Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Implementación de un switch ARP-Path basado en el lenguaje P4 con capacidades para seguridad perimetral

Autor: Miguel Briso-Montiano Marco

Tutor: Isaías Martínez Yelmo

2018

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Implementación de un switch ARP-Path basado en el lenguaje P4 con capacidades para seguridad perimetral

Autor: Miguel Briso-Montiano Marco

Tutor/es: Isaías Martínez Yelmo

TRIBUNAL:

Presidente: José Manuel Arco Rodríguez

Vocal 1º: José Antonio Portilla Figueras

Vocal 2º: Isaías Martínez Yelmo

CALIFICACIÓN: _____

FECHA: _____

AGRADECIMIENTOS

Este trabajo se ha realizado en el contexto de una Ayuda de Iniciación en la Actividad Investigadora, concedida al autor por la Universidad de Alcalá.

Quiero dar las gracias a mi amigo David Rodero por introducirme en el desarrollo de aplicaciones ONOS. Su ayuda fue inestimable para comprender el funcionamiento de ONOS y ahorrarme mucho trabajo preliminar.

Por último, también quiero mencionar al grupo de investigación GIST-NETSERV, en el que he estado trabajando los dos cursos de Máster, que me proporcionó los recursos materiales y digitales para evaluar la implementación presentada en este TFM.

RESUMEN

En este Trabajo de Fin de Máster se presenta una aplicación práctica de las tecnologías relacionadas con el lenguaje P4, que extiende la programabilidad de las redes definidas por software a los planos de datos de los nodos de la red. Se ha diseñado una implementación a nivel local del protocolo ARP-Path en una plataforma compatible con P4 para desarrollar un switch P4 con funcionamiento autónomo, que a su vez dispone de una interfaz para un plano de control externo mediante el nuevo protocolo P4Runtime. Por último, se han desarrollado aplicaciones sobre el controlador SDN ONOS que permiten definir políticas de seguridad y recolección de estadísticas sobre el switch desarrollado.

Palabras clave: P4, SDN, software-defined networks, data center networks

ABSTRACT

In this Master's Thesis, a practical application of the technologies related to the P4 language is presented. P4 extends the programmability of the software-defined networks to the data planes of the network nodes. A local implementation of the ARP-Path protocol has been designed on a P4-compatible platform to implement an autonomous P4 switch, which in turn has an interface for an external control plane through the new P4Runtime protocol. Finally, applications have been developed on the ONOS SDN controller that allow the definition of security policies and the collection of statistics on the developed switch.

Palabras clave: P4, SDN, software-defined networks, data center networks

Contenido

1 Introducción

En los últimos años la arquitectura SDN (Redes definidas por software) ha sufrido un auge importante como alternativa a las arquitecturas de redes tradicionales. En las arquitecturas tradicionales, en cada nodo de la red estaba integrado un plano de control -lógica que establece el estado de reenvío- y un plano de datos -secuencia de operaciones que atraviesa un paquete al ser procesado-. SDN es una propuesta que sitúa el plano de control en una entidad centralizada conocida como controlador, de tal forma que sustituya los planos de control locales en los nodos de la red. El controlador posee una visión general de la red en su conjunto, y se comunica con los nodos mediante el uso de protocolos estandarizados.

En la actualidad el protocolo SDN más popular es OpenFlow, que permite definir el comportamiento de los nodos de la red con una granularidad fina. Sin embargo, OpenFlow asume un plano de datos definido en el estándar al que todo dispositivo se debe ajustar para garantizar su compatibilidad. Es decir, OpenFlow ofrece una interfaz de control abierta, pero únicamente con soporte para los protocolos y acciones definidos por el estándar, lo que supone una limitación muy importante para el desarrollo de nuevos protocolos, y en la optimización del rendimiento de las tecnologías SDN.

La evolución de SDN para convertirse en una arquitectura completablemte abierta pasa por extender la programabilidad inicialmente pensada únicamente para el plano de control, al plano de datos de los nodos de la red. En este campo, se está popularizando en los últimos años una nueva tecnología que permite la programación del plano de datos mediante un lenguaje de dominio específico para describir planos de datos de forma abierta, no ambigua, e independiente del fabricante: el lenguaje P4.

P4 es un lenguaje de alto nivel diseñado para la descripción de sistemas procesadores de paquetes, agnóstico respecto a la plataforma donde se ejecute el código, y respecto a la localización del plano de control (local o en un controlador SDN). Con P4 es posible diseñar con detalle el procesado de paquetes en un dispositivo de red: desde la desencapsulación de cabeceras, al número y tipo de tablas de búsqueda. Esto permite superar las limitaciones de OpenFlow, pudiéndose soportar en un plano de datos definido con P4 cabeceras e instrucciones de procesado arbitrarias, lo que facilita por una parte el desarrollo de nuevos protocolos y servicios, y por otra, optimizar el procesado de paquetes adecuándolo a una aplicación o protocolo específico.

La propuesta de este TFM consiste en la implementación en una plataforma reconfigurable con P4 de un switch que ejecute el protocolo ARP-Path, y que a su vez ofrezca una interfaz de control remota para su uso por un controlador SDN. De esta forma, se dispondría de un switch en el que el estado de reenvío puede ser modificado tanto a nivel interno (ARP-Path), como por una entidad externa (controlador SDN). Tanto el protocolo ARP-Path como la interfaz de control externa estarán especificadas sobre un plano de datos descrito por el lenguaje P4.

2 Objetivos

Los objetivos planteados en el desarrollo de este Trabajo de Fin de Máster se pueden resumir en los siguientes puntos:

- **Estudio del lenguaje P4 y de sus estándares asociados**: como paso previo al desarrollo de cualquier funcionalidad, es necesario disponer de un conocimento profundo de las tecnologías utilizadas, de cara a favorecer la optimización de los diseños a desarrollar.
- **Implementación local del protocolo ARP-Path en un switch reconfigurable mediante el lenguaje P4**: se plantea que la lógica del protocolo ARP-Path esté embebida dentro del propio programa P4, y que no dependa de un plano de control externo. Ello plantea un reto, ya que la lógica de control del protocolo no se puede expresar con primitivas P4 estándar, lo que obligará a emplear un enfoque novedoso a la vez que se cumple con los estándares del lenguaje.
- **Desarrollo de programas P4 que integren el protocolo ARP-Path con un plano de control externo**: se plantea un modelo de switch híbrido, que sea capaz de encaminar tráfico de forma autónoma, pero que a la vez esté integrado en un entorno SDN, de forma que un controlador SDN pueda definir el estado de reenvío de los nodos de la red.
- **Implementación de aplicaciones sobre un controlador SDN**: continuando con el punto anterior, se plantea el desarrollo de aplicaciones que permitan definir sobre el switch desarrollado políticas de encaminamiento, seguridad, y recolección de estadísticas, empleando una interfaz de comunicación estandarizada.

3 Estado del arte

3.1 Introducción a las redes definidas por software

El auge de los smartphones, el desarrollo de las tecnologías de virtualización y computación en la nube, o la necesidad de mayor ancho de banda debido a aplicaciones Big Data son algunos de los motivos que han puesto de relieve la necesidad de nuevos modelos de red adaptados a los nuevos patrones y necesidades de tráfico en Internet. La red actual tiene toda una serie de limitaciones que dificultan su adaptación a los requerimientos actuales, como, por ejemplo, la excesiva complejidad del entramado de protocolos, o la dependencia de los fabricantes de hardware a la hora de desarrollar nuevas tecnologías. Estas limitaciones propiciaron el desarrollo de un nuevo paradigma en el diseño de redes: SDN (Software Defined Networks, redes definidas por software).

SDN es una arquitectura que separa el plano de control del plano de datos, y lo sitúa en una inteligencia centralizada que posee una visión global de la red. Esta inteligencia se implementa mediante un controlador software programable con varias interfaces.

La interfaz *Southbound* es una interfaz que permite programar el comportamiento de los elementos de red (capa de infraestructura) de una forma agnóstica respecto del fabricante. La interfaz *Northbound* comunica las aplicaciones (capa de aplicación) con el controlador (capa de control) mediante una API abierta. Con esta arquitectura se facilita el desarrollo y despliegue de aplicaciones independientes de los elementos de la infraestructura de red.

En los últimos tiempos, ha surgido de forma adicional los conceptos de interfaces *East/Westbound*, que hacen referencia a la interconexión entre controladores, de forma que sea posible coordinar una política común entre varios controladores que gestionan dominios distintos.

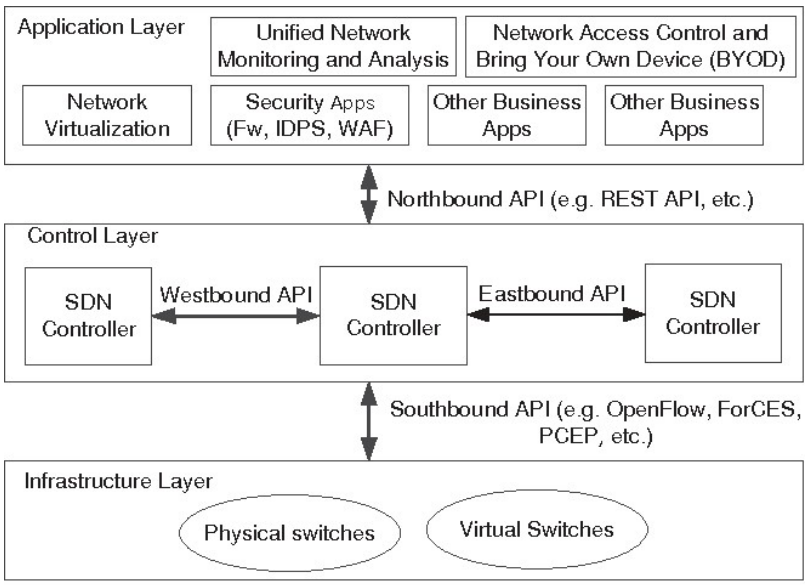


Figura 1 Arquitectura de SDN [1]

Entre las ventajas de las redes definidas por software se encuentran una gestión de la red independiente respecto a los fabricantes de hardware, mayor control sobre la red y facilidad para la innovación.

3.1.1 OpenFlow

En los últimos años se han desarrollado múltiples tecnologías basadas en SDN, pero la que goza de mayor popularidad es OpenFlow. OpenFlow [2] es un estándar SDN definido por la Open Networking Foundation, un consorcio dedicado a difundir la tecnología SDN.

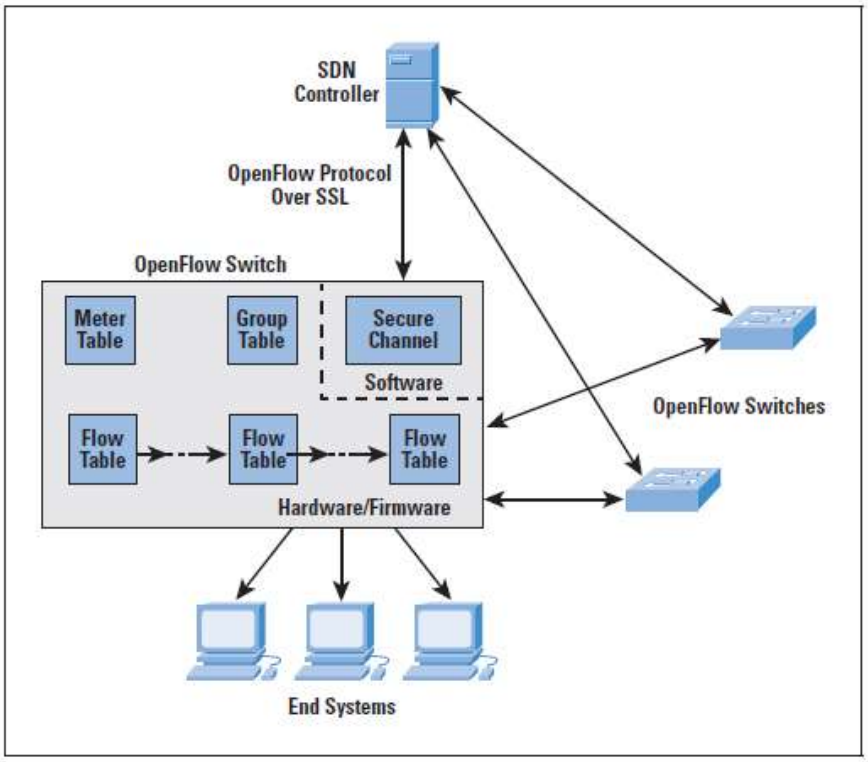


Figura 2 Arquitectura switch OpenFlow [3]

La Figura 2 muestra el modelo de red OpenFlow, que consta de un controlador conectado a switches compatibles con OpenFlow mediante conexiones TLS que transportan mensajes del protocolo OpenFlow. En el controlador reside la lógica de control de la red, que permite programar el plano de datos residente en los switches de acuerdo al encaminamiento deseado. Un switch OpenFlow contiene por una o varias *tablas de flujos (flow tables)*, cuyo contenido determina su estado de reenvío.

Por tanto, para modificar el encaminamiento de un switch, el controlador puede añadir, modificar o borrar entradas de sus tablas. Las entradas de las tablas contienen campos de correspondencia (*match fields*), que permiten seleccionar la entrada adecuada cuando sus valores coinciden con los campos de las cabeceras de los paquetes, y ejecutar las acciones asociadas. Es decir, las entradas de las tablas permiten clasificar los paquetes en flujos sobre los que aplicar su correspondiente encaminamiento. Por ello, en un contexto OpenFlow, un flujo se puede definir como una secuencia de paquetes a los que se aplica una entrada de flujo determinada.

3.1.2 Operaciones Packet-In y Packet-out

Dos operaciones habituales en arquitecturas SDN son *Packet-In* y *Packet-Out*, que implican el envío de un paquete desde el switch al controlador, y desde el controlador al switch, respectivamente. Estas operaciones suelen ejecutarse cuando el switch no sabe cómo reencaminar un paquete. Para estos casos, el controlador suele configurar que se le reenvíe el paquete mediante la operación Packet-In para examinar sus cabeceras. En base a ella, decidirá el tratamiento a aplicar (por ejemplo, reenvío por un puerto), y reenviará el paquete al switch correspondiente indicándole el tratamiento a aplicar sobre el paquete mediante la operación Packet-Out. Para poder encaminar posteriores paquetes de ese flujo sin su intervención, el controlador puede instalar las reglas de flujo correspondientes en los switches de la red.

3.1.3 Enfoques para instalación de reglas de flujo

Existen tres métodos para la provisión de reglas de flujo en los switches de la red:

- **Proactivo**: consiste en instalar las reglas para los flujos deseados en los switches correspondientes antes de que esos flujos envíen paquetes a la red.
- **Reactivo**: se instalan entradas como respuesta a un mensaje Packet-in enviado desde el switch. Dicho mensaje puede contener el paquete completo o bien una parte de él en la que se encuentren las cabeceras. El controlador puede examinarlo y decidir instalar reglas de flujo, o simplemente reencaminar el paquete mediante un mensaje Packet-out.
- **Híbrido**: Es una mezcla de los dos anteriores. Consiste en la provisión de reglas de flujo en respuesta a un Packet-In en todos los switches que compongan la ruta que debe seguir el paquete.

3.2 El lenguaje P4

P4 [4] es un lenguaje de alto nivel de descripción de sistemas procesadores de paquetes de red. El lenguaje P4 permite de definir de forma no ambigua un plano de datos o plano de reenvío, entendiendo como tal las operaciones que un procesador de paquetes ejecuta sobre un paquete desde que se recibe hasta que se reenvía, se procesa o se descarta (parseo, modificación de cabeceras, búsqueda en tablas de encaminamiento, etc). Por otra parte, el plano de control, se encarga de configurar y de establecer el estado de reenvío en un dispositivo (por ejemplo, en un router, el plano de control establece las rutas en las tablas de enrutamiento).

La creación y revisión del lenguaje P4 corresponde a la entidad P4 Language Consortium [5], compuesta por operadores de telecomunicaciones y servicios, y fabricantes de hardware. Recientemente P4 ha pasado a ser un proyecto dependiente de la ONF (Open Networking Foundation) [6], consorcio internacional destinado a promover la adopción de SDN mediante estándares, lo que contribuirá a ampliar la difusión y el uso de P4.

P4 se emplea únicamente para definir el plano de datos, quedando la interacción de éste con el plano de control fuera del estándar, aunque en el apartado 3.4 se describirá una propuesta destinada a convertirse en el estándar, P4Runtime [7]. Esto implica que el lenguaje P4 no está pensado para establecer el estado de reenvío, sino únicamente para exponer a un plano de control externo una serie de funcionalidades que éste puede configurar. Por tanto, en un programa P4 en princpio no se puede especificar la lógica de reenvío.

P4 inicialmente fue diseñado para implementar planos de datos sobre plataformas hardware o software, dando lugar a dispositivos procesadores de paquetes programables. P4 es agnóstico

respecto de la plataforma donde se implemente, haciendo que un mismo programa P4 pueda mapearse a dispositivos distintos (por ejemplo, un soft switch, o una FPGA). Para ello es preciso disponer de un compilador que genere una salida adecuada para la plataforma específica, puesto que el P4 Lanaguage Consortium sólo pretende proporcionar un compilador frontend estándar -parte del compilador que traduce el código P4 a una representación intermedia-, siendo los fabricantes u otras entidades los responsables de implementar su compilador backend -parte del compilador que transforma la representación del frontend a código máquina entendible por un dispositivo P4 . En los apartados 3.5 y 3.6 se analizarán tanto el compilador frontend estándar como algunas de las plataformas que disponen de un backend P4.

Aunque inicialmente P4 fue diseñado para programar planos de datos, últimamente ha cobrado importancia su capacidad para describir planos de datos de switches legacy de una forma unívoca. En el apartado 3.8 se describirá como se puede aprovechar la capacidad de P4 como lenguaje meramente descriptivo.

En este documento se analizarán someramente las versiones del lenguaje y se describirá brevemente su sintaxis y sus características con ejemplos de código.

3.2.1 Versiones del lenguaje

A fecha de la elaboración de este documento existen dos versiones del lenguaje: P4-14 (obsoleta, pero es la única soportada por varios backends), y P4-16. En este documento toda referencia al lenguaje P4 hará referencia a la versión P4-16, a no ser que se especifique lo contrario.

En general, las diferencias entre ambas versiones son dos:

- **Precisión en el comportamiento**: P4-16 dispone de una sintaxis más precisa para definir el comportamiento de un programa P4 respecto a P4-14.
- **Generalización**: P4-14 asumía una serie de recursos funcionales que toda plataforma debía implementar para cumplir de forma estricta con el estándar. P4-16 asume una cantidad mucho menor de funcionalidades para cumplir el estándar, dando lugar a un lenguaje mucho más generalizable para cualquier plataforma.

Sin embargo, muchas de las entidades definidas en P4-14 y eliminadas en P4-16 se han trasladado a la Portable Switch Architecture (PSA)[8].

3.2.1.1 Diferencias entre un dispositivo programable con P4 y un dispositivo tradicional

Existen fundamentalmente dos diferencias entre los dispositivos P4 con respecto a los tradicionales:

- El plano de datos no es estático, sino que queda definido por el programa P4.
- El plano de control se comunica con el plano de datos a través una API generada por el compilador P4 (Runtime API). A través de esta API, el plano de control establece el estado de reenvío en el plano de datos.

3.2.1.2 Conceptos de arquitectura, target, y programa

La especificación de P4-16 define 3 conceptos esenciales para entender cómo se aplica el lenguaje P4 a plataformas programables:

- ***Programa***: Un programa P4 representa un plano de datos específico generado a partir de código escrito en lenguaje P4, y por ello, a lo largo del documento, este término hará referencia tanto al código P4 como a la implementación del plano de datos definido por dicho código.
- ***Target***: Se define como target a toda aquella plataforma hardware o software sobre la que se pueda ejecutar un programa P4.
- ***Arquitectura:*** La arquitectura es un contrato entre el programa y el target. Específicamente, define los bloques programables y los recursos del target con interfaz P4.

Un fabricante, cuando proporcione un hardware o software programable mediante P4, debe proporcionar tanto el compilador como un fichero P4 que defina su arquitectura. Conocer la arquitectura es factor imprescindible para el desarrollo de programas P4, puesto que de ella depende el flujo del programa P4 y los recursos que se puedan emplear. P4-16 afirma que el comportamiento de un mismo programa P4, ejecutándose sobre dos targets distintos que compartan una arquitectura común y dispongan de los mismos recursos de computación y transmisión, debería ser idéntico.

La Figura 3 muestra gráficamente como se relacionan los tres conceptos anteriores:

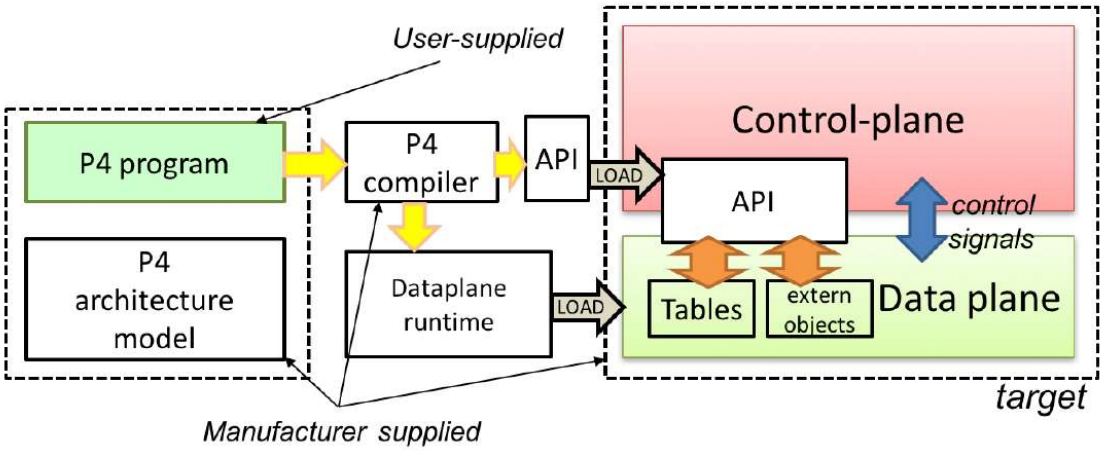


Figura 3 Proceso de despliegue de programa P4 en un target [9]

El fabricante proporciona el target, la arquitectura y el compilador. A través de un programa P4 definido por el usuario y la arquitectura, el compilador genera dos artefactos: un programa P4 compilado, y una API para interaccionar con dicho programa. Los formatos de estos objetos dependen de cada target. Para desplegar el programa P4 en un target, se carga tanto el programa P4 compilado, como la API generada en el plano de control. El estándar emplea una terminología muy genérica en este apartado, dado que la especificación es común a todo tipo de targets.

3.2.2 Sintaxis del lenguaje P4

En este apartado se revisarán los principales conceptos acerca de la sintaxis del lenguaje P4. La especificación emplea un gran porcentaje de páginas para describir en profundidad los tipos de datos, sin embargo, en este documento se omitirá la descripción de este asunto, remitiendo al lector a la especificación si se desea conocer en detalle. Basta conocer que P4 permite definir

tanto datos de tipo cadena de bits como enteros, y agruparlos en variables de tipo estructurado (similar a los structs de C).

En este apartado se revisarán las principales abstracciones incluidas en el lenguaje P4:

- ***Cabeceras***: Describen el formato de las cabeceras a procesar.
- ***Metadatos***: Modelan información asociada a la ejecución del programa para un paquete.
- ***Parsers***: Describe todas las posibles secuencias de cabeceras de un paquete soportadas por el programa, y extrae información de ellas.
- ***Acciones***: Fragmentos de código que describen la manipulación de cabeceras y metadatos de un paquete.
- ***Tablas***: Modelan una estructura generalizada para implementar tablas de búsqueda (tablas de enrutamiento, listas de control de acceso, etc…) sobre las que se implementa el estado de reenvío.
- ***Bloques de control***: Establece el flujo de control de un programa.
- ***Externs***: Objetos que encapsulan funcionalidades específicas de un target para su uso en programas P4.
- ***Anotaciones***: Proporcionan un medio para extender el lenguaje de cara a implementar información para el compilador.

El lenguaje P4 se puede dividir en tres sublenguajes:

- Un sublenguaje para la definición de arquitecturas.
- Un sublenguaje para la descripción de Parsers.
- Un sublenguaje para la descripción de bloques de control.

De los tres sublenguajes, el programador de P4 usualmente empleará los dos últimos, ya que la arquitectura la aporta el fabricante.

La especificación del lenguaje P4 define una librería estándar que contiene entidades de uso común, *core.p4*, que debe ser incluida en todo programa P4, mediante una directriz #include similar a la del lenguaje C.

3.2.2.1 Cabecera

Una cabecera es una agrupación de variables que representa la información correspondiente a la cabecera de un protocolo presente en un paquete. El formato de la cabecera es totalmente libre, siendo ésta una de las principales características de P4, y por ello permite definir tanto protocolos estándar (Ethernet, IPv4, TCP, etc.…) como protocolos específicos o de experimentación. Para especificar un tipo de cabecera se debe anteponer la palabra clave *header* al nombre que se le quiera dar, y a continuación definir cada campo con el tipo de dato y su nombre, en orden de aparición en el paquete. Un ejemplo de definición de tipo de la cabecera Ethernet es el siguiente:

```
header ethernet_t {
   bit<48> dstAddr;
   bit<48> srcAddr;
   bit<16> etherType; }
```

En este ejemplo, se define el tipo de cabecera ethernet_t, que consta de 2 campos de 48 bits (direcciones MAC destino y origen), y otro de 16 bits, el Ethernet Type. Se pueden instanciar tantas variables de tipo cabecera como se quieran, es decir, el tipo de cabecera definido anteriormente sirve para extraer todas las apariciones en un paquete de la cabecera Ethernet, correspondiéndose cada aparición con cada variable instanciada de tipo ethenet_t. En los apartados correspondientes a Parser y al Deparser se detallará como se extraen o se incluyen cabeceras en los paquetes.

En P4 existe el concepto de header stack, o pila de cabeceras. Se trata de un conjunto de cabeceras del mismo tipo organizadas en un vector, para el que se dispone de métodos específicos para añadir o eliminar cabeceras de acuerdo al modelo de pila. Un uso de esta entidad puede ser aplicable para modelar pilas de cabeceras MPLS. En este documento se describe la implementación de un programa P4 que emplea un header stack, y donde se podrá profundizar en su funcionamiento (véase el apartado 4.6).

Para finalizar este apartado, un aspecto muy relevante de las cabeceras es que disponen de un bit de validez, que indica que la variable de tipo cabecera contiene datos considerados válidos (por ejemplo, que los campos contenidos corresponden a una cabecera extraída). El bit de validez se establece como válido bien al extraer datos del paquete, o bien de forma manual. Una cabecera con el bit de validez marcado como no válido no se puede incluir en un paquete, tal como se explicará en el análisis de la entidad Deparser.

3.2.2.2 Metadato

Un metadato es una variable que representa un dato relacionado con el procesado del paquete en el programa P4, que proporciona información alternativa a la disponible en las cabeceras (por ejemplo, por donde se ha recibido el paquete, o si es un paquete copia de otro). Los metadatos pueden asociarse en variables struct (similares a C), siendo habitual que se empleen este tipo de variables para agrupar conjuntos de metadatos. Los metadatos también pueden ser de tipo cabecera, pudiéndose emplear para almacenar cabeceras de forma temporal.

