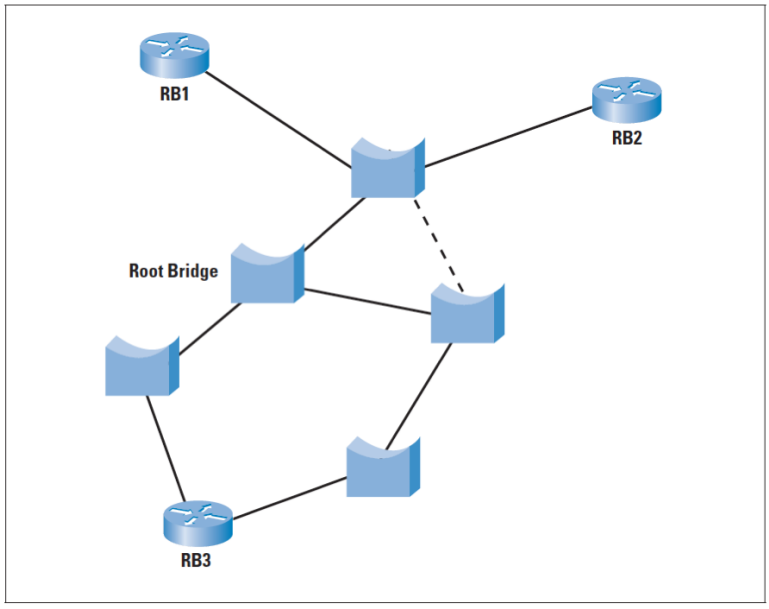


Figura 20. RBridges conectados por una LAN de bridges

Los RBridges RB1, RB2 y RB3 perciben el enlace como en la siguiente figura, un enlace único compartido en el que RB3 tiene dos puertos conectados. Como consecuencia, introducir RBridges en una red Ethernet particiona los árboles expandidos en árboles expandidos más pequeños.

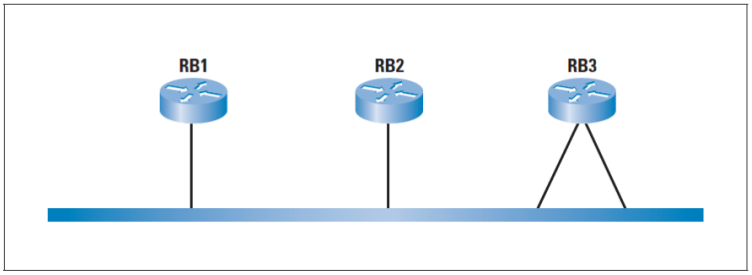


Figura 21. Figura 20 tal y como la perciben los RBridges: un único enlace compartido en el que RB3 tiene dos puertos conectados

Tipos de enlaces y la cabecera salto-a-salto (hop-by-hop)

Además de la cabecera TRILL, cuando un RBridge R1 está enviando una trama encapsulada con TRILL a un vecino RBRidge R2, hay una cabecera adicional que es específica al tipo de enlace que conecta R1 y R2. Aunque TRILL va por encima de Ethernet, un enlace entre dos RBridges o más podría ser cualquier tipo de enlace; por ejemplo, aparte de Ethernet, podría ser un enlace del *Point-to-Point Protocol* (PPP) [Car+11], un túnel IP o *IP Security* (IPSec), un camino *Multiprotocol Label Switching (*MPLS), etc.

Si el enlace es Ethernet, la cabecera "externa" será una cabecera Ethernet. Si es un enlace PPP, será una cabecera PPP. La cabecera "externa" para Ethernet (en el caso de un enlace Ethernet) tiene dos propósitos:

- Si hay puentes en el enlace, percibirán el paquete como un paquete Ethernet normal. Las tablas de aprendizaje de los puentes en dicho enlace verán sólo las direcciones de los RBridges en el mismo.

- Permite a R1, cuando está enviando tráfico a un enlace con múltiples vecinos (digamos R2 y R3), especificar qué RBridge es el próximo para el envío, R2 o R3, mandando el paquete en unicast al RBridge del siguiente salto, *next-hop RBridge*, elegido. Por ejemplo, podría ser que tanto R2 como R3 tuvieran el mismo coste hacia el destino, por lo que R1 debería especificar a cuál reenvía el paquete; de otro modo, ambos reenviarían el paquete y éste quedaría duplicado.

Por lo tanto, tal y como se ilustra en la Figura 22, un paquete con encapsulado TRILL puede tener tres cabeceras:

- La cabecera exterior o cabecera *hop-by-hop*, que se renueva en cada salto, es específica al tipo de enlace que conecta a los RBridges vecinos y, cuando se reenvía entre R1 y R2, especifica R1 como origen y R2 como destino.

- La cabecera TRILL, que de manera similar a una cabecera de capa 3 se mantiene intacta en su viaje desde el primer RBridge hasta el último, que especifica el primer RBridge (aquel que encapsuló el paquete con la cabecera TRILL) como el RBridge de entrada, y el último RBridge (aquel que desencapsulará el paquete) como el RBridge de salida.

- La cabecera interna Ethernet, que especifica la pareja de nodos que se están comunicando como origen y destino.

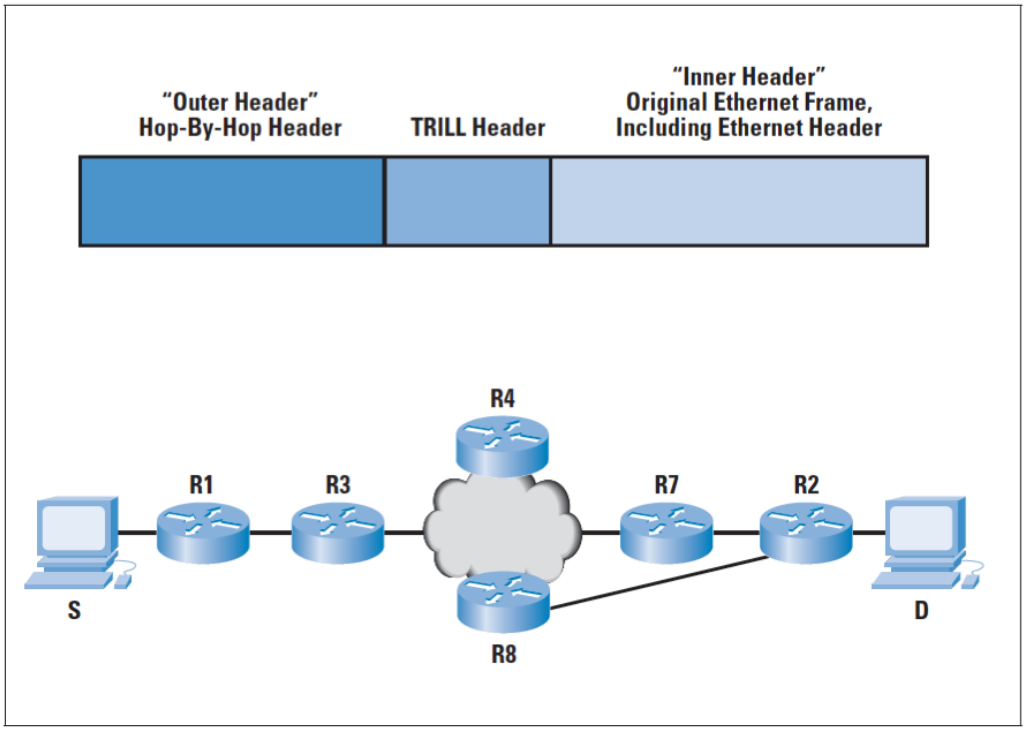


Figura 22. Cabeceras de los paquetes TRILL

Prestando atención al caso concreto de la Figura 22, asúmase que S transmite un paquete Ethernet a D. La cabecera interna Ethernet tendrá origen = S y destino = D. Al llegar a R1, este RBridge la encapsulará con una cabecera TRILL, donde el RBridge de entrada = R1 y el RBridge de salida = R2. Finalmente la reenviará hacia R3, no sin antes añadir la cabecera apropiada para el enlace. Si el enlace de Ethernet, la cabecera externa indicará origen = R1 y destino = R3. Cuando R3 la reciba y reenvíe hacia R7, R3 mantendrá la cabecera TRILL tal y como está (lo único que variará será el contador de saltos, que disminuirá), pero quitará la cabecera externa y pondrá una nueva indicando origen = R3 y destino = R7. De manera similar, R7 reenviará la trama a R2. Si es un enlace PPP, no se indicará ni origen ni destino. Cuando R2 manda la trama a D, R2 quita primero la cabecera TRILL y D verá el paquete exactamente igual que S lo transmitió.

Implicaciones de VLAN en TRILL

El concepto de LAN Virtual, *Virtual LAN* (VLAN), en Ethernet genera conjuntos de nodos finales que comparten la misma infraestructura (enlaces y puentes), de manera que los nodos finales en el mismo conjunto puedan hablar entre ellos directamente (usando Ethernet), mientras que aquellos en VLANs diferentes han de

comunicarse a través de un router. Los nodos IP, aunque en general no son conscientes de las etiquetas VLAN, perciben las diferentes VLANs como diferentes subredes IP.

Para ver la utilidad de VLAN en TRILL, sigamos el ejemplo de la siguiente figura. Si hay múltiples RBridges en un mismo enlace, junto con nodos finales, es importante que sólo uno de ellos encapsule un paquete de cierto nodo final. Como se muestra en la Figura 23, si tanto R1 como R2 encapsularan un paquete unicast de S, se mandarían dos copias al destino. Sin embargo, si S estuviera mandando un paquete multidestino (ya sea multicast o con destino desconocido), entonces la copia que encapsulara R1 sería reenviada y recibida por R2, que la desencapsularía, y volvería a ser recibida por R1, que la encapsularía de nuevo y enviaría de nuevo a la red, creando un bucle. Este bucle además no se soluciona con el contador de saltos de TRILL, pues la cabecera se elimina y añade cada vez, ignorándose dicho campo. De este ejemplo en la figura, se ve que es importante que todos los RBridges conectados a un mismo enlace sepan sobre los otros, si no, puede haber problemas.

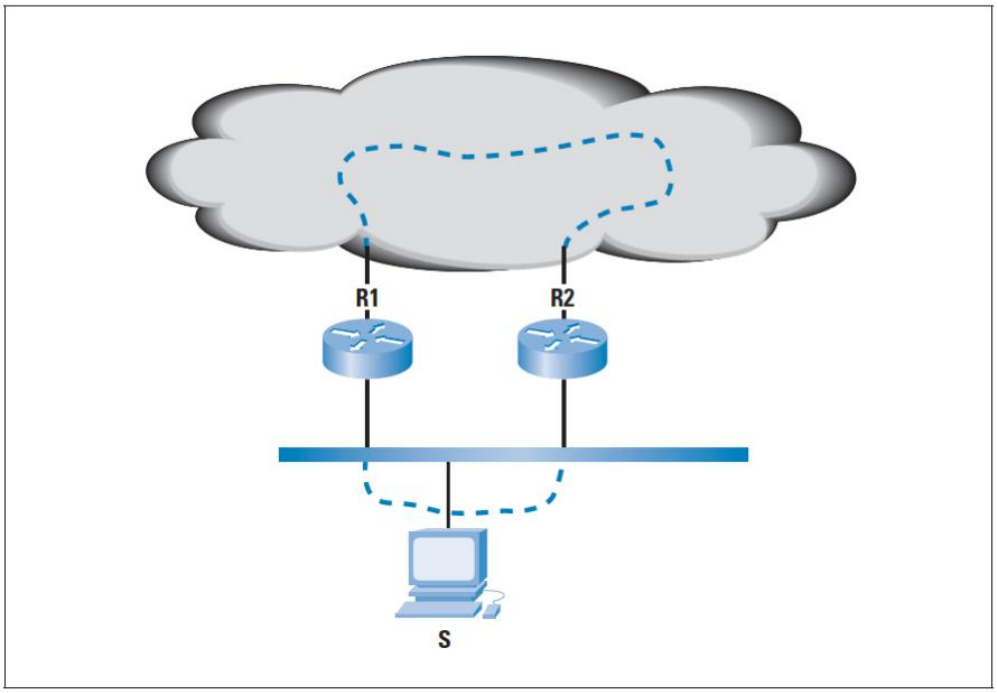


Figura 23. Enlace con múltiples RBridges conectados

Para solucionar los posibles problemas, IS-IS posee un protocol de elección en el que uno de los RBridges se elige como el RBridge Designado, *Designated RBridge* (DRB). Para repartir la tarea de encapsular y desencapsular tráfico, el DRB puede delegar el trabajo basándose en las VLAN. En otras palabras, si R1 es DRB, R1 puede dejar a R2 la tarea de encapsular/desencapsular tráfico para un conjunto de VLANs, sean VLANs x, y, z, y dejar a R3 otro grupo, mientras que R1 maneja el resto.

La elección se hace a través de mensajes *Hello*, donde los RBridges se anuncian a sí mismos. Desafortunadamente, cierta configuración de los puentes, ya sea intencional o por error, puede ser que haya dividido el enlace para tráfico marcado con la VLAN y, pero compartirlo para aquel marcado con la VLAN x. Esta situación complica la elección. Por ejemplo, al transmitir un mensaje *Hello* al enlace, el

RBridge R1 debe asignarlo a una VLAN y si escoge la VLAN y, su vecino R2 no vería el mensaje y por ello, ignorando que hay múltiples RBridges en el enlace, tanto R1 como R2 encapsularían un paquete proveniente de la VLAN x, surgiendo de nuevo el problema inicial.

TRILL maneja la anterior situación haciendo que el DRB (por defecto) transmita los mensajes *Hello* en todas las VLANs que tenga permitidas en su configuración de puerto. El DRB elige una VLAN, digamos VLAN A, para la comunicación entre RBridges en el enlace, e informa a los otros RBridges del enlace que deben usar dicha VLAN A. Los otros RBridges transmiten mensajes IS-IS (incluyendo los *Hello* y LSPs) y paquetes encapsulados TRILL, poniendo la VLAN A en la cabecera externa. La etiqueta VLAN en la cabecera interna es la que representa a la comunidad a la que el nodo final pertenece, mientras que la etiqueta VLAN en la cabecera externa sólo tiene el propósito de atravesar saltos Ethernet entre RBridges. Si se conoce que no hay otro tipo de bridges, los RBridges (incluyendo el DRB) pueden ser configurados para que envíen mensajes *Hello* sólo en la VLAN especificada por el DRB.

Protocolo Hello modificado

Como ya es conocido del apartado anterior, IS-IS posee un protocol de elección en el que los routers (o RBridges en el caso de TRILL) envían mensajes *Hello*. Estos mensajes transmitidos por R1 no sólo anuncian R1 a sus vecinos, sino que también contienen una lista de vecinos de los que R1 ya ha recibido mensajes *Hello*. R2 no considerará a R1 como vecino a no ser que R2 se vea a sí mismo listado en los mensajes *Hello* de R1, indicando que la conectividad es en los dos sentidos. A la hora de elegir el DRB, R2 ignorará cualquier router con el que no tenga conectividad en ambos sentidos. Por lo tanto, si hay un enlace con propiedades especiales de conectividad, es posible que los routers del mismo enlace se agrupen en distintos conjuntos, cada uno con su propio DRB y presentando un enlace diferente al resto.

Un aspecto sorprendente del uso de IS-IS en TRILL fue que el protocolo *Hello* tuvo que ser modificado ligeramente. En IS-IS para capa 3, los mensajes *Hello* se ajustan al tamaño máximo, porque un posible modo de fallo hardware es que un enlace entre R1 y R2 puede ser capaz de enviar paquetes pequeños, pero no grandes, y en capa 3 se decidió asumir que era mejor que R1 y R2 no se vieran como vecinos en vez de utilizar un enlace que no se usa en toda su capacidad. En IS-IS, los paquetes LSP pueden ser fragmentados sólo por el origen R1. Todos los routers acuerdan el tamaño máximo de fragmento LSP que se garantiza que podrá pasar a través de todos los enlaces y los enlaces que no pueden reenviar dicho tamaño no se tienen en cuenta en la topología y, de hecho, en IS-IS capa 3 ni siquiera serían descubiertos porque el mensaje *Hello* no sería visto por el router vecino.

En TRILL, es importante que sólo un único RBridge sea elegido como DRB, porque es el DRB el que determina qué RBridge encapsulará/desencapsulará paquetes para qué VLAN. Para asegurar esto, TRILL modificó el protocolo *Hello*. Este diseño, además de solucionar el problemas de los múltiples DRB, permite a TRILL descubrir qué enlaces pueden manejar paquetes jumbo (*jumbograms*). A continuación se describen los cambios:

- Limitar el tamaño de los mensajes *Hello* y no ajustarlos a ningún máximo (para evitar impedimentos artificiales a la hora de conocer a los vecinos).

- Elegir un DRB basándose exclusivamente en prioridad (no en la conectividad en ambos sentidos como en IS-IS de capa 3). En otras palabras, que cierto RBridge R2 permita a otro de alta prioridad R1 ser DRB incluso si no consta en su lista de vecinos.

- Tener un mecanismo separado para testear, usando paquetes de diferentes tamaños, para ver qué tamaño de paquetes pueden ser enviados por el enlace.

Para evitar la confusión causada por este cambio en el protocolo *Hello* que tan bien había funcionado durante décadas para capa 3, se escribió una RFC adicional [Eas+11] para explicar específicamente el mecanismo de *Hello* en TRILL.

Tramas multidestino

Árboles múltiples

En el diseño original de TRILL, los RBridges calculaban un árbol único compartido, basado en la base de datos LSP, y todo el tráfico multidestino era encaminado a través de dicho árbol. Sin embargo, para ser capaz de repartir la carga por los distintos enlaces a la hora de transmitir tráfico multidestino, en la fase temprana del estándar TRILL, se añadió la opción de usar múltiples árboles.

En TRILL, el RBridge con la mayor prioridad de ser raíz del árbol anuncia al resto de RBridges (a través de su LSP) cuántos árboles, y qué árboles, deberían ser calculados. Un árbol es calculado como árbol de caminos mínimos desde un raíz dado, con un algoritmo de prioridades determinado para que todos los RBridges calculen el mismo árbol. El puente raíz puede ser un RBrige o un pseudonodo. En algunos casos, un raíz puede estar particularmente tan bien situado en la topología que su árbol forma caminos bastante buenos para todos los pares de nodos, pero es deseable tener árboles múltiples diferentes calculados desde el mismo raíz. TRILL consigue esta configuración haciendo que el raíz adquiera múltiples *nicknames*, uno por cada árbol, y usando el número del árbol en el algoritmo de desempate, de forma que aunque todos los árboles desde ese raíz serán todavía árboles de caminos mínimos, en los diferentes árboles se elegirán diferentes enlaces.

Cuando R1 encapsula una trama multidestino, R1 pone a uno el flag que indica multidestino y especifica el *nickname* del raíz en el campo del RBridge de salida de la cabecera TRILL.

Filtrado

Una trama multidestino será etiquetada con una VLAN (en la cabecera interna). La trama no necesita ser repartida a todos los RBridges, sino sólo a aquellos que tengan puertos conectados a nodos finales en dicha VLAN. Por lo tanto, los RBridges anuncian, en sus LSPs, a qué VLAN están “conectados”.

Aparte, TRILL permite filtrado basado en direcciones MAC de capa 2 derivadas de grupos IP Multicast. Los RBridges anuncian el conjunto de las direcciones MAC que desean recibir. El primer RBridge que acepta una mensaje de control de IP Multicast, tal como *Internet Group Management Protocol* (IGMP), lo examina y aprende qué cliente "escuchando" multicast o router multicast está conectado. Este examen permite a R1 informar en su LSP qué grupos IP Multicast desea recibir (o todos los grupos si un router multicast está conectado).

Otra parte concreta del envío multidestino es la comprobación del envío por el camino inverso, *Reverse Path Forwarding* (RPF). Para evitar bucles, cuando R está calculando qué subconjunto de sus puertos pertenecen a un árbol en particular, R también calcula, para cada puerto, el conjunto de RBridges de entrada cuyo tráfico en dicho árbol debería llegar por ese puerto.

Así pues, el procesamiento de las tramas recibidas por R, con cabecera TRILL indicando entrada = R1 y salida/raíz del árbol = R2, es el siguiente:

- Si el puerto por el que R recibe el paquete no está incluído en el árbol R2, se descarta.
- Si el puerto por el que R recibe el paquete está en el árbol R2, pero R1 no está listado en la información RPF para dicho puerto en el árbol R2, se descarta.
- Para cada otro puerto en R2, si se puede alcanzar la VLAN especificada a través de dicho puerto y la dirección IP Multicast se ha solicitado por un RBridge en el camino que atraviesa dicho puerto, transmitir el paquete por dicho puerto.

2.2 Encaminamiento en centros de datos

Se denonima *data center* o *centro de procesamiento de datos* (CPD) a aquella ubicación donde se concentran los recursos necesarios para el procesamiento de la información de una organización. Dichos recursos consisten en unas instalaciones debidamente acondicionadas, ordenadores y redes de comunicaciones, siendo éste último el punto que nos atañe en el actual documento. Otros términos posibles pueden ser *centro de cómputo*, *centro de cálculo* o *centro de datos* (de la traducción directa del inglés data center), que es el que utilizaremos principalmente. Los CPD actuales pueden llegar a contener miles de ordenadores con requerimientos muy elevados de ancho de banda. La arquitectura de la red normalmente estará compuesta por un conjunto de árboles mallados de elementos de encaminamiento, en el que los elementos más especializados y caros se encuentran en la parte más alta de la jerarquía.

En el futuro, un porcentaje considerable de las comunicaciones en Internet sucederán a lo largo de las redes de centros de datos. Estas redes suelen tener un estudio de ingeniería considerable, con algunos elementos comunes de diseño. Sin

embargo, los protocolos de encaminamiento, envío y gestión utilizados están diseñados para redes LAN tradicionales (que podrían considerarse aquellas utilizadas en la sección anterior, para el encaminamiento en redes empresariales) y no son totalmente adecuados, aunque sí la mejor opción dado su bajo coste y su alta compatibilidad, lo que hace que se imponga TCP/IP frente a otras opciones como [InfiniBand] o [Myrinet]. Desafortunadamente, incluso desplegando la arquitectura con los mejores switches/routers, con el gran coste asociado, la topología resultante no aprovechará adecuadamente el ancho de banda agregado. Por razones de prestaciones y precio (resultado de su economía de escala), Ethernet se impone como tecnología dominante en CPDs sobre tecnologías de redes de almacenamiento mediante mecanismos de compatibilidad como es el caso de caso de *Fiber Channel over Ethernet*, FCoE.

Recientemente, ha aumentado el número de redes data center basadas en Ethernet y redes de capa dos dado su excelente ratio rendimiento/precio y la facilidad de configuración que poseen. La creciente importancia, por razones económicas, del modelo *scale out* (basado en el uso de switches y servidores de bajo costo, aumentándolos en número) frente al modelo *scale up* (basado en aumentar la velocidad de los switches y los enlaces que forman dicha red), está llevando a las redes data center a alcanzar dimensiones de gran escala. Tal y como se explica en [Gre+08], esto conlleva a que las nuevas generaciones de arquitecturas de centros de datos tengan como base principal el uso de dispositivos *commodity*, es decir, genéricos y de bajo coste, y la escalabilidad como principal objetivo.

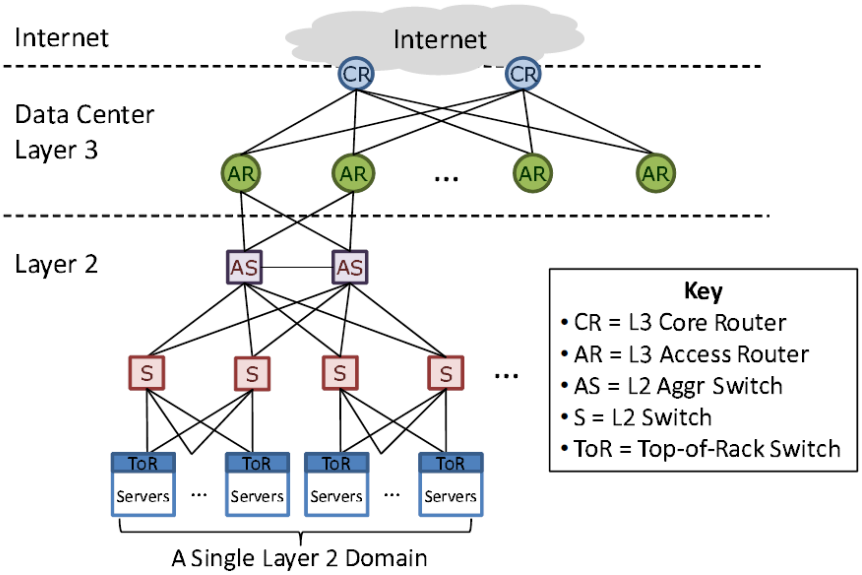


Figura 24. Una arquitectura de red convencional para centros de datos (adaptado de una figura de [CDC04])

En la siguiente sección, se estudiarán una serie de propuestas específicas para encaminamiento en centros de datos, en comparación con los dos estándares principales para encaminamiento más genérico ya estudiados en secciones anteriores, SPB y TRILL.

2.2.1 VL2

VL2 [Gre+09b] es una propuesta de protocolo para data centers relativamente reciente que es capaz de escalar para soportar centros de datos de gran tamaño, garantizando capacidad alta y uniforme entre servidores, aislamiento de uso entre servicios y semática de capa 2 Ethernet. Para ello, VL2 utiliza un tipo de direccionamiento plano de manera que las instancias de los servicios puedan ser situadas en cualquier parte de la red, reparto de carga Valiant (*Valiant Load Balancing*, VLB) [ZM04] para distribuir el tráfico uniformemente a través de los caminos en la red, y un método de resolución de direcciones basado en el sistema final para ser más escalable, sin tener que introducir complejidad al plano de control de la red. Además, una peculiaridad del diseño de VL2 es que fue conducido por un análisis previo del tráfico y datos generados a partir de un proveedor de servicios real y de tamaño grande.

Al igual que PortLand [Mys+09], que describiremos a continuación, esta propuesta fue presentada en la ACM SIGCOMM Conference de 2009. Además el mismo autor, A. Greenberg, poco antes ya había publicado artículos sobre la tendencia futura en arquitecturas de data center (escalables y realizadas con *commodities*) [Gre+08] y sobre el estudio de los costes de los centros de datos y su optimización [Gre+09a].

A continuación detallamos el diseño de VL2 [Gre+09b], mostrando previamente parte del estudio del tráfico en redes de centros de datos que se realizó y en el que se inspiró después la arquitectura de VL2, que además servirá como apoyo a la de mostrar el resto de diseños y su justificación.

2.2.1.1 Medidas e Implicaciones

En el artículo que describe VL2 [Gre+09b] se considera que, antes de comenzar su diseño, es necesario entender los entornos de centros de datos en los que se aplicará. Aunque ya se conocen las principales necesidades, desarrollar los mecanismos concretos en los que construir la red requiere entender de manera más cuantitativa la matriz de tráfico (quién manda cuánta cantidad de datos a quién y cuándo) y variabilidad (cómo de frecuentes son los cambios en el estado de la red debido a cambios en la demanda o fallos de enlace o switch, etc).

En los estudios de medida realizados, se encontraron dos resultados clave a la hora de diseñar la red. Primero, los modelos de tráfico en un centro de datos son muy divergentes (dado que ni con 50 matrices de tráfico representativas se cubrían mínimamente el total de matrices vistas), y cambian rápidamente y de manera impredecible. Segundo, las topologías jerárquicas son intrínsicamente poco fiables, incluso poniendo gran esfuerzo y gasto en aumentar la fiabilidad de los dispositivos de la red cercanos a la parte superior de la jerarquía, igualmente suceden fallos en los mismos, implicando significativos tiempos sin operación.

Análisis de Tráfico de Centros de Datos

El análisis de datos Netflow y SNMP (*Simple Network Management Protocol*) mostró las siguientes tendencias:

1. La proporción de volumen de tráfico entre servidores en comparación con el tráfico que entra y sale de los centros de datos es actualmente de 4:1 (excluyendo aplicaciones de entrega de contenidos en red).

2. La computación en centros de datos se centra donde los accesos a altas velocidades de datos, en memoria o disco, son rápidos y baratos.

3. La demanda de ancho de banda entre servidores dentro de un centro de datos, crece a mayor velocidad que la demanda de ancho de banda hacia hosts externos.

4. La red es un cuello de botella para la computación. A menudo se observan switches ToR (*Top of Rack*) con enlaces cuya utilización supera el 80%.

Análisis de la Distribución de Flujos

Distribución del tamaño de los flujos

La siguiente figura muestra la naturaleza de los flujos en los centros de datos monitorizados. En rojo y con '+': número de flujos para cierto tamaño; en azul y con 'o': distribución de bytes en la red, es decir, probabilidad de que cierto byte al azar pertenezca a un flujo de cierto tamaño. La parte superior muestra dichos valores en porcentaje y la inferior en acumulado.

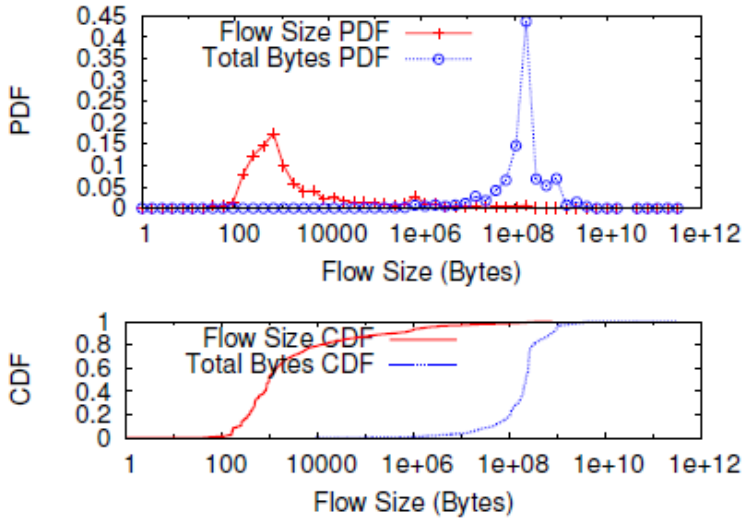


Figura 25. Estadísticos sobre flujos en los centros de datos monitorizados [Gre+09b]

Como se puede observar, los flujos "ratones" (como se denominan en el artículo, por ser pequeños) son numerosos: un 99% de los flujos son más pequeños de 100MB. Sin embargo, más del 90% de los bytes en la red pertenecen a flujos entre

100MB y 1GB, es decir, que casi todos los bytes en el centro de datos se transportan en flujos cuyas longitudes varían en ese rango. Además, es importante ver que son poco habituales los flujos con un tamaño mayor a unos pocos GB.

De manera similar a las características de los flujos en Internet [CBP95], existen muchos "ratones". Por otro lado, la distribución es más sencilla y uniforme en el caso de los centros de datos. La razón es que en los centros de datos, los flujos internos surgen dentro de un entorno de ingeniería conducido por una serie de cuidadosas decisiones de diseño.

Número de flujos simultáneos

En este caso, se demuestra que más del 50% del tiempo, una máquina media en el centro de datos tendrá alrededor de 10 flujos simultáneos, pero al menos el 5% del tiempo tendrá más de 80 flujos simultáneos. Casi nunca se ven más de 100 flujos a la vez.

Análisis de la Matriz de Tráfico

En este caso la pregunta es: "*¿Existe alguna regularidad en el tráfico que pueda ser aprovechada a través de medidas cuidadosas e ingeniería de tráfico?*". Sorprendentemente, el número de matrices de tráfico representativas en el análisis que se realizó es bastante grande. Por ejemplo, en una serie de 864 matrices analizadas indicando un día de tráfico en el centro de datos, incluso tomando 50 o 60 como referencia, el margen de error permanece con un valor mayor al 60%. Esto indica que la variabilidad en el tráfico de centros de datos no es fácilmente resumible en unos pocos modelos y, por lo tanto, la ingeniería de tráfico aplicada en base a unas pocas matrices de referencia no funcionaría correctamente en la práctica.

La siguiente pregunta sería: "*¿Cómo de predecible es el tráfico en el siguiente intervalo de tiempo conocido el tráfico actual?*". Tampoco es previsible. Esta falta de previsibilidad parte del uso de aleatoriedad para mejorar el funcionamiento de aplicaciones de centros de datos. Por ejemplo, los sistemas de archivos distribuidos reparten los datos al azar entre todos los servidores para distribución de carga y redundancia.

Características de los fallos

Se recogieron logs de fallo durante más de un año de ocho centros de datos de producción que contenían cientos de miles de servidores, con cientos de servicios de la "nube" (*cloud services*) y servían a un millón de usuarios. Los fallos analizados fueron tanto hardware, como software, de switches, routers, cortafuegos, etc. Se define un fallo como el evento que ocurre cuando un sistema o componente es incapaz de realizar su función requerida por más de 30 segundos.

Como se esperaba, la mayoría de los fallos eran pequeños (es decir, el 50% de los fallos implicaban menos de 4 dispositivos y el 95% menos de 20). Sin embargo, los tiempos de caída pueden ser significativos: el 95% de los fallos se resuelve en 10 minutos, el 98% en menos de 1 hora, el 99,6% en menos de 1 día, pero el 0,09%

duraban más de 10 días. Además, las causas principales de las caídas fueron malas configuraciones de red, bugs de firmware y componentes defectuosos.

2.2.1.2 Diseño

Los principios de diseño en VL2 son:

- **Aleatorizar para sobrellevar la volatibilidad**: Es decir, VL2 utiliza VLB (para que la distribución del tráfico dependa del destino, es decir, sea aleatoria) para poder sobrellevar la gran divergencia e impredecibilidad del tráfico en centros de datos.

- **Construir sobre una tecnología de red comprobada**: VL2 se basa en el encaminamiento de IP y otras tecnologías ya disponibles en switches *commodity* como el encaminamiento con estado de enlace y ECMP. VL2 usa un protocolo de estado de enlace para mantener la topología de switches, pero no para conocer información de los hosts, lo que evita tener que aprender grandes cantidades de información de hosts en constante cambio. Además, el diseño de encaminamiento utiliza ECMP.

- **Separar nombres de localizadores**: El esquema de direccionamiento de VL2 separa los nombres asignados a aplicaciones, de sus localizaciones, y utiliza un sistema de directorio fiable y escalable para mantener los mapeados entre nombres y localizaciones.

- **Abarcar a los sistemas finales**: Por ejemplo, el agente VL2 permite un control fino de los caminos ajustando la aleatoriedad usada en VLB. Este agente también reemplaza la funcionalidad de los mensajes ARP con consultas al sistema de directorio de VL2.

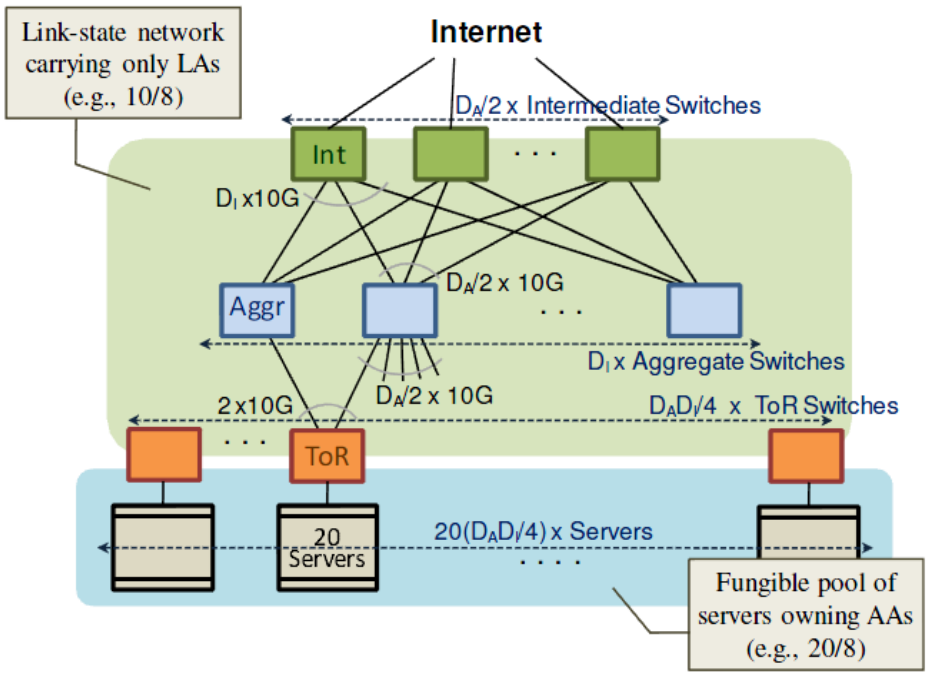


Figura 26. Un ejemplo de red Clos, muy adecuada para VLB [Gre+09b]

En cuanto a la arquitectura escogida para VL2, ya se conoce que las topologías convencionales jerárquicas tienen un ancho de banda de bisección (*bisection bandwidth*) muy pobre y además se ven más afectadas por los fallos. Por lo tanto, para VL2 se construye una red amplia, ofreciendo enorme capacidad en base a usar un gran número de dispositivos sencillos y baratos, como muestra la Figura 26. Se trata de un ejemplo de red Clos plegada [DT04].

Envío de Paquetes y Resolución de Direcciones

VL2 utiliza dos familias diferentes de direcciones IP, como se ilustra en la Figura 27. La infraestructura de la red opera usando, por un lado, las direcciones significativas topológicamente, *Locator Addresses* (LAs) y, por otro, las de aplicación, *Application Addresses* (AAs). Se asignan LAs a todos los switches e interfaces, y los switches corren un protocolo de encaminamiento de estado de enlace que sólo disemina estas LAs. Esto permite a los switches obtener toda la topología de switches y encaminar los paquetes a través de caminos mínimos en base a dichas LAs. Mientras que las AAs permanecen inalteradas al margen de la localización en cada momento del servidor. Cada AA está asociada a una LA y el sistema de directorio de VL2 almacena el mapeado de ambas, que se crea cuando un servidor comienza un servicio y se le asigna una AA.

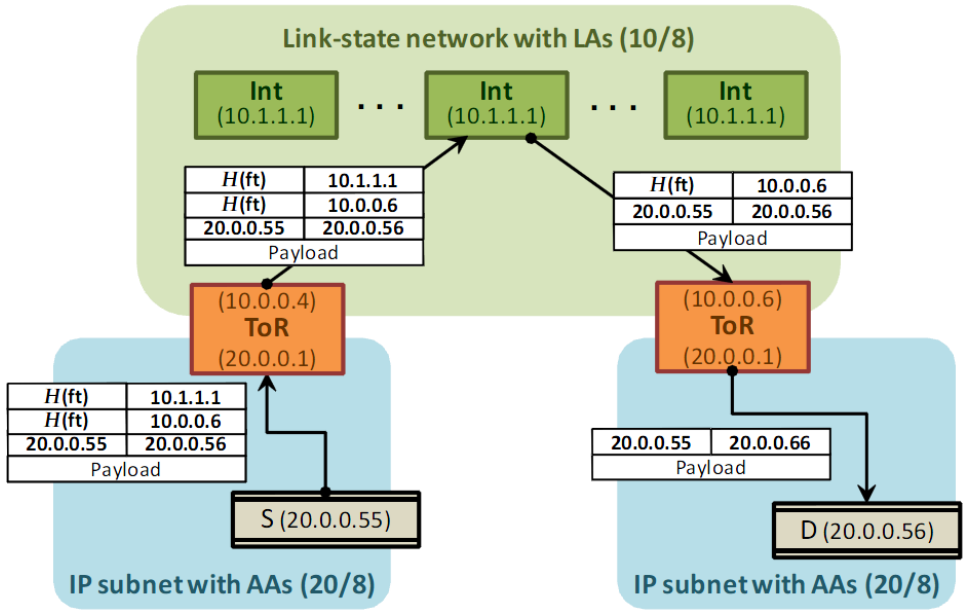


Figura 27. VLB en un ejemplo de red VL2. El emisor *S* envía paquetes al destino *D* mediante un switch intermedio escogido aleatoriamente y usando encapsulamiento IP-in-IP. AAs pertenecen a 20/8 y Las a 10/8. H(ft) es una function hash de la tupla [Gre+09b]

Para encaminar tráfico entre servidores, que usarán direcciones AA, sobre una red que conoce las rutas para las direcciones LA, el agente VL2 en cada servidor captura los paquetes en el host y los encapsula con la dirección LA del ToR de destino tal y como se muestra en la figura anterior. Cuando el paquete llega al ToR destino, el switch desencapsula el paquete y lo reparte al destino AA que contiene la cabecera interna.

Los servidores que comparten servicio están configurados de manera que piensan que todos ellos pertenecen a la misma subred IP. Por lo tanto, cuando una aplicación manda un paquete a una AA por primera vez, se emite un ARP y el agente VL2 recoge ARP Request, broadcast, y lo convierte en una consulta unicast al sistema de directorio de VL2. El directorio responde con la LA del ToR destino y el agente guardará en caché el mapeado de AA a LA para evitar futuras consultas. Un servidor no puede enviar tráfico a una AA si el servicio de directorio no quiere proporcionarle la LA correspondiente. De esta manera se pueden aplicar políticas de control de acceso.

Reparto Aleatorio de Tráfico por Múltiple Caminos

Dado que las matrices de tráfico se conoce que son aleatorias, para evitar la generación de puntos calientes (*hot spots*), es decir, puntos con elevada carga de trabajo en comparación con el resto de la red, VL2 pone en práctica dos mecanismos relacionados: VLB y ECMP. Los objetivos de ambos son parecidos, pero se aplican los dos para que uno quite las limitaciones del otro.

En la figura anterior, se ve un ejemplo en el que VLB encapula para transmitir a través de un switch intermedio al azar (en verde), luego éste desencapsula y lo manda al ToR destino, que volverá a desencapsular para mandar al destino. Para evitar tener que actualizar todos los agentes cada vez que un switch intermedio no está disponible, en la práctica se asigna la misma dirección LA a todos los switches de este tipo. Mientras, ECMP se encarga de mandar los paquetes encapsulados a los switches intermedios y de hacer que los fallos de enlace o switch sean transparentes a los agentes VL2.

Compatibilidad "Hacia Atrás"

A los servidores que necesiten ser directamente alcanzables desde Internet (es decir, desde fuera del centro de datos) se les asignan dos direcciones: una LA aparte de la AA usada para comunicación dentro del centro de datos. Esta LA se escoge de un grupo anunciado a través de BGP. Así el tráfico puede comunicarse directamente con el servidor, y el tráfico del servidor saldrá a través de los switches intermedios, mientras que dentro de los enlaces del centro de datos se difundirá medianet ECMP.

Además VL2 proporciona semántica de capa 2 para que las aplicaciones tengan compatibilidad "hacia atrás", y eso incluye soportar broadcast y multicast. VL2 elimina complementamente las fuentes más comunes de broadcast: ARP y DHCP. ARP se reemplaza por el directorio, y los mensajes DHCP se interceptan en el ToR usando agentes relay convencionales de DHCP y reenviando en unicast el tráfico a los servidores DHCP. Para manejar el resto de tráfico broadcast de capa 2, a cada servicio se le asigna una dirección IP multicast, y todo el tráfico broadcast en ese servicio se maneja vía IP multicast usando dicha dirección multicast. El agente VL2 limita el tráfico broadcast para prevenir tormentas.

El Sistema de Directorio de VL2

El directorio de VL2 proporciona tres funciones clave: búsqueda de mapeos de AA a LA, actualización de estos y un mecanismo de actualización de caché reactivo

de manera que las consultas y actualizaciones se realicen rápidamente. Los objetivos de diseño son facilitar la escalabilidad, fiabilidad y alto rendimiento. El directorio se compone para ello de un número modesto (50-100 servidores de cada 100.000) de servidores de directorio, y un pequeño número (5-10 servidores) de servidores para replicación asíncrona y almacenamiento fiable de mapeos.

2.2.1.3 Implementación y Evaluación

Para evaluar VL2 se utiliza un prototipo corriendo en una plataforma de 80 servidores y 10 switches *commodity*. Sus principales objetivos eran mostrar que VL2 puede construirse a partir de componentes disponibles hoy en día, y segundo, que la implementación cumple los objetivos propuestos inicialmente y que son: proporcionar alta capacidad de manera uniforme (su prototipo maneja 2,7TB de datos entre 75 servidores en 395 segundos, manteniendo una velocidad que es igual al 94% de la máxima posible), así como aislamiento en su funcionamiento (*performance isolation*), todo ello utilizando semántica de capa 2. Pero además, se comprueba también el buen funcionamiento de VLB y se mide los rendimientos del sistema de directorio y de convergencia tras fallos de enlace.

En el paper de VL2 se encuentra la implementación más en detalle (fotos), así como los resultados pertinentes (muy concretos de dicha implementación). Para más información, se recomienda su lectura.

2.2.2 PortLand

PortLand [Mys+09] es otra propuesta de arquitectura para redes de data centers relativamente reciente que posee un control centralizado, pseudo-direcciones MAC basadas en la localización (que se asignan a los hosts y a los switches a partir de un protocolo de descubrimiento) y prohibición de giros Up/Down (véase Apéndice A) para prevenir bucles.

Esta propuesta fue presentada en la ACM SIGCOMM Conference de 2009, pero ideas con fundamentos similares también pudieron encontrarse no sólo en la misma edición de la conferencia (*VL2* [Gre+09]), sino también previamente en el SIGCOMM 2008, como "*A Scalable, Commodity Data Center Network Architecture*" [ALV08], que curiosamente tiene a A. Vahdat como autor común con la propuesta de PortLand y que inicia el concepto de la falsa topología *fat-tree*, pues no es un *fat tree* de por sí (explicado en el Apéndice B), pero se asemeja. Las tres propuestas tienen además en común el uso del adjetivo "escalable" para sus arquitecturas de redes de centros de datos. Pero además, A. Vahdat presentó posteriormente lo que se podría considerar como una evolución para PortLand, "*Hedera: Dynamic Flow Scheduling for Data Center Networks*" [ARR+10], que intenta demostrar, entre otras cosas, la ventaja de tener un controlador centralizado que realice decisiones en base al estado concreto y actual de la red, y para ello utiliza el mismo banco de pruebas de PortLand.

A continuación detallamos el diseño de PortLand, que concretamente inspiró el desarrollo de una de las arquitecturas que se expondrán más adelante en esta memoria como trabajo de la presente Tesis: Torii-HLMAC.

2.2.2.1Diseño

El objetivo de PortLand es desarrollar un entorno para redes de data center escalable capaz de encaminar en base a la capa 2 (*switching*) en base a un sistema de asignación de direcciones. Se considera que construír y mantener data centers con decenas de miles de elementos de computación requiere modularidad, planificación por adelantado y mínima interacción humana (pues si no, sería muy complejo). Por lo tanto, la topología base de un data center no debería evolucionar o modificarse rápidamente. Cuando la red se expande, normalmente implica añadir más "hojas" (es decir, filas de servidores) a la topología de múltiples raíces (*multi-rooted*) que se describe a continuación.

En el paper que describe PortLand, consideran el diseño de un dominio de capa 2, tolerante a fallos y escalable sobre lo que ellos denominan un *fat-tree*, aunque concretamente la topología usada es la que se muestra en la siguiente figura y no es, estrictamente hablando, un *fat tree* (Apéndice B), dado que los enlaces no van "engordando" hacia arriba y posee bucles. Esto ha causado confusión en publicaciones que han seguido usando el término en base al paper de PortLand, pero esta topología es en realidad un ejemplo particular de una red Clos [Clo52].

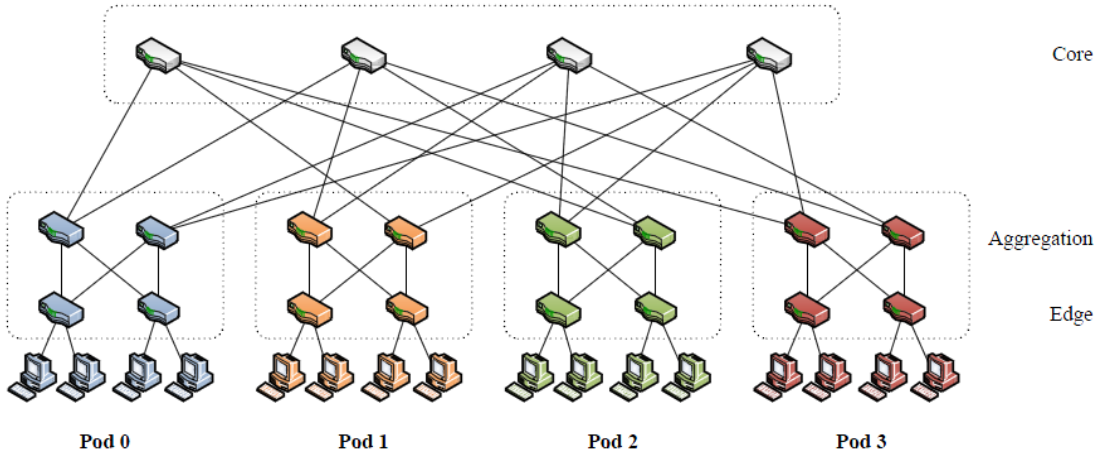


Figura 28. Topología ejemplo usada en la descripción de PortLand [Mys+09]

Esta topología se compone de 3 niveles jerárquicos de switches. El primer nivel lo componen los *core* (o raíz), el segundo los *aggregation* (o de agregación) y el tercero los *edge* (o frontera). A su vez, los switches de segundo y tercer nivel forman lo que se denomina un *pod*. En la imagen se puede ver cómo cada switch frontera tiene 2 hosts finales conectados y, por tanto, cada *pod*, tiene 4 en total. En general, una topología en tres etapas como la de la figura construída en base a switches de k puertos puede soportar comunicación no bloqueante entre $k^3/4$ hosts usando $5k^2/4$ switches de k puertos cada uno. A su vez, la topología contendrá k *pods* individuales, cada uno con $k^2/4$ hosts. En el ejemplo de PortLand, los switches son de 4 puertos cada uno, lo que permite comunicación no bloqueante entre 16 hosts en base al uso de 20 switches, conteniendo 4 pods.

Fabric Manager (Gestor Centralizado)

PortLand emplea un gestor centralizado al que bautiza como *fabric manager*. Éste mantiene un estado "blando" o volátil, comúnmente denominado "*soft state*", sobre la información de configuración de la red, así como de la topología. El *fabric manager* es un proceso que corre en una máquina dedicada y es responsable de ayudar a la resolución de consultas ARP, tolerancia a fallos y envío multicast. Puede ser un host conectado de manera redundante en la topología o puede correr en una red de control por separado.

Direcciones Pseudo MAC basadas en Posicionamiento

La base del encaminamiento eficiente, así como de la migración de máquinas, en el diseño de PortLand son las direcciones *Pseudo MAC* (PMAC) jerárquicas. PortLand asigna una única PMAC a cada host final. Esta dirección PMAC codifica la localización del host en la topología, lo que implica que, por ejemplo, todos los hosts de un mismo pod tendrán el mismo prefijo en su PMAC asignada. Los hosts permanecen sin modificar, creyendo que mantienen su dirección *Actual MAC* (AMAC), MAC reales. Cuando un host realiza una petición ARP, recibirá la PMAC del host destino. El envío de tramas se basa en las PMAC, generando tablas muy pequeñas de encaminamiento. Los switches frontera reescriben las cabeceras de PMAC a AMAC y viceversa para mantener la ilusión de que las direcciones MAC no se modifican en los hosts finales.

Los switches frontera aprenden un número de pod único y un número de posición único dentro de cada pod. Para ello emplean el protocolo *Location Discovery Protocol* (LDP), que se comenta en un apartado a continuación, para asignar estos valores. Así pues, los switches frontera asignarán una dirección PMAC de 48 bits a cada host directamente conectado, de la forma ***pod.position.port.vmid***, donde *pod* (16 bits) indica el número de pod del switch frontera, *position* (8 bits) su posición en el pod, y *port* (8 bits) el número de puerto local del switch al que el host está conectado. Además, el campo *vmid* (16 bits) permite multiplexar diferentes máquinas virtuales en una misma máquina física (o diferentes hosts físicos situados tras un bridge fuera de la topología).

Cuando un switch frontera observa una dirección MAC que no se había visto antes, el switch calcula la PMAC y mapea en una tabla local la IP y AMAC del host a su PMAC asignada, para futuras traducciones (de AMAC a PMAC y viceversa). A su vez, comunica este mapeo al *fabric manager*, como se puede ver en la figura siguiente.

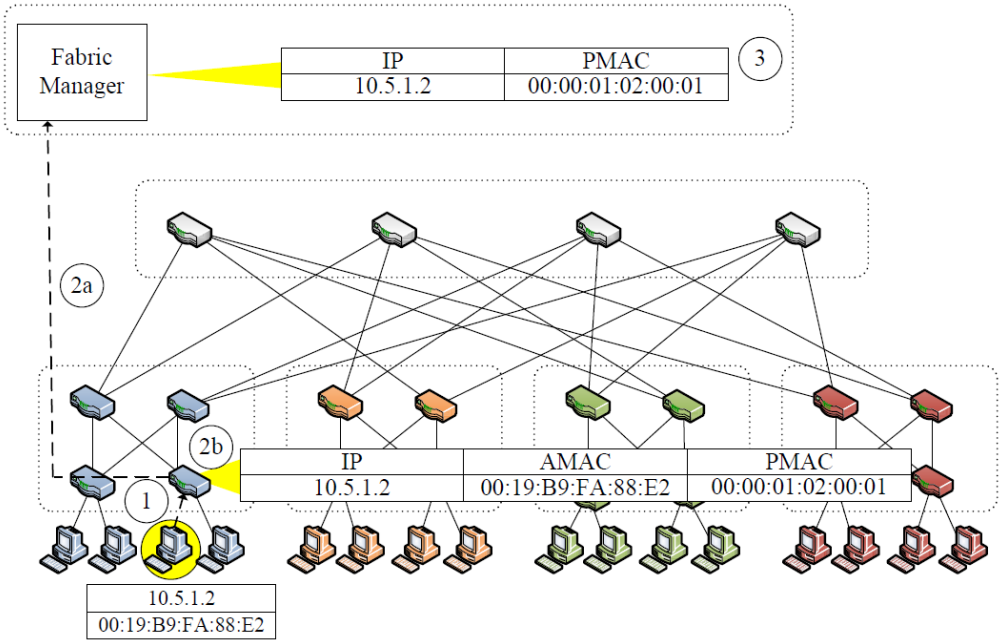


Figura 29. Mapeo de direcciones AMAC-PMAC en PortLand [Mys+09]

Proxy-based ARP

Por defecto, Ethernet reenvía en broadcast los ARPs a todos los hosts en el mismo dominio de capa 2. En PortLand, el *fabric manager* se utiliza, entre otras funciones, para reducir la inundanción de tramas en la topología por estos mensajes broadcast. Para ello, los switches frontera interceptan los mensajes ARP Request y se los pasa al *fabric manager*, que consultará si ya existe una entrada para la IP objetivo (es decir, si ya se realizó un envío broadcast previo y se aprendió la información recogida del ARP) y no es necesario inundar entonces de nuevo.

Descubrimiento de Localizaciones Distribuido

Los switches de PortLand usan su posición (es decir la PMAC) en la topología global para el encaminamiento. Esta posición podría ser asignada por un administrador, pero en la publicación presenta como alternativa un protocolo de descubrimiento (*Location Discovery Protocol* – LDP), que utiliza el envío de mensajes periódicos (*Location Discovery Message* – LDM) a través de todos los puertos de los switches, para definir la posición y monitorizar la validez de la misma. Estos mensajes contienen la siguiente información (inicialmente todos los valores son desconocidos menos el identificador de switch y el número de puerto de envío):

- Identificador de switch (*switch_id*)
- Número de pod (*pod*)
- Posición (*pos*)
- Nivel en el árbol (*level*)
- Dirección del puerto arriba/abajo (*dir*)

Encaminamiento Demostrado Sin Bucles

Una vez que los switches han establecido sus posiciones locales usando LDP, emplean las actualizaciones de sus vecinos para generar sus tablas de encaminamiento. Por ejemplo, los switches core aprenden el número de pod de los switches de agregación directamente conectados. Así cuando se ha de envíar una trama, un core switch simplemente inspecciona los bits correspondientes al número de pod de la dirección PMAC de destino. De manera similar, los switches del nivel intermedio aprenden el número de posición de todos los switches frontera directamente conectados, y deben determinar si un paquete está destinado a un host en el mismo pod o en uno diferente, examinando la PMAC.

En el caso de la comunicación multicast, PortLand mapea los grupos a un core switch usando una función hash y todos los paquetes multicast se envían a través de ese core. Los switches frontera reenvían las peticiones *join* de IGMP al *fabric manager* usando la PMAC del host que se quiere unir al grupo. De esta forma el *fabric manager* instala el estado de reenvío en todos los switches core y de agregación necesarios para garantizar la entrega del paquete multicast a los switches frontera con al menos un host interesado en el envío.

Se garantiza que no hay bucles en la transmisión gracias al uso del reenvío Up/Down (es decir, primero el paquete va hacia arriba y luego hacia abajo, y una vez que empieza a transmitirse hacia abajo, nunca vuelve hacia arriba).

Encaminamiento Tolerante a Fallos

PortLand detecta fallos de enlace en base a los mensajes LDM (del protocolo LDP ya comentado antes). Si no se recibe un LDM a lo largo de cierto periodo de tiempo configurado, un switch considera que ha sucedido un fallo de enlace.

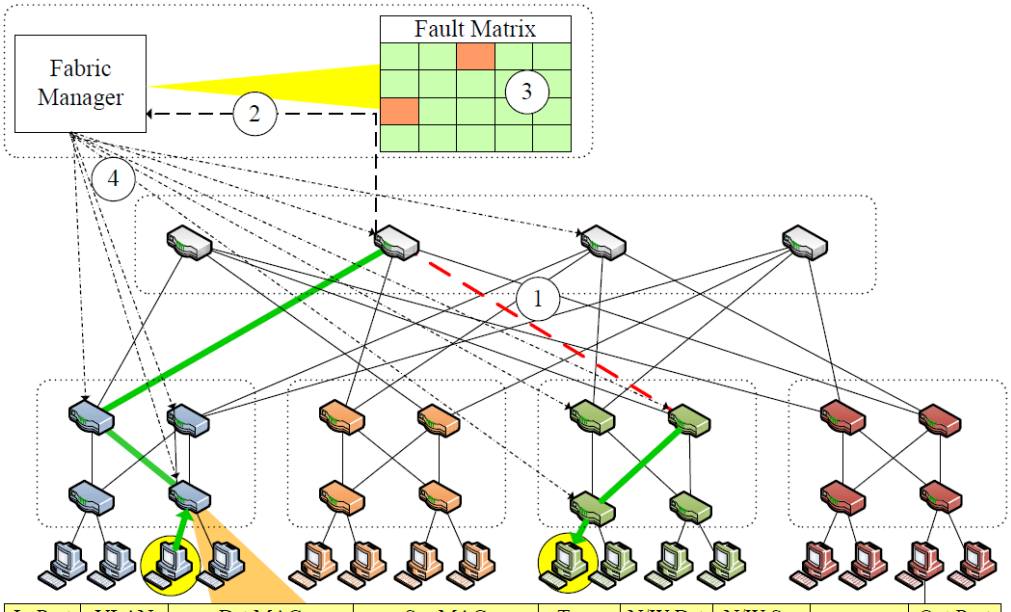


Figura 30. Detección de fallo para tramas unicast en PortLand [Mys+09]

En el caso de tramas unicast (figura anterior), el switch que lo detecta (paso 1), informa al *fabric manager* sobre el fallo (paso 2), y éste último actualiza la matriz lógica que posee de la conectividad por enlaces en toda la topología (paso 3). Finalmente, el *fabric manager* informa a todos los switches afectados por dicho fallo de enlace (paso 4) para que recalculen sus tablas de envío.

Para el resto de tramas, multicast y broadcast, en el documento descriptivo de PortLand se considera un escenario de fallo en el que no exista cobertura completa de transmisión de una trama de este tipo desde un solo core switch. Se resuelve el caso de multicast, pero en broadcast no se detalla mucho más, sólo se comenta que conlleva implicaciones peores.

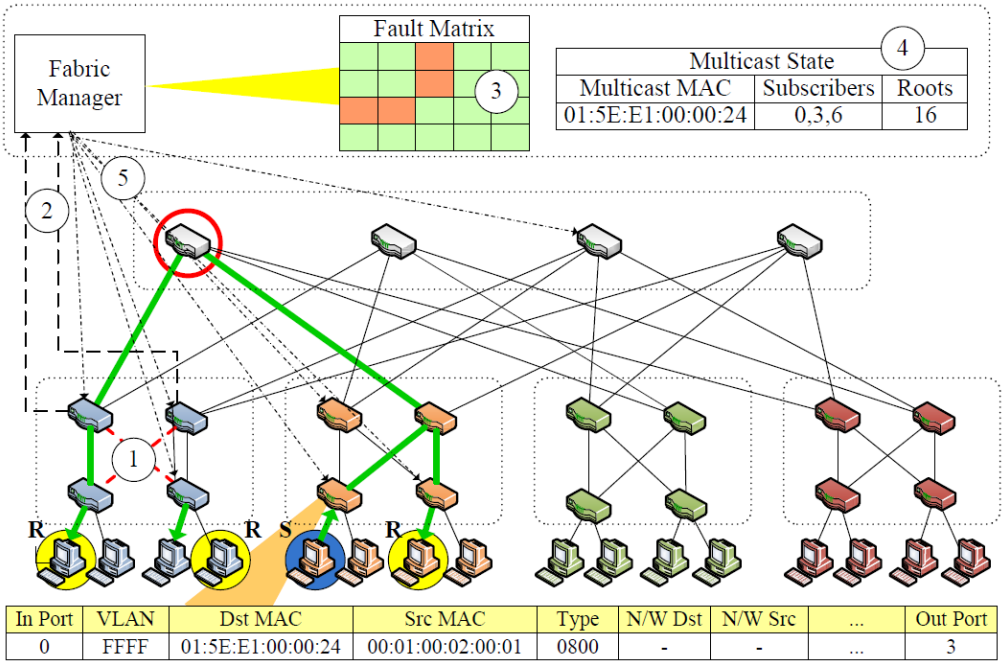


Figura 31. Detección de fallo para tramas multicast en PortLand [Mys+09]

El caso de fallo se muestra en la anterior figura. Considerando que el host que realiza el envío es "S" y los receptores son tres y están marcados con "R", si fallan los dos enlaces marcados en 1, no hay core posible desde el que se puedan alcanzar los 3 receptores. En este caso, los switches detectarán el fallo y notificarán al *fabric manager* (paso 2), que actualizará su matriz lógica de fallos (paso 3) y recalculará las entradas de envío para todos los grupos multicast afectados (paso 4). Finalmente insertará el estado correspondiente de envío en las tablas correspondientes (paso 5). Una vez esto ha sucedido, el estado de reenvío de tramas quedaría como se muestra en la figura siguiente (nótese que ahora cada paquete se reenvía a dos cores por separado):

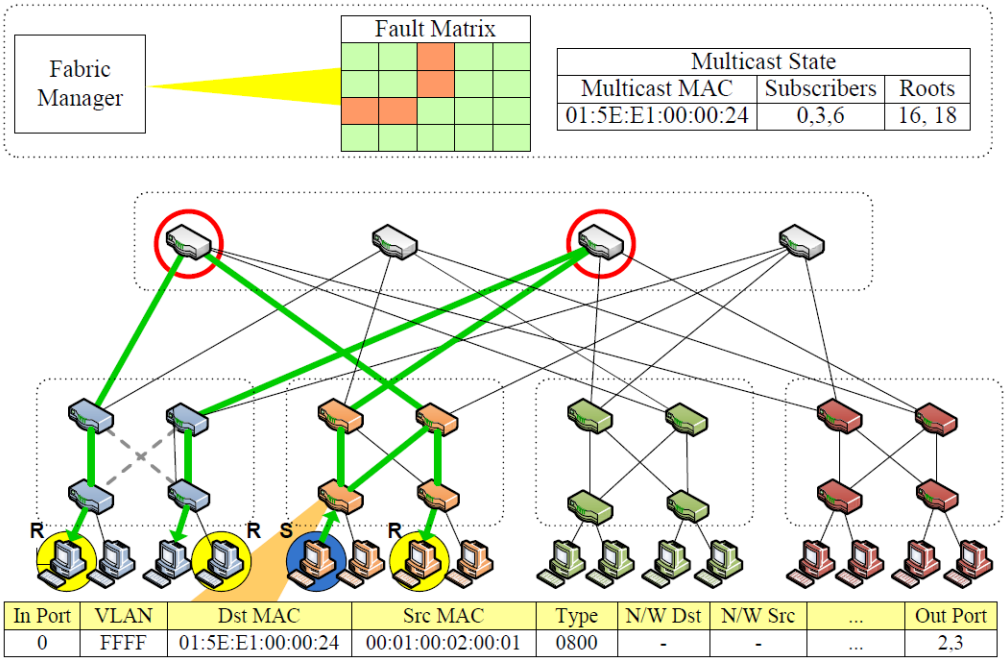


Figura 32. Tras la detección de fallo para tramas multicast en PortLand [Mys+09]

2.2.2.2 Implementación y Evaluación

El equipo de PortLand realizó una evaluación basada en la implementación del protocolo en 20 switches sobre tarjetas NetFPGA de 4 puertos GigE y 16 hosts finales. Cada switch tenía instalado la versión 0.8.9 de OpenFlow, para interactuar con el *fabric manager*, implementado como un controlador OpenFlow.

A la hora de evaluar, se centraron en ver la eficiencia y escalabilidad de dicha implementación. Para ello presentaron medidas para caracterizar la convergencia y la sobrecarga que implicaba la parte de control, tanto para tramas unicast como para multicast. También presentan cálculos de escalabilidad en función del número de hosts y el número de mensajes ARP que se envían por segundo, para ver la cantidad de tráfico con el que tendría que lidiar el *fabric manager*, así como el número de cores (capacidad de procesamiento) necesarios. Finalmente se presenta el caso de migración de máquinas virtuales, migrando una y considerando que ésta genera un ARP gratuito tras la migración.