Existen dos clases de metadatos: los proporcionados por el fabricante, y los definidos por el programador. En el lenguaje P4 no existe distinción alguna, pero usualmente los metadatos asociados por el fabricante suelen contener información proporcionada por el target (por ejemplo, número de elementos en cola, o puerto de entrada del paquete), o destinada para su consumo por el target (por ejemplo, para indicar el puerto de salida del paquete el programa escribe su valor en el metadato correspondiente, siendo el target el encargado de leer el valor de dicho metadato y reenviar el paquete por el puerto). En cuanto a los metadatos definidos por el programador, el número y tipo es completamente arbitrario.

Los metadatos definidos por el fabricante y la interrelación de estos con el programa P4 vienen especificados en la arquitectura. Un ejemplo de declaración de un tipo struct de metadatos es el siguiente:

```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_port; }
```

donde el fragmento de código anterior podría hacer referencia al puerto por donde se recibió el paquete (*ingress_port*), y el puerto por donde se debe reenviar el paquete (*egress_port*). Este

es un ejemplo paradigmático de metadatos proporcionados por el fabricante: el valor del metadato *ingress_port* lo establecerá el target antes de que se ejecute el programa P4 (pudiendo el programa modificar este valor), mientras que el metadato *egress_port* serviría para indicar el puerto de salida en base a la lógica del programa P4. El proceso de cómo el target lee este metadato y reenvía el paquete no es competencia del programa P4.

Por último, hay que indicar que los metadatos están asociados a una instancia de paquete, esto es, que no se pueden emplear para mantener estado entre paquetes, ya que el estándar indica que su valor debe reinicializarse con cada paquete procesado.

3.2.2.3 Parser

Un Parser es un tipo de bloque programable en el que se define la extracción de los valores de las cabeceras de un paquete (parsing, en inglés) modelada mediante una máquina de estados que representa todas las posibles combinaciones en el orden de aparición de las cabeceras de un paquete. La arquitectura define el número y la localización de este tipo de bloques programables.

Para detallar el funcionamiento de un Parser, se va a recurrir a un ejemplo ilustrativo:

Considérese una red cuyos switches P4 requieren conocer las cabeceras Ethernet, IPv4, TCP, y UDP, y que el siguiente diagrama representa las posibles transiciones entre cabeceras. El texto que aparece cerca de las flechas es la condición que se debe cumplir para que se cambie el estado:

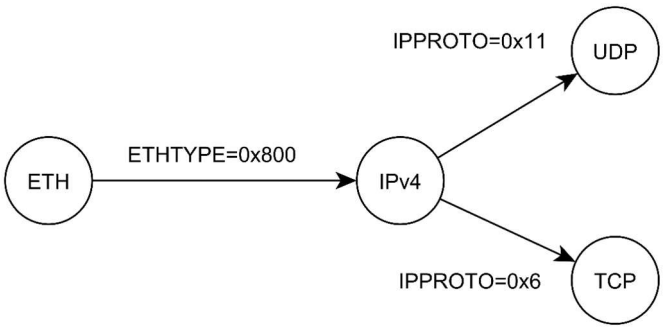


Figura 4 Diagrama de transiciones Ethernet-IPv4-TCP/UDP

Suponiendo ya definidos el tipo de cabeceras Ethernet, IPv4, UDP y TCP, declaradas cabeceras con los nombres ethernet, *ipv4*, *tcp*, y *udp*, respectivamente, y agrupadas en un struct *hdr*, se presenta a continuación la implementación de un Parser en P4. NOTA: *packet.extract()* es un método que extrae del paquete los valores indicados en la cabecera pasada como argumento.

```
state start {
    transition parse_ethernet;
}

state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        0x800: parse_ipv4;
        default: accept;
```

```
    }
}

state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition select(hdr.ipv4.ipProto) {
        0x6: parse_tcp;
        0x11: parse_udp;
        default: accept;
    }
}

state parse_tcp {
    packet.extract(hdr.tcp);
    transition accept;
}

state parse_udp {
    packet.extract(hdr.udp);
    transition accept;
}
```

El código anterior define 5 estados, uno por cada cabecera del paquete más un estado inicial, *start*, que redirige incondicionalmente al estado *parse_ethernet*, ya que se asume que el primer protocolo que aparece en un paquete es Ethernet. Asimismo, cada estado extrae los valores de su cabecera asociada, y establece el siguiente estado mediante la palabra clave *transition*. La palabra clave *accept* referencia el fin del Parser (no hay más cabeceras que extraer). Si se emplea *transition.select()*, el siguiente estado se determina entre varios posibles en base al valor de la condición del argumento. Por ejemplo, en el estado parse_ipv4 pueden darse tres transiciones posibles:

- El paquete es TCP y se extrae la cabecera TCP
- El paquete es UDP y se extrae la cabecera UDP
- El paquete no es TCP, ni UDP, por lo que termina la extracción de cabeceras

Para determinar qué transición hay que aplicar tras IPv4, se examina el campo *Protocol,* que indica el siguiente protocolo en la pila.

Como ya se ha referenciado en el apartado de cabeceras, la capacidad de P4 de definir cabeceras arbitrarias permite diseñar Parsers que procesen paquetes que posean cabeceras no estándar y/o cabeceras en un orden arbitrario, lo que supone una de las áreas con mayor potencial de P4.

3.2.2.4 Deparser

El Deparser implementa la operación opuesta al Parser: serializa sobre la carga útil del paquete las cabeceras indicadas expresamente por el programador. Para serializar una cabecera se emplea el método *Packet.emit()* de core.p4, donde el argumento es la cabecera a incluir en el paquete. Las cabeceras se incluyen antes de la carga útil, donde la carga útil en un contexto P4 hace referencia al conjunto de bytes del paquete que no han sido procesados por un Parser.

Por ejemplo, un Deparser compatible con el ejemplo de Parser presentado anteriormente puede ser el siguiente:

```
packet.emit(hdr.ethernet);
packet.emit(hdr.ipv4);
packet.emit(hdr.tcp);
packet.emit(hdr.udp);
```

Sobre el ejemplo anterior hay que realizar dos precisiones:

- Solamente se serializan las cabeceras cuyo bit de validez sea verdadero: Tal como se explicó líneas arriba, al extraer una cabecera en un Parser automáticamente se establece como verdadero el bit de validez. Dado que, en el Parser, la extracción de TCP es excluyente con la de UDP solo se serializaría una cabecera de capa 4 (sin cambios sobre el bit de validez en el programa P4). Por la misma razón, con un paquete ARP no se habrá extraído ninguna cabecera IP, ni TCP, ni UDP, por lo que solamente se serializará la cabecera Ethernet.
- El orden de serialización de las cabeceras viene definido por el orden de las llamadas a *Packet.emit()*, situando cada cabecera serializada entre la cabecera serializada anteriormente y la carga útil. Un paquete con cabeceras Ethernet, IPv4 y TCP válidas, situará primero Ethernet, después IPv4, posteriormente TCP, y por último la carga útil del paquete.

El número y localización de Deparsers vendrá definido en la arquitectura del target.

3.2.2.5 Acciones

Las acciones son un conjunto de sentencias lógicas agrupadas, que pueden tomar argumentos y retornar valores. Suponiendo una acción denominada *forward* para modificar el puerto de salida de un paquete referenciado por el metadato *egress_port* perteneciente al conjunto *standard_metadata* en base al valor pasado como argumento, su definición sería la siguiente:

```
action forward(bit<9> port) {
    standard_metadata.egress_port = port;
}
```

Un detalle recogido en la especificación es que la ejecución de las sentencias que componen una acción podría ser paralela (si el target lo soporta), aunque es posible especificar un comportamiento secuencial, como se verá en el apartado de anotaciones.

3.2.2.6 Tablas

Las tablas son las entidades sobre las que el plano de control establece el estado de reenvío del programa P4. Se trata de estructuras de datos que mapean valores definidos en cabeceras o metadatos (denominados campos de correspondencia, *match fields*) a la ejecución de una acción. El conjunto de valores de match fields se puede denominar *match key*, *lookup key*, o simplemente clave. Es decir, la match key es el índice de búsqueda en una tabla, compuesta por todos los valores de los match fields indicados para una tabla.

La asociación de una match key, y una acción se denomina entrada de tabla, table entry, o simplemente entrada. En una tabla se pueden añadir, modificar, o eliminar entradas, proporcionando una match key, y la acción asociada con sus correspondientes parámetros. En P4, cada entrada de una tabla se identifica unívocamente por su valor de match key, y, por tanto, en una tabla no pueden existir dos entradas con dos valores de match key idénticos.

La declaración de una tabla consta de 3 partes:

- Match key
- Acciones posibles
- Detalles de implementación

3.2.2.6.1 Match Key

En una tabla se pueden especificar un conjunto arbitrario de match fields, y se pueden emplear tanto campos de cabecera como metadatos. Por ejemplo, se puede especificar una tabla cuyos match fields sean la dirección MAC destino (un campo de cabecera), y el puerto de entrada del paquete (metadato). Cuando se quiere aplicar una tabla a un paquete, el programa extrae los valores instantáneos de los campos definidos en los match fields, y realiza una búsqueda comparativa entre los valores definidos en cada entrada. Si existe una coincidencia entre todos los match fields, existe una entrada y se dice que ha habido un *hit*, en caso contrario, un *miss*.

El tipo de comparaciones en la búsqueda puede ser variable y distinto para cada match field. El estándar define 3 tipos de comparaciones:

- ***Exacta***: Sólo existe coincidencia si ambos valores (el de búsqueda y el de la entrada examinada) contienen el mismo valor.
- ***Ternaria***: Cada entrada especifica un valor más una máscara expresada en bits que permite ignorar bits en la comparación
- ***LPM***: Es un caso específico de la comparación ternaria, con la diferencia de que en vez de emplear una máscara se proporciona un prefijo que indica el número de bits a comparar desde el bit de mayor peso del valor. Usualmente empleado en tablas de enrutamiento con direcciones IP.

De lo anterior se deduce que cuando se emplean match fields de tipo ternario o LPM, puede haber varias coincidencias en entradas. Para resolver este tipo de conflictos se emplean valores de prioridad para entradas con match fields ternarios, y selección de entrada con el prefijo más largo para LPM. Lógicamente, no es posible compatibilizar match fields ternarios con match fields LPM.

Cada arquitectura puede definir más tipos de comparativas de match fields. Es importante resaltar, como ya se ha dicho líneas atrás, que sólo podrá existir una coincidencia con una entrada si todos y cada uno de los match fields coinciden.

3.2.2.6.2 Acciones en tablas

Una entrada de una tabla P4 asocia un valor de match key a una acción. Esta acción no es una acción arbitraria, sino que se debe escoger de entre un conjunto de acciones permitidas, enumeradas en la definición de la tabla. A diferencia del valor de match key, la acción no tiene por qué ser única: múltiples entradas pueden ejecutar la misma acción. Si la acción fuera parametrizada, los valores de los parámetros deben estar indicados en la entrada. La selección de la acción asociada a una entrada es competencia del plano de control.

La acción por defecto es la acción que se ejecuta cuando hay un miss en una tabla P4. La especificación de acción por defecto es opcional, y en su caso, debe escogerse de entre las acciones permitidas. Si no se desea emplear ninguna acción por defecto en una tabla, en el caso de un miss no se realiza ninguna operación.

3.2.2.6.3 Propiedades adicionales

En este apartado se especifican atributos de la tabla como su tamaño (número máximo de entradas), o la acción por defecto. También se pueden especificar atributos específicos de tablas implementadas por un target concreto. Por ejemplo, será habitual la existencia de targets que soporten entradas de tablas con un tiempo de caducidad fijado, por lo que se podría emplear un atributo que permita definir si la tabla emplea entradas con caducidad.

3.2.2.6.4 Descripción del flujo de datos en una tabla

La Figura 5 muestra gráficamente los procesos de construcción de match key, búsqueda y ejecución de acción asociados a una tabla P4:

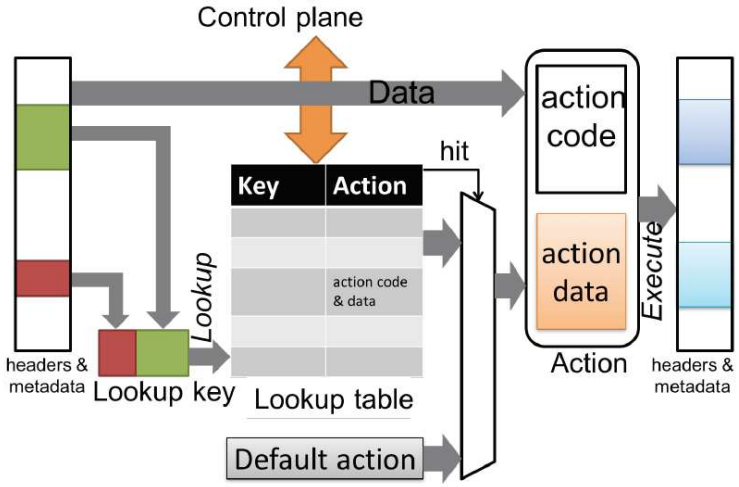


Figura 5 Flujo de datos en una tabla P4 [9]

El primer caso es construir la match key (lookup key en la imagen) a partir de los valores correspondientes, ya sean éstos campos de cabecera o metadatos. A continuación, se realiza una búsqueda en la tabla. Si existe un hit, o está definida una acción por defecto, se ejecuta el código asociado a la acción, con los posibles parámetros contenidos en la entrada. El código de la acción manipula las cabeceras y metadatos correspondientes. En este sentido, una tabla puede verse como una transformación que mapea vectores de bits (match key) a vectores de bits (modificaciones sobre cabeceras o metadatos).

3.2.2.6.5 Ejemplo

Para poder comprender mejor el concepto de tabla, se presenta un ejemplo basado en una tabla de rutas de un router IP. Una tabla de enrutamiento asocia una dirección IP con la entrada que contenga como match key el valor de dirección coincidente con el prefijo más largo, esto es, se realiza una búsqueda LPM. En un caso básico, se puede querer enrutar el paquete, se puede descartar el paquete, o no ejecutar ninguna acción (quizá pueda existir otra tabla posterior que sabrá gestionar el paquete).

Con la información anterior, la tabla de enrutamiento se puede modelar como una tabla P4 con las siguientes características:

- Match Key: Un único match field, la dirección IP destino del paquete, con tipo de comparación LPM.
- Acciones: Enrutar, tirar paquete o no realizar ninguna acción.

La definición de la tabla anterior sería la siguiente:

```
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }

    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

La acción *ipv4_forward* representaría la acción de enrutar el paquete, que a su vez realizaría las siguientes operaciones:

- Establecer el puerto de salida del paquete.
- Establecer como dirección MAC origen la MAC destino de la trama recibida.
- Establecer como dirección MAC destino del paquete la MAC del siguiente salto.
- Decrementar una unidad el TTL.

La acción *ipv4_forward* requiere de dos parámetros: el puerto de salida y la dirección MAC del siguiente salto. La implementación sería la siguiente:

```
action ipv4_forward(bit<48> dstAddr, bit<9> port) {
    standard_metadata.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

Para finalizar este apartado se presentan las siguientes características sobre las tablas, a modo de resumen:

- Las tablas son las entidades P4 que implementan el estado de reenvío de un programa P4.
- El programa P4 no puede ni insertar, ni modificar, ni borrar entradas de tablas. Dicho de otra forma, el contenido de las tablas es competencia exclusiva del plano de control, quedando fuera del estándar P4 la interacción entre el programa y el plano de control.
- El valor de la match key (como agrupación de los match fields) debe ser único en cada tabla.
- Se puede especificar el tipo de comparación a aplicar a cada match field en base a tipos de comparación definidos por el estándar o por la arquitectura.
- Solamente se puede especificar una acción por entrada, que se debe seleccionar de entre el conjunto de acciones permitidas en la tabla.

3.2.2.7 Externs

Los externs son objetos P4 específicos de cada target, diseñados para proporcionar acceso al programa P4 a funcionalidades específicas del target no implementadas con código P4, bien por limitaciones del lenguaje, bien por motivos de rendimiento. Los externs pueden ser configurados desde el plano de control, y pueden ofrecer métodos que se pueden ejecutar desde el programa P4 para su interacción.

Muchos de las entidades P4 definidas en la especificación P4-14 se han modelado como externs en P4-16 en la arquitectura PSA.

Un ejemplo de extern podría ser la definición de una entidad Checksum16, que puede calcular de un modo eficiente un CRC de 16 bits. Su definición en el fichero P4 de la arquitectura podría ser la siguiente:

```
extern Checksum16 {
    Checksum16();
    bit<16> get<D>(in D data);
}
```

En su definición aparece el constructor, y los métodos que expone al programa P4.

3.2.2.8 Bloques de control

Los bloques de control es el último tipo de bloque programable que falta por analizar. Las tablas y las llamadas directas a acciones solamente pueden existir en un bloque de control, así como las sentencias de control tipo if… else. En P4 no existen sentencias de control de tipo for, ni while. En un bucle de control pueden realizarse tantas llamadas a tablas como se deseen, y se puede controlar el flujo de ejecución entre tablas mediante las sentencias de tipo if…else, y mediante sentencias tipo hit…miss, y switch, que únicamente son aplicables a llamadas a tablas.

La sentencia tipo hit…miss, define dos bloques de código excluyentes para ejecutar dependientes del resultado de una búsqueda en tabla. Un hit en la tabla desencadenará la ejecución de un fragmento de código, y un miss ejecutará otro fragmento de código. Por otra parte, la sentencia tipo switch permite ejecutar un bloque de código seleccionado a partir de la acción de la tabla ejecutada a partir de la búsqueda. Los fragmentos de código definidos en las sentencias anteriores tienen una ejecución posterior a la acción de la tabla, e independiente de ésta (por ejemplo, un miss en una tabla desencadenará primero la ejecución de la acción por defecto, si la hubiere, y posteriormente, el código definido para el miss).

El número y localización de bloques de control vienen definidos en la arquitectura. Continuando con el ejemplo de router expuesto a lo largo de este capítulo, el bloque de control realizaría la llamada a la tabla ipv4_lpm solamente si el paquete contiene una cabecera IP, es decir, si el bit de validez de la cabecera IP es verdadero:

```
control Ingress()
{
    apply {
        if (hdr.ipv4.isValid()) {
            ipv4_lpm.apply();
```

```
        }
    }
}
```

3.2.2.9 Anotaciones

En un programa se pueden escribir anotaciones de la forma @anotación, que son directivas que tienen un amplio abanico de usos, desde limitar el uso de una acción (@defaultonly sólo permite que una acción pueda ser acción por defecto, y que no se pueda utilizar en entradas) a ocultar objetos P4 al plano de control (@hidden puede usarse para ocultar la existencia de una tabla).

3.2.2.10 Asociación de programa P4 a arquitectura

La asociación de un programa P4 a una arquitectura puede entenderse, haciendo una analogía con el lenguaje C, con los ficheros .h y .c. En los ficheros .h se declaran variables y funciones, mientras que en los ficheros .c se realiza la implementación de dichas funciones. La arquitectura sería análoga a un fichero .h, mientras que el programa P4 lo sería a un fichero .c.

Todo programa P4 que se desee implementar sobre una arquitectura debe ceñirse al número y tipo de los bloques programables definidos por ésta, y definir dichos bloques con el código deseado. Si no se desea emplear todos los bloques programables definidos por la arquitectura, puede dejarse el contenido del bloque programable en cuestión en blanco.

La definición de los bloques de control disponibles en la arquitectura suele definir argumentos parametrizables en los bloques. En estos argumentos parametrizables suelen incluirse los metadatos definidos por la arquitectura, los metadatos propios, o el propio paquete. Los argumentos parametrizables pueden tener acceso de lectura, escritura o ambos, identificados por las palabras reservadas *in*, *out*, *inout*, respectivamente.

Un ejemplo de arquitectura se muestra a continuación, limitándose a la definición de los bloques programables. La arquitectura en cuestión es V1Switch, que modela en P4-16 la arquitectura de switch estándar de P4-14. Obsérvese como cada bloque ofrece dos tipos parametrizables H, y M. H hace referencia a un struct de cabeceras, y M a un struct de metadatos. El hecho de requerir tipos parametrizables se debe a la libertad que P4 ofrece al programador de definir sus propias cabeceras y metadatos.

```
parser Parser<H, M>(packet_in b,out H parsedHdr,inout M meta,
            inout standard_metadata_t standard_metadata);

control VerifyChecksum<H, M>(in H hdr, inout M meta);

@pipeline
control Ingress<H, M>(inout H hdr,inout M meta,
              inout standard_metadata_t standard_metadata);

@pipeline
control Egress<H, M>(inout H hdr, inout M meta,
              inout standard_metadata_t standard_metadata);

control ComputeChecksum<H, M>(inout H hdr, inout M meta);

@deparser
control Deparser<H>(packet_out b, in H hdr);
```

```
package V1Switch<H, M>(Parser<H, M> p,
                       VerifyChecksum<H, M> vr,
                       Ingress<H, M> ig,
                       Egress<H, M> eg,
                       ComputeChecksum<H, M> ck,
                       Deparser<H> dep);
```

Del código anterior se extrae que la arquitectura V1Switch tiene 6 bloques programables: Parser, VerifyChecksum, Ingress, Egress, ComputeChecksum y Deparser. Un paquete atraviesa los bloques en el orden en que se han enumerado.

Por último, un programa P4 que quiera ejecutarse sobre esta arquitectura debe indicar en el propio programa la implementación de cada bloque de control:

```
V1Switch(ParserImpl(),
         VerifyChecksumImpl(),
         IngressImpl(),
         EgressImpl(),
         ComputeChecksumImpl(),
         DeparserImpl()) main;
```

3.3 Portable Switch Architecture (PSA)

La primera versión de P4-14 definía en el estándar un gran número de entidades P4 con propósitos específicos que una arquitectura que se quisiera adecuar al estándar debía implementar. Dada la diversidad de targets sobre los que se planteó un soporte de P4, y sus amplias características, el P4 Language Consortium decidió desarrollar una nueva especificación de P4 mucho más abierta, P4-16. De esta forma, los requisitos de un target para cumplir con P4-16 eran más fáciles de conseguir.

El hecho de definir el lenguaje P4 de una forma muy genérica podría contribuir a la aparición de sublenguajes adaptados a cada target. Para evitar este hecho, y contribuir a una cierta uniformidad entre los targets, el P4 Language Consortium decidió desarrollar una arquitectura generalista, pensada para ser implementada en un amplio abanico de targets, y con capacidades para adaptarse a todo tipo de aplicaciones.

Esta arquitectura se denomina Portable Switch Architecture (PSA) [8], y es un estándar del P4 Language Consortium cuya versión 1.0 se lanzó el 1 de marzo de 2018. PSA recoge la herencia de la arquitectura de switch propuesta en P4-14 y la amplía con nuevas capacidades. Muchos objetos P4 con funcionalidades específicas de P4-14 se han incluido como externs en PSA. PSA representaría el concepto de librería estándar asociada a cualquier lenguaje de programación.

En este apartado se ofrece al lector un análisis somero sobre PSA, prestando especial atención a los externs que ofrece.

3.3.1 Bloques programables y componentes de PSA

PSA define un total de 6 bloques programables con P4 y 2 bloques de funcionalidad fija (configurables desde el plano de control, pero no programables mediante código P4). Los 6 bloques programables incluyen dos parsers, dos deparsers, y dos bloques de control, Ingress y Egress. Por otra parte, los bloques de funcionalidad fija son el motor de replicación de paquetes, *Packet Buffer and Replication Engine* (PRE), y el motor de colas, *Buffer Queuing Engine* (BQE).

La Figura 6 muestra la disposición secuencia de estos bloques.

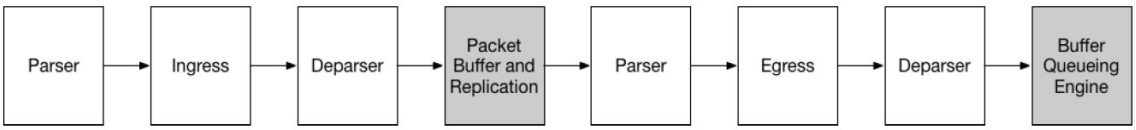


Figura 6 Arquitectura PSA [8]

El componente PRE se encarga de realizar copias del paquete saliente de los bloques de ingreso con destino a los puertos de salida configurados, es decir, permite implementar Multicast (y Broadcast como un caso específico).

Por otra parte, el componente BQE tiene como misión la configuración y planificación de las colas de salida del switch. Una aplicación indicada para ser implementada en el BQE es la planificación de colas por prioridades.

La configuración del PRE y del BQE se realiza a través del plano de control, de forma análoga al control sobre las tablas y externs definidos en P4. Los programas P4 pueden utilizar las funcionalidades del PRE y del BQE a través del uso de metadatos estándar definidos en la PSA. Por ejemplo, un metadato definido es el grupo Multicast (*multicast_group*), un identificador que se asocia a una *sesión Multicast* configurada en el PRE por el plano de control. El concepto de sesión Multicast en PSA se refiere a la asociación de un grupo Multicast con un conjunto de puertos por los que se difundirán los paquetes asociados al grupo. Así, cuando un programa P4 quiera difundir un paquete, escribirá en el metadato *multicast_group* el identificador correspondiente, de forma que cuando el paquete llegue al PRE, éste difundirá el paquete por los puertos indicados tras la lectura del metadato. Un concepto que ha salido a relucir con este ejemplo es la necesidad del programador de conocer la disposición de los bloques programables. Por la localización del PRE, se deduce que no es posible solicitar la difusión de paquetes en los bloques de egreso (salida), ya que el componente encargado de tal fin ya ha sido atravesado.

Por último, una precisión a realizar es que la PSA plantea que la determinación del destino del paquete se realice exclusivamente en el bloque de control de ingreso, y se empleen los bloques de egreso para operaciones adicionales, tales como modificaciones sobre el paquete, o recogida de estadísticas. La razón de ello es que la planificación de colas de salida muchas veces tiene en cuenta el puerto de salida al que va destinado el paquete, y el BQE requiere que no cambie durante los bloques de egreso para garantizar una planificación de colas correcta.

3.3.2 Packet Paths definidos en PSA

Un Packet Path, en el contexto de PSA, se entiende como la secuencia de bloques que atraviesa un paquete atendiendo a su tipo (unicast, multicast, clonado, etc…).

La Figura 7 representa los Packet Paths disponibles en PSA donde las abreviaciones que contienen poseen los siguientes significados:

- **NFP**: Paquete recibido por un puerto.
- **NFCPU**: Paquete recibido desde el plano de control.
- **NU**: Paquete unicast.
- **NM**: Paquete multicast (una copia para ser enviada por un puerto).
- **NTP**: Paquete destinado a ser reenviado por un puerto.
- **NTCPU**: Paquete destinado a ser enviado al plano de control.

- **CI2E**: Paquete clonado desde el bloque de control de Ingreso.
- **CE2E**: Paquete clonado desde el bloque de control de egreso.

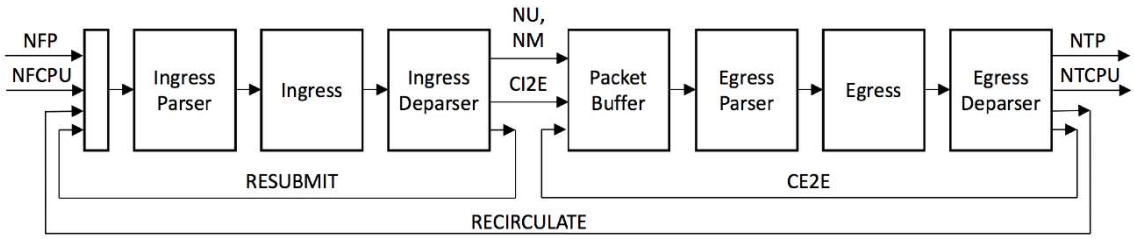


Figura 7 Packet Paths en PSA [8]

RESUBMIT y RECIRCULATE permiten implementar algoritmos de procesado iterativo en el plano de datos. Estas dos operaciones pueden emplearse para desarrollar tunelizaciones, donde cada iteración se correspondería con una capa del paquete tunelado. El máximo número de iteraciones soportadas depende de cada target.

3.3.3 Tipos, metadatos, y tipos de comparación de PSA

La especificación de PSA incluye la definición de tipos y metadatos para su uso en programas P4 que usen esta arquitectura. Cada bloque programable puede poseer su propio conjunto de metadatos.

Por otra parte, PSA incluye dos tipos más de comparación en tablas respecto a los tres contenidos en el estándar P4 (exacto, ternario y LPM):

- ***Rango***: Verifica si el valor a comparar se encuentra dentro de un rango numérico.
- ***Selector***: Especifica que el valor se utilizará para calcular un hash que permita seleccionar una acción dinámicamente (véase el extern Action Selector analizado a continuación).

3.3.4 Externs en PSA

En este apartado se van a enumerar la mayor parte de los externs que ofrece PSA para su uso en programas P4. Los externs descritos formaban parte del lenguaje P4-14, y han sido adaptados sin apenas cambios para su uso en P4-16. Los fragmentos de código P4 incluidos por cada extern se han obtenido de [8].

3.3.4.1 Hash

PSA permite aplicar distintos tipos de hashes a cabeceras o metadatos. Los tipos de hashes definidos en PSA son identidad, CRCs de 16 y 32 bits, complemento a 1 en 16 bits y una implementación propia de cada target.

```
 extern Hash<O> {
  /// Constructor
  Hash(PSA_HashAlgorithm_t algo);
  /// Varias sobrecargas para calcular un hash de acuerdo a varios
parámetros
  O get_hash<D>(in D data);
  O get_hash<T, D>(in T base, in D data, in T max);
}
```

3.3.4.2 Checksum

Emplea el mismo conjunto de operaciones que el extern Hash, pero ofrece una interfaz distinta, más orientada a comprobación y actualización de campos de verificación en cabeceras (por ejemplo, el campo Checksum definido en la cabecera IP). Existen dos implementaciones, una básica (la presentada a continuación), y otra con funcionalidades extendidas (InternetChecksum):

```
extern Checksum<W> {
  /// Constructor
  Checksum(PSA_HashAlgorithm_t hash);

  /// Limpiar ultimo resultado calculado
  void clear();

  /// Recalcular checksum para los datos indicados
  void update<T>(in T data);

  /// Obtener checksum calculado
  W    get();
}
```

3.3.4.3 Counter

El extern Counter permite contar tanto paquetes como bytes mediante su instanciación en un programa P4, y su lectura mediante el plano de control, no pudiéndose leer desde el propio programa P4. Existen dos clasificaciones:

- De paquetes o de bytes
- Directos o indirectos

Un contador directo está asociado a una entrada de una tabla concreta, y se incrementa automáticamente cuando existe un hit en la entrada. Por el contrario, un contador indirecto puede llamarse en un lugar arbitrario en el programa P4, y requiere de una llamada explícita a un método del extern para incrementar su valor.

```
extern Counter<W, S> {
  Counter(bit<32> n_counters, PSA_CounterType_t type);
  void count(in S index); }


extern DirectCounter<W> {
  DirectCounter(PSA_CounterType_t type);
  void count(); }
```

3.3.4.4 Meters

La implementación de medidor de tráfico, o meter, definida en [10] permite clasificar tráfico en tres colores (verde, amarillo, o rojo, que representan conformidad, exceso y violación, respectivamente) en función de los parámetros CIR (bps), CBS (b), PIR (bps), y PBS (b). Una aplicación habitual de la clasificación de paquetes por colores es la provisión de calidades de servicio (QoS) diferenciadas. Así, por ejemplo, un paquete clasificado como verde podría tener prioridad sobre uno amarillo, o un paquete rojo puede ser descartado como consecuencia del incumplimiento de un SLA (acuerdo de nivel de servicio). [10] define clasificaciones *color aware* y *color blind*, PSA implementa ambas mediante sobrecargas de un método del extern.

En P4, la clasificación resultante de un meter se puede emplear para cualquier funcionalidad que desee implementar el programador. Las clasificaciones de los meters en P4 son idénticas a las de los contadores: existen meters de paquetes y bytes, y meters directos e indirectos.

Los meters solamente pueden ser configurados desde el plano de control mediante los parámetros ya citados, pero pueden leerse tanto desde el plano de control como el programa P4. Es evidente que el programa P4 debe tener acceso al color de clasificación asignado por el meter a un paquete, de cara a implementar lógica dependiente de la clasificación.

```
extern Meter<S> {
  Meter(bit<32> n_meters, PSA_MeterType_t type);

  // Sobrecarga para meters color-aware
  PSA_MeterColor_t execute(in S index, in PSA_MeterColor_t color);

  // Sobrecarga para meters color-blind
  PSA_MeterColor_t execute(in S index);

}
extern DirectMeter {
  DirectMeter(PSA_MeterType_t type);

  PSA_MeterColor_t execute(in PSA_MeterColor_t color);
  PSA_MeterColor_t execute();
}
```

3.3.4.5 Registros

Un registro es un extern que implementa una variable de un tipo definido por el programador pensada para almacenar estado en el programa, a diferencia de los metadatos, que son reinicializados con cada nuevo paquete. Los registros pueden ser leídos y escritos tanto por el plano de control como por el programa P4, y pueden emplearse para un amplio abanico de usos, tanto de encaminamiento, como estadísticos (almacenamiento temporal de estadísticas para ser consumidas por el plano de control, por ejemplo), o incluso de configuración (modificar el flujo del programa en base al valor de un registro).

```
extern Register<T, S> {
  /// Sobrecargas de constructores
  Register(bit<32> size);
  Register(bit<32> size, T initial_value);

  T    read  (in S index);
  void write (in S index, in T value);

}
```

3.3.4.6 Action Profile

Los Action Profiles implementan una indirección aplicada a las acciones de las entradas de las tablas. De hecho, un Action Profile puede modelarse como un puntero a una acción, de tal forma que para una tabla, varias entradas pueden referenciar una misma acción mediante su asociación a un Action Profile. Esto cobra especial utilidad cuando existe un elevado número de entradas que ejecutan la misma acción con los mismos parámetros, puesto que con la indirección proporcionada por el Action Profile se puede almacenar una única vez en memoria una acción asociada a N entradas de tabla.

La Figura 8 muestra una comparativa entre una tabla estándar (a), y una tabla implementada mediante un Action Profile (b):

Table entry	Key (h.f: lpm)	Action spec.
t1	01001*	set_port(1)
t2	1100*	set_port(2)
t3	101*	set_port(1)

(a) Direct table.

Table entry	Key (h.f: lpm)	Member ref.
t1	01001*	m1
t2	1100*	m2
t3	101*	m1

Member ref.	Action spec.
m1	set_port(1)
m2	set_port(2)

(b) Indirect table with action profile implementation.

Figura 8 Tabla con Action Profile [8]

En la imagen se observa que, para la tabla estándar, toda entrada posee su correspondiente match key, y acción. Sin embargo, para la tabla con Action Profile, cada entrada no posee una acción, sino un puntero a un miembro del Action Profile. A su vez cada miembro del Action Profile implementa una acción con un parámetro concreto. Comparando ambos casos, se observa que en ambas tablas solamente existen dos acciones distintas, set_port(1), y set_port(2), de forma que la tabla con Action Profile puede ahorrar memoria respecto a la estándar, puesto que únicamente almacena en memoria las dos acciones que se necesitan, mientras que una tabla estándar puede tener almacenadas acciones repetidas en distintas entradas (t1 y t3).

Los Action Profiles solamente pueden ser configurados por el plano de control y no disponen de métodos para el programa P4.

```
extern ActionProfile {
  /// Constructor indicando tamaño máximo de miembros
  ActionProfile(bit<32> size);

}
```

La implementación de una tabla con Action Profiles se indica en la definición de la propia tabla, en el apartado de propiedades adicionales. Un ejemplo de definición de una tabla con Action Profiles se presenta a continuación:

```
ActionProfile(128) ap;

  table indirect {
    key = {hdr.ipv4.dst_address: exact;}
    actions = { foo; NoAction; }
    psa_implementation = ap;
  }
```

3.3.4.7 Action Selector

Un Action Selector es una extensión del Action Profile, basándose en el mismo principio de indirección, pero pudiendo apuntar a un conjunto de acciones en vez de a una sola. De esta forma una entrada puede quedar asociada a varias acciones, de entre las cuales sólo se ejecutará una de ellas cada vez que exista un hit. Para ello, el Action Selector incluye un mecanismo de

selección basado en hash para escoger de forma dinámica -en cada hit- cuál de las acciones del conjunto se ejecuta.

Un conjunto de acciones se denomina grupo. Para que una acción pueda ser incluida en un grupo, primero debe pasar a ser miembro del Action Selector, de forma análoga a los Action Profiles. A un grupo solamente pueden pertenecer acciones del mismo tipo, diferenciándose únicamente en los parámetros.

La Figura 9 representa gráficamente un Action Selector con dos grupos y tres miembros:

Table entry	Key (h.f: lpm)	Member/ Group ref.
t1	01001*	g1
t2	1100*	m2
t3	101*	g2

Group ref.	Members
g1	m1, m2
g2	m1
g3	m2

Member ref.	Action spec.
m1	set_port(1)
m2	set_port(2)

Figura 9 Action Selector [8]

Como se observa, una tabla implementada con un Action Selector puede tener entradas cuya acción apunte a un miembro, o a un grupo. El grupo g1 contiene dos miembros, por lo que para la entrada t1 se podrá ejecutar la acción apuntada por el miembro m1 o m2, según la salida del hash.

```
extern ActionSelector {
  /// Constructor, indicando tipo de algoritmo, número máximo de miembros,
y número de bits de salida del hash
  ActionSelector(PSA_HashAlgorithm_t algo, bit<32> size, bit<32>
outputWidth);
}
```

El hash se aplica al conjunto de match fields cuyo tipo de comparación sea *selector*. La match key de las tablas implementadas con Action Selector son un caso especial, puesto que únicamente se compone de los match fields con tipo de comparación distinto a *selector*. Por ello, se puede afirmar que los match fields de tipo selector no son match fields tal como se definieron en el apartado 3.2.2, aunque compartan sintaxis.

La creación de miembros y grupos en un Action Selector es competencia del plano de control.

Para clarificar lo expuesto, se muestra un ejemplo de una tabla implementada con Action Selectors:

```
control Ctrl(inout H hdr, inout M meta) {

  action foo() { meta.foo = 1; }

  ActionSelector(PSA_HashAlgorithm_t.CRC16, 128, 10) as;

  table indirect_with_selection {
    key = {
      hdr.ipv4.dst_address: exact;
```

```
      hdr.ipv4.src_address: selector;
      hdr.ipv4.protocol: selector;
    }
    actions = { foo; NoAction; }
    psa_implementation = as;
  }

  apply {
    indirect_with_selection.apply();
  }
}
```

En el ejemplo anterior, se instancia un Action Selector empleando un CRC de 16 bits, de hasta 128 entradas, y 10 bits de salida del hash. A continuación, se define una tabla con 3 match fields: la dirección IP destino, con comparación exacta, y la dirección IP origen y el protocolo como campos sobre los que se aplicará la operación CRC16 del Action Selector para seleccionar. Tal como se ha expuesto, la match key de las entradas de esta tabla será única y exclusivamente la dirección IP destino.

Una aplicación típica de los Action Selectors es ECMP/WCMP (Equal/Weighted Cost Multipath). Estos protocolos se emplean para balancear la carga ofrecida por distintos flujos de tráfico dirigidos a un mismo destino entre distintas rutas de igual coste. Es práctica habitual seleccionar la ruta en base al hash de las direcciones IP origen y destino, y el puerto y protocolos de capa 4. Para este caso, en P4, estos campos aparecerían como match fields de tipo selector, haciendo que la operación de hash del Action Selector se aplique sobre ellos.

3.3.4.8 Random

Este extern proporciona un número aleatorio en el rango especificado de acuerdo a una distribución aleatoria

```
extern Random<T> {

  /// Constructor indicando el rango en el que se quiere obtener el número
aleatorio
  Random(T min, T max);

  T read();

}
```

3.3.4.9 Packet Digest

Este extern implementa un mecanismo flexible de envío de mensajes al plano de control. El contenido del mensaje es arbitrario, y puede contener tanto campos de cabeceras como metadatos. Packet Digest se creó con la finalidad de generar notificaciones al plano de control, para comunicarle información de su interés, con requerimientos temporales configurables.

Por ejemplo, un learning switch programado en P4 puede generar una notificación mediante un digest cada vez que detecte una MAC desconocida (por ejemplo, con un miss en una tabla). El digest enviado puede contener la MAC indicada y el puerto por el que se recibió el paquete con dicha MAC origen. Con esta información, el plano de control podría insertar una entrada en una tabla de encaminamiento para asociar el reenvío de paquetes con esa MAC por ese puerto.

```
extern Digest<T> {
  Digest();
  void pack(in T data);           /// Emite datos hacia el plano de
control

}
```

3.4 P4Runtime

Los estándares P4-14 y P4-16 no contemplan la interacción del plano de control con el plano de datos implementado con el programa P4, haciendo referencia de forma abstracta al concepto de Runtime API. Por Runtime API P4-16 entiende una interfaz API de comunicación entre el plano de datos y el plano de control. Mediante la Runtime API, el plano de control puede añadir, modificar, o eliminar entradas de tablas P4, modificando de esta forma el estado de reenvío del switch, e interaccionar con los externs del programa P4 a través de la interfaz de configuración que ofrezcan al plano de control.

Por tanto, la interrelación del programa P4 con el plano de control, es algo no estandarizado en el lenguaje P4, y por ello, los primeros targets disponibles implementaban una Runtime API específica y adaptada a sus capacidades, como puede ser el caso de Thrift en BMv2 [11], que se mencionará más en detalle en el capítulo 3.6.

El hecho de disponer de Runtime APIs distintas por target dificulta enormemente la operación de una red compuesta por varios targets P4 distintos, y, por tanto, la adopción de P4, máxime cuando uno de sus objetivos es salvar las diferencias entre fabricantes. Por ello, en 2017, el P4 Language Consortium decidió crear el API Working Group, con apoyo de Google y Barefoot Networks, con el objeto de crear una Runtime API estandarizada para el control de programas P4, independientemente del target escogido.

Una de las primeras cuestiones que tuvo que justificar el API Working Group fue el descarte de OpenFlow como Runtime API estandarizada, siendo este protocolo un estándar plenamente adoptado en SDN. La justificación del descarte radica en que OpenFlow ofrece una interfaz estándar, pero a la vez estática, quedando restringido el control del dispositivo a las operaciones definidas en el estándar. Dicho de otra forma, un target puede implementar un plano de datos novedoso (haciendo uso, por ejemplo, de un protocolo con una cabecera nueva), pero su funcionalidad quedará limitada por la interfaz OpenFlow, excluyéndose gran parte de las funcionalidades que aporta P4 (cabeceras arbitrarias, externs, etc…). La modificación del núcleo OpenFlow para soportar la interacción de objetos P4 tampoco se observó como una opción viable, por lo que OpenFlow fue desechado como Runtime API (a pesar de que existió un soporte OpenFlow-P4 experimental [12] sobre el target BMv2).

La Runtime API estándar debía integrarse completamente con el lenguaje P4, con capacidades para gestionar tablas y externs, y también provisionar programas P4 dinámicamente (cambiar el programa P4 de un target sin necesidad de reiniciar el sistema). Con esos objetivos se ideó la propuesta P4Runtime [7], que implementa una Runtime API basada en gRPC [13] y Protocol Buffers [14].

En este apartado se resumirán los principales aspectos de P4Runtime y los formatos de mensajes que soporta. Para ello es preciso realizar una breve introducción a Protocol Buffers y gRPC, como tecnologías sobre las que se sustenta P4Runtime.

3.4.1 Protocol Buffers

Protocol Buffers [14], o simplemente Protobuf, es una tecnología de serialización de datos desarrollada y empleada por Google, de licencia abierta. Protobuf se emplea para serialización de datos de forma eficiente, con tamaños de 3 a 10 veces menores respecto a XML, y con una velocidad hasta 50 veces más rápida. Estas características hacen de Protobuf una tecnología de serialización apta para entornos automatizados, como microservicios, donde no se requiere

revisar la información por humanos, dado que el formato de Protobuf no es legible, a diferencia de XML o JSON.

Protobuf define un lenguaje para definir el tipo y estructura de datos a serializar. Actualmente existen dos sintaxis, proto2 (con soporte, pero en proceso de obsolescencia) y proto3, la versión actual, y la utilizada en P4Runtime. A partir de un fichero proto3 (con extensión .proto), el plugin de Protobuf autogenera una API en el lenguaje deseado (C++,C#,Java,Go, o Python) para manipular ficheros de con métodos adecuados al formato deseado.

A modo de ejemplo, se presenta un ejemplo de sintaxis en proto3:

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

El ejemplo anterior define una estructura de datos Person con dos campos obligatorios, *name* e *id*, y un campo opcional, *email*. Por cada campo especificado se incluye su tipo de datos.

Para ese formato de datos, un ejemplo de uso de la API autogenerada en C++ es el siguiente:

```
id = john.id();
name = john.name();
email = john.email();
```

, donde el tipo de variable devuelta por cada método depende del tipo especificado en proto3.

La capacidad de Protobuf para generar APIs es clave en P4Runtime para adecuar los formatos de sus mensajes a cada programa P4; en breve se analizará cómo.

3.4.2 gRPC

gRPC [13] es un framework RPC (llamadas a procedimiento remoto) diseñado para implementar microservicios de una forma muy eficiente, gracias al uso de HTTP/2 [15] y Protobuf. Inicialmente fue diseñado por Google para como un framework RPC de uso interno, pero en 2015 lo convirtió en software libre, pasando a ser un proyecto de la Cloud Native Computing Foundation [16]. gRPC está considerado a día de hoy como el framework RPC más eficiente.

gRPC posee un plugin que autogenera un servicio RPC (API cliente y servidor) a partir de una descripción del mismo en lenguaje Protobuf (fichero .proto) con algunas adiciones para indicar el tipo de comunicación (escritura, lectura, canal de comunicación unidireccional o bidireccional). Los lenguajes actualmente disponibles en gRPC son C++, Java, Python, Objective-C, Go, PHP, C#, Ruby, node.js, Android Java, PHP, y Dart.

Como ya se ha comentado, la eficiencia de gRPC se consigue por medio de dos factores:

- Se emplea Protobuf para la serialización de datos, consiguiéndose compactar la información, y, por tanto, reduciéndose el tamaño de la comunicación. gRPC emplea

Protobuf por defecto como formato de datos, aunque es posible especificar otro, como JSON.

- Las llamadas RPC se transportan sobre HTTP/2. HTTP/2 emplea la misma estructura que HTTP/1 (mensajes y cabeceras) pero con dos cambios esenciales: la codificación de los mensajes es binaria, y realiza una gestión mucho más eficiente de las conexiones TCP, al soportar la multiplexación de múltiples peticiones/respuesta sobre un mismo mensaje HTTP y conexión TCP.

P4Runtime se implementa como microservicio gRPC, y por ello, la especificación se ha definido sobre un fichero .proto, similarmente a como se implementa cualquier otro servicio gRPC. Esto implica que solamente habrá una única implementación de P4Runtime, que se corresponde con las APIs cliente y servidor autogeneradas por el plugin gRPC. Por tanto, si un fabricante quiere soportar P4Runtime sobre un target, no tiene más que autogenerar la API servidor de P4Runtime, e integrarla en su target.

gRPC implementa cifrado y autenticación mediante certificados, por lo que es apto para su despliegue industrial.

3.4.3 Características generales de P4Runtime

P4Runtime debe implementar una Runtime API que pueda adaptarse específicamente a cada programa P4, pero a su vez que posea una implementación independiente del programa P4. Para aunar estos dos objetivos aparentemente contradictorios se ha ideado un servicio independiente del programa P4, que requiere que se le proporcione información acerca de dicho programa para poder adaptarse a él y poder controlarlo.

Un detalle a tener en cuenta es que el target P4 implementa la parte de servidor de P4Runtime, mientras que el plano de control implementa la parte cliente. Es decir, en P4Runtime, al plano de control se le denominará ***cliente P4Runtime***.

P4Runtime define una serie de operaciones reducidas, que emplean unos formatos abiertos de mensajes especificados en Protobuf. Por formato abierto se entiende un formato en el que se define el número y tipo de elementos que *puede* contener. El formato exacto se determinará a partir de un fichero de datos generado por el compilador del programa P4, adicional al programa compilado, que se denomina P4Info. En P4Info se especifica la información que debe conocer P4Runtime sobre el programa P4 para interaccionar con él.

Es decir, un compilador P4 compatible con P4Runtime tendría dos salidas a partir de un programa P4:

- El programa compilado, cuyo formato es específico de cada arquitectura.
- Un fichero P4Info.

El proceso de compilación de un programa P4 se muestra en la Figura 10, donde se observa que el fichero P4Info es consumido tanto por el cliente P4Runtime como por el servidor P4Runtime. Una precisión a realizar es que la configuración en el target incluye tanto el programa P4 compilado, como el fichero P4Info para ese programa. El programa P4 compilado lo consume el target para reconfigurarse internamente, mientras que P4Info lo consume el servidor P4Runtime para implementar la API a utilizar por el cliente.

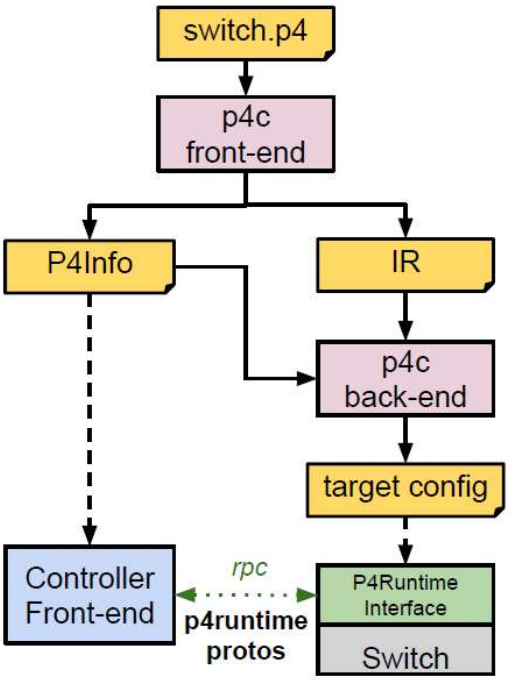


Figura 10 Generación de P4Info con el compilador p4c [17]

Más concretamente, P4Info permite al cliente P4Runtime:

- Conocer qué objetos expone el programa para ser controlados por P4Runtime. Por ejemplo, cuántas tablas posee el programa, y su tipo.
- Realizar verificaciones de formato sobre los mensajes a enviar al servidor, de acuerdo al objeto P4 concreto (por ejemplo, comprobar que la entrada a instalar para una tabla tiene el formato requerido).
- Crear una API más amigable para el programador: En P4Info, cada objeto P4 tiene asociado un identificador numérico poco amigable para el programador, dado que el servidor emplea estos identificadores para localizar el objeto P4 correspondiente. Es decir, los mensajes entre cliente y servidor no transportan los nombres de los objetos, sino sus identificadores. P4Info también incluye junto al identificador, el nombre del objeto tal como fue declarado en el programa P4. Esto permite al cliente P4Runtime implementar un servicio de traducción nombre-identificador, de forma que el programador del cliente pueda referirse por el nombre del objeto para operar con él, en vez de tener que usar un identificador.

Un ejemplo del contenido de P4Info se ilustra en la Figura 11, donde se muestra qué información contiene P4Info para una acción y una tabla. Para la acción *ipv4_forward* se incluye información relativa a los parámetros que soporta, incluyendo su longitud en bits. A su vez, para la tabla *ipv4_lpm*, P4Info indica que posee un único match field de tipo LPM y longitud 32 bits, y que posee una única acción permitida. De esta forma, el plano de control puede conocer qué recursos expone el programa P4, y qué parámetros debe utilizar para configurarlos.

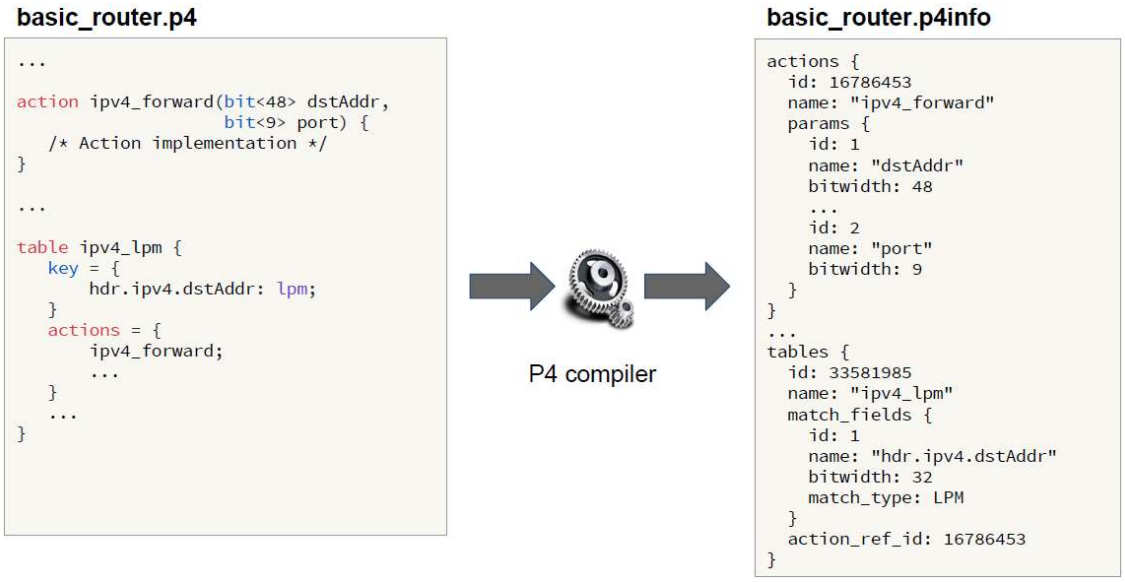


Figura 11 Mapeo de P4 a P4Info [18]

Es perfectamente posible utilizar un único plano de control para interaccionar con varios programas P4 a la vez, por ejemplo, con un controlador SDN. El controlador debe implementar un cliente P4Runtime (conexión gRPC) por cada switch P4 que desee controlar, pudiendo consumir cada cliente un fichero P4Info distinto, dependiendo del programa P4 que corra en cada switch. La Figura 12 muestra el funcionamiento de P4Runtime en una arquitectura SDN, abstrayéndose el controlador del tipo de target a través de una interfaz común P4Runtime entre dispositivos de distintos fabricantes.