En el paper de PortLand se encuentra la implementación más en detalle, así como los resultados pertinentes (muy concretos de dicha implementación). Para más información, se recomienda su lectura.

2.2.3 DCell, BCube, SecondNet y DAC

De manera similar a las propuestas VL2 y PortLand, también en la misma conferencia Sigcomm, en los años 2008 y siguientes, aparecieron DCell [Guo+08], BCube [Guo+09] y SecondNet [Guo+10], así como la propuesta DAC, "*Generic and Automatic Address Configuration for Data Center Networks*" [CGW+10]. Todas ellas

propuestas para arquitecturas de redes de centro de datos que, en este caso, comparten al autor C. Guo.

En este grupo de investigación de C. Guo, al contrario de VL2 y PortLand, que se centran más en el protocolo de comunicación partiendo de una topología concreta, las propuestas iniciales DCell y BCube aportan más la idea de un estudio teórico, más analítico, de arquitecturas con formas de topologías diferentes y en un sentido más genérico. Como sus nombres indican, DCell (propuesta del Sigcomm 2008) aporta una topología para centros de datos compuesta por *células*. Mientras que BCube (Sigcomm 2009), tras conocer además la propuesta *fat-tree* del grupo de A. Vahdat del Sigcomm 2008 ("*A Scalable, Commodity Data Center Network Architecture*"), propone una topología que, como ellos mismos comentan en el artículo, está muy relacionada con lo que sería la topología de hipercubo generalizado [BA84], de ahí el nombre, y que en su análisis realiza una comparativa directa con DCell y dicho *fat-tree*. En todo caso, tanto DCell como BCube, partían del uso de switches *commodity* o, como también los denominan en sus artículos, *mini-switches*.

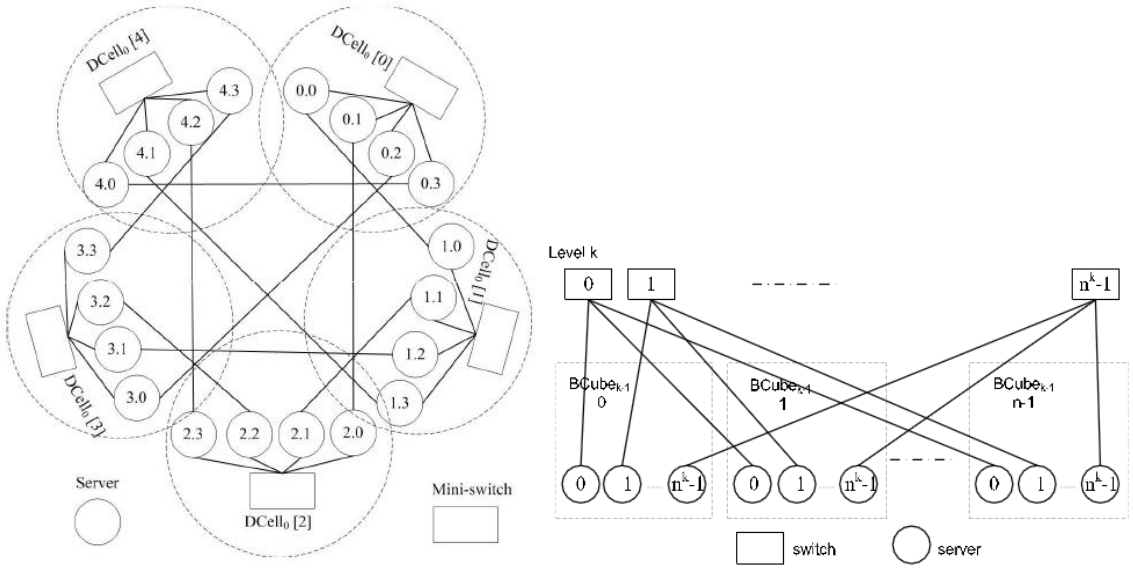


Figura 33. A la izquierda, la topología propuesta para DCell [Guo+08] y, a la derecha, la de BCube [Guo+09]

Así en el año 2010, tras la publicación de VL2 y PortLand, surgen dos nuevas ramas, por un lado SecondNet y, por otro lado, la propuesta genérica DAC. SecondNet se trata de un centro de datos virtual, *virtual data center* (VDC), que utiliza encaminamiento por puerto en base al origen, *port-switching source routing* (PSSR), y es una propuesta pensada para ser utilizada en cualquier tipo de topología. Es el *VDC manager* el que administra todo el centro de datos físico, controla todos los recursos y la admisión de peticiones de nuevos VDCs, que compartirán un mismo espacio de direcciones IP, y aplicando cierto algoritmo detallado en el artículo, se decide cómo las máquinas y nodos virtuales de un VDC se mapean dentro de servidores y caminos físicos. Por otro lado, PSSR consiste en realizar el encaminamiento en base a transportar, en la propia cabecera, un listado de los puertos de salida de los switches intermedios, en vez de llevar direcciones que luego los switches tengan que consultar en cierta tabla para buscar su estado (puerto de

salida), con el correspondiente uso de un protocolo de señalización para conocer dicho estado.

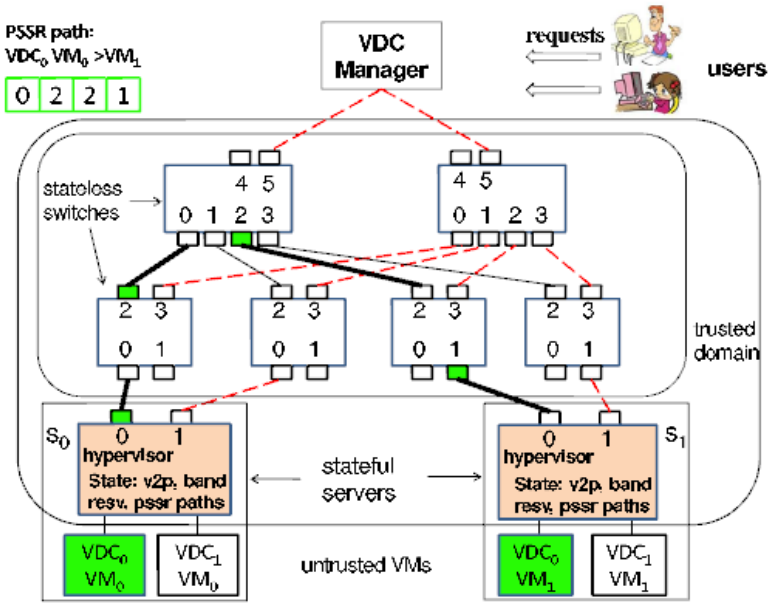


Figura 34. Arquitectura de SecondNet. La líneas punteadas rojas forman la señalización de un árbol expandido, mientras que las negras muestran un camino formado por PSSR [Guo+10]

Finalmente, DAC propone un sistema de autoconfiguración de direcciones genérico para cualquier red de centro de datos. Esta propuesta es muy similar a PortLand, salvo que PortLand es específica de la topología *fat-tree* y DAC se supone la primera capaz de emplearse en cualquier topología. Una característica importante compartida para todos los centros de datos es que estos pertenecen y son operados por una única organización y DAC se aprovecha de esta propiedad para utilizar un gestor de configuración centralizado, al que llaman *DAC manager*. Como DAC es una solución genérica, no puede suponer nada de antemano, por lo que usa los dos siguientes grafos como entrada para funcionar:

1. *Blueprint*: Los centros de datos tienen estructuras bien definidas, así que antes de desplegar la red, se debe generar un *blueprint* (Figura 35a), esquema, plano o anteproyecto, para guiar la construcción del centro de datos. Éste se compone de:

    - Interconexiones entre dispositivos
    - ID lógicas para cada dispositivo

2. *Topología de red física:* La topología física (Figura 35b) se construye siguiendo las interconexiones definidas en el *blueprint*. En esta topología física, se usa la dirección MAC como ID único para identificar a un dispositivo. Si el dispositivo tiene múltiples direcciones MAC, se usa la menor.

Los resultados de simulación muestran que DAC puede encontrar hasta los funcionamientos incorrectos más complicados de detectar y es capaz de autoconfigurar grandes centros de datos con 3,8 millones de dispositivos en tan sólo 46 segundos.

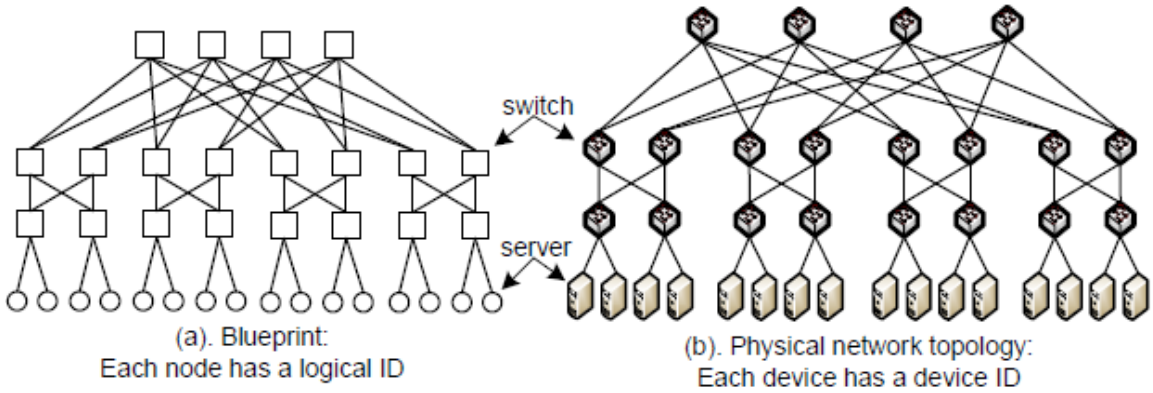


Figura 35. Un ejemplo de *blueprint* en DAC, y la topología física construída siguiendo las interconexiones del *blueprint* [CGW+10]

Éste ha sido un pequeño resumen de las cuatro propuestas y su evolución. Para más información se recomiendo la lectura de los respectivos artículos.

2.2.4 Otras propuestas

Otras propuestas ya no tan directamente relacionadas con las anteriores, pero sí enfoques interesantes a la hora de pensar en arquitecturas de centros de datos son:

- "*A Scalable Multi-Tenant Network Architecture for Virtualized Datacenters*" [MYM+11], que pone en evidencia la tendencia actual en investigación a los centros de datos virtualizados.

- "*Towards predictable datacenter networks*" [BCK+11], que presenta la necesidad de que las redes de centros de datos sean algo más predecibles de lo que son actualmente (recuérdese el análisis inicial que se presentó en la sección de VL2) para mejorar su rendimiento, y para ello propone la creación de redes virtuales.

- "*Jellyfish: Networking Data Centers Randomly*" [SHP+11], que supone una visión nueva a la hora de construir arquitecturas en base a la generación de una topología con un grafo aleatorio, garantizando así la flexibilidad de cambiar la topología (al no estar prefijada como otras propuestas) en cualquier momento sin perder la capacidad de aportar el mayor ancho de banda en la red en todo momento. Las topologías generadas van adquiriendo una forma similar a una medusa (*jellyfish*), de ahí el nombre, sin embargo, esta aleatoriedad genera otros problemas como es el encaminamiento y el cableado.

- *"An OpenFlow-based energy-efficient data center approach*" [JP12], que directamente se trata ya de una propuesta más enfocada al consumo, otra de las grandes ramas de investigación de redes de centros de datos, junto a la de mejoras de rendimiento en base a arquitecturas eficientes.

Capítulo 3

Planteamiento

En este capítulo primero se define el problema a resolver, es decir, conocidas las ventajas de los bridges frente a otras tecnologías, definir qué inconvenientes deben, y pueden, ser superados, especialmente en los escenarios prioritarios como los centros de datos. En una segunda parte, se analizará el espacio de soluciones, lo que viene a ser realizar una valoración del estado del arte con vistas a su aplicación en el problema planteado. Finalmente se expone el proceso de diseño y las decisiones tomadas, así como las conclusiones alcanzadas.

Se ha de tener en cuenta en este análisis, que el caso del encaminamiento en centros de datos es uno concreto dentro del encaminamiento en redes empresariales. Es decir, el estudio en el capítulo del estado del arte para el encaminamiento en redes empresariales podría ser aplicable en encaminamiento en centros de datos. Sin embargo, al ser los centros de datos un caso concreto, en el que muchas veces se estudia en profundidad la arquitectura de la red para aprovechar mejor los recursos, en la práctica se diseñan encaminamientos específicos a dicha topología fijada, pues serán más eficientes. Por lo tanto, el análisis se realizará de manera genérica para ambos encaminamientos y puntualmente se añadirán notas para uno de los dos.

3.1 Definición del problema

Una definición que resume el problema que nos ocupa podría ser la siguiente: los modernos conmutadores Ethernet (*bridges*) han hecho posibles nuevos límites para las redes locales, que antes eran de pequeño tamaño y corto alcance. El bajo coste por puerto, alto rendimiento, simplicidad de configuración y la independencia del direccionamiento IP son algunas de sus grandes ventajas frente a los routers. En el caso concreto de las redes de centros de datos, a las ventajas anteriores se suma su alta compatibilidad (TCP/IP) frente a opciones específicas como Myrinet e InfiniBand, que quedan casi desbancados para usos en centros de datos muy concretos y con necesidades específicas.

Así pues, inicialmente se plantea la necesidad de un modelo de red que no sólo venza los inconvenientes de los bridges frente a los routers (redes empresariales), Myrinet e InfiniBand (redes de centros de datos), sino que potencie sus ventajas y además supere a las soluciones actuales, como son SPB y TRILL (redes empresariales), y principalmente VL2 y PortLand (redes de centros de datos), ya detalladas en el capítulo anterior del estudio del Estado del Arte y que se verá en la sección del análisis del espacio de soluciones de este mismo capítulo.

A continuación se examinarán primero los conceptos de *bridging/switching* y *routing*, para luego pasar a formular las ventajas e inconvenientes de los bridges y routers, y así finalmente poder exponer los problemas a resolver.

3.1.1 Conceptos de *bridging/switching* y de *routing*

En entornos de trabajo no muy cercanos al tema concreto tratado en la presente Tesis, es posible que los términos *bridging* (o *switching*) y *routing* sean complicados de diferenciar. De hecho, incluso se podría decir que existe confusión sobre sus definiciones en entornos especializados, como comenta [Pep10]. Radia Perlman [Per04] considera que el *routing* es transporte (*relay*) en capa 3 (capa de red), mientras que el *bridging* es una función idéntica pero realizada en capa 2 (capa de enlace), siendo la diferencia que en la capa 3 se trata de comunicación entre sistemas finales y en la capa 2 es entre dispositivos vecinos.

Siguiendo el artículo [Pep10], es posible realizar un análisis pragmático comparando lo que es el *bridging* Ethernet (y concretamente el *transparent bridging* tal y como se define en el grupo IEEE 802.1) con el *routing* IP (o CLNS, *Connectionless-mode Network Service*). Así pues, se podrían caracterizar diversas diferencias, principalmente las siguientes:

Objetivo de diseño: IP fue diseñado para soportar una infraestructura de red de encaminamiento de paquetes global, mientras que el *bridging* Ethernet fue diseñado para emular un único enlace compartido. Por lo tanto, diversas decisiones de diseño se veían afectadas por estas perspectivas: escalabilidad frente a transparencia.

Envío de datos: En IP se envían los datagramas en base a las tablas de encaminamiento IP, nunca se realizan múltiples copias y además se descartan tramas para destinos desconocidos. Mientras que en Ethernet simplemente se inunda por todos los puertos (salvo por el que recibe), emulando el cable compartido. Es decir, en IP prima el conocimiento del envío y descarte por defecto, mientras que en Ethernet por defecto se reenvía inundando para que los datos lleguen, aunque incluso no sean conocidos.

Detección de bucles: IP (como la mayoría protocolos de capa 3) tiene un contador de saltos en su cabecera que detecta los bucles, campo TTL, pero Ethernet (así como la mayoría de protocolos de capa 2) no, entre otras razones porque las tramas no deben ser modificadas por los bridges. Además, la inundación en el envío de datos realizada en Ethernet puede incluso crear múltiples copias de un paquete que está dentro de un bucle y dichas copias crecen exponencialmente con cada iteración en el bucle, resultando en un colapso de la red.

Multicast: Los routers bloquean los paquetes broadcast o multicast a no ser que estén configurados explícitamente para enviarlos. En el caso de multicast, esto se realiza en base a subscripciones de los hosts que estén interesados en ciertos paquetes. Los bridges transparentes no pueden realizar este filtrado al basar su envío en la inundación y si lo intentaran hacer en base a la inspección de paquetes IGMP, sólo podrían aplicarlo para direcciones multicast (IP) conocidas, pues con las desconocidas deberían seguir inundando.

Tablas de encaminamiento: Las tablas de encaminamiento en IP se construyen en base a intercambio de información entre sus conexiones y sus rutas estáticas, mientras que las tablas de los bridges se construyen averiguando, escuchando el tráfico y extrayendo direcciones MAC de las tramas, y esta información nunca se intercambia entre bridges.

Direccionamiento: Las direcciones de capa 3 son configurables y normalmente incluyen cierta información sobre la topología, siendo así escalables. Las direcciones de capa 2 se suponen estáticas (fijadas por el fabricante) y pueden estar colocadas aleatoriamente en cualquier punto de la red dado que son planas, asegurándose su unicidad y permanencia en el tiempo.

Escalabilidad: Todos los protocolos de capa 3 tienen algún mecanismo que agrupa en conjuntos, para resumir, la información de encaminamiento, permitiendo que sea escalable. Pero las direcciones de capa 2 no tienen ningún tipo de información topológica y por ello, los bridges no pueden crear conjuntos, ni resumir dicha información o generar otro mecanismo escalable, y a lo único que pueden optar es a aprender sólo ciertas direcciones (por ejemplo, aprender sólo información de los switches frontera, habiendo generado una jerarquía antes).

Seguridad: El filtrado de paquetes entre subredes IP es una características común de cualquier router decente, permitiendo diseñar zonas diferentes de seguridad en la red, algunos switches tienen esta funcionalidad, a cambio de aumentar la complejidad de configuración. Sin embargo, el mayor problema es que se puede "engañar" (hacer *spoofing*) a un bridge simplemente mandando tramas con direcciones MAC erróneas, y esto no es posible que suceda con routers a no ser que

se vaya más allá y se truque el protocolo de encaminamiento en sí. Es posible reducir este riesgo implementando seguridad por puertos, pero es evidente que es complejo hacer seguro a un entorno diseñado para emular un simple enlace compartido.

Fragmentación: IP se diseñó para abarcar múltiples medios físicos con diferentes características y soportar fragmentación de datagramas, por ejemplo. El concepto de bridging fue diseñado para conectar segmentos con tecnología uniforme, lo que estaba bien siempre y cuando el tamaño de la unidad máxima de transferencia (*Maximum Transmission Unit*, MTU) Ethernet fuese constante, y esto no sucede con las denominadas tramas *jumbo* (tramas Ethernet con más de 1500 bytes de carga).

Orden de los paquetes: Los paquetes desordenados son comunes en cualquier topología multicamino, por lo que los protocolos de capa 3 fueron diseñados para tratar con ellos, ya fuera reordenándolos (TCP) o descartándolos (UDP), pero esto no sucede para capa 2 y es un problema.

Diferentes medios: IP puede (por definición) usarse en todo tipo de tecnologías de enlace de datos, mientras que en *bridging* se fuerza a usar una única tecnología de capa 2.

Coste: Los switches de capa 2 son usualmente más baratos que los de capa 3 (normalmente combinación de capa 2/3 en práctica). Las razones para la diferencia de coste son numerosas, incluyendo:

- Lo más habitual en la industria son los switches de capa 2 y además ya se dispone de dispositivos de bridging de alta velocidad y bajo coste.
- La búsqueda de direcciones MAC en tablas es más sencilla que la de direcciones IP y más fácil de implementar en hardware. Se necesita utilizar simples memorias CAM (*Content Addressable Memory*) para las tablas de MAC y TCAM (*Ternary CAM*) con lógica de salida adicional para las tablas de IP.
- Los dispositivos de capa 3 deben realizar filtrado de paquetes IP e implementar listas de acceso en hardware (normalmente con incluso mayores TCAM) es caro.

Configuración: En su definición más básica, los bridges son dispositivos *plug-and-play*, mientras que los routers siempre requieren configuración (en el mejor caso, es necesario configurar al menos las subredes IP y los protocolos de encaminamiento IP).

Multicaminos (*equal-cost multipath*): Los routers pueden balancear la carga entre caminos de igual coste a través de la red. Esto no es una opción inicial entre bridges, que además suelen deshabilitar caminos redundantes para evitar bucles (como en el caso de STP), pero protocolos como TRILL o SPB lo permiten.

Solución de problemas: Es imposible solucionar los problemas de una red de puentes desde un host final dado que la red está diseñada para ser invisible. Sin

embargo, la mayoría de protocolos de capa 3 tienen mecanismos para informar sobre errores que, al menos, darán una pista sobre dónde puede estar el problema.

Predicibilidad: Las tablas de encaminamiento en capa 3 sólo se modifican mediante los protocolos de plano de control, mientras que las de capa 2 se modifican en cada momento fuera del plano de control, con el propio plano de datos, es decir, se analiza y aprende la dirección MAC origen de cada trama que llega.

Movilidad de hosts: Precisamente, el hecho de que se aprenda fuera del plano de control en capa 2 hace que, por otro lado, la movilidad de hosts sea instantánea (en cuanto el host que se ha movido mande un mensaje en broadcast, por ejemplo un ARP gratuito, todos los bridges reajustarán sus tablas de encaminamiento). En una red de routers se puede hacer algo similar, pero siempre con retardos mucho mayores al realizarse dentro del plano de control.

Así pues, conociendo las diferencias entre un tipo de encaminamiento y otro, en los siguientes apartados se clasificarán y detallarán las ventajas e inconvenientes de cada una de las tecnologías.

3.1.2 Ventajas e inconvenientes de los bridges

El éxito de los bridges, o puentes, Ethernet descrito en el capítulo primero tiene dos aspectos: el de transmisión y el de conmutación. Por una parte la eficiencia y economía de Ethernet como tecnología de transporte de datos en línea (capa física) y por otra el alto rendimiento y la configuración automática de los bridges Ethernet en el encaminamiento (*bridging*) de los datos entre segmentos. Todo ello consecuencia de la simplicidad de la trama Ethernet y de las economías de escala de Ethernet al alcanzarse volúmenes muy grandes de unidades fabricadas en poco tiempo.

Considerando la función de encaminamiento en las redes, las principales ventajas de los bridges respecto a los routers son las siguientes:

Autoconfiguración: Los bridges transparentes Ethernet no precisan configuración en su funcionamiento básico. Mediante el mecanismo de aprendizaje hacia atrás (*backward learning)*, aprenden el camino por donde encaminar las tramas, enviándolas por el puerto por donde se recibieron recientemente tramas con esa dirección origen. Aunque algunas funciones de los bridges actuales requieren configuración detallada, como es el caso de las redes locales virtuales (VLAN), el funcionamiento básico de un bridge no requiere configuración alguna.

Independencia del direccionamiento IP: Como se comentó respecto a la movilidad de hosts en el apartado anterior, no se requiere modificación alguna cuando un sistema final se mueve de una parte a otra de la red conmutada. Esta es una gran ventaja frente a los routers, que requieren configuración detallada de las rutas y gestión de las direcciones IP de los interfaces.

Por otra parte, los inconvenientes de los bridges son:

Desaprovechamiento de la infraestructura: Por definición, un árbol de expansión consta de un número de enlaces fijo e igual a *N-1* siendo *N* el número de nodos. Siendo *M* el número de enlaces existentes en la red, el número *D* de enlaces deshabilitados por el Protocolo de Árbol de Expansión es:

$$D = M - (N - 1)$$

La utilización de una red es el número de enlaces activos respecto al total de enlaces, viene dada por:

$$U = \frac{N - 1}{M}$$

Por lo que cuanto mayor sea el número de enlaces de la red para un número de nodos constante, o dicho de otro modo, cuanto mayor sea el grado medio (*M/N*) de los nodos de la red (número medio de enlaces por nodo), menor será el porcentaje de utilización de la red.

Tiempo de convergencia: Los protocolos del Árbol de Expansión (STP) convergen en 45-60 segundos, frente a los 5-10 segundos de los protocolos de encaminamiento estándar como OSPF. La razón de esta diferencia es que los bridges temporizan antes de habilitar los puertos para reenviar tramas con el fin de prevenir un colapso que podría crearse al habilitar puertos si existen aún bucles transitorios (esto se evita en los protocolos de encaminamiento con el campo TTL, del que carecen las tramas Ethernet, por no poder ser alteradas en los bridges). Dado que en muchos casos al bridge no le es posible detectar la finalización del algoritmo, es frecuente en los algoritmos de capa dos (como STP) esperar la convergencia del algoritmo mediante temporizadores que esperan un tiempo superior al caso peor de convergencia del algoritmo, lo que resulta excesivamente lento para los requisitos de las redes para recuperación rápida ante fallos de un elemento. Esta situación cambia con el protocolo de Árbol de Expansión Rápido (RSTP), el cual emplea otros mecanismos basados en decisión local entre bridges vecinos para la habilitación inmediata de puertos alternativos y presenta tiempos de convergencia del orden de décimas de segundo, salvo algunas situaciones específicas de caída del nodo raíz en que la realimentación de información obsoleta entre los puentes conectados a dicho nodo puede alargar la convergencia a decenas de segundos.

Caminos no mínimos: Como consecuencia de utilizarse un árbol único de difusión para el encaminamiento de las tramas por la red, los caminos entre los nodos no son mínimos, siendo solamente mínimos los caminos entre cada nodo y el bridge raíz, y los caminos entre los nodos de cada rama del árbol de expansión, no entre nodos situados en distinta rama del árbol de expansión, lo cuales pueden estar unidos mediante enlaces más directos, pero que son deshabilitados por el protocolo de Árbol de Expansión.

Falta de escalabilidad: Los bridges presentan falta de escalabilidad de las redes formadas por bridges exclusivamente formando un único dominio de difusión. Esta falta de escalabilidad tiene las siguientes causas:

- **Proliferación de direcciones MAC**: A medida que la red contiene más sistemas finales, los bridges tienen que aprender en las cachés todas las direcciones MAC de los sistemas activos, con lo que éstas se saturan, produciéndose inundaciones innecesarias de las tramas cuyo destino no figura en la caché del bridge.

- **Inundación de tráfico de difusión:** El tráfico de difusión, como en el caso de los mensajes ARP para resolución de direcciones, se distribuye por todo el dominio de difusión, a través del árbol de expansión hasta todos los sistemas finales. Esto supone una cierta capacidad de la red consumida con esta sobrecarga. Y supone también que todos los sistemas finales deben procesar estas tramas, consumiendo inútilmente capacidad de proceso del sistema final, porque solamente una mínima parte de estas tramas va realmente destinada al sistema final.

- **Efectividad del Árbol de Expansión:** El tamaño de una red de bridges está limitado porque el rendimiento del árbol de expansión disminuye al aumentar su tamaño: los caminos son proporcionalmente más largos y los temporizadores que determinan el tiempo de convergencia aumentarían si se sobrepasan las recomendaciones del estándar en cuanto a dimensiones de la red.

3.1.3 Ventajas e inconvenientes de los routers

En esta segunda parte se analizan las ventajas de los routers, o encaminadores, respecto a los bridges desde el punto de vista de los routers:

Aprovechamiento de la infraestructura: Los routers poseen ventajas innegables frente a los bridges. Estas ventajas son las del encaminamiento en capa tres respecto a los protocolos de capa 2, tales como el aprovechamiento de toda la infraestructura sin necesidad de deshabilitar enlaces. Los protocolos de encaminamiento utilizan toda la infraestructura mientras que los protocolos de capa 2 como los protocolos de Árbol de Expansión (STP y RSTP) deshabilitan enlaces para evitar la formación de bucles en la red, creando un árbol de expansión (*spanning tree)* para la difusión de las tramas, sobre el que opera el aprendizaje de direcciones MAC por los puertos de los bridges transparentes Ethernet *(backward learning)*.

Prevención de bucles: Como ya se conoce del apartado de *bridging* y *routing*, otra ventaja de los routers frente a los bridges es la prevención de bucles mediante el uso del campo TTL en los paquetes IP. Los routers soportan bien los bucles transitorios (inevitables en los algoritmos distribuidos), no solamente porque tienen un campo TTL, sino porque solamente encaminan en una dirección.

Difusión: El encaminamiento evita la difusión (*broadcast*), que es el punto débil de los bridges. Cuando un bridge no tiene en su caché la dirección MAC destino de una trama recibida, replica la trama en todos los puertos del bridge excepto el puerto por donde se recibió la trama. Esta *inundación* reduce la seguridad de la red al poder ser observadas tramas en todo el dominio conmutado. Aunque las redes locales

virtuales (VLAN) crean dominios de difusión separados, son solamente una solución parcial al problema del broadcast.

Tráfico multicast: Los routers procesan por defecto de forma más eficiente que los bridges el tráfico multicast, dado que el comportamiento por defecto de los bridges consiste en difundir indiscriminadamente (*broadcast*) el tráfico multicast recibido. Existen mecanismos para evitar la inundación como la inspección IGMP (*IGMP snooping*) en capa 2.

Rutas óptimas: Los routers permiten la optimización de las rutas o caminos en la red, ahorrando saltos innecesarios respecto a los caminos establecidos por el Árbol de Expansión.

Reparto de carga: Los protocolos de encaminamiento permiten implementar en capa 3 un reparto de carga de forma sencilla, flexible y potente, mientras en capa 2 son de implementación más compleja. La elección de caminos puede hacerse con criterios flexibles utilizando políticas, rutas estáticas, distancias administrativas, etc.

Direccionamiento jerárquico: Las direcciones IP permiten un direccionamiento jerarquizado, que permite agregar rutas. Las direcciones de capa dos (direcciones MAC), son planas y no indican posición topológica.

Compartimentación: Los routers permiten una compartimentación segura de la red en segmentos conmutados independientes. De esta forma un fallo en una LAN por mal funcionamiento o error de configuración no afecta a toda la red sino solamente a la red en fallo.

Por el contrario los principales inconvenientes de los routers son:

Necesidad de configuración y administración IP: Los routers precisan configuración de rutas y administración de direcciones IP. Además, si hay cambios de topología en la red o actualizaciones de equipos o enlaces esta configuración debe ser modificada.

Coste superior: Como también se conoce del apartado de *bridging* y *routing*, el coste es un factor determinante y, en este caso, los costes por puerto de un router convencional son alrededor de diez veces superiores a los de un bridge, lo que es una clara desventaja.

3.1.4 Problemas a resolver

Pese a la gran cantidad de ventajas de los routers, es posible superar los inconvenientes de los bridges en base a un diseño más apropiado, intentando asemejarse a los routers (como demuestran SPB y TRILL). Sin embargo, no es posible evitar la necesidad de configuración y administración IP, ni el coste de los routers, lo que son dos grandes lacras del *routing*. Es por esto que el *switching* se presenta como una gran alternativa a explorar y en la que se trabaja en la presente Tesis.

Consecuentemente con el planteamiento del problema, los problemas principales a resolver, vistos desde el lado de los bridges, ya se han descrito y coinciden con los inconvenientes de los bridges (que a su vez son parte de las ventajas de los routers frente a estos): desaprovechamiento de la infraestructura, tiempo de convergencia, caminos no mínimos y falta de escalabilidad. Los tres primeros problemas apuntan a utilizar un encaminamiento alternativo al árbol de expansión clásico de los bridges, que podría optimizar o mejorar todos ellos. La falta de escalabilidad, sin embargo, es consecuencia de varios problemas como se ha visto y aparece cuando se trata de aumentar el tamaño de los dominios conmutados.

3.2 Análisis del espacio de soluciones

Una vez revisado el estado del arte se procede a su valoración y al análisis del espacio de soluciones para continuar después con la descripción de las decisiones de diseño.

Los objetivos establecidos en la definición del problema consisten básicamente en combinar las ventajas de los bridges (siendo la autoconfiguración el punto positivo más importante) con las ventajas de los routers. Se trata pues de mejorar las prestaciones de los bridges en eficacia de utilización de la infraestructura, convergencia rápida, caminos óptimos en la red y aumentar su escalabilidad. Como ya se comentó previamente, los tres primeros problemas se podrían solucionar con un método de encaminamiento alternativo al árbol de expansión clásico. De acuerdo con lo expuesto en el estado del arte, los planteamientos más actualizados para superar estos problemas pueden agruparse en las siguientes categorías:

- Protocolos de encaminamiento basados en el uso de árboles de expansión múltiples (MSTP). Incluso evolucionados, de manera que se creen caminos mínimos entre todos los nodos de la topología y en base a un protocolo de estado de enlace, simplificando la configuración: **SPB**.

- Protocolos de encaminamiento sobre direcciones de capa dos: **RBridges** (**TRILL**).

- Protocolos de encaminamiento específicos para topologías concretas de centros de datos, en los que se pueden aplicar la jerarquización de nodos, como son **VL2** y **PortLand**.

Cada una de estas categorías presenta ventajas e inconvenientes propios:

- Como principal inconveniente de SPB tenemos la dependencia del uso de un protocolo de estado de enlace, que facilita mucho la configuración respecto a su predecesor, MSTP, pero desvirtúa la esencia del *switching* (autoconfiguración), requiere un procesado mayor en los switches, además de seguir necesitando cierta parte de configuración. Como ventajas, SPB permite una utilización mucho mayor de la infraestructura, la reconfiguración que posee es rápida y sus caminos son mínimos

(aunque quizás en ciertas circunstancias de la red, no óptimos, pero sí mucho mejores que los generados por el árbol clásico de expansión).

- En el caso de los RBridges, el inconveniente es el volumen de procesado requerido por bridge, lo que puede limitar la escalabilidad en redes muy grandes (algo similar a lo que ocurre en SPB). Las ventajas son la relativa rapidez de convergencia, así como un mayor aprovechamiento de la infraestructura.

- Finalmente, protocolos que añaden una arquitectura como es el caso de VL2 y PortLand para el caso concreto de los centros de datos, mejoran el aprovechamiento de la infraestructura (no sólo por el protocolo, sino también por el diseño de la arquitectura) y aumentan la escalabilidad al usar direccionamiento jerárquico. Sin embargo, necesitan un gestor, agente o directorio que realice los mapeos de direcciones, así como el control de otras funciones. En el caso de PortLand es centralizado y puede suponer incluso un cuello de botella, mientras que en VL2 no lo es, pero por ello necesita un cierto número de servidores sincronizados, así como otros que funcionen como respaldo de los principales.

3.3 Proceso de diseño

En esta sección se detallará brevemente el proceso de diseño seguido, así como las consideraciones realizadas y decisiones tomadas. Primero se hablará de las implicaciones que conlleva aprovechar las ventajas de los bridges al máximo (encaminamiento y autoconfiguración en segmento único IP), y a continuación se detallarán los requisitos para añadir las ventajas de los routers (aprovechamiento de la infraestructura, convergencia rápida, caminos óptimos en la red y escalabilidad).

3.3.1 Encaminamiento y autoconfiguración en segmento único IP

Como decisión inicial de diseño, incluida en los requisitos, se asume un segmento único o subred IP en toda la red que evite la necesidad de cambio de dirección IP cuando un sistema final se traslada de un punto a otro dentro de dicha red. Para ello es necesaria la eliminación de los routers IP salvo en los bordes de la topología.

El problema planteado consiste en encaminar en capa dos entre sistemas finales conectados en un segmento único IP. Para realizar el encaminamiento se precisa un sistema de direccionamiento en capa dos. El encaminamiento basado en direcciones MAC estándar presenta los inconvenientes de no agregabilidad y de que la dirección MAC es un identificador y no un localizador *(locator)*, no tiene contenido topológico,

por lo que no es agregable. Para mantener la independencia entre capas y del protocolo de red, tampoco es aconsejable el empleo de información de IP para encaminar en capa dos, salvo optimizaciones opcionales. Además deben contemplarse los casos de direcciones IP fijas. Se excluye pues el uso de direcciones IP.

En el caso de la configuración, el problema se centra en la necesidad de configurar las direcciones IP y diseñar las subredes IP dentro de la red. Cuando un sistema final se mueve de un punto a otro de la red campus, su dirección IP deberá modificarse con arreglo a la nueva subred a la que pertenece. Para evitar este problema, se parte de suponer una subred IP única en la red, que independice la dirección IP del punto de conexión en la red.

3.3.2 Aprovechamiento de la infraestructura

El primer requisito de nuestro diseño, que toma como base una de las ventajas de los routers, es el aprovechamiento de la infraestructura. Para ello, el primer paso sería eliminar el bloqueo de enlaces tal y como hace el árbol de expansión clásico. El segundo sería no sólo no bloquear, sino utilizar dichos enlaces. El tercer y más estricto requisito, al que se debería tender por ser la solución óptima, sería que los enlaces realizasen un buen reparto de carga, de manera que el ancho de banda de bisección (*bisection bandwidth*) efectivo de la topología fuese el máximo que dicha arquitectura pudiese dar físicamente. Para maximizar este valor, una variable que actúa, es la arquitectura de la topología, como se acaba de comentar, pero una vez fijada una arquitectura, aunque no aproveche la capacidad de los enlaces de la topología al máximo, el reparto de carga realizado por el encaminamiento debería al menos alcanzar el máximo que permite dicha arquitectura. Es importante destacar que el encaminamiento por caminos mínimos no es óptimo necesariamente en reparto de carga al centrarse en caminos mínimos exclusivamente y no tener en cuenta la matriz de tráfico.

3.3.3 Convergencia rápida del encaminamiento

Un requisito no menos importante es la necesidad de una convergencia rápida del encaminamiento ante alteraciones de la red por caída de nodos o de enlaces. Un argumento habitual contra RSTP es que el tiempo de convergencia tras reconfiguración es excesivamente largo, lo que es cierto comparado con los equipos SDH, pero no respecto a los algoritmos de encaminamiento de estado de enlaces como Djikstra.

En el artículo [Fra+09] se presenta un estudio detallado de los diferentes factores que afectan a la convergencia de protocolos basados en estado de enlace en redes ISP de gran tamaño. Básicamente dice que el tiempo de convergencia se puede caracterizar como $D + O + F + SPT + RIB + DD$, donde D (*detection time*) es el tiempo de detección de fallo del enlace, O (*origination time*) es el tiempo que se tarda en generar los nuevos mensajes de estado describiendo la nueva topología (tras el fallo

de enlace), *F* (*flooding time*) es el tiempo completo de inundación de la red desde el nodo que detecta el fallo, SPT (*shortest-path tree time*) es el tiempo es calcular los nuevos árboles de caminos mínimos, RIB (*RIB time*) es el tiempo necesario para actualizar las tablas de encaminamiento (basadas en la información aportada por la RIB, *Routing Information Base*, y la FIB, *Forwarding Information Base*) y DD (*distribution time*), que es el tiempo empleado en distribuir la información de las actualizaciones de las tablas de encaminamiento en el caso de una arquitectura distribuida.

Analizando estos parámetros, en el artículo llegan a la conclusión de que los tiempos D, O y DD son pequeños en comparación con el resto. Mientras que F dependerá de la topología de la red (y por lo tanto de los retardos de propagación de enlace), y el tiempo SPT dependerá del número de nodos en la red (pero puede ser reducido significativamente usando un cálculo incremental). Así pues, el RIB es el factor más significativo ya que dependerá linealmente del número de prefijos (en el caso de routers) afectados por el cambio.

En el caso concreto de nuestro estudio, es posible ver que estos parámetros están directamente relacionados con el uso de protocolos de estado de enlace (necesarios en SPB y TRILL), por lo que quizás un buen punto de partida sería trabajar sin el uso de estos protocolos, para evitar así posibles problemas de convergencia.

3.3.4 Caminos óptimos

Además del aprovechamiento de la infraestructura comentado previamente, otro requisito sería obtener caminos mínimos en las comunicaciones entre nodos.

En el caso de redes empresariales, SPB y TRILL ya nos dan esta posibilidad, en base al uso de un protocolo de estado de enlace. Así pues, en nuestro planteamiento se busca conseguir caminos óptimos, pero eliminando el uso de dicho protocolo, para evitar dicha complejidad, carga y procesamiento extra en la red.

En el caso de redes de centros de datos, escogeremos inicialmente una arquitectura de red, concretamente la del falso *fat tree* extensible definido en [Vah+10], para luego hacer una propuesta de encaminamiento con caminos mínimos en base a dicha topología y estudiar si es generalizable a topologías similares de centros de datos. Como se puede suponer, los caminos mínimos en este tipo de redes están fijados por la arquitectura: por ejemplo, este *fat tree* tiene un camino mínimo de 5 nodos (o 6 enlaces) si la comunicación es entre nodos de diferentes *pods*, o menor si comparten *pods* o incluso switch frontera. Así pues, el objetivo consiste en no superar dichos valores establecidos.

3.3.5 Escalabilidad

Los requisitos de escalabilidad se centran primero en una solución sin protocolo de estado de enlace, como son las opciones SPB y TRILL, que como ya se comentaba previamente, limitaba la escalabilidad. Mientras que para el caso de los centros de datos, se pretende realizar una propuesta distribuida, que no necesite gestores o directorios como en VL2 y PortLand, que en la práctica suponen un problema adicional e incluso, un cuello de botella en algunos casos.

Además, se deberían evitar en la medida de lo posible los siguientes problemas ya comentados:

- **Proliferación de direcciones MAC**: Para evitar que los bridges llenen sus cachés, éstas podrían generarse sólo puntualmente cuando la comunicación fuera necesaria y caducar después. También podrían utilizarse técnicas similares a las de SPBM y SPBV, en las que los bridges sólo necesiten aprender las direcciones de los bridges frontera, de manera que se reduce el número de direcciones totales a aprender.

- **Inundación de tráfico de difusión:** Para evitar esta inundación, una idea sería el uso de proxies. Otra idea podría llevar a cambiar el mecanismo básico de ARP con difusión, pero éste debe mantenerse por defecto por ser el único que garantiza la resolución de la dirección en todos los casos, incluido el caso de desplazamiento silencioso de un sistema final. Para el problema de la difusión de ARP, la exploración del estado del arte da los conocidos resultados de uso de *proxies* ARP, es el caso por ejemplo de EtherProxy [EC09].

- **Efectividad del Árbol de Expansión:** Evitando el uso de árboles de expansión, eliminaremos este problema.

3.4 Conclusiones

En este capítulo se ha definido el problema a resolver, así como los problemas parciales a resolver. Se ha realizado la valoración del estado del arte de los protocolos de encaminamiento en capa 2 con vistas a su aplicación en el problema planteado.

Finalmente se ha expuesto el proceso de diseño y las decisiones tomadas para definir el desarrollo de la Tesis: lo principal es eliminar las limitaciones del protocolo de árbol de expansión, pero también se pretende obtener las ventajas de los routers en base a planteamientos menos complejos que SPB y TRILL en redes empresariales (en base a eliminar el uso de un protocolo de estado de enlace) y que VL2 y PortLand en redes de centros de datos (haciendo el encaminamiento distribuido, sin gestores, ni directorios).

Capítulo 4

Contribuciones al encaminamiento en redes empresariales

En los siguientes apartados se describirán las diferentes contribuciones realizadas en la presente Tesis respecto al ámbito del encaminamiento en redes empresariales.

Para ello se comienza con ARP-Path, el primer protocolo de la que luego se denominaría familia All-Path, categoría de bridges llamados así porque se exploran todos los caminos simultáneamente en la red mediante una trama broadcast que recorre todos los enlaces de la red para encontrar un camino o establecer un árbol multicast.

Sobre ARP-Path se profundizará en su evolución, los problemas que fueron surgiendo, tanto meramente teóricos como a la hora de realizar implementaciones del protocolo (véase Apéndice E para las herramientas utilizadas), y las decisiones que se fueron tomando hasta llegar a su versión actual. Sin embargo, en el camino de dicho análisis aparecieron otras ramas de la familia All-Path como son Flow-Path (una variante de ARP-Path que no aprende por dirección origen, sino por flujo), ARP-Path asterisco (versión que fusiona el concepto de ARP-Path con Flow-Path y lo

lleva un paso más allá) o ARP-Path multipath (ARP-Path con opción de multicaminos para distribuir la carga). Todos estos protocolos se describirán en los próximos apartados, incluyendo las distintas implementaciones y el análisis de las características de cada uno, sobre todo en comparación de unos protocolos con otros, dentro de la misma familia, sus ventajas e inconvenientes.

4.1 ARP-Path

ARP-Path es el nombre otorgado al protocolo de encaminamiento en redes *bridging* desarrollado por el grupo GIST-NETSERV, dedicado a la investigación de servicios de red dentro del Grupo de Ingeniería de Servicios Telemáticos de la Universidad de Alcalá.

En la primera etapa de su desarrollo era conocido como FastPath [Iba+10a], nombre que fue finalmente descartado por coincidir con otra (de hecho, más concretamente, otras) tecnología preexistente. La elección del segundo (y actual) nombre, ARP-Path [Iba+11a], es debida a que este protocolo crea caminos (*paths*) en base a aprendizaje basado exclusivamente en los mensajes ARP (*Address Resolution Protocol*) intercambiados en la red, que es más específico que el nombre inicial.

Para tener una primera idea del funcionamiento de ARP-Path, basta con echar un vistazo a los dos artículos ya referenciados. Sin embargo, en este apartado, se detallarán más específicamente la evolución y las decisiones tomadas a la hora de desarrollar ARP-Path, hasta describir su estado actual.

4.1.1 Base: ARP-Path (FastPath)

Al comenzar la Tesis, el protocolo ARP-Path se encontraba en su primera fase, denominándose aún FastPath. Este protocolo crea caminos entre pares de hosts en una red de bridges en base a la creación de tablas de encaminamiento que son aprendidas en cada puente única y exclusivamente con los mensajes ARP (es decir, el *ARP Request* y su respuesta, *ARP Reply*), aunque en esta primera propuesta la idea quedaba abierta al aprendizaje con cualquier trama broadcast.

La razón de usar los mensajes ARP se basa en el hecho de que la utilización de IP en la capa 3 produce que se resuelvan las direcciones de los hosts previamente a toda comunicación, es decir, siempre se intercambiarán un par de mensajes ARP antes de cualquier mensaje de datos. Dado que la funcionalidad del ARP es "descubrir" en cierto modo a los hosts, se pensó adecuado dotarle de una funcionalidad extra generando, ya de paso, las rutas entre dichos hosts. Por supuesto, se ha tomado ARP al tratarse de IPv4, pero en el caso de IPv6 el equivalente sería utilizar los mensajes NDP (*Neighbor Discovery Protocol*).

Así pues, el funcionamiento de ARP-Path en su desarrollo inicial era y se dividía en las dos siguientes fases:

Descubrimiento del camino (*ARP Request*): El mecanismo para establecer caminos de baja latencia (*fast paths*) se inspira parcialmente en el *Reverse Path Forwarding* [DM78]. Un *fast path* es el camino más rápido (y por tanto, único) creado por la primera copia de un ARP Request que alcanza el host destino. El proceso, que se describe en la próxima figura, funciona de la siguiente manera:

Cuando el host origen A quiere comunicarse con otro destino C, primero manda un ARP Request encapsulado en una trama broadcast para resolver la dirección IP del destino C. El puente frontera de A, llamado 1, recibe la trama de A y asocia la dirección MAC global de A al puerto a través del que recibe el mensaje, cerrando (pasando su estado a *locked*) temporalmente el aprendizaje de la dirección A en este puerto y bloqueando el resto de puertos del bridge 1 de manera que no aprendan, ni reenvíen, más tramas broadcast con dirección origen A. Por lo tanto, las tramas broadcast con dirección origen A que lleguen a cualquier otro puerto del bridge será descartadas y consideradas tramas que llegan tarde (*late frames*).

key(MAC)	port	timer	state
A	1	0	*Locked*

Tabla 1: Ejemplo de entrada en la tabla de direccionamiento de un switch en la fase de descubrimiento del camino de ARP-Path

Después de recibir la trama, el bridge 1 la reenviará por todos sus puertos excepto por aquel por el que la recibió. Los puentes 2 y 4, que recibirán dicha trama de parte de 1, se comportarán de manera idéntica a 1, haciendo *lock* de la dirección A en el puerto que primero recibe la trama y reenviando por todos los puertos excepto por el cual la recibieron. Esto hará que se envíen tramas entre ellos, que serán copias tardías (*late frames*) y serán descartadas, tal y como se ve en la siguiente figura. Lo mismo ocurrirá esta vez con los puentes 3, 5 y 6, que recibirán la trama de 2 y 4. Finalmente el host destino C recibirá la trama.

Por lo tanto, la asociación temporal (*locking*) de la dirección A en un puerto de cada bridge se propaga a través de la red como un árbol enraizado en el host A, y ahora existe una cadena activa de bridges entre A y C con un puerto de entrada asociado a la dirección A.

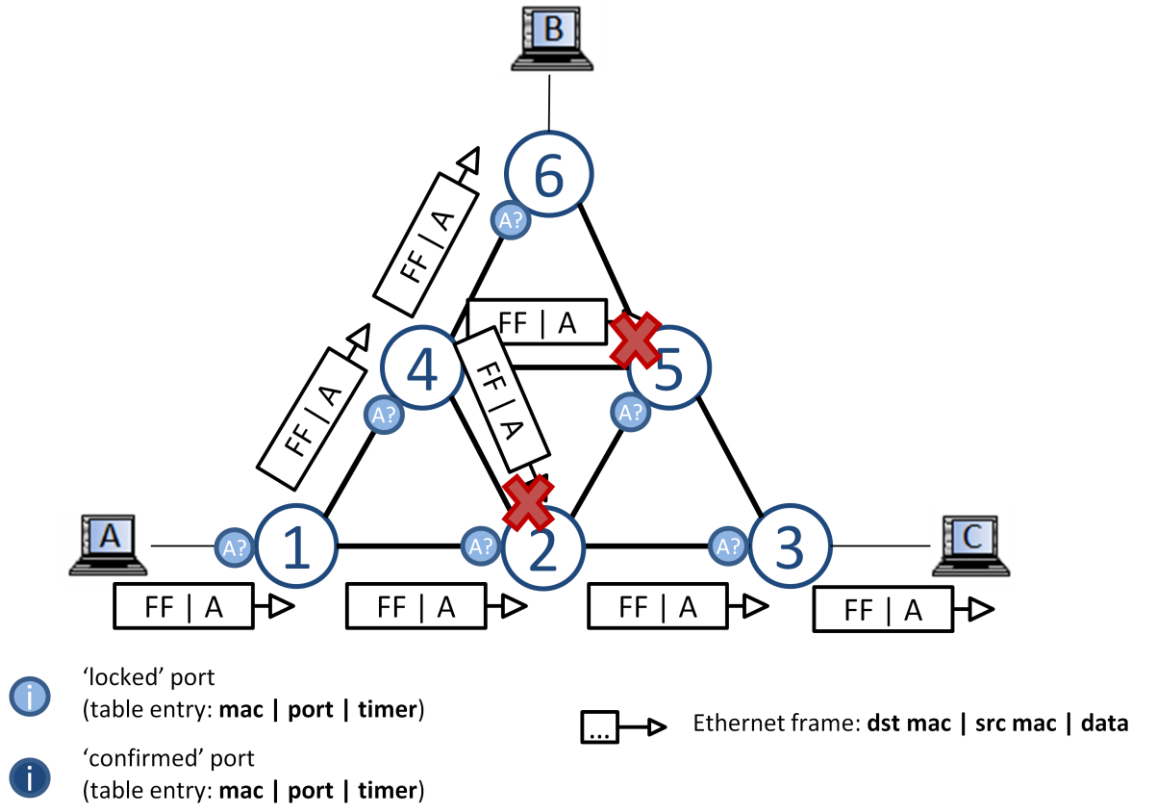


Figura 36. Aprendizaje de la dirección del host origen (A) mediante el mensaje ARP Request en ARP-Path (FastPath). Las tramas con una cruz en rojo son algunas de las bloqueadas por ser tardías (*late frames*) y ya estar aprendidas en otro puerto (el más rápido)

Confirmación del camino (*ARP Reply*): La confirmación del camino se realiza con la respuesta que da C al ARP Request, que es un mensaje ARP Reply, y en sentido inverso.

Así pues, cuando el puente frontera de C, denominado 3, recibe el mensaje unicast ARP Reply, éste asocia la dirección de C al puerto de entrada y a su vez, si la dirección destino (que ahora es A) ya está en estado *locked* (como es el caso), confirmará el aprendizaje tanto de C, como de A, cada uno en su puerto. Después, al ser una trama unicast, ésta no se reenviará por todos sus puertos menos el de entrada (como sucedía en el caso del ARP Request), sino que buscará la dirección destino entre las direcciones aprendidas en 3 y reenviará la trama por el puerto asociado a A.

key(MAC)	port	timer	state
A	1	0	*Confirmed*
C	2	0	*Confirmed*

Tabla 2: Ejemplo de entrada en la tabla de direccionamiento de un switch en la fase de confirmación del camino de ARP-Path

A continuación, la trama llegará al bridge 2, que de nuevo asociará la dirección de C al puerto de entrada y la reenviará por el puerto asociado a A, confirmando una vez más el aprendizaje de los puertos tanto para C como para A. Y lo mismo sucederá con el bridge 1, el que enviará finalmente la trama al destino A, que, tras haber recibido la respuesta a su solicitud ARP, comenzará la comunicación y envío de datos que quería realizar.

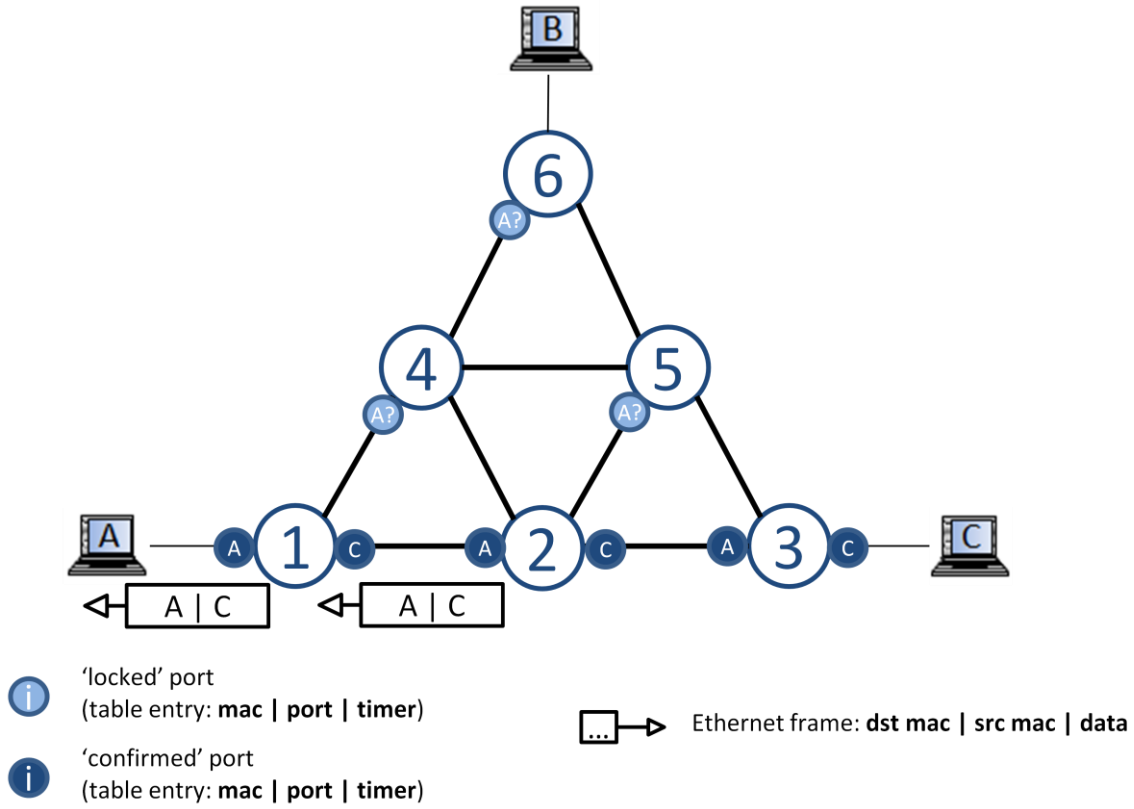


Figura 37. Confirmación de la dirección del host origen (A) y host destino (C) mediante el mensaje ARP Reply, creando un camino entre A y C, en ARP-Path (FastPath)

El mecanismo de confirmación inicialmente genera un camino que es simétrico, es decir, que coincide en ambas direcciones A-C y C-A.

Los **estados** de '*locked*' (asociación temporal) y '*confirmed*' (aprendizaje confirmado) de cada una de las fases tienen un **temporizador** (timer), que caducará en caso de no ser refrescado. La misión del estado '*locked*' es evitar bucles que podrían crearse por las tramas que se envían replicadas por todos los puertos y el valor de su timer es pequeño (< 1 segundo). Mientras que el estado '*confirmed*' muestra un camino aprendido y su misión es encaminar tramas, por lo que su timer será mayor (de varios minutos). El **refresco** de dichos timers se realiza de la siguiente forma:

- Si el estado de la entrada de A es '*locked*', éste se refresca con cualquier trama ARP que tenga como origen A y que llegue por el puerto asociado.

- Si el estado de la entrada de A es '*confirmed*', éste se refresca con cualquier trama de datos que tenga como origen A y que llegue por el puerto asociado.

En el caso de que se emitan dos ARPs simultáneos en direcciones opuestas (de A a C y de C a A, a la vez) y que además escojan dos caminos diferentes, cosa que es poco probable pero posible, se utilizarán mecanismos de prioridad basados para prevenir crear más de un camino. En este caso se escogió un mecanismo basado en la dirección MAC que, si existía un camino ya inicial, sólo confirmaba aquel creado por la trama ARP Reply cuya dirección origen fuese mayor que la dirección destino. De esto último se hablará más detalladamente en apartados posteriores, dado que fue una de las razones que propiciaron la evolución del protocolo, al mostrar problemas.

De los mecanismos de reparación hablaremos en un apartado dedicado específico.

4.1.1.1 Implementación en OMNeT++, Linux y OpenFlow/NetFPGA

Al comenzar la Tesis, en el año 2010, la recién detallada versión de ARP-Path era la que se encontraba desarrollada, así como implementada en tres plataformas por tres diferentes autores.

Implementación en OMNeT++

La primera de ellas estaba realizada sobre el simulador OMNeT++ (versión 4) y la librería, o framework, inet (versión 1), que ya son conocidas del capítulo dedicado al estado del arte. La implementación estaba basada en el módulo *MACRelayUnitNP*, que se encuentra dentro de la sección dedicada a Ethernet *linklayer/etherswitch/* de la librería inet (en dicha versión 1) y, como su nombre indica, implementa la *relay unit* de un switch Ethernet.

El módulo *MACRelayUnitNP* está formado por tres archivos, un *.h con la declaración de variables y funciones, un *.c con las definiciones y finalmente un *.ned que define al módulo como módulo simple dentro del entorno de OMNeT++. Así pues, se creó una nueva carpeta *linklayer/ARPPathSwitch/* y dentro se copiaron estos tres ficheros renombrados a *MACRelayUnitAPB* (APB = ARP-Path Bridges). Cada vez que llega un mensaje al módulo dentro de la ejecución de una simulación, en este módulo se llama a la función `handleMessage`, que mira si es un mensaje interno del simulador (automensaje) o una trama (mensaje de la red) y, en caso de ser una trama, llama a `processFrame`. Así pues, lo que se hizo es añadir la función `handleAndDispatchFrame` y llamarla dentro de `processFrame`.

```
void MACRelayUnitAPB::handleAndDispatchFrame(EtherFrame
*frame, int inputport)
```

Tabla 3: Declaración de la función que contiene la lógica de ARP-Path en la implementación de OMNeT++

Esta función contiene toda la lógica del protocolo ARP-Path y para ello se le pasan como parámetros la trama Ethernet (`EtherFrame *frame`) y el puerto de

entrada (`int inputport`). Y con estos dos parámetros de entrada, la lógica aplicada es la siguiente:

- Si se recibe una trama ARP Request, se comprobará si existe una entrada para ese origen en la tabla de direcciones. Si es así, se comprobará si el puerto donde se ha recibido la trama es el mismo que se tiene indicado en la tabla. Si el puerto coincide, se refrescará el temporizador de la entrada, si no, la trama será descartada. Por otro lado, si no existe la entrada, se añadirá una, indicando en la tabla la MAC origen del ARP, el puerto de entrada, el estado ('*locked*') y el tiempo de introducción de la entrada.

- Si lo que se recibe es una trama ARP Reply, se comprobará si existe una entrada en la tabla de direcciones para el destino indicado por la dirección MAC de la misma. Si es así, se comprobará el estado de la misma, ya que si está bloqueada, se deberá confirmar, y además, añadir una nueva entrada para el origen del ARP Reply, que pasará directamente a estado confirmado. Si ya está confirmada, se debe refrescar la tabla previamente. Es posible, sin embargo, que las tramas de tipo ARP Reply lleguen por un puerto distinto a uno que se ha confirmado. En ese caso, es necesario aplicar un sistema de prioridad para indicar en qué casos se debe descartar la trama y en qué casos se debe cambiar el contenido de la tabla de direcciones. Este sistema de prioridades es necesario en el caso de dos hosts que comiencen el descubrimiento del camino al contrario en paralelo (ARPs simultáneos) y elijan dos caminos diferentes.

- Finalmente, si es otro tipo de trama, se enviará por uno o más puertos correspondientes según si es broadcast o unicast.

Además de añadir esta función con la lógica de ARP-Path, se implementa una nueva tabla de direccionamiento dentro del módulo, denominada `addressTable`. La tabla de direcciones está implementada mediante la estructura `typedef std::map` definida a partir de la librería std de C++. Este tipo de estructura permite la búsqueda de los elementos de la tabla a partir de un elemento clave único en cada entrada, en este caso la dirección MAC (`MACAddress`), haciendo que la búsqueda en la tabla sea mucho más eficiente. Cada entrada de la tabla `AddressEntry` es, por tanto, una estructura formada por un número de puerto, un tiempo de generación de la entrada, y un estado ('*locked*', que en el código se define como '*blocked*' o '*confirmed*', definida como '*learnt*'), accedida por la dirección MAC correspondiente. Es posible observar en la siguiente tabla, que contiene el código que define la tabla de direcciones, que las entradas también contienen un campo para la dirección IP asociada a la dirección MAC. Ésta se añadió para ser usada en el caso de implementar la funcionalidad de EtherProxy, pero no es necesaria para la lógica del protocolo en sí.

```
struct AddressEntry
{
  int port;            // Puerto de entrada
  simtime_t inTime;    // Tiempo de generación
  char* status;        // Estado (locked/learnt)
  IPvXAddress ip;      // Dirección IP (EtherProxy)
};

struct MAC_compare
{
  bool operator()
     (const MACAddress& u1, const MACAddress& u2) const
  {return u1.compareTo(u2) < 0;}
};

typedef std::map<MACAddress, AddressEntry, MAC_compare>
AddressTable;

AddressTable addressTable;
```

Tabla 4: Definición de la tabla de direccionamiento de ARP-Path en la implementación de OMNeT++

En la siguiente figura se muestra un ejemplo de esta tabla con dos direcciones aprendidas, dentro de la ejecución de una simulación en OMNeT++.

```
Address Table (switch10) (2 entries):
  ---------------------------------------------------------------
  | Address           | Port | Status    | IP                   |
  ---------------------------------------------------------------
  | 00-00-00-00-01-01 |  1   | blocked   | <unspec>             |
  ---------------------------------------------------------------
  | 00-00-00-00-01-09 |  1   | blocked   | <unspec>             |
  ---------------------------------------------------------------
```

Figura 38. Tabla de direcciones de una simulación de ARP-Path en la implementación de OMNeT++

Como ya se conoce, las entradas de la tabla caducan con un temporizador. En la implementación de ARP-Path en OMNeT++, en vez de generar un temporizador que genere un evento de borrado para cada entrada cuando se cumpla el tiempo X (siendo X la duración del timer), lo que se realizó en la práctica fue un borrado '*offline*'. Es decir, cuando se genera una entrada (o se refresca), se apunta el tiempo de simulación actual y, cada vez que una nueva trama llega al puente que contiene dicha entrada, se comprueba si la suma del tiempo de las entradas con el tiempo X, definido para el temporizador, es mayor que el tiempo actual de simulación. Si ocurre así, se elimina la entrada de la tabla, pues quiere decir que está obsoleta. Éste método es más sencillo y ahorra más recursos que generar eventos de temporización para cada entrada, y como se puede apreciar, mientras no lleguen tramas a un puente, éste puede tener entradas ya obsoletas, pero en la práctica eso no molesta, pues en cuanto se utilice el puente y llegue una trama se actualizará toda la tabla.

Para más información sobre esta implementación, se recomiendo la lectura de [RR10].