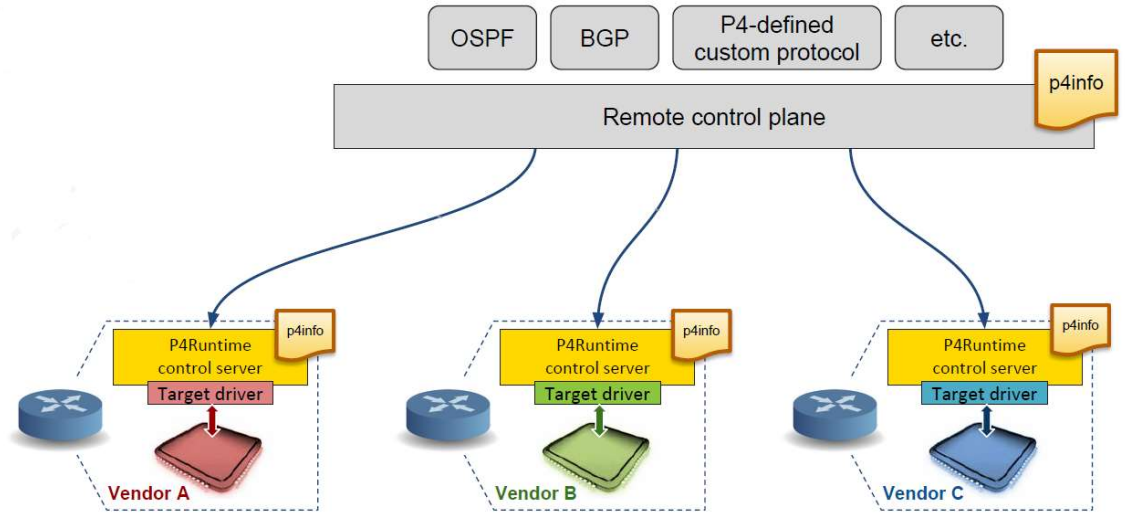


Figura 12 P4Runtime en entorno SDN [18]

P4Runtime ofrece capacidades para provisión de programas P4: el cliente tiene la capacidad de cambiar el programa P4 del target al que esté conectado. La naturaleza reconfigurable de los targets programables con P4 permite un cambio de programa P4 instantáneo desde el punto de vista humano, sin ninguna necesidad de reiniciar el dispositivo. Cuando el cliente P4Runtime

realice este tipo de operación, obviamente debe consumir un nuevo fichero P4Info, que será el correspondiente al nuevo programa P4 provisionado.

P4Runtime es agnóstico respecto a la localización del cliente. El cliente puede estar físicamente dentro del dispositivo (plano de control y plano de datos en el mismo dispositivo) o en un lugar externo (caso típico de un controlador SDN, como ya aparece en la Figura 12).

Por último, hay que reseñar que el servidor P4Runtime emplea un servicio con estado. El servidor no actúa como un mero intermediario entre la configuración interna del target, y el plano de control, sino que dispone de registros donde almacena la configuración solicitada por el plano de control, e interacciona con el programa P4 del target para implementar las operaciones solicitadas. Por tanto, en un target P4Runtime existiría por una parte el estado del programa P4 en su servidor P4Runtime, a modo de base de datos, y el estado del programa P4 mapeado a primitivas internas del target que implementan la lógica de reenvío.

3.4.4 Soporte multicliente

P4Runtime incluye soporte para gestionar escenarios con varios clientes operando con un único servidor. Esto puede ser útil en casos en los que se desee segmentar el plano de control (por ejemplo, encaminamiento, y seguridad de forma independiente) o disponer de un cliente para tomar estadísticas, por ejemplo.

Para ello, P4Runtime, define los conceptos de *rol*, *maestro* y *esclavo*. Un rol define los objetos P4 a los que tiene acceso un cliente. Por ejemplo, un cliente dedicado a implementar el encaminamiento de la red puede tener acceso a las tablas P4 relacionadas con dicho fin, pero no a tablas dedicadas a otros fines, como estadísticas o seguridad.

Cada rol posee un único maestro, que es el cliente que posee acceso de lectura/escritura sobre los objetos P4 pertenecientes a dicho rol, recepción y envío de paquetes al plano de datos (Packet-in, Packet-Out), y capacidades de provisión de programa P4. Para un rol, los clientes que no son el maestro son los esclavos, y únicamente tienen acceso de lectura sobre los objetos P4 de ese rol (pueden leer tablas o contadores, por ejemplo).

Para implementar esta funcionalidad, P4Runtime emplea dos identificadores, el *role_id* (identificador de rol) y el *election_id* (identificador indirecto de maestro). Cada role_id identifica a un rol concreto, y el election_id permite identificar al maestro dentro de un rol indicado. El maestro es el cliente con el valor más alto de election_id. Los mecanismos de asignación de identificadores para role_id, y election_id quedan fuera del ámbito de P4Runtime.

3.4.5 Operaciones disponibles en P4Runtime

Existen 5 tipos de operaciones disponibles en P4Runtime. En este punto se analizarán las funcionalidades que aportan las operaciones, dejando la revisión de formatos de mensajes para el siguiente apartado.

Las operaciones disponibles en P4Runtime se enumeran a continuación:

- Write
- Read
- SetForwardingPipelineConfig
- GetForwardingPipelineConfig
- StreamChannel

3.4.5.1 Write

Esta operación se emplea para modificar el estado de los objetos P4. Para ello, se requiere que el cliente que solicite esta operación indique su role_id, y su election_id, como forma que tiene el servidor de verificar que el cliente es el maestro, y por ello, posee acceso de escritura.

Esta operación requiere conocer el identificador de dispositivo asignado al servidor (device_id), el tipo de operación (insertar, modificar, y borrar). Además, debe incluirse la información con la que se quiera modificar el estado del dispositivo (para modificar una tabla, habría que incluir la entrada correspondiente). Por último, contiene un campo en el que se puede especificar que las operaciones solicitadas actúen como una una transacción atómica.

3.4.5.2 Read

Es la operación que se emplea para realizar lecturas sobre objetos P4. Dado que no requiere acceso de escritura, en esta operación no se especifica ni el role_id, ni el election_id. Asimismo, el cliente debe proporcionar la información que se desee consultar (el objeto P4, y otros parámetros si procede).

P4Runtime permite especificar el formato WILDCARD mediante un parámetro vacío o valor 0. Por ejemplo, WILDCARD puede emplearse para leer todas las entradas de una tabla, especificando su identificador, y dejando la match key vacía, o leer todas las tablas, dejando vacío el identificador de tabla. En el apartado correspondiente a formatos se revisará el formato de este tipo de mensajes.

3.4.5.3 Set/GetForwardingPipelineConfig

Las operaciones SetForwardingPipelineConfig y GetForwardingPipelineConfig se emplean para provisionar un programa P4 y obtener el programa P4 de un dispositivo, respectivamente. El formato del programa P4 contenido es dependiente del target al corresponderse con la salida entregada por el compilador P4.

Para ejecutar la operación SetForwardingPipelineConfig es preciso indicar los valores de role_id y election_id, al ser una operación restringida al cliente maestro. En ambas operaciones hay que especificar el identificador de dispositivo.

A la hora de provisionar un nuevo programa P4, existen varios modos de hacerlo, y un target no tiene por qué soportar todos ellos. Los modos son:

- **VERIFY**: Comprobar que el nuevo programa P4 es ejecutable en el target (verificar el programa).
- **VERIFY_AND_SAVE**: Verifica el programa, y lo almacena en el target, pero no se ejecuta, permaneciendo el antiguo programa en ejecución. Las sucesivas operaciones del cliente se realizarán sobre el nuevo programa.
- **VERIFY_AND_COMMIT**: Verifica el programa, lo almacena, e inicia su ejecución. El estado de reenvío instalado en el programa anterior se pierde.
- **COMMIT**: Ejecuta el programa almacenado previamente en una operación de tipo VERIFY_AND_SAVE.
- **RECONCILE_AND_COMMIT**: Verifica, almacena y ejecuta el programa, intentando preservar el estado de reenvío del programa anterior en el nuevo programa (si el nuevo programa incluye objetos comunes con el antiguo podría conservarse la configuración de estos).

3.4.5.4 StreamChannel

Esta operación abre un canal bidireccional (gRPC stream) entre cliente y servidor donde cada parte puede enviar mensajes de forma asíncrona respecto a la otra, y no se sigue el patrón petición-respuesta. Este canal se emplea con 3 propósitos:

- **Operación de arbitraje de maestro**: Los distintos clientes envían mensajes indicando su election_id, y el servidor selecciona como maestro al cliente con mayor election_id.
- **Operaciones Packet-In y Packet-Out**: estas operaciones se corresponden con la recepción y reenvío de paquetes desde y hacia el plano de datos del dispositivo (el programa P4 en ejecución). Estas operaciones solamente las puede realizar un cliente maestro. A los paquetes contenidos en los mensajes Packet-In/Packet-Out se les puede añadir una cabecera definida en P4, específica para estas operaciones, con las anotaciones @controller-header. En el apartado 4.4 se verá un ejemplo concreto de implementación; este tipo de cabeceras permiten indicar al plano de control detalles acerca del contexto del paquete, por ejemplo, el número de puerto por el que se recibió un Packet-In.
- **Notificaciones generadas por el extern Digest:** Permite recibir mensajes asociados a una lista de notificaciones Digest (véase el apartado 3.3.4.9). Una notificación reenviada desde el plano de datos debe ser confirmada (envío de ACK) por el plano de control, como mecanismo que tiene P4Runtime de evitar una generación continua de notificaciones.
- **Notificaciones de caducidad de entrada de tabla:** P4Runtime permite especififcar entradas con un tiempo de caducidad. El StreamChannel permite al cliente P4Runtime recibir notificaciones acerca de entradas cuyo tiempo de caducidad ha expirado, para que el plano de control tome alguna medida si lo considera oportuno (borrarla, por ejemplo).
- **Persistencia del canal**.

3.4.6 Formato de mensajes en P4Runtime

En las operaciones *Write* y *Read* es preciso especificar un mensaje que contenga información relativa a los objetos P4 a escribir o leer. Para ello se emplea un tipo de mensaje denominado *Entity*, que agrupa los formatos de un conjunto de objetos que pueden estar presentes en un programa. Las operaciones Read y Write pueden contener varias *Entities* en una única petición.

Los externs definidos en PSA son tratados como *ciudadanos de primera clase* en P4Runtime, esto es, que se incluye soporte específico para ellos sin necesidad de recurrir a entidades genéricas. Por ello, entre los mensajes que se pueden incluir en una Entity están los externs de PSA.

Una Entity incluye alguno de los mensajes listados a continuación:

- Extern
- Entrada de tabla
- Miembro de un Action Profile/Action Selector
- Grupo de un Action Selector
- Meter indirecto
- Meter directo
- Contador indirecto

- Contador directo
- Entrada de PRE (sesión multicast, por ejemplo)
- Value Set (conjunto de valores para implementar Parsers dinámicos; en experimentación)
- Registro
- Entrada de Digest (véase el extern Packet Digest de PSA en 3.3.4.9)

De los mensajes anteriores solamente se revisará el formato de las entradas de tablas, por su relevancia frente al resto de mensajes. Un mensaje *TableEntry* representa la entrada de una tabla de un programa P4. Un *TableEntry* puede contener los siguientes objetos:

- Identificador de tabla (el valor a introducir en este campo está en P4Info)
- Match fields
- Acción
- Prioridad
- Metadato del controlador: valor opaco asignado por el cliente que debe aparecer en cada lectura.
- Configuración de meter, si la tabla incluye meters directos.
- Configuración de contador, si la tabla incluye contadores directos
- Valor booleano indicando si la entrada a instalar es la entrada por defecto (al insertar la acción por defecto, no se debe incluir ningún match field).
- Tiempo en nanosegundos desde que se produjo el último hit en la entrada (presente sólo en operaciones Read)

Revisando más a fondo el formato, la lista de match fields está compuesta por un número variable de mensajes *FieldMatch*. A su vez, cada mensaje match encapsula un identificador de match field (contenido en P4Info) y un mensaje perteneciente a los tipos de comparación ya vistos. El formato de estos tipos se adapta al tipo de comparación: por ejemplo, un mensaje LPM contiene un valor y un prefijo, y un mensaje Range contiene dos valores.

Asimismo, en el valor de acción del mensaje *TableEntry* se debe incluir la acción asociada a la match key. La acción es en sí un mensaje *TableAction* que puede incluir una acción clásica, un miembro de un grupo de un *ActionProfile*, o un grupo de un *ActionProfile*. El mensaje que representa una acción en un *TableAction* incluye un identificador (contenido también en P4Info), y sus correspondientes parámetros.

La Figura 13 relaciona los mensajes anteriores de una forma gráfica:

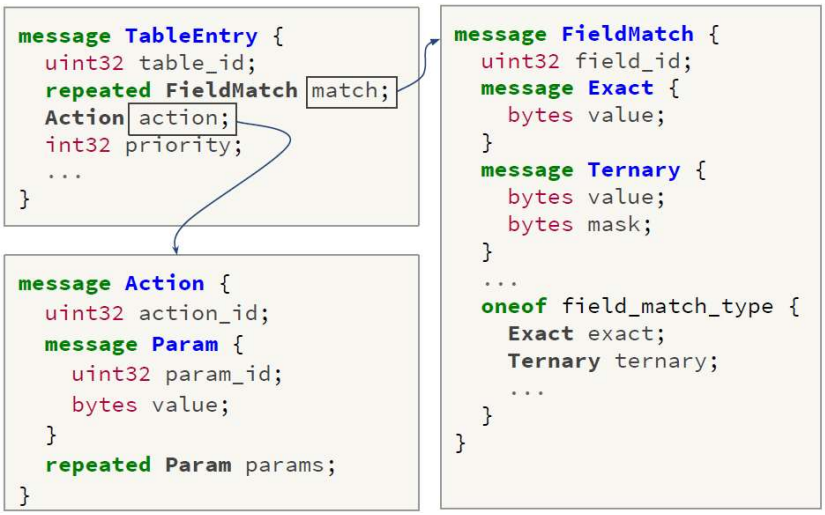


Figura 13 Representación de un mensaje TableEntry [18]

Con toda la información anterior, ya se está en disposición de entender mejor la interrelación entre el cliente P4Runtime y el fichero P4Info. Cuando el cliente quiera, por ejemplo, instalar una entrada en una tabla concreta, debe ejecutar una operación *Write* enviando un mensaje *TableEntry*. Para indicar la tabla, el cliente obtendrá los identificadores de la tabla, de cada match field, y de la acción, del P4Info. Para el caso de los match fields, incluirá una lista de objetos *FieldMatch* que contengan mensajes con el tipo de comparación soportados por la tabla concreta. Si el mensaje construido no tuviera el formato esperado por la tabla del programa, será rechazado por el servidor.

Respecto a la lectura WILDCARD a la que ya se hizo referencia, se mencionó que se solicitaba no incluyendo un campo en el mensaje o introduciendo en él un valor de 0. Si se quisieran leer todas las tablas de un servidor P4Runtime, el identificador de *TableEntry* se establecería a 0.

3.4.7 gNMI: gRPC Network Management Interface

P4Runtime no dispone de capacidades para gestionar ni la configuración ni el estado de los dispositivos de red, limitándose al control del estado de reenvío del programa P4. Pero es evidente que para una aplicación de plano de control resulta imprescindible conocer el estado del dispositivo, como, por ejemplo, el estado de las interfaces, de cara a proporcionar tolerancia frente a fallos.

En este aspecto, el de la gestión del estado y la configuración, P4Runtime convergió con otro proyecto: gNMI. gNMI (gRPC Network Management Interface) [19] es una interfaz de gestión de la configuración y del estado de dispositivos de red, impulsada por el consorcio OpenConfig [20]. gNMI emplea modelos de datos definidos por OpenConfig con el lenguaje YANG.

gNMI basa su implementación en gRPC, por lo que al compartir tecnología con P4Runtime, el API Working Group propuso integrar gNMI como un componente del estándar P4Runtime. De esta forma, un cliente que implemente el estándar P4Runtime podrá controlar el estado de reenvío con la API de P4Runtime en sí, y gestionar la configuración y el estado del dispositivo con gNMI, todo ello a través de la misma interfaz gRPC.

Los modelos de configuración YANG de OpenConfig ya son utilizados por NETCONF [21] y RESTCONF [22]. Sin embargo, gNMI tiene la ventaja de estar basado en un sistema más eficiente

(gRPC), e implementar el concepto de telemetría continua. Por telemetría continua (streaming telemetry [23]) se entiende un modelo de monitorización de red en el que los dispositivos entregan continuamente información sobre los ítems a los que se haya suscrito la entidad encargada de la monitorización de la red, sin necesidad de realizar sondeos.

gNMI define 4 tipos de operaciones:

- **Capabilities**: Permite obtener los modelos OpenConfig, codificaciones y versión de gNMI soportados.
- **Get**: Obtiene el estado de los items OpenConfig solicitados.
- **Set**: Modifica el estado de los items OpenConfig indicados.
- **Subscribe**: Permite implementar telemetría continua sobre los ítems a los que el cliente gNMI se suscriba.

Entre las operaciones anteriores, la de mayor relevancia es Subscribe. La operación Subscribe establece un canal bidireccional gRPC (del mismo tipo que el StreamChannel de P4Runtime) y posee 3 modos de operación:

- ***STREAM***: Envía inmediatamente el valor de un ítem cuando se produce un cambio.
- ***ONCE***: Envía inmediatamente el valor de un ítem cuando se produce un cambio, pero cierra el canal.
- ***POLL***: Implementa el modelo de sondeo clásico.

La integración entre un cliente/servidor P4Runtime y otro cliente/servidor gNMI puede soportarse sobre una única conexión TCP, y de hecho así lo especifica el estándar P4Runtime. Un ejemplo directo de integración entre P4Runtime y gNMI pueden ser los procedimientos de protección frente a fallos. Un plano de control que implemente sendos clientes P4Runtime y gNMI puede detectar la caída de una interfaz mediante la suscripción al ítem correspondiente (operación Subscribe) con el cliente gNMI, y actualizar el plano de reenvío del dispositivo mediante el cliente P4Runtime.

3.5 El compilador p4c

El P4 Language Consortium pretende facilitar la adopción de P4 dividiendo el concepto de compilador P4 en dos partes: un frontend estándar, p4c [24], y un backend específico de cada target.

El compilador frontend se encarga de traducir el languaje P4 a una representación intermedia (IR). Esta IR está mapeada al 100% con un programa P4, de tal forma que a partir de una IR se puede obtener el código P4 mediante un proceso de decompilación. El compilador backend es el encargado de procesar la IR, y convertirlo a una representación entendible por su target.

Los compiladores backend podrían clasificarse en dos tipos, según el tipo de target sobre el que se ejecute el programa P4:

- **De target estático**: La salida del backend es un target ejecutable que implementa la lógica del programa P4. Cada programa P4 tendría un ejecutable distinto.
- **De target dinámico**: La salida del backend es una descripción del programa P4 entendible por el target, que, al ser consumida por éste, se reconfigura internamente, adaptándose a la lógica del programa P4.

El concepto de provisión dinámica de programa P4 analizado en P4Runtime se adapta al concepto de target dinámico, pues en este tipo de targets el proceso de reconfiguración (carga del nuevo programa P4) suele ser instantáneo desde el punto de vista humano, y sin interrupción de servicio del dispositivo.

El compilador frontend p4c está siendo desarrollado bajo el marco del P4 Language Consortium. p4c se plantea que sea estándar frente a todos los targets, y ofrece un diseño modular que facilita su integración con compiladores backend.

p4c es asimismo el software encargado de generar el fichero P4Info que necesita P4Runtime. A día de hoy, p4c es un compilador en una fase de desarrollo alfa pero generalmente funcional para el desarrollo de programas P4. p4c también incluye 4 backends incluidos de serie:

- BMv2, que se analizará con detalle en el próximo apartado.
- eBPF (Extended Berkeley Packet Filter), que genera código C que se puede compilar y cargar en el kernel de Linux para implementar filtros de paquetes.
- P4test, empleado para depuración, reconstruye el programa P4 a partir de su IR.
- P4c-graphs, que genera representaciones visuales de los bloques programables de un programa P4

3.6 Revisión de targets P4

En este apartado se revisarán algunos de los principales targets programables con P4, tanto dinámicos como estáticos, centrándose en BMv2 como switch software P4 de referencia.

3.6.1 Behavioral Model versión 2 (BMv2)

Behavioral Model versión 2 [25], conocido ampliamente como BMv2, es un framework que permite implementar en C++ switches software de espacio de usuario programables con P4. BMv2 ha sido desarrollado por Barefoot Networks bajo una licencia Apache 2.0. BMv2 no es un target en sí mismo, sino que representa un conjunto de librerías C/C++ que permiten implementar todo tipo de targets, aunque al propio framework BMv2 se le considera de forma general un target en sí mismo. BMv2 tiene un propósito meramente académico o de depuración, y no ofrece un rendimiento a nivel comercial [26].

BMv2 se plantea para su uso como un simulador de arquitecturas específicas, que normalmente están implementadas en hardware, de forma que permita realizar la depuración de programas P4 sin emplear hardware real, ahorrando tiempo y costes. Su nombre, *modelo de comportamiento,* se refiere a modelar una arquitectura P4 basada en un target hardware. Aunque pueda ser tomado como un simulador de targets P4, BMv2 implementa switches en software plenamente funcionales.

BMv2 surgió a partir de un framework previo, p4c-behavioral [27]. Este target era de tipo estático, por lo que generaba un nuevo ejecutable con cada programa P4 compilado. Sin embargo, al estar planteado como target de testeo de programa P4, el propio proceso de compilación suponía un lastre para la depuración de programas P4. Este hecho animó a definir una nueva versión que fuera reconfigurable, esto es, que proporcionara un software estático e independiente del programa P4 a ejecutar, por lo que BMv2 se lanzó en junio de 2016.

Un target BMv2 compilado es un ejecutable que no posee por sí mismo ninguna capacidad de conmutar paquetes, dado que necesita dos elementos adicionales:

- La carga de un programa P4 que implemente el plano de datos deseado.
- Un proceso que implemente el plano de control de dicho plano de datos.

La forma que tiene BMv2 de implementar la lógica de un programa P4 es mediante el uso de un conjunto de clases y funciones que implementan los tipos de objetos que existen en P4, como, por ejemplo, tablas, Parsers, contadores, etc… Cada objeto P4 tiene una clase específica para soportarlo.

BMv2 consume ficheros JSON que contienen la representación del programa P4 compilado. El target BMv2 se sirve del fichero JSON para crear los objetos necesarios y entrelazarlos entre ellos para implementar la lógica descrita en el programa P4. BMv2 puede cargar programas P4 dinámicamente, sin necesidad de reiniciar el software, simplemente proporcionándole un nuevo fichero JSON, en lo que se conoce como *SWAP*. Esta operación de SWAP se estudiará en profundidad en el apartado 4.5.2.

La Figura 14 representa el despliegue de un programa P4 a partir de un fichero JSON que se corresponde con el programa P4 compilado:

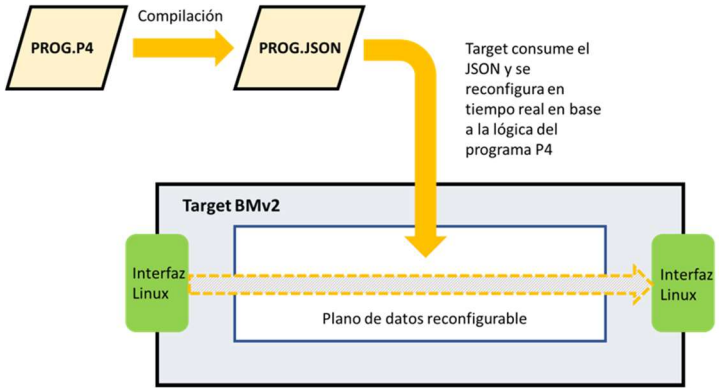


Figura 14 Despliegue de programa P4 en BMv2

Al ser desarrollado bajo la versión P4-14, BMv2 a día de hoy cuenta con soporte de muchos de los externs y componentes de PSA, y que eran parte del estándar de P4-14, como pueden ser los contadores, registros, Action Selectors o el Packet Replication Engine (PRE).

Hay que resaltar que BMv2 se diseñó antes de que se planteara la existencia de P4Runtime, y por ello implementa una Runtime API específica [11] basada en el framework RPC Thrift [28]. La Runtime API Thrift de BMv2 realiza funciones similares a las de P4Runtime, pero con formatos nativos de BMv2, siendo más sencilla por tanto (como consecuencia de ser específica para BMv2). Esta API Thrift hoy en día sigue teniendo valor, ya que BMv2 proporciona un intérprete de comandos que permite controlar programas P4 de forma manual, algo que no existe para P4Runtime. Adicionalmente muchos de los externs de PSA no disponen de la implementación en P4Runtime, por lo que se hace uso de Thrift para configurarlos hasta que la implementación de P4Runtime lo soporte.

BMv2 tiene una limitación respecto al estándar P4-16: las acciones no pueden poseer sentencias condicionales. Por tanto, a la hora de desarrollar y evaluar programas P4 que se ejecuten sobre BMv2, hay que tener en cuenta estos factores. Las restricciones específicas sobre P4 que puedan poseer los targets sobre el estándar son vistas como algo natural, y de hecho BMv2 se ha planteado como la implementación de P4 con menos restricciones.

BMv2 dispone también de un notificador de eventos, un depurador, y un generador de mensajes (funcionalidad similar al extern Packet Digest), implementados con la librería Nanomsg [29] utilizando sockets IPC (Inter-Process Communication). La información que proporciona BMv2 es muy extensa, y describe con alta precisión el flujo que sigue cada paquete dentro del programa P4.

El repositorio de BMv2 incluye 3 targets:

- **simple_router**: Es el más sencillo de los 3, y proporciona el mínimo de recursos para ejecutar un programa P4.
- **L2_switch**: Es una versión evolucionada de simple_router que incorpora el componente PRE para soportar difusión de paquetes, y un plano de control muy sencillo que implementa la lógica para un programa emulador de un learning switch.
- **Simple_switch**: Es el target estándar e implementa todas las funcionalidades de que dispone BMv2, modelado en base a la arquitectura de P4-14.

Se espera que en un futuro próximo se diseñe un nuevo target basado en PSA, que sustituya a simple_switch como el target estándar.