Implementación en Linux

La primera prueba de concepto del protocolo ARP-Path en una plataforma "real" (es decir, fuera de un simulador) fue la implementación del protocolo en Linux con versión de kernel 2.6, que operaba en espacio de kernel y de usuario en base a la utilización de ebtables [ebtables]. La decisión fue la de usar la funcionalidad ya existente en el kernel sin añadir cambios, lo que hace más fácil la depuración de la aplicación y su ejecución en cualquier distribución de Linux relativamente reciente. Para la ejecución de esta plataforma es necesario:

- *brctl*: Herramienta para configurar el bridge
- *ifconfig*: Herramienta para activar el puente (sus interfaces) ya configurado
- *ebtables*: Herramienta que realiza el filtrado actuando como puente Ethernet
- Un kernel Linux compatible y con soporte de *bridging* y *ebtables*. En nuestro caso se utilizó el kernel 2.6.31-14 (sistema operativo Linux Ubuntu)

La decisión de reenviar o no un paquete desde un puerto del bridge a otro se realiza usando las reglas de filtrado de *ebtables*. El programa *ebtables* se utiliza para configurar el filtro de paquetes de bridge Ethernet que se ejecutará en el kernel y las reglas de filtrado de *ebtables* se aplican por un programa situado en espacio de usuario que inspecciona todo el tráfico relevante. Para copiar los paquetes desde el kernel al espacio de usuario, se utiliza el *ulog target* de ebtables, que a su vez hace uso de la infraestructura *netlink* como canal de comunicación. El tráfico excesivo fácilmente sobrecargaría la comunicación *netlink* y eso generaría mensajes del tipo "*recvmsg: No buffer space available*" (búffer desbordado), sin posibilidad de solucionarse configurando el valor de memoria en */proc/sys/net/core/rmem_max*. Por lo que para reducir el impacto de este cuello de botella todo lo posible, las reglas de *ebtables* fueron adaptadas de manera que el programa en espacio de usuario sólo recibiera mensaje que no podían ser procesados por las reglas activas de *ebtables*, por ejemplo, todo el tráfico de caminos confirmados se procesa con las reglas de *ebtables*, sin más inspección por parte de espacio de usuario.

La detección de fallo de enlace está implementada en un hilo aparte que escucha en un socket *NETLINK_ROUTE* de la familia de direcciones *NETLINK*. Cualquier cambio en el estado de la interfaz de un enlace genera eventos, en base a mensajes netlink del tipo *RTM_NEWLINK* y *RTM_DELLINK*.

Para probar la implementación se eligieron tres placas Soekris de 500 Mhz con enlaces a 100 Mbps por su portabilidad, pues en la práctica eran pequeñas cajas que se asemejaban a pequeños switches reales (y no un PC completo que ocupa mucho más). Con esta implementación de Linux/Soekris (Linux sobre Soekris) se construyó

una topología en triángulo (al ser tres cajas) y se probaron las siguientes pruebas (todas ellas satisfactorias): ping entre diferentes hosts, emisión de vídeo con VLC en UDP y HTTP y conectividad a un servidor externo de Internet, todo ello probando los casos en los que falla un enlace y se requiere reparación. El retardo de envío o *forwarding* de esta plataforma rondaba los 0,9 milisegundos. En las siguientes figuras se pueden ver un esquema y una foto de dichas pruebas.

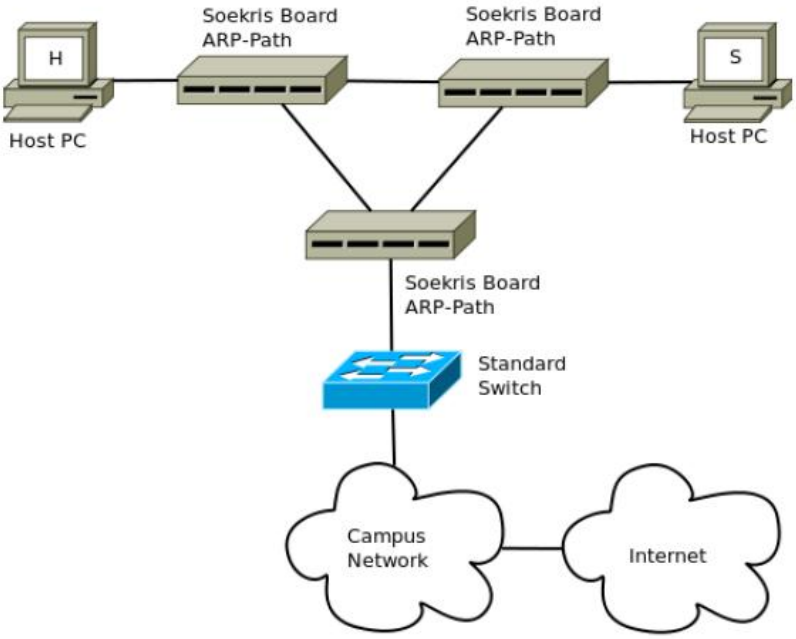


Figura 39. Esquema utilizado para probar la implementación de ARP-Path en Linux/Soekris

Figura 40. Foto de una de las pruebas en marcha de la implementación de ARP-Path en Linux/Soekris

Esta implementación tenía ciertos problemas, principalmente derivados del hecho de necesitar el espacio de usuario para procesar ciertos paquetes y de tener un búffer de envío entre kernel y usuario. En cuanto la frecuencia de datos aumentaba (por ejemplo, en el caso de vídeo emitido en UDP con el programa VideoLan VLC [VLC] de un origen a un destino) el búffer trabajaba demasiado, lo que llevaba a descartar ciertos paquetes y que la comunicación no fuera óptima.

Para más información sobre esta implementación, se recomiendo la lectura de [Iba+10b] y [Iba+11b].

Implementación en OpenFlow/NetFPGA

Tras la implementación de ARP-Path en Linux, el objetivo marcado fue la implementación con enlaces de velocidad a 1 Gbps. La plataforma más adecuada en este caso sería una implementación pura en NetFPGA, pero dada su alta curva inicial de aprendizaje (programación de cero de toda la lógica del protocolo en Verilog), se decidió realizar la implementación de un controlador OpenFlow con la lógica de ARP-Path y utilizar las tarjetas NetFPGA como switches OpenFlow, dado que era un proyecto ya existente y depurado en la comunidad NetFPGA, y además el switch OpenFlow no comercial que mejores prestaciones tenía (y que sigue teniendo actualmente). Como controlador OpenFlow se escogió NOX (actualmente NOX Classic, dada la evolución de esta herramienta a POX), que permite el desarrollo del controlador en C++ y Python.

El primer paso que se dio, previo a la adquisición de las tarjetas NetFPGA, para ver que todo funcionaba correctamente, fue la implementación de la lógica de ARP-Path en NOX y la prueba de dicho controlador sobre switches virtualizados gracias a la utilización de Mininet. Una vez verificado, no haría falta ningún cambio en el controlador, gracias a la independencia de plataformas que proporciona OpenFlow.

El módulo ARP-Path en NOX, desarrollo exclusivamente en código Python, consiste en varios ficheros, pero sólo uno define la lógica del protocolo, llamado "`modswitch.py`". Este archivo describe, como el resto de los módulos de NOX, dos funciones importantes: "`__init__`", que inicializa el componente, las variables de entorno y las trazas (para poder analizar los resultados después), e "`install`", que básicamente lo que hace es registrar eventos del switch en el controlador (por ejemplo, el evento de recibir una trama en el switch, de escuchar cambios en el estado de la interfaz de un enlace, etc). En este caso, el evento más importante es el evento registrado para paquetes recibidos desconocidos y para ello se registra la llamada a la función "`handle_packet_in`" para cada paquete cuya dirección sea desconocida en el switch OpenFlow. Dentro de esta función, que recibe como parámetros de entrada la ID del switch OpenFlow que recibió el paquete (recordemos que el controlador es centralizado y manejará de todos los switches en la red a la vez), el paquete en sí y el puerto de entrada, se desarrolla toda la lógica de ARP-Path de manera análoga al caso de OMNeT++.

Para probar el código de NOX seguimos el esquema que se muestra en la siguiente figura. Arrancamos primero el controlador NOX con una sentencia como ésta "`./nox_core -i ptcp: modswitch`", y a continuación la red de switches virtualizados podrán conectarse a él ejecutando en Mininet lo siguiente "`sudo mn --controller=remote --ip=CONTROLLER_IP --port=CONTROLLER_PORT --custom TOPOLOGY_PATH -topo mytopo`", donde `CONTROLLER_IP` indicará la IP del controlador, `CONTROLLER_PORT` el puerto y `TOPOLOGY_PATH` la topología virtual de switches OpenFlow. Una vez que el diálogo ha sido creado, el controlador mostrará información sobre los eventos que han sido registrados como cuando un switch es reconocido como activo dentro de la topología "*a new host has joined the network*" o cuando alguno recibe un paquete desconocido "*switch X has received the following frame*", e irá aplicando acciones. Gracias a un comando de

Mininet, es posible realizar ping entre todos los hosts de la topología y ver si es correcto, y con eso fuimos capaces de ver que la implementación de ARP-Path era correcta, probando con diferentes topologías. Pero sobre todo la clave para ver que el funcionamiento era el esperado fue la impresión de trazas por parte del controlador en un fichero, con las que pudimos ver que no sólo el ping funcionaba correctamente, sino que la implementación hacía exactamente lo que le pedíamos.

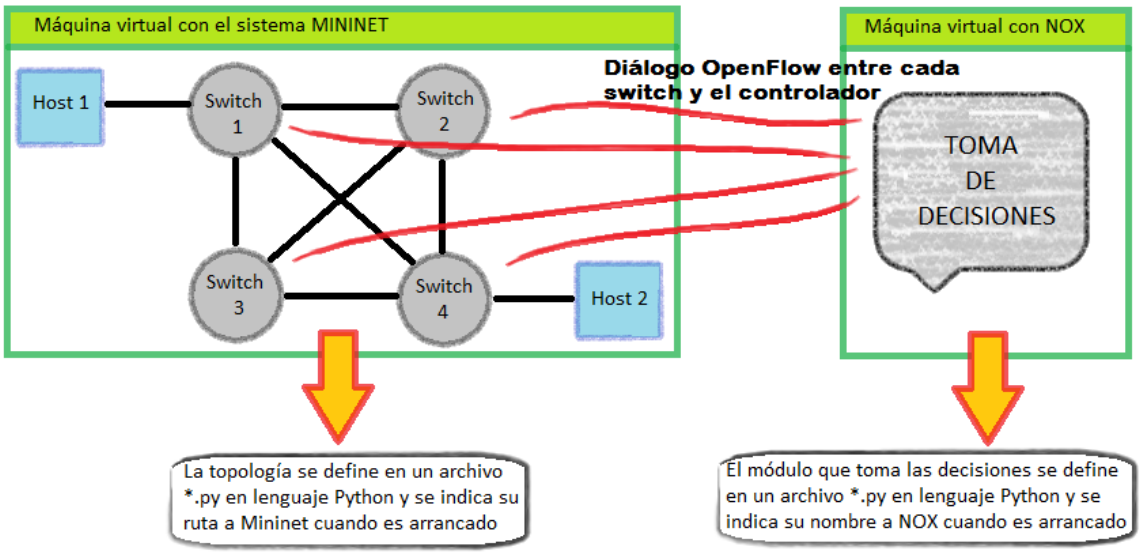


Figura 41. Esquema utilizado para probar la implementación de ARP-Path en OpenFlow/Mininet

Una vez comprobado que el código del controlador OpenFlow era correcto e implementaba la lógica de ARP-Path, se diseñó el esquema de prueba con tarjetas NetFPGA 1G que se muestra en la figura a continuación. Se trata de una malla compuesta de cuatro NetFPGAs con el proyecto de OpenFlow switch cargada y actuando como tal, y con un controlador OpenFlow (NOX) que implementa la lógica de ARP-Path para cada switch, instalado en una máquina independiente, concretamente con el sistema operativo Linux openSUSE [openSUSE]. Cada tarjeta NetFPGA fue instalada en un PC independiente con sistema operativo Linux CentOS [CentOS], realizando la instalación tal y como se indica en la guía de [NetFPGA] para tarjetas de 1 Gbps por separado (dado que también se venden las tarjetas con el sistema completo ya instalado).

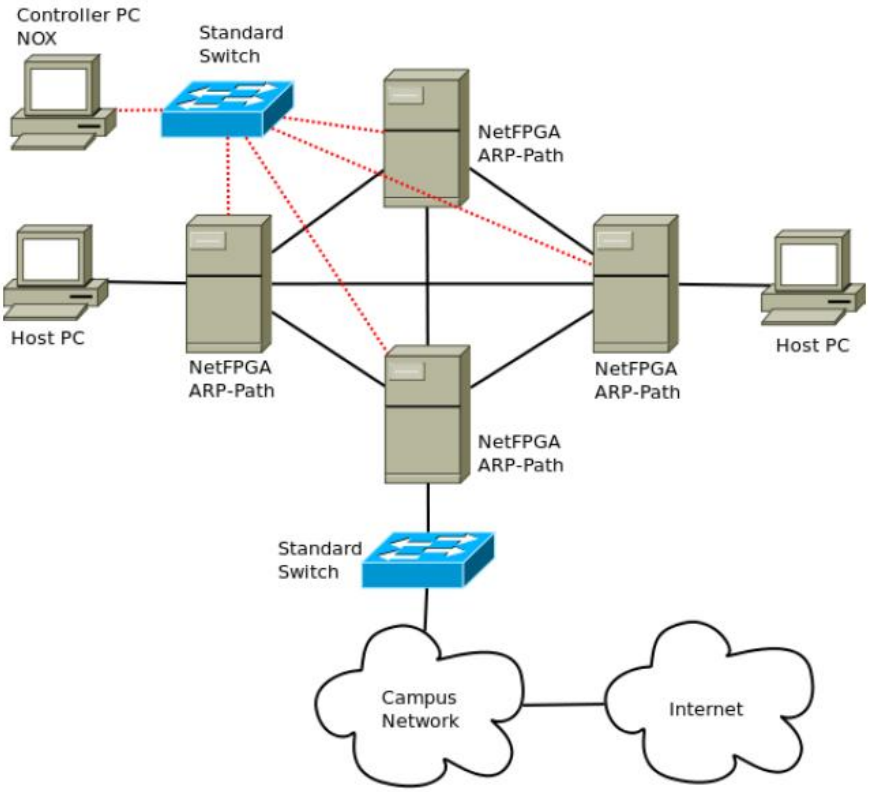


Figura 42. Esquema utilizado para probar la implementación de ARP-Path en OpenFlow/NetFPGA

Con este esquema se realizaron pruebas parecidas a la anterior implementación: ping entre hosts, streaming de vídeo con VLC con UDP y HTTP, conexión a un servidor externo de Internet, pruebas de casos de fallo de enlace, incluso se realizaron tests de estrés de mensajes ARP con la herramienta *arp-sk tool* [arp-sk] de hasta 1000 ARPs por segundo, todos ellos pasados satisfactoriamente.

Esta implementación no sólo tenía los problemas de búffer y paquetes perdidos de la anterior, sino que su tiempo de forwarding rondaba los 0,5 milisegundos (frente a los 0,9 milisegundos de la anterior). Sin embargo, el problema que tenía era otro, los tiempos de establecimiento del camino (y actualización) eran lentos, dado que cuando sólo se realizaba forwarding, era porque las tablas de direcciones ya estaban aprendidas y era puro hardware, mientras que en el aprendizaje se necesitaban crear esas tablas y, para ello, se necesitaba acceder al controlador OpenFlow una vez por cada switch al que llegase el paquete (a la ida y a la vuelta), lo que es lógicamente lento. Además, el paquete de NetFPGA de la instalación todavía no soportaba la detección de caída de enlace, por lo que había que mirar el estado con una función en bucle y lo hacía un proceso lento, pero esto se solucionó con posteriores actualizaciones del paquete.

Para más información sobre esta implementación, se recomienda la lectura de [Iba+10b], [Roj10] y [Iba+11b].

Por último, como curiosidad, queda decir que la implementación de ARP-Path en OpenFlow no sólo se probó con switches virtualizados en Mininet y sobre el proyecto de switch OpenFlow de NetFPGA, sino también sobre software de switch

OpenFlow en Linux (concretamente en Linux Ubuntu sobre las mismas tarjetas Soekris que se utilizaron para la anterior implementación) con enlaces a 100 Mbps y sobre un switch comercial de NEC programable con OpenFlow [NECProgrammableFlow] con puertos a 1 Gbps, gracias a una colaboración con los laboratorios de NEC en Heidelberg [NECLabs]. Estos dos últimos casos se realizaron ya con una nueva versión del protocolo, de la que hablaremos a continuación. Sin embargo, resulta interesante realizar una comparativa de rendimiento de estas implementaciones como resumen, mostrando los resultados aproximados de tiempos de forwarding (principalmente estos se medían en base al ping, de ahí la imprecisión):

- Linux/Soekris(100Mbps): ~0,9ms
- OpenFlow/Mininet(Virtual): ~0,05 ms
- OpenFlow/NetFPGA(1Gbps): ~0,6 ms
- OpenFlow/Linux(100Mbps): ~3ms
- OpenFlow/NECswitch(1Gbps): ~0,6ms

Resulta notable cómo la NetFPGA y el switch de NEC apenas se diferencian, y que el caso de Linux sea tan lento, aunque es razonable al estar implementado el switch OpenFlow como software.

4.1.2 Problemas derivados de la primera versión de ARP-Path (FastPath) y conclusiones

A pesar de las bondades ya presentadas de ARP-Path en su versión inicial (FastPath) y de que incluso se realizaron varias implementaciones con éxito, al profundizar sobre su funcionamiento (principalmente mediante simulaciones con OMNeT++) se descubrieron algunos problemas en él, que sucedían de manera poco habitual, pero no por ello eran menos importantes. A continuación, se hablará de ellos y qué decisiones se tomaron, para presentar después ARP-Path unidireccional, segunda versión del protocolo y evolución de FastPath como tal, pasándose a denominar ya formalmente ARP-Path.

4.1.2.1 Problemas con la asimetría de caminos

Como ya se conoce, ARP-Path en su primera versión necesitaba un mecanismo de prioridad para un caso poco probable, pero no por ello menos importante, que era el de dos mensajes ARP simultáneos, en direcciones opuestas, eligiendo un camino diferente.

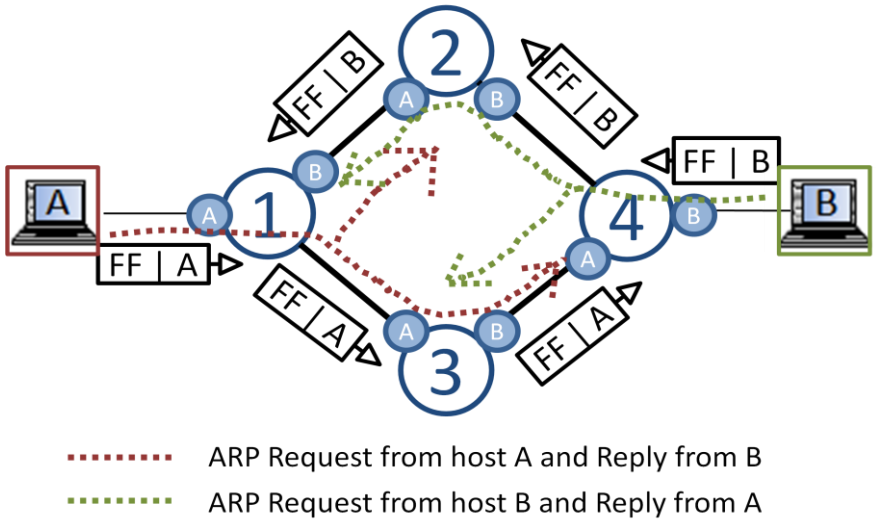


Figura 43. Caso de 2 mensajes ARP simultáneos (ARP Request), en direcciones opuestas, eligiendo un camino diferente en ARP-Path (FastPath)

En la figura anterior, que contiene un ejemplo simple de este caso, es posible ver que la trama con origen A ha escogido como más rápido el camino inferior (en rojo), mientras que aquella con origen B ha escogido el superior (en verde). Así cuando el ARP Request de A llega a B, y viceversa, en cada host se generará el ARP Reply de respuesta, que intentarán confirmar ambos caminos, como vemos en la siguiente figura, lo que no es posible. Es por ello, que cuando el ARP Reply de A hacia B llega al switch 4 (verde), y cuando el ARP Reply de B hacia A llega al switch 1, en la práctica verán que ya existe un camino creado, que es el que se confirma con el otro ARP Reply y se marca rodeado en naranja. Es decir, por ejemplo, el ARP Reply de A hacia B llegará por un tercer puerto del switch 4 que no es ni el que está ya confirmado para B, ni el que está ya confirmado para A, ambos rodeados en naranja y confirmados por el ARP Reply de B hacia A.

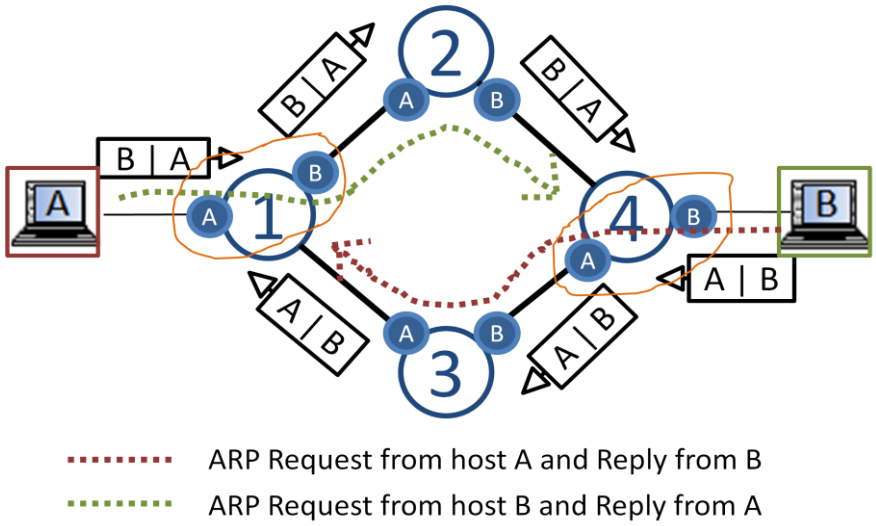


Figura 44. Caso de 2 mensajes ARP simultáneos (ARP Reply), en direcciones opuestas, eligiendo un camino diferente en ARP-Path (FastPath)

Para solucionar este caso, es necesario aplicar un método de prioridad y reconstruir uno de los dos caminos ya confirmados, para ello si A < B se reconstruirá el camino del switch 4 y A se asignará al nuevo puerto, y si B < A entonces se reconstruirá el camino del switch 1 y B se asignará al nuevo puerto, quedando el camino ya simétrico.

Es importante notar la necesidad de simetría. En el caso de que no se aplicase mecanismo de prioridad alguno y se dejasen los caminos como están (o se destruyesen ambos, dejando siempre el último confirmado como válido y único), la diferencia sería que el camino no sería simétrico. Esto puede parecer trivial e irrelevante, pero en la práctica genera un problema. Por ejemplo, en la figura anterior, las tramas de A a B irían por el camino de arriba y entrarían por un puerto diferente al confirmado en el switch 4, lo que podría ser aceptado inicialmente, y algo similar ocurriría con el tráfico de B a A que va por el camino de abajo y entra por un puerto diferente de 1. Sin embargo, como ya conocemos en este protocolo, el refresco de las entradas de A y B se realiza con cualquier trama de datos que tenga como origen dichas direcciones y lleguen por el puerto asociado, por lo que el tráfico de A a B que llega por otro puerto en el switch 4, no refrescaría la entrada de A en el switch 4 y caducaría, causando un problema grave de repación sin ser necesaria, y lo mismo sucedería con la entrada de B en el switch 1. Pero no sólo eso, sino que la entrada de B en el switch 2 y la entrada de A en el switch 3 también caducarían, dado que no llega tráfico por esos puertos con esa dirección origen, causando la desaparición del camino usado en sí.

Así pues, la primera versión de ARP-Path (FastPath), necesitaba que sus caminos fueran siempre simétricos, de tal forma que el refresco, y por tanto el protocolo en sí, funcionara correctamente. Para ello, se aplicaba el mecanismo de prioridad basado en reconfirmación del último camino, si ya existía uno, cuando la trama ARP Reply poseía dirección de origen < dirección de destino. Sin embargo, aún así, esa simetría en la práctica era imposible de cumplir para todos los casos y ahí es cuando entra el siguiente problema: la fluctuación de caminos.

4.1.2.2 Fluctuación de caminos

Este problema es doble, por un lado porque causa oscilaciones graves y, por otro, porque demuestra que la simetría no es posible aplicarla en todos los casos de esta primera versión del protocolo, causando el problema anterior ya mencionado.

Para mostrar el problema, vamos a utilizar un ejemplo una vez más. A continuación vemos dos figuras. En la primera se puede apreciar que se ha construido un camino de comunicación entre A y B perfectamente simétrico. En la segunda, C comienza una comunicación con A, siendo el ARP Request más rápido el que llega por la parte de arriba de la topología, pero cuando el ARP Reply desde A a C va confirmando ese camino se encuentra con un dilema en el switch 6. El dilema es básicamente qué hacer con la confirmación del puerto para la dirección A:

- Si se confirma el nuevo puerto de arriba, se perderá la simetría para la comunicación entre A y B.

- Si se deja el puerto ya confirmado, no existirá simetría para la comunicación entre A y C.

- Si se aplica el mecanismo de prioridad ya descrito, necesario para garantizar la simetría en el caso de ARPs simultáneos, es posible que la dirección A sea menor que B, y menor que C también, por lo que cada vez que llegue un ARP Reply se cumplirá que que A < B y que A < C y se cambiará de nuevo el camino, por lo que habrá oscilación de caminos y, lo que es más importante, seguirá existiendo un camino asimétrico con los problemas que de ello derivan.

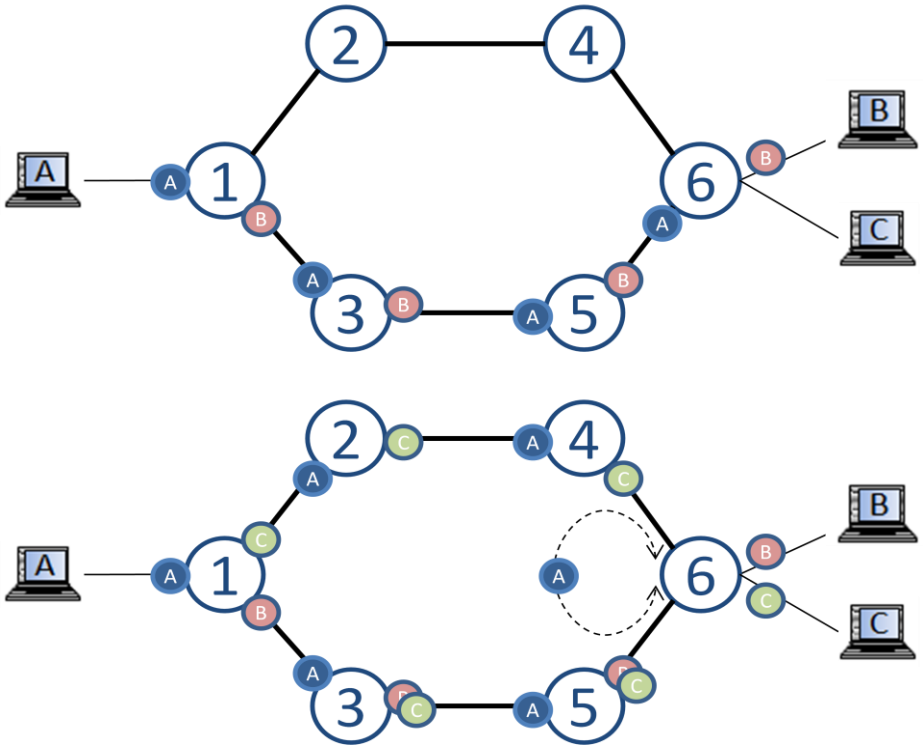


Figura 45. Fluctuación de caminos en ARP-Path (FastPath)

Con este caso teórico, se sacan a la luz dos problemas y dos conclusiones:

1. Las fluctuaciones u oscilaciones de caminos, que en ningún caso son recomendables y que podrían llegar a resultar graves con un mayor número de hosts. Nótese que en el ejemplo son sólo tres y ya hay problemas, pero podemos imaginar qué pasaría si en el switch 6 hubiese muchos hosts conectados comunicándose con A y todos ellos con una dirección MAC mayor a la de A.

   Conclusión: El mecanismo de prioridad, en caso de ser necesaria su aplicación, no puede basarse únicamente en la comparativa de los valores de la MAC origen y destino.

2. Hay casos en que la asimetría de caminos es irremediable. Puede resultar un ejemplo muy rebuscado, pero sin duda con tres hosts ya se dan casos en los que no se puede garantizar simetría y se puede imaginar que con

un número mayor de hosts puede que incluso lo raro sea no encontrar algún caso de asimetría. Y recordemos, la asimetría no es permisible, pues causa falta de refresco y reparaciones innecesarias.

Conclusión: Existen dos opciones, que generaron dos vertientes del protocolo: o se deja de utilizar exclusivamente la MAC origen para la creación de caminos (véase el apartado de Flow-Path) o se varía el protocolo actual de manera que la asimetría sea permisible (véase el apartado de ARP-Path unidireccional, como evolución directa del protocolo).

4.1.2.3 Necesidad de flag de reparación

Mientras que los dos problemas anteriores derivaron de un análisis teórico, este tercero apareció en la práctica tras el análisis paso a paso de una simulación de OMNeT++ que fallaba.

Como ya es sabido, esta primera versión del protocolo, genera un camino con el ARP Request, que luego es confirmado con el ARP Reply. Este camino inicialmente se puede decir que es invariable en el tiempo. Sólo en los siguientes casos podría ser modificado:

- En el caso en el que los temporizadores caduquen y entonces el camino se elimine.
- En el caso en el que se aplique el mecanismo de prioridad para simetría ya conocido.
- En el caso de necesidad de reparación del camino, es decir, en el caso de recibir un mensaje especial de reparación (para distinguirlo del resto de casos).

Este problema está relacionado con la reparación de caminos. En esta primera versión se habían pensado diversas opciones, que luego evolucionaron a lo que veremos en un apartado posterior. En el simulador OMNeT++ se había implementado una de ellas, sin embargo, no es necesario explicar cómo funcionaba exactamente para entender el concepto.

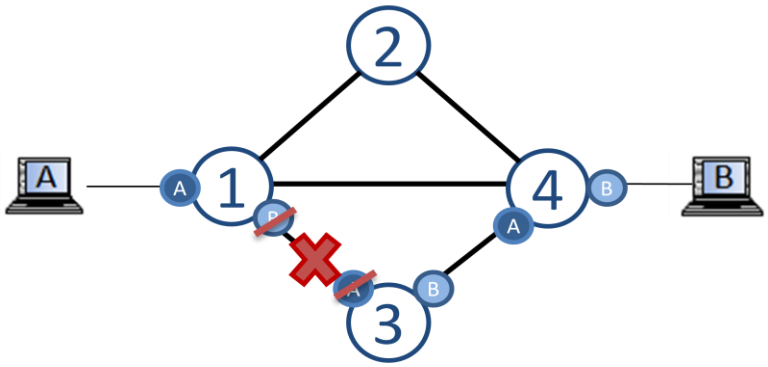


Figura 46. Ejemplo de caída de enlace de un camino en ARP-Path (FastPath)

En la figura anterior vemos un ejemplo de caída de un enlace entre el switch 1 y el 3. En ese caso, las entradas correspondientes a los puertos del enlace caído en 1 y 3 se borran. El mecanismo de reparación necesitará una nueva selección de camino entre A y B, por lo que se necesita un nuevo ARP Request que seleccione un camino alternativo. Sin embargo, no se puede utilizar un mensaje ARP en sí, porque los caminos no se pueden modificar con mensajes de datos, ni con ARPs, sólo con mensajes especiales y si no usamos un mensaje especial, no habría forma de modificar la entrada de A en el switch 4 que ya no es válida.

Así pues, el mensaje "especial" de reparación vence a cualquier aprendizaje y crea el nuevo camino y además lo hace igual que el ARP, primero asocia un puerto con un mensaje análogo al ARP Request, y luego el destino enviará una respuesta con otro mensaje "especial" análogo al ARP Reply que lo confirmará. Hasta ahí todo bien y correcto. El problema viene cuando se da el caso concreto que sucedió en la simulación. Imaginemos que el camino seleccionado tras reparar ha sido el compuesto por los switches 1 y 4 y ya está confirmado, pero resulta que llega una copia lenta del mensaje especial de reparación desde el switch 2 al 4. En el caso de aprendizaje habitual con ARP no sucedería nada, ya que el primer camino confirmado es fijo y los siguientes ARP Request se descartarían. Sin embargo, hemos dicho que este mensaje especial destruye cualquier aprendizaje (pues así ha sido diseñado para hacer posible la reparación), por lo que este mensaje de 2 a 4, destruiría el camino formado por 1 y 4, y confirmaría el suyo, que es el formado por los switches 1, 2 y 4. Es decir, confirmaría el camino más lento en la práctica, el del último mensaje especial, lo que no es deseable. Y no sólo eso, sino que esto puede incluso dar lugar a bucles y tormentas de tramas en ciertas topología y es concretamente lo que sucedió en la simulación, que aparecieron casos de bucles tras la reparación.

Para solucionar este problema, es necesaria la inclusión de un flag de reparación para cada entrada de la tabla. Este flag se activa cuando acaba de suceder una reparación y caduca al cabo de un tiempo (por ejemplo, varios segundos). De manera que si el flag está activo, no habrá mensaje alguno que pueda modificar el camino y se evite el problema de que se confirme el camino más lento y los posibles bucles. La inclusión de este flag tiene el mismo efecto que si el estado '*locked*' no dejase de bloquear una vez ha sido confirmado y durase "algo más", hasta que las posibles tramas duplicadas dejen de estar en la red.

4.1.3 ARP-Path unidireccional

El protocolo ARP-Path unidireccional es una evolución de ARP-Path derivada del problema de fluctuación de caminos que se mencionó en apartados anteriores. La otra alternativa era Flow-Path, que se explicará más adelante.

Así pues, en esta evolución, se decide realizar un cambio que permita la asimetría de caminos, conservando el aprendizaje por MAC, y así evitar la necesidad de utilizar un mecanismo de prioridades. Curiosamente, la necesidad del flag de reparación en las entradas también se elimina a su vez.

La idea es dejar de crear caminos entre pares de direcciones MAC origen/destino, para pasar a generar caminos sólo hacia el origen, utilizando el descubrimiento de camino que ya conocíamos de la versión inicial basado en los mensajes ARP. Con esta filosofía, ahora se crean caminos independientes con cada mensaje ARP hacia el origen de quien emite la trama y deja de existir lo que antes denominábamos "confirmación del camino", pues el camino no necesita confirmarse, se descubre y existirá como tal mientras se utilice (y por tanto, se refresque). A continuación se describe la formación del camino para los dos tipos de mensaje ARP:

Descubrimiento y generación del camino hacia el origen (*ARP Request*): Cuando un host A quiere comunicarse con uno C, emitirá un ARP Request. Con éste mensaje se realiza exactamente lo mismo que ya se hacía en la versión anterior, se descubre el camino más rápido hasta C, asociando en cada puente de la red un puerto a su dirección A, es decir, creando un árbol de encaminamiento enraizado en el host A, tal y como se ve en la figura siguiente.

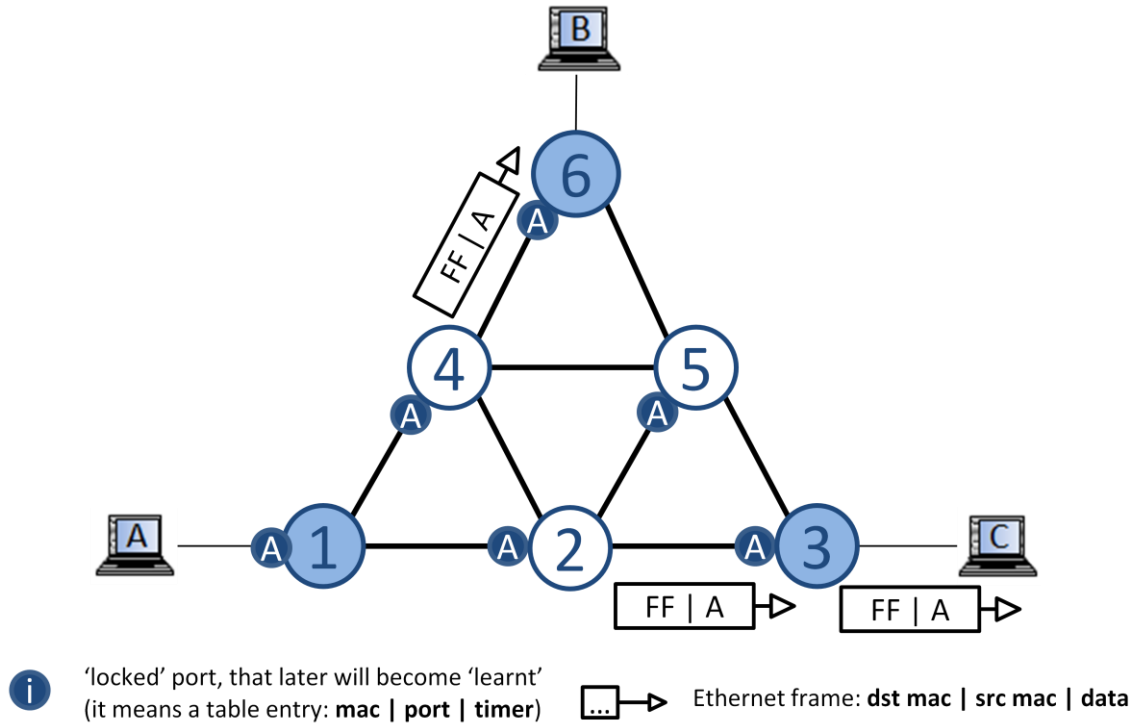


Figura 47. Aprendizaje de la dirección del host origen (A) mediante el mensaje ARP Request en ARP-Path (unidireccional)

La diferencia radica en que, una vez propagado por la red en ARP Request, el camino desde cualquier host hasta A ya ha sido creado y no hace falta confirmarlo. Inicialmente su estado es '*locked*' y no se admiten mensajes ARP por otros puertos, para evitar bucles, y posteriormente, tras cierto tiempo, pasa directamente al estado '*learnt*', es decir, estado aprendido (recordemos que antes sólo se cambiaba de estado al recibir el ARP Reply).

Generación del camino hacia el destino (*ARP Reply*): El mensaje ARP Request recibe un ARP Reply como respuesta, que ahora se utiliza para aprender el camino

hacia el destino C, pero no para confirmar, ni afectar en ningún caso, al camino creado por el ARP Request para el origen A. El aprendizaje con el ARP Reply es análogo al realizado con el ARP Request, pero sin ser broadcast, tal y como se ve a continuación:

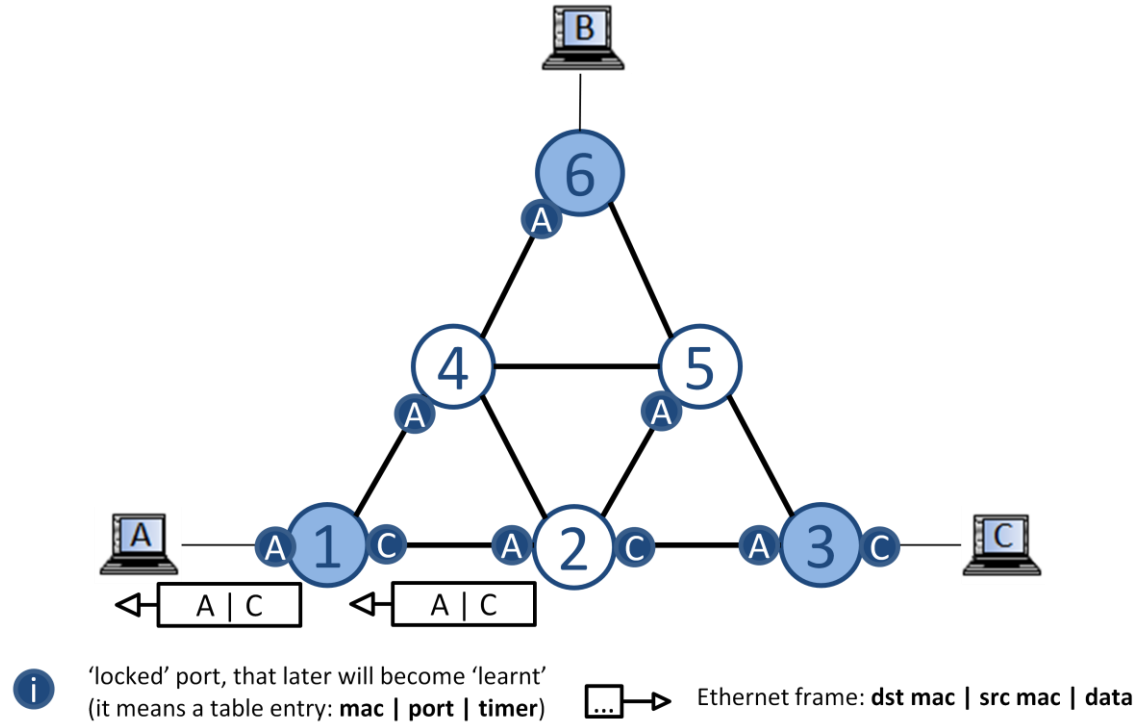


Figura 48. Aprendizaje de la dirección del host destino (C) mediante el mensaje ARP Reply en ARP-Path (unidireccional)

Es decir, tanto el ARP Request como el ARP Reply se utilizan para crear caminos hacia el origen de cada una de los mensajes, que son totalmente independientes y no necesitan ser confirmados. La única diferencia entre ellos es que el primero es una trama broadcast y descubre toda la red, mientras que el segundo es una trama unicast y sólo sigue el camino ya descubierto. Por lo que el camino creado por el ARP Request, primero crea las entradas en estado '*locked*' para evitar posibles bucles y tras cierto tiempo las pasa a '*learnt*'; mientras que el ARP Reply directamente crea entradas en estado '*learnt*', puesto que no necesita evitar bucles al ser una mensaje unicast.

Los **estados** de '*locked*' (aprendizaje con bloqueo) y '*learnt*' (aprendizaje) de cada una de las fases también tienen un **temporizador** (timer), que caducará en caso de no ser refrescado. La misión del estado '*locked*' es evitar bucles que podrían crearse por las tramas que se envían replicadas por todos los puertos y el valor de su timer es pequeño (< 1 segundo). Mientras que el estado '*learnt*' muestra un camino aprendido y su misión es encaminar tramas, por lo que su timer será mayor (de varios minutos). El **refresco** de dichos timers se realiza de la siguiente forma:

- Si el estado de la entrada de A es '*locked*', éste se refresca con cualquier trama ARP Request que tenga como origen A y que llegue por el puerto asociado.

- Si el estado de la entrada de A es '*learnt*', éste se refresca con cualquier trama de datos que tenga como destino A y que haga uso de la entrada.

Podemos ver la clara analogía con la versión anterior respecto a los estados, salvo que van seguidos y uno no confirma al otro. La diferencia evidente es que el estado principal de '*learnt*' antes se refrescaba con tramas cuyo origen fuese la MAC de la entrada, mientras que ahora se refrescan con tramas cuyo destino es dicha MAC. Es decir, no se refresca el camino hacia atrás, sino hacia adelante y a esto lo denominamos *forward refresh* o refresco hacia delante. Este tipo de refresco supone que el camino hacia el destino, hacia delante, será correcto mientras se sigan enviando tramas hacia dicho destino, ya que si no, provocaría la reparación del camino y sería reconstruido sin necesidad de que caduque antes. Por lo que las entradas sólo caducarán cuando se deje de emitir tráfico a ese destino. Por ejemplo, en el caso de la figura, si A quería emitir hacia C, se crea un camino en ambos sentidos, pero si sólo se utiliza en la dirección de A a C, las entradas para A caducarán, mientras que las de C se refrescarán y seguirán siendo válidas hasta que se pare dicha comunicación. Este tipo de comunicación unidireccional no era válido para la versión anterior, porque si sólo se emitía tráfico de A a C, la entrada de C no se refrescaba (sólo la de origen, A) y acababa necesitando reparación.

La segunda diferencia respecto a los estados, quizás no tan evidente, es la duración del primer estado ('*locked*') y de las entradas en general. En la primera versión, el estado 'locked' no siempre dura lo mismo, dado que su duración depende del momento en que el camino se confirme (o no), por lo que las entradas más cercanas al destino tienen tiempos '*locked*' menores que las más lejanas, ya que se confirman antes. Además, en la primera versión, las entradas que no se confirman, caducan tras el estado de '*locked*', mientras que en la segunda versión todas pasan al segundo estado de manera independiente. En las siguientes figuras se muestra esto.

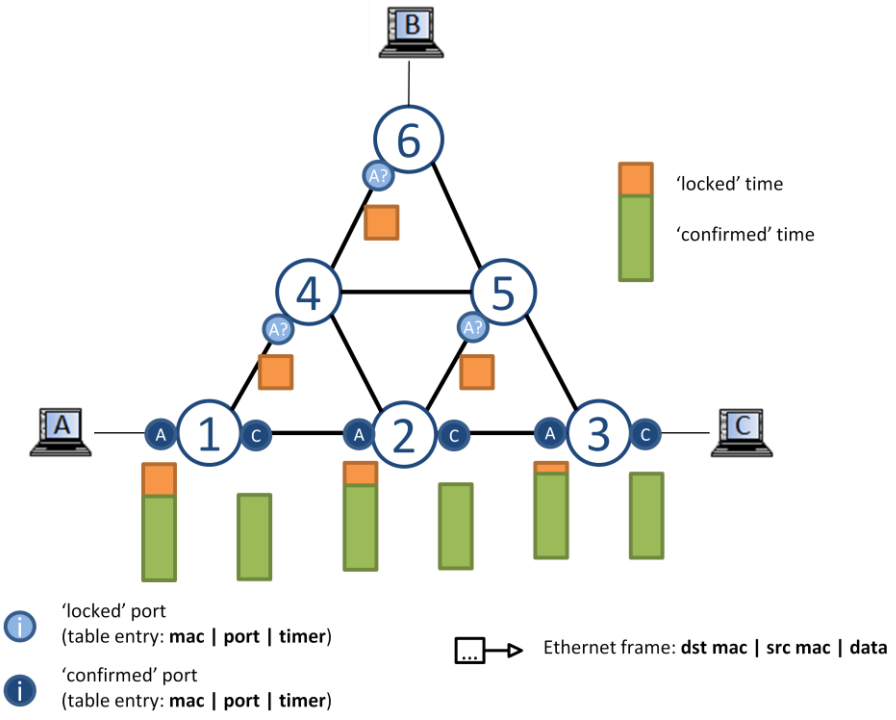


Figura 49. Duración de las entradas y sus respectivos estados en ARP-Path (FastPath)

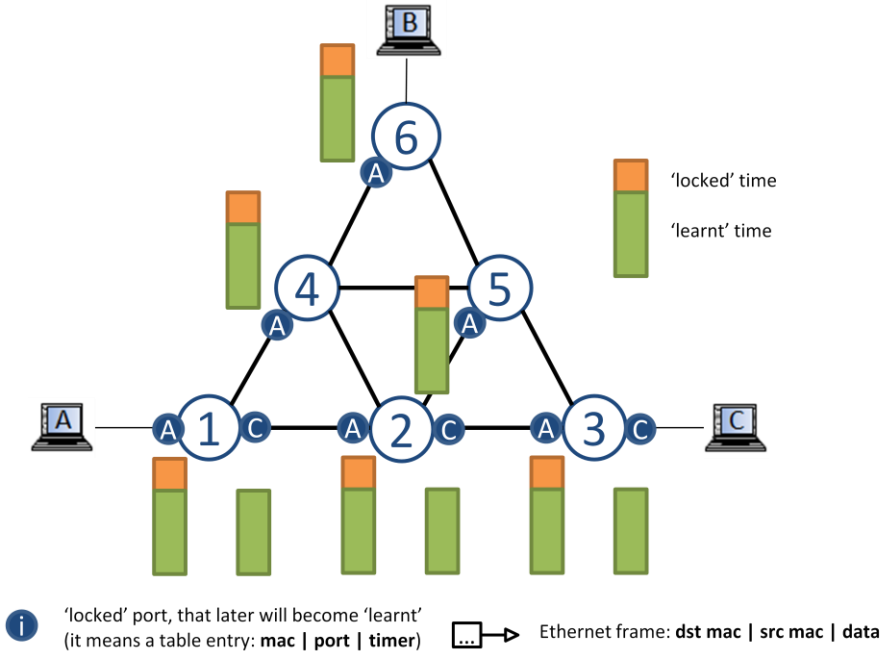


Figura 50. Duración de las entradas y sus respectivos estados en ARP-Path (unidireccional)

El hecho de que el protocolo genere los caminos de manera independiente y que estos se refresquen hacia adelante, elimina la necesidad de que estos caminos tengan que ser simétricos y por tanto, quita el principal problema de la versión anterior. Además, al no tener que ser simétricos, no es necesario ningún mecanismo de prioridad a aplicar y no habrá problemas de oscilaciones. Finalmente, como los caminos no se confirman, no se necesitará ningún flag de reparación, pues siempre tendrá que expirar el estado de '*locked*' para poder pasar a '*learnt*', y ese tiempo evita todo tipo de bucles y de flags extra. Así pues, con esta nueva versión del protocolo se solucionan los problemas de la anterior.

Otra diferencia principal, es que los caminos en estado '*learnt*' no son tan fijos como lo eran en '*confirmed*'. Esto se decidió como un "extra", dado que con las anteriores características ya solucionan los problemas de la anterior versión. Sin embargo, los caminos antes sólo se podían destruír con un mensaje especial (por ejemplo, de reparación) y esto parecía complicar algo los casos de movilidad de hosts, por ejemplo. Así que se decidió que cualquier ARP reconstruía los caminos cada vez, siempre y cuando el estado fuese '*learnt*'. Esto simplifica mucho el protocolo y hace más sencilla la movilidad, pero en la práctica tiene un problema, que se analizará en el siguiente apartado y se soluciona con la tercera versión del protocolo (de la que se habla más adelante también), que se queda en un paso intermedio: ni fija totalmente los caminos requiriendo mensajes especiales, ni los deja totalmente flexibles.

Finalmente, en esta versión del protocolo se separa el tratamiento de los mensajes broadcast. Existen tablas de encaminamiento que se generan a partir de los ARP (tanto broadcast como unicast) llamadas *Learning Tables* (LT) y que se utilizan por las tramas de datos unicast, pero el resto de mensajes broadcast utilizan otra tabla independiente llamada *Broadcast Table* (BT), que sólo tiene tiempo de

'*locked*' para evitar los bucles, pero no '*learnt*' dado que no se aprenden rutas con ellos. Esta diferenciación es la que marca el nombre del protocolo, que acaba siendo ARP-Path (porque sólo aprende con los mensajes ARP), frente al antiguo FastPath.

4.1.3.1 Implementación en NetFPGA (y actualización de las implementaciones en OMNeT++ y OpenFlow)

Con la nueva versión de ARP-Path llegaron las actualizaciones de las implementaciones en el simulador OMNeT++ y en el controlador NOX de OpenFlow. La implementación Linux no se actualizó debido a su complejidad y porque OMNeT++ ya nos daba la posibilidad de simular y OpenFlow de probar en una plataforma real, siendo éste último mucho más sencillo de reprogramar que la versión de Linux. Con estas actualizaciones, habiendo resuelto los problemas de la versión anterior, se decidió el desarrollo de una plataforma totalmente hardware (recordemos que OpenFlow tenía parte en software, que era el controlador y ralentizaba partes de la ejecución del protocolo en la práctica), y para ello se escogieron, por su facilidad de desarrollo (el diseño de la tarjeta ya está hecho y depurado, sólo es necesario programarlo), las tarjetas NetFPGA 1G (1Gbps).

La implementación de ARP-Path en NetFPGA sigue el esquema de otras implementaciones, como pueden ser el switch o el router estándar, dos de los proyectos más comunes y depurados del paquete básico de instalación de NetFPGA 1Gbps. Como se ve en la siguiente figura, el proyecto se compone de las siguientes carpetas:

- *include* – Contiene las declaraciones de variables y funciones más básicas, como son las del propio protocolo ARP-Path, de la memoria y de los registros utilizados, e incluso la definición del proyecto en sí en un fichero *.xml.

- *lib* – Contiene las *funcionalidades software* del proyecto, que pueden ser escritas en C, Perl o Python. Por ejemplo, en C está programada la inicialización de parámetros de las tarjetas y en python se pueden preparar tests de pruebas para las tarjetas.

- *src* – Contiene todo el código Verilog del proyecto, es decir, la lógica del protocolo en sí. Hay varios ficheros, algunos reutilizados de otros proyectos (memoria, implementación de tablas,…) y uno que implementa la lógica de ARP-Path denominado *arppath.v*.

- *synth* – Contiene el resultado de la síntesis o sintetización del proyecto completo, que es un *bitfile* (fichero que contiene bit a bit el proyecto desarrollado) que posteriormente se descargará en la tarjeta NetFPGA.

- *vsim* – Contiene algunos ficheros para la simulación del protocolo (proceso previo a la síntesis).

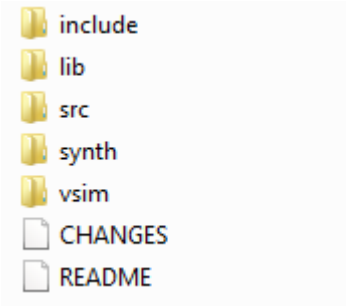


Figura 51. División en directorios del proyecto ARP-Path en NetFPGA 1G

Por lo que, de todos los directorios, podemos decir que hay dos principales en la implementación: *lib*, que permite meter los parámetros iniciales del switch y arrancarlo (parte software), y *src*, que contiene la lógica de ARP-Path (parte hardware).

Hardware NetFPGA (*src*): El código de *arppath.v* es Verilog, que como otros lenguajes de descripción hardware (*Hardware Description Language*, HDL), por ejemplo VHDL, no es puramente un código secuencial (como es el caso de Python y C++, los lenguajes utilizados para OpenFlow y OMNeT++, respectivamente), sino que también se compone en su mayoría de código concurrente, es decir, sentencias de código que se ejecutan simultáneamente, sin prioridad entre unas y otras. Por lo tanto, la programación deja de ser tan sencilla como crear sentencias condicionales típicas *if/else* y se necesita una máquina de estados algo más compleja. Además, se han de tener en cuenta las restricciones hardware y los recursos limitados del sistema.

Este código implementa la fase de búsqueda del puerto de salida en una tabla de direccionamiento para un switch ARP-Path y siguiendo el protocolo ARP-Path. Dichas tablas están implementadas como tablas hash que a su vez utilizan la implementación de un hash universal (véase *lookup_module.v*). El módulo (que comienza con `module arppath`) funciona de la siguiente manera: cuando un paquete llega, se almacena en la memoria FIFO *input_fifo* y se disecciona en los componentes del paquete ("*parsea*", de *parse* en inglés) con el módulo *parse_pkt* a la vez. La salida diseccionada se utiliza en la máquina de estados de este módulo para buscar valores en las tablas de direccionamiento. Dependiendo del tipo de paquete (si es unicast, broadcast,… o si es un mensaje especial de reparación) los tipos de búsqueda pueden variar, y según los valores de dichas búsquedas, la máquina de estados irá avanzando.

La máquina de estados posee 15 estados, entre ellos: "*esperando para disección de paquete*", dos estados para "*chequeo de las tablas de mensajes especiales de reparación*", otros dos para la "*búsqueda común de entradas*" (el caso más utilizado), "*descartar paquete*", "*enviar paquete*", "*enviar paquete especial de reparación*", etc. Como es posible ver, con cada paquete se consulta habitualmente más de un estado de la máquina de estados.

Gran parte del comienzo del código son definiciones, a continuación va la lógica del protocolo (con la máquina de estados) y casi la mitad final del código es código utilizado para chequear las funcionalidades implementadas con el simulador,

previsamente a la descarga del código en la tarjeta. Este código final se define en un módulo diferente denominado `module arppath_tester();`.

Software NetFPGA (*lib*): Para la parte software hay código en C y en Python. En C se encuentra el módulo *cli.c*, que es básicamente una interfaz de línea de comandos (*Command-Line Interface*, CLI), y que se basa en otros como arppath.c (funciones de acceso al switch ARP-Path en Verilog, como es escritura y lectura en registros concretos), *reg_defines_arppath.h* (con definiciones de algunos parámetros y las direcciones en memoria de los registros del switch ARP-Path) *nf2util.c*, *cliutil.c* y *util.c* (que contienen funciones útiles varias), Este módulo permite el arranque de las tarjetas (que previamente se habrán cargado con el código Verilog ya compilado) y la inicialización de variables, y básicamente lo que hace es coger los valores pasados en el comando y aplicarlos a los diferentes registros del switch ARP-Path.

Posteriormente se añadió, también programado en C, el módulo *cliloop.c*. Se trata de un módulo que, como su nombre indica, implementa un bucle que chequea constantemente el estado de los puertos de la tarjeta NetFPGA, para ver el estado de los enlaces, y así borrar entradas de las tablas y aplicar la reparación cuando así convenga. Inicialmente esta funcionalidad debería haber sido implementada en hardware, pero dada su complejidad fue realizada finalmente en software. Por ello, lógicamente, es una funcionalidad bastante lenta.

Por último, en la carpeta de Python se encuentran dos ficheros: *test_pkts.py* (fichero de tests) y *testlib.py* (que es la librería utilizada para el primer archivo). El fichero de tests contiene código Python definiendo una serie de tests para probar las tarjetas NetFPGA. El código definido prueba casi exactamente las mismas funcionalidades que en `module arppath_tester();` Verilog, la diferencia es que estas pruebas se realizan ya sobre la tarjeta física y con tráfico real, frente al código Verilog que emplea una simulación. Con este código se pueden repetir casos de uso concretos y depurar el switch ARP-Path, de hecho fueron especialmente útiles al comienzo del proyecto, dado que en Verilog sólo se puede simular una tarjeta, pero con Python podíamos hacer tests de más de una tarjeta, probando así casos que no podrían probarse con una sola.

Gracias a esta implementación, se demostró el funcionamiento de un switch ARP-Path casi totalmente de fábrica, sin limitaciones por ejecución en espacio de usuario (como pasaba con Linux), sin cuellos de botella (como con OpenFlow), y con un único problema, que fue la necesaria implementación en software de la detección de fallo de enlace.

Los primeros escenarios de pruebas se realizaron con cuatro tarjetas NetFPGA montadas sobre un único PC con cuatro buses PCI y con una instalación de los paquetes NetFPGA sobre el sistema operativo CentOS. Posteriormente se buscó la implementación de switches únicos, más pequeños, tal y como era la implementación de Linux/Soekris, para lo que se descartó cualquier diseño hardware (dada su dificultad y coste) y se probó la instalación de las tarjetas NetFPGA individualmente sobre las tarjetas Soekris, que poseían un bus PCI, utilizando una versión de CentOS anterior. Dicha instalación fue muy satisfactoria y actualmente corresponden a la implementación de un switch ARP-Path más cercana a un switch de fábrica.

Por último, como curiosidad, adelantamos los tiempos de forwarding (medidos con ping) para comparar con el resto de implementaciones, que tras la actualización a la última versión de ARP-Path no variaron en sus valores:

- Linux/Soekris(100Mbps): ~0,9ms
- OpenFlow/Mininet(Virtual): ~0,05 ms
- OpenFlow/NetFPGA(1Gbps): ~0,6 ms
- OpenFlow/Linux(100Mbps): ~3ms
- OpenFlow/NECswitch(1Gbps): ~0,6ms
- NetFPGA(1Gbps): ~0,6ms

Como es posible apreciar, el rendimiento de forwarding de la tarjeta NetFPGA en solitario es casi exacto al de OpenFlow/NetFPGA. Esto se debe a que el forwarding se realiza en ambos casos en hardware, mediante la implementación de tablas de direccionamiento hardware. La diferencia entre las dos implementaciones se ve en el tiempo de establecimiento de los caminos y en la reparación, que en OpenFlow era de varios milisegundos (pues se necesita un acceso al controlador por cada switch en la red) y en la implementación pura en NetFPGA seguía siendo inferior a un milisegundo.

4.1.4 Problemas derivados de la segunda versión de ARP-Path (versión unidireccional) y conclusiones

Esta segunda versión de ARP-Path resuelve los principales problemas de la primera. Sin embargo, ya adelantamos un problema teórico que surgió después y viene de la decisión de que los caminos pudiesen ser reconstruidos con cualquier mensaje ARP, frente a la versión inicial, que fijaba los caminos para siempre y, aunque no era un problema en sí, hacía que la movilidad de hosts fuese más rígida, por ejemplo. A continuación se detalla.

4.1.4.1 Inestabilidad de caminos con ARP

Este problema surge de manera meramente teórica, tras el análisis profundo del protocolo, pues en la práctica, tras las diversas implementaciones y sus análisis (que veremos en un apartado posterior) no se descubrió ningún otro problema mientras que se vieron solucionados los de la primera versión. Se trata de la inestabilidad de caminos con ARP, al reconstruirse los caminos cada vez que un host vuelve a emitir un mensaje ARP. Esta inestabilidad a su vez puede causar dos problemas: oscilaciones de tráfico y desorden de algunas tramas. Evidentemente, son dos posibles problemas que en la práctica no se han visto, pero que pueden darse en

determinadas circunstancias (incluso ambos a la vez) y que analizaremos a continuación por separado.

Para explicar estos posibles efectos, utilizaremos un ejemplo. Supongamos que existe cierta comunicación estable entre un host A y otro C, tal y como se muestra en la siguiente figura (de A a C el camino es 1-2-3 y de C a A es 3-2-1).

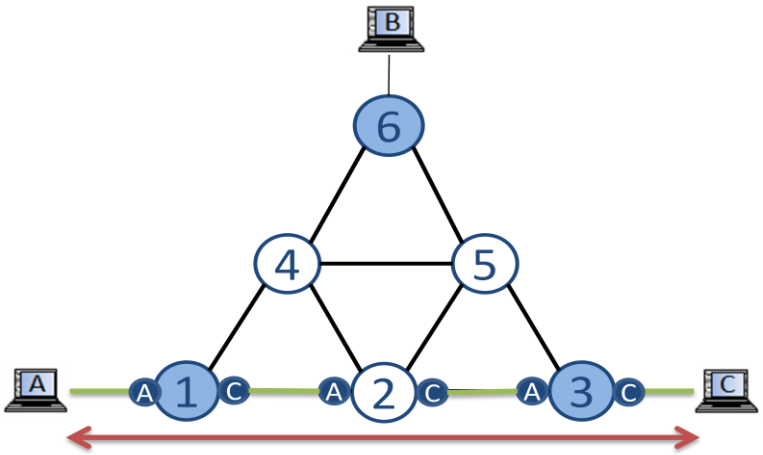


Figura 52. Ejemplo de comunicación estable entre un par de hosts con ARP-Path

Oscilación del tráfico

Establecida dicha comunicación, imaginemos que ahora el host A quiere iniciar una segunda comunicación con otro, B. Para ello, comienza emitiendo un ARP Request para descubrir la dirección de B y, dado que los caminos se reconstruyen con cada ARP, reconstruirá ya de paso todos los caminos desde cualquier host a A. El problema viene cuando los únicos enlaces que se estaban usando son aquellos que componen el camino 1-2-3, lo que causa que sean más lentos y que, casi seguramente, el nuevo camino hacia A será uno diferente, más rápido. Por ejemplo, en la figura siguiente se muestra un caso posible en el que ahora el camino de A a C sigue siendo 1-2-3, pero el de C a A pasa a ser 3-5-4-1.

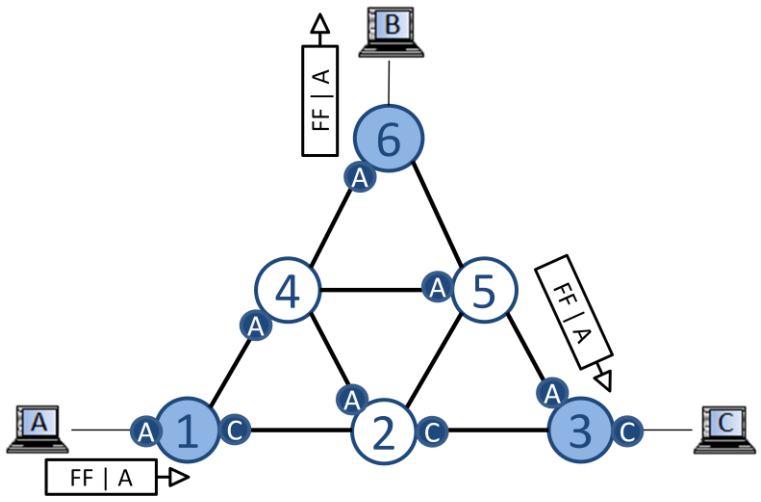


Figura 53. Ejemplo de reconstrucción de un camino para un host A por la emisión de un ARP

Lo que acentúa el problema y genera las oscilaciones es que esta situación se repetirá con cada nuevo ARP. Es decir, si ahora se generase otro ARP, es posible que se volviese a elegir un camino que no está siendo usado. Esto causa oscilaciones que no son en absoluto necesarias y además puede sumarse a otro efecto negativo que se describe a continuación.

Desorden de tramas

Siguiendo el ejemplo anterior, imaginemos que cuando A envía el ARP, C se encontraba mandando datos con destino al host A. Como el camino se transforma en medio del envío, pueden darse casos de desorden de tramas como el que se muestra en la siguiente figura (en gris las tramas de datos y en blanco el ARP Request). Por ejemplo, supongamos que los enlaces entre 1 y 2, y 2 y 3 están bastante saturados de tráfico, entonces el host C primero manda la trama 1 y la 2, y éstas empiezan a recorrer el camino 3-2-1 hacia A (camino viejo marcado con entradas en naranja). Entonces es cuando A manda el ARP Request y éste genera un nuevo camino hacia A, y cuando C manda la trama 3 se dirige directamente por el camino nuevo 3-5-4-1 (camino nuevo marcado con entradas en azul). En el caso de que 3-5-4-1 esté menos saturado que el antiguo 3-2-1, se puede dar el caso de que primero llegue la trama 3, luego la 1 y luego la 2. Es más, el desorden puede ser múltiple, dado que incluso podría llegar primero la 3, luego la 2 (que llega entonces al switch 2 y encuentra el nuevo camino por 4 y por 1, más rápido), y finalmente la 1, que seguía atascada en el camino previo.