BMv2 posee un sistema de entrada/salida de paquetes modular, denominado *DevMgr*, y permite diseñar subsistemas I/O para recepción y envío de paquetes sin necesidad de modificar el código del framework. BMv2 dispone de 3 subsistemas de I/O de paquetes:

- BMI: Interfaz basada en la librería libpcap [30] para la recepción y el reenvío de paquetes.
- PCAP: Interfaz que permite leer y escribir paquetes en ficheros PCAP.
- Nanomsg: Interfaz que emplea sockets Nanomsg para reenvío y recepción de paquetes.

Por defecto, BMv2 emplea el subsistema BMI.

3.6.1.1 Soporte de P4Runtime en BMv2

BMv2 por defecto no incorpora soporte de P4Runtime, para lo cual se necesita instalar la librería PI [31]. PI es un framework que implementa una librería en C para proporcionar una interfaz de control de programas P4 de una forma agnóstica respecto al target. Es decir, PI podría emplearse para funcionar con otros targets distintos a BMv2, incluyendo código que enlazase PI con la API interna de cada target. BMv2 incluye un driver, *PI_imp*, que enlaza la API interna de BMv2 con las funciones estándar definidas en PI. El concepto de API interna de BMv2 hace referencia al conjunto de métodos de interacción con el programa P4 que BMv2 expone a las Runtime APIs. Es decir, tanto PI como la Runtime API Thrift internamente hacen uso de los mismos métodos.

PI incluye un servidor P4Runtime en C++, y representa actualmente la implementación de referencia de P4Runtime. El servidor P4Runtime emplea las funciones de PI para interaccionar con la API interna de BMv2 que controla el programa P4.

La Figura 15 representa un switch plenamente funcional basado en BMv2 y P4Runtime, donde se destaca que el plano de control (Cliente P4Runtime) y el plano de datos (target BMv2) son dos procesos distintos a nivel de sistema operativo, que necesitan ser instanciados para proporcionar la funcionalidad de un switch:

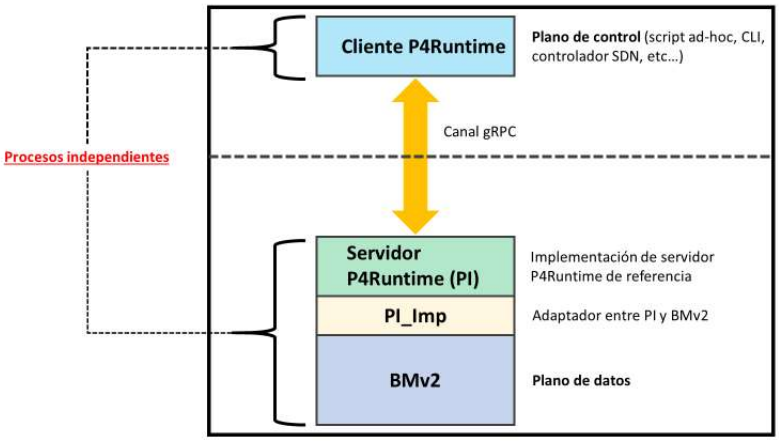


Figura 15 Integración de BMv2 con P4Runtime

PI también proporciona un soporte de gNMI, todavía en desarrollo temprano, delegando la gestión de los modelos de datos OpenConfig en el software Sysrepo [32].

Los targets por defecto enumerados de BMv2 no incluyen soporte para P4Runtime, y por ello, BMv2 incluye una versión modificada de simple_switch, denominada simple_switch_grpc, que soporta P4Runtime y gNMI.

3.6.2 Otros targets P4

3.6.2.1 Barefoot Tofino

Barefoot Tofino [33] es el nombre de un modelo de ASIC para switches Ethernet, completamente programable en P4, desarrollado por Barefoot Networks. Tofino es compatible con P4Runtime, siendo su primera implementación en hardware. Tofino está basado en una arquitectura denominada PISA (Protocol-Independent Switch Architecture). Implementa un ASIC reprogramable orientado al procesamiento de paquetes, alcanzando velocidades en su modelo más rápido de hasta 6,5 Tbps, lo que hace de él el chip reprogramable de red más rápido hasta ahora. Barefoot Tofino es un componente presente en switches desarrollados dentro de la iniciativa Open Compute Project [34], como la gama *Wedge100B*.

El compilador de Tofino está integrado en el entorno Barefoot Capilano [35], que incluye varias Runtime APIs para el control de programas P4.

3.6.2.2 Netronome Agilio Smart NIC

Los dispositivos Agilio SmartNIC de Netronome poseen soporte para el lenguaje P4 a través de la integración de un compilador P4 en el entorno de desarrollo Agilio SDK. Agilio SDK genera una representación YAML del programa P4 compilado, a partir de la cual Agilio SDK genera un programa en C para la SmartNIC.

Agilio SDK permite introducir llamadas a código C desde la sintaxis de P4. En este sentido, P4 no actuaría como un lenguaje de implementación de un plano de datos, sino más bien como una herramienta para generar un programa en C que posteriormente pueda ser modificado para incluir capacidades no soportadas en P4 (en cierto modo un enfoque análogo a los objetos externs de P4-16).

3.6.2.3 T4P4S

T4P4S [36] es un compilador P4 que tiene soporte para varios targets de tipo estático (DPDK, Freescale, Open WRT), desarrollado por la Universidad de Eötlös Loránd (Budapest). T4P4S solamente ofrece soporte parcial para la especificación P4-14, y emplea una capa de abstracción denominada NetHAL para la traducción entre la IR del programa P4, y la tecnología específica del target, entre las que destaca DPDK [37] que permite la implementación eficiente de procesamiento de paquetes en espacio de usuario.

La Runtime API de T4P4S está basada en una interfaz implementada sobre paquetes IP.

3.6.2.4 PISCES

PISCES [38] es una implementación basada en Open vSwitch (OvS) [39], el software switch de referencia. Fue desarrollado por un equipo formado por miembros de las Universidades de Princeton y Stanford, VMware, y Barefoot Networks. PISCES permite compilar versiones de OvS programables con P4.

A fecha de la escritura de este documento, el repositorio de PISCES solamente ofrece una demo que genera tráfico a altas velocidades entre dos hosts interconectados por un switch.

3.6.2.5 NetFPGA SUME

P4 es compatible con la plataforma NetFPGA [40] en su modelo SUME. NetFPGA SUME implementa la arquitectura SimpleSumeSwitch, basada en 3 bloques programables (un Parser, un único bloque de control, y un Deparser), y tres tipos de metadatos (sume, digest, y user, los dos últimos definidos por el programador). Un esquema de su arquitectura se presenta a continuación:

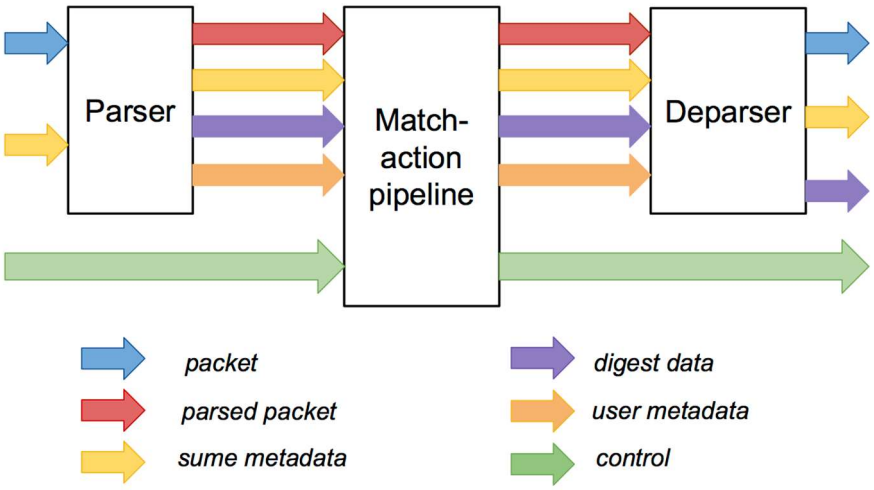


Figura 16 Arquitectura SimpleSumeSwitch [41]

El compilador para este target es Xilinx P4-SDNet, que genera un módulo HDL con la lógica del programa P4. Este módulo HDL emplea interfaces AXI-Stream para recepción y reenvío de paquetes, y una interfaz AXI-Lite para su control.

La Runtime API que posee este target es específica de cada programa P4 (cambia con cada programa), y la genera el compilador. Está basada en C, pero posee un wrapper con Python. Se espera que NetFPGA Sume ofrezca soporte para P4Runtime en un futuro.

3.7 ONOS

ONOS (Open Network Operating System) [22] es un controlador SDN de código abierto y modular, implementado en Java y diseñado para su uso en redes de proveedores de servicios. Las principales características de ONOS son su escalabilidad, su alta disponibilidad, su rendimiento y sus modelos de abstracción para el desarrollo de aplicaciones.

ONOS ha sido desarrollado por On.Lab (Open Networking Lab), y cuenta con la colaboración de importantes compañías de la industria. Además, ONOS es uno de los proyectos colaborativos de la Linux Foundation desde octubre de 2015.

Para proporcionar los requisitos de escalabilidad, disponibilidad y rendimiento necesarios en las redes de proveedores, ONOS se despliega como un servicio funcionando en un cluster de servidores. Cada servidor posee el mismo software de ONOS. Las distintas instancias de ONOS se comunican entre ellas para mostrarse ante el resto de la red y las aplicaciones como una única entidad.

3.7.1 Arquitectura de ONOS

La arquitectura de ONOS se puede subdividir en tres partes: una interfaz *Southbound* multi-protocolo que se comunica con los dispositivos de la red, un núcleo independiente de protocolos que rastrea y sirve información sobre el estado de la red, y una interfaz *Northbound* a través de la cual las aplicaciones se comunican con el núcleo. La arquitectura de ONOS está formada por unidades funcionales que se denominan subsistemas. La Figura 17 representa la arquitectura interna de ONOS:

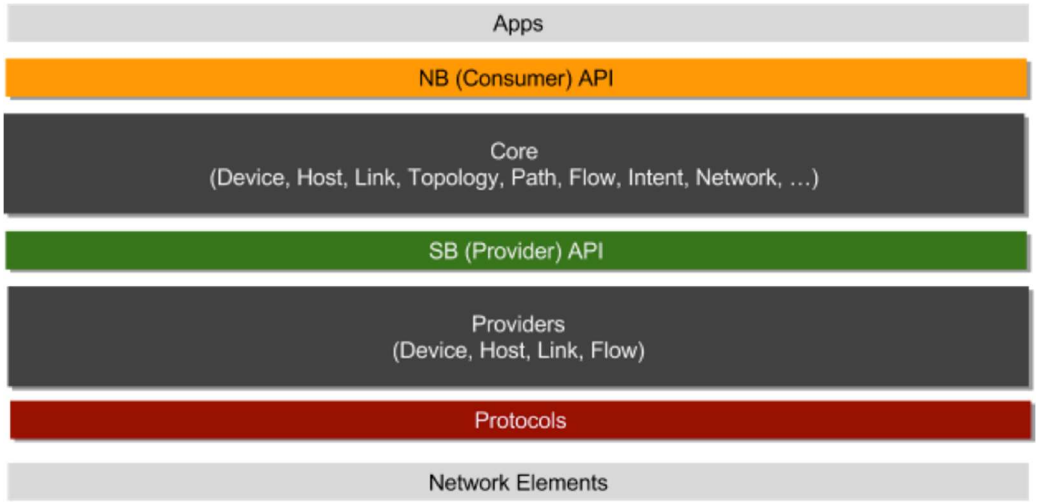


Figura 17 Arquitectura de ONOS [42]

La interfaz *Southbound* interactúa con los Providers mediante protocolos específicos adecuados al dispositivo concreto (OpenFlow, P4Runtime, REST, etc…), mientras que la interfaz *Northbound* aísla a las aplicaciones de la implementación de cada protocolo. Esto permite que se puedan crear aplicaciones totalmente desacopladas respecto de los protocolos de comunicación implementados en la capa de infraestructura. Por ejemplo, se puede diseñar una misma aplicación de encaminamiento que pueda funcionar sobre switches OpenFlow, P4Runtime, o incluso una red híbrida con switches que implementen varios protocolos de comunicación.

Refiriéndonos específicamente al concepto de tablas (tablas P4, o tablas de flujo en OpenFlow), ONOS ofrece una interfaz *Northbound* para la inserción de reglas (equivalente al concepto de entradas de tabla en P4) en tablas específicas, el servicio *Flow Rule*. A través de ese servicio, ONOS podría instalar tanto una regla en una tabla de flujo en un dispositivo OpenFlow, como una entrada en una tabla P4 en un dispositivo P4Runtime. Las aplicaciones que realizan uso directo de este servicio se denominan *Pipeline-Aware*, dado que requieren conocer la disposición interna de las tablas en los planos de datos de la capa de infraestructura (nodos de la red).

ONOS posee una API con un nivel de abstracción por encima de *Flow Rule*, el servicio *Flow Objective,* que abstrae al programador del conocimiento de la estructura interna de tablas de un dispositivo concreto. En un ejemplo basado en un P4, se puede tener un programa que incluya una tabla específica para encaminamiento, y otra tabla específica para control de acceso. Si se emplease *Flow Rule*, se está obligando a que el programador conozca la organización en tablas del dispositivo. *Flow Objetive* permite que la aplicación especifique el comportamiento que se desea del dispositivo sin hacer referencia alguna a su organización interna. Esto se consigue porque *Flow Objective* conoce la estructura del plano de datos, y se encarga de provisionar las reglas de flujo en las tablas adecuadas, mediante un componente denominado *Pipeliner*, que es específico de cada driver de dispositivo (un *Pipeliner* podría ser específico de un programa P4). Así, podría decirse que *Flow Objective* añade un nivel de abstracción más a la abstracción sobre los protocolos *Southbound* entre la capa de controlador y de infraestructura. Las aplicaciones que realizan uso de este servicio no requieren conocer la disposición de tablas del plano de datos, por lo que se denominan aplicaciones *Pipeline-Agnostic*.

La Figura 18 representa la disposición lógica de los elementos señalados:

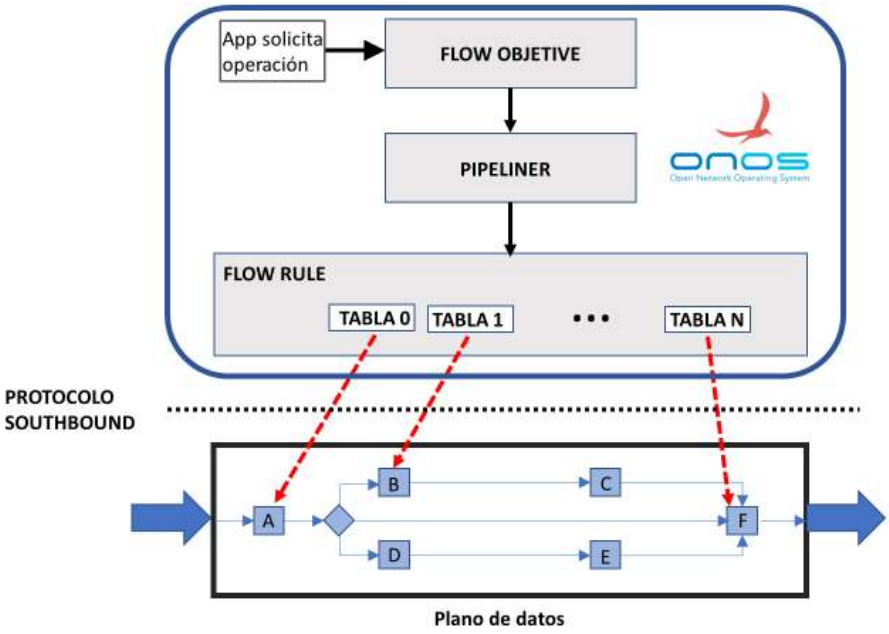


Figura 18 Interacción entre Flow Objetive y Flow Rule en ONOS

Un nivel de abstracción superior se consigue con *Intent Framework*, que permite a la aplicación definir comportamientos sobre la red sin especificar la configuración de elementos de red para definir tal política. Un ejemplo puede ser la especificación de caminos tolerantes a fallos de red.

La aplicación puede especificar la provisión de un camino entre dos hosts, encargándose ONOS de seleccionar por qué switches debe transcurrir ese camino, y en caso de caída de enlaces, reconstruir el camino de una forma transparente a la aplicación.

ONOS posee también soporte para crear aplicaciones de red externas a ONOS, que no se ejecutan como un módulo, sino que estarían compuestas por software externo que se comunica con ONOS a través de REST, o gRPC.

3.7.2 Soporte de P4Runtime en ONOS

ONOS fue el primer controlador SDN que en disponer de soporte para P4Runtime. La utilización de ONOS como plano de control permite disponer de una API mucho más amigable que la proporcionada directamente por P4Runtime (heredada de gRPC y Protobuf), así como de un amplio conjunto de aplicaciones que pueden funcionar sobre dispositivos P4 gracias al enfoque de abstracción entre las interfaces *Northbound* y *Southbound* que posee ONOS.

Sin embargo, el soporte de P4Runtime es funcional, pero todavía es parcial, ya que no se han implementado todas las funcionalidades relativas a los externs de PSA. Asimismo, ONOS tampoco posee soporte para modificar el programa P4 de un dispositivo, asumiendo ONOS que el programa P4 se provisiona tras la conexión con el dispositivo y permanece inmutable, aunque hay planes para incluirlo a medio plazo.

Los subsistemas que se han implementado en ONOS relacionados con P4Runtime se muestran en la Figura 19:

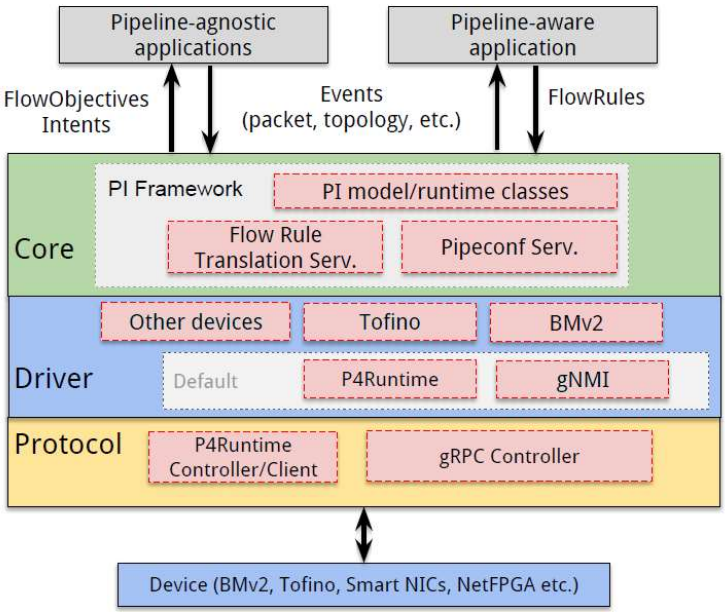


Figura 19 Subsistemas relacionados con P4Runtime en ONOS [43]

Aún con ciertas limitaciones, las principales funcionalidades de P4Runtime sí están soportadas (lectura y escritura de tablas). ONOS posee soporte para tres targets P4: BMv2, Barefoot Tofino y Mellanox Spectrum. El hecho de que ONOS no pueda interaccionar con cualquier dispositivo P4Runtime es una limitación relacionada con el formato en el que se expresa cada archivo de configuración del plano de datos, que es dependiente de cada target. En este punto hay que recordar que el formato en el que se expresa cada programa P4 compilado es distinto del dispositivo, y por ello, ONOS debe atender a estas diferencias en las operaciones de provisión

de programa P4. Por ello, cada target P4Runtime soportado en ONOS debe contar con un driver específico que combine el proveedor genérico P4Runtime con una extensión específica para tratar con el programa P4 compilado para el target (un JSON para BMv2, archivo binario para Barefoot Tofino, etc…).

Sobre P4Runtime, ONOS dispone de una API con un formato más amigable para el programador, con el nombre de *PI Framework* (*Program/Pipeline Independent*), que no tiene ninguna relación con la librería homónima vista en el apartado 3.6.1 (PI en BMv2 es una API de interacción entre el servidor P4Runtime y la API interna de BMv2).

El PI Framework es un conjunto de clases y servicios destinada a modelar planos de datos programables, actuando como una API envolvente sobre P4Runtime. El PI Framework haría de puente entre la API P4Runtime basada en gRPC y Protobuf, y el framework generalista *Flow Rule*.

Para la provisión, comprensión y control por ONOS de un programa P4 se requiere la creación de un artefacto denominado *Pipeconf*. Un Pipeconf es una aplicación ONOS (archivo .oar) que contiene el código Java necesario para poder desplegar y controlar desde ONOS un programa P4. Un Pipeconf puede registrarse y desplegarse a través del servicio de ONOS *PiPipeconfService*, haciéndolo visible para todas las instancias de ONOS (recordar el funcionamiento en cluster de ONOS).

Un Pipeconf puede incluir tres tipos de componentes:

- Lógica para la provisión de un programa P4
- Intérprete de programa P4
- Extensiones de driver

La lógica para la provisión del programa incluye la obtención del fichero P4 compilado (JSON en el caso de BMv2), el despliegue del programa en el target mediante la operación SetForwardingPipelineConfig de P4Runtime, la activación en ONOS de las extensiones que implemente el Pipeconf, y la actualización del Driver del dispositivo para que ONOS sea consciente del programa P4 desplegado.

El intérprete del programa P4 es una implementación de la interfaz *PiPipelineIntepreter*. Se trata del mapeo realizado de forma manual por el programador entre los objetos P4 definidos en el programa, y la API Northbound genérica de ONOS. En el intérprete normalmente se mapean match fields, tablas, y otros objetos P4, y se añade código para gestionar las operaciones Packet-In y Packet-Out. Para entender mejor como funciona un intérprete, se proporciona un ejemplo:

En el apartado 3.2.2 se introdujo un ejemplo de tabla de router. Esa tabla se denominaba *ipv4_forward* empleada como match key la dirección IP destino del paquete (*hdr.ipv4.dstAddr*), y especificaba una acción *forward*. Para poder introducir entradas desde ONOS empleando el framework *Flow Rule* de forma agnóstica hay que:

- Mapear el nombre de tabla P4 a un número de tabla (*TableId*)
- Mapear el match field *hdr.ipv4.dstAddr* a un criterio selector de tráfico (*Criteria*)
- Mapear la acción *forward* a las instrucciones correspondientes (*Instructions*)

El objetivo de lo anterior es, como ya se ha dicho, abstraer al programador de aplicaciones de ONOS del conocimiento del formato de los objetos del programa P4. Para ello, se le especificará

a ONOS las especificaciones a continuación cuando desee introducir una nueva entrada en la tabla de enrutamiento mediante el servicio *Flow Rule*:

- Cuando el número de tabla sea 0, la entrada se refiere a *ipv4_forward.*
- Cuando el criterio de selección de tráfico sea dirección IP destino (IPV4_DST), se debe emplear como match field *hdr.ipv4.dstAddr*.
- Cuando se especifiquen las instrucciones de establecer puerto de salida (*OUTPUT*) y reescribir MACs origen y destino (*L2Modificatio*n), se deben sustituir por la acción *forward* con los parámetros de puerto de salida y MAC destino correspodientes.

Se trata, por tanto, de exponer los recursos del programa P4 a la API Northbound de ONOS, aislando a las aplicaciones del conocimiento del plano de datos.

Por último, se pueden especificar *comportamientos* (extensiones de driver) al Pipeconf. Entre los comportamientos que se pueden añadir están la implementación de recogida periódica de estadísticas de puertos (*PortStatisticsDiscovery*) para que ONOS actualice los bytes y paquetes recibidos y reenviados por cada puerto, o un componente Pipeliner específico, que abstraiga al programador *Northbound* del número y funcionalidad de las tablas P4, y permita la utilización del framework *Flow Objetive* con un programa P4 específico.

3.8 Stratum

Stratum [44] es un proyecto de la ONF (Open Networking Foundation) destinado a implementar un Sistema Operativo en dispositivos de red que exponga una API basada en P4Runtime. Stratum es visto como el siguiente paso en la evolución de SDN hacia sistemas más abiertos y programables.

Stratum pretende desarrollar un Sistema Operativo basado en Open Network Linux [45] (distribución de Linux para switches hardware), y en interfaces P4Runtime para control del estado de reenvío, gNMI para configuración y monitorización, y gNOI (gRPC Network Operations Interface) [46] para provisión de operaciones específicas (por ejemplo, obtención de certificados de identidad). P4Runtime, gNMI, y gNOI están basados en gRPC, por lo que se implementarían mediante un único servidor. La ONF planea convertir a Stratum en un componente estándar en todos los dispositivos de red.

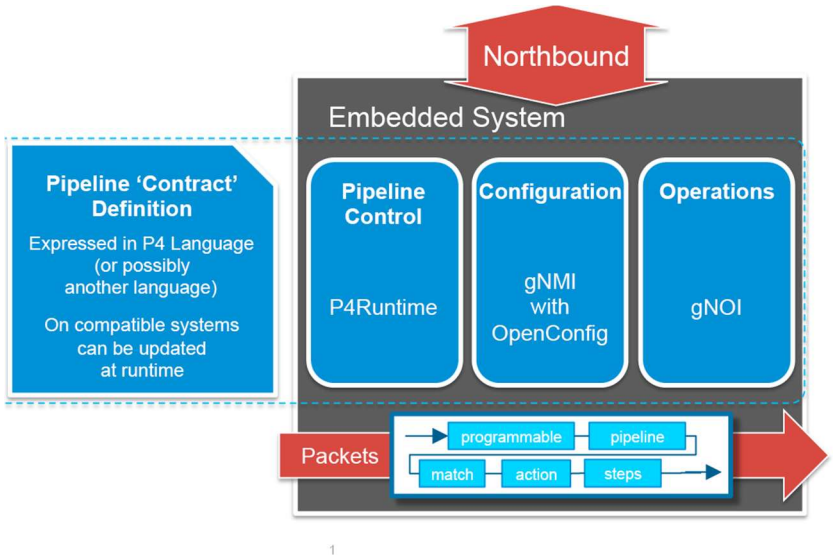


Figura 20 Arquitectura de funcionamiento de Stratum [44]

En el apartado 3.2, se habló del potencial de P4 no solo como lenguaje de programación, sino también como lenguaje de descripción de dispositivos legacy. Stratum pretende implementarse tanto en dispositivos P4 programables como en dispositivos legacy, siendo ambos tipos controlables mediante P4Runtime.

¿Cómo se puede controlar un dispositivo legacy mediante P4Runtime? Describiendo su plano de datos mediante P4. Si se puede describir de forma exacta un plano de datos de un switch legacy con P4, puede generarse una API con P4Runtime para poder controlarlo, solamente enlazando P4Runtime con la API interna del dispositivo. Lo que se consigue con P4Runtime es controlar de forma exacta, sin ningún tipo de ambigüedad, todo tipo de dispositivos.

OpenFlow, el estándar SDN actual, expone una interfaz idéntica en todo tipo de dispositivos, pero los dispositivos pueden no implementar todas las primitivas OpenFlow (un switch L2 puede no soportar MPLS), y además la disposición interna del dispositivo puede llevar a inconsistencias entre el estado de reenvío expresado en OpenFlow, y el implementado internamente.