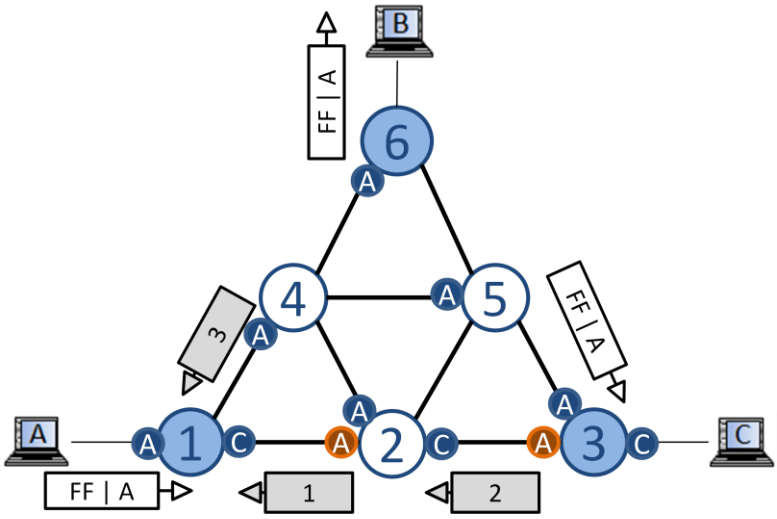


Figura 54. Ejemplo de desorden de tramas para un host A por la emisión de un ARP

Esta situación resulta un problema porque a nivel de enlace las tramas deberían ordenadas, al margen de que luego los niveles superiores tengan la capacidad de ordenarlas o no.

Es importante remarcar el hecho de que estos dos problemas derivan de la emisión de un ARP por parte de uno de los hosts realizando la comunicación a un tercer host. Sin embargo, no es necesario que haya más de dos hosts en la red para que sucedan estos problemas. La emisión de un ARP por parte de un host dentro de una comunicación en proceso puede deberse al caso de la renovación de la caché de

ARP y dependerá de si el sistema operativo del host es Microsoft [ArpCacheLife] o Linux [arpLinux], entre otros, además de la propia configuración del host. Por ejemplo, en el caso de Microsoft se aplicaría el valor de ArpCacheMinReferencedLife, que es el valor que puede permanecer una entrada de ARP en uso (es decir, renovada por comunicación todavía activa) en caché hasta que decida renovarse y que son 10 minutos. Por lo que cada 10 minutos podría emitirse un ARP y reconstruir los caminos, provocando los problemas ya comentados incluso con sólo dos hosts en la topología.

Conclusión: Quizás fijar los caminos para siempre con el primer aprendizaje es demasiada inmovilidad, pero la elasticidad de reconstruir los caminos con cada ARP es también excesiva, por lo que se debe encontrar un punto intermedio.

4.1.5 ARP-Path unidireccional alternativo/retardado

La tercera versión del protocolo fue denominada ARP-Path unidireccional alternativo/retardado porque inicialmente surgió como una alternativa de aprendizaje de ARP-Path unidireccional, pero en la práctica su "retardo" en el aprendizaje definitivo (de ahí el "retardado") es la clave que permite la elasticidad suficiente en los caminos sin pasarse, es decir, ese punto medio que se buscaba frente al problema de la inmovilidad excesiva de la primera versión y la inestabilidad excesiva de la segunda.

Esta versión aún no ha sido implementada, pero a continuación se muestran sus ideas principales, concretamente aquellas que diferencian esta versión de la anterior, pues la mayor parte del protocolo permanece intacta (cómo son las *Learning Tables* y *Broadcast Tables*, LT y BT). Nótese que estas ideas se encuentran en su fase inicial y requerirían un análisis más profundo.

La fase de descubrimiento de camino entre un host origen y otro destino es idéntica a la versión anterior. El problema, como ya se conoce, llega cuando un nuevo ARP entra en conflicto con entradas ya existentes, puesto que en la versión anterior se tomaba la decisión de destruir todo y ahora se intenta conservar en lo posible los caminos ya existentes. Así pues, veamos cómo actúa esta versión con dichos conflictos partiendo de la topología, con una comunicación ya existente, que se muestra a continuación:

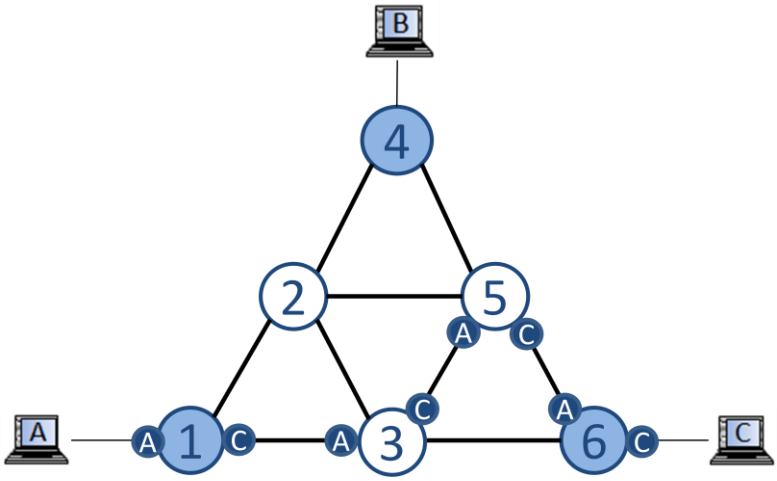


Figura 55. Otro ejemplo de comunicación estable entre un par de hosts con ARP-Path

Descubrimiento y generación del camino hacia el origen con una comunicación ya existente conflictiva (*ARP Request* conflictivo): Cuando en el escenario anterior el host C emite un ARP Request, ya sea para renovar la caché de ARP o para empezar una comunicación con un nuevo destino, puede suceder que el nuevo camino descubierto para el host C sea diferente del anterior. En la versión anterior de ARP-Path simplemente se destruiría el camino previo, pero en esta se apunta la primera selección en un estado '*locked*', como entrada secundaria. Entonces, si una de las copias del ARP Request se recibe por el puerto del camino viejo antes de que ese estado '*locked*' pase a '*learnt*', la entrada secundaria se elimina y la primaria se refresca en 'locked', pero si no, la entrada primaria es la que se elimina (pues si no llega el broadcast por ella es que lógicamente no es válida, dado que el broadcast se propaga por todos los puentes de la red) y la secundaria pasa a ser la nueva entrada de camino aprendida. En todo caso, sólo se propaga la primera copia recibida, la más rápida (como se hacía antes), el resto sólo servirán para descartar posibles entradas secundarias, pero sin propagarse después. En la imagen siguiente vemos un ejemplo en el que se añaden nuevas entradas para C y hay un conflicto en el switch 3, que añade una entrada secundaria temporal marcada en verde, pero que acaba renovando la entrada vieja al llegar una de las copias del ARP Request y borrando la nueva entrada en verde, manteniéndose los caminos sin variar.

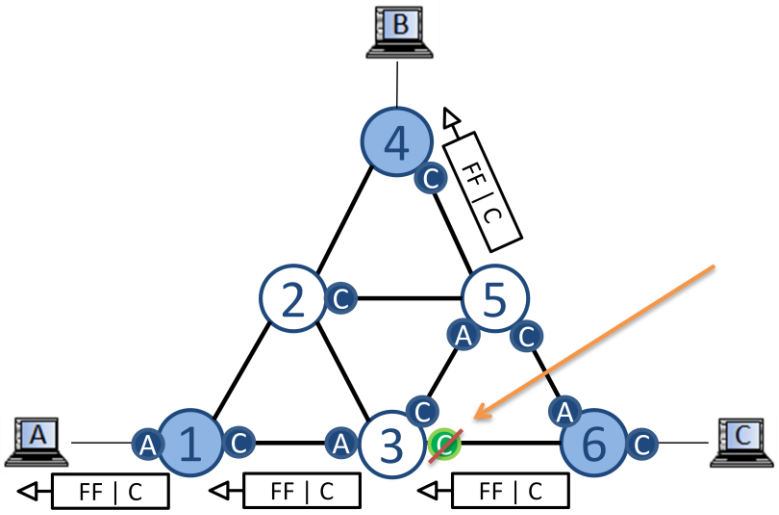


Figura 56. Conflicto de entrada en tabla con ARP Request y solución aplicada en ARP-Path

La diferencia con la versión anterior es evidente, se conserva el camino viejo. Pero quizás la diferencia con la primera versión (FastPath) no lo es tanto, dado que ésta también conservaba los caminos iniciales. Para ver en que se destingue de FastPath, hay que pensar que en esta versión los ARP Request siguen el camino más rápido y las copias lentas se descartan. Por ejemplo en el switch 3 se propaga la primera copia, aquella que genera la entrada secundaria, y las que llegan después se descartan, mientras que en FastPath sólo se propagaban las copias que llegaban por la entrada asignada y el resto, aunque fuera más rápidas, se descartaban, haciendo que el ARP Request se propagase más lento por la red porque muchas veces los enlaces con entradas asignadas están en uso y son más lentos. No sólo eso, sino que en esta última versión si el camino anterior ya no está disponible, da la oportunidad de crear uno nuevo sin coste de reparación adicional.

Generación del camino hacia el destino con una comunicación ya existente conflictiva (*ARP Reply* conflictivo): En este caso para el ARP Reply, las entradas en conflicto nunca se modifican. Imaginemos el mismo escenario anterior en el que ahora entra en juego un host B que quiere comunicarse con A. Éste emite un ARP Request y luego A le responde con un ARP Reply. Como se ve en la siguiente figura, el puente 5 tiene un conflicto porque recibe el ARP Reply por otro puerto. En la primera versión (FastPath) se aplicaría un mecanismo de prioridad (recordemos que los caminos han de ser simétricos, cosa que no es posible en este escenario de hecho), en la segunda versión se cambiaría el puerto modificando el tráfico de C a A ya existente (con los problemas que conlleva), y en esta tercera versión no es necesario hacer nada, lo único es que este último camino creado no será simétrico. Alternativamente, si el ARP Reply se emitiera en broadcast, podría aplicarse lo mismo que con el ARP Request, creando entradas secundarias temporales por si la primera falla.

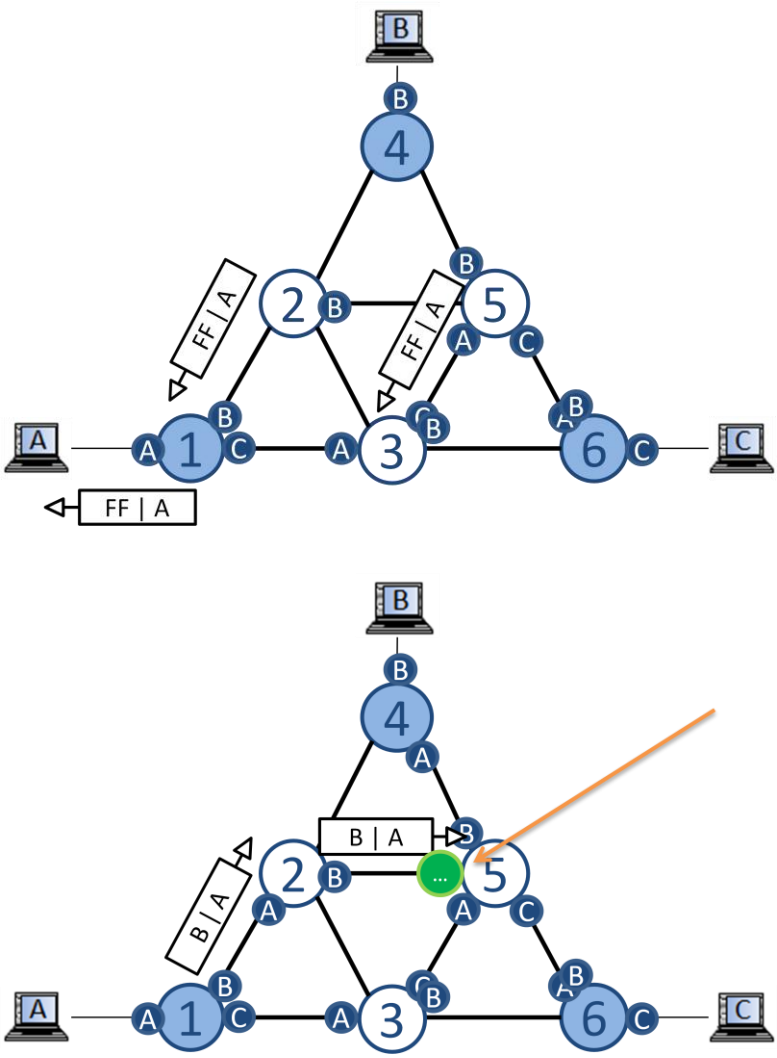


Figura 57. Conflicto de entrada en tabla con ARP Reply y solución aplicada en ARP-Path

Tras estas decisiones, los caminos quedarían como se muestra en la última figura. La comunicación entre A y C no varía y se mantiene simétrica con el camino 1-3-5-6, mientras que entre A y B el camino hacia B es 1-2-5-4 y hacia A es 4-5-3-1, asimétrico.

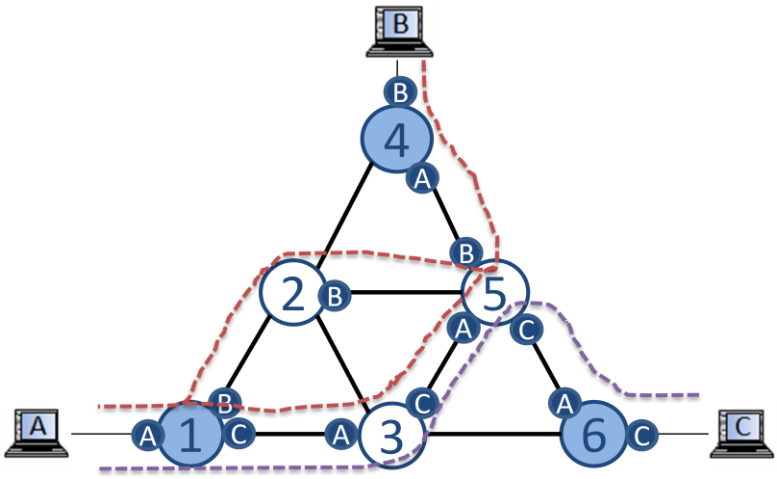


Figura 58. Estado de los caminos generados tras los conflictos con los mensajes ARP en ARP-Path

4.1.6 Reparación de caminos

A lo largo del capítulo se ha descrito la evolución del protocolo ARP-Path, los problemas que fueron surgiendo y las decisiones tomadas. Pero sin embargo, sólo se ha hablado de la creación inicial de los caminos, pero no de cómo "repararlos" (es decir, crear un camino alternativo) en caso de que algún enlace o nodo deje de funcionar.

Los métodos de reparación de caminos dependen en cierto modo de la versión del protocolo y fueron evolucionando con él, pero la base es la misma: cuando se descubre el fallo (de manera reactiva con una trama de datos), generar un mensaje de la misma naturaleza que el ARP para volver a descubrir y crear el camino, y es por ello que no sólo ARP-Path realiza esto, sino que las variantes del protocolo que veremos a continuación (Flow-Path, ARP-Path asterisco y ARP-Path multipath) no difieren casi nada de esta idea salvo por las características específicas de la variante.

Dentro de estos métodos, observaremos que hay distintas versiones, que se pueden utilizar indistintamente y cada una tiene sus ventajas e inconvenientes. También se hablará de casos concretos en los que no se necesita ningún mensaje especial de reparación. Pero concretamente, dentro de esta apartado, se describirá la evolución de los métodos de reparación y las ideas que fueron surgiendo y que fueron descartadas y no, para exponer finalmente el método que debería aplicarse en la última versión del protocolo: ARP-Path unidireccional alternativo/retardado.

4.1.6.1 Reparación de caminos en ARP-Path (FastPath)

Esta versión del protocolo aún no tenía limitado el aprendizaje de caminos a los mensajes ARP, inicialmente el aprendizaje se extendía a cualquier mensaje broadcast, porque todavía no se habían explorado los posibles problemas.

La idea principal de reparación es una reparación reactiva. Es decir, cuando un enlace falla, las entradas asociadas al puerto asociado se eliminan, entonces cuando llega una nueva trama de datos y no encuentra el puerto de salida es que algo ha fallado y se inicia la reparación. Es entonces cuando se tiene que conseguir avisar al puente frontera del origen para que genere un nuevo mensaje similar al ARP Request y que llegue al puente frontera del destino y éste genere uno similar al ARP Reply. Es decir, un mensaje broadcast y uno unicast (y han de ser "especiales" para poder reconstruir el camino que es fijo para el resto de tráfico "normal"), más el que avisa al puente frontera origen desde el lugar del fallo. Éste último mensaje se pensó como broadcast, unicast… pero al final ninguna de estas ideas llegaron a madurar al completo al tener diversos problemas el protocolo, como ya se conoce.

4.1.6.2 Reparación de caminos en ARP-Path unidireccional

Conocido el hecho de que no se pueden crear caminos simétricos para cada flujo si sólo se aprende la dirección MAC, es cuando aparece la versión ARP-Path unidireccional y es aquí donde se desarrollan los principales métodos de reparación. Además nace una nueva idea, como ahora los caminos son únicos por hosts (y no van combinados como origen/destino como en la anterior versión), quizás basta con reparar sólo uno de los sentidos. Es decir, si una trama va hacia cierto destino y su camino ya no es válido, quizás sólo es necesario redescubrir el camino hacia dicho destino, pues el camino hacia el origen ni siquiera hemos de dar por supuesto que se esté utilizando.

Entonces, en esta fase de evolución de ARP-Path, se desarrollan dos ideas de reparación, la denominada reparación "hacia atrás" (que regresa "hacia atrás", hacia el origen, para repetir dos procesos análogos al ARP Request y ARP Reply) y la reparación "hacia adelante" (que se envía "hacia adelante", hacia el destino, para realizar sólo un proceso análogo al ARP Request para el destino, obviando el redescubrimiento del origen).

A continuación se hablará de ambas por separado y de las ventajas e inconvenientes de cada una, en un tercer apartado se hablará concretamente de una posible simplificación de uno de los métodos de reparación, y finalmente, en un cuarto apartado se describirán las tablas y los mensajes necesarios para todo el proceso de reparación.

Reparación "hacia atrás" (o "hacia el origen")

El proceso de reparación "*hacia atrás*" se realiza en 3 pasos como ya se introducía con FastPath. El primer paso es regresar al puente frontera del origen, el segundo es repetir un proceso análogo al ARP Request y el tercero uno análogo al ARP Reply. Queda claro que el segundo será emitir en broadcast y que el tercero podría ser en broadcast, pero en unicast es igual de efectivo y ahorramos mensajes. La duda inicial quedaba en si el primer mensaje bastaba con ser unicast o debía ser broadcast para propagarse por toda la red para poder descubrir el puente origen.

Inicialmente se pudo pensar que el primer mensaje podía ser transmitido "*hacia atrás*", pues al fin y al cabo éste viene enviado desde el origen y sería tan fácil como retransmitir ese mensaje hacia atrás (*loopback*) por el camino por el que vino. Sin

embargo, esta idea sólo es factible en los casos en los que el camino es simétrico (no en ARP-Path) o si no es muy costosa, pero se analizará en su apartado específico. Por lo tanto, en la práctica ese mensaje que va hacia el origen, debe se emitido en broadcast para poder encontrarlo. A este mensaje que busca un puente frontera de un host concreto, en este caso el host origen, le denominamos *PathFail*.

A continuación se muestra un ejemplo, en el que falla el enlace entre los puentes 2 y 3, que contenían el camino entre A y C, y entonces una trama de A a C provoca una reparación "hacia atrás" que comienza con un *PathFail* buscando el puente frontera de A.

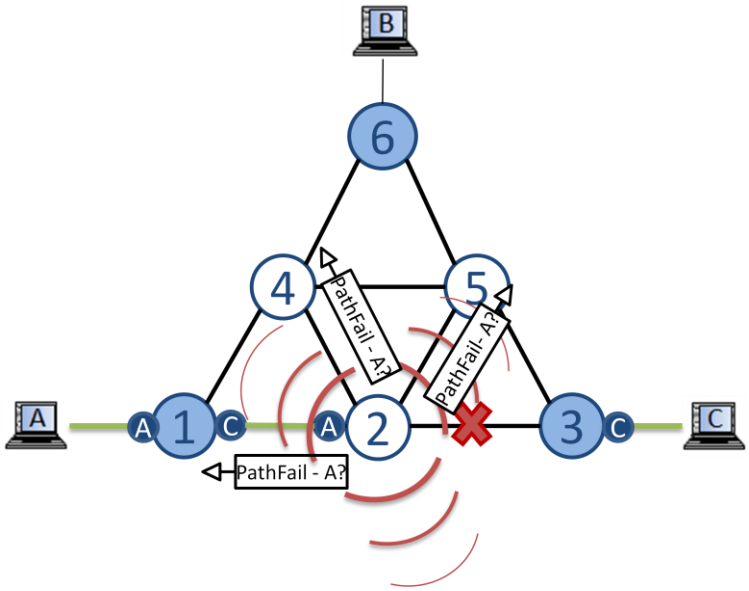


Figura 59. Emisión de un *PathFail* en la reparación "hacia atrás" de ARP-Path

Tras el *PathFail* que descubre el puente frontera del origen A, se emite un *PathRequest* del puente frontera del origen al puente frontera del destino, que se responderá desde este último puente al primero con un *PathReply*. Como su nombre indica, estos dos mensajes actúan de forma análoga a un ARP Request y un ARP Reply, la diferencia es que los mensajes de reparación tienen prioridad mayor.

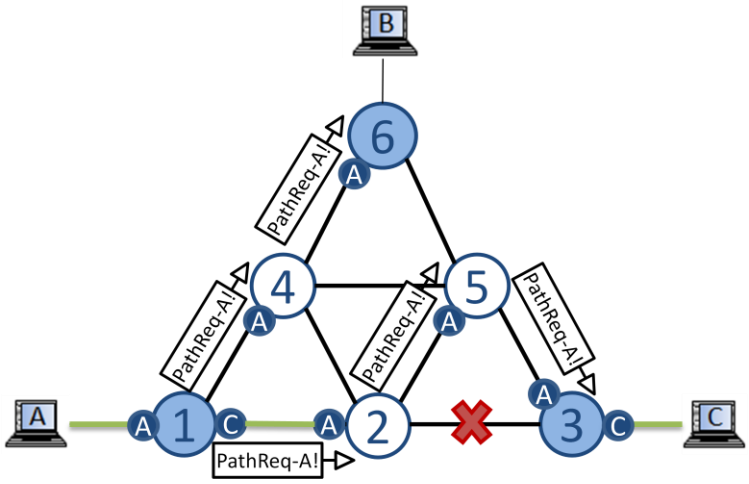


Figura 60. Emisión de un *PathRequest* en la reparación "hacia atrás" de ARP-Path

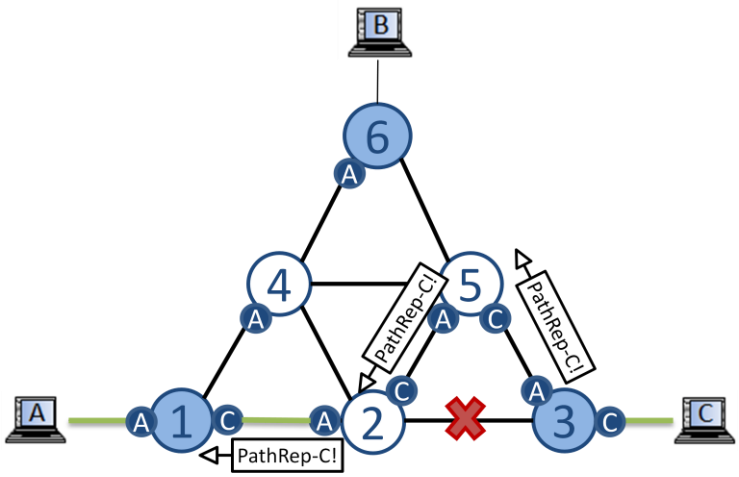


Figura 61. Emisión de un *PathReply* en la reparación "hacia atrás" de ARP-Path

¿Ventajas e inconvenientes? La ventaja inicial es que se reparan dos caminos con tres mensajes y además, en caso de que el primer mensaje pudiera ser unicast, serían menos mensajes incluso que el próximo método (pero de momento no se ha podido probar que pueda ser unicast sin un alto coste). Los inconvenientes son que es demasiado trabajo de mensajes si quizás el camino del origen no es necesario que sea reparado (ya que por ejemplo no se usa o sí se usa, pero por su camino no está caído al ser asimétrica a veces la comunicación), pero también si el camino origen es necesario que sea reparado, porque el método completo se aplicará dos veces si la comunicación es bidireccional (una en cada sentido), a no ser que se añada una tabla de reparación, *Repair Table* (RT), que frene las reparaciones innecesarias, realizando sólo las primeras en cierto periodo de tiempo con un temporizador.

Reparación "hacia adelante" (o "hacia el destino")

El proceso de reparación "*hacia adelante*" se realiza en 2 pasos, al contrario de los 3 del proceso anterior, pero no repara ambos caminos (hacia origen y hacia destino), sólo el camino hacia el destino. El primer paso es avanzar hasta el puente frontera del destino y el segundo es repetir un proceso análogo al ARP Request. El hecho de no se responda después con el ARP Reply es precisamente que este proceso pretende ahorrar ese último mensaje y por eso se busca el destino directamente, además como el refresco es hacia delante también, no es necesario confirmar con ninguna trama más el camino redescubierto con el mensaje similar al ARP Request.

En este método, cuando una trama detecta el camino incompleto, se intenta localizar directamente el puente del host destino y reconstruír sólo este camino. Inicialmente se puede pensar que el destino "estás más lejos" o que "es más difícil conocer su localización" si no es inundando en la red, pero en la práctica el destino puede estar más cerca o lejos del camino según sea la topología y el enlace que se haya caído (casi podríamos decir que en una proporción del 50% de los casos más cerca y el resto más lejos), y además para localizar el puente del origen ya vimos que al final también había que inundar la red. Así pues, este primer mensaje es idéntico para ambos métodos, es un *PathFail* que inunda la red, sólo que aquí se busca el switch frontera del host destino.

En las siguientes figuras vemos el ejemplo anterior, pero aplicando este segundo método. Ahora el switch 2 emite un *PathFail* en busca del puente frontera de C (y no A como antes) y a continuación dicho puente emitirá un *PathRequest* para reconstruir todos los caminos hasta C, mientras que no se responde con ningún mensaje adicional como antes.

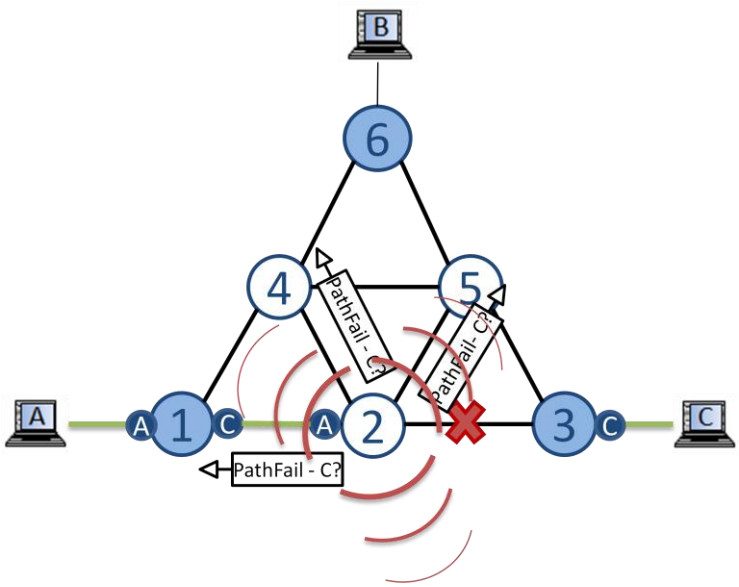


Figura 62. Emisión de un *PathFail* en la reparación "hacia adelante" de ARP-Path

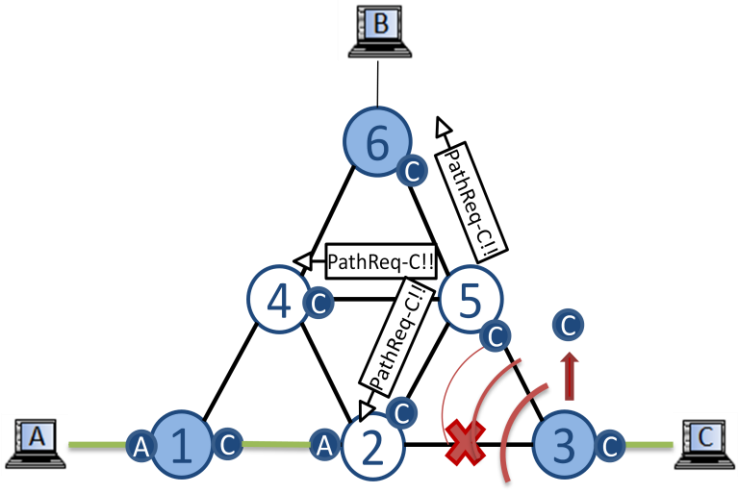


Figura 63. Emisión de un *PathRequest* en la reparación "hacia adelante" de ARP-Path

¿Ventajas e inconvenientes? En la práctica este método es más conveniente que el anterior, puesto que nos ahorramos el último mensaje y además, como ya se comentaba anteriormente, no es necesario reparar el camino hacia el origen porque si no se usa o su camino no es fallido, no es necesario y si se necesita reparar, ya se encargará la trama que lo vaya a usar de repararlo.

Loopback de tramas

Finalmente, para el caso del *PathFail* en unicast, es decir, evitar el uso del *PathFail* realizando un *loopback* (devolver hacia atrás) de las tramas que no

encuentran un camino y así evitar la inundación de la red con este mensaje en el método de reparación "hacia atrás", ya se comentaba anteriormente que inicialmente se había descartado su uso por ser poco viable, aunque al ser una idea relativamente novedosa en ARP-Path todavía está en fase de exploración. En este apartado se analiza su funcionamiento.

La idea es que una trama que viene de cierto origen A y va hacia un destino B quizás debería conocer su camino de vuelta a A. Así en el caso de que no haya entrada para B en cierto puente y se necesite reparación, y se esté usando el método de reparación hacia el origen, podría devolverse esa trama por el puerto por el que entró. Entonces el puente anterior recibiría la trama de A a B por un puerto asociado a B, lo que no es posible si no se trata de una trama que está siendo devuelta (*loopback frame*) y seguirá enviando la trama hacia atrás por la entrada asociada a A. Sin embargo, existen dos problemas con este concepto debido a que ARP-Path no genera caminos de comunicación simétricos:

1. **Caso en el que no hay entrada para el origen**: En el siguiente ejemplo de la figura hay asimetría entre los caminos hacia A y hacia B. Si una trama que va de A a B encuentra el enlace del switch 3 al 6 caído, no tiene entrada de vuelta hacia A en ese switch, aunque puede devolver simplemente la trama por donde entró. Al llegar al switch 2, éste sabe que es una trama devuelta (porque el destino es B y entrada por el puerto asignado a B), pero no sabe cuál es el camino hacia A, ni tiene manera de saberlo, a no ser que lo hubiese apuntado antes.

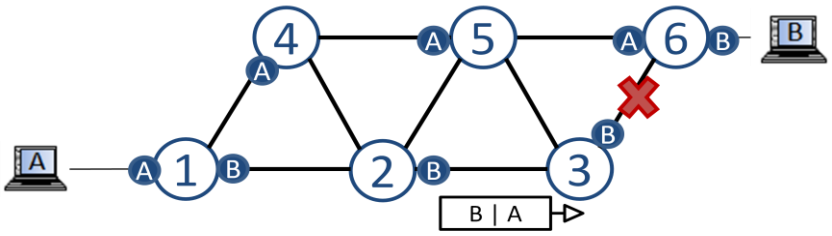


Figura 64. Loopback de tramas en ARP-Path – Caso en el que no hay entrada para el origen

2. **Caso en el que no hay entrada para el destino**: Ahora imaginemos el ejemplo de la figura siguiente. La trama de A a B llega al switch 5, que lo devuelve hacia atrás al 2. Al llegar al switch 2, éste sabe que es una trama devuelta y la envía por la entrada asignada a A, llegando al 4. Cuando el switch 4 la recibe, no tiene forma de saber ahora que es una trama devuelta, pues no tiene entrada para B.

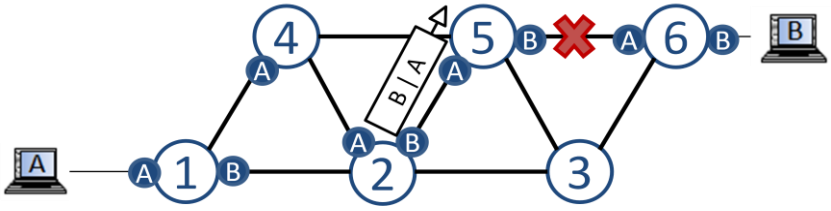


Figura 65. Loopback de tramas en ARP-Path – Caso en el que no hay entrada para el destino

Una solución sería tratar la trama como la reparación en el switch 5, es decir, se busca la entrada para B, pero como no existe, se devuelve hacia atrás por el puerto asociada a la entrada de A. Sin embargo, si hacemos esto, no hay manera de diferenciar esta trama de una que realmente necesita reparación, entonces se añadiría una entrada a la tabla de reparación RT sin ser ésta necesaria. Y no queda ahí, sino que al llegar al switch 1, no entra por el puerto asociado a B, que esta vez sí existe, y no hay manera de diferenciar esa trama *loopback*, de otra trama de datos normal, por lo que se mandaría por el puerto asociado al destino a B, y nunca llegaría al origen.

La mejor solución es, una vez más, apuntar temporalmente el camino de vuelta hacia A, que será simétrico al de B (porque la trama está siguiendo el camino hacia B) y que no tiene por qué coincidir con las entradas de camino hacia A.

En el caso de decidir apuntar temporalmente entradas asociadas al origen, debido a los dos problemas anteriores causados por la asimetría de caminos, es posible que se generen bucles si en las entradas temporales sólo se asocia la dirección origen y no algo más característico del flujo, como la dirección de destino también. Esto es muy costoso, porque quiere decir que estaríamos generando un número de entradas temporales en una nueva tabla igual al número de flujos activos, lo que es un número bastante mayor al número de hosts activos, que son las entradas que se apuntan actualmente en ARP-Path. Además, una de las variantes del protocolo que se estudiará a continuación, Flow-Path, ya apunta entradas por cada flujo y genera caminos simétricos, así que bastaría con utilizar Flow-Path en vez de ARP-Path.

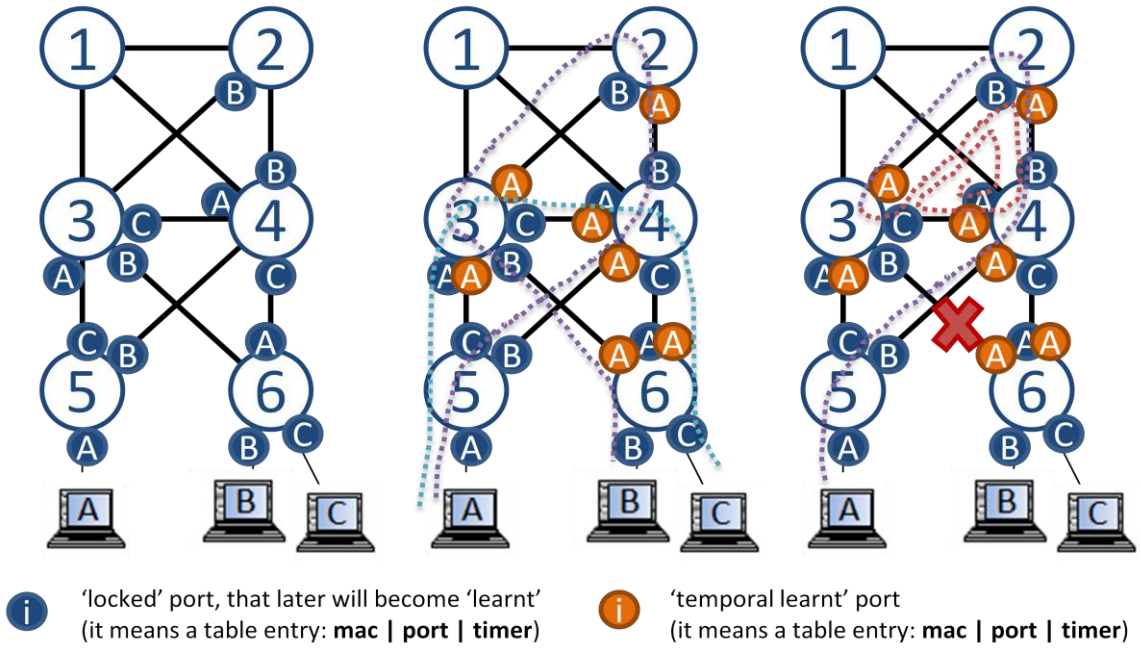


Figura 66. Loopback de tramas en ARP-Path – Caso de bucle si sólo se apunta temporalmente la dirección origen

En la figura anterior se ve el posible problema de bucles. A la izquierda vemos los caminos para los hosts A, B y C. En el centro vemos la comunicación de A a B marcada en morado y de A a C en azul, y las entradas temporales apuntadas para A en ambos casos. Si sólo se apunta A, se puede dar el bucle que se muestra a la derecha, una trama va de A hacia B y pasa por el switch 5, 4, 2 y en 3 encuentra el fallo de enlace. Entonces se envía de vuelta por la entrada temporal hasta 2, que la envía por su entrada temporal a 4. Pero en 4, existen dos entradas temporales y si se coge una al azar, puede cogerse la que vuelve a 3, y entonces crear un bucle al volver a 2 y de nuevo a 4, que volvería a elegir. Es un bucle finito si ante varias entradas se escoge una al azar, pero un problema al fin y al cabo, que se soluciona añadiendo el destino en las entradas, que sería B y entonces no habría casos de múltiples entradas, ni de bucles, pero como ya se sabe, añadir una entrada por cada flujo activo es demasiado costoso y para eso lo rentable sería utilizar directamente Flow-Path, que se verá en una sección posterior.

Tablas y mensajes necesarios para la reparación en ARP-Path

En todo este proceso de reparación (dividido en dos métodos), los mensajes sólo llegan al puente frontera para no afectar en nada al funcionamiento de los hosts, pero para ello los puentes frontera necesitan saber que son frontera de dichos hosts. Es por esto que se añaden un par de tablas más a las ya existentes *Learning Table* (LT) y *Broadcast Table* (BT), que son la tabla de vecinos, *Hello Table* (HeT) y la tabla de hosts, *Host Table* (HoT). Estas dos tablas son complementarias, HeT se aprende en base a mensajes *Hello* que se intercambian entre los bridges ARP-Path y HoT se aprende para aquellos puertos que reciben tráfico de datos, pero no mensajes *Hello*, y que quiere decir que dichos puentes son los frontera de los hosts origen que envían el tráfico.

Por otro lado, el proceso de reparación también necesita una tercera tabla, *Repair Table* (RT). La función de esta tabla es evitar las múltiples reparaciones. Esto es porque la primera trama que descubra el camino iniciará una reparación del camino, pero mientras ésta se está realizando es posible que lleguen otras tramas y se encuentren el mismo problema e inicien más reparaciones. Para evitar esto último, se añade una entrada a la RT con cierto temporizador en el switch en el que se inicia la reparación y así se bloquearán las múltiples reparaciones. Adicionalmente también se pueden añadir entradas de RT en los puentes frontera para que no se repare más de una vez el mismo camino en el caso de que se intente reparar en las dos direcciones, como se comentaba en la desventaja del método de reparación “hacia atrás”.

En cuanto a los tipos de mensajes especiales necesarios, se podrían resumir en: mensajes *Hello* (para definir puentes frontera y no), *PathFail*, *PathRequest* y *PathReply* (éste sólo utilizado en el primer método). Todos estos mensajes tienen como MAC destino una dirección multicast específica para ARP-Path (*AllPathBridgesMcast*) y como origen la MAC la dirección del switch que emite el mensaje, aunque en la práctica este campo podría ser cualquiera, dado que si la dirección multicast es *AllPathBridgesMcast* estará claro que el mensaje lo ha creado un bridge ARP-Path y no es necesario tampoco saber cual. Dentro del mensaje estan encapsulados tres campos: tipo (Hello = 1, PathFail = 2, PathRequest = 3, PathReply = 4), “dirección a reparar” (puede ser el origen o destino de la trama) y “segunda

dirección a reparar" (que será la otra dirección, destino u origen respectivamente, y se añade por si se está utilizando el método de reparación "hacia adelante" y se necesita reparar también su camino). Concretamente en las implementaciones realizadas, además se encapsularon las tramas en una trama LLC con los campos DSAP = SSAP = 0x43 y el campo de control = 0x03, en vez de utilizar el campo Ethertype de la trama Ethernet.

Figura 67. Campos en un mensaje especial de reparación ARP-Path (Destino de la trama Ethernet = AllPathBridgesMcast, Origen en el bridge, Tipo 1, 2, 3 o 4, y las dos direcciones a ser reparadas – aunque sólo se repara la primera en el segundo método de reparación)

La necesidad de los mensajes especiales radica en el hecho de que los caminos en ARP-Path, inicialmente (FastPath), se fijaban con el primer ARP y luego nuevos mensajes ARP eran descartados para aprendizaje. Sin embargo, en esta segunda versión todos los caminos se reconstruyen con cualquier ARP, por lo que en la práctica podrían usarse los mismos métodos de reparación, sustituyendo el *PathRequest* por un ARP Request creado en el puente frontera del origen, que se respondería directamente con un ARP Reply por parte del host, no siendo necesario tampoco el *PathReply* y conservando el intercambio de mensajes *Hello* y el mensaje *PathFail*, que ahora debería portar también las direcciones IP de los hosts (para generar el ARP). Utilizar directamente mensajes ARP disminuye el procesamiento requerido para los mensajes especiales *PathRequest* y *PathReply*, pero a cambio necesita que el mensaje *PathFail* tenga más campos (para guardar las IPs). En el siguiente apartado se hablará de qué suponen todas estas posibilidades para la reparación.

4.1.6.3 Reparación de caminos en ARP-Path unidireccional alternativo/retardado

Esta tercera versión varía en un pequeño matiz respecto a la anterior, así que es importante ver ahora en qué afecta este matiz a todo lo definido en el apartado anterior.

Como ya se sabe, el matiz es que ya no se reconstruyen los caminos por completo y se debe evitar modificar los caminos existentes en lo posible. Así pues, el uso del *PathFail* no varía, así como los mensajes *Hello* intercambiados necesarios para que el *PathFail* llegue a destino, pero los mensajes de generación de caminos *PathRequest* y *PathReply* sí deberán cambiarse, pues no deben modificar los caminos existentes en lo posible. Analicemos cómo deberían cambiar partiendo del aprendizaje que se realiza con los mensajes ARP en esta versión de ARP-Path.

PathRequest

Si el *PathRequest* es idéntico al ARP Request, existe posibilidad de bucles a la hora de transmitir los datos en unicast. Para verlo mejor se ilustra el problema a

continuación. Si se cae el enlace entre 1 y 3, se elimina el camino hacia A y se quiere recuperar, imaginemos que se emite un ARP Request. En ese caso, el mensaje seguirá el recorrido de la línea punteada en verde, añadiendo nuevas entradas (A verde) y entradas alternativas (a verde) en aquellas en las que ya existe una. Entonces al llegar al switch 4, el mensaje broadcast dejará de propagarse, pero refrescará la entrada azul (antigua), creando un bucle para cualquier trama de datos que vaya hacia A, como se puede observar en la figura.

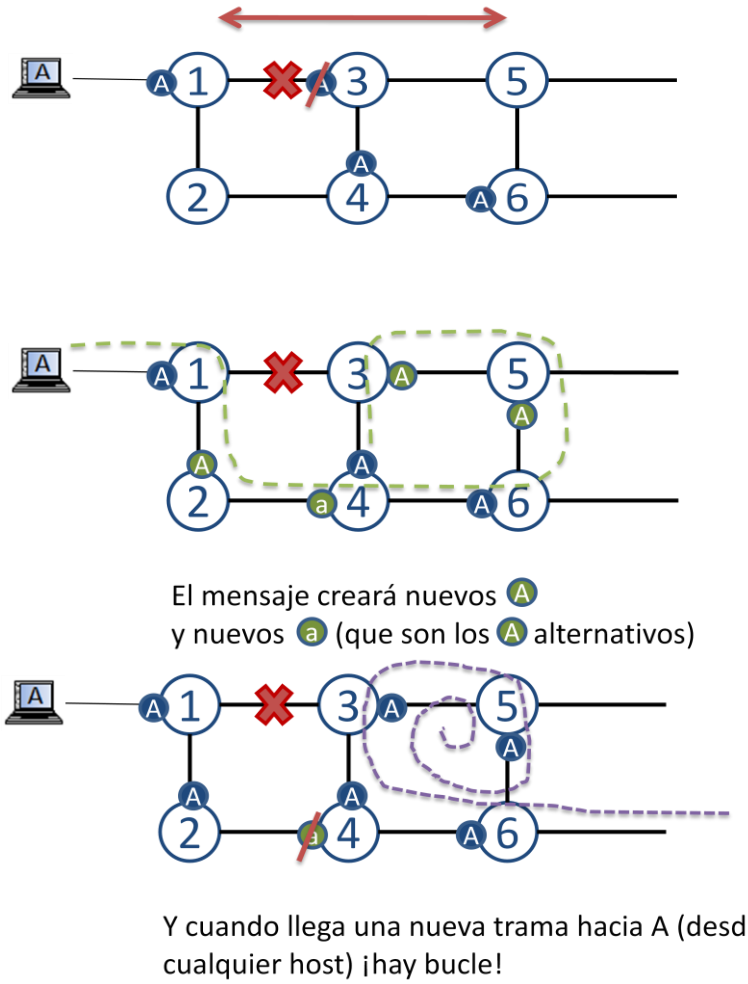


Figura 68. Problema de bucle en ARP-Path si el PathRequest es idéntico al ARP Request en aprendizaje

Este problema surge del hecho de que se está refrescando una entrada que dependía del camino que acaba de perderse. Para solucionarlo, hay que modificar el *PathRequest* respecto al funcionamiento del ARP Request, o cambiar directamente el procesamiento del ARP Request si se quiere utilizar en su lugar. El cambio consiste en dos cosas:

1. Cuando un enlace se cae, la entrada asociada al puerto no se elimina como se hace siempre, sino que mantiene con un flag o estado especial de "puerto caído".

2. Entonces, cuando la trama (*PathRequest* o ARP Request) está inundando la red, al llegar al switch que contiene esa "entrada caída" (el número de

switches con entradas de este tipo será igual a dos por cada enlace caído), realiza su aprendizaje normal, pero no se propaga más, aunque sea la primera copia.

Con estos dos cambios, el switch 3 no enviaría la copia al switch 4 (ni a ningún otro que dependa del camino viejo, que es lo que puede causar bucles al fin y al cabo) y no se refrescaría la entrada antigua, renovándose con la entrada alternativa apuntada.

La solución más básica sería que el *PathRequest* reconstruyese todo el camino por completo, ignorando los caminos viejos. Pero los dos cambios necesarios para evitar bucles tampoco suponen un coste excesivo, sólo guardar algunas entradas "caídas" de vez en cuando y, de hecho, se inunda algo menos. En todo caso esta propuesta con dos cambios necesita un análisis en profundidad.

PathReply

Si el PathReply es idéntico al ARP Reply, sería unicast y no exploraría todos los caminos, por lo que habría que tomar la decisión de conservar o no un camino con un mensaje unicast y esto puede llevar a una mala decisión, ya que con un mensaje de este tipo es imposible saberlo a ciencia cierta. Es necesario recordar que el mensaje PathReply sólo es necesario con el método de reparación "hacia atrás", no con el de "hacia adelante".

En las figuras siguientes se muestra un ejemplo. Se cae el enlace entre 2 y 3 y llega una trama al 3, que vuelve hacia atrás y con un mensaje en broadcast se recupera el camino hacia C, como se ve en la segunda imagen (aparece un alternativo en rojo y se borra el antiguo). Cuando desde el host A se responde con el mensaje en unicast, al llegar al switch 4 tiene que hacer una decisión (marcada en naranja): dejar la entrada antigua de A o actualizarla con la nueva opción. En el ejemplo de la tercera figura se ve claro que esa entrada no es válida y debería ser eliminada, pero la cuarta figura hace referencia a un caso análogo en el que la entrada de A sí sigue siendo válida. Entonces se crea un dilema, pues si no se cambia y no es válida, el camino a A necesitaría aún repararse, y si se cambia y sí es válida, el camino a A se modifica innecesariamente, así como quizás flujos en uso, con los problemas que de ello deriva y ya se conocen.

La solución entonces es que el *PathReply* (o el ARP Reply, de usarse éste último) sea emitido en broadcast de manera análoga al *PathRequest*, para explorar todos los caminos y saber a ciencia cierta qué caminos se han de modificar y cuáles no. Sin embargo, el hecho de que este mensaje necesite propagarse por toda la red hace que el método de reparación "hacia atrás" sea aún más costoso y se den más razones de usar la reparación "hacia adelante" en su lugar.

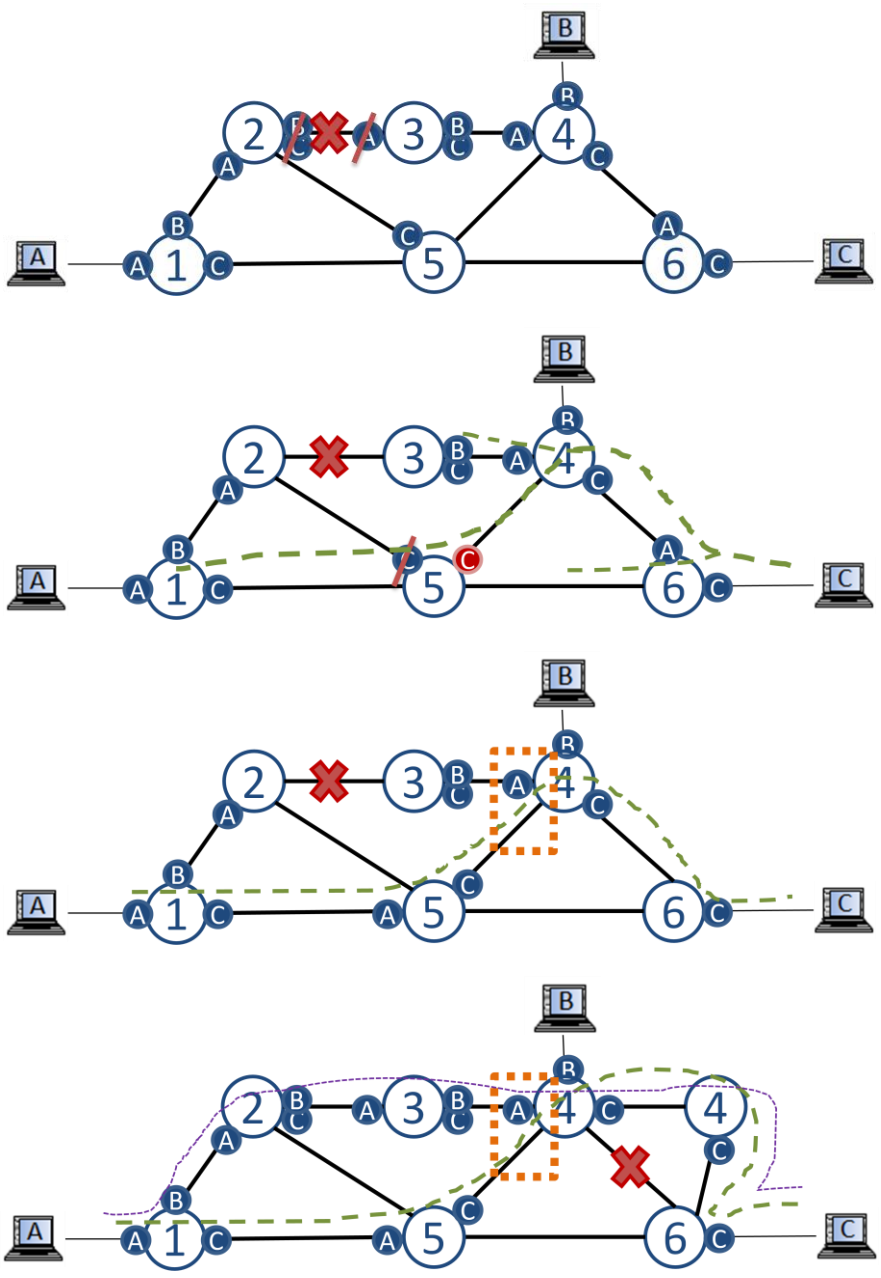


Figura 69. Problema de mala decisión en ARP-Path si el PathRequest es idéntico al ARP Reply en aprendizaje

4.1.7 Características de ARP-Path

En este último apartado de la sección de ARP-Path se realizará un análisis de las características principales del protocolo ARP-Path, que no sólo se presenta como una nueva y auténtica alternativa de *bridging* (frente a SPB o TRILL, que tendían más hacia el *routing* como ya se comentaba previamente), sino que comienza la ideología base de los protocolos de la familia All-Path, que en posteriores secciones se completará con Flow-Path, ARP-Path "asterisco" y ARP-Path multipath.

4.1.7.1 Autoconfiguración

Los puentes ARP-Path no requieren ninguna configuración previa. Es posible configurar opcionalmente los parámetros de los temporizadores que definen el tiempo de duración de las entradas en las tablas y el periodo de intercambio de mensajes *Hello* entre los puentes, pero no es obligatorio. En las diferentes implementaciones realizadas, estos parámetros estaban prefijados con los siguientes valores:

- LT (*Learning Table*) '*lock time*' = 1 segundo
- LT (*Learning Table*) '*learnt time*' = 120 segundos
- BT (*Broadcast Table*) = 1 segundo (misión análoga a LT '*lock time*')
- RT (*Repair Table*) = 1 segundo
- HoT (*Host Table*) = 120 segundos (relacionado con LT '*learnt time*')
- HeT (*Hello Table*) = 11 segundos (ligeramente mayor a *Hello frequency*)
- *Hello frequency* = 10 segundos

En ningún caso tuvo que modificarse ninguno y el funcionamiento siempre fue correcto, pero se deja la posibilidad de realizar un ajuste fino y modificarlos. Por ejemplo, si los flujos tienen un intercambio de mensajes muy lento, podrían subirse los parámetros de LT '*learnt time*' y HoT para que no caduquen las entradas, o si el intercambio de mensajes es rápido, se podrían bajar sus valores para que las entradas en desuso caducasen rápidamente y dejasen espacio para nuevas. También el valor de LT '*lock time*' podría incrementarse si el tamaño de la red es muy grande, pero normalmente 1 segundo es un valor alto de por sí.

Por lo tanto se puede decir que los puentes ARP-Path son verdaderos dispositivos *plug-and-play* dado que basta formar la red deseada y encenderlos, para que las comunicaciones entre hosts puedan comenzar, sin necesidad de configurar nada, ni de esperar en el inicio a ningún mecanismo de intercambio de mensajes a que converja.

4.1.7.2 Caminos de baja latencia

La segunda característica de ARP-Path es que con su mecanismo genera caminos de baja latencia, es decir mínimos si se considera como coste la latencia de los enlaces, y además estos se crean justo en el momento previo al comienzo de la comunicación por lo que tienen en cuenta el estado actual de la red a la hora de ser generados. Esta es otra diferencia principal frente a SPB y TRILL, que crean los caminos en base a un coste y no tienen en cuenta, en cada momento, el estado de la red para crearlos, pudiendo darse casos puntuales en los que ciertos enlaces estén muy cargados y otros no, dado que se eligieron como mínimos, pero no eran de mínima latencia en el momento de iniciar las comunicaciones.

Estos caminos además se realizan entre hosts en la versión inicial de ARP-Path, pero podrían realizarse variantes con encapsulado MACinMAC (ARP-Path M) y con VLANs (ARP-Path V), de manera análoga a SPBM y SPBV, respectivamente.

A partir de esta característica, es importante matizar sobre algunos puntos:

Caminos indeterminados

Es evidente que dado que los caminos se generan en cada momento, estos serán indeterminados. Mientras que en SPB y TRILL es posible realizar un cálculo "a mano" de las rutas una vez conocida la topología, en ARP-Path se puede estimar qué posibles caminos tendrás más o menos posibilidades de ser escogidos, pero nunca se puede asegurar cuál será el escogido al final. Esta incertidumbre puede parecer inicialmente una desventaja, pero en la práctica lo importante es saber que siempre se escoge el camino mínimo de menor latencia, mientras que en SPB y TRILL no se puede garantizar que en todo momento el camino elegido sea el de menor latencia, dado que estos últimos no tienen en cuenta el estado de la red.

Baja latencia en un sentido de la generación

Cuando hablamos de baja latencia en ARP-Path es necesario matizar un punto al respecto. Como es conocido, ARP-Path tiene dos fases (una por cada mensaje ARP), pero sólo la primera de ellas (ARP Request) realiza la exploración del camino de latencia menor, pues la segunda (ARP Reply) se limita a seguir el camino ya marcado. Esto quiere decir que aunque ARP-Path crea a la vez los caminos hacia el origen y hacia el destino, sólo es un camino de baja latencia hacia el destino, pero no tiene por qué serlo también en dirección contraria, por ejemplo si existe asimetría de ocupación en los enlaces.

Además, en la versión más actualizada de ARP-Path no se reconstruyen cada vez los caminos. Esto quiere decir que aunque se realice una nueva exploración con ARP-Path y exista un nuevo camino más rápido, no se modificará el inicial al completo, que quizás ahora es más lento.

Para demostrar este punto, a continuación se muestran resultados experimentales:

> Demostración de baja latencia en un sentido mediante emisión de tráfico UDP y TCP en la implementación de ARP-Path en NetFPGA:

A lo largo de la realización de la presente Tesis, se realizó una estancia de colaboración en el *Cambridge Computer Laboratory* [CCL], de la Universidad de Cambridge (Cambridge, Reino Unido), con el grupo de investigación de Jon Crowcroft [JC] y la colaboración de Andrew Moore [AWM]. Gracias a su ayuda, se pudieron realizar una serie de pruebas en topologías de hasta 9 tarjetas NetFPGA (dado que en nuestro laboratorio sólo disponíamos de 4) con la implementación NetFPGA realizada de ARP-Path unidireccional ya descrita en apartados anteriores.

En la siguiente figura se muestra el primer montaje realizado, que nos demuestra cómo ARP-Path construye caminos de baja latencia en un único sentido. Se trata de una topología en malla compuesta por 9 switches ARP-Path

(implementados en NetFPGA) en la que el switch 0 tiene un host conectado que hace de servidor y el switch 8 tiene otro, que contiene a su vez 8 máquinas virtuales que hacen de 8 clientes que solicitarán tráfico del servidor. Todos los hosts y máquinas virtuales utilizan sistemas Linux (principalmente Ubuntu 10.04) y todos ellos tienen IP asignadas dentro de la misma subred.

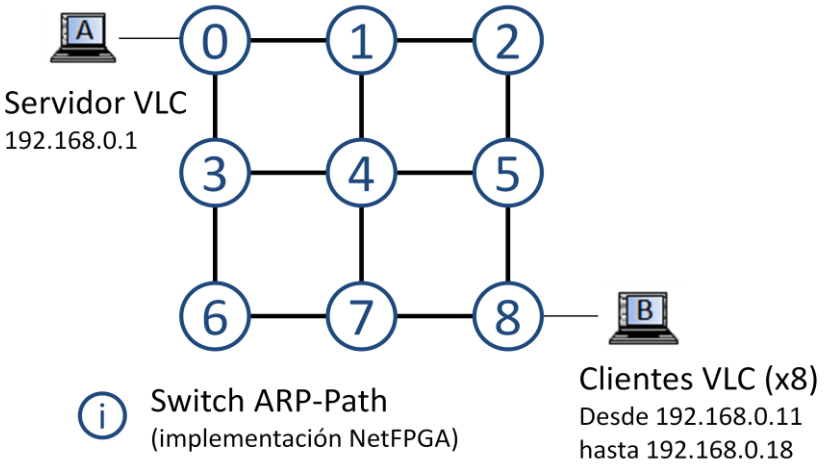


Figura 70. Topología en malla realizada en las pruebas con switches ARP-Path implementados sobre tarjetas NetFPGAs del Cambridge Computer Laboratory

Para demostrar que la baja latencia es en un sentido se realizan dos pruebas: emisión de tráfico UDP del servidor a los 8 clientes y emisión de tráfico HTTP/TCP del servidor a los 8 clientes, en ambos casos la emisión es de un capítulo de 30 minutos de una serie en formato *.avi con las propiedades de emisión por defecto que ofrece el programa VLC y el tráfico siempre va en el sentido de servidor a cliente en la red. Los resultados de distribución de tráfico son diferentes y se muestran y explican a continuación:

- **Emisión UDP**: Cuando se emite tráfico UDP, en el servidor VLC se introducen directamente las IPs de destino (clientes) y se comienza a emitir tráfico tras realizar una petición ARP para descubrir dicho destino en la red. Así pues la emisión de todas las peticiones ARP se realizarán por parte del servidor según se vayan incluyendo clientes, yendo el ARP Request desde el switch 0 al 8, y el ARP Reply en sentido inverso desde el 8 al 0; es decir, el ARP Request va en el sentido del tráfico y el ARP Reply en sentido inverso. Por lo tanto, y dado que es el ARP Request el que explora el camino de mínima latencia y éste coincide con el sentido de emisión del tráfico, la elección del camino se verá afectada por el tráfico ya presente en la red.

  Los resultados fueron los siguientes:

  - En el caso de emitir desde el servidor a todos los clientes a la vez, las peticiones ARP se realizaban todas al principio y a la vez, antes de emitir cualquier paquete de tráfico, por lo que todos los ARP Request encontraban la red vacía de tráfico y todos escogieron el mismo camino: 0-1-4-7-8, que era el camino más rápido en

hardware (es decir, compuesto por las tarjetas NetFPGA que en ese momento se encontraban con tiempos de procesamiento menores y los enlaces con menor tiempo de propagación) del switch 0 al 8. Se realizaron 2 repeticiones con 8 destinos cada vez, y en los 16 mensajes ARP Request emitidos escogieron en todas las ocasiones ese mismo camino.

- Sin embargo, en el caso de emitir desde el servidor a los clientes uno a uno, las peticiones ARP no se realizaban con la red vacía, por lo que el tráfico ya existente (creciente con cada nuevo cliente) afectaba a cada una de las peticiones ARP. En este caso, se repartía la carga por toda la red, dado que el camino de mínimo latencia era siempre uno diferente. De estos resultados se hablará más detalladamente en el apartado dedicado a reparto de carga de ARP-Path.

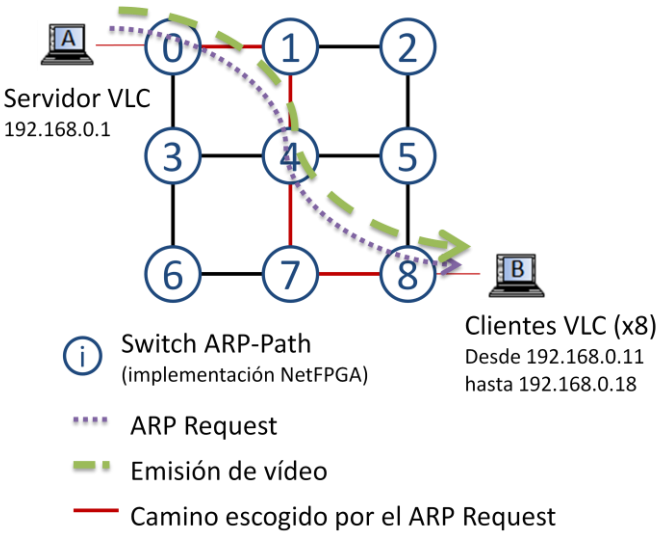


Figura 71. Camino hardware más rápido de A a B en la topología de malla de NetFPGAs del CCL

- **Emisión HTTP/TCP**: Cuando se emite tráfico HTTP, en el servidor no se indican los clientes sino que estos solicitan la emisión de tráfico individualmente. Así pues la emisión de todas las peticiones ARP se realizarán por parte de cada cliente según se les vaya indicando a cada uno el servidor, yendo el ARP Request desde el switch 8 al 0, y el ARP Reply en sentido inverso desde el 0 al 8; es decir, el ARP Request va en sentido contrario al tráfico y el ARP Reply en el mismo sentido, justo al revés que en el caso de emisión UDP. Por lo tanto, y dado que es el ARP Request el que explora el camino de mínima latencia y éste lleva sentido contrario al de emisión del tráfico, la elección del camino no se verá afectada por el tráfico ya presente en la red.

  En este caso, al margen de que todas las peticiones de los clientes se realizaran a la vez, o de una en una, el resultado fue el mismo: siempre se escogía el camino 0-1-4-5-8, que era el camino más rápido en hardware del switch 8 al 0. Se realizaron 2 repeticiones con 8 clientes, y siempre

fue el mismo en los 16 mensajes ARP Request emitidos. Esto es porque en este caso el ARP Request va en sentido contrario (de 8 a 0) al tráfico que se emite (de 0 a 8) y por tanto no se ve afectado a la hora de escoger el camino de mínima latencia, que en ese sentido no varía. Esto demuestra cómo ARP-Path genera caminos de baja latencia en un solo sentido: el del ARP Request que los explora.

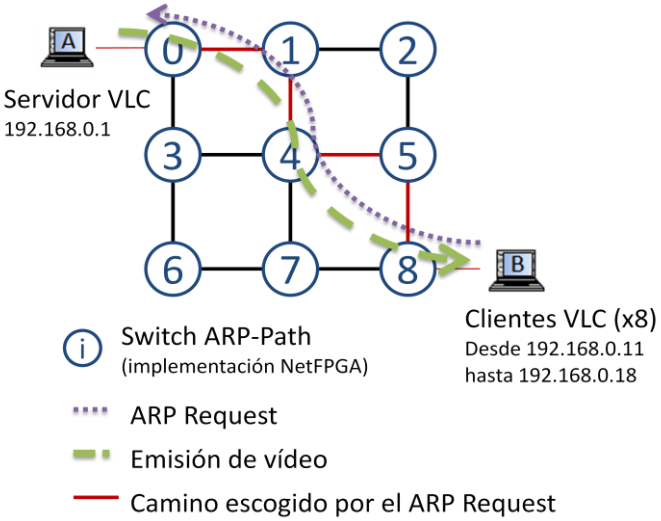


Figura 72. Camino hardware más rápido de B a A en la topología de malla de NetFPGAs del CCL

Nótese que éste es un caso concreto en el que el tráfico sólo sigue un sentido (de servidor a cliente) para demostrar el funcionamiento de ARP-Path, lo que no es habitual. En la práctica, dado que en una red los clientes y servidores se encontrarán dispersos en la red, y el tráfico no tendrá un sentido único, ARP-Path tiene un muy buen aprovechamiento de la topología y reparto de carga, como veremos en un apartado posterior.

También es importante tener en cuenta que esta prueba sólo podía realizarse en hardware, que es cuando el camino más rápido es uno solo al no haber caminos idénticos, aunque la diferencia sea mínima con otro camino. Si las pruebas se hubiesen realizado en un entorno de simulación, no habría diferencia hardware entre los caminos y sí habría caminos idénticos, por los que el simulador habría repartido entre unos y otros por una cuestión de probabilidades.

Baja latencia global de caminos

Pese al matiz anterior, ARP-Path garantiza en espacio (a lo largo de la topología) y en tiempo (a lo largo de los diferentes flujos emitidos), que la latencia global de los caminos escogidos para las comunicaciones será mínima. Pues en cada momento, en cada situación, se escogerá el camino de mínima latencia.

Para demostrar este punto, a continuación se muestran resultados experimentales:

Demostración de baja latencia global mediante la comparativa en emisión de tráfico UDP en la implementación de ARP-Path y SPB en OMNeT++:

A la hora de realizar las diferentes implementaciones de ARP-Path se vio la necesidad de realizar una comparativa con las tecnologías emergentes del mismo ámbito, y concretamente respecto a SPB (dado que los RBridges de TRILL están más cerca del *routing* que del *bridging*). Se pretendía realizar una comparativa de ambos protocolos en base al uso de una tecnología de implementación común, para que ésta no dependiese de la implementación sino sólo de su funcionamiento. Sin embargo, la tarea de realizar un switch ARP-Path comercial era muy complicada, así que se decidió realizar una comparativa en base al simulador OMNeT++.

Como es lógico, en OMNeT++ no existe una implementación de SPB (al menos a fecha de hoy, enero de 2013), ni siquiera del protocolo de enlace modificado IS-IS. Así que la decisión fue crear un proyecto de SPB basado en ARP-Path, en el que el broadcast/multicast crea los mismos árboles de ARP-Path y el envío unicast (que es lo que al final realmente afectará a la medida de latencias) se realiza en base a unas tablas de encaminamiento construidas a mano, basándose en las decisiones que se tomarían en SPB. Las tablas de encaminamiento son únicas para una topología cada vez y en la comparativa usamos la que se muestra a continuación, *dataCenter250.ned*, que se trata de una red de centro de datos en la que los puentes frontera (*edge bridges*) tienen 25 hosts cada uno. Las rutas creadas son entre bridges, no entre hosts (dado la complejidad que supondría construír todas las rutas, y simétricas, entre los 250 hosts); por ejemplo, el puente 0 dirige el tráfico que va a los puentes 1,2,5,6 y 9 por su enlace superior (el que va al puente A), y el resto del tráfico (hacia los puentes 3,4,7,8) va dirigido por su enlace inferior (el que va al puente B). Siguiendo la lógica anterior, se construyeron caminos bien distribuidos por toda la red y simétricos para SPB.

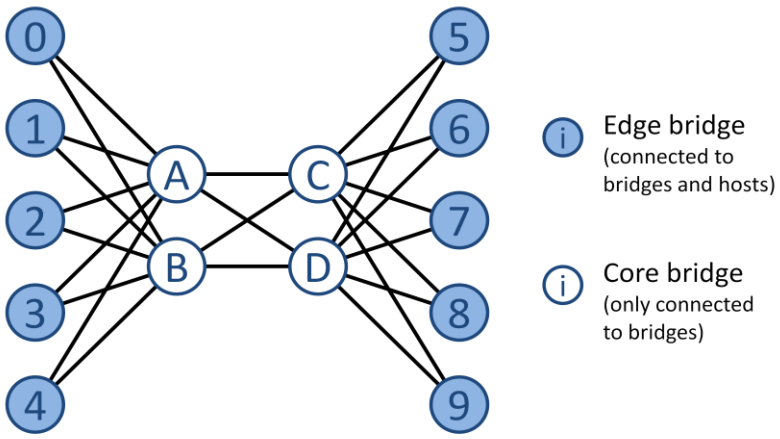


Figura 73. Topología de red de centro de datos con 25 hosts por edge bridge, 250 en total, denominada dataCenter250.ned en nuestro proyecto de OMNeT++

Tras una simulación con el generador de flujos (especificaciones en Apéndice F) implementado en OMNeT++ de 5000 segundos, con una frecuencia de generación exponencial de media 1.6 segundos y tráfico generado uniformemente entre todos los hosts, enlaces a 100Mbps y retardo de propagación de 1 microsegundo, y tiempo

de procesamiento en los switches de 2 microsegundos, los resultados fueron los siguientes:

5000s/1.6s/Uniforme	Latencia máxima	Latencia mínima	Latencia media
SPB	1,884 ms	0,021 ms	0,335 ms
ARP-Path	0,811 ms	0,022 ms	0,343 ms

Tabla 5: Resultados de la comparativa de latencias SPB vs ARP-Path en OMNeT++ con una simulación de 5000s con frecuencia del generador de flujos 1.6 segundos y tráfico uniforme

En una segunda simulación, se repiten los mismos parámetros salvo que ahora el tráfico no se genera de manera uniforme entre todos los hosts de la red, sino que un host (en concreto el numerado como host150) tiene un peso de 100, es decir, tiene un posibilidad 100 veces mayor que el resto de hosts de ser escogido como parte (origen o destino) de cada nuevo flujo que se cree por parte del generador.

5000s/1.6s/No unif. (peso 100 en 1 host)	Latencia máxima	Latencia mínima	Latencia media
SPB	0,789 ms	0,038 ms	0,333 ms
ARP-Path	0,665 ms	0,041 ms	0,333 ms

Tabla 6: Resultados de la comparativa de latencias SPB vs ARP-Path en OMNeT++ con una simulación de 5000s con frecuencia del generador de flujos 1.6 segundos y tráfico no uniforme (peso 100 en el host 150)

Nótese que OMNeT++ aporta promedios de latencias para cada camino en la red y luego nosotros hemos obtenido la media de las latencias medias para obtener la media global. En el caso de las latencias máximas y mínimas, OMNeT++ aporta las latencias máximas y mínimas de todos las latencias medidas a lo largo de la simulación en toda la red, y en la tabla se calcula la media de esas máximas y de esas mínimas, por lo que no son máximos y mínimos globales, sino promedios de dichos valores, para dar una idea más global y menos puntual de cierta simulación o camino lento o rápido.

Como se puede observar, mientras que la latencia mínima y media se mantienen muy similares en ARP-Path y SPB, la diferencia lo da el caso de latencia máxima, que en SPB es siempre mayor y, en el primer caso, hasta de 1 milisegundo más, lo que es bastante relevante. La razón de que esto suceda es que SPB calcula los caminos de antemano, sin el estado de la red, y causa que en algunos casos se utilice un camino que en cierto momento es especialmente lento y que ARP-Path habría esquivado con su exploración de caminos por latencias, haciendo que SPB tenga máximos más altos. En el caso de latencias mínimas, siempre hay un mínimo de la red que viene dado de la suma de tiempos de propagación, colas y procesamiento en los puentes, y podemos decir que ARP-Path y SPB tienen tiempos mínimos similares y promedios también, que se acercan a los tiempos óptimos que permiten la red.

Por lo que se puede concluir que mientras que ARP-Path ofrece caminos mínimos similares a SPB, sin necesidad de configuración ni protocolo de estado de enlace, además su varianza de latencias respecto de la media es bastante menor que la de SPB.

4.1.7.3 Aprovechamiento de la topología y reparto de carga

ARP-Path, al igual que SPB y TRILL, y al contrario de RSTP, no excluye ningún enlace de la topología a la hora de poder ser usado en las comunicaciones, es decir, por definición aprovecha los recursos de la topología al completo. Pero no sólo eso, dado que construye caminos de baja latencia, los nuevos caminos tenderán a utilizar partes de la topología no utilizadas, repartiendo de manera natural la carga a lo largo de la red.

Para demostrar este punto, a continuación se muestran resultados experimentales:

> Demostración de reparto de carga mediante emisión de tráfico UDP en la implementación de ARP-Path en OMNeT++:

A la hora de demostrar el reparto de carga que realiza de manera nativa ARP-Path se escogió primero la aplicación de un generador de flujos (Apéndice F) aleatorio en SimPy [SimPy], paquete de simulación basado Python, cuyos resultados fueron tan satisfactorios que se decidió pasar el modelo a OMNeT++, que simula paquetes y tráfico en realidad de flujos y simulación un poco más a alto nivel. La topología utilizada fue la misma que para SimPy, que es la malla de 3x3 que se mostraba en la Figura 72.

Así pues, se realizaron 6 simulaciones de 15000 segundos cada una, para hacer la media, todas con enlaces de 100Mbps, frecuencia de generación de flujos 1.6 segundos uniforme entre todos los hosts, y se midieron las rutas generadas entre los hosts del puente 0 y el puente 8 en ambos sentidos (en verde se muestra el porcentaje de rutas totales que escogió cada enlace) y la utilización del enlace en cada sentido de recepción (en negro). Como se ve en la siguiente figura, tanto el porcentaje de rutas, como la utilización están bastante equilibrados a lo largo de la red. De hecho, en el puente 0 se divide casi en 50% y 50% entre el puente 1 y 3, y estos a su vez dividen de nuevo entre dos (25% y 25%) para los puentes que continúan el camino, y al llegar al puente 8 tenemos más o menos 50% desde el puente 5 y 50% desde el 7. El hecho de que no sea exactamente 50% y 50% (como sucedía con SimPy) posiblemente se debe a que no se haya alcanzado un estado transitorio de la simulación y entonces algunas rutas llevan flujos mayores.

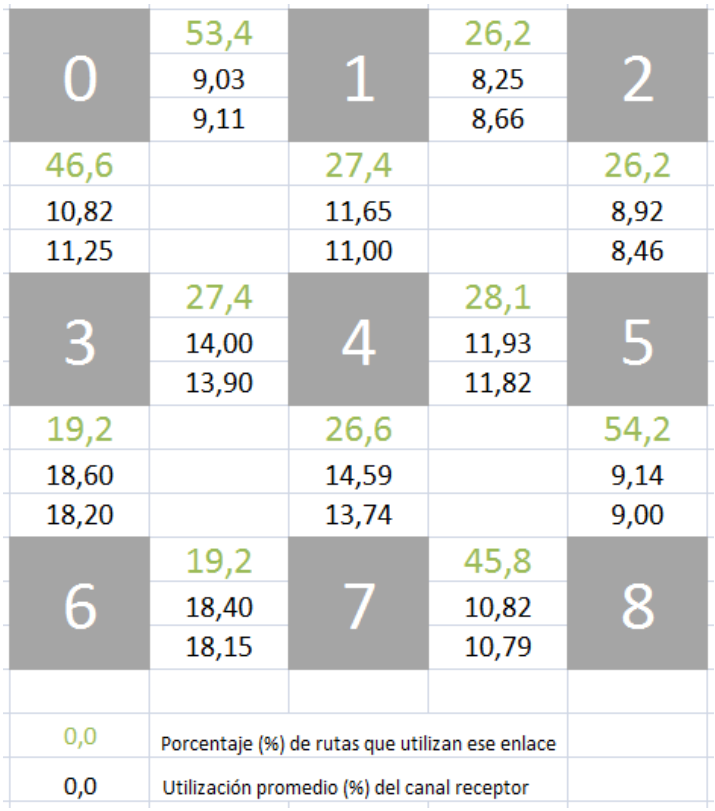


Figura 74. Reparto de carga en topología de malla para OMNeT++ con generador de flujos uniforme de frecuencia 1.6 segundos y enlaces a 100Mbps

Demostración de reparto de carga mediante emisión de tráfico UDP en la implementación de ARP-Path en NetFPGA:

Tras demostrar el reparto de carga con el simulador OMNeT++, se decidió probar dicho reparto en una plataforma real, en este caso las tarjetas NetFPGA. Lógicamente, es imposible transformar el generador de flujos (Apéndice F) a un generador real con tantos hosts, por lo que en su lugar se utilizaron máquinas virtuales y flujos de vídeo *streaming* UDP, tal y como se adelantaba gracias a la estancia de colaboración en el *Cambridge Computer Laboratory.* Eso sí, la topología en malla se mantuvo, para comparar directamente unos resultados con otros.

El escenario realizado es el mismo ya descrito para el apartado de latencia mínima en un único sentido. En el switch 0 hay un servidor VLC y en el 8 están los clientes VLC y se emite tráfico *streaming* de vídeo UDP desde servidor a cada cliente, uno a uno. En la Figura 75 se muestra los resultados obtenidos para una media de 4 repeticiones en las que el número de clientes en el puente 8 es 4, mientras que en la Figura 76 se muestran los resultados para una media de 4 repeticiones también, pero con 8 clientes en el puente 8, en lugar de 4. Nótese que, a diferencia del caso anterior, una vez creados los 4 y 8 flujos (respectivamente), ya se miden las rutas y que el tiempo de ejecución no es un parámetro (pues no se crearán más flujos). Además, no es posible medir la utilización del canal en este caso (de ahí los valores a cero).

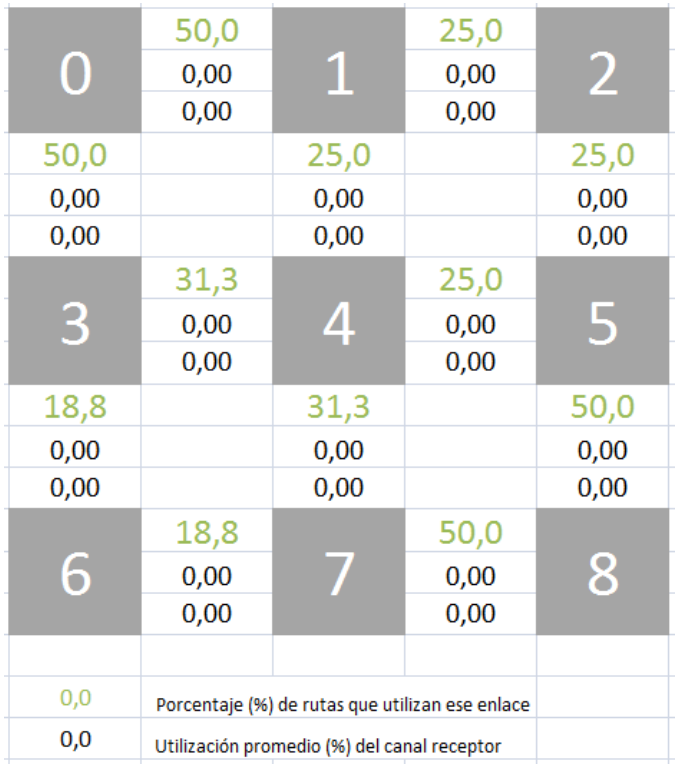


Figura 75. Reparto de carga en topología de malla para switches ARP-Path implementados en tarjetas NetFPGA con tráfico *streaming* UDP de vídeo entre los puentes 0 y 8 (1 servidor y 4 clientes)

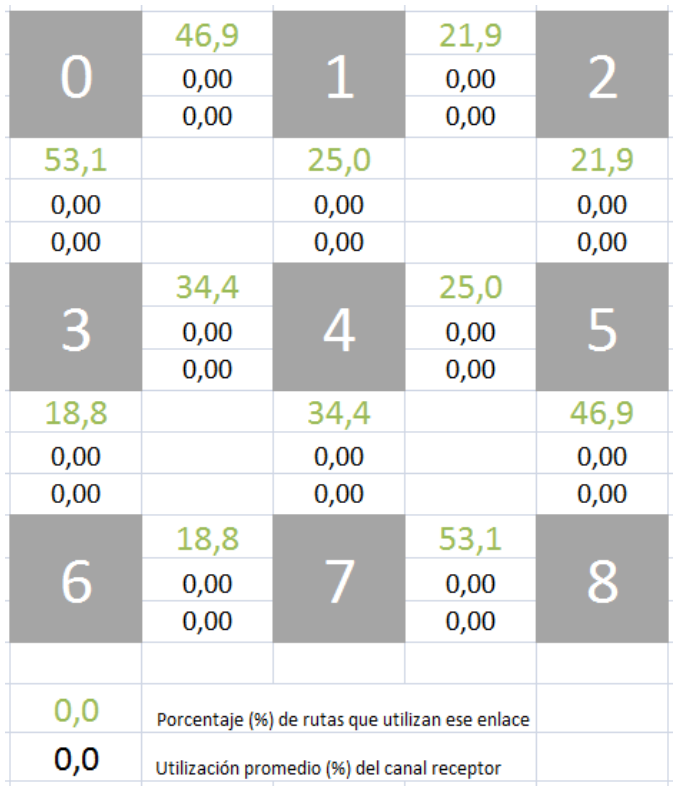


Figura 76. Reparto de carga en topología de malla para switches ARP-Path implementados en tarjetas NetFPGA con tráfico *streaming* UDP de vídeo entre los puentes 0 y 8 (1 servidor y 8 clientes)

Como es posible apreciar, en ambos casos existe reparto de carga y es incluso similar a los resultados obtenidos en OMNeT++ en los que el tráfico se va dividiendo entre dos en cada bifurcación, es decir, en cada puente el 50% del tráfico que llega va por un enlace al siguiente puente y el 50% restante al otro. Es cierto que en este caso todos los flujos tienen el mismo tamaño, entonces el número de rutas encaja más porque todas las rutas llevan flujos iguales (cosa que no sucede en OMNeT++), pero también es cierto que con sólo 4 y 8 flujos existe un reparto de carga muy bueno, similar al reparto que sucede con un mayor número de flujos (cientos, en el caso de simulaciones de OMNeT++ de más de 10000 segundos) y mayor tráfico.

Finalmente, como curiosidad, a continuación se muestran algunas imágenes del montaje realizado en el CCL que consistía en:

- 9 switches ARP-Path, que eran PCs con sistema operativo CentOS y la tarjeta NetFPGA instalada con el proyecto de ARP-Path
- 1 pantalla para arrancar y monitorizar los switches anteriores
- 1 host servidor VLC y 1 host con 4/8 máquinas virtuales clientes VLC

Figura 77. Algunos de los switches ARP-Path (cada switch es un PC con una tarjeta NetFPGA instalada en él) del montaje en malla realizado en el CCL

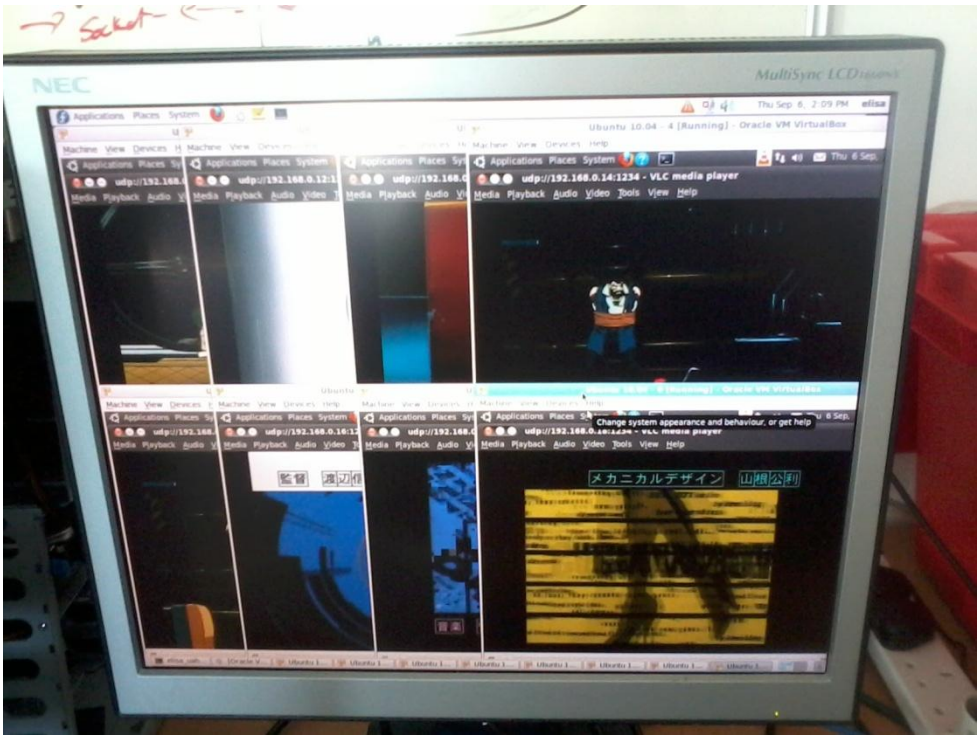


Figura 78. Host conectado al switch 8 que contiene las 8 máquinas virtuales que actúan como clientes VLC, en las que se ve como reciben el tráfico *streaming* de vídeo

4.1.7.4 Propagación de mensajes broadcast

Por último, una característica no ventajosa de ARP-Path es que difunde los mensajes broadcast por toda la red. De hecho, en el caso concreto del ARP Request, éste es necesario para descubrir todo camino nuevo, pero para caminos ya descubiertos podría hacerse uso de un proxy ARP para reducir la inundación [EC09]. En todo caso, la utilización de proxies es totalmente compatible con ARP-Path, salvo en el caso de los ARP Request, que ha de permitirse su propagación para caminos nuevos, pero si es un camino nuevo seguramente no se encontrará su entrada en el proxy tampoco, así que inicialmente también es compatible.

De todos modos, el efecto principal a limitar es la carga excesiva en los hosts, dado que el número de enlaces por los que se propaga "de más" es menor para el caso de enlaces entre switches que para enlaces de switch a host. Partiendo de este punto habría dos opciones:

- Proxies a la salida del ARP (actuando en el puente frontera del host origen), que no deberían usarse en todos los casos para ARP-Path porque utiliza dicho mecanismo para descubrir los caminos.

- Proxies a la llegada del ARP (actuando en el puente frontera del host destino), que resuelven los ARPs ya conocidos de los hosts a los que sirven o que directamente no propagan el mensaje al host si el ARP no le corresponde a él.

El primer tipo de proxy reduciría la propagación "extra" de los enlaces entre switches respecto al segundo tipo, dado que tanto el primero como el segundo eliminan el envío por los enlaces entre switches y hosts, pero además su complejidad es algo mayor porque no siempre se podría aplicar al necesitar ARP-Path el mecanismo del ARP para explorar los caminos y necesitaría una valoración de cuándo aplicar el proxy y cuando no para un ahorro de propagación relativamente no muy grande en comparación con el segundo método. El segundo tipo de proxy ofrece un ahorro similar en la práctica que el primer tipo (que es el ahorro de propagación por los enlaces de los hosts, que son mayores en número), pero a cambio necesita conocer los hosts que tiene conectados en cada momento, para lo que quizás bastaría con el ARP gratuito que emiten los hosts al conectarse a un puente o con algún mensaje especial intercambiado entre host y puente frontera.

4.2 Flow-Path

El protocolo Flow-Path es una variante de ARP-Path derivada del problema de fluctuación de caminos que se mencionó en apartados anteriores (la otra solución, como ya se conoce, es la evolución a ARP-Path unidireccional, ya explicada). Esta variante es el segundo protocolo de la familia All-Path, tras ARP-Path. Al contrario de ARP-Path, Flow-Path no posee una evolución tan detallada del protocolo al tratarse de una variante surgida de un problema de la versión inicial de ARP-Path que luego se solucionó en posteriores modificaciones. Sin embargo, al margen de solucionar ciertos problemas de la versión inicial de ARP-Path (FastPath), posteriormente pasó a considerarse la versión de flujos de ARP-Path, con características y propiedades concretas al margen de dar solución a dichos problemas, y su ideonidad para ciertos escenarios se explicará en un apartado posterior.

Como curiosidad comentar que inicialmente este protocolo era conocido como SA-DA, haciendo referencia directa a lo que se verá a continuación: que las entradas ahora están asociadas a tanto la dirección origen (SA, *source address*) como la destino (DA, *destination address*). Sin embargo, recientemente se escogió el nombre de Flow-Path, siguiendo la estructura del nombre de ARP-Path y el hecho de que ahora los caminos se crean en base a flujos (*flow* en inglés).

4.2.1 Funcionamiento de Flow-Path

De manera análoga a ARP-Path, Flow-Path basa la creación de sus caminos en la inspección de los mensajes ARP. A continuación se explica cómo se generan y reparan los caminos en este protocolo.

4.2.1.1 Creación de caminos

El funcionamiento de Flow-Path, de manera análoga a ARP-Path, es y se divide en las dos siguientes fases:

Descubrimiento del camino (*ARP Request*): El mecanismo es análogo al realizado a ARP-Path, en el que se establecen caminos de baja latencia que son aquellos creados por la primera copia de un ARP Request que alcanza el host destino. El proceso, que se describe en la próxima figura, funciona de la siguiente manera:

Cuando el host origen A quiere comunicarse con otro destino C, primero manda un ARP Request encapsulado en una trama broadcast para resolver la dirección IP del destino C. El puente frontera de A, llamado 1, recibe la trama de A y asocia la dirección MAC global de A al puerto a través del que recibe el mensaje, cerrando (pasando su estado a *locked*) temporalmente el aprendizaje del flujo con origen en A en este puerto y bloqueando el resto de puertos del bridge 1 de manera que no aprendan, ni reenvíen, más tramas broadcast de dicho flujo con origen en A. La diferencia con ARP-Path, es que Flow-Path además de la dirección A, necesita asociar la dirección destino, desconocida (es el ARP el que está preguntando por ella), es decir, el flujo completo. Por lo tanto, además de A, se asocian en la entrada la IP origen y la IP destino (que sí son conocidas, inspeccionando el mensaje ARP), que también caracterizan el flujo, y así con el ARP Reply se podrá confirmar la dirección destino y el flujo, ignorando dichas IPs apuntadas para el encaminamiento y siendo sólo necesarias para la generación del camino. Así pues, las tramas broadcast con dirección origen A, IP origen IP_A e IP destino IP_C que lleguen a cualquier otro puerto del bridge será descartadas y consideradas tramas que llegan tarde (*late frames*).

Como se puede apreciar en la siguiente tabla, ahora la entrada de las tablas de los switches Flow-Path tienen un par de campos más, para las IPs, y además la clave no es sólo una dirección MAC, sino la combinación de dos: origen y destino, que caracterizan el flujo.

key(src-dst)	srcIP	dstIP	port	timer	state
A-FF	IP_A	IP_c	1	0	*Locked*

Tabla 7: Ejemplo de entrada en la tabla de direccionamiento de un switch en la fase de descubrimiento del camino de Flow-Path

Después de recibir la trama, el bridge 1 la reenviará por todos sus puertos excepto por aquel por el que la recibió. Los puentes 2 y 4, que recibirán dicha trama de parte de 1, se comportarán de manera idéntica a 1, haciendo *lock* de la dirección A y las IPs en el puerto que primero recibe la trama y reenviando por todos los puertos excepto por el cual la recibieron. Esto hará que se envíen tramas entre ellos, que serán copias tardías (*late frames*) y serán descartadas, tal y como se ve en la siguiente figura. Lo mismo ocurrirá esta vez con los puentes 3, 5 y 6, que recibirán la trama de 2 y 4. Finalmente el host destino C recibirá la trama.

Por lo tanto, la asociación temporal (*locking*) de la dirección A y las IPs en un puerto de cada bridge se propaga a través de la red como un árbol enraizado en el host A, y ahora existe una cadena activa de bridges entre A y C con un puerto de entrada asociado a la dirección A y las IPs, que caracterizan el flujo entre A y C.

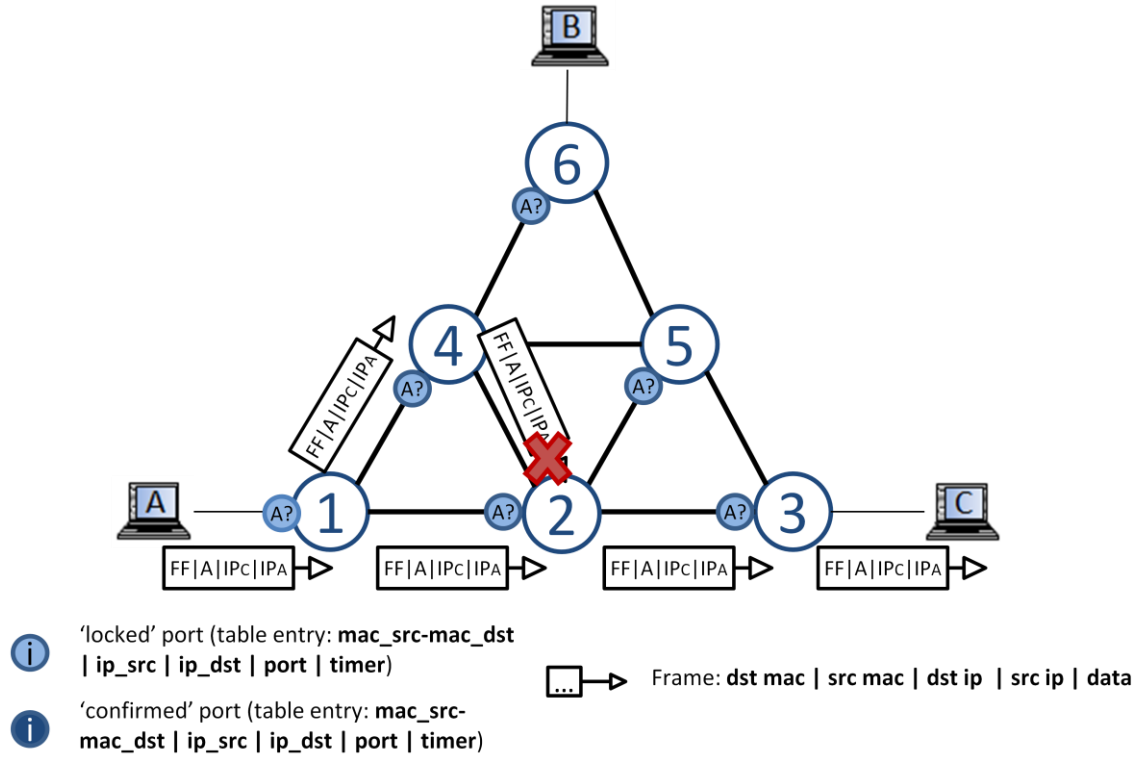


Figura 79. Aprendizaje de la dirección del host origen (A) mediante el mensaje ARP Request en Flow-Path. Las tramas con una cruz en rojo son algunas de las bloqueadas por ser tardías (*late frames*) y ya estar aprendidas en otro puerto (el más rápido)

Confirmación del camino (*ARP Reply*): La confirmación del camino se realiza con la respuesta que da C al ARP Request, que es un mensaje ARP Reply, y en sentido inverso.

Cuando el puente 3, frontera de C, recibe el mensaje unicast ARP Reply, éste realiza una doble asociación con la dirección de C: por un lado la asocia junto con A al puerto de entrada y a su vez, si la dirección destino (que ahora es A) ya está en estado *locked* (como es el caso) y coinciden las IPs contenidas en el ARP, confirmará el aprendizaje tanto de C-A (entrada de flujo que dirige hacia C), como de A-C (entrada de flujo que dirige hacia A), cada uno en su puerto. Después, al ser una trama unicast, ésta no se reenviará por todos sus puertos menos el de entrada (como sucedía en el caso del ARP Request), sino que buscará la dirección destino entre las direcciones aprendidas en 3 y reenviará la trama por el puerto asociado a A-C.

key(src-dst)	srcIP	dstIP	port	timer	state
A-C	IP_A	IP_C	1	0	*Confirmed*
C-A	IP_A	IP_C	2	0	*Confirmed*

Tabla 8: Ejemplo de entrada en la tabla de direccionamiento de un switch en la fase de confirmación del camino de ARP-Path

A continuación, la trama llegará al bridge 2, que de nuevo asociará las direcciones C-A al puerto de entrada y la reenviará por el puerto asociado a A-C, confirmando una vez más el aprendizaje de los puertos tanto para C-A como para A-C. Y lo mismo sucederá con el bridge 1, el que enviará finalmente la trama al destino A, que, tras haber recibido la respuesta a su solicitud ARP, comenzará la comunicación y envío de datos que quería realizar.

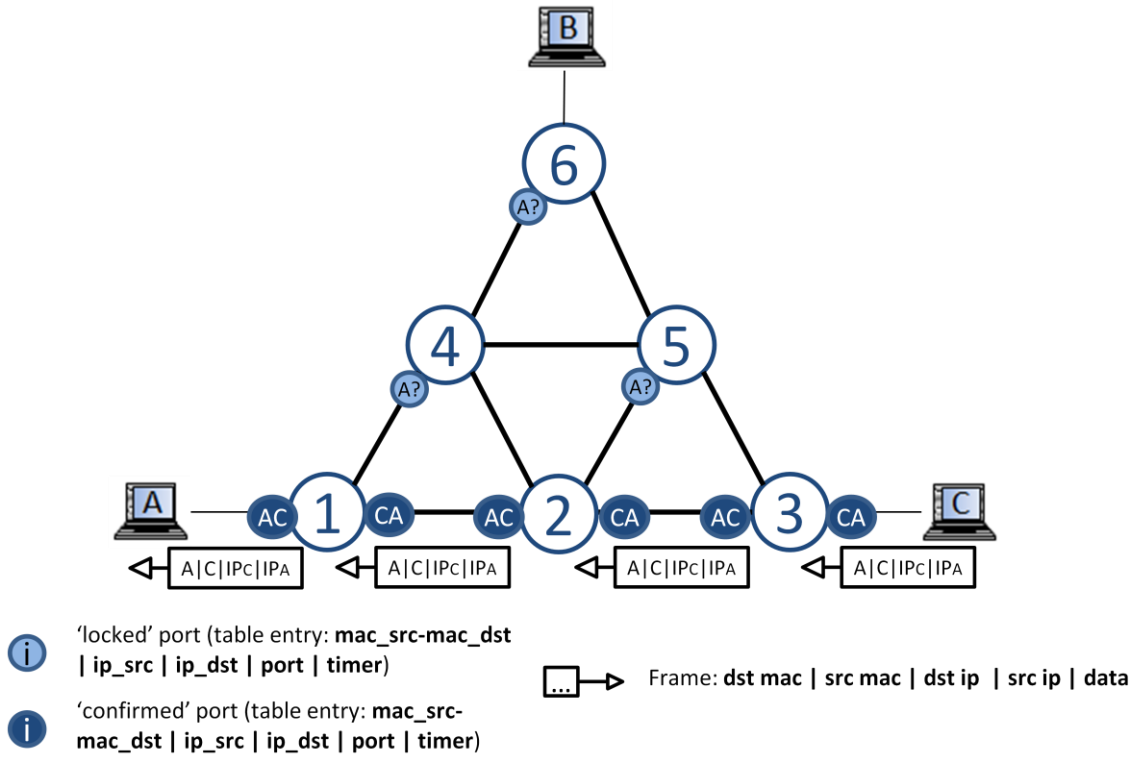


Figura 80. Confirmación de la dirección del host origen (A) y host destino (C) mediante el mensaje ARP Reply, creando un camino entre A y C, en Flow-Path

El mecanismo de confirmación genera un camino que es simétrico, es decir, que coincide en ambas direcciones A-C y C-A. Nótese que ahora las claves de las entradas de la tabla de encaminamiento son una combinación de dos direcciones, en las que la primera indica la dirección de encaminamiento y la segunda sólo acompaña para diferenciar el flujo de otros que tengan la misma primera dirección; es decir, podría haber entradas que comiencen por A asociadas a diferentes puertos, pues unas serán de unos flujos y otras de otros.

Los **estados** de '*locked*' (asociación temporal) y '*confirmed*' (aprendizaje confirmado) de cada una de las fases tienen un **temporizador** (timer), que caducará en caso de no ser refrescado. La misión del estado '*locked*' es evitar bucles que podrían crearse por las tramas que se envían replicadas por todos los puertos y el valor de su timer es pequeño (< 1 segundo). Mientras que el estado '*confirmed*' muestra un camino aprendido y su misión es encaminar tramas, por lo que su timer será mayor (de varios minutos). El **refresco** de dichos timers se realiza de la siguiente forma:

- Si el estado de la entrada de A-¿? es '*locked*', éste se refresca con cualquier trama ARP que tenga como origen A, las IPs coincidan con las del flujo y que llegue por el puerto asociado.

- Si el estado de la entrada de A-C es '*confirmed*', éste se refresca con cualquier trama de datos que tenga como origen A, como destino C y que llegue por el puerto asociado.

En el caso de que se emitan dos ARPs simultáneos en direcciones opuestas (de A a C y de C a A, a la vez) y que además escojan dos caminos diferentes, cosa que es poco probable pero posible, se utilizarán mecanismos de prioridad basados para prevenir crear más de un camino. En este caso se escogió un mecanismo basado en la dirección MAC que, si existía un camino ya inicial, sólo confirmaba aquel creado por la trama ARP Reply cuya dirección origen fuese mayor que la dirección destino. A diferencia de FastPath (la primera versión de ARP-Path, que es de la que parte y nace Flow-Path), este mecanismo no genera el problema de las oscilaciones, dado que no se comparten entradas para diferentes flujos, que era la causa de dicho problema.

4.2.1.2 Reparación de caminos

El mecanismo de reparación de caminos en Flow-Path es análogo al de ARP-Path, salvo que en este caso no existe una diferenciación entre dos mecanismo de reparación "hacia atrás" y "hacia adelante" y sólo existe la opción "hacia atrás", puesto que se debe recuperar el camino completo del flujo cada vez, al no ser ya independientes los árboles creados por dirección origen.

Como se observa en la siguiente figura, cuando una trama del flujo entre los hosts A y C no encuentra entrada para el puerto de salida hacia C (entrada C-A), se emite un PathFail hasta el puente frontera de A, que emitirá un PathRequest hasta el puente frontera de C y que éste último responderá con un PathReply hasta el frontera de A.

Loopback de tramas

La diferencia es que ahora en Flow-Path todos los caminos son siempre simétricos, al contrario de ARP-Path, que no puede garantizarlo en todos los casos. Al ser todos simétricos, se garantiza que el *loopback* de tramas es posible, por lo que el mensaje PathFail desaparece, ya ni siquiera será un mensaje broadcast, pues la misma trama unicast que descubrió el enlace caído se podrá reenvíar hacia atrás por el camino por el que vino.

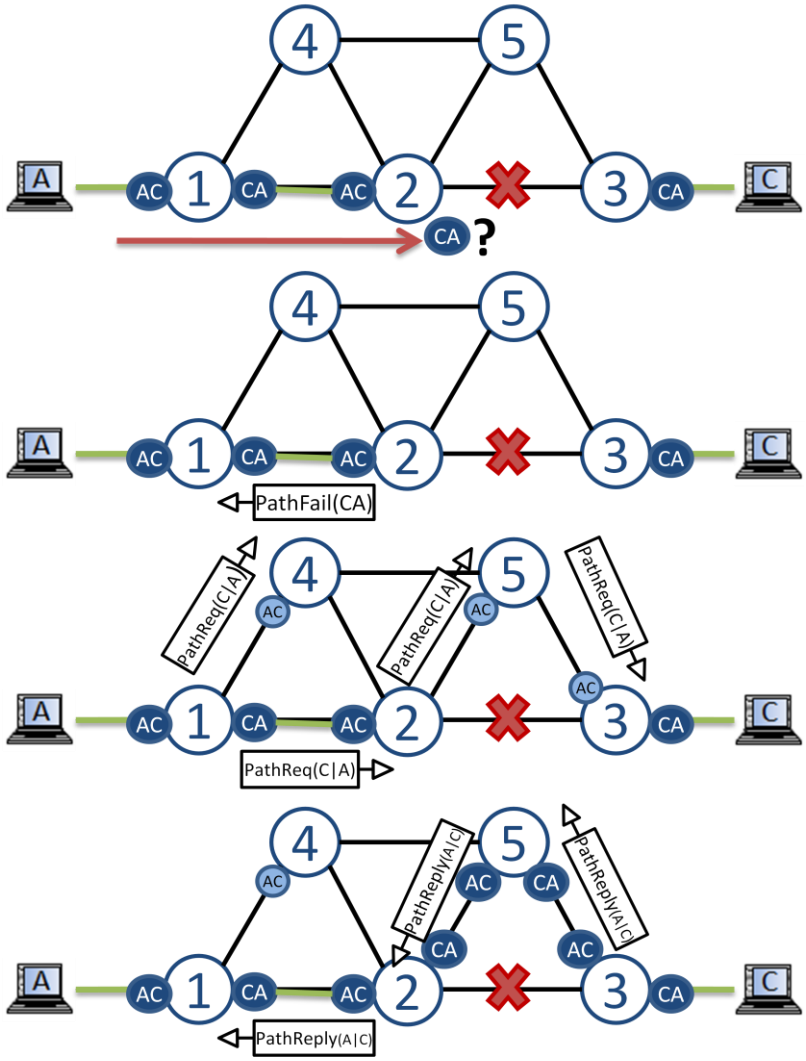


Figura 81. Reparación de caminos para Flow-Path en tres pasos: PathFail, PathRequest y PathReply

Tablas y mensajes necesarios para la reparación en Flow-Path

En cuanto a las tablas y mensajes necesarios, son análogos a los necesarios en ARP-Path. Las tablas necesarias son *Learning Table* (LT), *Broadcast Table* (BT), *Hello Table* (HeT), *Host Table* (HoT) y *Repair Table* (RT). Los tipos de mensajes especiales necesarios, se podrían resumir en: mensajes *Hello* (para definir puentes frontera y no), *PathRequest* y *PathReply* (el mensaje *PathFail* ya no es necesario pues se sustituye por el *loopback* de tramas). Todos estos mensajes tienen como MAC destino una dirección multicast específica para ARP-Path (*AllPathBridgesMcast*) y como origen la MAC la dirección del switch que emite el mensaje, aunque en la práctica este campo podría ser cualquiera, dado que si la dirección multicast es *AllPathBridgesMcast* estará claro que el mensaje lo ha creado un bridge ARP-Path y no es necesario tampoco saber cual. Dentro del mensaje estan encapsulados tres campos: tipo (Hello = 1, ~~PathFail = 2,~~ PathRequest = 3, PathReply = 4), "dirección origen del flujo a reparar" (siempre será el origen de la trama) y "dirección destino

del flujo a reparar" (destino de la trama). Es decir, el formato de la trama de reparación coincide con la de ARP-Path.

La necesidad de los mensajes especiales radica en el hecho de que los caminos en ARP-Path, inicialmente (FastPath), se fijaban con el primer ARP y luego nuevos mensajes ARP eran descartados para aprendizaje y Flow-Path evolucionó desde esa misma versión, conservando el bloqueo de aprendizaje con el primer ARP. Sin embargo, opcionalmente Flow-Path podría evolucionar hacia un protocolo menos bloqueante (como la última versión de ARP-Path, la versión "retardada") y quizás utilizar igualmente mensajes ARP creados en los puentes frontera, en lugar de los mensajes especiales, para disminuir el tiempo de procesamiento de estas tramas especiales tal y como se puede hacer en las últimas versiones de ARP-Path. De esta manera, el único mensaje especial necesario para Flow-Path sería el *Hello*, al sustituirse el *PathFail* por el *loopback* de tramas, el *PathRequest* por un ARP Request y el *PathReply* por un ARP Reply, pero por supuesto es una opción que queda pendiente de exploración aún.

4.2.2 Implementación en OMNeT++ y OpenFlow

Tras descubrir los problemas de la primera versión de ARP-Path (FastPath), surgieron dos nuevas variantes de desarrollo: ARP-Path unidireccional y Flow-Path. Así pues, se actualizaron las implementaciones en OMNeT++ y OpenFlow para Flow-Path (no así la de Linux, orientada a prueba de concepto y no a evaluación), al igual que se hizo para ARP-Path unidireccional. A continuación se detallan algunos matices de estas implementaciones para Flow-Path, puesto que la base en la práctica es la misma que la de FastPath.

Implementación en OMNeT++

Esta implementación está realizada sobre el simulador OMNeT++ (versión 4) y la librería, o framework, inet (versión 1), ya conocidas. La base de la implementación, al igual que con ARP-Path, es el módulo *MACRelayUnitNP*, que se encuentra dentro de la sección dedicada a Ethernet *linklayer/etherswitch/* de la librería inet (en dicha versión 1) y, como su nombre indica, implementa la *relay unit* de un switch Ethernet.

Así pues, la única diferencia entre el código de ARP-Path y Flow-Path es la lógica aplicada en la función `handleAndDispatchFrame`, dado que el resto de la estructura de los switches All-Path tiene mucho en común, como ya se ha visto.

Implementación en OpenFlow

Al igual que en la implementación anterior, en el caso de OpenFlow lo único que varía respecto a ARP-Path es la lógica aplicada en la función principal "`handle_packet_in`", que pasa a contener la lógica de Flow-Path. Después, las pruebas se realizaron sobre la máquina virtual Mininet y switches OpenFlow sobre Linux en tarjetas Soekris.

A continuación se muestra una tabla comparativa de tiempos de creación de caminos (*path set up*) y ping entre hosts (*pinging time*) entre las implementaciones OpenFlow (Mininet y Soekris) de ARP-Path y Flow-Path. Como es posible apreciar, los tiempos de ping apenas varían, mientras que sí lo hacen a la hora de crear los caminos. Esto se debe a que los tiempos de ping se basan en los tiempos de forwarding de cada plataforma, que a su vez se basan en la tabla de encaminamiento que genera OpenFlow y que es idéntica para ARP-Path y Flow-Path; mientras que en la creación del camino influye la lógica de protocolo y ejecución del código Python en el controlador OpenFlow, que como ya sabemos es ligeramente más compleja para Flow-Path que para ARP-Path, de ahí que sea más lenta también.

Mininet OpenFlow	Path set up	Pinging time	Soekris OpenFlow	Path set up	Pinging time
ARP-Path	61,6 ms	49 µs	ARP-Path	69,1 ms	10,2 ms
Flow-Path	71,1 ms	51 µs	Flow-Path	99,1 ms	10,2 ms

Tabla 9: Comparativa de tiempos de creación de caminos y ping en la implementación OpenFlow (Mininet y Soekris) de ARP-Path y Flow-Path

4.2.3 Evaluación de Flow-Path frente a ARP-Path

En el siguiente apartado se realiza primero un análisis de las soluciones aportadas por Flow-Path respecto a los problemas de FastPath y, en segundo lugar, una comparativa de estas dos variantes surgidas de FastPath, que son ARP-Path (versión unidireccional, en su evolución tras FastPath) y Flow-Path, y qué ventajas pueden tener una y otra versión, dadas sus diferentes características a la hora de almacenar entradas de encaminamiento en tablas y distribuir el tráfico en la red.

4.2.3.1 Problemas de FastPath y soluciones aportadas en Flow-Path

A continuación se enumeran brevemente los problemas de FastPath y cómo se solucionan en Flow-Path.

Problemas con la asimetría de caminos

FastPath necesitaba que sus caminos fueran siempre simétricos, de tal forma que el refresco, y por tanto el protocolo en sí, funcionara correctamente. Sin embargo, esa simetría en la práctica era imposible de cumplir para todos los casos dada la naturaleza de ARP-Path en la que los caminos se crean en base a una única dirección, compartida para múltiples flujos.

Flow-Path sin embargo crea un camino por flujo, por lo que garantiza que los caminos siempre serán simétricos, descartando este problema. En los casos en los que haya múltiples mensajes ARP se aplicará un mecanismo de prioridad que se aplicaba en FastPath, en el que sólo confirma el camino la trama ARP Reply que

poseía dirección de origen < dirección de destino, en caso de haber ya otro aprendizaje anterior. En este caso la dirección de origen y de destino, son únicas, las que forman el flujo, por lo que siempre habrá un extremo en el que dicha condición se cumpla y otro en el que no.

Fluctuación de caminos

En Flow-Path, el mecanismo de prioridad sólo se aplica para dos direcciones, las que forman el flujo, dado que las entradas no se comparten entre diferentes flujos, por lo que no hay posibilidad de fluctuación de caminos.

Necesidad de flag de reparación

En este caso, para Flow-Path este problema persiste, pues se siguen confirmando las entradas con el ARP Reply (o mensaje *PathReply*). La solución, al igual que con FastPath, es incluir un flag de reparación en las entradas de las tablas, para que se conserve el bloqueo tras la llegada del *PathRequest* y no se produzcan bucles.

Este problema se resuelve en posteriores versiones de ARP-Path con el hecho de que dejan de confirmarse los caminos y los temporizadores pasan a funcionar de manera diferente. Es decir, siguen existiendo dos estados, uno de bloqueo y otro de aprendizaje, pero el tiempo de bloqueo es fijo y de ahí se pasa al de aprendizaje, al margen de que se haya recibido el ARP Reply o no.

Así pues, otra opción sería explorar la evolución de Flow-Path en un sentido similar a la última versión de ARP-Path, como ya se comentaba previamente. De este modo, el ARP Reply (o *PathReply* en su lugar), no confirmaría el camino, sino que sólo aportaría los datos que faltan en la entrada, es decir, sólo aportaría el valor de la dirección destino del flujo, pero el estado de bloqueo no cambiaría a confirmado inmediatamente, sino tras cierto tiempo, evitando la necesidad del flag.

4.2.3.2 Comparativa de almacenamiento y distribución de carga en Flow-Path frente a ARP-Path

Tras mostrar cómo afronta Flow-Path los problemas de FastPath, la evaluación del protocolo Flow-Path se centra principalmente en las diferencias con ARP-Path, dado que fueron dos ramas de desarrollo que surgieron tras FastPath. Como ya sabemos, Flow-Path garantiza simetría de los caminos y además un mejor balanceo de carga, dado que genera más caminos (al ser por flujo y no por host). Pero el coste es que el pseudocódigo es ligeramente algo más complejo, aunque aún simple, y el tamaño de las tablas de direccionamiento aumenta también con el aumento de caminos.

En los siguientes apartados analizamos el almacenamiento y la distribución de carga de manera comparativa entre Flow-Path y ARP-Path. Para ello se realiza un análisis teórico y en la segunda parte además se utilizan algunos resultados obtenidos con el simulador OMNeT++.

Comparativa de almacenamiento de entradas

Una de las diferencias claves es el tamaño de la tabla de encaminamiento porque las entradas de la tabla se crean para cada flujo en lugar de para cada host, y es evidente que el número de flujos superará al de hosts por lo que el número de entradas de tabla necesarias será mayor para Flow-Path que para ARP-Path. A continuación analizamos teóricamente la diferencia entre ambos.

Número de entradas

Sea T_{AP} el número medio de entradas totales en cierta topología para ARP-Path y T_{FP} para Flow-Path. Este número en el caso de ARP-Path dependerá del número de hosts y de todos los bridges que requieran esa entrada para otros hosts, mientras que para Flow-Path dependerá del número de flujos y el número de bridges medio atravesado en un camino. Así pues, las expresiones quedan de la siguiente manera:

$$T_{AP} = H * (b + b_s)$$
$$T_{FP} = F_u * b = H * (H - 1) * b$$

Tabla 10: Número medio de entradas totales en la topología para ARP-Path (AP) y Flow-Path (FP)

Donde:

T_{AP} es el número medio de entradas totales para ARP-Path

T_{AP} es el número medio de entradas totales para Flow-Path

H es el número medio de hosts activos en la topología

F_u es el número medio de flujos unidireccionales activos en la topología

b es el número de bridges que componen el camino medio de la topología

b_s es el número de bridges que se comparten (s de *shared*) con otros destinos además de b

Nótese que dos hosts dan lugar a dos flujos unidireccionales o un flujo bidireccional, es decir, que siempre que haya comunicación en los dos sentidos el número de flujos bidireccionales será la mitad que los unidireccionales ($F_u = F_b * 2$) y además el número de flujos bidireccionales siempre es una combinación de pares de hosts activos ($F_b = \frac{H*(H-1)}{2} \rightarrow F_u = H * (H - 1)$).

Por otro lado, la diferencia los segundos multiplicandos $b + b_s$ y b se debe a que ARP-Path crea un árbol para un host origen compartido por múltiples destinos, mientras que Flow-Path genera un único camino y no lo comparte, por lo que b muestra el camino medio en la topología y b_s serían el resto de ramas, compartidas como el camino, que forman el árbol que genera ARP-Path. El máximo de dicha suma sería B, que es el número total de bridges en la topología, es decir, cuando el árbol

para cierto host utiliza todos los bridges de la red y por tanto tiene entradas en todos ellos.

Si calculamos el ratio R entre el número de entradas para ver la proporción entre las entradas necesarias para Flow-Path respecto a ARP-Path, obtendríamos lo siguiente:

$$R = \frac{T_{FP}}{T_{AP}} = \frac{H*(H-1)*b}{H*(b+b_S)} = \frac{(H-1)*b}{(b+b_S)} = (H-1)*\frac{b}{b+b_S}$$

Tabla 11: Ratio entre el número de entradas para Flow-Path y para ARP-Path

Es decir, al contrario de lo que se puede pensar habitualmente, el número de entradas de Flow-Path no es tan grande como el cuadrado del número de hosts en la topología, sino que respecto a ARP-Path sólo es un múltiplo de los hosts activos en la topología y además se ve multiplicado por el factor $\frac{b}{b+b_s} < 1$ que dependerá de lo grande que sean los "árboles" generados en la topología respecto al tamaño del camino medio, y que en general podría representarse como el factor $\frac{b}{B}$, es decir, número de bridges que componen el camino medio entre número de bridges totales en la topología.

Comparativa de distribución de carga

La distribución de carga que proporciona el protocolo de ARP-Path es bastante buena como ya se demostró en apartados anteriores con las implementaciones en OMNeT++ y NetFPGA. Sin embargo, Flow-Path no comparte caminos por hosts, sino que individualiza el tráfico por flujos, lo que nos da la idea de que distribuirá mejor el tráfico, especialmente en casos en los que un host específico tenga que manejar mucho tráfico por tratarse de un servidor con cierta carga de trabajo superior a la media (un denominado *hot spot*) y sólo exista un camino hacia él en ARP-Path, cuando podrían utilizarse varios en Flow-Path.

Para comprobar la mejor distribución de Flow-Path con *hot spots*, se realizó una comparación con el simulador OMNeT++ y dos matrices de tráfico diferentes. La topología utilizada es la que se muestra a continuación con 9 switches (numerados de 1 a 9) y 25 hosts conectados a los switches de los extremos 1 y 9, de manera similar a las demostraciones de distribución de carga ya realizadas para ARP-Path.

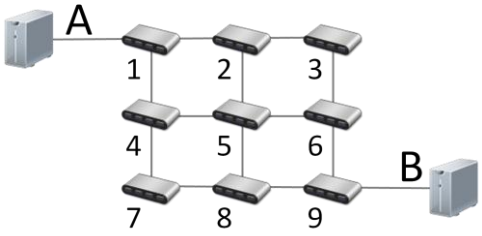


Figura 82. Reparación de caminos para Flow-Path en tres pasos: PathFail, PathRequest y PathReply

Para mostrar el mejor y el peor caso de Flow-Path en comparativa con ARP-Path se prepararon dos configuraciones de simulación, cada una con 25 flujos UDP. Cada simulación a su vez se repitió 16 veces con diferentes semillas de ejecución, y en todos los casos se midió y comparó la utilización del canal para cada enlace de la red. Para simular el mejor caso para Flow-Path, se configuró que los 25 hosts de A enviasen tráfico a un único host de B (*hot spot*), de manera que ARP-Path sólo crease un camino compartido para ese destino, mientras que Flow-Path crearía 25 caminos, igual al número de flujos, permitiendo una mejor distribución de carga. El peor caso para Flow-Path es aquel en el que iguala en distribución a ARP-Path, es decir, el caso en el que los 25 hosts de A transmiten tráfico a los 25 hosts de B, creando 25 flujos individuales (sin destinos compartidos, ni caminos compartidos para ARP-Path por tanto). A continuación se muestra una repetición, de una semilla, de cada uno de los casos.

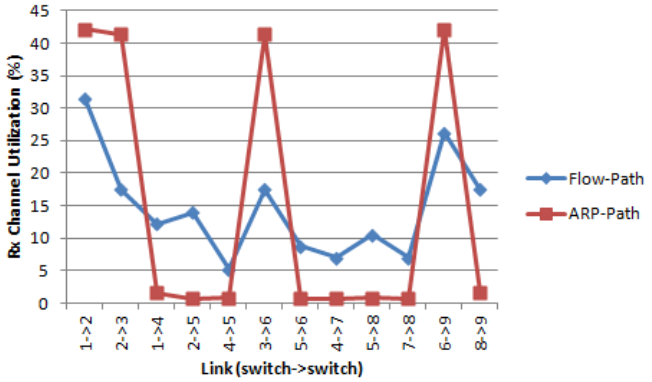


Figura 83. Comparativa de reparto de carga entre Flow-Path y ARP-Path en el mejor caso para Flow-Path

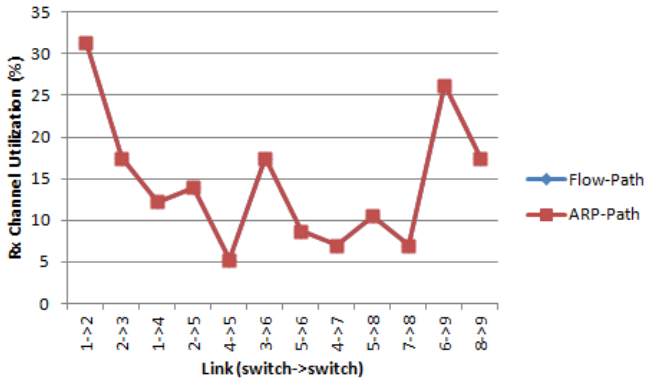


Figura 84. Comparativa de reparto de carga entre Flow-Path y ARP-Path en el peor caso para Flow-Path

Es posible apreciar que en el mejor caso para Flow-Path, ARP-Path escoge un único camino, que es el 1-2-3-6-9 mientras que Flow-Path reparte sus 25 caminos entre todos los enlaces, sin generar picos. En el "peor" caso para Flow-Path, el reparto es idéntico para Flow-Path y ARP-Path al crearse 25 caminos independientes (y estos coinciden porque la semilla es la misma para ambas simulaciones). En realidad no es un mejor y peor caso para Flow-Path (de ahí las comillas), sino que Flow-Path siempre reparte por igual (nótese que la línea para Flow-Path coincide en ambas gráficas) y es ARP-Path el que reparte algo peor cuanto más se comparten sus caminos entre los hosts. Es decir, en el caso en el que

los flujos son totalmente independientes el número de entradas para ambos protocolos coincide:

$$T_{AP} = H * (b + b_s) = H * (b + 0) = F_u * b = T_{FP}$$

Tabla 12: Caso en el que el reparto de carga y el número de entradas para Flow-Path y ARP-Path coincide: cuando todos los flujos son independientes

Puesto que no hay bridges compartidos para ningún camino y el número de flujos unidireccionales es 50 (25 en cada sentido) y coincide con el número de hosts activos, también 50. Así que coincide el número de entradas y coincide el reparto. Y es en cuanto empiezan a compartirse entradas para ARP-Path, cuando éste se ahorra entradas a costa de disminuir en parte la distribución de carga, mientras que Flow-Path sigue utilizando las mismas entradas y distribuyendo igual.

En cualquier caso es importante matizar que estos resultados se muestran para casos extremos en la comparativa de ARP-Path y Flow-Path y que en la práctica el reparto de ARP-Path es muy bueno. De hecho se realizaron largas simulaciones con el generador de flujos (Apéndice F) y diferentes pesos, y la distribución de carga fue muy similar para ARP-Path y Flow-Path, lo que demuestra que la diferencia la dan topologías, cantidades y patrones de tráfico concretas, en las que Flow-Path pueda ser más ventajoso de utilizar frente a ARP-Path a costa, lógicamente, de un mayor número de entradas en las tablas de encaminamiento.

Conclusiones

El uso de Flow-Path es recomendable frente a ARP-Path en el caso de data centers con *hot spots* en el que es recomendable que los caminos no se compartan y el tráfico se reparta aún mejor, con el mayor coste de almacenamiento.

Otra ventaja que ofrece Flow-Path es que sus caminos siempre son simétricos. Pero como desventajas tenemos la necesidad de usar cierta información IP (direcciones) para establecer los caminos, una lógica de generación de caminos ligeramente más complicada y el hecho de que la reparación es más costosa también, puesto que se tiene que reparar cada flujo por separado, mientras que en ARP-Path se comparten caminos para un mismo host y una trama puede reparar el camino de diferentes flujos.

4.3 ARP-Path “asterisco” (ARP-Path*)

El denominado ARP-Path “asterisco” (ARP-Path*) surge como idea teórica de fusión de ARP-Path y Flow-Path, cogiendo las ventajas de cada uno de manera adaptativa.

Como ya se conoce de apartados anteriores, ARP-Path ofrece buena distribución de carga en la red, pero Flow-Path aún la mejora más para redes en las que existan

diversos caminos para un mismo host y no se aprovechen, sobre todo en los casos en los que algún host tenga una carga de trabajo superior al resto (los denominados *hot spots*) y sea especialmente necesario distribuir su tráfico y no compartir caminos con otros hosts. Sin embargo, Flow-Path aumenta el número de entradas de las tablas de encaminamiento de manera considerable, por lo que no es una opción que se pueda usar por defecto y se deben analizar primero las características de la red para decidir si debe ser utilizado o no. Por lo tanto ARP-Path* surge como una alternativa derivada de la última versión de ARP-Path (ARP-Path unidireccional alternativo/retardado), que se adapta a las características de la red para fusionarse en parte con Flow-Path cuando así se necesite, distribuyendo la carga sólo así cuando se necesite e incrementando el número de tablas el mínimo respecto a ARP-Path.

A continuación se muestra la idea de funcionamiento de ARP-Path*, para después pasar a realizar una comparativa con ARP-Path y Flow-Path. A diferencia del resto de protocolos de la presente Tesis, ARP-Path* aún no ha sido implementado en ninguna plataforma por lo que su estudio es meramente teórico.

4.3.1 Funcionamiento de ARP-Path*

El funcionamiento base de ARP-Path* es idéntico a ARP-Path. Así que en los apartados siguientes simplemente se matizarán las diferencias y lo que hace de este protocolo un paso intermedio entre ARP-Path y Flow-Path.

4.3.1.1 Creación de caminos

A la hora de crear caminos en una red desde cero, es decir, sin caminos preexistentes, tal y como se mostraba en las explicaciones de anteriores protocolos, no hay diferencia entre ARP-Path* y ARP-Path. Las diferencias surgen cuando ya existe una entrada que afecta a las decisiones de un ARP Request o ARP Reply, dado que las decisiones tomadas ahora son diferentes para el caso del ARP Reply.

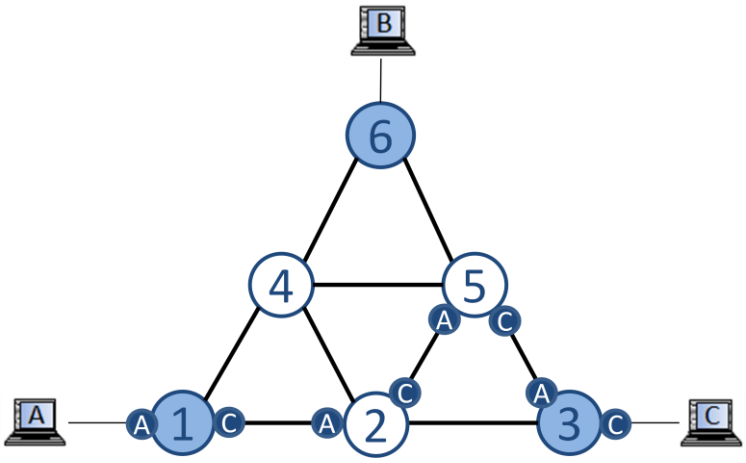


Figura 85. Ejemplo de comunicación estable entre un par de hosts con ARP-Path*

Redescubrimiento y regeneración del camino hacia el origen (*ARP Request*): Para explicar qué sucede con el ARP Request, tomemos como referencia la imagen anterior en la que existe un camino ya establecido entre un par de hosts A y C. Si se emite un nuevo ARP Request desde C hasta A, el procedimiento seguido es exactamente el mismo al que se realiza en ARP-Path: se apunta la dirección alternativa temporalmente hasta que se refresca la ya existente.