Con P4Runtime, por el contrario, el programa P4 actúa como un contrato entre P4Runtime y el switch, de forma que no habrá comportamientos ambiguos, a la vez que se consigue que la API esté completamente adaptada al dispositivo, bien ofreciendo una API con exactamente las capacidades soportadas por el dispositivo legacy, bien porque está adaptada al programa P4 que se ejecuta en el dispositivo.

La web del proyecto Stratum especifica explícitamente cuatro casos de uso:

- Redes para Cloud Computing empleando controladores SDN propietarios.
- Switches Fabric SDN empleando Trellis [47]).
- Plataformas Cloud para redes móviles 5G empleando CORD [48].
- Thick switches (soporte para switches legacy).

Estos cuatro casos se ilustran gráficamente en la Figura 21:

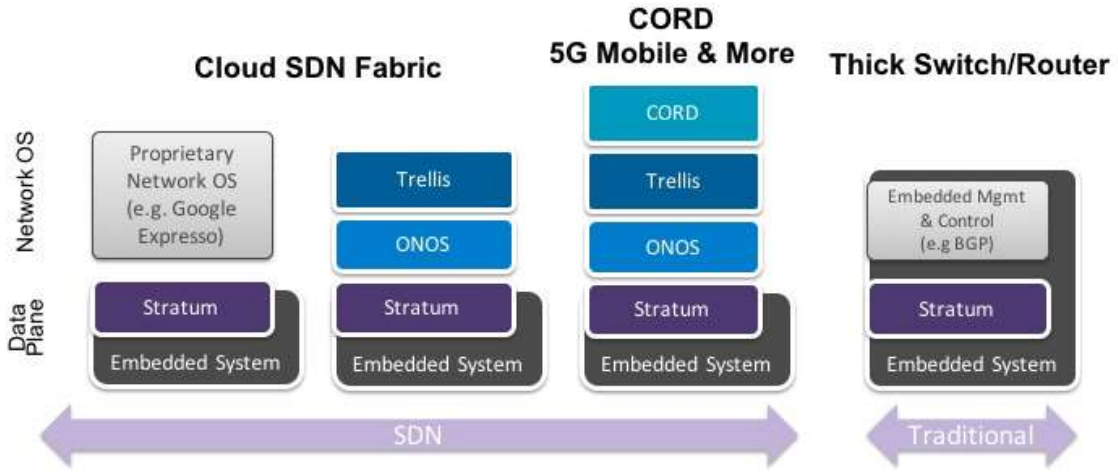


Figura 21 Casos de uso de Stratum [6]

Se pretende que Stratum esté disponible a principios de 2019.

3.9 El protocolo ARP-Path

La mayor limitación de las redes de capa 2 basadas en Ethernet es la necesidad de implementar mecanismos de eliminación de bucles mediante árboles de difusión o protocolos de encaminamiento en capa 2. Las soluciones más famosas basadas en árboles de difusión incluyen Rapid Spanning Tree Protocol (RSTP) [49] y Multiple Spanning Tree Protcol (MSTP) [50], teniendo como principal contrapartida el uso ineficiente de la red. Por otra parte, entre los protocolos de encaminamiento en capa 2 destacan *Shortest Path Bridging IEEE 802.1aq* (SPB) [51] del IEEE, y TRILL [52] del IETF. Si bien estos protocolos permiten utilizar rutas óptimas a través de mecanismos de enrutamiento análogos a los de capa 3, también introducen sobrecarga debido a los mensajes inherentes a los protocolos.

En este apartado se va a estudiar el protocolo *ARP-Path* [53], perteneciente a la familia de protocolos *All-Path*. Los protocolos All-Path están basados en la exploración simultánea de todos los posibles caminos entre un par de hosts mediante una trama de difusión, evitando bucles con el bloqueo temporal en un switch de la asociación del primer puerto de llegada de la trama con su dirección MAC origen.

ARP-Path implementa un encaminamiento en capa 2, seleccionando los caminos más cortos en base a la difusión controlada de tramas. Para ello se basa en los mensajes del protocolo ARP: *ARP-Request* y *ARP-Reply*, empleados para la resolución de una dirección MAC en base a una dirección IP. Dado que toda comunicación entre dos hosts comienza por una resolución de MAC, los caminos se podrán generar bajo demanda.

ARP-Path es una evolución de los procesos de aprendizaje de direcciones MAC en learning switches. La difusión de tramas ARP-Request se realiza mediante el bloqueo de aprendizaje de direcciones MAC, entendiendo como tal la asociación temporal de una dirección MAC al puerto por el que se recibió la primera trama con la dirección MAC origen considerada. Este mecanismo implica que las tramas con dicha dirección MAC origen recibidas con posterioridad serán descartadas, de tal forma que no se generan bucles como consecuencia de difundir únicamente aquellas recibidas por el puerto bloqueado. El hecho de bloquear el aprendizaje de una dirección MAC en base a la primera trama recibida garantiza la formación del camino de menor latencia

para un instante determinado. El bloqueo de aprendizaje de direcciones MAC incluye un temporizador cuyo tiempo de expiración sea mayor que la latencia asociada al camino del bucle más lento que se forme en la red.

Precisamente, ARP-Path incorpora un mecanismo de balanceo de carga inherente a la exploración de la red: la difusión de tramas por aquellos enlaces con mayor carga de tráfico comportará mayor latencia respecto a otros enlaces más descongestionados, de forma que los caminos generados tiendan a incluir estos últimos.

3.9.1 Procedimiento de exploración de caminos en ARP-Path

La Figura 22 muestra gráficamente el descubrimiento del camino más corto en entre dos hosts, a la vez que ilustra el proceso de prevención de bucles.

El proceso empieza en el nodo 2 que, tras recibir una trama de difusión de tipo *ARP-Request*, difunde la misma por todos sus puertos menos por el de entrada (a los nodos 1 y 3), y bloquea temporalmente (un tiempo identificado por BT) el aprendizaje de la dirección S (dirección MAC origen de la trama), asociándola al puerto de llegada. Este bloqueo temporal provoca el descarte de tramas de difusión con la MAC origen de S si se reciben por otro puerto distinto. Esto está representado en a).

Los nodos 1 y 3 habrán recibido sendas tramas de 2, por lo que realizan el mismo proceso. Para el nodo 1, la trama se difundirá hacia los nodos 3 y 4, mientras que para el nodo 3, la difusión se realizará hacia los nodos 1, 4 y 5. Tanto en el nodo 1 como el nodo 5 se habrá bloqueado el aprendizaje de la dirección S, de forma que el nodo 3 descartará la trama recibida desde 1, e igualmente 1 descartará la trama recibida por 3, evitando de esta forma la creación de un bucle.

Por último, en c), el nodo 5 difundirá la trama hacia el nodo 4 y hacia el host D, alcanzando el ARP-Request su destino (el host cuya MAC solicita S), de forma que el camino 5-3-2 constituye el camino más corto desde D hacia S. El nodo 4 difundirá copias de la trama hacia 3 y hacia 5, descartándose en ambos casos. Asimismo 5 recibirá una trama proveniente de 4 que será descartada igualmente. Nótese cómo esta trama identificaría un camino alternativo 5-4-1-2, que es de mayor latencia respecto a 5-3-2.

Es de resaltar que el descubrimiento de camino en ARP-Path guarda semejanzas con el concepto de Reverse Path Forwarding [54], ya que en un switch, el camino para alcanzar un host destino queda determinado por el puerto donde se recibe una trama procedente de ese mismo host.

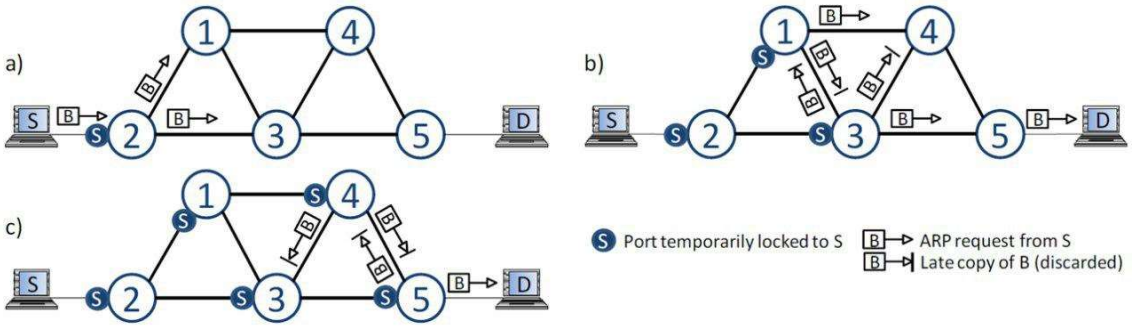


Figura 22 Descubrimiento de camino en ARP-Path [55]

3.9.2 Procedimiento de confirmación de camino en ARP-Path

La confirmación de camino en ARP-Path es el procedimiento por el que se establece un camino entre el host origen que envió el mensaje *ARP-Request* y el host destino para el que el origen solicita su dirección MAC. Este proceso está basado en el procesamiento del mensaje *ARP-Reply*. La Figura 23 representa de forma gráfica el proceso.

En a), el host D, tras recibir un *ARP-Request* procedente de S, envía un mensaje un *ARP-Reply* que recibe el nodo 5. El nodo 5 conoce el camino hacia el host S porque previamente bloqueó su MAC en un puerto, de forma que reenvía el mensaje por dicho puerto. A su vez, confirma tanto las direcciones de S como D asociándolas a sus correspondientes puertos, y estableciendo un temporizador de caché (un tiempo denominado LT) para ambas direcciones, similar a los existentes en los learning switches estándar.

En b), el nodo 3 recibe el mensaje *ARP-Reply*, procedente de 5, realizando el mismo proceso de confirmación de las direcciones S y D. Finalmente en c), el nodo 2 reenvía la trama hacia S, de forma que tras confirmar las direcciones S y D, queda establecido un camino bidireccional entre S y D. Los sucesivos paquetes que atraviesen este camino actualizarán los temporizadores de caché de los nodos, evitando que las entradas con las direcciones MAC en los nodos 2, 3, 5 caduquen mientras exista tráfico entre S y D.

Finalmente, en los nodos 1 y 4, el bloqueo temporal de la dirección S expirará eventualmente, al no haberse confirmado ningún camino a través de dichos nodos.

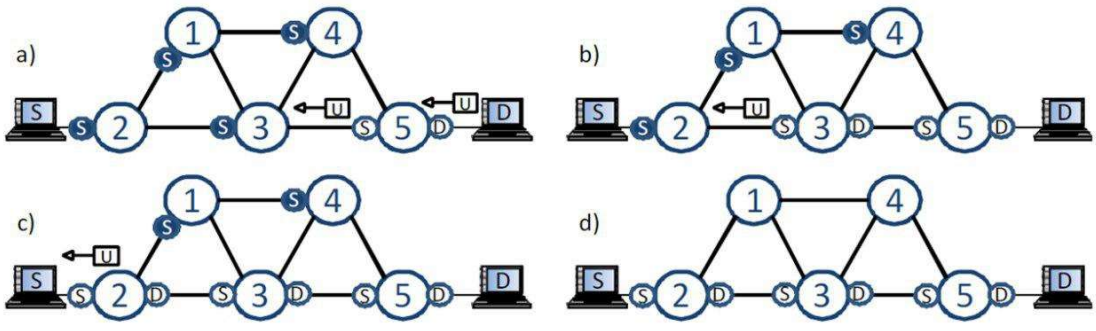


Figura 23 Confirmación de camino en ARP-Path [55]

3.9.3 Reparación de caminos

ARP-Path incorpora mecanismos de reparación de caminos. Para ello, es preciso que los switches conozcan las direcciones MAC de los hosts a los que estén conectados de forma directa (sin otros

switches ARP-Path). Esto se consigue mediante el envío de mensajes *HELLO* entre los switches de la red. Un switch transmite un mensaje *HELLO* de forma periódica por todos sus puertos. Si un switch no recibe mensajes *HELLO* por un puerto, es indicativo de que ese puerto no está conectado a un switch ARP-Path, de forma que el switch conoce que posee conexiones directas a los hosts cuyas direcciones MAC se aprendan en ese puerto.

Cuando se produce un fallo en un elemento de red, como, por ejemplo, la caída de un enlace, los switches implicados eliminan aquellas direcciones MAC cuyos caminos atraviesen el enlace afectado. Si se recibiese una trama unicast cuyo destino se desconozca, el switch afectado genera un mensaje *Path_Fail* empleando como dirección MAC destino una dirección multicast asociada a todos los switches ARP-Path, encapsulando en su interior la trama unicast original. Cuando este mensaje eventualmente llegue al switch conectado al host origen, generará un mensaje *Path_Request* cuyo funcionamiento es análogo al de un *ARP-Request*, difundiéndose por la red, y alcanzando eventualmente el switch conectado al host destino. Éste confirmará el camino mediante un mensaje *Path_Reply*.

3.10 Firewalls

En el ámbito de las redes, un cortafuegos o firewall es un sistema cuya misión es proteger una red interna de accesos no autorizados desde el exterior, mediante el filtrado de paquetes.

3.10.1 Tipos de firewalls

Los firewalls tradicionalmente se han clasificado en tres tipos [56]:

- Filtro de paquetes
- Filtro de paquetes con estado
- Proxy de aplicación

Un filtro de paquetes emplea únicamente información presente en las cabeceras de los paquetes que lo atraviesen para denegar o permitir el acceso. Este tipo de firewalls no almacena información de estado, por lo que cada paquete se trata de forma independiente. Al no almacenar estado, los filtros de paquetes pueden ser vulnerables a ataques como el escaneo de puertos TCP, donde el atacante envía segmentos TCP a todos los puertos posibles con los flags SYN y ACK activos sin previa petición de conexión. Cuando la víctima reciba dichos segmentos en los puertos en los que esté escuchando, responderá con el flag RST activo, delatando de esta manera qué puertos mantiene activos.

Los filtros de paquetes con estado surgen para dar respuesta a las limitaciones que implica no mantener estado. Estos tipos de firewall mantienen un registro de las conexiones TCP activas, de forma que un escaneo de puertos no tenga éxito, puesto que no permitirá el acceso a un segmento TCP SYNACK si no ha observado un TCP SYN previo. Como contrapartida, el rendimiento frente a un filtro de paquetes disminuye al tener que mantener estado, y realizar comprobaciones más costosas.

Un proxy de aplicación es un sistema más evolucionado frente a los filtros de paquetes con estado, pudiendo examinar el contenido del paquete más allá de las cabeceras de red (la carga útil), e implementando un firewall a nivel de aplicación. Por ejemplo, este tipo de firewall puede examinar mensajes HTTP para analizar las cabeceras incluidas, o incluso modificar el contenido. Un aspecto relevante de los proxies de aplicación es que ejerce de intermediario entre la red externa y la interna, de forma que no existe una comunicación extremo a extremo con los

paquetes originales, sino que el proxy crea un nuevo paquete por cada petición/respuesta a partir del contenido del paquete original recibido. Como es lógico, el uso de proxies de aplicación implica la aceptación de mayor latencia respecto a los dos tipos anteriores.

3.10.2 Firewalls, NFV y P4

Tradicionalmente, los firewalls se han implementado en un equipo hardware dedicado, situado en una localización concreta de la red, habitualmente en la interfaz a otras redes externas. Sin embargo, este enfoque ha comenzado a cambiar tras la aparición del concepto de Network Functions Virtualization (NFV). NFV es un paradigma que implementa servicios de red sobre módulos software, favoreciendo el desarrollo de nuevos servicios y aportando mucha mayor flexibilidad en su utilización. Un servicio NFV está compuesto por un conjunto de funciones de red virtuales (VNF).

Un firewall tradicional puede implementarse como un servicio virtualizado, de forma que ya no sea necesario dedicar un hardware específico, y pueda ser instanciado bajo demanda en cualquier lugar de la red, en contraposición a la localización física de un dispositivo hardware. SDN puede emplearse para implementar servicios NFV. Por ejemplo, un firewall puede implementarse en un switch software mediante un conjunto de reglas de flujo que implementen la política de seguridad deseada.

La flexibilidad aportada por el lenguaje P4 sin duda contribuirá al desarrollo de nuevos servicios NFV, siendo la seguridad en redes un campo con amplio potencial. Particularmente, P4 se plantea como método para implementar VNFs directamente sobre el chip procesador de red, en vez de implementarse sobre sistemas operativos, aumentando la velocidad de ejecución de las VNFs. La capacidad que posee P4 para recoger de estadísticas de forma eficiente, e implementar operaciones arbitrarias sobre paquetes, conjuntamente con su capacidad para provisionar nuevos programas de forma dinámica, abren la puerta al desarrollo de políticas de seguridad con una granularidad mayor respecto a las actuales. Un ejemplo podría ser un programa P4 destinado a analizar mensajes sencillos de capa de aplicación, que puede ser ejecutado directamente sobre hardware a la vez que posee la versatilidad de poder ser desplegado en cualquier dispositivo P4 de la red, en cualquier momento.

4 Diseño del sistema desarrollado

4.1 Introducción

En este apartado se detallará el sistema implementado en este TFM. Se ha desarrollado un switch software, programable con P4, que constituye un sistema integrado por un componente ARP-Path implementado con un extern P4, y con una interfaz P4Runtime para su uso con un controlador SDN externo. Se trata, por tanto, de un sistema híbrido, con un estado de reenvío establecido por dos planos de control independientes, uno local (ARP-Path), y otro situado en un controlador externo. El switch se ha implementado en BMv2.

Para implementar el protocolo ARP-Path, se ha diseñado un extern que proporciona primitivas al programa P4 para modificar el estado de reenvío (bloqueo y aprendizaje de direcciones MAC). A través de llamadas a métodos del extern, se ha desarrollado un fragmento de código P4 que implementa el protocolo ARP-Path a nivel básico. Esto supone una novedad respecto a los switches BMv2 estándar, ya que el switch desarrollado es capaz de encaminar paquetes

autónomamente en cualquier topología sin necesidad de emplear ningún proceso que implemente un plano de control externo (véase Figura 15).

A su vez, se ha diseñado soporte para provisionar el estado de reenvío de forma externa mediante un controlador ONOS y P4Runtime, creando un programa P4, *hybrid.p4* y una aplicación y Pipeconf específicos en ONOS. En el programa *hybrid.p4*, junto a la lógica del protocolo ARP-Path, se incluyen recursos que permiten al controlador ONOS aplicar encaminamiento en capa 2, así como recoger estadísticas, de forma que a través de la lógica del Pipeconf, se puedan exponer los recursos del switch a aplicaciones de ONOS.

Asimismo, se ha realizado una extensión del programa *hybrid.p4*, *acl_hybrid.p4* que incluye funcionalidades adicionales: filtrado, monitorización, conformado de tráfico, y soporte para ECMP/WCMP en capa 2. Para ello se ha creado un Pipeconf adicional en ONOS y una aplicación específica para su control, proporcionando una interfaz agnóstica al programador de aplicaciones ONOS. Además, se han desarrollado los componentes necesarios para permitir la provisión dinámica de uno u otro programa, sin perder el estado de reenvío, tanto en la parte controlada por ONOS, como en la controlada por ARP-Path.

Por último, se ha implementado una aplicación soportada sobre la GUI de ONOS que permite visualizar en tiempo real los caminos que siguen los paquetes de flujos establecidos entre dos hosts. Esta aplicación está basada en el concepto de header-stack en P4, y se compone de dos partes: código P4 y BMv2 para recoger estadísticas y la ruta seguida por el paquete en la red, y una aplicación ONOS que procesa la información anterior y la muestra gráficamente. Con ello se ha conseguido el doble objetivo de presentar una aplicación novedosa con P4, y desarrollar una herramienta de depuración al poder verificar los caminos establecidos.

Una representación gráfica del trabajo realizado se presenta en la Figura 24, donde se presenta la disposición lógica de los componentes desarrollados (en color amarillo):

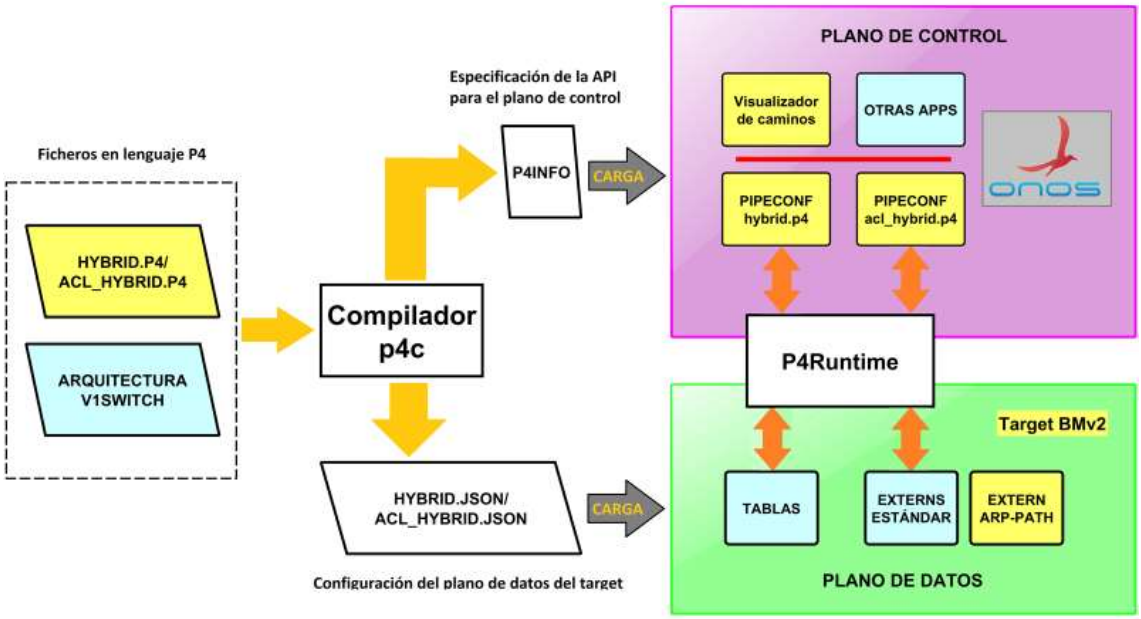


Figura 24 Esquema de los componentes desarrollados

El proceso de funcionamiento comienza con la compilación de alguno de los dos programas P4 desarrollados, *hybrid.p4* o *acl_hybrid.p4*. Ambos programas implementan el protocolo ARP-Path a partir de un extern. Tras compilar el programa escogido, se generan dos ejecutables: un fichero JSON que describe el programa P4, para su consumo por un target BMv2, y un fichero P4Info que describe la API específica del programa P4 para un cliente P4Runtime.

El fichero JSON se carga en un target BMv2 que necesariamente debe soportar el extern ARP-Path. Dicho de otra forma, los programas *hybrid.p4* y *acl_hybrid.p4* no pueden funcionar en el target estándar BMv2, Simple_Switch, ya que éste no puede mapear la lógica ARP-Path a componentes internos. Al cargar el programa P4, el target BMv2 provisionará los recursos para implementar el plano de datos deseado, lo que incluye la creación de un objeto extern a partir de las referencias a él del fichero JSON.

Por otra parte, se va a emplear como plano de control un controlador ONOS que se comunica con el target BMv2 mediante el protocolo P4Runtime. Cada programa P4 posee un Pipeconf específico, que expone los recursos del plano de datos a ONOS. Para ello, es preciso cargar la API expresada en el fichero P4Info correspondiente al programa P4 desplegado en BMv2. Con los Pipeconfs desarrollados, las aplicaciones ONOS pueden interaccionar con los switches BMv2, incluyendo a la aplicación de visualización de caminos desarrollada.

La estructura de este apartado es como sigue:

En el apartado 4.2, se presenta la arquitectura del target BMv2 desarrollado.

En el apartado 4.3, se analizará el diseño de ARP-PATH en BMv2 y P4, las posibles alternativas técnicas, y la solución finalmente implementada.

En el apartado 4.4, se detalla el programa *hybrid.p4*, y la integración con un controlador ONOS.

En el apartado 4.5, se presenta el programa *acl_hybrid.p4*. También se analiza la problemática asociada a la operación SWAP en BMv2, y las soluciones ideadas para preservar el estado de reenvío. Finalmente, se detallan los componentes desarrollados para ONOS que implementan una interfaz de control sobre el programa P4.

Finalmente, en el apartado 4.6 se detalla la implementación de la aplicación de visualización de caminos, haciendo hincapié en las operaciones del plano de datos necesarias para soportar su funcionalidad.

4.2 Estructura del target BMv2 implementado

Para facilitar la integración con ARP-Path se ha diseñado un target personalizado mediante el framework BMv2. Como ya se señaló en el apartado 3.6.1, un target BMv2 sobre el que se carga un programa P4 implementa exclusivamente un plano de datos. Es decir, el proceso correspondiente a la ejecución de un switch BMv2 no es capaz por sí mismo de conmutar paquetes. Para ello, es preciso ejecutar un proceso paralelo que implemente el plano de control, ya sea una interfaz de comandos como la que incluye BMv2, o un controlador SDN.

El switch BMv2 podrá encaminar tráfico una vez que el plano de control haya provisionado un estado de reenvío, lo que incluye no sólo la instalación de entradas en tablas, sino también la configuración de sesiones Multicast en el *Packet Replication Engine*. Tal como se ilustró en el apartado 3.3.1, el PRE es el componente encargado de implementar Multicast mediante la configuración de sesiones que asocian un identificador de grupo Multicast a un conjunto de

puertos. Broadcast puede entenderse como una sesión Multicast que tiene asociados todos los puertos del switch (excluyendo, evidentemente el puerto de entrada del paquete).

Por tanto, dado que ARP-Path está basado en la difusión controlada de paquetes, es preciso configurar una sesión Multicast para implementar Broadcast. Sin embargo, se desea que el switch ARP-Path BMv2 pueda funcionar autónomamente en ausencia de plano de control externo, es decir, que no sea necesario lanzar otro proceso distinto al del switch para poder encaminar paquetes. Por ello, el propio target BMv2 debe configurar el PRE al inicializarse, siendo ésta una de las razones que han justificado el uso de un target personalizado.

De forma general, las razones por las que se ha desarrollado un target propio son:

- La configuración de una sesión Broadcast en el PRE por defecto.
- Incluir el extern ARP-Path como un componente disponible para su instanciación en programas P4.
- Mejorar la eficiencia en la difusión de paquetes Multicast respecto a otros targets BMv2.
- Desechar funcionalidades no utilizadas (soporte para colas por prioridad).
- Recoger medidas estadísticas (véase el apartado 4.6.1).

De los puntos anteriores, se debe resaltar la implementación de la difusión de paquetes Multicast. BMv2 implementa la difusión de paquetes mediante la transmisión secuencial de cada una de las copias del paquete por puerto. En un sistema hardware, la transmisión se podría realizar en paralelo. El problema en este punto no es la limitación técnica en sí, sino que por defecto el orden de transmisión es estático y está definido por el identificador de puerto. Así, la copia Multicast que primero se envía se corresponde con el puerto cuyo identificador sea menor. En ARP-Path esto implicaría que, a menor identificador de puerto, mayor probabilidad de que el camino se establezca a través de dicho puerto, ya que las copias de un ARP-Request se transmitirían en orden creciente de identificador de puerto. Por ello, se ha implementado un sistema de difusión en el que la transmisión sigue siendo secuencial, pero el orden de transmisión de las copias pasa a ser aleatorio, de tal forma que se facilite la variabilidad de caminos en ARP-Path.