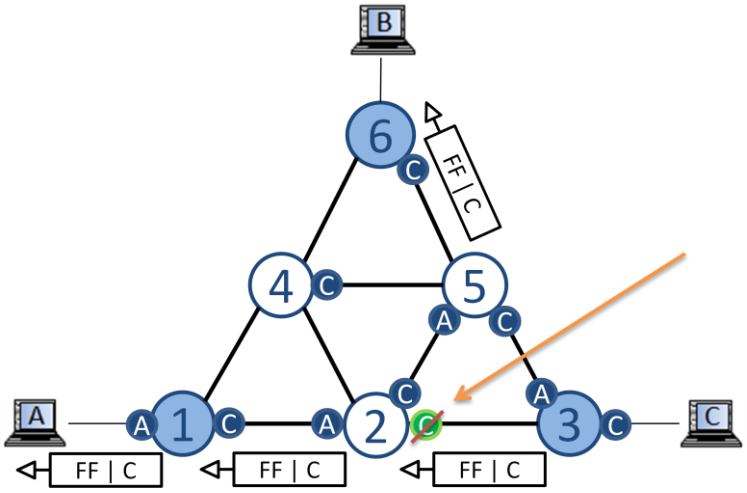


Figura 86. El ARP Request toma las mismas decisiones en ARP-Path* que en ARP-Path (unidireccional alternativo/retardado)

Regeneración del camino hacia el destino (*ARP Reply*): Tomemos de nuevo el ejemplo de la Figura 85 con comunicación estable. Entonces ahora B quiere comunicarse con A y emite un ARP Request, al que A responde con un ARP Reply. Como se ve en las siguientes figuras, el aprendizaje de B no se ve afectado, dado que no había nada aprendido previamente de B. Sin embargo, el aprendizaje de A sí se ve afectado, puesto que el ARP Reply se encuentra con una entrada preexistente de A al llegar al puente 5. Entonces la decisión que se toma es añadir una nueva entrada, pero para distinguirla de la anterior se añade con formato de flujo (Flow-Path), es decir, como AB, mientras que la entrada de A anterior se mantiene genérica para el resto de flujos como A*, de ahí el nombre de ARP-Path*, de ese asterisco que se aplica a las entradas genéricas.

La diferencia por tanto de ARP-Path* con ARP-Path es que éste nunca cambiaba entradas ya existentes con el ARP Reply y ARP-Path* no las cambia, pero añade una entrada extra para el nuevo flujo. Además se asegura que el flujo es nuevo porque el ARP Request se mantiene igual. Es decir, si se generase un ARP Request desde A o desde C, el camino se conservaría en ambos sentidos y el ARP Reply de respuesta a cada petición utilizaría siempre el camino ya existente, sin posibilidad de llegar dicha trama por un puerto diferente. Por lo que las tramas ARP Reply que llegan por un puerto diferente y obligan a añadir una nueva entrada son, obligatoriamente, de un flujo diferente.

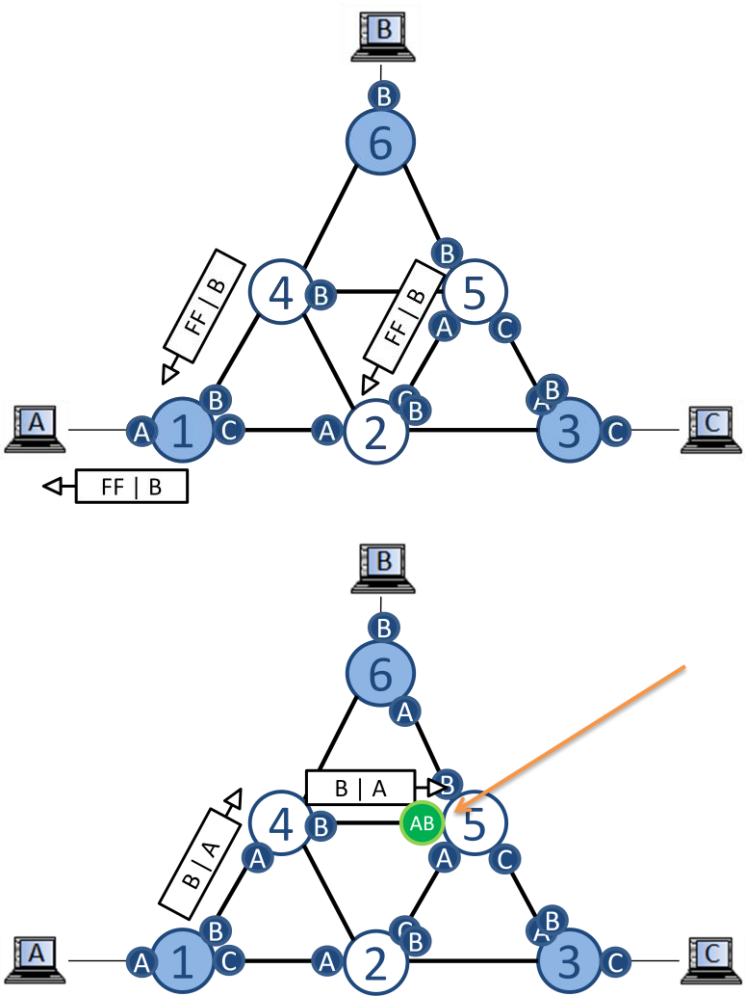


Figura 87. Añadiendo una entrada de flujo AB en un puente que ya posee una genérica A* en ARP-Path*

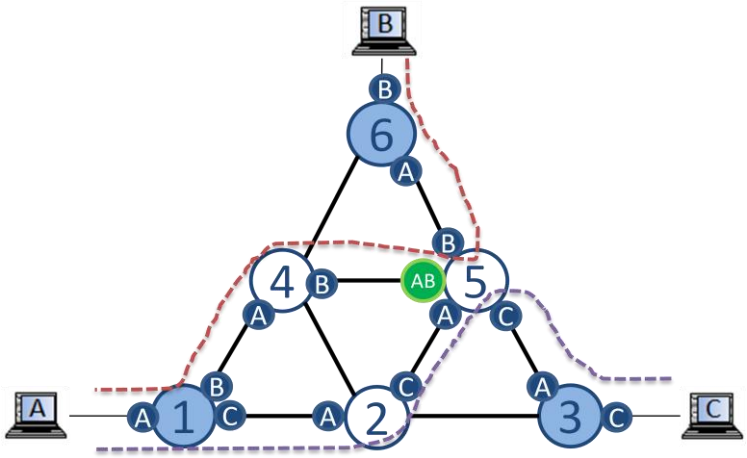


Figura 88. Añadiendo una entrada de flujo AB en un puente que ya posee una genérica A* en ARP-Path*

El resultado a la hora de encaminar es el que se muestra en la anterior figura. Ahora existen dos caminos hacia A, uno es 1-4-5-6 y el otro 1-2-5-3, que sería sólo un camino compartido en el caso de ARP-Path, uno sería 1-2-5-6 y el otro 1-2-5-3,

compartiendo todo el tramo 1-2-5 y dejando el puente 4 y los enlaces hacia él sin usar. De este modo, cuando una trama llega al puente 5 con destino hacia A, ahora se realiza una doble búsqueda: primero se busca si existe la entrada concreta de su flujo (destino-origen) y, si no existe, lo segundo es buscar la entrada genérica de A; si finalmente la entrada no se encontrara de ninguna de las dos formas, comenzaría el proceso de reparación.

Con este método se crea una fusión entre ARP-Path y Flow-Path, creando caminos extras basados en el flujo cuando así se necesita por el tráfico, pero sin tener que añadir entradas de flujo para todos los casos (en el ejemplo anterior hay sólo una entrada de flujo y el resto son genéricas). Es decir, ARP-Path* aprovecha la ventaja de Flow-Path de distribución de carga sin tener que aumentar tanto el número de entradas y sin necesitar las IPs, y además añade ventajas en la reparación (como se verá a continuación), pero no conserva la simetría en todos los casos (a continuación se demuestra) y necesita dos búsquedas por tabla en lugar de una que necesitan ARP-Path y Flow-Path.

De forma adicional, se podría analizar el hecho de seguir añadiendo entradas aún más específicas, no sólo del estilo Flow-Path apuntando origen y destino, sino incluso apuntando más datos del flujo, como pueden ser el tipo de protocolo de transporte, los puertos de conexión, etc.

Nótese que los caminos alternativos sólo se crean con el ARP Reply para garantizar que son de otro flujo (y además para evitar la necesidad de utilizar las IPs para diferenciarlos), aunque debería realizarse un análisis más profundo para ver si en casos tras caída de enlaces sigue cumpliéndose la misma condición.

4.3.1.2 Reparación de caminos

La reparación de caminos en ARP-Path* es idéntica a la que se realiza en ARP-Path con la excepción de que ahora hay casos en los que no es necesario aplicarla al disponer de diversos caminos para un mismo destino en ciertas situaciones. En todo caso, las tablas y mensajes especiales utilizados coinciden en caso de necesitar reparar.

Por ejemplo, tomando la última imagen en la que disponíamos de dos entradas para A en el puente 5, una específica para el flujo entre A y B (AB) y otra genérica (A*), se pueden dar dos situaciones tal y como se ve en las siguientes figuras: que se caiga el enlace asociado a la entrada específica o el de la genérica. En el caso de que se caiga la específica, se borraría sin más y la genérica comenzaría a aplicarse en el encaminamiento; mientras que si se borra la genérica, se tomaría alguna de las específicas (si hay más de una puede aplicarse el criterio de la última que fue refrescada, por ejemplo) como nueva genérica. Lógicamente, el camino hacia B o hacia C sí tendrían que ser reparados, pero el de A no, evitando una de las reparaciones sin que las tramas siquiera sean conscientes de ello.

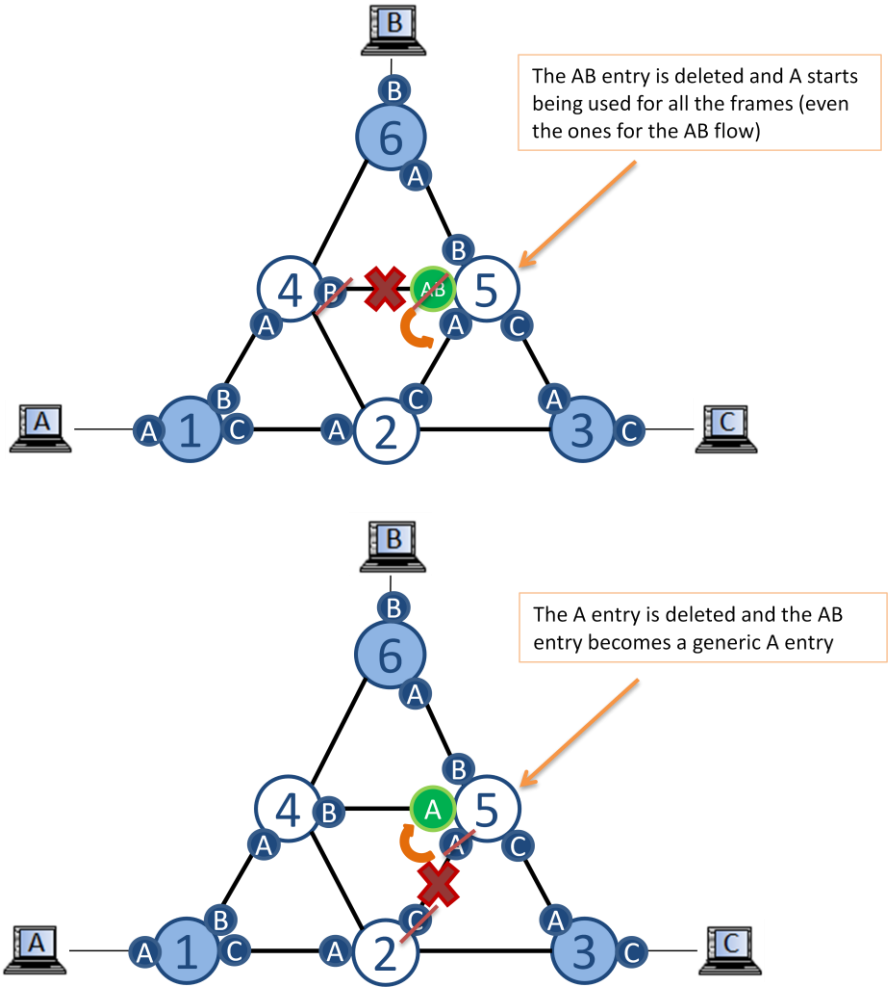


Figura 89. Caída de un enlace para un puente que posee más de una entrada para un destino, 2 casos: arriba, caso en el que se cae el enlace de la entrada específica y abajo, caso en el que se cae el enlace de la entrada genérica (o asterisco)

El hecho de reparar de esta forma instantánea los caminos crea situaciones en las que la simetría de caminos se pierde, tal y como se muestra en las figuras a continuación. Por lo tanto, el *loopback* de tramas queda inicialmente descartado al igual que con ARP-Path. Si Flow-Path aplicase el mismo método también perdería la simetría que posee en sus caminos, pero no sólo eso, sino que no refrescaría adecuadamente, al no tener *forward refresh*, así que no sería viable.

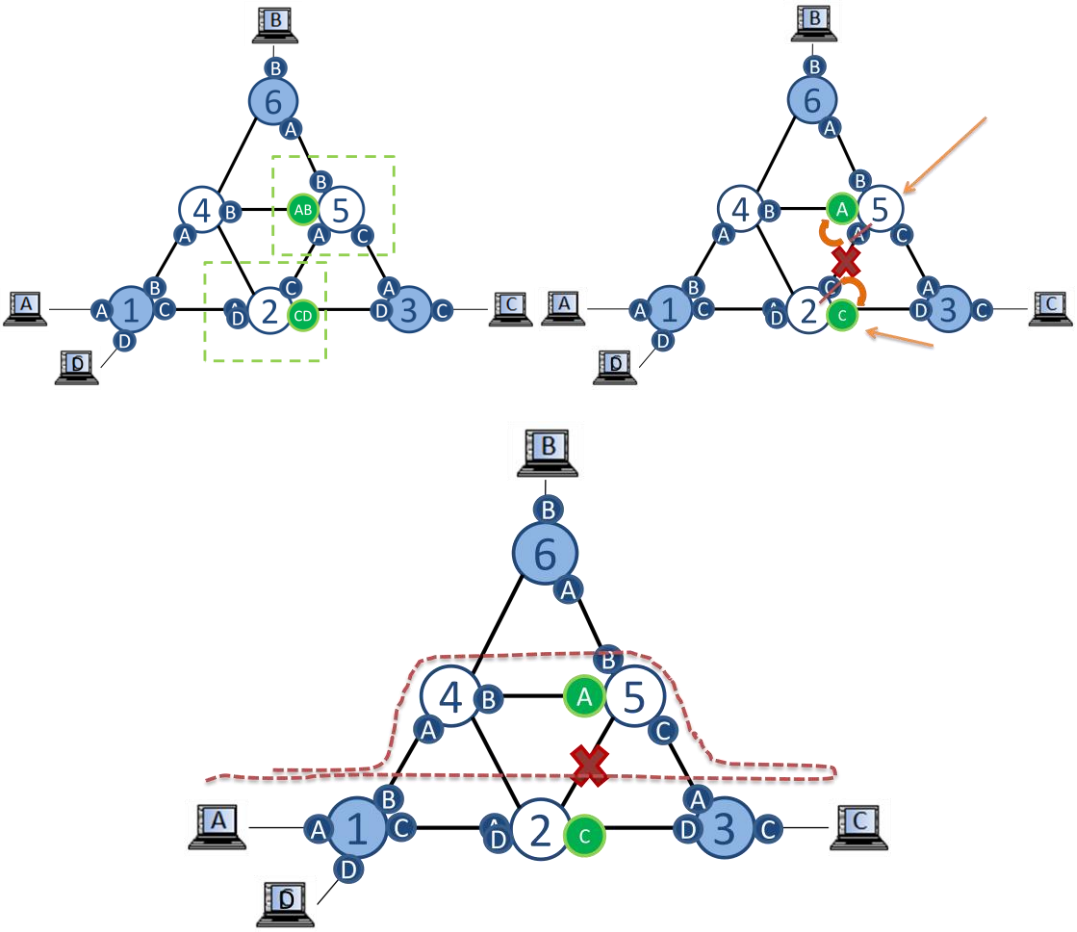


Figura 90. Caída de un enlace que afecta a dos puentes con más de una entrada, la entrada CD del puente 2 pasa a ser C* y la AB del 5 pasa a ser A*, de manera que el camino entre A y C ahora es asimétrico

4.3.2 Evaluación de ARP-Path* frente a ARP-Path y Flow-Path

Sería necesario realizar una implementación de ARP-Path*, por ejemplo con OMNeT++ y un análisis teórico, para hacer una comparativa rigurosa, pero los matices clave se resumen en la siguiente tabla.

Por supuesto, todavía queda mucho por explorar además en ARP-Path*.

Característica	ARP-Path	ARP-Path*	Flow-Path
Distribución de carga	Buena	Buena y adaptable	Muy buena
Número de entradas	Bajo	Bajo (sube según aumenta la distribución adaptada)	Mayor que ARP-Path (depende de los hosts activos y la topología)
Simetría de caminos	No (no necesaria)	No (pero más habitual que en ARP-Path)	Sí (y necesaria por el refresco)
Tipo de refresco	Hacia adelante (*forward refresh*)	Hacia adelante (*forward refresh*)	Hacia atrás
Número de búsquedas en tabla	1 por dirección	2 (1 por flujo y 1 por dirección)	1 por flujo
Reparación	"Hacia adelante" (PathFail broadcast hacia destino a reparar y PathRequest broadcast)	"Hacia adelante" (igual que ARP-Path, pero a veces no es necesario reparar...)	"Hacia atrás" (*loopback* de tramas, PathRequest broadcast y PathReply unicast; necesario flag de reparción)

Tabla 13: Resumen comparativo de ARP-Path, Flow-Path y ARP-Path*

4.4 ARP-Path multipath

En este última sección de las contribuciones al encaminamiento en redes empresariales se muestra ARP-Path multipath que, como su nombre indica, se trata de una variante multicamino (*multipath*) del protocolo ARP-Path. El mecanismo básico de ARP-path se basa en la exploracion simultánea de caminos múltiples en la red, por lo que parece en principio razonable estudiar su uso para el establecimiento de caminos múltiples. La idea de ARP-Path multipath tenía como fundamento el hecho de crear una versión multicamino de ARP-Path, que distribuyera el tráfico de ARP-Path por múltiples caminos de igual coste, al igual que sucede con ECMP en SPB. Sin embargo, en la práctica se ha demostrado que ARP-Path de por sí es un protocolo que tiende a distribuir de manera muy eficaz el tráfico por toda la red, lo que ha hecho que la exploración de ARP-Path multipath quede algo postergada.

Se investigaron varias aproximaciones, de las que a continuación se muestra una, en concreto la que utiliza la etiqueta VLAN para generar los múltiples caminos.

4.4.1 Funcionamiento de ARP-Path multipath

El funcionamiento base de ARP-Path multipath es análogo a ARP-Path, con la diferencia de que ahora se construye más de un único camino con la misma exploración gracias a la utilización de la etiqueta VLAN con un fin específico multicamino.

4.4.1.1 Creación de caminos

A continuación se muestra el aprendizaje realizado en ARP-Path multipath en base al uso de la etiqueta VLAN. En este caso, no hay diferencia entre el ARP Request y el ARP Reply, pues ambos se emiten en broadcast al ser necesario para que puedan descubrir todos los caminos posibles para ambos hosts y no sólo para el origen. Alternativamente, para que el ARP Reply no se propague por la red cada vez que sea necesario, se podrían utilizar mensajes especiales en lugar del ARP Request y ARP Reply, emitidos periódicamente, en lugar de irregularmente como sucede con el ARP.

Descubrimiento y generación del camino hacia el origen y hacia el destino (*ARP Request/ARP Reply*): Cuando se emite un mensaje ARP (del tipo que sea), éste inicialmente llegará desde el host origen que lo emite hasta el puente frontera, entonces el puente frontera le añade el etiquetado VLAN para crear los diferentes caminos y lo propaga por toda la red.

Dicho etiquetado consiste en ir numerando las etiquetas VLAN de manera jerárquica. Dicha jerarquía se puede crear con los 12 bits disponibles en el identificador de VLAN (VID). Por ejemplo, en las siguientes figuras se muestra un caso de ARP-Path multipath en el que las jerarquías se forman con las decenas, es decir, del 1 al 9 estamos en el primer nivel de la jerarquía, que a su vez derivan a 9 niveles más: del 11 al 19 desde el 1, del 21 al 29 desde el 2, etc. Se pasa de un nivel a otro cada vez que la trama llega a un puente, de manera que podemos ver por ejemplo en la siguiente figura cómo la trama pasa a tener VID=1 tras el puente 1, VID=11 al llegar al puente 4 y VID=111 al atravesar el puente 7, y así indefinidamente.

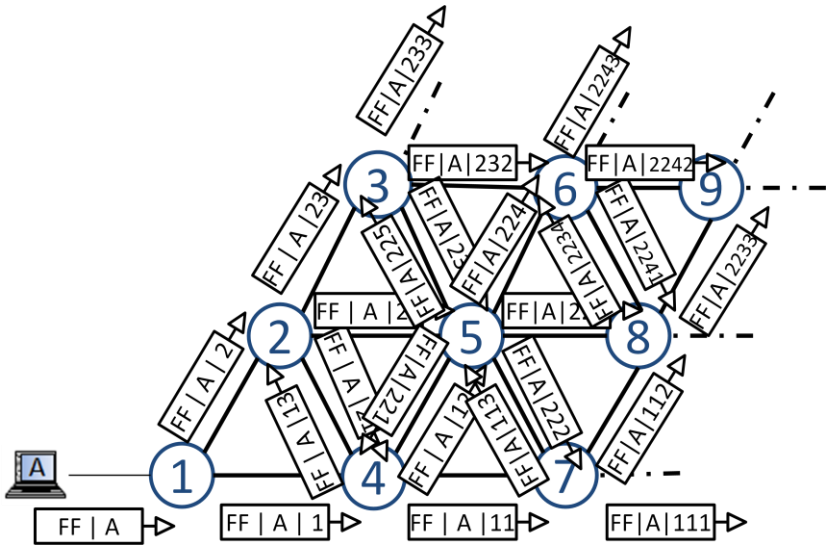


Figura 91. Ejemplo de etiquetado jerárquico de VLANs en ARP-Path multipath

Así pues, las tramas con el mensaje ARP son reetiquetadas en cada puente para ir creando la jerarquía, mientras que el aprendizaje sigue siendo igual: la primera copia se apunta como la más rápida, propagándose por todos los puertos menos por el que llegó y posteriores tramas son descartadas. La diferencia es que ahora ARP-Path multipath utiliza dicha jerarquía para apuntar caminos alternativos y entonces si una trama es una copia tardía, se descartará, pero antes se apuntará como camino alternativo si porta con ella una VID distinta a la VID que portaba la trama que llegó la primera. En la siguiente figura se muestran las diferentes entradas del camino creado hacia el host A, donde en cada puerto se indica el VID asociado a la entrada principal en azul oscuro y los VIDs alternativos en azul más claro.

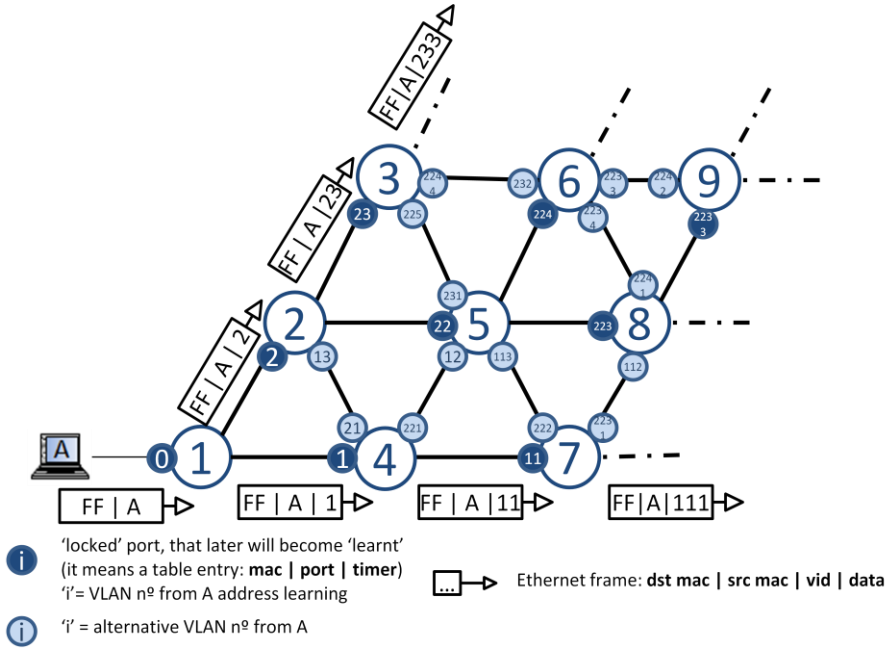


Figura 92. Aprendizaje de multicaminos con los mensajes ARP en ARP-Path multipath

Dado que el tamaño de la etiqueta VLAN tiene un número de bits limitado, habrá cierto punto en el que la jerarquía no podrá crear más niveles. Al llegar a ese punto,

no se crean más niveles y entonces se apuntarán menos caminos alternativos, pero por lo demás todo sigue igual. A continuación un ejemplo en el que el final de la jerarquía se acaba en el número 99 y ya no se pueden crear más niveles.

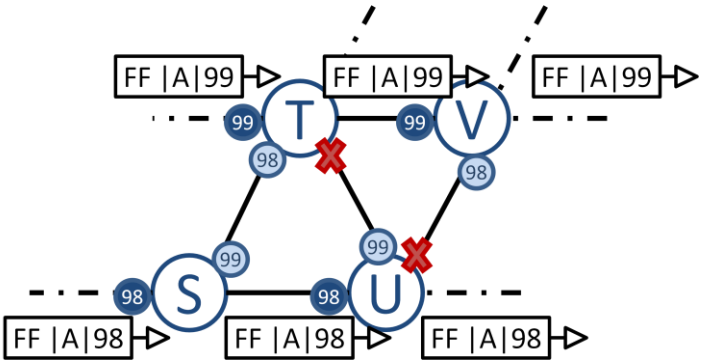
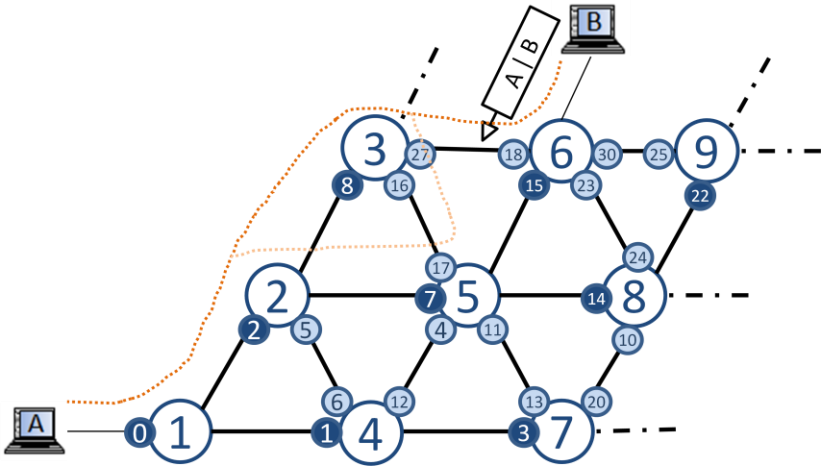


Figura 93. Ejemplo de fin de etiquetado jerárquico de VLANs en ARP-Path multipath

Una vez el camino hacia cierto host ha sido creado, la trama puede escoger cualquiera de los caminos posibles que le ofrece cada puente. Sólo sabe que cierto camino es el primero y el resto alternativos, la toma de decisiones puede basarse en otros parámetros, como las características del flujo o la utilización del enlace en ese momento. La única condición es que cada dos pasos, en uno de ellos se debe "subir" en la jerarquía, es decir, pasar a una camino alternativo con VID de una jerarquía superior, para evitar los bucles. Así pues, un bit de la etiqueta VLAN se utilizará para indicar si se subió o no en el último paso y conocer esta condición, quedando 11 bits libres para numerar los niveles de jerarquía. En las siguientes figuras, se muestran algunos ejemplos de encaminamiento.



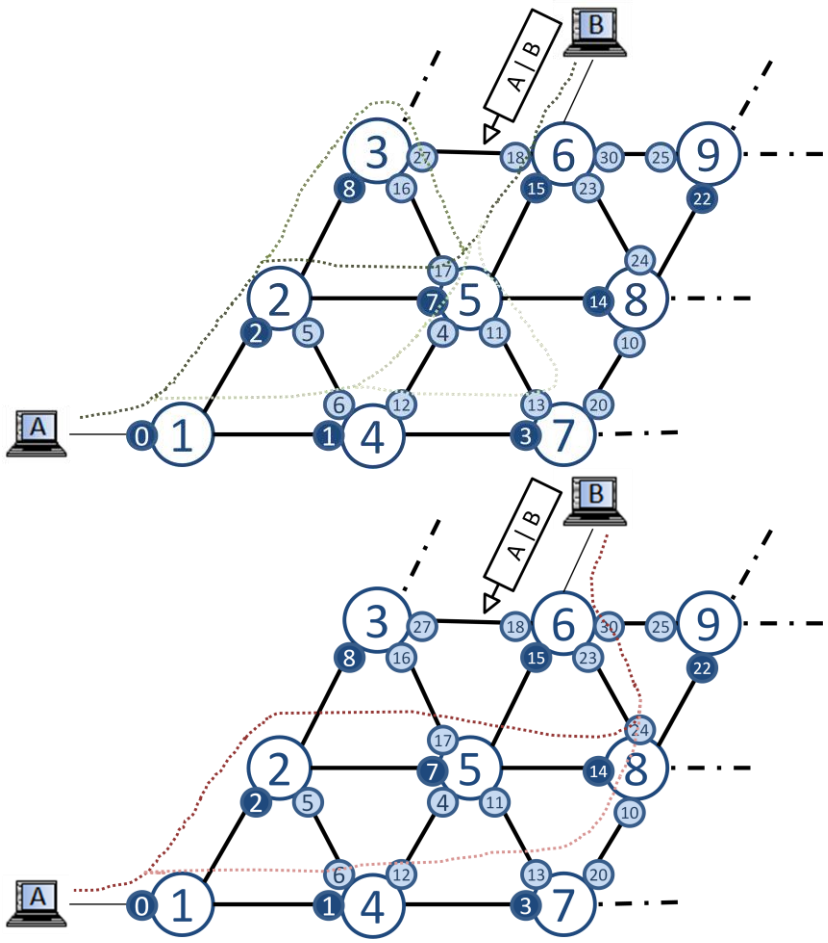


Figura 94. Ejemplo de fin de etiquetado jerárquico de VLANs en ARP-Path multipath

4.4.1.2 Reparación de caminos

La reparación de caminos de ARP-Path multipath es idéntica a ARP-Path, salvo que ahora se necesita un campo en las tramas especiales de reparación que indique el VID o, en su caso, se necesita seguir utilizando la etiqueta VLAN en los nuevos mensajes ARP generados para reparar el camino.

Sin embargo, en ARP-Path multipath sucede como en ARP-Path*, cuando un camino se cae, siempre se siguen teniendo caminos alternativos, por lo que en la práctica pocas veces se repararán los caminos: se hará cuando se caigan todos los alternativos o quizás un porcentaje de ellos.

4.4.2 Implementación en OpenFlow/Mininet

El protocolo ARP-Path multipath fue implementado en OpenFlow y probado en la máquina virtual Mininet sobre la base de la segunda versión de ARP-Path (ARP-Path unidireccional, no alternativo/retardado), con satisfactorios resultados en los que se escogían diferentes caminos para encaminar tráfico hacia un mismo destino. Además, al caerse enlaces no se iniciaba la reparación, y se seguía encaminando,

hasta que se borraban todas las entradas y no quedaban más caminos posibles para dirigirse hacia cierto destino.

4.4.3 Evaluación de ARP-Path multipath frente a ARP-Path y Flow-Path

Para comparar en detalle ARP-Path multipath con los protocolos ya existentes en la familia All-Path, sería necesario estudiarlo más a fondo. Podemos hacer sin embargo una evaluación cualitativa. Si realizáramos una tabla como la que se hizo para ARP-Path*, podríamos decir que ARP-Path multipath se encuentra en una posición intermedia similar a la de ARP-Path*: con distribución muy buena (incluso con más múltiples caminos en ARP-Path multipath que en ARP-Path* por defecto), aunque no adaptable (se crean siempre los caminos), un número de entradas relativamente bajo (depende del número de los hosts activos y el número de puertos por bridge, y no de casi el cuadrado de los hosts activos como en Flow-Path), una única búsqueda en tabla y reparación que en la mayoría de los casos no será necesaria debido a los múltiples caminos que existirán para todos los hosts.

Como desventajas principales se presenta el hecho de que los puentes tienen que analizar y reetiquetar todas los mensajes ARP, lo que es una labor costosa, y además se puede dar el caso de que cierto mensaje ARP sea especialmente rápido, gane la carrera rápidamente a todos los puentes de la topología y deje de generar multicaminos. Además, el sistema basado en VLAN jerarquizadas no se adapta bien a la necesidad de un número bajo-medio de caminos múltiples y tiende a crear muchos multicaminos más cerca del host destino, pues según nos alejamos del host, los VIDs se van incrementando, y agotando, y llega un momento que dejan de apuntarse tantos caminos alternativos o incluso puede que no se apunte ninguno más.

Capítulo 5

Contribuciones al encaminamiento en centros de datos

En este capítulo se describen las contribuciones realizadas en la presente Tesis respecto al ámbito del encaminamiento en centros de datos. Concretamente se estudia el protocolo y arquitectura Torii-HLMAC. Nótese que aunque se muestra como un capítulo aparte, podría considerarse una subsección, de redes más concretas, dentro del capítulo anterior.

5.1 Torii-HLMAC

A lo largo del Capítulo 4 se describió la familia All-Path partiendo desde el comienzo del protocolo ARP-Path. Esta familia es perfectamente aplicable en redes de centros de datos. Sin embargo, como se exploró en el capítulo dedicado al estado del arte, los protocolos específicos para redes de centros de datos utilizan en muchos casos topologías de red específicas y conocidas, con planteamientos diferentes a los modelos de redes empresariales, dado que los modelos de redes

para centros de datos tienden más a expandirse en recursos (utilizando dispositivos más baratos) en lugar de actualizarlos a recursos más potentes, lo que se denominaba comparativa del modelo *scale-out* frente al *scale-up* [Vah+10].

Por lo tanto, partiendo de una topología de red más o menos concreta, podemos delimitar ciertos parámetros y simplificar ciertas partes de las comunicaciones, y de ahí surge Torii-HLMAC [Roj12]: una arquitectura distribuida y tolerante a fallos para centros de datos, con direccionamiento múltiples basado en árbol. Esta propuesta parte de simplificar la arquitectura de PortLand [Mys+09], pero luego es generalizable a muchas otras, como veremos en apartados posteriores.

5.1.1 Introducción

En la presente sección de Torii-HLMAC (鳥居HLMAC), en adelante Torii, exploraremos una combinación de funciones distribuidas para hacer simple y más escalable el envío de tramas en *fat trees*, entre otras arquitecturas. Uno de los objetivos de Torii es mejorar PortLand con mecanismos alternativos, distribuidos y, a su vez, más sencillos. Para ello, Torii se vale del uso de direcciones pseudo MAC topológicas, pero no una única dirección por nodo, sino asignando múltiples direcciones simples (inspirándose en TRE [Iba+09]), de manera que se facilita el envío de tramas multicamino o multipath, así como se garantiza la tolerancia a fallos, dado que es posible encaminar directamente la trama por un camino alternativo sin necesidad de tablas.

5.1.1.1 Origen del nombre del protocolo

Como curiosidad, se va a comentar la procedencia del nombre del proyecto Torii-HLMAC. La idea era buscar un nombre que incluyese las siglas HLMAC (por el uso indispensable en el protocolo de direcciones *Hierarchical Local MAC*, HLMAC, explicadas en el Apéndice D) y a la vez la arquitectura de centro de datos para la que estaba diseñada, un *fat tree*. De ahí, la primera idea que viene a la cabeza es Tree-HLMAC, sin embargo, parece un nombre poco llamativo, así que se pretendía darle un toque más exótico.

Así pues, si transcribimos (que no traducimos) la palabra *tree* al japonés, nos surge *tsurii*, que es una palabra que ellos usan para denominar al árbol de Navidad. Si somos un poco más rebuscados y transcribimos *tree* al japonés, pero siendo pronunciado por un hispanohablante (que lo pronunciaría algo así como "*tri*"), la transcripción nos queda *torii*, que curiosamente es un elemento típicamente japonés, que a su vez tiene cierta forma similar a un *fat tree* (más "gordo" en lo alto, y con ramas hacia abajo).

Un torī (en japonés 鳥居) es un arco tradicional japonés que suele encontrarse a la entrada de los santuarios Shinto (Jinja), marcando la frontera entre el espacio profano y el sagrado. Consisten de dos columnas sobre las que se sustentan dos travesaños paralelos, frecuentemente coloreados de tonalidades rojas o bermellonas. Algunos poseen tablas escritas montadas entre las barras horizontales.

Tradicionalmente, los torī eran de madera o piedra, pero recientemente se han comenzado a hacer en acero o acero inoxidable.

Figura 95. Torii del Santuario Itsukushima (isla de Itsukushima, Prefectura de Hiroshima)

5.1.2 Funcionamiento de Torii-HLMAC

En el siguiente apartado se muestra la base de funcionamiento de Torii y, para ello, se toma como ejemplo la topología de PortLand.

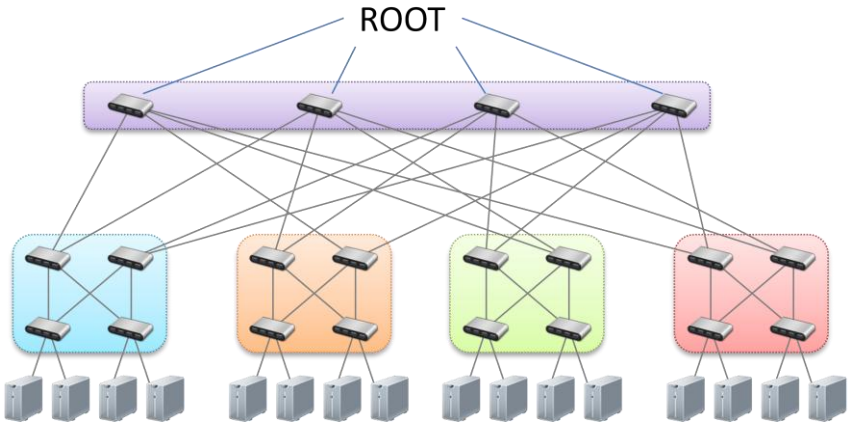


Figura 96. Topología de PortLand, escogida para la descripción del protocolo Torii-HLMAC

5.1.2.1 Creación de caminos

Un aspecto esencial de Torii es que no utiliza tablas de encaminamiento. En ARP-Path, éstas son muy sencillas y se crean en base a la inspección de los mensajes ARP. De hecho, en general en la familia All-Path una evaluación constante consiste en la comparación del tamaño de dichas tablas (más grandes en Flow-Path, menores en ARP-Path, etc). Sin embargo, Torii no necesita crear, ni hacer uso, de ningún tipo de tabla de encaminamiento porque los puentes adquieren una identidad al

comienzo, en la instalación de la red, y con dicha identidad y las direcciones origen y destino de las tramas basta para poder encaminar cualquier trama hacia el destino.

Estructura de direcciones múltiples basada en árbol y asignación automática con RSTP extendido

Como se acaba de comentar, Torii reparte una serie de "identidades" por todos los puentes de la red. Estas identidades son direcciones MAC jerárquicas locales (*Hierarchical Local MAC*, HLMAC), véase Apéndice C para más detalle.

Torii asigna una dirección HLMAC a cada puerto conectado "hacia arriba" de cada puente en la topología, como se muestra en la figura a continuación. La asignación se realiza de arriba abajo, donde "arriba" es la posición de un puente que se añade actuando como raíz (que puede ser virtual) y "abajo" es la posición de los puentes frontera desde los que los servidores se conectan a la red. Primero se asigna una dirección HLMAC a los puentes superiores, el siguiente nivel de puentes tendrá dos direcciones HLMAC (uno por cada enlace a los puentes superiores) y continúa así, según vamos bajando, en potencias de dos. El hecho de que el número de HLMACs aumente en potencias de dos no es algo característico de Torii, sino en este caso de la topología PortLand, en el apartado de generalización de topologías de matizará más sobre este punto.

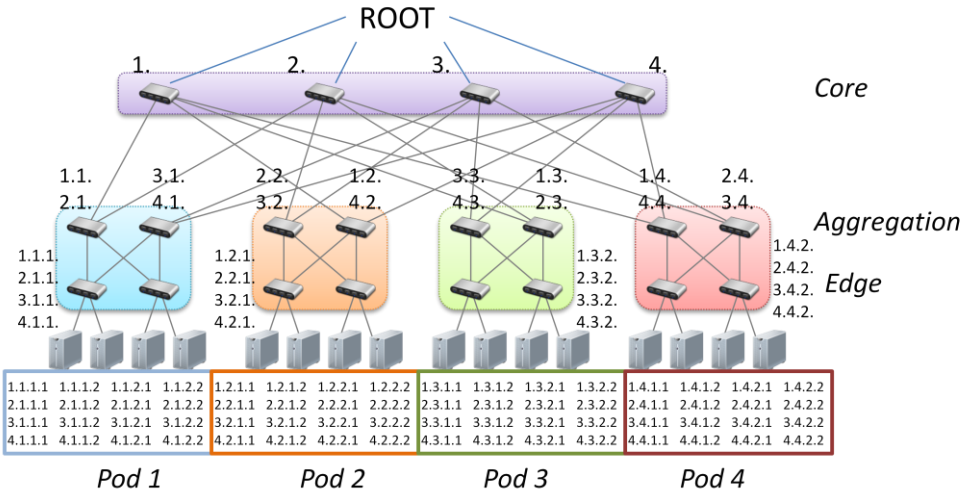


Figura 97. Asignación de múltiples direcciones jerárquicas (HLMAC) para Torii con el protocolo RSTP extendido, desde un nodo "ROOT" (raíz) virtual

Como se conoce del Apéndice C, las direcciones HLMAC son direcciones MAC locales (privadas), es decir, direcciones cuyo bit U/L está fijado a 1. Dado que las MAC poseen 48 bits y el primero define si la trama es broadcast/multicast o unicast y el segundo estará fijado a 1 (bit U/L), los 46 bits restantes disponibles para el propósito del direccionamiento codificarán por defecto hasta 6 niveles jerárquicos diferentes, con 6 bits para el primero y 8 para cada nivel restante. La HLMAC de un puente se expresa de la siguiente forma con puntos `a.b.c`… que se trata de la cadena de IDs de los puertos designados a, b, c,… atravesados en el camino descendiente desde el puente raíz (root) hasta el puente donde la dirección está siendo asignada. A la hora de representar una HLMAC, si ésta termina en ceros estos se suprimen y sólo se indica el último valor seguido de '.', para simplificar; además

también se ignora el valor del bit U/L a 1, suponiendo que el primer byte sólo tiene 6 bits.

Para construír el árbol de expansión y asignar direcciones jerárquicas a los puentes, se utiliza una versión modificada del protocolo RSTP, RSTP Extendido o *Extended RSTP*, que se define en HURP [Iba+10c]. Sin embargo, este método de asignación es independiente del funcionamiento de Torii en sí, que sólo necesita dichas "identidades" (HLMACs) asignadas para funcionar.

El proceso se describe a continuación. El primer paso es fijar la prioridad del puente superior (que puede ser virtual o no, pero en todo caso no soportará ningún tipo de tráfico, pues se considerará fuera de la topología) como máxima, para que siempre sea el raíz. Una vez que el puente raíz es asignado, éste obtiene la dirección HLMAC 0. (es decir 0.0.0.0.0.0) y el proceso de construcción del árbol de expansión comienza desde el raíz hacia las hojas. Este proceso iterativo consiste en BPDUs que son enviadas por el puente padre, incluyendo el número de puerto del puerto designado. Éste número de puerto tendrá un rango de valores de 1 a 4 (concretamente en esta topología, dado que los switches que se usan son todos de 4 puertos, y el puerto 0 no se utiliza), que corresponderá con el número de pod al que el puerto está conectado, como se muestra en la Figura 97. Por ejemplo, en el pod 1, el primer switch de agregación tiene la dirección 1.1. (del core switch 1 y el puerto designado 1) y también la 2.1. (del core switch 2 y el puerto designado 1) como direcciones HLMAC.

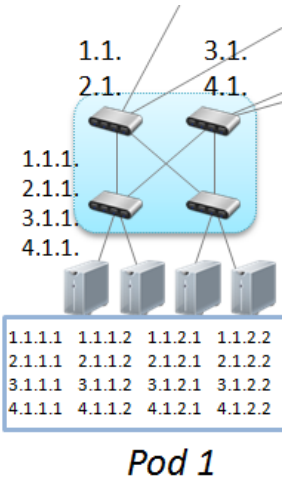


Figura 98. Asignación de múltiples direcciones jerárquicas para el pod 1

Como consecuencia de este proceso, cada nodo obtiene una o más (hasta cuatro en el caso de la topología que se usa en la descripción) direcciones HLMAC topológicas del árbol. El número de direcciones HLMAC alternativas en los switches frontera coincidirá con el número de switches core (nivel superior), en este caso cuatro, y el prefijo de la dirección HLMAC se utilizará para distribuír el tráfico basándose en la aplicación de una función hash. Concretamente, para verlo más claro, si nos centramos en el caso del pod 1 de la figura recién mostrada, nos percataremos de que cada host tiene cuatro direcciones asignadas. En el switch de agregación de la izquierda tenemos 1.1. y 2.1. (de los switches core 1 y 2 respectivamente) y en el de la derecha tenemos 3.1. y 4.1. (de los core 3 y 4). Si bajamos un nivel, a uno de los switch frontera (concretamente el que está más a la

izquierda), éste está conectado a los dos switches de agregación y por tanto tiene las cuatro direcciones siguientes 1.1.1., 2.1.1., 3.1.1. y 4.1.1. Finalmente, el host de la izquierda, de ese mismo switch frontera, observamos que tienen las direcciones asignadas 1.1.1.1., 2.1.1.1., 3.1.1.1. y 4.1.1.1., todas ellas válidas a la hora de realizar el encaminamiento.

El rol de puente raíz puede ser implementado como una pareja de puentes raíz compartiendo por completo su sistema de direccionamiento hacia los puertos descendientes, con enlaces redundantes, de manera que se garantice que los puentes 1, 2, 3 y 4 siempre obtengan la misma dirección, mediante ambos enlaces. Los puentes raíz no transportan información de usuario porque el plano de "reflexión" está situado en los puentes core (1, 2, 3 y 4), así que su diseño se centra sólo en la dependencia inicial de asignación de direcciones.

Los hosts no guardan las direcciones como tal (pues no son conscientes del protocolo, ni dependen de él a la hora de enviar el tráfico), sino que son los puentes frontera los que son conscientes de que tienen hosts conectados (no habrán recibido ninguna BPDU por los puertos conectados a hosts) y los que guardan esa asociación de direcciones para realizar las traducciones más adelante. Nótese que entre las cuatro direcciones la única diferencia es el prefijo, que coincide con el core switch que se ha de atravesar a la hora de realizar el encaminamiento. De manera que las direcciones son fácilmente deducibles unas de otras si así se necesita, dado que lo único que varía es dicho prefijo.

Condiciones en la configuración para la asignación de direcciones

El sistema de asignación de direcciones recién descrito posee una condición de configuración previa. Ésta es que el cableado se debe realizar de forma organizada, para que las direcciones queden ordenadas y, en los hosts, la única diferencia entre sus diferentes direcciones sea el prefijo, como se comentaba en la sección anterior. De esta forma, cuando se necesita reparar un camino, es posible deducir directamente y sobre la marcha un camino alternativo, simplemente cambiando dicho prefijo.

Si no existiera dicha configuración previa, la asignación se realizaría del mismo modo e, igualmente, los hosts poseerían cuatro variantes de HLMACs. Sin embargo, no se podría garantizar que estas direcciones variasen exclusivamente en el prefijo, sino que seguramente variarían también en otras partes de la misma. Esto haría que las direcciones dejasen de ser deducibles unas de otras con un simple cambio de prefijo. Inicialmente esto no causa ningún problema en el encaminamiento como tal, todo podría seguir siendo igual, sin embargo, a la hora de reparar un camino, no se podría deducir sobre la marcha un camino alternativo y habría que hacer un broadcast de la trama, para buscar la nueva dirección. Esto se explica un poco más en la sección dedicada a la tolerancia de fallos en Torii.

Otras posibles variantes en la asignación de direcciones

Como se puede observar, en este tipo de topologías de tres niveles más el nivel de hosts, sólo son necesarios 4 bytes a la hora de generar las direcciones HLMAC. El primero para designar el switch core, el segundo para designar el puerto del core al

switch de agregación, el tercero para el puerto del switch de agregación al switch frontera y el cuarto para el puerto del switch frontera al host.

Dado que las HLMACs poseen hasta 6 bytes, sería posible usar otro tipo de estructuras. Por ejemplo, se podría usar un menor número de bits por nivel, ya que quizás un octeto completo es demasiado para numerar todos los puentes que hay en ciertos niveles, y se podrían utilizar algunos de los bits restantes para otras opciones, como indicar si la trama se envía hacia atrás o necesita reparación. Otra posible opción sería el uso de los 2 octetos restantes para definir máquinas virtuales o múltiples hosts conectados al switch frontera mediante otro switch estándar, fuera de la topología, al estilo de PortLand. Inicialmente no se han incluido para así definir y demostrar la asignación más sencilla de direcciones en Torii, sin complicar más el concepto, pero son opciones perfectamente posibles y compatibles con el protocolo.

Envío de mensajes en Torii-HLMAC

El encaminamiento de tramas se realiza directamente mediante descodificación de direcciones destino. Una vez que las HLMACs están asignadas, a la hora de encaminar, los switches Torii necesitan distinguir dos parámetros:

- **Tipo de trama**: Si la trama es broadcast/multicast o unicast. Esto se conoce a partir del bit de la trama que así lo indica.

- **Dirección de la trama**: Si la trama va "hacia arriba" o "hacia abajo". Esto se conoce a partir del puerto de entrada y el tipo de switch. Tras el intercambio de BPDUs para la asignación de direcciones HLMAC, cada switch conoce el camino hacia el "root", es decir, conoce si un puerto está orientado hacia arriba (poseerá una HLMAC asignada) o hacia abajo (no poseerá HLMAC asignada). Además, es necesario conocer el tipo de switch para saber la dirección, pues si es un switch frontera o de agregación, la trama irá hacia abajo si viene de arriba, y viceversa, pero si es un switch core, la trama siempre irá hacia abajo.

Una vez que se conocen estos dos parámetros, la lógica aplicada en cada switch de la topología se rige por el siguiente pseudocódigo:

```
Si la trama es BROADCAST o MULTICAST:
  Si la trama va HACIA ARRIBA:
    Si el switch es frontera → traducir MAC del host a HLMAC
    Enviar trama por el puerto de salida correspondiente a la HLMAC
    Hacer "broadcast hacia abajo" de la trama
  Si la trama va HACIA ABAJO:
    Si el switch es frontera → traducir HLMAC a MAC del host
    Hacer "broadcast hacia abajo" de la trama

Si la trama es UNICAST:
  Si la trama va HACIA ARRIBA:
    Si el switch es frontera → traducir MAC del host a HLMAC
    Enviar trama por el puerto de salida correspondiente a la HLMAC
  Si la trama va HACIA ABAJO:
    Si el switch es frontera → traducir HLMAC a MAC del host
    Enviar trama por el puerto de salida correspondiente a la HLMAC
```

Tabla 14: Pseudocódigo con la lógica de encaminamiento de tramas en Torii

En el pseudocódigo “broadcast hacia abajo” significa propagación de la trama por todos los puertos que apuntan “hacia abajo”, y a la hora de traducir de la MAC física del host a una HLMAC concreta (de las múltiples que hay), se aplica una función hash que utilice como parámetros de entrada los del flujos (más parámetros implican una función algo más compleja, pero quizás más distribución de carga), por ejemplo, pero esa decisión es independiente del protocolo Torii en sí. Se puede decir que las tramas siguen dos sentidos en su recorrido, primero “hacia arriba” desde el origen y después “hacia abajo” hasta el destino, y al cambio de sentido lo denominaremos “rebote”.

A continuación se detallan más las distintas partes del pseudocódigo, así como se muestran figuras para visualizar de manera más sencilla el funcionamiento de Torii.

Broadcast y Multicast

Cuando un host A manda una trama broadcast (o multicast), ésta tendrá dirección “*hacia arriba*” (véase pseudocódigo) y llegará primeramente al switch que le sirve. Este switch escogerá un prefijo basándose en una función hash (el rango del prefijo estará entre el valor 1 y el 4 en la topología que se está usando para la definición) para asignar la nueva HLMAC a la dirección MAC de origen de A. El prefijo determina el switch core a través del que se realizará el broadcast. Las tramas broadcast desde un mismo host específico pueden usar diferentes prefijos para conseguir distribuir la carga y diversidad de caminos.

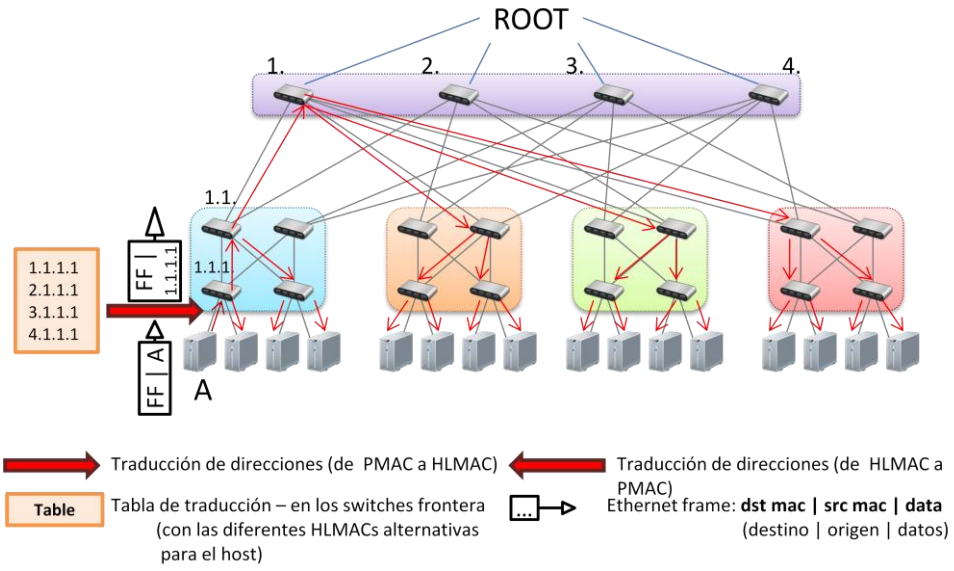


Figura 99. Trama broadcast desde el host A. La dirección destino (broadcast) se mantiene igual mientras que la dirección origen A pasa a ser 1.1.1.1. en la trama tras atravesar el switch frontera cuando el prefijo 1 se ha escogido por hash

Por ejemplo, si se escoge el prefijo 1, la dirección origen A de la trama se traducirá a la correspondiente dirección Torii-HLMAC (véase la anterior figura en la que la dirección A es cambiada por la dirección 1.1.1.1. por el switch frontera) mientras que la dirección destino (broadcast) permanecerá tal cual (FF:FF:FF:FF:FF:FF).

Una vez el prefijo ha sido escogido y la traducción de direcciones de MAC (global) a HLMAC (local) se ha hecho, la trama continúa su ruta "*hacia arriba*" siendo dirigida hacia el switch core y también replicada hacia abajo por todos los enlaces excepto por aquel asociado al puerto de entrada de la trama, tal y como se ve en la figura anterior. Para saber por qué puerto superior se tiene que reenviar la trama para que alcance el switch core asociado al prefijo, es necesario recordar que todos los puertos que apuntan hacia arriba tendrán una HLMAC asociada (pues así se van asignando según el procedimiento) y bastará con quitar el último octeto de la dirección y mirar qué puerto del switch tiene la dirección resultante asociada. Por ejemplo, en el caso de la figura anterior, la trama con dirección origen 1.1.1.1. se reenvía por el puerto asociado a 1.1.1. y más adelante por aquel asociado a 1.1. y así hasta alcanzar el switch core 1.

Cuando la trama alcanza el switch core asociado al prefijo seleccionado comienza la ruta "*hacia abajo*" (véase pseudocódigo). En este segundo tramo (después de que la trama "rebote" en el switch core al que ha llegado), lo único que hace la trama es replicarse hacia abajo por todos los enlaces, continuando el broadcast hacia abajo o "*down-broadcast*".

Cuando se alcanza cada uno de los switches frontera, estos realizan la traducción inversa de HLMAC a MAC original y transmiten la trama a los hosts que tengan conectados. Para realizar este cambio de direcciones origen en la trama, se mira la tabla de direcciones guardadas. Esta tabla se forma con los mensajes ARP que llegan a cada switch frontera, los ARP siempre se desencapsulan y se guardan tanto las MACs, como las HLMACs asociadas, así es fácil recuperar la MAC original en el caso de intercambio de tramas unicast, de manera que para el host pasa desapercibido el cambio de MAC a HLMAC y viceversa.

En el caso de tramas multicast, el mecanismo aplicado es el mismo. Como se puede ver, el envío broadcast y multicast se realiza a través de un árbol de expansión tal y como ocurre en el caso clásico de Ethernet, por lo que no hay bucles. La diferencia está en los múltiples árboles disponibles a la hora de realizar el envío (cuatro en el caso de la figura), donde la trama se reenvía sólo por uno de ellos, siendo éste escogido en base a la función hash aplicada en el primer switch que alcanza la trama, el switch frontera.

Por último, se hace necesario destacar que el uso de proxies ARP [EC09] es perfectamente factible, tanto distribuidos en los puentes frontera, como centralizados al estilo de PortLand, de manera que estos aprendan de los mensajes ARP Request y ARP Reply para reducir el número de mensajes enviados en broadcast.

Unicast

En el caso de tramas unicast, por ejemplo de un host B a uno A, ésta primero tendrá dirección "*hacia arriba*" (véase pseudocódigo) y llegará al switch que le sirve. En el switch frontera, primero se tendrá que decidir el prefijo a usar (es decir, el switch core a través del que se va a realizar la comunicación), y luego traducir B por su HLMAC con dicho prefijo (que es conocida por el switch, al ser el que le sirve) y también A por su HLMAC correspondiente, con el mismo prefijo. Ésta última se

conoce necesariamente de un mensaje ARP Request –o Reply- que se haya realizado y salvado previamente. Así pues, es condición necesaria que antes de toda comunicación unicast, exista una petición ARP; de no ser así, el switch frontera iniciará un proceso de reparación para encontrar la dirección correspondiente (véase la sección correspondiente a tolerancia de fallos en Torii).

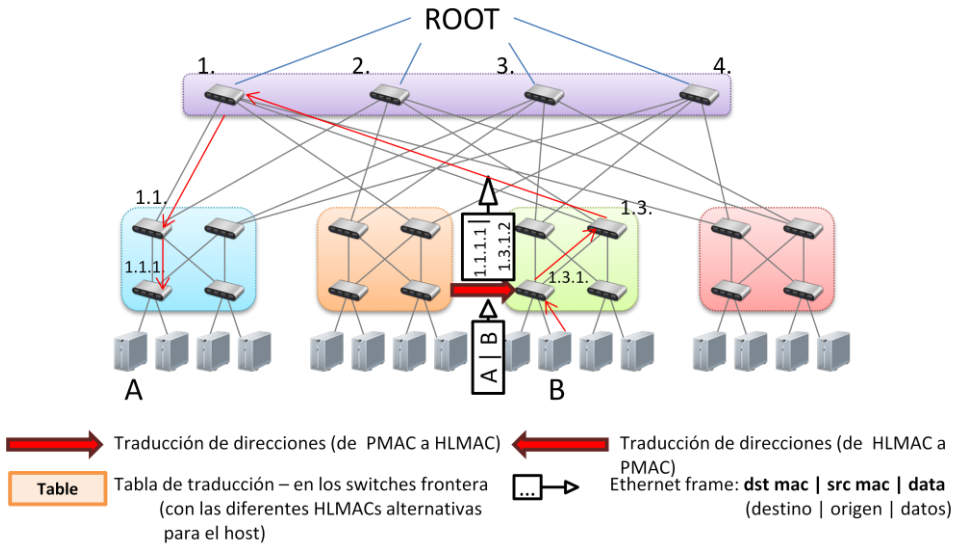


Figura 100. Trama unicast desde el host B al A. Ambas direcciones son traducidas en el switch frontera de B, A pasa a ser 1.1.1.1. y B se convierte en 1.3.1.2.

Por ejemplo, en la figura anterior podemos ver que el prefijo seleccionado ha sido 1. Por lo tanto, el switch frontera 1.3.1., que sirve a B, traducirá la dirección MAC del host B a la dirección HLMAC 1.3.1.2. (deducida directamente de la propia dirección del switch) en la trama y la MAC del A la convertirá a 1.1.1.1. (ésta última obtenida de la tabla de direcciones salvadas de mensajes ARP). Nótese que las direcciones salvadas pueden existir de cualquier ARP, aunque el ARP Request de A se hubiese realizado con otro prefijo (digamos el 3), y se hubiese guardado la dirección 3.1.1.1., sería perfectamente posible obtener la dirección 1.1.1.1., pues las direcciones son deducibles unas de otras cambiando exclusivamente el prefijo.

Una vez se ha realizado la traducción en el puente frontera, la trama unicast deberá seguir su camino "*hacia arriba*" hacia el switch core acorde al prefijo, para ello, las tramas son reenviadas por el puerto asociado al prefijo correspondiente en cada switch atravesado (véase figura anterior). Por ejemplo, en la figura anterior, la trama va al switch 1.3.1. y luego al 1.3., para finalmente llegar al 1., siguiendo el prefijo 1. A diferencia de las tramas broadcast, no es estrictamente necesario que las tramas unicast lleguen hasta el switch core en todos los casos dado que esto sólo es necesario para propagar la trama por toda la red. En el caso de unicast, el "rebote" puede comenzar antes si la dirección HLMAC de origen y destino coinciden en los bytes siguientes al prefijo, en ese casoel "rebote" se produce en el puente que tiene asignada dicha dirección. Es decir, si sólo coincide el prefijo y es 1, el "rebote" se produce en el puente core 1., pero si coinciden 1.3. (como en la figura siguiente), el "rebote" se produce en el puente 1.3., antes de llegar al core.

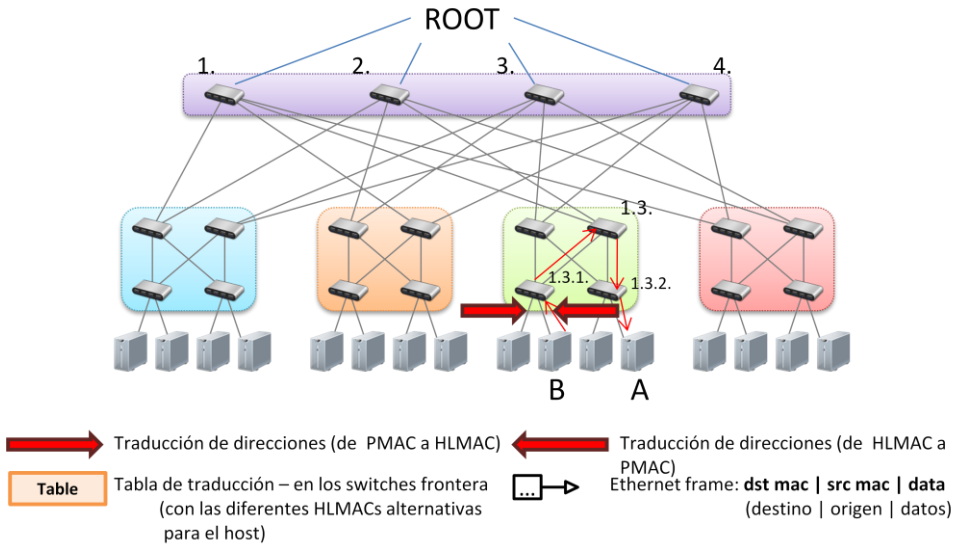


Figura 101. Envío con "rebote" anticipado de trama unicast desde B hasta A, que comparten el pod y por tanto la trama no necesita llegar hasta el puente core

Cuando la trama "rebota" y comienza su camino "*hacia abajo*", ésta ya no es reenviada en base a mirar exclusivamente el prefijo (para llegar al switch core), sino que se mira concretamente la dirección HLMAC destino y se reenvía por el puerto resultante de la siguiente operación (mostrada en la figura):

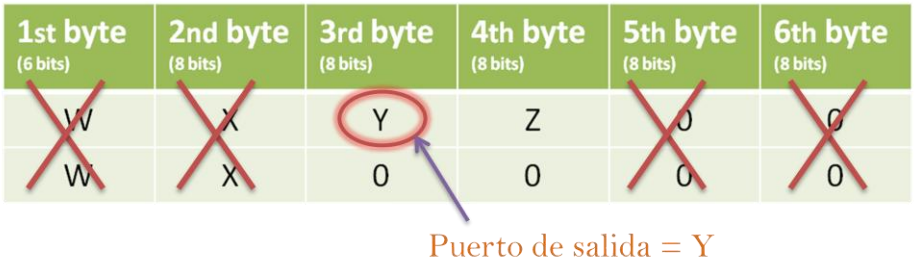


Figura 102. Puerto de salida para tramas que van "hacia abajo" en Torii. Si la HLMAC destino es W.X.Y.Z. y la del switch es W.X., la resta nos da como resultado Y.Z. y por tanto, el puerto de salida del switch W.X. por el que se reenviará la trama será el Y

Por ejemplo, en el caso de la figura 105, tras alcanzar el switch 1., éste mira la dirección de destino, que es 1.1.1.1. y así deduce que el puerto de salida por el que ha de reenvíar la trama es 1 (dado que 1.1.1.1. – 1. = 1.1.1. y el primer byte de los tres resultantes es 1). Al enviarse por el puerto 1, la trama llegará al switch 1.1. y de ahí se reenviará por el puerto 1 de nuevo para llegar al switch 1.1.1., frontera de 1.1.1.1., es decir, A (el destino de la trama).