El target desarrollado es un switch BMv2 multihilo, escrito en C++. El target se ha denominado *ARP-P4.* La arquitectura de *ARP-P4* tiene mucho en común con el target *l2_switch* de BMv2, pero incluye algunas funcionalidades del target *simple_switch*.

Un target BMv2 es una clase derivada de *bm::SwitchWContexts* o bien *bm::Switch*, que a su vez deriva de la primera. *bm::SwitchWContexts* permitiría crear un target capaz de ejecutar varios programas P4 excluyentes en paralelo , como por ejemplo, un programa P4 para IPv4 y otro para IPv6. En este caso el paquete atravesaría un programa u otro en base su versión de IP. Sin embargo, esta funcionalidad no está implementada todavía, por lo que todos los targets desarrollados hasta el momento derivan de *bm::Switch*.

Para implementar un target, se requiere, como ya se ha dicho, crear una clase derivada de alguna de las dos referenciadas, e implementar los métodos *receive_()* y *start_and_return_()*. El primer método es llamado automáticamente por el componente *DevMgr* cada vez que se recibe un paquete. *Start_and_return_()* está pensado para iniciar el procesamiento de paquetes, si bien es opcional implementar esta operación en este método.

Los puntos que resaltar sobre el target *ARP-P4* son las siguientes:

- Tres hilos de procesamiento paralelos para el plano de datos.
- Cuatro bloques programables con P4: Parser, Ingreso, Egreso y Deparser
- Soporte para multicast mediante el componente PRE.
- Sesión Multicast para broadcast implementada por defecto.
- Soporte para clonación de paquetes en el bloque de control de Ingreso.
- Compatibilidad con P4Runtime.

La arquitectura P4 empleada es *V1Switch*, la única disponible para targets P4 a día de hoy, heredando por tanto, el conjunto de metadatos estándar definidos para esta arquitectura, *standard_metadata*. Existen dos bloques de control, *Ingreso* y *Egreso*. Ingreso tiene la funcionalidad genérica de determinar el destino del paquete, mientras que Egreso está pensado para realizar un procesado posterior del paquete una vez ya se ha fijado su destino. Es por ello que el puerto de salida -el metadato *standard_metadata.egress_port*- únicamente puede modificarse en el bloque de Ingreso, debiendo permanecer inalterado en el de egreso.

En la inicialización de *ARP-P4* se configura una sesión multicast en el PRE asociada a todos los puertos a través de la cual se implementa Broadcast. Esto no excluye que el PRE también pueda soportar otras sesiones configuradas mediante P4Runtime.

Se han seguido algunas de las recomendaciones para mejorar el rendimiento expuestas en [26]. Específicamente, se ha balanceado a la distribución de los bloques de control entre los tres hilos asignados al plano de datos. La funcionalidad de los 3 hilos es la siguiente:

- El *hilo de recepción* se encarga de recibir un paquete, llamar al bloque Parser, e introducir el paquete en la cola de entrada, que comunica con el hilo de procesado.
- El *hilo de procesado* recoge paquetes de la cola de entrada, y llama a los bloques Ingreso y Egreso. Si el paquete tiene que ser difundido por varios puertos, mediante el PRE se realizarán tantas copias del paquete como puertos existan para la sesión multicast y cada uno de los paquetes clonados atravesará el bloque de Egreso. Tras atravesar el bloque de Egreso, el paquete se depositará en la cola de salida.
- El *hilo de transmisión* recoge paquetes de la cola de salida, llama al Deparser y lo reenvía por el puerto correspondiente.

Por último, *ARP-P4* tiene soporte para P4Runtime, para lo cual se ha adaptado la clase *switch_runner* disponible en el target *simple_switch_grpc*. *Switch_runner* inicializa el servidor P4Runtime, y dispone de soporte para la gestión de mensajes Packet-In y Packet-Out mediante la especificación de un número de puerto que identifica al plano de control P4Runtime. Es habitual emplear el número 255 como identificador de puerto del plano de control, de forma que todos los paquetes dirigidos a este puerto serán interceptados e inyectados al servidor P4Runtime para su envío en un mensaje Packet-In.

La Figura 25 representa la estructura del target *ARP-P4*, mostrándose la secuencia que siguen los paquetes recibidos, las operaciones Packet-In y Packet-Out, y la localización de los bloques programables mediante P4.

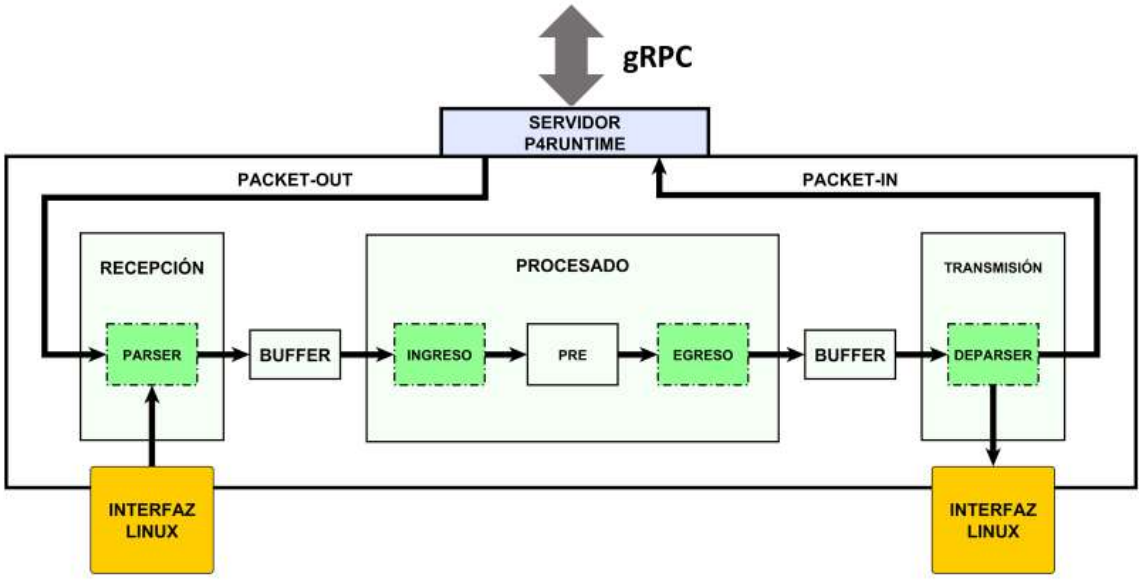


Figura 25 Estructura del target BMv2 desarrollado

4.3 Implementación del protocolo ARP-Path

En este apartado se revisará la implementación del protocolo ARP-Path en el target *ARP-P4*. Primero se analizará la lógica del protocolo desde una perspectiva conceptual, posteriormente se analizará una implementación sobre código P4, y finalmente, se detallará la solución escogida, que emplea un objeto extern.

4.3.1 Lógica del protocolo ARP-Path

El estado de reenvío de ARP-Path está basado en una tabla formada por tuplas compuesta por una dirección MAC, un número de puerto, y una marca de tiempo. El número de puerto identifica el puerto por el que se alcanza la dirección MAC asociada, y la marca de tiempo indica la validez de la entrada, estando caducada si el tiempo actual es mayor que dicho valor. De los elementos anteriores, el único que no se puede repetir entre todas las tuplas de la tabla es la dirección MAC, por lo que se puede afirmar que la clave de cada entrada sería la dirección MAC. Se considera que, en una búsqueda, no hay coincidencia en una tabla si una entrada está caducada, aunque sus valores MAC y puerto coincidan con los de búsqueda. Por tanto, en todas las búsquedas referenciadas en este apartado, se asume implícitamente una comprobación de la caducidad de la entrada.

Conceptualmente existen tres tipos de búsquedas de entradas en esta tabla:

- **Por dirección MAC origen**: para comprobar si existe una entrada con dicha MAC, y añadir una nueva entrada en caso negativo.
- **Por MAC origen y puerto**: para difundir si existe coincidencia o rechazar en caso contrario paquetes broadcast.
- **Por MAC destino**: para encaminar paquetes unicast en base al número de puerto.

La búsqueda por dirección MAC origen solamente se emplea en paquetes ARP. Si no existe coincidencia en la tabla, se inserta una entrada con la MAC origen del paquete, el número de puerto de entrada, y una marca de tiempo. El valor de marca de tiempo es la suma del instante

actual más un valor de *BT* (Blocking Time) si el paquete es broadcast (ARP Request), o *LT* (Learning Time) si es unicast (ARP Reply).

La búsqueda por MAC origen y puerto se emplea para bloquear la difusión paquetes broadcast, evitando la formación de bucles. Este tipo de búsqueda solamente ocurre con paquetes broadcast. El valor de puerto permite tomar la decisión de difundir o no un paquete broadcast cuya MAC origen coincida con el valor de MAC de una entrada:

- Si el valor de puerto coincide con el puerto de entrada del paquete, el paquete puede difundirse sin provocar bucles.
- Si el valor de puerto no coincidiese por el puerto de entrada, es indicativo de que ya existe un puerto asociado a la MAC origen distinto del puerto del paquete, por lo que se rechaza el paquete para evitar la creación de bucles.

La coincidencia con una entrada en este tipo de búsqueda provoca una posible actualización de la marca de tiempo si su valor es menor que el instante de tiempo actual más un valor de BT.

Por último, la búsqueda por MAC destino se emplea para encaminar paquetes unicast a partir del puerto asociado a una MAC. El puerto indica la interfaz a partir de la cual se alcanzaría el host destino a través de un camino ya formado. En condiciones de operación normal de la red, cada vez que se realiza esta búsqueda siempre debe existir una coincidencia, ya que debe existir un camino previamente formado. Por cada coincidencia con una entrada, se actualiza incondicionalmente el valor de entrada al instante actual más LT.

Por otra parte, la existencia de entradas caducadas supone una pérdida de rendimiento al aumentar innecesariamente el espacio de búsqueda, y un aumento de la memoria consumida, razón por la cual deben eliminarse. Por ello, en la implementación de ARP-Path debe existir una rutina periódica que realice un barrido de todas las entradas de la tabla, borrando aquéllas que están caducadas. A su vez, debe llegarse a un compromiso entre el periodo de ejecución de la rutina de borrado y la tolerancia a la existencia de entradas caducadas. Un periodo de ejecución de 0s garantiza la inexistencia de entradas caducadas, pero posiblemente añada una sobrecarga innecesaria al procesamiento de cada paquete. Un periodo de tiempo relativamente alto puede implicar la existencia de un número de entradas caducadas que pueda lastrar el tiempo de búsqueda en tabla si ésta contiene un número elevado de entradas.

4.3.2 Implementación sobre P4 y P4Runtime

La implementación del protocolo ARP-Path conceptualmente es posible mediante P4Runtime y el uso del extern Register (registro). Se emplearía una tabla para almacenar direcciones MAC, y un vector de tantos registros como direcciones MAC haya aprendido el sistema.

El puerto y la marca de tiempo para una MAC se almacenarían en un registro, cuyo índice dentro del vector de registros estaría asociado a una MAC concreta. La tabla se utilizaría para realizar una asociación entre una MAC y su índice en el vector de registros, a través de la definición de una acción que escribiese en un metadato el índice para la MAC que ha provocado un hit en la tabla.

Posteriormente, accediendo al registro se verificaría la caducidad de la entrada, y se procesaría el puerto en función del tipo de paquete (unicast o broadcast), actualizando a su vez la marca de tiempo (a partir de la marca de tiempo del paquete, presente en la arquitectura PSA).

El cliente P4Runtime sería el encargado de ejecutar periódicamente la rutina de barrido de entradas, para eliminar todas aquellas que estuviesen caducadas.

Sin embargo, a día de hoy, esta opción no es realizable, ya que el soporte para lectura/escritura en registros no está implementado todavía en P4Runtime.

4.3.3 Implementación sobre externs

Otra solución para implementar ARP-Path en un target a nivel local es mediante el desarrollo de un extern específico. Recordando lo dicho en el apartado 3.2.2, un extern es un objeto que expone una funcionalidad específica de un target (no implementada mediante código P4), a un programa P4 mediante una interfaz. Un extern está embebido dentro del propio target, y su implementación es desconocida para el programa P4, exponiendo una serie de métodos que se pueden llamar en el programa. En el caso de BMv2, un extern sería una clase programada en C++.

Asimismo, P4Runtime posee soporte para externs, como uno de los posibles mensajes que puede aparecer dentro de un mensaje *Entity* (véase apartado 3.4.6), cuyo formato interno es totalmente libre, por lo que se pueden implementar primitivas de comunicación entre un extern y un cliente P4Runtime.

En cierto modo, se puede considerar que la implementación de ARP-Path en un extern es el caso opuesto a la implementación nativa sobre P4 y P4Runtime del apartado anterior:

- Un extern permite una implementación eficiente, al estar desarollada con la propia tecnología del target, y no depender de las limitaciones impuestas por P4.
- Un extern es una implementación específica para cada target, a diferencia de la implementación sobre P4 y P4Runtime.
- La complejidad en la implementación de un extern depende esencialmente del target, mientras que la implementación con P4 y P4Runtime suele ser mucho más rápida y sencilla, algo que puede compensar su falta de rendimiento.

Se ha implementado un extern *ArpPath* que modela la tabla ARP-Path, e implementa primitivas para su modificación dentro de un programa P4. El extern dispone de una interfaz a través de la cual un programa P4 puede solicitar servicios del extern, tales como encaminar tramas unicast o difundir de forma controlada tramas broadcast.

La implementación del protocolo ARP-Path se ha realizado en un bloque de control denominado *ArpPathPipeline*, que instancia el citado extern *ArpPath* y llama a sus métodos de acuerdo con la lógica del protocolo. De esta forma, se puede integrar ARP-Path en cualquier programa P4 mediante la inclusión y llamada a este bloque de control.

4.3.3.1 Interfaz P4 del extern ArpPath

La interfaz que posee el extern para programas P4 se incluye a continuación:

```
extern ArpPath {
    ArpPath(bit<64> size);
    void flood(in bit<48> addr, in bit<9> port, in bit<16> ethType);
    void forward(in bit<48> addr);
    void path_reply(in bit<48> addr, in bit<9> port);
    void num_entries(out bit<32> n);
}
```

Como se observa, el extern ArpPath define un constructor cuyo parámetro a incluir es el tamaño de tabla deseado, y cuatro métodos para su uso directo en el código P4:

- **Flood**: Difunde o descarta tramas broadcast en función de su puerto de entrada. Si la MAC origen no existiese en la tabla ARP-Path, se inserta una nueva entrada con BT segundos de tiempo de caducidad, y se difunde el paquete. Recibe como parámetros la MAC origen, el puerto de entrada de la trama, y su campo Ethernet Type para comprobar si es un mensaje ARP.
- **Forward**: Encamina tramas unicast cuyas MAC destino estén contenidas en la tabla ARP-Path, refrescando LT segundos su tiempo de caducidad. En caso contrario, el paquete puede descartarse o enviarse a un cliente P4Runtime, según la configuración. Requiere como único parámetro la dirección MAC destino de la trama.
- **Path_Reply**: Inserta o refresca la entrada ARP-Path identificada por la dirección MAC pasada como parámetro. En ambos casos el tiempo de caducidad de la entrada se establece a LT segundos. Este método está orientado a insertar entradas para las MAC origen de tramas ARP-Reply, aunque podría usarse en programas P4 para insertar entradas de forma explícita.
- **Num_entries**: Escribe en el parámetro pasado como argumento el número de entradas presentes en la tabla ARP-Path independientemente de su validez (caducadas o no).

Los métodos *Flood* y *Forward* operan directamente sobre el conjunto de metadatos estándar de la arquitectura *V1Switch* para el reenvío de tramas. Así, por ejemplo, *Forward* establece el puerto de salida de la trama escribiendo en el metadato *egress_port* el valor del identificador del puerto de salida.

4.3.3.2 Implementación del extern ArpPath en BMv2

BMv2 permite crear externs mediante el desarrollo de clases derivadas de *bm::ExternType*. Sobre la clase extern debe implementarse su funcionalidad mediante código C++, y exponer los métodos de la interfaz que se desee exponer al programa P4. Para el caso del extern que nos ocupa, los métodos a exponer serían *flood, forward, path_reply*, y *num_entries*, sin perjuicio de que la clase tenga otros métodos internos o destinados para su uso por un cliente P4Runtime.

La clase *ArpPath*, que modela el extern homónimo, incluye un mapa de memoria sobre el que se implementa la tabla ArpPath, y sus métodos asociados de inserción, búsqueda, actualización, y borrado de entradas.

En un extern BMv2 existen dos tipos de métodos:

- Métodos de interfaz: Modelan primitivas instanciables en programas P4.
- Métodos internos: Incluyen métodos utilizados por métodos de interfaz, y métodos destinados a su uso por un cliente P4Runtime.

Como métodos de interfaz se incluyen, como no puede ser de otra manera, los cuatro métodos de la interfaz P4 del extern: *flood, forward, path_reply, y num_entries*. Para que estos métodos puedan referenciarse desde un programa P4 es necesario emplear una serie de macros para registrar el extern y sus métodos, de forma que BMv2 reconozca la clase desarrollada como un extern P4.

La Figura 26 representa gráficamente la interrelación entre la interfaz P4 y la clase C++ que implementan de forma conjunta el extern ArpPath:

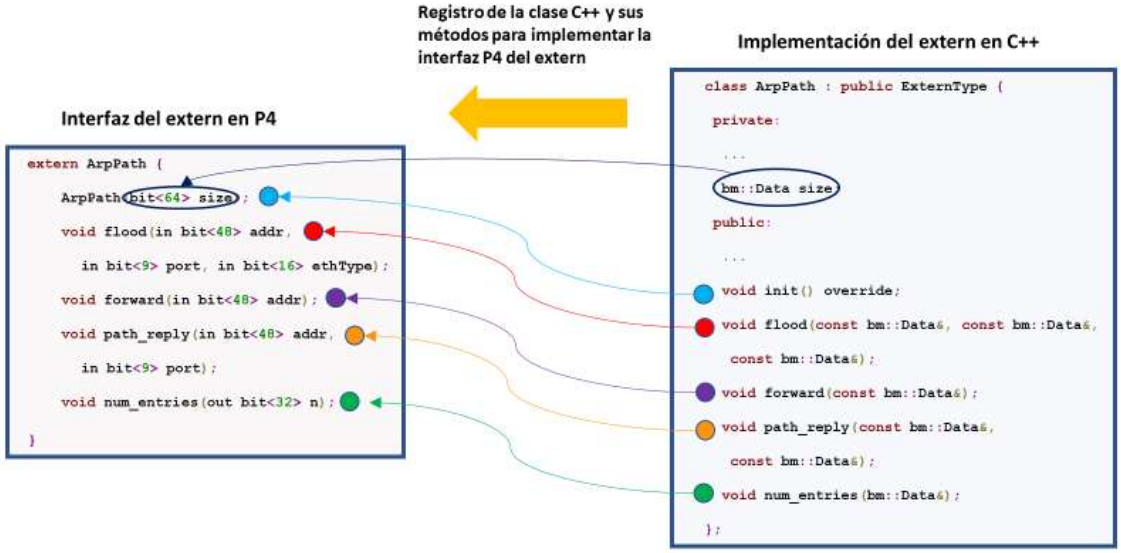


Figura 26 Registro de una clase C++ como extern en BMv2

4.3.3.3 Bloque de control ARP-Path en el código P4

El bloque de control *ArpPathPipeline* implementa la lógica del protocolo ARP-Path mediante las llamadas a métodos del extern según el tipo de paquete procesado. Este bloque de control puede ser incluido y llamado desde un programa P4 arbitrario para añadir soporte del protocolo ARP-Path. Por su simplicidad, se incluye el código P4 de ArpPathPipeline a continuación:

```
control ArpPathPipeline(inout headers_t hdr, inout
intrinsic_metadata_t intrinsic_metadata, inout standard_metadata_t
standard_metadata, ArpPath extern_arp_path) {
    apply {
        if (hdr.ethernet.dstAddr == BROADCAST) {
            extern_arp_path.flood(hdr.ethernet.srcAddr,
standard_metadata.ingress_port, intrinsic_metadata.l2_next_hdr);
        }
        else {
            extern_arp_path.forward(hdr.ethernet.dstAddr);
            if (intrinsic_metadata.l2_next_hdr == ARP_TYPE) {
                extern_arp_path.path_reply(hdr.ethernet.srcAddr,
standard_metadata.ingress_port);
            }
        }
    }
}
```

El bloque de control *ArpPathPipeline* recibe como parámetros de entrada las cabeceras del paquete, los metadatos estándar de *V1Switch* (*standard_metdata*), los metadatos definidos por el usuario (*intrinsic_metadata*) y una referencia a un extern ArpPath definido anteriormente.

El flujo de control del código sigue la lógica del protocolo ARP-Path: en función del tipo de trama se llama a métodos distintos del extern. Para tramas broadcast, se solicita al extern que difunda la trama si es pertinente (*flood*). Para tramas unicast, se encamina la trama por un puerto previamente aprendido (*forward*), y si además la trama es ARP-Reply, se aprende o refresca la dirección MAC origen (*path_reply*).

La Figura 27 representa la lógica expresada en el bloque *ArpPathPipeline*, considerando tanto la lógica residente en P4 como en el extern. Para entender el diagrama, hay que realizar las siguientes precisiones:

- En verde, lógica expresada en el código P4.
- En rojo, lógica del método *flood* del extern.
- En azul, lógica del método *forward* del extern.
- En morado, lógica del método *path_reply* del extern.
- LT_port hace referencia al puerto almacenado para una MAC, ingess_port hace referencia al puerto de entrada del paquete, y miss_port es el puerto por el que reenviar cuando no existe una entrada en tabla (por defecto, se tira el paquete).
- DMAC hace referencia a la MAC destino, y SMAC a la MAC origen.
- El reenvío unicast y la difusión implican refresco temporal de la entrada de tabla afectada.

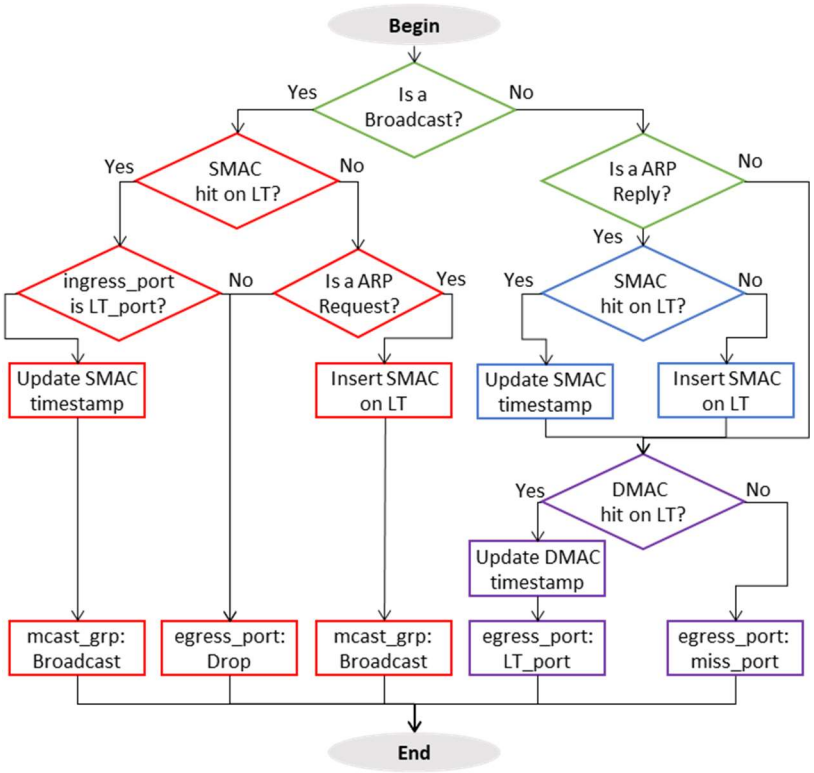


Figura 27 Diagrama de flujo de la lógica de ArpPathPipeline

La Figura 28 presenta gráficamente cómo interacciona el extern ArpPath en el contexto de un programa P4 desplegado en BMv2:

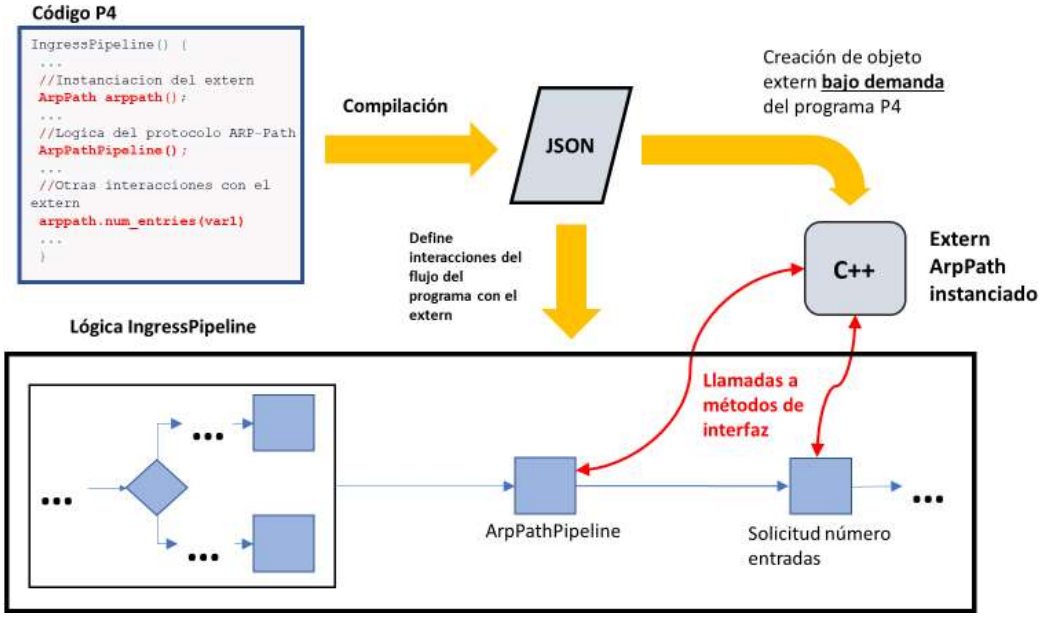


Figura 28 Interrelación del extern ArpPath con un programa P4

En primer lugar, se parte de código P4 que contiene referencias al extern y llamadas a sus métodos. Al compilarlo, el archivo resultante será un fichero JSON que conservará las referencias del código P4. Al consumir el fichero JSON, el target BMv2 realizará las siguientes acciones:

- Tras leer la declaración del extern en el fichero JSON, BMv2 creará un objeto de la clase ArpPath y lo inicializará con el parámetro *size* indicado en el programa P4.
- Las llamadas a métodos de la interfaz del extern en el programa P4, se traducirán en llamadas a los métodos de la clase ArpPath.

Como se observa, el uso del extern ArpPath abstrae al programa P4 de la implementación de todas las operaciones asociadas a la tabla ARP-Path, desde las búsquedas, al borrado de entradas caducadas. El código anterior implementa la lógica del protocolo, pero el uso del extern puede ir más allá de ARP-Path, ya que otras partes del programa también podrían llamar a métodos del extern.