Finalmente, cuando la trama alcanza el switch frontera de la dirección destino, se realizará la traducción inversa de las direcciones HLMAC, tanto origen como destino, de manera que la trama sea recibida por A y como la emitió B, y que todo el encaminamiento sea transparente para ambos extremos. Además, al igual que para tramas broadcast, si la trama unicast se trata de un mensaje ARP, la dirección origen HLMAC y su MAC asociada (esto se conoce desencapsulando el ARP Reply) se guardan en la tabla de direcciones guardadas. De esta forma, gracias a los mensajes

ARP, tanto ARP Request (broadcast) y ARP Reply (unicast), los switches frontera serán capaces de realizar traducciones, y sus inversas, para futuras comunicaciones unicast.

Por ejemplo, si miramos la figura que está a continuación, podemos ver cómo en el switch 1.1.1. (que es frontera de 1.1.1.1., es decir, el host A) se realiza la traducción inversa y la dirección HLMAC 1.1.1.1. pasa a ser A, y la 1.3.1.2. pasa a ser B. Además, si se tratase de un ARP Reply, la dirección B se guardaría junto con su HLMAC asociada 1.3.1.2., así como todas sus variantes de prefijo (2.3.1.2., 3.3.1.2. y 4.3.1.2.).

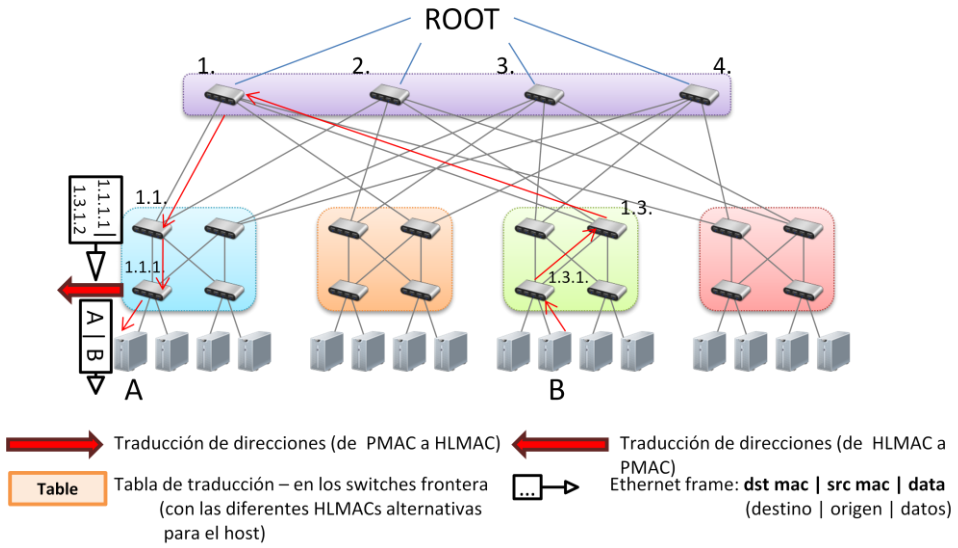


Figura 103. Trama unicast desde el host B al A. Se realiza la traducción inversa para ambas direcciones en el switch frontera de A, 1.1.1.1. vuelve a ser A y 1.3.1.2. pasa a ser B

Nótese que las tramas unicast siempre tendrán el mismo prefijo para ambas Torii-HLMACs. Esto es porque estas tramas, tanto en un sentido como en otro, siempre viajarán a través del mismo switch core de manera que la comunicación sea bidireccional. Por lo tanto, las tramas unicast en las que sus direcciones HLMAC origen y destino posean diferentes prefijos podrán usarse en eventos especiales, tales como la notificación de una acción (por ejemplo, un enlace caído cuando así se descubra) a cualquier switch en la topología. En todo caso, una variante de Torii podría no necesariamente generar tráfico bidireccional y utilizar alguno de los bits libres para indicar dichas notificaciones o acciones especiales.

5.1.2.2 Reparación de caminos

Cuando hay un cambio en la topología (ya sea por caída de un enlace o de un switch), no se realiza ningún mecanismo de intercambio de mensajes, ni se regenera el árbol, ni se reasignan en los nodos las HLMACs, que permanecerán tal cual estaban. Únicamente los switches conectados directamente al punto de fallo lo conocerán y sabrán que ese punto no será válido para ningún camino, por lo menos durante un tiempo.

Por lo tanto, cuando una trama llega a un switch de la topología y el puerto escogido de salida sea uno fallido (ya sea porque sea el escogido de salida para una trama unicast o uno de los múltiples en el caso de broadcast/multicast), es cuando comienza el procedimiento de reparación del camino. El modo de actuación que se aplica depende del tipo de trama, pero la base es siempre la misma, la deducción directa de una dirección HLMAC alternativa (con un simple cambio de prefijo), es decir, reparación sobre la marcha sin cálculos extras, en el mismo momento que la trama descubre el punto de fallo en el camino, y tras deducir el nuevo camino habrá que volver hacia atrás o, en el mejor de los casos, continuar desde donde se está, pero nunca seguir avanzando pues ya no es posible.

Broadcast y Multicast

En este caso, como no hay necesidad de seguir un camino específico (es decir, no necesitamos acordar un nuevo camino con el otro lado de la comunicación, pues ésta es unidireccional) y como lo que prima es evitar la aparición de múltiples reenvíos y duplicación de tramas al hacer el broadcast, el camino alternativo escogido será aquel que esté *más próximo* al punto de fallo. Con *más próximo*, nos referimos al camino alternativo que requiere menos pasos hacia atrás para continuar el broadcast. Nótese que cuantos menos pasos hacia atrás se den, menos probabilidades existirán de repetir mensajes en el reenvío broadcast.

Por ejemplo, en la figura que se muestra a continuación, tenemos un fallo del switch 1. al 1.3.. En este caso, no transmitimos por el resto de enlaces (aquel que va del 1. al 1.2., ni el que va del 1. al 1.4. – nótesen las cruces en los tres enlaces, el que ha fallado y estos dos últimos) y partimos a buscar la alternativa *más próxima*. Esta alternativa se encuentra transmitiendo la trama de vuelta hacia el switch 1.1. y desde ahí, tomando el camino por el switch core número 2.

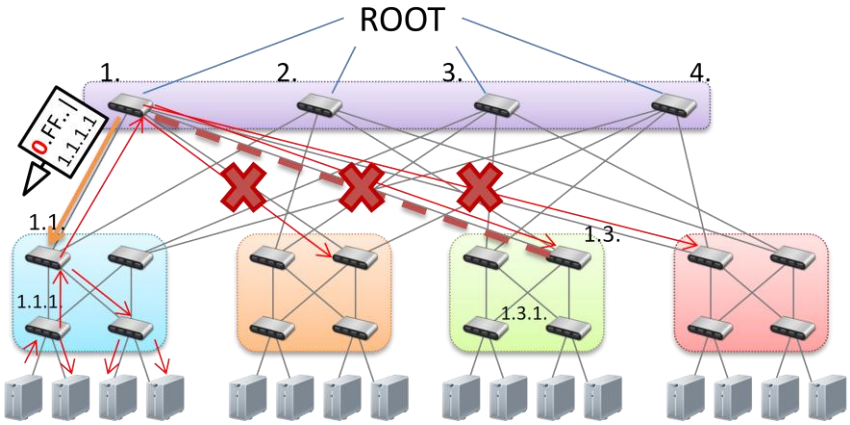


Figura 104. Fallo de enlace en broadcast en el tercer enlace de la comunicación. Paso hacia atrás de la trama

A la hora de transmitir la trama broadcast hacia atrás (hacia el switch que la mandó previamente) se decidió utilizar el primer octeto de la dirección destino y ponerlo a 0, para que fuera característico de esta situación y facilitase el procesado. Nótese que el primer octeto dentro de la topología, y debido a las características de las direcciones Torii-HLMAC, nunca podría ser 0 (básicamente porque el bit U/L estará siempre fijado a 1). Así que si el primer byte es 0, se supone directamente que

la dirección destino es FF:FF:FF:FF:FF:FF pero que ha sido modificada temporalmente por el caso de la reparación.

Para el caso de multicast no es tan sencillo como poner el primer byte a 0, dado que la dirección multicast no es fija, como en el caso del broadcast y si anulamos el primer byte poniéndolo a 0, perderemos la dirección destino de referencia. Así que algunas opciones serían utilizar los dos últimos bytes de la dirección origen para indicarlo (que inicialmente en los ejemplos básicos no se usan para nada y siempre están a 0) o incluso el propio primer byte, cambiándolo a un valor especial que no sea un prefijo válido (un valor de rango) o poniendo el bit U/L a 0, de manera que la dirección no sea local (cosa que no puede suceder dentro de la topología, puesto que las direcciones Torii son siempre locales) e indique que es una trama multicast fallida.

En ambos casos, el hecho de usar un prefijo o un byte para informar del error es totalmente opcional, para facilitar el procesado de la trama de reparación, ya que se limitaría a mirar un byte, en vez de analizar más partes (como direcciones o puertos de entrada). Es opcional porque el switch, al recibir la trama de vuelta, es capaz de saber que es una trama devuelta a partir de la dirección destino (broadcast/multicast), la origen y el puerto de entrada.

Una vez la trama ha vuelto al switch que la mandó, éste reconoce la acción y marca temporalmente el camino indicado por el prefijo de la dirección origen como inválido para broadcast. A continuación recupera la dirección destino si es broadcast (recuperando el FF en vez del 0) y cambia el prefijo de la dirección origen por su alternativa, mandando la trama por esta alternativa. Por ejemplo, en el caso de la figura, se marcaría como inválido el camino del core 1 y se escogería como alternativo el core 2.

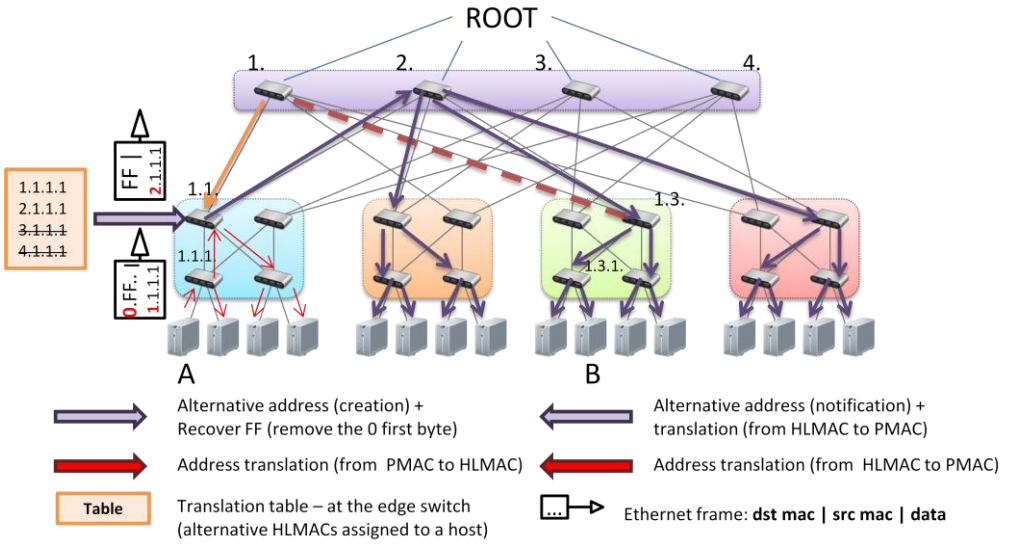


Figura 105. Fallo de enlace en broadcast en el tercer enlace de la comunicación. Elección de la alternativa y reenvío por la misma, tras cambio de prefijo en la trama

Sin embargo, hay un caso especial en el que el método anterior produciría réplicas de las tramas y que se resuelve de una manera especial. Es el caso en el que

falla el último enlace de la comunicación tal y como se muestra en las siguientes figuras.

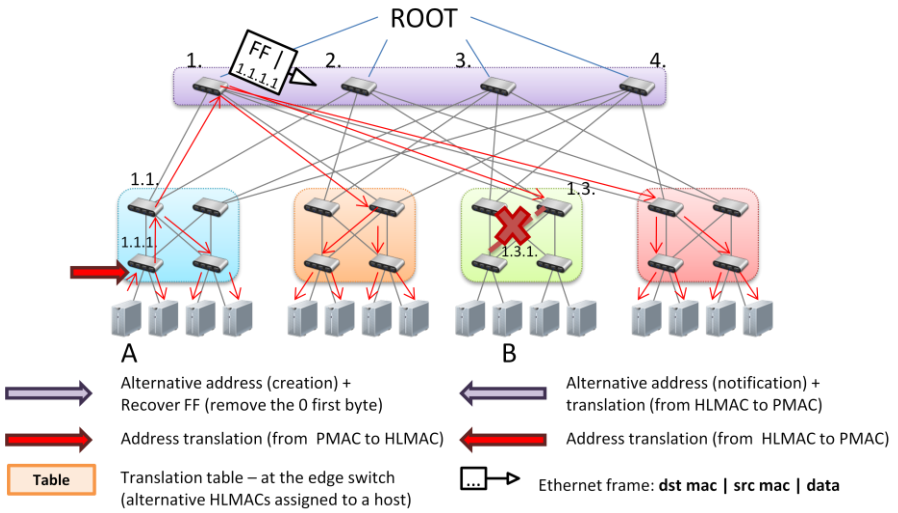


Figura 106. Fallo de enlace en broadcast en el cuarto (último) enlace de la comunicación

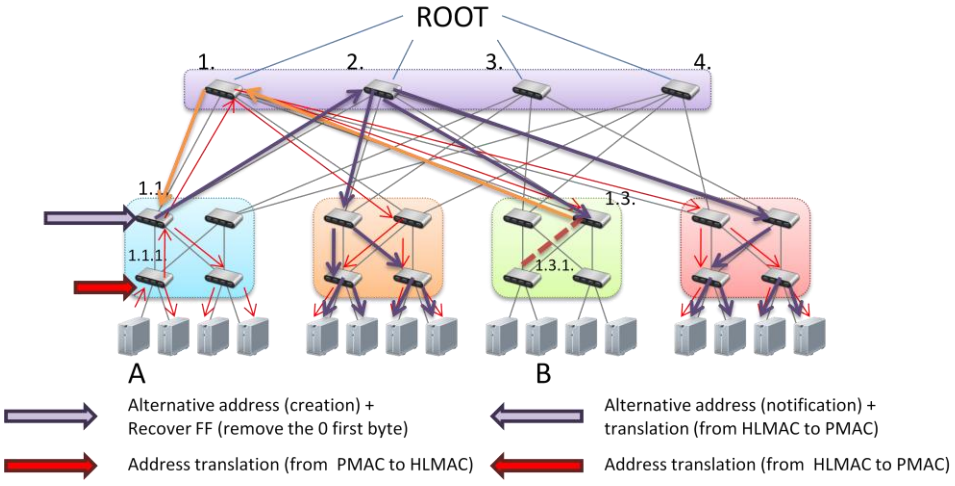


Figura 107. Fallo de enlace en broadcast en el cuarto (último) enlace de la comunicación. Tramas duplicadas (flechas moradas sobre flechas rojas) al aplicar el método general

En este caso especial, para reparar la trama no vuelve hacia atrás, sino que da una vuelta por el pod, para alcanzar a todos los hosts. Es un método en el que se siguen 3 pasos:

1. Se reenvía la trama por el enlace que sí es válido, cambiando primero el primer byte por un 0. Por ejemplo, en la siguiente figura se ve cómo la trama va del switch 1.3. al 1.3.2.

2. Tras reenviarse la trama y llegar al switch frontera, éste reenvía la trama por todos sus puertos: a sus hosts quitando el 0 y "hacia arriba" dejándolo. Se puede ver cómo la trama sube del switch 1.3.2. al 3.3. (o 4.3.) tras reenviarse hacia los hosts.

3. El switch que está arriba y recibe la trama, la reenvía "hacia abajo" por el enlace del puerto por donde no le ha llegado. La trama baja del 3.3. (o 4.3.) al 1.3.1., que finalmente la puede mandar al host B y su vecino.

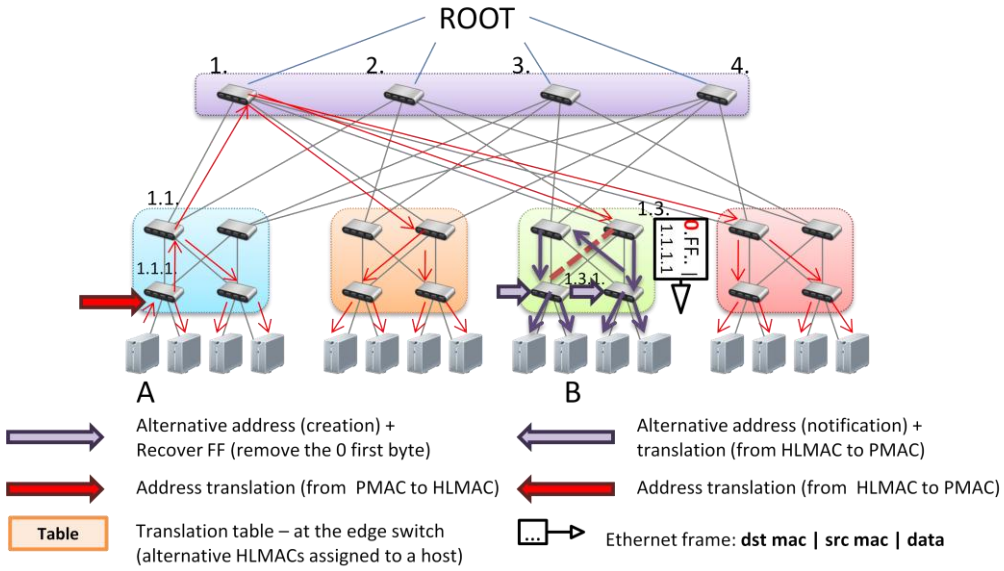


Figura 108. Fallo de enlace en broadcast en el cuarto (último) enlace de la comunicación. Tres pasos de reparación y no hay tramas duplicadas

La forma de diferenciar que no se trata de un caso del método anterior (que recordemos también se marcaba con 0) es que la trama se sabe que ya ha pasado por el switch core, es decir, está en la segunda parte del recorrido en el que la trama va "*hacia abajo*" y esto último a la vez se conoce porque el switch que descubre el fallo de enlace no es padre del origen de la trama, en la figura: el switch 1.3./2.3. es imposible que contenga un hijo origen 1.**1**.1.1. (si el switch fuese el 3.3./4.3. sucede lo mismo, el segundo byte 3 indica que es imposible ser padre de la dirección 1.1.1.1. cuyo segundo byte es 1).

De todos modos, el caso de reparación de enlaces para comunicaciones broadcast y multicast necesita un análisis más en profundidad, sobre todo en el caso de generalización de topologías, el cual trataremos a continuación.

Unicast

En el caso de tramas unicast, no existe la posibilidad de duplicar tramas a la hora de realizar el envío alternativo de reparación (como en el caso de broadcast), sin embargo, si existe una condición adicional y es que la comunicación ha de ser bidireccional, por lo que se ha de garantizar que ambos lados de la comunicación escogerán el mismo camino alternativo en caso de fallo. Para asegurar que el mismo camino se escoge en ambos lados, se ha de fijar un criterio, como por ejemplo escoger el camino del siguiente número de switch core, en orden, aún incluso cuando ésta no sea la alternativa *más próxima*. Además, es necesario que se realice una notificación a los extremos, de manera que se bloquee el camino inválido por un tiempo (con cierto temporizador) y se escoja el alternativo mientras tanto.

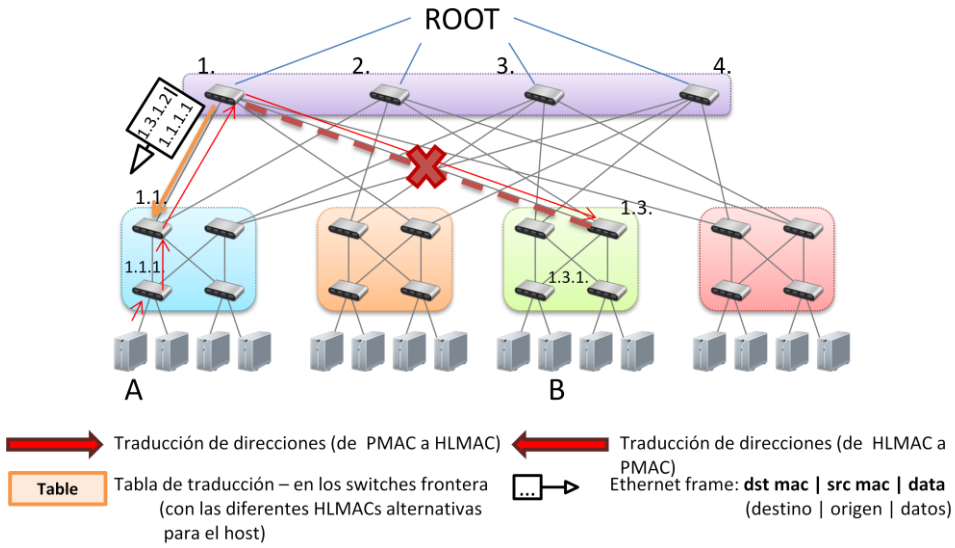


Figura 109. Fallo de enlace en unicast. Paso hacia atrás de la trama

En la última figura se puede ver como, tras descubrir la trama en el switch core 1 que el camino no es válido, la trama es reenviada hacia atrás, de vuelta al switch del que procedía, el 1.1. También se puede apreciar cómo la trama no es modificada de ninguna manera, pues no es necesario cambiar ningún byte, ni señalizar de ningún modo este paso hacia atrás, ya que si el switch 1.1. deduce el puerto de salida a partir de la dirección de destino 1.3.1.2. obtendría el puerto por el que ha entrado la trama, lo cual es un indicador de que la trama está siendo devuelta por una reparación.

Una vez que el primer switch recibe la trama devuelta, éste tiene que decidir cuál sería el camino alternativo. Si el criterio es escoger el siguiente switch core por orden y la trama iba por el 1, la decisión sería usar el 2 para el camino alternativo. Es entonces cuando el switch 1.1. descubre que tiene camino directo hacia la ruta que pasa por el switch core 2 y decide transmitir la trama por dicho camino alternativo.

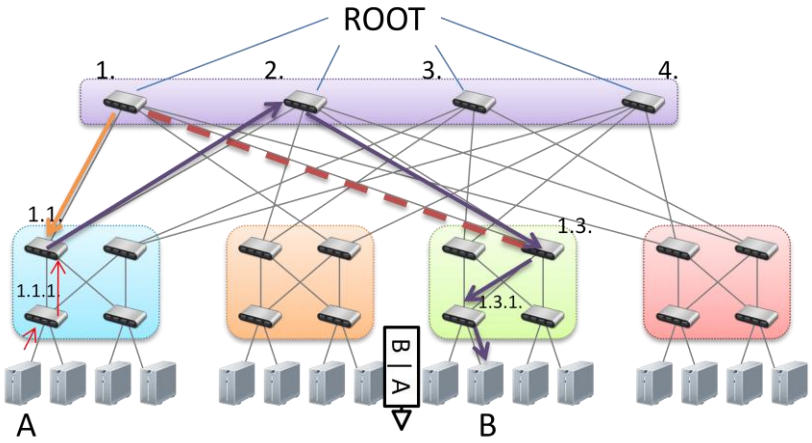


Figura 110. Fallo de enlace en unicast. La trama alcanza el camino alternativo y se reenvía al destino por este nuevo camino

Una vez se ha alcanzado el switch que da acceso al camino alternativo para la reparación, es necesario generar una notificación, para que tanto el switch frontera

del origen A, como el del destino B de la trama, dejen de utilizar el camino antiguo que ya no será válido (al menos durante un tiempo, fijado por un temporizador). Si no se emitiera la notificación, la trama sería reencaminada a su destino igualmente, pero así evitamos futuros procesamientos para caminos que se conocen fallidos.

La notificación se puede hacer de dos maneras:

- **Notificación al origen y al destino con la trama redirigida**: En este caso se modifica el prefijo de la trama de origen con el nuevo switch core utilizado y se envía hacia el origen y hacia el destino, para que sus switches frontera la intercepten (y en el caso del destino también la envíe hacia el destino) y queden notificados. Dado que el prefijo de origen y destino no coincidirán, se sabrá que es una trama especial (notificación) y es posible saber cuál va hacia el destino (que es una trama de datos y notificación a la vez) y enviarla en base a cambiar virtualmente el prefijo del destino con el que tiene el origen, y saber cuál vuelve al origen (que es sólo notificación, descartando los datos). Es decir, habrá dos mensajes de notificación por origen y si la comunicación es bidireccional (de A a B y de B a A a la vez), podrá haber hasta cuatro.

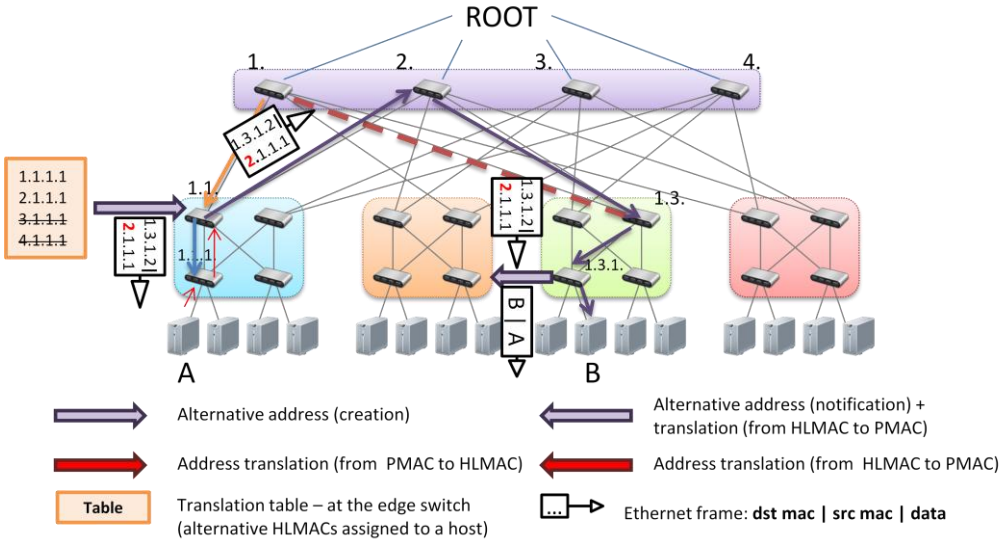


Figura 111. Fallo de enlace en unicast. Se envía trama de notificación al origen y la trama original + notificación al destino

- **Notificación sólo al origen**: Este caso es sólo la parte de notificación al origen del anterior. Es una manera de ahorrarse la notificación al destino, pero por otro lado la notificación al destino no supone un coste muy elevado, sólo un procesado ligero en los puentes intermedios, que tienen que deducir que es una trama especial y que el destino no es el marcado como tal, sino cambiando virtualmente su prefijo por el que posee el origen. En la siguiente figura se muestra un ejemplo donde se puede ver que la trama hacia el origen es igual, pero la trama hacia el destino ya no es especial, sino que cambia ambos prefijos al nuevo, 2, de manera que el nuevo origen es 2.1.1.1. y el nuevo destino 2.3.1.2. y no es necesario realizar ninguna sustitución como antes (en la figura anterior

el destino era 1.3.1.2., pero se suponía como destino 2.3.1.2. al ser una trama especial cuyo origen poseía prefijo 2).

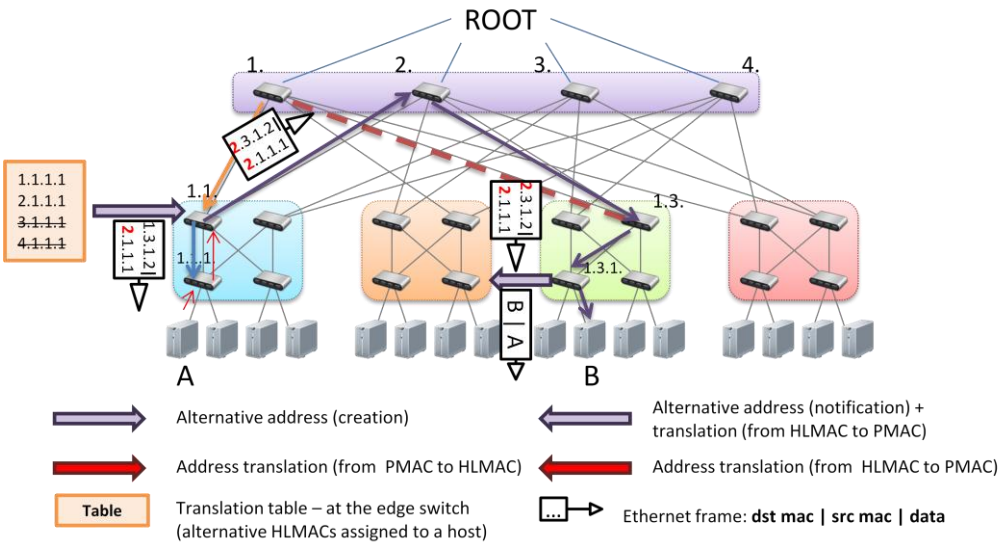


Figura 112. Fallo de enlace en unicast. Se envía trama de notificación al origen y la trama original (sin notificación) al destino

5.1.3 Implementación de Torii-HLMAC en OMNeT++

La implementación de Torii sólo se ha realizado sobre la plataforma OMNeT++ como prueba de concepto y verificación del protocolo, pues era necesario demostrar principalmente que con sólo la lógica aplicada en los puentes era posible encaminar, sin utilizar tablas de encaminamiento adicionales en ningún caso. El proyecto de Torii-HLMAC, dado que inicialmente no poseía ese nombre se denominó `HLMACinDataCenter`, con las siglas HDC.

El desarrollo del código sigue la misma base que las implementaciones previas de la familia All-Path en OMNeT, se basa en el módulo MACRelay del switch Ethernet y se modifica la función `handleAndDispatchFrame` de manera que aplique la lógica del protocolo Torii-HLMAC, en este caso. En la siguiente imagen se ve una captura del comienzo de dicha función, en la que se puede apreciar es que lo primero que se mira es la dirección de la trama (esencial a la hora de encaminar, como ya sabemos).

MACRelayUnitHDC.cc | MACRelayUnitHDC.h

```
void MACRelayUnitHDC::handleAndDispatchFrame(EtherFrame *frame, int inputport)
{
    MACAddress srcAddress = frame->getSrc(); //source address for the incoming frame (MACAdress type)
    MACAddress dstAddress = frame->getDest(); //destination address for the incoming frame (MACAdress type)

    //'frameDirection' is defined by the 'inputport' and the 'isRoot' variables
    Direction frameDirection = UPDOWNtoDOWN;
    if(portLocationTable[inputport] == DOWN and !isRoot) frameDirection = DOWNtoUP;

    EV << "->(handleAndDispatchFrame):" << endl;
```

Figura 113. Vista del módulo MACRelayUnitHDC.cc, que implementa la lógica de los switches Torii. Comienzo de la definición *handleAndDispatchFrame*

La parte que define el módulo MACRelayUnitHDC también necesita añadir algunos parámetros, dado que en OMNeT++ no se ha implementado el método RSTP extendido para propagar las HLMACs por la red. Como ya se conoce, el hecho de usar RSTP extendido es una sugerencia, pero en la práctica valdría cualquier otro que asignara las HLMACs de manera jerárquica. Por lo tanto, en OMNeT++ se introducen todos estos parámetros necesarios de manera manual, aunque deberían asignarse automáticamente por un protocolo al inicio. Los parámetros introducidos para el desarrollo de Torii son:

HLMACaddress: Cadena que define las HLMACs que posee un switch y sus puertos asociados a las mismas (recuérdese que toda HLMAC está asociada a un puerto).

PMACconnected: Cadena que define MACs de hosts conectados a un switch y los puertos por los que están conectados. Esta cadena en general será una cadena vacía, dado que las direcciones MAC se aprenderán con los mensajes ARP durante la ejecución de las simulaciones, pero se añade para simular casos concretos.

portLocation: Cadena que define los puertos de un switch y si están situados mirando "*hacia arriba*" o "*hacia abajo*" en la topología.

portHost: Cadena que define los puertos que están conectados a hosts (o switches normales) y no a otro switch Torii, es decir, puertos por los que no se habrían recibido BPDUs extendidas.

portLinkDown: Cadena que añade la posibilidad de la caída de enlaces en la simulación. Para ello se indican los puertos que se caen y en qué intervalos de tiempo (en segundos) de la simulación.

previousSetup: Booleano que indica si existe configuración previa en la topología.

realAddressLearning: Booleano que indica si los hosts aprenden las direcciones MAC reales con los mensajes ARP o las direcciones HLMAC.

Estos dos últimos parámetros (booleanos) vienen de cuatro posibilidades (dos por cada variable) que surgieron al comenzar el desarrollo de Torii en Omnet++. No se descartó ninguna y por no hacer diferentes códigos, se incluyeron como parámetros, para ver qué opciones eran las más beneficiosas. En la práctica (tras diversas simulaciones) se ha visto que las mejores opciones son que exista configuración previa (*previousSetup = true*) y que se aprendan las direcciones reales MAC y no las HLMAC de los hosts (*realAddressLearning = true*). El código está desarrollado para las cuatro posibilidades, pero sólo se ha probado y refinado para la parte en la que ambas variables son *true*, dado que es ésta la definición final de Torii por la que se optó.

A continuación se muestra una captura con estos parámetros, que se encuentran a partir del comentario `//HLMAC parameters`, y antes están los `//Original parameters`, que son los parámetros de la familia de switches All-

Path y muchos de ellos no se utilizan, como son aquellos referentes a la tabla de direcciones (`addressTable`…).

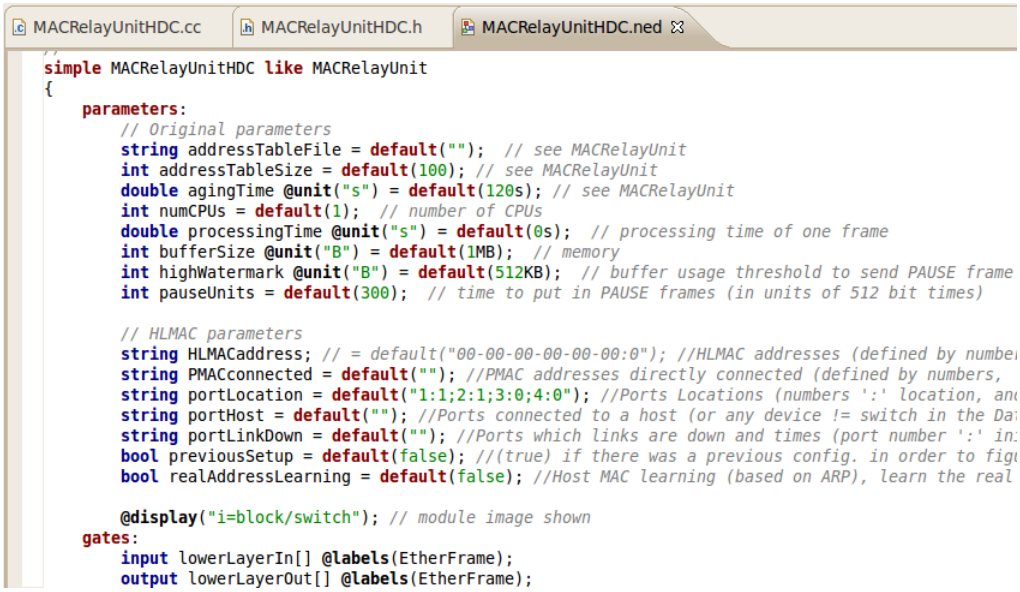


Figura 114. Vista del módulo MACRelayUnitHDC.ned, que define el módulo MACRelay que irá dentro del switch Torii, con sus parámetros y puertas de interconexión

Torii establece otras particularidades. Ahora los switches son *commodity* y fijamos el número de puertos a un máximo, que en el caso concreto de OMNeT++ serán cinco, porque el puerto 0 se ignora (no asigna direcciones) y son los otros cuatro los que darán los valores de las HLMACs. A continuación se muestra el módulo del switch Torii (`EtherSwitchHDC.ned`) que a su vez contiene el módulo de la capa de transmisión *relay* (`MACRelayUnitHDC.ned`), donde se señala el trozo de línea que fija el número de puertos a cinco.

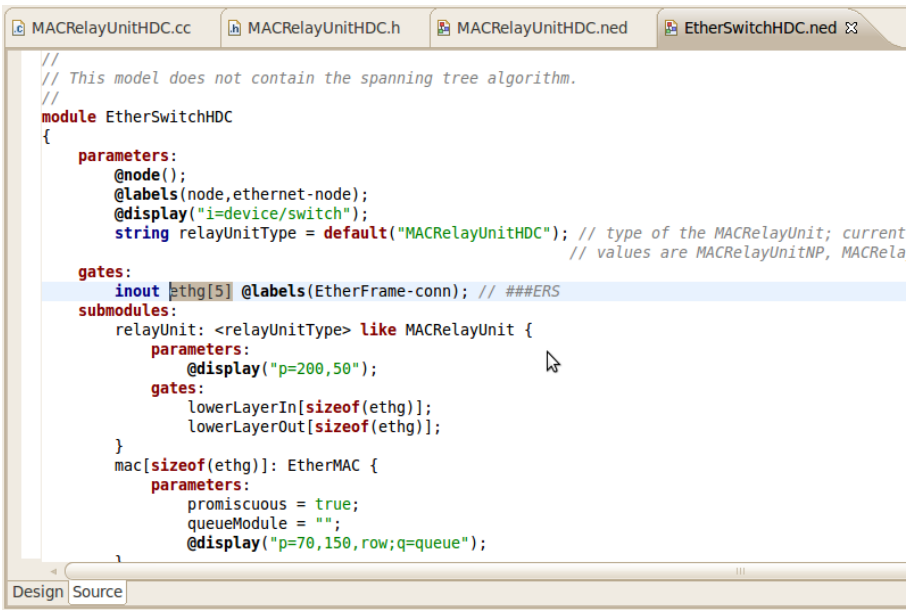


Figura 115. Vista del fichero del nodo EtherSwitchHDC.ned, que define el switch Torii, con el número de puertos fijado a cinco (*ethg[5]*)

A la hora de simular el protocolo Torii-HLMAC ya no es posible utilizar cualquier topología, sino sólo aquellas empleadas en centros de datos y específicas para el protocolo. En el caso de Torii, la topología que se utilizó como prueba de concepto fue la de PortLand, para la que se generó un fichero `dataCenterPortland.ned` compuesto de switches Torii `EtherSwitchHDC.ned` y hosts comunes, tal y como se muestra en la siguiente imagen, que muestra la visión de diseño del módulo de la topología.

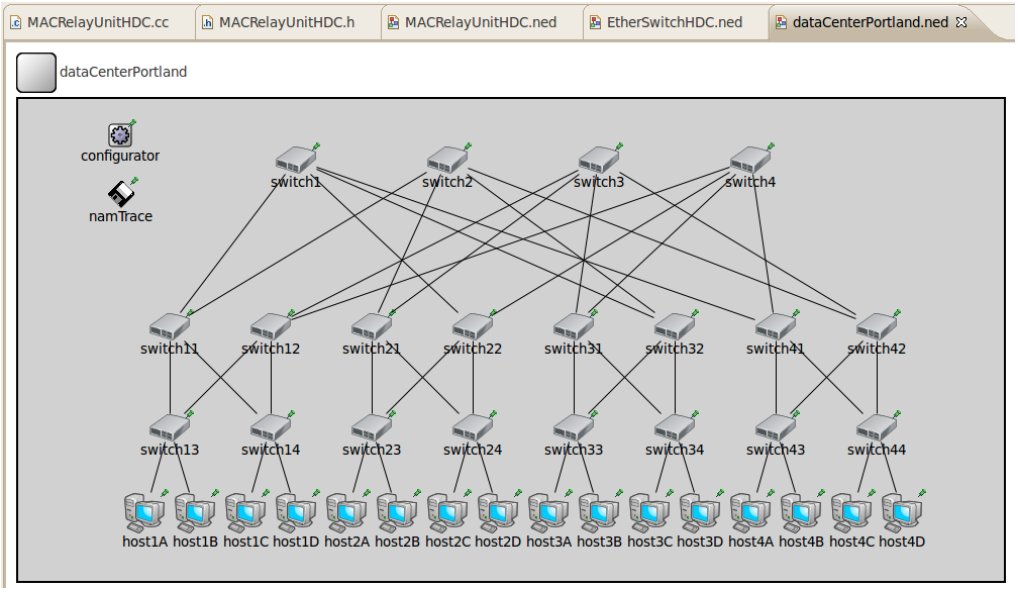


Figura 116. Vista del fichero de la topología de pruebas dataCenterPortland.ned

Además, al ser las conexiones controladas (*previousSetup = true*) para que se forme una jerarquía ordenada de HLMACs, esto se tiene que indicar en la visión de código del módulo de la topología.

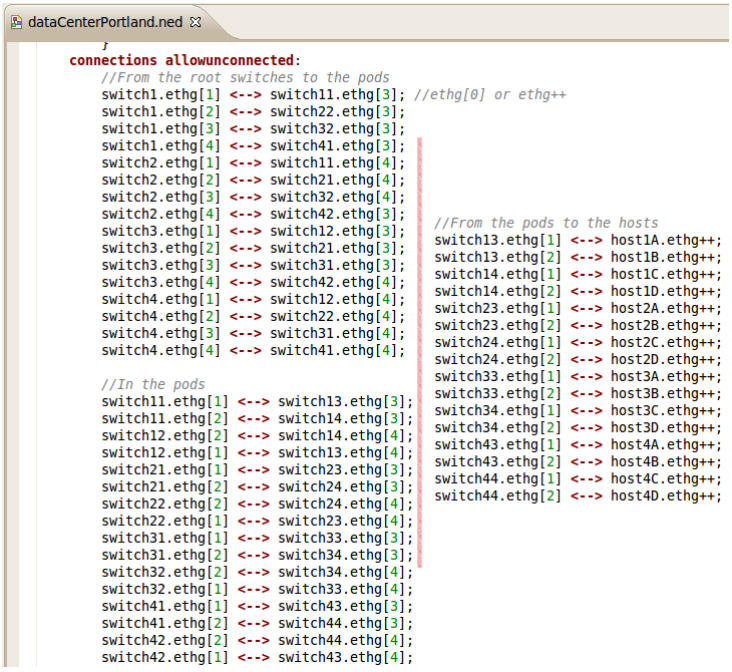


Figura 117. Vista de las conexiones entre switches y hosts del fichero de la topología de pruebas dataCenterPortland.ned

Finalmente, en el fichero de configuración de la simulación `dataCenterPortland.ini` lo primero que haremos es poner a `true` los booleanos de los parámetros *previousSetup* y *realAddressLearning*, dado que es esta versión de Torii por la que finalmente se optó. A continuación definiremos las cadenas *HLMACaddress*, *portLocation* y *portHost* para cada nodo que así lo necesite. Es necesario recordar que estos parámetros se obtendrían directamente de haberse implementado RSTP extendido en el proyecto, pero al estar sólo implementada la lógica de Torii, se deben introducir manualmente.

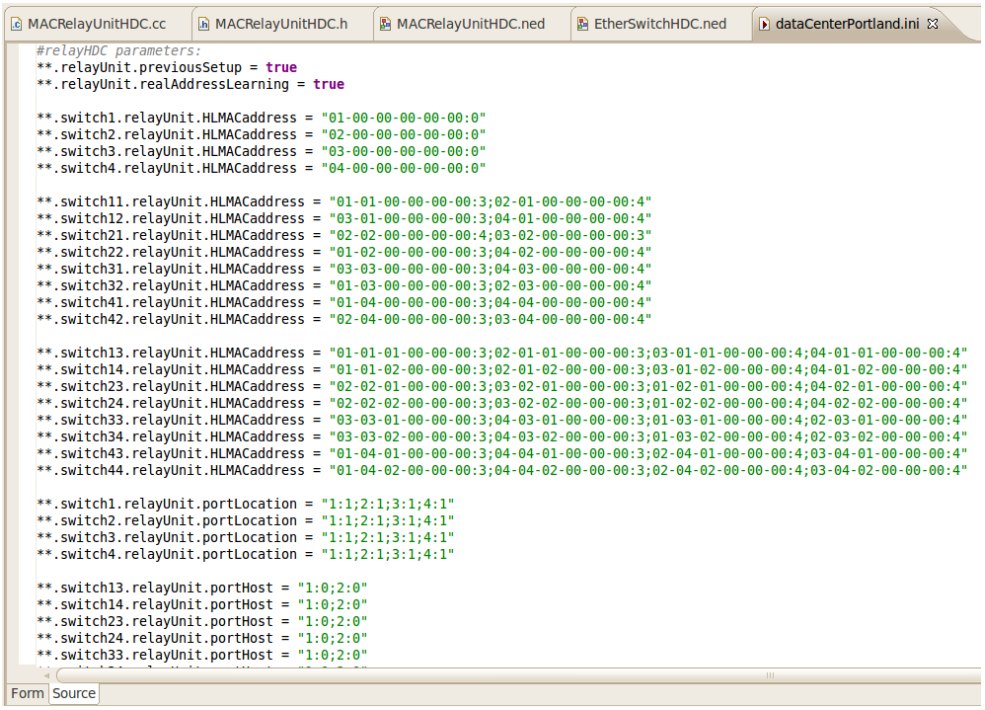


Figura 118. Vista de los parámetros Torii-HLMAC del fichero de configuración de pruebas dataCenterPortland.ini

En este mismo fichero definimos los diferentes casos de envío en Torii (unicast y broadcast), que se demostraron con tráfico UDP y los mensajes ARP que se envían siempre previa toda comunicación, y también se definieron todos los posibles escenarios de fallo de un enlace en la topología generada. En la siguiente captura se puede ver que la primera parte define 5 casos (los 5 enlaces que tiene que atravesar el paquete desde que se genera en el *host1A*) de fallo de enlace para el envío de un mensaje broadcast, concretamente un ARP Request, mientras que en la segunda parte se definen otros 5 de fallo de enlace para el envío de un mensaje unicast, concretamente un mensaje UDP. Nótese que para el caso de broadcast el fallo de enlace comienza en el segundo 0, antes de emitir el primer ARP, mientras que para unicast comienza en el segundo 20, cuando ya se habrá emitido el ARP y existirá comunicación unicast entre hosts. Nótese también cómo cada fallo de enlace se define dos veces, una por cada puerto de cada switch extremo del enlace.

```
### ARP BCAST+UCAST case 1
#**.switch13.relayUnit.portLinkDown = "3:0-3000"
#**.switch11.relayUnit.portLinkDown = "1:0-3000"
### ARP BCAST+UCAST case 2
#**.switch11.relayUnit.portLinkDown = "3:0-3000"
#**.switch1.relayUnit.portLinkDown = "1:0-3000"
### ARP BCAST+UCAST case 3
#**.switch1.relayUnit.portLinkDown = "4:0-3000"
#**.switch41.relayUnit.portLinkDown = "3:0-3000"
### ARP BCAST+UCAST case 4
#**.switch41.relayUnit.portLinkDown = "1:0-3000"
#**.switch43.relayUnit.portLinkDown = "3:0-3000"
### ARP BCAST case 5 + Recover at second 1
#**.switch43.relayUnit.portLinkDown = "2:0-1"

### UDP UCAST case 1
#**.switch13.relayUnit.portLinkDown = "3:20-3000"
#**.switch11.relayUnit.portLinkDown = "1:20-3000"
### UDP UCAST case 2
#**.switch11.relayUnit.portLinkDown = "3:20-3000"
#**.switch1.relayUnit.portLinkDown = "1:20-3000"
### UDP UCAST case 3
#**.switch1.relayUnit.portLinkDown = "4:20-3000"
#**.switch41.relayUnit.portLinkDown = "3:20-3000"
### UDP UCAST case 4
#**.switch41.relayUnit.portLinkDown = "1:20-3000"
#**.switch43.relayUnit.portLinkDown = "3:20-3000"
### UDP UCAST + Recover at second 60
#**.switch43.relayUnit.portLinkDown = "2:20-30"
```

Figura 119. Vista de las pruebas para los casos posibles de fallo de enlace (broadcast y unicast) del fichero de configuración de pruebas dataCenterPortland.ini

5.1.4 Generalización de Torii-HLMAC a otras topologías

En los anteriores apartados se ha descrito Torii HLMAC con la topología base utilizada en el documento de PortLand. Esta topología se basa en el uso de switches de bajo coste (y por tanto prestaciones), los llamados *commodity switches*. Además, todos los enlaces que interconectan estos puentes poseen la misma capacidad.

Como ya sabemos, esta topología se denomina “fat tree” en el documento de PortLand, pero en realidad se trata de una red Clos y se puede generalizar a tamaños mayores [Vah+10], en los que también se puede aplicar Torii-HLMAC . La diferencia entre un auténtico *fat tree* y una red Clos es que la primera topología va incrementando la capacidad de sus enlaces según nos vamos acercando a los switches core, mientras que en la segunda todos los enlaces son iguales. Los fat tress son también muy convenientes para redes Ethernet gracias a la compatibilidad de agregación de la evolución de la familia Ethernet (usando enlaces de 100Mbps, 1Gbps, 10Gbps, etc). Por lo tanto, en la práctica, el uso de un tipo de topología u otro dependerá de la propiedad que se desee más: menor coste por el hecho de usar componentes más baratos (red Clos) o menor complejidad a la hora de realizar las interconexiones, es decir, menos enlaces implicados en la construcción de la topología (*fat tree*).

En el caso de Torii, ambos tipos de topología podrían ser usados. De hecho, los *fat trees* son especialmente adecuados para ahorrar en complejidad de interconexión una vez que hay un mínimo de cuatro switches core, dado que cuatro son suficientes para aportar a la red suficiente variedad de caminos (cuatro posibilidades). La única condición para usar Torii en las diferentes topologías es que los nodos estén organizados en "*pods*", es decir en grupos, de manera que desde el puente del nivel superior de la jerarquía hasta el puente frontera los enlaces formen un árbol. Esta condición es necesaria sólo para que la propagación broadcast de tramas en la topología no cause bucles, pues el resto del funcionamiento de Torii no tiene mayor inconveniente en el uso de otras topologías aunque no tenga grupos de "*pods*".

En la siguiente figura se muestran dos tipos diferentes de topologías de cuatro niveles de switches (en lugar de los tres niveles que mostrábamos en los anteriores apartados) en las que se podría aplicar el protocolo Torii-HLMAC. La topología de la izquierda es una red Clos, que necesita un cableado más complicado, pero permite el uso de switches *commodity* de 4 puertos y cada host podría obtener hasta 8 direcciones Torii-HLMAC. En la derecha está la red *fat tree* equivalente, que necesita un cableado menor y la capacidad de los enlaces aumenta según subimos en la jerarquía (enlaces más gruesos), pero no permitiría el uso de switches *commodity* y cada host sólo podría tener hasta 2 direcciones Torii-HLMAC, lo que quizás podría ser suficiente para ciertas arquitecturas. La elección de una u otra siempre dependerá de los requerimientos de diseño.

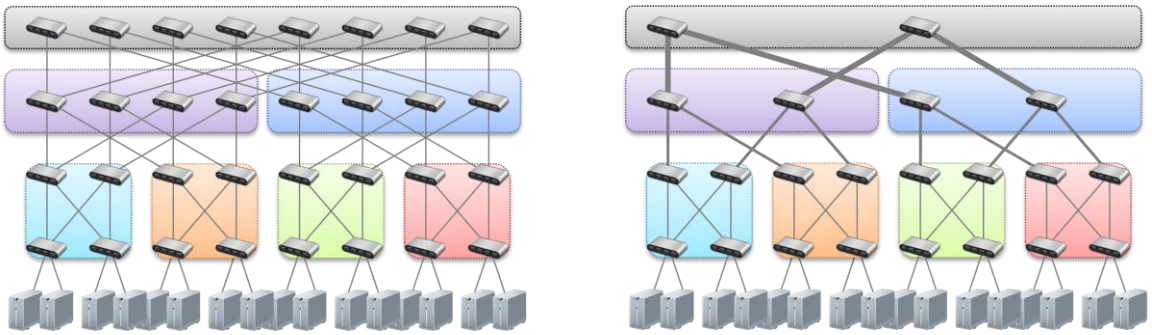


Figura 120. Red Clos (izquierda) y la red *fat tree* equivalente (derecha) de 4 niveles de jerarquía y 8 puentes frontera, con "*pods*" marcados como grupos de colores

Por último, una posible generalización mayor de topologías, que no tengan que agrupar sus switches en "*pods*" sería posible añadiendo la filosofía del 'lock' en pequeñas tablas en los switches. Es decir, los "*pods*" son necesarios para evitar los bucles la propagación de las tramas broadcast, entonces si utilizamos una tabla análoga a la BT (*Broadcast Table*) de la familia All-Path en los switches Torii ya no habría bucles y se podría usar, por ejemplo, cualquier tipo de *fat tree* con las mismas ventajas de Torii: encaminamiento sin tablas (sólo habría tablas de pequeña duración 'lock' para las tramas broadcast), multicaminos, reparación sobre la marcha, etc. Además, para que estas tablas de bloqueo fueran más pequeñas, se podría bloquear por dirección de puente frontera en lugar de por dirección de host (nótese que la dirección del puente frontera está contenida siempre en la de host).

5.1.5 Evaluación de Torii-HLMAC frente a la familia All-Path

Torii-HLMAC mejora PortLand de diferentes maneras: las direcciones múltiples se asignan de manera automática y distribuida, sin necesidad de un gestor centralizado, ni distribuido (como sería el caso de VL2), además no necesita tablas de encaminamiento en los switches y la reparación de caminos se realiza sobre la marcha.

En el caso de la comparativa directa con la familia All-Path, esta última genérica para cualquier topología, se podrían matizar los siguientes puntos:

- **Tamaño de tablas:** Torii-HLMAC no posee tablas de encaminamiento, sólo tablas en los switches frontera para guardar las traducciones (aprendidas del ARP) y puntualmente, si se generaliza para más topologías como se exponía en el apartado anterior, tablas de bloqueo para evitar bucles en la propagación de tramas broadcast. Así que el tamaño de las tablas en Torii-HLMAC es siempre menor que otros protocolos de la familia All-Path.

- **Distribución de carga:** En el caso de la distribución de carga, sería necesario realizar una comparativa directa con la herramienta de simulación OMNeT++. Actualmente la distribución es mayor para las variantes de ARP-Path multipath y Flow-Path dentro de la familia All-Path, y Torii-HLMAC posee multicaminos por definición, así que su distribución de carga debería compararse con estas variantes. Sin embargo, la diferencia radica en el hecho de que la selección de caminos en All-Path se hace por latencias y estado en cada momento de la red, mientras que en Torii-HLMAC inicialmente se decide por una función hash con parámetros del flujo o la comunicación.

- **Simetría de caminos:** La familia All-Path tiene variantes que garantizan la simetría y otras que no. En el caso de Torii-HLMAC, los caminos se construyen simétricos para que la asimetría (diferente valor del byte prefijo en el HLMAC) sea signo de trama especial de reparación, pero en la práctica se podría usar un bit al margen del byte y entonces los caminos no tendrían por qué ser simétricos.

- **Tipo de refresco:** La familia All-Path tiene un refresco de las entradas de la tabla de encaminamiento “hacia atrás” en Flow-Path y “hacia adelante” en el resto. En Torii-HLMAC no existe refresco al no haber tablas de encaminamiento, el único refresco sería en la tabla de ‘lock’ y análogo a All-Path (de haberla) y en las tablas de los puentes frontera que refrescaría el aprendizaje con nuevos mensajes ARP.

- **Número de búsquedas en tabla:** Una vez más, Torii-HLMAC no requiere búsquedas en tablas a la hora de encaminar frente al caso de la familia All-Path.

- **Reparación:** Torii-HLMAC tiene reparación automática sobre la marcha, frente a la familia All-Path que necesita repetir el proceso de los mensajes ARP para reconstruir cierto camino. Además, los mensajes de notificación para los puentes frontera en Torii-HLMAC no tienen por qué ser mensajes con un encapsulado especial, pues las propias direcciones HLMAC pueden usarse para indicar que son notificaciones, ya sea haciendo que los prefijos sean diferentes o con un bit específico para ello.

- **Aprendizaje con ARP:** Éste es un punto común entre ambas familias, tanto All-Path como Torii-HLMAC utilizan la propagación del ARP para aprender algo. En el caso de All-Path se generan todas las tablas de encaminamiento, mientas que en Torii-HLMAC se aprenden las HLMACs de los vecinos en los puentes frontera, para poder realizar posteriormente la traducción de MAC física a HLMAC correspondiente.

- **Tipo de topologías:** La familia All-Path es válida para cualquier topología, mientras que Torii-HLMAC es específico de topologías de centros de datos, como es el *fat tree*.

- **Configuración previa:** La familia All-Path no requiere en ningún momento configuración previa, Torii-HLMAC tampoco, pero sí necesita un cableado correcto de la topología, para que la jerarquía sea ordenada.

Capítulo 6

Conclusiones y trabajo futuro

6.1 Conclusiones

En la presente Tesis se presentan una serie de contribuciones en arquitecturas de redes de conmutadores transparentes Ethernet de altas prestaciones, primero de manera genérica para redes empresariales y a continuación en el caso concreto, enmarcado dentro del anterior, de redes de centros de datos. Estas contribuciones dan como resultado la familia All-Path y el protocolo Torii-HLMAC, compitiendo de manera directa con los enfoques de los recientes estándares del IEEE, SPB, y del IETF, TRILL RBridges ambos basados en encaminamiento de estado de enlaces.

All-Path es una familia de protocolos para conmutadores transparentes en la que todos tienen en común el hecho de generar rutas entre pares de dispositivos en base a la inspección del paquete ARP generado previamente a toda comunicación. Estos caminos se eligen con la primera copia del ARP Request que alcanza el destino, por lo que son caminos mínimos, de baja latencia, tomando como base el estado puntual de la red. Se trata sin duda de un nuevo concepto de puentes transparentes

basado en la exploración de todos los caminos y la selección de los mismos por su menor latencia, mediante aprendizaje de direcciones en capa dos. Se trataba de un área no explorada hasta la fecha en redes conmutadas dado que el paradigma dominante está basado en la hibridación de puentes y enrutadores. Como se ha indicado en el capítulo de estado del arte, este paradigma (encaminamiento mediante protocolo estado de enlaces en capa dos) es adoptado para los Rbridges de salida mientras que para SPB se adopta tras intentar aproximaciones basadas en árboles múltiples de expansión que luego son abandonadas, entre otras razones por la problematica de obtener simetría en el aprendizaje de direcciones con VLANs. Al respecto además del concepto de VLANs, una de las indagaciones de la Tesis es que, aunque las etiquetas VLAN eran esenciales en MSTP para poder establecer árboles de expansión múltiples (y con ello posibilitar el uso de todos los enlaces e incluso obtener caminos mínimos), éstas no son imprescindibles (salvo para su función natural de separación de tráfico y virtualización de redes) si se dispone de mecanismos de prevención de bucles alternativos como es el caso de la familia All-Path. Esta independencia del uso de VLANs es otra característica diferenciadora frente a TRILL y SPB. Finalmente, con respecto al encapsulado MAC-in-MAC, con aprendizaje de direcciones de puentes en vez de direcciones de hosts, es un mecanismo de escalabilidad para redes Ethernet igualmente aplicable en SPB como en All-Path.

Dentro de esta familia, el protocolo base es ARP-Path, que comenzó denominándose FastPath y evolucionando en la versión ARP-Path unidireccional alternativo o retardado, explicando así los porqués de dicha evolución, explorando ARP-Path en anchura y profundidad. Otra variante es Flow-Path, que sigue la misma filosofía de ARP-Path pero crea los caminos en base a flujos (pares de hosts), en lugar de hosts individuales. También se añaden a la familia ARP-Path* y ARP-Path multipath. De todos estos protocolos, aunque principalmente de ARP-Path y Flow-Path, se realiza una comparativa de sus diferencias que principalmente es un mejor reparto de carga a cambio de tamaños de tablas mayores, y se analizan los escenarios en los que un versión u otra puede ser más ventajosa. Además ARP-Path se ha implementado en diversas plataformas, cuyos resultados muestran las ventajas características de la familia All-Path como son los caminos de baja latencia (comparando directamente con SPB) o el excelente reparto de carga que efectúa el protocolo de manera nativa.

En el caso de Torii-HLMAC se presenta un protocolo específico para arquitecturas de redes de centros de datos, inicialmente basado en la red de la propuesta conocida como PortLand, que luego es generalizada a más tipos de redes, incluyendo los denominados *fat tree* de manera genérica, los cuales constituyen la topología de referencia en centros de datos. Es decir, en un entorno específico de redes de centros de datos para topologías predefinidas, Torii-HLMAC mejora y simplifica las arquitecturas de PortLand y *fat trees* en general, aportando, a cambio de un cableado riguroso, caminos múltiples de forma nativa sin necesidad de tablas de encaminamiento y reparación sin pérdidas de tramas ni retardo significativo, reenviándose la trama automáticamente por un camino alternativo en cuanto se descubre el fallo de enlace sin enviar mensajes adicionales. La necesidad de un cableado planificado a la hora de construir la red es una desventaja, pero que no supone un añadido grande puestos que las redes de centros de datos siempre

requieren cierta planificación y, a cambio, tenemos todas las ventajas mencionadas. Torii-HLMAC tiene en común con la familia All-Path el uso de los mensajes ARP, en este caso para propagar las direcciones físicas reales de los hosts y guardarlas en los puentes frontera, a fin de que estos puedan traducir de MAC física a HLMAC, y viceversa, en los siguientes intercambios de mensajes de la comunicación. Se presenta también la implementación de Torii-HLMAC en OMNeT++.

6.2 Trabajo futuro

Son diversas las líneas de trabajo que se ven más relevantes y prioritarias en las diferentes contribuciones y se muestran a continuación:

- **Respecto a la familia All-Path:**
    - Posible exploración y evolución del protocolo Flow-Path hacia una versión similar a ARP-Path, en la que no sea necesario el indicador (flag) de reparación y no haya confirmación de estados, sino que evolucionen independientemente del ARP Reply como respuesta al ARP Request.
    - Exploración de alternativas de ARP-Path multipath, no basadas en VLAN o con multicaminos más controlados.
    - Optimización del tráfico multicast en ARP-Path (y en general en toda la familia All-Path) para que no se propaguen los mensajes multicast por toda la red, como broadcast, sino sólo a los hosts que se han unido al grupo de comunicación.
    - Implementación y verificación de la última versión del protocolo ARP-Path, ARP-Path unidireccional alternativo/retardado, en OMNeT++.
    - Implementación y verificación de las versiones ARP-Path* y ARP-Path multipath en OMNeT++, para realizar comparativas de distribución de carga y tamaños de tablas con ARP-Path después.
    - Implementación de un generador de flujos basado en TCP (en lugar de UDP, que es el actual) y repetición de simulaciones de reparto de carga para verificar las bondades de ARP-Path en un escenario en el que el ARP Request va en sentido contrario al sentido del tráfico (en UDP el ARP Request va en el mismo sentido del tráfico).
    - Realización de más simulaciones de ARP-Path versus Flow-Path, con la última variante de ARP-Path, para caracterizar de forma precisa sus ventajas en reparto de carga.

- Realización de más simulaciones en diferentes topologías y escenarios de comparativa de latencias ARP-Path versus SPB.

- Exploración del uso de la familia All-Path para el grupo de estandarización de *Audio Video Bridges* en IEEE 802.1, donde el mecanismo de descubrimiento de múltiples caminos es muy adecuado para crear caminos alternativos (combinado evidentemente con mecanismos de anuncio de ancho de banda y reserva de capacidad en los switches).

- Unificación de redes inalámbricas y cableadas bajo un paradigma común, siguiendo el esquema de *BSS Bridging* [Kle12], en el que se integren los puntos de de acceso en una arquitectura de puente que permita aplicar directamente los protocolos de capa dos para redes cableadas a redes conmutadas que contengan también redes inalámbricas IEEE 802.11.

- **Respecto a Torii-HLMAC:**

  - Implementación y verificación en OMNeT++ del protocolo generalizado (actualmente sólo probado en la topología de PortLand).

  - Simulación avanzada de Torii-HLMAC en OMNeT++ para realizar comparativas directas con la familia All-Path en cuanto a latencias y reparto de carga.

  - Mejora de la propagación multicast en Torii-HLMAC para que no se propaguen los mensajes multicast por toda la red, como broadcast, sino sólo a los hosts que se han unido al grupo de comunicación.

Definiciones

Algoritmo de Caminos Mínimos: Véase *Dijkstra.*

Backbone: Conexión troncal principal de Internet o de una red en general. Actualmente está compuesta de un gran número de routers comerciales, gubernamentales, universitarios y otros de gran capacidad interconectados que llevan los datos a través de países, continentes y océanos del mundo mediante cables de fibra óptica

Bridge Designado: El término bridge designado se emplea en el mismo sentido que en el estándar 802.1D. Es el bridge que encamina el tráfico de una LAN determinada. El bridge designado de un segmento LAN es el bridge designado para todos los sistemas finales de esa LAN.

Centro de Procesamiento de Datos (CPD): También denominados Centro de Proceso de Datos, Centro de Cómputo, Centro de Cálculo o Centro de Datos por el equivalente del inglés *Data Center*, se trata de aquellas localizaciones que concentran todos los recursos necesarios para el procesamiento de la información de una organización, compuestos generalmente de sistemas de almacenamiento y de telecomunicaciones.

Common and Internal Spanning Tree: Árbol de expansión calculado por STP, RSTP más la conectividad a través de las regiones y bridges MSTP calculada por MSTP para asegurar la interconexión simple y completa.

Commodity: Se dice que cierto dispositivo es *commodity* cuando se trata de un dispositivo que cumple la función básica que se le pide y es de bajo coste. En el caso general, el término que podría usarse es el de "genérico", así como en el caso de las farmacéuticas existen medicamentos de marca y genéricos. Es un término que se aplica también a servicios.

En el caso concreto de este documento se utiliza el término de ***commodity switches*** para referirse al tipo de switches genérico, de bajo coste y no de una marca, capacidad, potencia o calidad específica. El uso de estos switches es una tendencia reciente en el marco de redes en centros de datos, pues es más sencillo aumentar el número de estos dispositivos que utilizar un número menor de switches de tecnología puntera, para ofrecer al final la misma calidad de servicio en la red.

Common Spanning Tree: Árbol de expansión calculado por STP, RSTP, MSTP para conectar regiones MST.

Data Center: Véase *Centro de Procesamiento de Datos*.

Designated Bridge: Véase *Bridge Designado*.

Dirección MAC: Identificador de 48 bits (6 bloques hexadecimales) que corresponde de forma única a una tarjeta o dispositivo de red.

Dijkstra (algoritmo): También llamado *Algoritmo de Caminos Mínimos*, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger W. Dijkstra, quien lo describió por primera vez en 1959.

Edge Bridge/Switch: Véase *Puente Frontera*.

Fiber Distributed Data Interface: Conjunto de estándares ISO y ANSI para la transmisión de datos en redes de computadoras de área extendida o local (LAN) mediante cable de fibra óptica. Se basa en la arquitectura token ring y permite una comunicación tipo full dúplex. Dado que puede abastecer a miles de usuarios, una LAN FDDI suele ser empleada como backbone para una red de área amplia (WAN).

Internal Spanning Tree: Árbol Interno de Expansión de una Región MST. Parte del CIST incluida dentro de una región MST.

Jerarquía Digital Síncrona: Protocolo estandarizado de multiplexación que transmite múltiple información digital sobre fibra óptica mediante el uso de láseres o diodos LED. Se desarrolló en EE.UU. bajo el nombre de SONET o ANSI T1X1 y posteriormente el CCITT (hoy UIT-T) en 1989 publicó una serie de recomendaciones donde quedaba definida con el nombre de SDH.

Legacy: Se dice que cierto sistema es *legacy* cuando se trata de un método, tecnología, aplicación o programa antiguos, porque nuevos sistemas (normalmente como evolución) han aparecido, eclipsando a los primeros. Un sistema *legacy* puede o no seguir usando, en todo caso impacta al sistema actual y a veces se requiere compatibilidad "hacia atrás" por si se sigue utilizando.

Link State Protcol: Véase *Protocolo de Estado de Enlace.*

Local Area Network: Véase *Red de Área Local*.

MAC Address: Véase *Dirección MAC*.

Mensaje *Hello*: Los denominados mensajes *Hello* (del inglés "Hola", como saludo) de manera genérica son mensajes, de un formato sólo determinado por el protocolo específico, intercambiados entre nodos de un enlace. La entrega periódica de estos mensajes facilita diversa información sobre conectividad del enlace, vecinos conectados, así como también identidades de nodos vecinos u otros datos (según el formato, frecuencia y protocolo en sí). En el caso concreto de la familia de protocolos All-Path descrita en el documento, estos mensajes permiten determinar a los switches All-Path si al otro lado del enlace tienen un vecino All-Path o no (es decir, si es otro tipo de switch o un dispositivos final o host).

Metropolitan Area Network: Véase *Red de Área Metropolitana.*

Multiple Spanning Tree Instance: Instancia de árbol multiple de expansión.

Protocolo de Estado de Enlace: Un protocolo de estado de enlace (*link state*) es un protocolo de encaminamiento en el que cada router R determina quiénes son sus vecinos y reenvía a todos los otros routers un paquete, denominado como *Link State Packet* (LSP), que consiste en información del tipo "*Yo soy R y mis routers vecinos son X (con un coste de enlace c1), Y (coste c2), y Z (coste c3)*".

Plug-and-Play (en español "enchufar y usar"): Tecnología que permite a cierto dispositivo el poder ser conectado o puesto en marcha sin tener que configurar, ni proporcionar ningún parámetro extra a sus controladores.

Puente Frontera: Se denomina puente frontera (o *edge bridge/switch*) a aquel puente que no está dentro del núcleo de cierta red y no sólo se conecta a otros puentes, sino que también está conectado a dispositivos finales, y por ello es frontera entre los dispositivos finales y el resto de la red.

Red de Área Amplia: Interconexión de ordenadores capaz de cubrir distancias amplias, aproximadamente desde unos 100 hasta unos 1000 kilómetros, proveyendo de servicio a un país o un continente. Un ejemplo de este tipo de redes sería RedIRIS o Internet.

Red de Área Local: Interconexión de una o varios ordenadores y periféricos. Su extensión está limitada físicamente a un edificio o a un entorno de 200 metros, con repetidores podría llegar a la distancia de un campo de 1 kilómetro.

Red de Área Metropolitana: Interconexión de ordenadores de tamaño similar a una ciudad o un campus grande (decenas de kilómetros). Una red de este tipo normalmente se trata de la interconexión de varias redes de área local usando una tecnología de alta velocidad.

Región MST: Conjunto de LANs y bridges MST conectados físicamente mediante puertos de los bridges MST.

Routing Bridge: Puente Encaminador. Dispositivo genérico que combina funcionalidad de encaminador y de puente.

RBridge: Routing Bridge con funcionalidad según la propuesta en grupo IETF RBridge [RBridge] (RFC 6325) del estándar TRILL (RFC 5556).

Synchronous Digital Hierarchy: Véase *Jerarquía Digital Síncrona*.

Wide Area Network: Véase *Red de Área Amplia.*

Abreviaturas

AE: *Árbol de Expansión.*

AMAC: *Actual MAC*.

ARP: *Address Resolution Protocol.*

ATM: *Asynchronous Transfer Mode*.

B-MAC: *Backbone-MAC*.

BCB: *Backbone Core Bridge.*

BGP: *Border Gateway Protocol.*

BPDU: *Bridge Protocol Data Units.*

CAM: *Content-Addressable Memory.*

CIST: *Common and Internal Spanning Tree*.

CLI: *Command-Line Interface*.

C-MAC: *Client-MAC.*

CPD: *Centro de Procesamiento de Datos.*

CST: *Common Spanning Tree*.

CSTI: *Common and Internal Spanning Tree Instance.*

DHCP: *Dynamic Host Configuration Protocol*.

DHT: *Distributed Hash Table.*

DLS: *Distributed Load Sharing.*

DRAM: *Dynamic RAM.*

DRB: *Designated RBridge.*

DSAP: *Destination Service Access Point.*

ECMP: *Equal Cost Multi-Path*.

ECMT: *Equal Cost Multi Tree*.

ESADI: *End-Station Address Distribution Information.*

FCoE: *Fiber Channel over Ethernet.*

FDDI: *Fiber Distributed Data Interface*.

FDB: *Forwarding Data Base.*

FID: *Filtering Data Base ID.*

FIFO: *First In, First Out.*

FPGA: *Field-Programmable Gate Array*.

HDL: *Hardware Description Languague*.

HLMAC: *Hierarchical Local MAC*.

HSRP: *Hot Standby Router Protocol.*

IEEE: *Institute of Electrical and Electronic Engineers.*

IETF: *Internet Engineering Task Force.*

IGMP: *Internet Group Management Protocol.*

IPsec: *IP Security.*

IPv4: *IP versión 4*.

IPv6: *IP versión 6*.

IS-IS: *Intermediate System to Intermediate System*.

ISO: *International Organization for Standardization.*

IST: *Internal Spanning Tree.*

ISP: *Internet Service Provider.*

LAN: *Local Area Network.*

LLC: *Logical Link Control.*

LDP: *Location Discovery Protocol.*

LSP: *Link State Packet.*

MAC: *Media Access Control.*

MAN: *Metropolitan Area Network.*

MMRP : *Multicast MAC Registration Protocol.*

MPLS: *Multiprotocol Label Switching.*

MSCI: *Multiple Spanning Tree Configuration Identifier.*

MSTI: *Multiple Spanning Tree Instance.*

MSTID: *Multiple Spanning Tree ID.*

MSTP: *Multiple Spanning Tree Protocol* (IEEE 802.1s → 802.1Q; 2005).

MST: *Multiple Spanning Tree.*

MT: *Multitree.*

MTU: *Maximum Transmission Unit*.

NAT: *Network Access Translator.*

NDP: *Neighbour Discovery Protocol.*

OSPF: *Open Shortest Path First.*

PBB: *Provider Backbone Bridge.*

PCI: *Peripheral Component Interconnect.*

PCI-e: *PCI Express.*

PLSB: *Provider Link State Bridging.*

PMAC: *Pseudo MAC.*

PPP: *Point-to-Point Protocol.*

PSSR: *Port-Switching Source Routing.*

PV: *Path Vector.*

RAM: *Random-Access Memory.*

RPF: *Reverse Path Forwarding*.

RSTAR: *RSTP-based STAR.*

RSTP: *Rapid Spanning Tree Protocol* (IEEE 802.1w; 2001).

RV: *Reflection Vector.*

SDH: *Synchronous Digital Hierarchy.*

SDS: *Source Dependent Spanning Trees.*

SFP: *Small Form-factor Pluggable.*

SNMP: *Simple Network Management Protocol*

SPB: *Shortest Path Bridging* (IEEE 802.1aq; 2012).

SPBM: *Shortest Path Bridging-MAC*.

SPBV: *Shortest Path Bridging-VID.*

SRAM: *Static RAM.*

SSAP: *Source Service Access Point.*

STAR: *Spanning Tree Alternate Routing Protocol.*

STI: *Spanning Tree Instance.*

STP: *Spanning Tree Protocol* (IEEE 802.1D; 1990/1998/2004).

TCAM: *Ternary Content-Addressable Memory (CAM).*

ToR: *Top of Rack.*

TLV: *Type-Length-Value.*

TRILL: *Transparent Inteconnection of Lots of Links* (RFC 5556/6325).

TTL: *Time To Live*.

VDC: *Virtual Data Center*.

VID: *VLAN ID.*

VLAN: *Virtual LAN.*

VLB: *Valiant Load Balancing*.

VRRP: *Virtual Router Redundancy Protocol.*

WAN: *Wide Area Network.*

Apéndice A: Protocolos de Prohibición de Giros

En este apéndice se describen los protocolos basados en prohibición de giros, que utilizan sistemas alternativos al árbol de expansión que buscan ser más eficientes en la prevención de bucles en la red de forma que se inhabiliten en menor grado los enlaces utilizados, prohibiendo determinados *giros* en los itinerarios por nodos de la red, en vez de inhabilitar enlaces. Para ello primero se explica el concepto del encaminamiento arriba/abajo (*up/down routing*) brevemente y después se muestran algunos algoritmos de prohibición de giros.

Encaminamiento arriba/abajo (Up/down routing)

Este encaminamiento se basa en asignar un sentido a todos los enlaces de la red según la posición del vértice del enlace en el árbol de distribución (arriba si está mas cercano al puente raíz, abajo si al contrario), para lo cual se asignan identificadores crecientes a los bridges partiendo del bridge raíz y descendiendo nivel a nivel, como se muestra en la siguiente figura. En el caso de enlaces entre nodos a la misma altura, estos reciben la orientación según la identidad del puente sea mayor o menor.

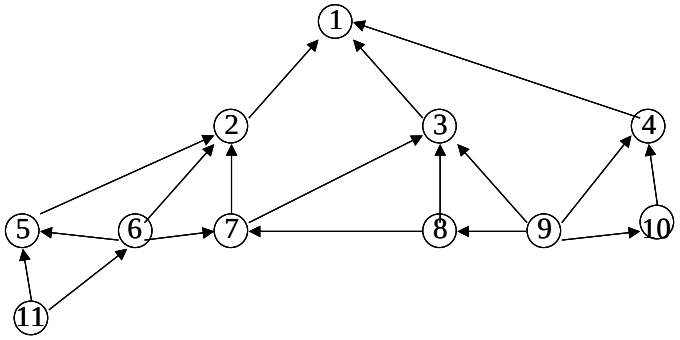


Figura 121. Ejemplo de asignación de identificadores crecientes en los bridges desde el raíz

El encaminamiento arriba/abajo evita los bucles porque un bucle siempre contiene dos giros abajo/arriba o arriba/abajo. Por lo tanto, si se obliga a que en cualquier recorrido, una vez que se ha girado hacia abajo, no pueda volver a girar hacia arriba, se evitarán los bucles en las tramas enviadas. Una ruta legal es la que nunca atraviesa un enlace en la dirección hacia arriba tras haber usado uno hacia abajo. La efectividad del encaminamiento arriba/abajo depende principalmente de la elección del árbol de distribución, y en particular del puente raíz. Para la topología típica de las redes campus estructuradas jerárquicamente esto tiene menor incidencia que para una red mallada arbitraria. El problema del encaminamiento

arriba/abajo es que no garantiza rutas óptimas, y que se hace menos eficiente a medida que la red se hace más compleja.

Algoritmos de Prohibición de Giros

Los algoritmos basados en Prohibición de Giros (*Turn Prohibition*, TP) constituyen una evolución del encaminamiento arriba/abajo en la que se optimizan los giros a prohibir.

Una versión de interés de TP es la propuesta en 2002 en [SKZ02]. Dicho interés proviene de la oportunidad de poder aplicar el denominado *Network Calculus* a las redes, para lo que éstas deben ser de tipo *feed-forward*. Para garantizar esto, las redes deben estar libres de bucles y esto se logra desarrollando un algoritmo que prohíbe algunos *giros* en la red, en lugar de cortar enlaces completos, como hacen los protocolos de árbol de expansión como STP. Otra propuesta posterior se denomina *Tree-Based Turn Prohibition Protocol* (TBTP) [Pel+04] y referencia los giros prohibidos o no con relación al árbol de expansión. La aplicación de la prohibición de giros en los algoritmos de encaminamiento como el de Djikstra (SPF) no es posible de forma simple (eliminando los caminos que contienen giros prohibidos de la tabla de encaminamiento) porque entonces los caminos no son mínimos debido a que Djikstra opera de forma escalonada.

Apéndice B: Red Clos

En el campo de las telecomunicaciones, una *Clos network* o red Clos es un tipo de red de circuitos en varias etapas que representa una idealización teórica de sistemas de conmutación multi-etapa telefónicos. El primero en formalizar la definición fue Charles Clos [Clo52], de ahí el nombre de las mismas.

Se requieren redes Clos cuando las necesidades del circuito de conmutación exceden la capacidad del camino más viable en el mismo. La ventaja clave de las redes Clos es que el número de puntos de cruce requeridos en la conexión de los switches/conmutadores en el circuito puede ser mucho menor que en el caso de que el circuito estuviese implementado por un solo switch. Cuando surgió la idea de la red Clos, el número de puntos de cruce era una indicación aproximada bastante razonable del coste del sistema total de conmutación. Mientras que esto último era aceptable en el sentido de cruces electromecánicos, se ha ido haciendo menos relevante con la llegada de los VLSI (integración de circuitos a gran escala).

Las redes Clos poseen tres etapas: entrada (*ingress stage*), central (*middle stage*) y salida (*egress stage*). Cada etapa está compuesta de una serie de conmutadores (véase Figura 122), a menudo llamados *crossbars*. En el caso telefónico, cada llamada que llega por un conmutador de entrada puede ser encaminada por cualquiera de los conmutadores de etapa central disponibles, hacia el conmutador de salida correspondiente. Un *crossbar* de etapa central está disponible para una nueva llamada concreta si tanto el enlace que conecta el conmutador de entrada con el central, como el enlace que conecta el conmutador central con el de salida, están libres.

Estas redes están definidas por tres enteros: *n*, *m* y *r*. *n* representa el número de fuentes que meten tráfico en cada uno de los *r* conmutadores de etapa de entrada. Cada conmutador de entrada posee *m* salidas y existen *m* conmutadores de etapa central. Existe una conexión entre cada conmutador de entrada y central, exactamente. Hay *r* conmutadores de salida, cada uno con *m* entradas y *n* salidas. Cada conmutador central está conectado exactamente una vez con cada conmutador de salida. Por lo tanto, cada etapa de entrada tiene *r* conmutadores de *n* entradas y *m* salidas cada uno, la etapa central tiene *m* de *r* entradas y *r* salidas, y la etapa de salida posee *r* de *m* entradas y *n* salidas.

El concepto de red Clos puede ser generalizado a cualquier número impar de etapas. Por ejemplo, reemplazando cada *crossbar* de etapa central por una red Clos de 3 etapas, se pueden construír redes Clos de 5 etapas. Aplicando el mismo proceso reiteradamente, se pueden generar redes Clos de 7, 9, 11,… etapas.

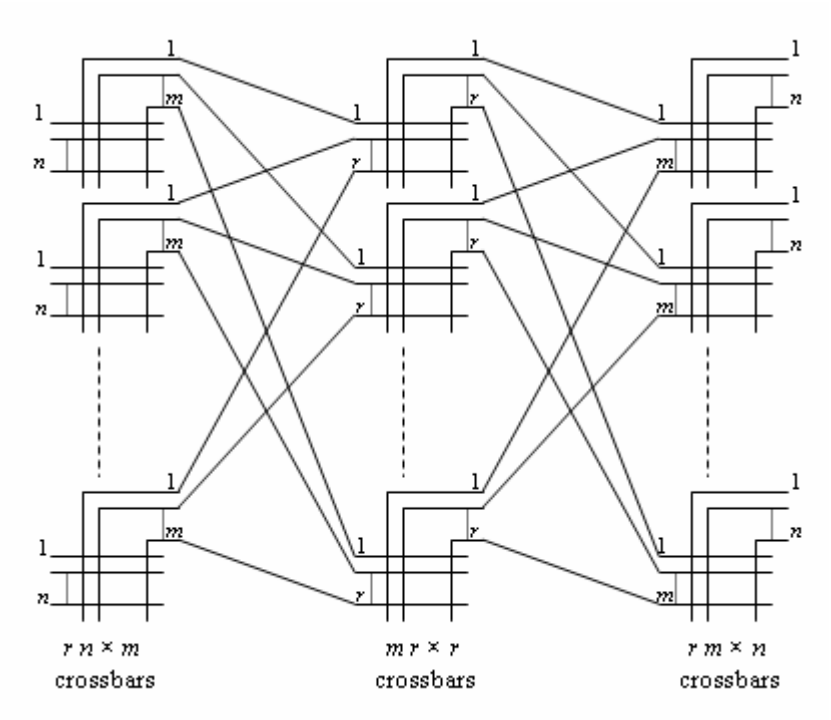


Figura 122. Diagrama de una red Clos [Wikipedia]

Apéndice C: Topología *fat tree*

La topología *fat tree*, sacada a la luz por Charles E. Leiserson [Lei85] del MIT (*Massachusetts Institute of Technology*), es una red universal de comunicaciones de demostrada eficiencia. A diferencia del concepto simple de *tree* o de red Clos, que poseen enlaces iguales, de pequeña capacidad, por toda la red, los enlaces de un fat tree se vuelven "más gordos" según subimos en la jerarquía hacia el root, es decir, sus capacidades aumentan.

Eligiendo apropiadamente la "gordura" o capacidad de los enlaces, la red puede ser diseñada para usar eficientemente cualquier ancho de banda disponible gracias a tecnologías de comunicación y encapsulado. En contraste, otras redes de comunicaciones, tales como los hipercubos o las mallas, poseen requerimientos de comunicación que siguen una ley matemática especificada de antemano y, por ello, no pueden ser diseñadas en base a tecnologías de empaquetamiento específicas.

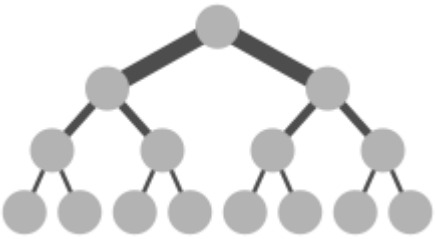


Figura 123. Diagrama de un *fat tree* [Wikipedia]

Algunos ejemplos aplicados de *fat tree* son la supercomputadora o superordenador Connection Machine Model CM5 (de entorno al año 1990), que usaba una red de interconexión *fat tree*. Mercury Computer Systems también usó una red hypertree, una variante de fat tree, en sus ordenadores destinados a computación paralela. Los fat tres son también preferidos en las arquitecturas en clúster de InfiniBand.

A finales de agosto de 2008, un grupo de investigadores de la UCSD (*University of California, San Diego*) publicaron un diseño escalable para una arquitectura de red que usa una topología inspirada en la topología *fat tree* (llamándolo de manera confusa *fat tree* en su paper). La arquitectura usa switches de bajo coste que son más baratos y más eficientes energéticamente que switches de más alto nivel, dedicados. Esta topología es en realidad un ejemplo especial de una red Clos, más que un *fat tree*. De esta arquitectura parte PortLand [Mys+09], diseño definido en el capítulo dedicado al estado del arte y que sirvió como inspiración inicial de Torii-HLMAC [Roj12].

Apéndice D: Direccionamiento Hierarchical Local MAC (HLMAC)

Las direcciones jerárquicas locales (*Hierarchical Local MAC*, HLMAC) son direcciones MAC de longitud variable encapsuladas en el campo estándar de dirección MAC de la trama de datos estándar, el cual tiene una longitud fija de seis octetos (48 bits). Al ser jerárquicas, se aprovechan los mensajes de creación de los árboles de expansión para establecer la jerarquía de direcciones y fijarlas. Los puentes que soportan este nuevo nombrado se van a llamar puentes combinados.

Tipos de HLMAC

El nombrado de los puentes con las direcciones HLMAC se hace de forma jerárquica según una de las siguientes alternativas:

Sin longitud explicita de prefijo:

El primer bit transmitido contiene el valor estándar para direcciones individuales y el segundo bit contiene el valor estándar para direcciones MAC locales (HLMAC). Se usa la identidad de puerto cero (0) para indicar la terminación de la dirección jerárquica dentro de los 48 bits de la dirección MAC, por lo que empezando por la derecha en la siguiente tabla los octetos a cero indican que la dirección es más corta que los 6 octetos disponibles para niveles de direccionamiento, puesto que estos octetos no tienen significado de dirección:

Octeto	0	1	2	3	4	5
Valor binario	01000101	10001100	00110011	11000011	00111100	00000000

Figura 124. Ejemplo de HLMAC sin longitud explícita de prefijo [Per11]

En la Figura 124 se representa la dirección HLMAC 5.140.51.195.60, que corresponde a un direccionamiento de cinco niveles, por lo que el último octeto, correspondiente a un nivel 6, se pone a cero. El primer octeto tiene los dos primeros bits asignados con los valores estándar comentados y los restantes bits se usan para el nivel 1 de la dirección HLMAC. Esta alternativa ofrece mayor sencillez de direccionamiento y mayor rango en el nivel 1 de la dirección, aunque cabe destacar que con este tipo de nombrado de puentes, el número de puertos máximo por puente de la red se limita a 255 puertos en los puentes intermedios y hoja, es decir, se usa el rango 1 a 255, y en el puente raíz, debido al uso de los dos primeros bits por el protocolo, el límite de de puertos es 63, es decir, se usa un rango de de 1 a 63.

Con longitud explicita de prefijo:

En el primer octeto, tras el primer y el segundo bits transmitidos que, como en el caso anterior, indican que se trata de una dirección individual y local

respectivamente, se utilizan por ejemplo los 3 siguientes bits (tercero, cuarto y quinto) de dicho primer octeto para indicar el numero de niveles de la dirección jerárquica, teniendo entre 1 y 6 niveles disponibles. El resto de bits del primer octeto, bit sexto a octavo ambos incluidos, se usan como nivel 1 de la dirección HLMAC. En este caso el numero máximo de puertos en los puentes se limita a 256, rango de 0 a 255, en el ámbito general (aunque en la realidad los puentes no tienen puerto 0, así que el numero máximo de puertos por puente será 255 aunque el rango permita 256) y, de forma particular, a 8 puertos en el primer nivel o nivel 1, que será el del puente raíz. La ventaja es que el número de niveles es fácilmente configurable.

Octeto	0	1	2	3	4	5
Valor binario	01101101	10001100	00110011	11000011	00111100	00000000

Figura 125. Ejemplo de HLMAC con longitud explícita de prefijo [Per11]

En la figura anterior se representa la dirección HLMAC 5.140.51.195.60 con cinco niveles, que se indican en el primer octeto con el campo longitud de dirección formado por los bits subrayados en la tabla.

Como el nombrado de los puentes con HLMAC se basa en el nombrado original de RSTP, se pueden realizar acciones de envío y retransmisión de tramas de cualquiera de las dos formas, teniendo así una alternativa válida al encaminamiento RSTP en caso de fallo, siempre y cuando, claro está, no haya que renombrar de nuevo los puentes con HLMAC, siendo imposible el encaminamiento de ninguna de las dos alternativas. En el caso de Torii-HLMAC se utilizan direcciones sin longitud explícita de prefijo.

Características de las direcciones HLMAC (respecto a MAC e IP)

Las direcciones HLMAC presentan ventajas operativas para el encaminamiento y direccionamiento dinámico puesto que expresan conectividad jerárquica desde el origen de la red (puente raíz). Tienen características comunes con las direcciones IP y MAC aunque, por ejemplo, tienen mas información topológica y menos identidad, pudiendo adquirirla si se asocian con la identidad del puente raíz o con una dirección MAC o IP.

En la siguiente tabla se muestran las principales características por comparación con las direcciones antes nombradas (IP y MAC):

Cualidad de la dirección	Direcciones MAC	Direcciones IP	Direcciones HLMAC
Longitud	48/64 bits	32/128 bits	48 bits/variable
Unicidad	Si	Si	No[40]
Validez	Mundial	Mundial	Local[40]
Jerárquicas/Planas	Planas	Jerárquicas	Jerárquicas
Vinculación al Hardware	De fábrica	No	Si (por número de puerto)
Asignación	IEEE/Fabricante	Manual/ Semiautomática	Automática
Identificación	NIC, físicamente, fabricante	Conectividad lógica, subred	Conectividad física, posición en el árbol de expansión
Adecuado para	NIC	Enlace enrutador	Enlace puentes combinados
Utilizada por	Puentes/ Conmutadores/ Clientes	Enrutadores	Puentes combinados
Estabilidad	Alta. Física	Media. Administrativa (ISP)	Media. Topológica
Actualización	Ninguna	Manual (excepto DHCP)	Automática

Tabla 15: Comparativa entre direcciones MAC, IP y HLMAC [Per11]

Apéndice E: Herramientas de simulación e implementación utilizadas en la Tesis

En el presente apartado se realizará una pequeña introducción de las diferentes herramientas que se han utilizado para llevar a cabo diferentes partes de la Tesis.

OpenFlow

Históricamente, las redes de ordenadores han evolucionado "caja a caja" o "dispositivo a dispositivo", es decir, con elementos individuales de red ocupando puestos específicos como los routers, switches, distribuidores de carga, traductores de direcciones de red (*Network Address Translation*; NAT) o cortafuegos (*firewall*). Las redes definidas por software, como OpenFlow, proponen dar la vuelta a dicha naturaleza, convirtiendo la red en conjunto en una plataforma y los elementos individuales de red en entidades programables. Las aplicaciones que se ejecutan en la plataforma de red pueden optimizar los flujos de tráfico de manera que escojan el camino mínimo, tal y como los protocolos distribuidos realizan actualmente, pero también pueden optimizar la red para maximizar la utilización de los enlaces, crear diferentes dominios de accesibilidad para diferentes usuarios, o hacer la movilidad de dispositivos sin cortes.

OpenFlow [OpenFlow], un estándar abierto que permite generar redes definidas por software, es una nueva tecnología de red que permitirá muchas nuevas aplicaciones y maneras de manejar redes. Se puede considerar que OpenFlow es una solución que facilita a los investigadores realizar experimentos en switches y routers heterogéneos de forma uniforme, sin la necesidad de que los fabricantes expongan en detalle el hardware y funcionamiento de sus dispositivos, ni de que los investigadores tengan que escribir software de control específico basado en la norma del fabricante.

Diseño

El diseño de OpenFlow se define de manera bastante clara en su whitepaper [MAB+08]. La idea es bastante básica: se aprovecha el hecho de que la mayoría de switches y routers contienen tablas de flujos (normalmente construidas en base al uso de memorias TCAM) para implementar diversas funciones. Mientras que dichas tablas son diferentes según el fabricante, el grupo de trabajo de OpenFlow identificó un interesante conjunto de funciones que corren en muchos switches y routers. Así, OpenFlow aprovecha este conjunto y aporta un protocolo abierto para programar las tablas de flujos en switches y routers diferentes. Los investigadores pueden controlar sus propios flujos – escogiendo qué rutas seguirán sus paquetes y el procesamiento que recibirán. De este modo, los investigadores pueden probar nuevos protocolos de encaminamiento, modelos de seguridad, esquemas de direccionamiento e incluso alternativas a IP.

Un switch OpenFlow contiene las siguientes partes:

1. Una tabla de flujos (*Flow Table*), con una acción asociada a cada entrada de flujo, para decirle al switch cómo procesar dicho flujo.

2. Un canal seguro de comunicación (*Secure Channel*) que conecta el switch con un proceso de control remoto, denominado controlador (*controller*), que permite intercambiar comandos y paquetes entre el controlador y el switch usando el protocolo OpenFlow.

3. El protocolo OpenFlow (*OpenFlow Protocol*), que proporciona un método abierto y estándar de comunicación entre un controlador y un switch. No es necesario programar el switch si simplemente se indica la interfaz del switch que se comunicará, usando el protocolo OpenFlow, con el controlador y que definirá las entradas de la tabla de flujos de manera externa.

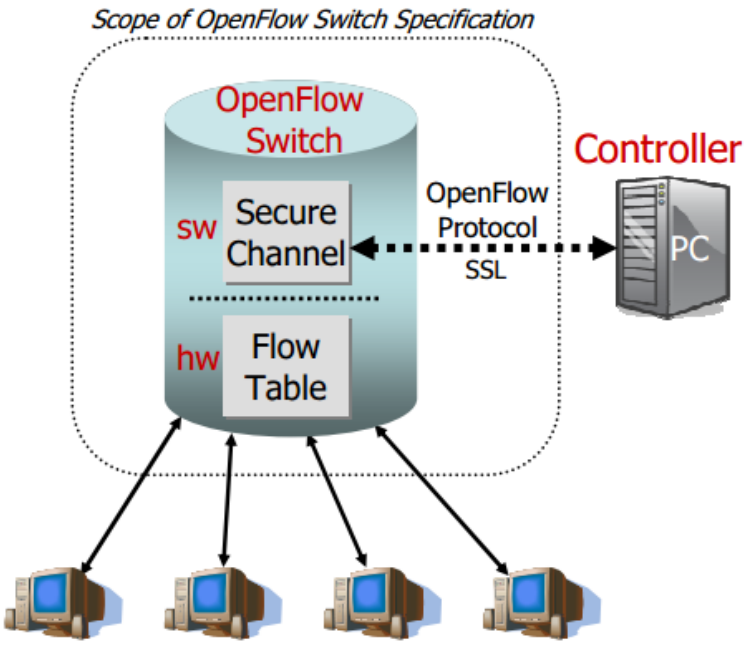


Figura 126. Switch OpenFlow idealizado. La tabla de flujos se controla con un controlador remoto a través de un canal seguro de comunicación [MAB+08]

La aplicación de OpenFlow es muy sencilla, sólo es necesario disponer de un dispositivo dedicado para que trabaje como controlador (cualquier PC con Linux, no necesariamente potente, es válida para instalar el software de controlador OpenFlow por ejemplo) y una serie de switches que soporten el estándar OpenFlow, con los que construiremos la topología sobre la que probaremos el prototipo de la aplicación que se haya desarrollado. Existen muchos tipos de switches OpenFlow, tanto de fabricantes (como NEC o IBM), como no: por ejemplo, podemos crear un switch OpenFlow a partir de un sistema operativo Linux o de una tarjeta NetFPGA (tecnología que veremos en detalle en el siguiente apartado) o incluso es posible crear una red virtual de switches OpenFlow a partir de la herramienta de virtualización Mininet [Mininet]. Y en el caso de controladores, un ejemplo típico es el controlador NOX [NOXRepo], que recientemente evolucionó en POX.

Posibles ejemplos de aplicación

A continuación, se muestran un par de aplicaciones reales basadas en posibles supuestos.

Ejemplo 1: Gestión del ancho de banda

Una red típica WAN posee una utilización del 30%-60% y necesita "reservar" ancho de banda para momentos de ráfagas de tráfico grandes (*burst*). Sin embargo, con OpenFlow se desarrolló un sistema en el que las aplicaciones internas (clientes) que necesitaran transferir grandes cantidades de datos (por ejemplo copias de seguridad de bases de datos, transmisión de grandes lotes de logs,…) pudieran usar el ancho de banda sobrante, libre. Para ello, los clientes registran la fuente, el destino y la cantidad de datos que ha de ser transferida, en un servicio central. Dicho servicio realiza varios cálculos y envía los resultados a los routers de manera que ellos sepan como encaminar esta gran cantidad de datos, cuando los enlaces estarían sin usar de ser otro modo. Las comunicaciones entre las aplicaciones y el servicio central son bidireccionales: las aplicaciones informan al servicio sobre sus necesidades, el servicio responde cuando el ancho de banda está disponible, y la aplicación informa al servicio cuando ha terminado. Mientras tanto, los routers informan al servicio con su nivel de uso en tiempo real. Como resultados, la red pasa a tener una utilización del 90%-95%.

Ejemplo 2: Servidor de juegos

Algunos compañeros de universidad configuran un servidor del juego *Quake* sobre un portátil sobrante para realizar una pequeña competición privada. La latencia a este servidor es importante no sólo a la hora de jugar, sino para ser justos en general, y los recursos son limitados al estar usándose un portátil que sobraba. Durante la competición, alguien coge el portátil y lo desconecta de la red Ethernet, uniéndose a la red Wi-Fi, y luego camina con el portátil hasta la cafetería y regresa una hora después. A lo largo de dicho paseo, el portátil cambia cuatro veces de IP, y el ancho de banda varía de 100Mbps en la red cableada Ethernet, a 11Mbps en la conexión 802.11b del edificio de ciencias de la computación, a 54Mbps en la conexión 802.11g en la cafetería. Sin embargo, la competición ha continuado sin interrupción Usando OpenFlow, los estudiantes escriben una aplicación de manera que al margen de la subred a la que el portátil se mueva, la dirección IP no varíe. También, en el caso de congestión de red, se hará que el tráfico hacia y desde el servidor tenga la máxima prioridad: los routers tratarán de tirar cualquier otro tráfico antes que el relativo al juego, garantizando por tanto una buena calidad de juego para todos. Y todo esto se ha hecho sin cambiar nada en el portátil, sólo en la red.

NetFPGA

Una tarjeta NetFPGA [NetFPGA], como su nombre indica, se trata de un dispositivo de lógica programable, *Field-Programmable Gate Array* (FPGA), dedicado al ámbito específico de las redes (Net). Básicamente se trata de una plataforma hardware y software, de código abierto, diseñada principalmente para la investigación y como ayuda al profesorado. Actualmente se han desarrollado más de

2000 sistemas basado en NetFPGA en más de 150 instituciones en más de 40 países alrededor del mundo, generando además más de 50 papers académicos.

Existen dos plataformas: NetFPGA-1G (1G) y NetFPGA-10G (10G). Éstas permiten construir, a investigadores y estudiantes, prototipos de sistemas de red de alta velocidad y gran rendimiento hardware:

- La plataforma NetFPGA-1G (desde 2007) es una tarjeta PCI que contiene una FPGA Xilinx Virtex-II Pro 50, 4 puertos Ethernet Gigabit, memoria estática *Static RAM* (SRAM) y memoria dinámica *Double-Date Rate* (DDR2) *Dynamic RAM* (DRAM).

Figura 127. Tarjeta NetFPGA-1G [NetFPGA]

- La plataforma NetFPGA-10G (desde 2012) es una tarjeta PCI-e que contiene una FPGA Xilinx Virtex 5, 4 interfaces SFP+, soporta tanto los modos de 1G y 10G, memoria estática QDRII SRAM y memoria dinámica RLDRAM.

Figura 128. Tarjeta NetFPGA-1G [NetFPGA]

Las plataformas hardware son las dos ya comentadas, mientras que el software se trata de una instalación en un PC, preferiblemente con el sistema operativo CentOS [CentOS] (pero pueden usarse otros). Tras el montaje de la tarjeta (conexión vía PCI o PCI-e), o las tarjetas (puede haber más de una por PC), y la instalación del

sistema operativo, se realiza la instalación del software de la NetFPGA, que contiene una serie de proyectos de referencia (entre ellos, el de switch OpenFlow por ejemplo) y, adicionalmente, se puede instalar el software de Xilinx (licencia incluida con la compra del hardware), para realizar nuevos proyectos, poder compilarlos y descargarlos en la tarjeta. También es posible comprar el sistema ya completo (no sólo la tarjeta), con el software instalado y todo listo para funcionar.

En cada momento, la tarjeta (o tarjetas) sólo puede tener un único proyecto cargado y, una vez cargado, actúa de manera totalmente independiente al software, que sólo es la base de instalación y configuración de parámetros de la tarjeta, principalmente. Además de los proyectos de referencia (creados por la universidad de Standford inicialmente y actualmente también la de Cambridge), la comunidad NetFPGA permite la contribución y aportación de otros autores.

OMNeT++

Finalmente, una última herramienta importante, quizás la que más, ha sido el simulador de eventos discretos OMNeT++ [OMNeT ++], que es de código abierto. Las herramientas de simulación de eventos discretos, son programas informáticos que permiten simular la operación de un sistema mediante la representación de ésta como una secuencia de eventos cronológicamente ordenados. Cada evento ocurre en un instante de tiempo y conlleva un cambio en el estado del sistema. La transición entre eventos ocurre de forma instantánea. Entre cada par de eventos consecutivos, el tiempo de simulación salta al comienzo del evento siguiente, se actualiza el estado de las variables y vuelve a ocurrir un salto al instante (fijado previamente) de comienzo del evento próximo. Estos saltos en el tiempo de simulación son los que implican la denominación como "*discretas*" de estas herramientas. OMNeT++ toma su nombre de *Objective Modular Network Testbench* conjuntamente con el lenguaje de programación que emplea C++.

Figura 129. Logotipo de OMNeT++ 4 (versión actual en 2012) [OMNeT++]

Diseño

La base del entorno de simulación es el núcleo o kernel. En éste se encuentran programadas las funciones generales que permiten la extensión modular del programa para la simulación en áreas específicas mediante herencia e interfaces en C++. Dentro de cada extensión o *framework* específico se pueden definir los modelos de simulación. Un modelo en OMNeT++ consiste en una serie de módulos jerárquicamente anidados, los cuales se comunican entre sí mediante mensajes. La relación existente entre los módulos se define empleando un lenguaje propio del simulador, denominado *Network Description Language* (NED), cuyo formato es muy sencillo y que refuerza el carácter modular del entorno.

Una vez se tienen definidos los módulos a usar, ya sean módulos existentes previamente o creados para tal fin, se pasa a definir topologías concretas sobre las que realizar una configuración de elementos y módulos implicados. Una vez decididos los rangos de parámetros, se crea un fichero de inicialización de la topología, por defecto "*omnet.ini*", que contiene los valores concretos de cada parámetro a usar en cada experimento y conjunto de medidas. Se crean instancias concretas de ejecución y se ejecuta la simulación.

El software permite dos entornos de ejecución, uno gráfico (*Tkenv*) y otro a través de un intérprete de comandos (*Cmdenv*). Trabajando en modo gráfico, se permite una simulación paso a paso (de eventos), hasta un determinado instante o simulación de forma continua con tres posibles velocidades (normal, rápida y exprés), lo que lo dota de mucha versatilidad. Durante las primeras etapas de desarrollo la interfaz visual (entorno gráfico) es una herramienta didáctica y de detección de errores muy útil aunque, conforme se avanza en el proceso, suele ser preferible el trabajo en modo línea de comandos al ser éste último mucho más veloz.

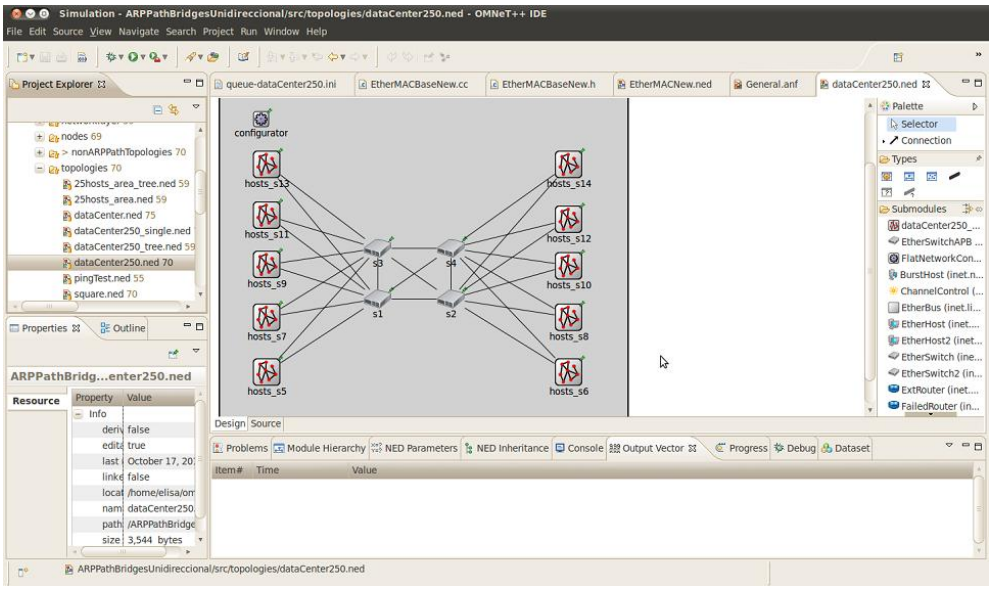


Figura 130. Interfaz visual de OMNeT++ 4.1 para Linux Ubuntu

A lo largo de la simulación se van produciendo una serie de eventos programados en el tiempo dentro de la cola de espera principal. Estos eventos pueden implementar bien mensajes entre los módulos simples o bien auto mensajes para simular temporizadores. La transmisión de mensajes se puede realizar entre módulos directamente conectados, a través las interfaces directas entre módulos simples o los canales de comunicación entre objetos y módulos compuestos. También se pueden transmitir mensajes directamente entre un par de módulos cualquiera en caso de ser necesario hacer llamadas por ejemplo pasando varios niveles de jerarquía.

Durante cada ejecución de la simulación se obtienen resultados de salida de los distintos módulos así como colecciones de estadísticas, que se van almacenando en ficheros escalares y vectoriales para su posterior procesado y visualización en

herramientas gráficas (este conjunto de medidas, cuáles obtener y cuándo, se indican en el *.ini anteriormente indicado o se aplicarán valores por defecto).

La librería INET

OMNeT++ es el simulador como tal, pero a la hora de desarrollar un proyecto concreto, sin tener que empezar de cero, es necesario utilizar una librería. En el caso de las redes de comunicación, un paquete muy útil es el *INET Framework*, que es un paquete de código abierto para simulación de redes de comunicación para el entorno de OMNeT. INET contiene modelos para diferentes protocolos de red, cableados y no (wireless), incluyendo UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSPF, y muchos otros.

Apéndice F: Simulador de flujos base para OMNeT++

A la hora de realizar pruebas de simulación en el entorno OMNeT++ [OMNeT++], la librería INET [INET] ofrece diversas aplicaciones como son las aplicaciones UDP y TCP, sin embargo actualmente no ofrece modelos de flujos para realizar simulaciones para poner en práctica protocolos en escenarios concretos, dado que las aplicaciones tienen que configurarse una a una y no ofrecen un modelo como tal, sino simplemente cierto tipo de aplicación parametrizable a gusto del usuario. Por lo tanto, para las simulaciones avanzadas de OMNeT++, en las que se trabaja con la familia All-Path principalmente para medir reparto de carga, utilización de enlace y latencias (para comparar con el estándar SPB), se implementó un simulador de OMNeT++ de apoyo a la librería INET usando la aplicación UDP que posee como base.

Este simulador de flujos se escribió primero en Python (con la librería SimPy [SimPy]) y posteriormente se implementó en OMNeT++ una vez demostrado su correcto funcionamiento. Para desarrollarlo se asume el estado cuasi estático documentado en el modelo estocástico definido en [Neu81]. El parámetro de entrada es la frecuencia de generación de flujos, así pues, cada vez que se crea un nuevo flujo, se le asigna una tasa de transferencia (Mbps) y un volumen de tráfico (MB) de acuerdo con el modelo de tráfico (a partir de la tasa y el tamaño se calcula la duración del flujo en el sistema) y un par de dispositivos (origen, destino) según la matriz de tráfico, y a partir de ahí se inicia la comunicación en base a flujos UDP con el correspondiente intercambio de mensajes ARP al comienzo de la comunicación y cada vez que vence la caché de ARP, tal y como está implementado por defecto en OMNeT++.

La matriz de tráfico por defecto es uniforme (todos los nodos tienen una probabilidad uniforme de ser escogidos como origen o destino), aunque pueden meterse pesos concretos en ciertos nodos como parámetros también, para que tengan una probabilidad mayor de ser elegidos. Mientras que el modelo de flujo posee las siguientes características:

- Tasa de transferencia:
    - 30% - 0,5 Mbps
    - 60% - 1 Mbps
    - 10% - 10 Mbps
- Tamaño: distribución Pareto (alfa=1,3) truncada:
    - Mínimo: 8 MB

- Máximo (truncado): 8 GB
- Media: 34, 7 MB

Por lo tanto, dado que la media de la tasa de transferencia es 1,75 Mbps (0,3*0,5 + 0,6*1 + 10*1) y la media del tamaño del flujo es 34,7 MB, la duración media de un flujo en este modelo será de unos 3 minutos, concretamente 159 segundos (34,7*8/1,75).

Referencias

[802.1Q] *IEEE 802.1Q-2003 IEEE standard for local and metropolitan area networks - Virtual Bridged Local Area Networks*- IEEE, 2003. Disponible en [Internet]: http://standards.ieee.org/getieee802/download/802.1Q-2003.pdf (Julio 2005).

[802.1s] IEEE. *802.1s-Multiple Spanning Trees.* Disponible en [Internet]: http://www. ieee802.org/1/ pages/802.1s. html

[ALV08] M. Al-Fares, A. Loukissas, A. Vahdat. *A Scalable, Commodity Data Center Network Architecture.* Proceedings of Sigcomm 2008. Disponible en [Internet]: http://www.cs.kent.edu/~javed/class-CXNET09S/papers-CXNET-2009/FaLV08-DataCenter-interconnect-p63-alfares.pdf

[Anr98] The Anritsu Company. *The must-have reference for multilayer switching*. Disponible en [Internet]: http://www.zurich.ibm.com/pdf/AnritsuGlossary.pdf

[ArpCacheLife] ArpCacheLife (TechNet Microsoft). Disponible en [Internet]: http://technet.microsoft.com/en-us/library/cc957524.aspx

[arpLinux] arp – Linux ARP kernel module. Disponible en [Internet]: http://linux.die.net/man/7/arp

[arp-sk] arp-sk: A swiss knife tool for ARP. Disponible en [Internet]: http://sid.rstack.org/arp-sk/

[ARR+10] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat. *Hedera: Dynamic Flow Scheduling for Data Center Networks.* Proceedings of NSDI 2010. Disponible en [Internet]: http://www1.icsi.berkeley.edu/~barath/papers/hedera-nsdi10.pdf

[AWM] Andrew W. Moore. Cambridge Computer Laboratory. Faculty of Computer Science and Technology. University of Cambridge. Disponible en [Internet]: http://www.cl.cam.ac.uk/~awm22/

[BA84] L. Bhuyan, D. Agrawal. *Generalized Hypercube and Hyperbus Structures for a Computer Network.* IEEE trans. Computers, April 1984.

[BCF+94] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawlk, C. L. S. ad J. N. Seizovic, W. Su, *Myrinet: A Gigabit-per-second local area network.* IEEE Micro, vol. 15, p. 29–36, Febrero 1994.

[BCK+11] H. Ballani, P. Costa, T. Karagiannis, A. Rowstron. *Towards Predictable Datacenter Networks.* Proceedings of Sigcomm 2011. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=2018465

[Car+11] J. Carlson et al. *PPP Transparent Interconnection of Lots of Links (TRILL) Protocol Control Protocol.* RFC 6361, Agosto 2011. Disponible en [Internet]: http://tools.ietf.org/html/rfc6361

[Car13] J. A. Carral. *Contribución al Diseño de Conmutadores Transparentes Avanzados Basados en Tecnología Ethernet*. Tesis doctoral, Departamento de Automática, Universidad de Alcalá. Abril 2013

[CCL] Cambridge Computer Laboratory. Faculty of Computer Science and Technology. University of Cambridge. Disponible en [Internet]: http://www.cl.cam.ac.uk/

[CDC04] *Cisco: Data Center: Load Balancing Data Center Services SRND*. Marzo 2004. Disponible en [Internet]: https://learningnetwork.cisco.com/servlet/JiveServlet/previewBody/3438-102-1-9467/cdccont_0900aecd800eb95a.pdf

[CentOS] CentOS: The Community ENTerprise Operating System. Disponible en [Internet]: http://www.centos.org/

[CGW+10] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, W. Wu. *Generic and Automatic Address Configuration for Data Center Networks*. Proceedings of Sigcomm 2010. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=1851190

[Cisco] *Cisco*. Disponible en [Internet]: http:// www.cisco.com

[Clo52] C. Clos. *A Study of Non-Blocking Switching Networks*. Bell System Technical Journal 32 (2): 406–424. 1953. ISSN 00058580. Disponible en [Internet]: http://www3.alcatel-lucent.com/bstj/vol32-1953/articles/bstj32-2-406.pdf

[CBP95] K. C. Claffy, H. –W. Braun, G. C. Polyzos. *A Parameterizable Methodology for Internet Traffic Flow Profiling*. JSAC, 13, 1995.

[DM78] S. Dalal, R. Metcalfe. *Reverse path forwarding of broadcast packets*. Communications of the ACM Vol. 21, No. 12 pp. 1040-1048, December 1978.

[DP88] Roy C. Dixon, Daniell A. Pitt. *Addressing, bridging and source routing*. IEEE Network. Enero 1988.

[DS80] Edsger W. Dijkstra, C.S. Scholten. *Termination detection for diffusing computations*. Information Processing Letters, 11(1): 1-4, Agosto 1980.

[DT04] W. J. Dally, B. Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers, 2004.

[Eas+11] D. Eastlake et al. *Routing Bridges (RBridges): Adjacency.* RFC 6327, Julio 2011. Disponible en [Internet]: http://tools.ietf.org/html/rfc6327

[ebtables] ebtables. Disponible en [Internet]: http://ebtables.sourceforge.net/

[EC09] K. Elmeleegy, A. L. Cox. *EtherProxy: Scaling Ethernet By Suppressing Broadcast Traffic.* Conference Publications, INFOCOM 2009, IEE. Pages 1584 – 1592. April 2009.

[Fed+12] D. Fedyk, P. Ashwood-Smith, D. Allan, N. Bragg, P. Unbehagen. *IS-IS Extensions Supporting IEEE 802.1aq Shortest Path Bridging.* RFC 6329, Abril 2012. Disponible en [Internet]: http://tools.ietf.org/html/rfc6329

[Fin05] N. Finn. *Shortest Path Bridging*. Septiembre 2005. Disponible en [Internet]: http://www.ieee802.org/1/files/public/docs2005/aq-nfinn-shortest-path-0905.pdf

[Fra+09] P. Francois, C. Filsfils, J. Evans, O. Bonaventure. *Achieving sub-second IGP convergence in large IP networks*. 2009 Cisco Systems, Inc. Disponible en [Internet]: http://www.cisco.com/en/US/solutions/collateral/ns341/ns524/ns610/cf-je-ccr-igp-convergence.pdf

[Gre+08] A. Greenberg, P. Lahiri, D. A Maltz, P. Patel, S. Sengupta. *Towards a Next Generation Data Center Architecture: Scalability and Commoditization*. Proceedings of Presto 2008, ACM workshop on Programmable routers for extensible services of tomorrow, co-located with Sigcomm 2008. Pages 57-62. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=1397732

[Gre+09a] A. Greenberg, J. Hamilton, D. A. Maltz, P. Patel. *The Cost of a Cloud: Research Problems in Data Center Networks*. Newsletter, ACM SIGCOMM Computer Communication Review. Volume 39 Issue 1, January 2009. Pages 68-73. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=1496103

[Gre+09b] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A Maltz, P. Patel, S. Sengupta. *VL2: A Scalable and Flexible Data Center Network*. Proceedings of Sigcomm 2009. Newsletter, ACM SIGCOMM Computer Communication Review. Volume 39 Issue 4, October 2009. Pages 51-62. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=1592576

[Guo+08] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, S. Lu. *DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers*. Proceedings of Sigcomm 2008. Disponible en [Internet]: http://research.microsoft.com/apps/pubs/default.aspx?id=75988

[Guo+09] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu. *BCube: A High Performance, Server-Centric Network Architecture for Modular Data*

Centers. Proceedings of Sigcomm 2009. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=1592577

[Guo+10] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, Y. Zhang. *SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees*. Proceedings of Co-NEXT 2010. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=1921168.1921188

[Iba+09] G. Ibáñez, A. García-Martínez, J. A. Carral, J. M. Arco, A. Azcorra. *Evaluation of Tree-based routing Ethernet*. IEEE Communication Letters, IEEE June 2009 Vol. 13 No 6 pp. 444 – 446. DOI: 10.1109/ LCOMM.2009.090469. Disponible en [Internet]: http://eprints.networks.imdea.org/91/1/Evaluation-of-Tree-based-Routing-Ethernet-2009-EN.pdf

[Iba+10a] G. Ibáñez, J. A. Carral, A. García-Martínez, J. M. Arco, D. Rivera, A. Azcorra. *Fast Path Ethernet Switching: On-demand, Efficient Transparent Bridges for Data Center and Campus Networks*. Proceedings of 17th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN). Disponible en [Internet]: http://adrian.idv.hk/doku.php/paper/icgara10-fastpath

[Iba+10b] G. Ibáñez, J. Naous, E. Rojas, D. Rivera, J. A. Carral, J. R. Velasco. *ARP-Path Bridges: Implementación de Shortest Path Bridges Ethernet basados en ARP sobre Linux y Openflow/NetFPGA.* En Proceedings de las XX Jornadas de Telecom I+D, 2010. Disponible en [Internet]: http://dspace.uah.es/dspace/bitstream/handle/10017/6771/10-TelecomID2010final8b.pdf?sequence=1

[Iba+10c] G. Ibáñez, A. García-Martínez, J. A. Carral, P. A. González, A. Azcorra, J. M. Arco. *HURP/HURBA: Zero-configuration hierarchical Up/Down routing and bridging architecture for Ethernet backbones and campus networks*. Computer Networks. Vol. 54, Issue 1, 15 January 2010, pp 41-56. Disponible en [Internet]: http://dspace.uah.es/dspace/bitstream/handle/10017/4144/COMPNW_2009%20HURBA.pdf?sequence=3

[Iba+11a] G. Ibáñez, J. A. Carral, J. M. Arco, D. Rivera, A. Montalvo. *ARP-Path: ARP-Based, Shortest Path Bridges*. Communications Letters, IEEE, July 2011, Volume 15, Pages: 770-772.

[Iba+11b] G. Ibáñez, B. De Schuymer, J. Naous, D. Rivera, E. Rojas, J. A. Carral. *Implementation of ARP-Path Low Latency Bridges in Linux and OpenFlow/NetFPGA*. Proceedings of 2011 IEEE 12th International Conference on High Performance Switching and Routing. Disponible en [Internet]: http://dspace.uah.es/dspace/bitstream/handle/10017/8677/11%20HPRS%20congreso%20cartagena.pdf?sequence=1

[InfiniBand] The InfiniBand Architecture. Disponible en [Internet]: http://www.infinibandta.org/specs

[INET] INET Framework (OMNeT++). Disponible en [Internet]: http://inet.omnetpp.org/

[JC] Jon Crowcroft. Cambridge Computer Laboratory. Faculty of Computer Science and Technology. University of Cambridge. Disponible en [Internet]: http://www.cl.cam.ac.uk/~jac22/

[JP12] M. Jarschel, R. Pries. *An OpenFlow-based energy-efficient data center approach.* Proceedings of Sigcomm 2012. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=2342373

[Kle12] P. Klein. *802.11 BSS Bridging* (Broadcom, August 2012). Disponible en [Internet]: http://ieee802.org/1/files/public/docs2012/new-phkl-11-bbs-bridging-0812-v3.pdf

[Kru56] J. B. Kruskal. *On the shortest spanning subtree of a graph and the travelling salesman problem.* En Proceedings Am. Math. Soc. 7(1) p. 48-50, Febreo 1956.

[Lei85] C. E. Leiserson. *Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing*. IEEE Transactions on Computers, Vol. 34, no. 10, Oct. 1985, pp. 892-901. Disponible en [Internet]: http://courses.csail.mit.edu/6.896/spring04/handouts/papers/fat_trees.pdf

[Lim12] T. A. Limoncelli. OpenFlow: *A Radical New Idea in Networking*. ACM Magazine. Volume 10 Issue 6, June 2012. Pages 40. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=2305856

[MAB+08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner. *OpenFlow: Enabling Innovation in Campus Networks*. Disponible en [Internet]: http://www.openflow.org//documents/openflow-wp-latest.pdf

[MEZ04] A. Myers, T.S. Eugene, H. Zhang. *Rethinking the Service Model: Scaling Ethernet to a Million Nodes*. En Proceedings of HOTNETS III. Nov. 2004

[Mininet] Mininet (OpenFlow). Disponible en [Internet]: http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet

[Moo65] Wikipedia. *Moore's law.*
Disponible en [Internet]: http://en.wikipedia.org/wiki/ Moore's_ law

[Moy98] J. Moy. *OSPF Version 2.* RFC 2328, Abril 1998. Disponible en [Internet]: http://www.ietf.org/rfc/rfc2328.txt

[MYM+11] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, Y. Pouffary. *NetLord: A Scalable Multi-Tenant Network Architecture for Virtualized Datacenters.* Proceedings of Sigcomm 2011. Disponible en [Internet]: http://dl.acm.org/citation.cfm?id=2018444

[Myrinet] Myrinet Overview (artículo [BCF+94]). Disponible en [Internet]: http://www.myricom.com/scs/myrinet/overview/

[Mys+09] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat. *PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric*. Proceedings of Sigcomm 2009. Disponible en [Internet]: http://cseweb.ucsd.edu/~vahdat/papers/portland-sigcomm09.pdf

[NECProgrammableFlow] NEC ProgrammableFlow Networking. Disponible en [Internet]: http://www.necam.com/SDN/ y también en la página de [OpenFlow]: http://www.openflow.org/wp/switch-nec/

[NECLabs] NEC Laboratories Europe. Disponible en [Internet]: http://uk.nec.com/en_GB/emea/about/neclab_eu/

[NetFPGA] NetFPGA. Disponible en [Internet]: http://netfpga.org/

[Neu81] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. Vol. 2 of Johns Hopkins Series in the Mathematical Sciences, Johns Hopkins University, Baltimore, Md, USA, 1981.

[NOXRepo] NOSRepo, home of two Open Source control platforms for Software Defined Networks (NOX and POX). Disponible en [Internet]: http://www.noxrepo.org/

[OMNeT++] OMNeT++. Disponible en [Internet]: http://www.omnetpp.org/

[OpenFlow] OpenFlow (artículos [MAB+08] y [Lim12]). Disponible en [Internet]: http://www.openflow.org/

[openSUSE] openSUSE: Linux for open minds. Disponible en [Internet]: http://www.opensuse.org/es/

[Ora90] D. Oran. *OSI IS-IS Intra-domain Routing Protocol*. RFC 1142, Febrero 1990. Disponible en [Internet]: http://tools.ietf.org/html/rfc1142

[PE11] R. Perlman and D. Eastlake. *Introduction to TRILL*. Internet Protocol Journal / The Internet Protocol Forum. 2011. Disponible en [Internet]: http://www.ipjforum.org/?p=582

[Pel+04] F. D. Pellegrini, D. Starobinski, M. G. Karpovsky, and L. B. Levitin, *Scalable cycle-breaking algorithms for gigabit Ethernet backbones.* En Proceedings of IEEE Infocom 2004, Marzo 2004

[Pep10] I. Pepelnjak. *Bridging and Routing: Is there a difference?* ("internetworking perspectives" blog). Julio 2010. Disponible en [Internet]: http://blog.ioshints.info/2010/07/bridging-and-routing-is-there.html

[Per04] R. Perlman. *RBridges: Transparent Routing.* INFOCOM 2004. Disponible en [Internet]: http://www.ieee-infocom.org/2004/Papers/26_1.PDF

[Per+11] R. Perlman et al. *Routing Bridges (RBridges): Base Protocol Specification*. RFC 6325, Julio 2011. Disponible en [Internet]: http://tools.ietf.org/html/rfc6325

[Per11] S. Pérez. Integración y prueba de RSTP en OMNeT++ e implementación de HLMAC. TFC, Universidad de Alcalá, Abril 2011

[Per85] R. Perlman. *An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN.* En Proceedings of Ninth ACM Data Communications Symposium, Vol. 20, No. 7, p. 44–52, Septiembre 1985, New York, USA.

[Pri57] R. C. Prim. *Shortest connection networks and some generalisations*. Bell Systems Technical journal. p.1389-1410, Noviembre 1957.

[PW87] D. A. Pitt, J. Winler. *Table Free Bridging*. En Proceedings IEEE JSAC SAC-5, 9. Diciembre 1987.

[RF91] B. Rajagopalan, M. Faiman. *Load sharing and shortest path routing in Transparently Interconnected LANs*. En Proceedings INFOCOM, 1991.

[Roj10] E. Rojas. *Preparación de un Escenario de Pruebas Real para un Nuevo Protocolo de Red*. En Proceedings CITTEL 2010 (15 Convención Científica de Ingeniería y Arquitectura).

[Roj12] E. Rojas, G. Ibáñez. *Torii-HLMAC: A Distributed, Faul-tolerant, Zero Configuration Fat Tree Date Center Architecture with Multiple Tree-based Addressing and Forwarding*. En Proceedings IEEE GLOBECOM 2012.

[RR10] D. Rivera, E. Rojas. *Implementación del Protocolo ARP-Path Bridges Utilizando el Simulador OMNeT++*. En Proceedings CITTEL 2010 (15 Convención Científica de Ingeniería y Arquitectura).

[RSTP] *LAN/MAN Standards Committee of the IEEE Computer Society, IEEE Standard for Local and metropolitan area networks–Common Specifications Part 3: Media Access Control (MAC) Bridges–Ammendment 2: Rapid Reconfiguration,* Junio 2001. (802.1w)
Disponible en [Internet]: http://www.ieee802.org/1/pages/802.1w.html

[RTA00] T. L. Rodeheffer, C. A. Thekkath, and D. C. Anderson, SmartBridge: A scalable bridge architecture. En Proceedings of ACM SIGCOMM 2000, Agosto 2000.
Disponible en [Internet]:
http://net.pku.edu.cn/~course/cs501/2004/readings/smartbridge.pdf

[SC88] D. Sincoskie and C. Cotton. *Extended Bridge Algorithms for Large Networks*. IEEE Network Magazine, 2(1), Enero 1988.

[SHP+11] A. Singla, C-Y. Hong, L. Popa, P. B. Godfrey. *Jellyfish: Networking Data Centers Randomly.* Proceedings of HotCloud 2011 and NSDI 2012. Disponible en [Internet]: http://www.hpl.hp.com/people/lucian_popa/jellyfish_nsdi.pdf

[Sho+91] M. Shoreder et al. *Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links*. IEEE Journal on Selected Areas in Communications, Vol. 9, No. 8, p. 1318-1335, October 1991.

[SKZ02] L. Starobinski, M.G. Karpovsky, L. Zakrevski. *Application of Network Calculus to General Topologies using Turn-Prohibition*. IEEE INFOCOM 2002 p. 1151-1159. 0-7803-7476-2/02.

[SimPy] SimPy Simulation Package. Disponible en [Internet]: http://simpy.sourceforge.net/

[STP] *IEEE 802.1D.IEEE standard for local and metropolitan area networks--Common specifications--Media access control (MAC) Bridges*. 1998.

[TP09] J. Touch and R. Perlman. *Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement*. RFC 5556, Mayo 2009. Disponible en [Internet]: http://tools.ietf.org/html/rfc5556

[Vah+10] A. Vahdat, M. Al-Fares, N. Farrington, R. N. Mysore, G. Porter, S. Radhakrishnan. *Scale-Out Netwoking in the Data Center*. Micro, IEEE (Vol. 30; Issue 4), July-Aug 2010. Disponible en [Internet]: http://nathanfarrington.com/papers/scale-out-micro10.pdf

[VLAN] *IEEE 802.1s/D15. Draft standards for Local and Metropolitan Area Networks- Virtual Bridged Local Area Networks- Amendment 3 to 802.1Q virtual bridged local area networks: Multiple Spanning Trees.*

[VLC] VideoLAN VLC Media Player. Disponible en [Internet]: http://www.videolan.org/vlc/index.html

[Wikipedia] Wikipedia Resources: http://www.wikipedia.org/

[ZM04] R. ZhangShen, N. McKeown. *Designing a Predictable Internet Backbone Network.* En Proceedings of Hotnets III. San Diego, Nov. 2004. Disponible en [Internet]: http://tiny-tera.stanford.edu/~nickm/papers/HotNetsIII.pdf