4.3.4 Conclusiones sobre la implementación de ARP-Path en P4

De nuevo, se vuelve a resaltar que un extern es específico de un target, por lo que un extern ARP-Path deberá desarrollarse en exclusiva para cada target, con su propia tecnología. Como BMv2 es un switch software destinado a experimentación principalmente, un extern ARP-Path supone la opción más adecuada al tener unos costes de implementación y depuración reducidos. Además, el extern desarrollado ha definido un nuevo concepto de switches BMv2 autónomos, en los que no es necesario instanciar un proceso externo (un plano de control P4Runtime, por ejemplo) para poder encaminar paquetes.

El extern desarrollado puede ampliarse para soportar interacciones con un cliente P4Runtime. De esta forma el cliente P4Runtime podría conocer el estado de reenvío de ARP-Path, o incluso modificarlo. Esto posibilita la creación de switches SDN plenamente híbridos, con controladores

SDN que pueden acceder y modificar estados de reenvío locales a través de mecanismos estandarizados con P4Runtime.

Sin embargo, crear un extern en targets hardware puede ser enormemente complicado, por lo que para estos casos una implementación nativa sobre P4 y P4Runtime posiblemente sea la opción más adecuada en términos de esfuerzo y tiempo de desarrollo.

Finalmente, es preciso comentar detalles relevantes acerca de la generación de caminos en ARP-Path. Idealmente, según el protocolo, el camino generado por un ARP Request se correspondería con el más rápido en este instante de tiempo (lo que no quiere decir que sea el más corto en saltos). La implementación ideal, en switches hardware, implica con toda probabilidad que la transmisión física en cada puerto son procesos paralelos: un paquete puede transmitirse por un puerto a la vez que por otro puerto se transmite otro paquete. Sin embargo, en un switch BMv2 no existe una transmisión paralela, dado que solamente existe un único hilo de ejecución compartido por todos los puertos. Esto es relevante en el proceso de difusión, porque la difusión de paquetes no es paralela, y, por tanto, el primer paquete broadcast reenviado tiene más probabilidad de llegar antes que el resto. Dado que la difusión broadcast es un proceso en serie, si los puertos por los que se deben difundir el paquete tienen un orden específico, se desvirtúa el concepto de balanceo de carga asociado a la exploración de la red, puesto que las primeras tramas broadcast tendrán mucha más probabilidad de llegar antes que las últimas. Para ello, una operación clave para que ARP-Path tenga la posibilidad de balancear la carga es la aleatorización en el orden de puertos al realizar la difusión de paquetes. De esta forma el puerto de transmisión de la primera trama broadcast varía, contribuyendo a la variabilidad de caminos generados.

A lo anterior hay que añadir además que salvo que se emplee un equipo hardware por switch BMv2, una red que contenga varios switches BMv2, provocará que el camino generado dependa también de la planificación de procesos del Sistema Operativo: el orden de ejecución de cada switch será determinante para la latencia de un camino. Todas estas son limitaciones con las que se debe lidiar cuando se trabaje con redes emuladas en un mismo equipo hardware.

4.4 Integración con un controlador ONOS

En este apartado se va a realizar el análisis de la integración de un switch BMv2 ARP-Path con un controlador ONOS empleando P4Runtime. El controlador ONOS tendrá la capacidad de implementar estado de reenvío en capa 2 complementario al de ARP-Path, y recoger estadísticas acerca del número de paquetes procesados por el switch. Para ello, se ha creado un programa P4 para el target *ARP-P4*, *hybrid.p4*, que incluye el componente ARP-Path desarrollado y define una serie de objetos P4 adicionales destinados al control específico por un cliente P4Runtime externo.

4.4.1 Programa hybrid.p4

El objetivo de este programa P4 es implementar un plano de datos controlado por dos planos de control: el plano de control ARP-Path ya descrito, y un plano de control P4Runtime, en este caso el controlador ONOS, destinado a gestionar el resto de objetos del programa. Al existir dos planos de control independientes destinados a la misma funcionalidad, encaminamiento en capa 2, uno debe tener prioridad sobre el otro. Se ha considerado que el controlador ONOS debe tener prioridad sobre el protocolo ARP-Path, ya que dispone de una visión completa de la red, y por ello, potencialmente puede realizar un encaminamiento más eficiente.

Para integrar un programa P4 con un controlador SDN, es común definir soporte para las operaciones Packet-In y Packet-Out.

Packet-In comprende el envío de un paquete proveniente del plano de datos hacia el controlador ONOS a través de P4Runtime. Cuando el plano de datos quiera realizar esta operación, el puerto de destino del paquete se establecerá con el valor *CPU_PORT*. El controlador normalmente necesita conocer información adicional sobre el contexto del paquete, siendo un ejemplo paradigmático, el puerto por el que el plano de datos recibió el paquete. Para ello, se emplea una cabecera *Packet-In* sobre Ethernet, que contiene un único campo, el puerto de ingreso, a partir del cual ONOS podrá conocer el puerto por el que el switch recibió el paquete. Está cabecera se establecerá como válida únicamente cuando el plano de datos solicite la operación Packet-In.

El contexto sobre el Packet-Out para este programa se reduce a indicar al plano de datos el encaminamiento a aplicar sobre el paquete. Se emplea una cabecera *Packet-Out* con dos campos, el puerto de egreso y el grupo multicast. El puerto de egreso indica al plano de control por donde encaminar el paquete recibido en el mensaje Packet-Out, mientras que el grupo multicast, si es distinto de 0, indica el grupo multicast a través del cual se debe difundir el paquete. Esta cabecera únicamente se parseará si un paquete se ha recibido por el puerto CPU_PORT, y se invalidará tras extraer sus campos.

Como el programa solamente expondrá recursos de capa 2, el Parser únicamente extraerá las cabeceras Packet-Out y Ethenet (por defecto). A su vez, el Deparser, en correspondencia solo tendrá soporte para añadir cabeceras Packet-In y Ethernet.

El programa dispone de una tabla de encaminamiento en capa 2, denominada *l2_fwd*. Esta tabla posee los siguientes match fields, todos ellos de tipo ternario:

- ***Dirección MAC destino***
- ***Dirección MAC origen***
- ***Ethernet Type***
- ***Puerto de entrada***

La especificación ternaria de todos los campos permite que para instalar una entrada no se tengan que añadir valores de todos y cada uno de los campos. Por ejemplo, puede añadirse una entrada especificando únicamente un valor específico de Ethenet Type, empleando una máscara de ceros en el resto de los campos. El hecho de emplear match fields ternarios obliga a especificar una prioridad por cada entrada para resolver colisiones entre varias entradas.

Las acciones definidas para la tabla *l2_fwd* son las siguientes:

- ***_nop***: No realizar ninguna acción
- ***Forward***: Establece el puerto de salida del paquete
- ***Drop***: Descarta el paquete
- ***Send_to_cpu***: Envía el paquete al cliente P4Runtime estableciendo el puerto de salida con el valor de CPU_PORT.

La definición en P4 de la tabla *l2_fwd* se muestra a continuación (*hybrid.p4*):

```
table l2_fwd {
        key = {
```

```
                hdr.ethernet.dstAddr: ternary;
                hdr.ethernet.srcAddr: ternary;
                intrinsic_metadata.l2_next_hdr: ternary;
                standard_metadata.ingress_port: ternary;
        }
        actions = {
                _nop;
                send_to_cpu;
                _drop;
                forward;
        }
}
```

hybrid.p4 también realiza una llamada al bloque de control ARP-Path, *ArpPathPipeline*, si tras llamar a la tabla *l2_fwd* no se ha modificado el puerto de salida. Es decir, se emplea el protocolo ARP-Path para encaminar el paquete solo si el controlador no ha establecido otro destino para el paquete. Por esto se dice que el encaminamiento indicado por el controlador tiene prioridad sobre el encaminamiento dictado por ARP-Path.

Por último, el programa define 3 vectores de contadores, con tantos elementos como puertos soporte el target:

- ***Contadores de ingreso***: cuentan los bytes y paquetes recibidos en cada puerto.
- ***Contadores de egreso***: cuentan los bytes y paquetes enviados por cada puerto.
- ***Contadores de descarte***: cuentan los bytes y paquetes recibidos en cada puerto descartados.

4.4.2 Creación de la aplicación Pipeconf en ONOS

Como ya se explico en el apartado 3.7.2, para gestionar el estado de reenvío de un programa P4 con ONOS, es preciso crear un Pipeconf que provisione el programa P4 y cargue el correspondiente archivo P4Info en el cliente P4Runtime, y añada extensiones de comportamiento adicionales.

Se ha creado una aplicación ONOS denominada *arp4th-pipeconf*, que gestiona la provisión del programa *hybrid.p4* en targets *ARP-P4*, y permite gestionar sus objetos P4.

Las extensiones definidas por el Pipeconf creado son tres:

- Una implementación de la interfaz *PiPipelineInterpreter*, que implemente un mapeo entre el programa P4 y el framework *Flow Rule*.
- Una implementación de la interfaz *PortStatisticsDiscovery*, que recoge estadísticas sobre puertos periódicamente.
- La implementación por defecto del componente Pipeliner para su uso por el framework *Flow Objective*, ya que existe una única tabla en el programa.

4.4.2.1 Implementación de la interfaz PIPipelineInterpreter

La clase que implementa la interfaz *PiPipelineInterpreter* tiene como objetivo realizar asociaciones entre entidades P4 y clases pertenecientes a la interfaz Northbound de ONOS, de forma que el estado de reenvío se pueda gestionar desde el framework *Flow Rule*, generalista para todo tipo de protocolos Southbound.

Las operaciones de traducción o mapeo contenidas en esta clase son de varios tipos:

- **Mapeo entre clases derivadas de *Criteria* y campos de cabecera y metadatos P4**: las clases derivadas de *Criteria* se emplean en ONOS para modelar match fields de una forma agnóstica respecto a protocolos Southbound. Se emplean para asociar los match fields empleados en las tablas de los programas P4 a campos *Criteria*, de forma que se pueden crear selectores de tráfico agnósticos respecto a los protocolos Southbound.
- **Mapeo entre IDs de tabla (*TableId*) y nombres de tabla en P4**: ONOS identifica las tablas presentes en un plano de datos mediante identificadores númericos. Cada tabla de un programa P4 puede asociarse a un *TableId* específico.
- **Mapeo entre Instrucciones y acciones P4**: Las instrucciones de ONOS representan el tratamiento a aplicar sobre un paquete, y tienen su correspondencia con acciones en P4. Para poder crear reglas de flujo en ONOS agnósticas respecto del protocolo Southbound, hay que realizar traducciones entre instrucciones y acciones en P4. Por ejemplo, la instrucción *Output* de ONOS puede mapearse con la acción *forward*, pero la instrucción *Controller* (enviar paquete al controlador, Packet-In) debe mapearse con la acción *forward* con el parámetro CPU_PORT. Como se observa en el ejemplo de la Tabla 1, la correspondencia puede no ser directa, y hay que manejarla con código adicional.
- **Mapeo entre operaciones Packet-In y Packet-Out**: En ONOS se debe extraer la información contenida en la cabecera Packet-In, y formar la cabecera Packet-Out. Para tratar la operación Packet-In, basta con extraer el puerto de entrada, y tratar al resto del paquete como una trama Ethernet. Para la operación Packet-Out, hay que mapear las instrucciones de ONOS a los campos definidos en la cabecera (grupo multicast y puerto de salida). La operación Packet-Out en *hybrid.p4* solamente soporta dos instrucciones de ONOS: reenviar el paquete por el puerto indicado, y difundir el paquete. Para la primera instrucción, se copia el puerto de salida deseado al campo correspondiente de la cabecera, y se establece a 0 el grupo multicast. Para implementar la difusión en Packet-Out, el campo de grupo multicast se establece a 1 (que ordenará al PRE que difunda al paquete).

La Tabla 1 muestra los mapeos realizados en el Pipeconf para las tablas, match fields, y acciones del programa P4.

Tabla 1 Mapeo entre ONOS y P4 para hybrid.p4

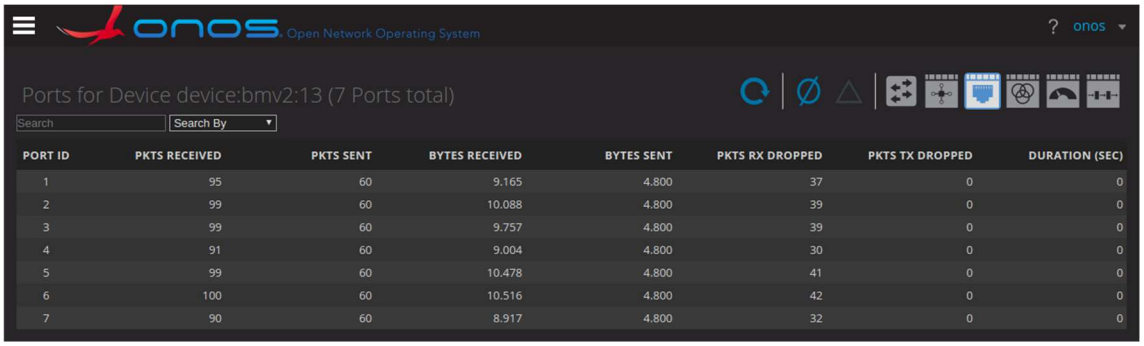
ONOS	***P4***
ID DE TABLA	**TABLA P4**
0	IngressImpl.l2_fwd
CRITERION TYPE	**P4 MATCH FIELD**
IN_PORT	standard_metadata.ingress_port
ETH_DST	hdr.ethernet.dstAddr
ETH_SRC	hdr.ethernet.srcAddr
ETH_TYPE	intrinsic_metadata.l2_next_header
INSTRUCTION	**ACCIÓN P4**
-	IngressImpl.drop_
OUTPUT(CONTROLLER)	IngressImpl.send_to_cpu
OUTPUT(puerto físico)	IngressImpl.forward(puerto físico)
NOACTION	IngressImpl.drop_

4.4.2.2 *Implementación de PortStatisticsDiscovery*

PortStatisticsDiscovery es una interfaz que permite a ONOS obtener estadísticas acerca de los puertos de un switch, en concreto de acerca de los bytes y paquetes recibidos, enviados y descartados en cada puerto.

Se ha implementado una clase que implementa la interfaz de PortStatisticsDiscovery que obtiene las estadísticas leyendo los contadores especificados en hybrid.p4 mediante P4Runtime, de forma que sus estadísticas puedan ser recogidas por ONOS y estén disponibles para su consulta por el usuario o las aplicaciones.

La Figura 29 muestra la visualización de estadísticas de puertos en la GUI de ONOS, para un switch BMv2:



PORT ID	PKTS RECEIVED	PKTS SENT	BYTES RECEIVED	BYTES SENT	PKTS RX DROPPED	PKTS TX DROPPED	DURATION (SEC)
1	95	60	9.165	4.800	37	0	0
2	99	60	10.088	4.800	39	0	0
3	99	60	9.757	4.800	39	0	0
4	91	60	9.004	4.800	30	0	0
5	99	60	10.478	4.800	41	0	0
6	100	60	10.516	4.800	42	0	0
7	90	60	8.917	4.800	32	0	0

Figura 29 Estadísticas de puertos en la GUI de ONOS

4.4.3 Integración del switch BMv2 con el controlador ONOS

4.4.3.1 *Descubrimiento de switches*

ONOS conoce la existencia de un switch P4Runtime y establece una conexión gRPC con él a través de la carga de la configuración del switch en el servicio *Network Configuration*[57], mediante un HTTP POST enviado a /onos/v1/network/configuration. En el cuerpo del POST se incluyen todas las características del dispositivo, incluyendo identificador unívoco, driver asociado (BMv2 en este caso), puertos y sus características, y el Pipeconf con el que debe funcionar (que ONOS provisionará a través del driver mediante la operación SetForwardingPipelineConfig).

La configuración del dispositivo a anunciar se expresa en una estructura JSON con el siguiente formato, del que se enumeran los campos más importantes:

- **Basic** (información básica)
  - **Driver**: Driver del dispositivo, lo que incluye el protocolo Southbound a utilizar.
  - **Name**: Nombre del dispositivo.
- **General Provider** (información específica).
  - **P4Runtime**:
    - IP: dirección IP del dispositivo con el servidor P4Runtime.
    - Port: Puerto donde escucha el servidor P4Runtime.
    - DeviceId: Identificador unívoco del dispositivo.
    - DeviceKeyId: Extensión del deviceId que indica qué instancia de ONOS es el maestro del dispositivo.

    - **gNMI**: si el dispositivo posee capacidades gNMI, puede emplearse para descubrir las interfaces del dispositivo.
        - IP: dirección IP del dispositivo (la misma IP especificada en P4Runtime).
        - Port: puerto del servidor gRPC (generalmente el mismo que P4Runtime).
- **piPipeconf**: Pipeconf a asociar con el dispositivo (programa P4).
    - **piPipeconfId**: identificador unívoco del Pipeconf en ONOS.
- **Ports**: información sobre interfaces del dispositivo, si no se utiliza gNMI.
    - **Number**: identificador numérico de puerto.
    - **Name**: Nombre de la interfaz.
    - **Enabled**: Estado de la interfaz.
    - **Removed**: Si la interfaz ha sido eliminada.
    - **Tipo**: Medio físico (cobre, fibra óptica, etc…)
    - **Speed**: velocidad en Mbps.

Dado que en P4Runtime el switch implementa el servidor y el controlador el cliente, el que debe iniciar la conexión es el controlador. Por ello, en entornos de emulación de redes como Mininet [58], hay que asegurarse de que el servidor gRPC del switch está completamente inicializado antes de que ONOS reciba el HTTP POST con la configuración. En caso contrario, la conexión puede fallar debido a que ONOS puede intentar establecerla antes de que el servidor inicie la escucha para nuevas conexiones. En el caso de BMv2, un retardo en función del número de interfaces entre el inicio del servidor y el envío de la configuración a ONOS suele funcionar bien.

Cuando ONOS recibe el mensaje de configuración, determina qué tipo de protocolos utilizar y cómo establecer conexión. Para un switch BMv2, ONOS extraerá la siguiente información sobre el mismo:

- Identificadores y nombres del dispositivo
- Driver a utilizar, BMv2 en este caso, que implica el uso de P4Runtime y la carga de programas P4 en formato JSON.
- IP y puerto donde se encuentra el servidor P4Runtime del switch.
- Programa P4 a cargar en el switch, identificado por un piPipeconfId.
- Interfaces del dispositivo, bien especificadas explícitamente, bien delegando su descubrimiento mediante gNMI.

Es habitual provisionar un programa P4 tras el establecimiento de conexión entre ONOS y el switch, es decir, se suele asumir que el switch no posee un programa P4 cargado previamente.

Una vez que el switch está conectado a ONOS, éste puede establecer reglas de flujo en el mismo (entradas en tablas P4), o recopilar estadísticas. Como el Pipeconf desarrollado implementa recolección de estadísticas en puertos mediante los contadores definidos en P4, ONOS por defecto realiza lecturas de los contadores cada 5 segundos, quedando estas disponibles para su consumo u observación por otras aplicaciones, o directamente por el usuario, por ejemplo, mediante consulta por comandos o GUI.

4.4.3.2 Descubrimiento de topología

Es muy habitual en SDN desarrollar aplicaciones agnósticas respecto de la topología de la red, que implementan algoritmos generalizables para cualquier tipo de topología. Por ello, los controladores poseen subsistemas de descubrimiento de topología. Por descubrir la topología

se entiende identificar los enlaces y puertos existentes entre switches y hosts. Suele emplearse el protocolo LLDP [59]. LLDP está basado en inyección de paquetes en switches desde el controlador, difusión de estos paquetes por la red, y reenvío de las copias difundidas de vuelta al controlador. El controlador puede conocer por dónde ha pasado el paquete LLDP que ha enviado, y determinar qué enlaces existen entre los switches a partir de esta información.

ONOS puede emplear LLDP para descubrir la topología de una red de switches BMv2 con el programa *hybrid.p4* gracias a dos componentes implementados:

- La operación Packet-In y Packet-Out implementada tanto en el plano de datos, como en el controlador ONOS.
- La tabla *l2_fwd,* uno de cuyos match fields es el campo Ethernet Type.

El subsistema LLDP de ONOS (*LLDP Provider* [60]) instala reglas de flujo solicitando el reenvío de tramas con Ethernet Type 0x88CC, inyectando paquetes en la red mediante Packet-Outs, y gestionando Packet-Ins, de los que extrae la información topológica. Como la aplicación LLDP de ONOS es agnóstica respecto del driver y protocolos Southbound del dispositivo, su comportamiento es idéntico en un switch OpenFlow y en un switch BMv2 con un programa P4 con las capacidades referidas. En concreto, la aplicación LLDP solicitará la instalación en todos los dispositivos de la red de la regla de flujo indicada.

A través de la traducción por el Pipeconf desarrollado, la regla de flujo se traducirá en una entrada en la tabla *l2_fwd* del programa P4, especificando el valor del campo match field Ethernet Type con 0x88CC (ignorando el resto de match fields, al ser ternarios), y la acción *send_to_cpu,* que identifica la operación Packet-In. Tras la recepción de un Packet-In por el cliente P4Runtime de ONOS, se ejecutará el método de mapeo para mensajes Packet-In definido en el Pipeconf, de forma que se traduzca la información generada por el programa P4 (cabecera Packet-In del programa P4) al formato estándar para Packet-Ins compartido por todos los protocolos Southbound de ONOS.

Por último, la aplicación LLDP enviará paquetes a la red mediante la interfaz Packet-Out generalista de ONOS. De forma análoga a la operación Packet-In, se ejecutará el método de mapeo del Pipeconf para envíar un mensaje Packet-Out entendible por el programa P4, en este caso una trama Ethernet encapsulada con la cabecera Packet-Out definida en P4.

Este ejemplo de uso de la aplicación LLDP ha ilustrado cómo se relaciona la interfaz Northbound con la interfaz Southbound mediante los mecanismos de traducción implementados en el Pipeconf. Asimismo, y muestra cómo en ONOS se pueden desarrollar aplicaciones para todo tipo de dispositivos sin necesidad de conocer sus características específicas (protocolos, drivers, etc…)

4.4.3.3 Descubrimiento de hosts

Por último, las aplicaciones SDN requieren conocer los hosts presentes en la red y su localización. LLDP no proporciona información relativa a hosts, dado que estos no entienden las tramas LLDP, y, por tanto, no participan de su lógica. Por ello, en SDN pueden emplearse técnicas de descubrimiento de hosts mediante interceptación de tramas ARP, o solicitud de asignación de direcciones mediante DHCP. Dado que las redes sobre las que se ha experimentado para este trabajo emplean direccionamiento IP estático, no se puede emplear mensajes DHCP para descubrir hosts.

ONOS posee un subsistema *HostProvider* [61] que le permite descubrir hosts tanto por ARP como por DHCP. *HostProvider* instala entradas en los switches de la red solicitando el reenvío (Packet-In) de tramas ARP y paquetes DHCP. ONOS también dispone de una aplicación que implementa un proxy ARP, *ProxyArp*, de forma que el controlador tiene la capacidad de gestionar toda comunicación ARP.

Sin embargo, el uso de *HostProvider* entra en colisión con ARP-Path, puesto que *HostProvider* reenvía toda trama ARP al controlador, mientras que ARP-Path las emplea para generar caminos entre dos hosts. Concretamente, *HostProvider* instalaría una entrada en la tabla *l2_fwd*, indicando que toda trama con Ethernet Type 0x806 se envíe al controlador, provocando que la trama ARP-Request no se difunda, y en consecuencia no se genere ningún camino. Por ello, es preciso buscar un enfoque nuevo que permita el descubrimiento de hosts.

Una alternativa sería emplear variantes del programa P4 en switches frontera (switches conectados a hosts), que clonasen las tramas ARP recibidas desde sus hosts directamente conectados: una trama sería procesada por el extern ARP-Path, y la otra sería enviada al controlador. La clonación de tramas puede realizarse mediante el extern *Clone* de *V1Switch*.

Existe un mecanismo más sencillo, que se corresponde con el implementado, basado en el servicio de ONOS *Network Config Host Provider*[62]. Este servicio emplea un mecanismo análogo al de *Network Configuration* para conocer los hosts, basado en el envío de la información de cada host en formato JSON mediante un HTTP POST a /onos/v1/network/configuration/hosts. El formato de la información básica a enviar sobre cada host es el siguiente:

- **Dirección MAC del host**
  - **Basic**: Información básica sobre el host.
    - **Name**: nombre del host.
    - **Locations**: Nombre de las interfaces e Ids de los switches a los que el host está conectado. En la implementación realizada, se asume que un host sólo está conectado a un switch.
    - **Ips**: Direcciones IP asignadas al host.

De esta forma, ONOS puede conocer los hosts presentes en la red sin necesidad de procesar tramas ARP. La principal desventaja de este método respecto a *HostProvider* es que por cada host que se adhiera a la red es necesario enviar un HTTP POST a ONOS, mientras que con *HostProvider* el descubrimiento sería transparente, mediante la interceptación de la primera trama ARP que envíe el host.

4.4.3.4 Aplicaciones necesarias para la gestión de una red BMv2

ONOS puede iniciarse con una serie de aplicaciones y subsistemas activados. Para switches basados en el target *ARP-P4* se deben incluir las siguientes:

- Driver P4Runtime
- Driver BMv2
- Subsistema gRPC
- LLDP Provider (opcional)
- Network Config Host Discovery (opcional)
- Arp4th.pipeconf (pipeconf correspondiente al programa *hybrid.p4*)

Tras la implementación de todas las operaciones definidas en este apartado, ONOS puede disponer de una red de switches híbridos ARP-Path/P4Runtime, que no requieren de su intervención para establecer comunicaciones entre los hosts de la red, pero de los que ONOS puede disponer de control absoluto si alguna aplicación lo establece. Por ejemplo, puede activarse la aplicación *Reactive Forwarding* [63], que implementa encaminamiento en capa 2 mediante reglas de flujo basadas en las direcciones MAC origen y destino, y el puerto de entrada, que estarían soportadas en la tabla *l2_fwd* del programa P4. Las reglas de flujo que instala esta aplicación tendrían prioridad sobre ARP-Path.

4.5 Extensión del programa P4 y provisión dinámica del mismo

En este apartado se aborda el desarrollo de un nuevo programa P4, *acl_hybrid.p4*, basado en *hybrid.p4*, que implementa dos funcionalidades adicionales: control de acceso, y soporte para ECMP. *Acl_hybrid.p4* incluye el contenido de *hybrid.p4* más el código adicional para las nuevas funcionalidades. Al igual que con *hybrid.p4*, es necesario desarrollar un nuevo Pipeconf, en el que además se implementará un componente Pipeliner, debido a la mayor complejidad del programa. Por último, se plantea dotar a ONOS de la capacidad para alternar de forma dinámica (operación SetForwardingPipelineConfig) entre ambos programas *hybrid.p4* y *acl_hybrid.p4*, sin perder el estado de reenvío, para lo cual se revisará la problemática asociada a esta operación en BMv2.

4.5.1 Programa acl_hybrid.p4

El programa *acl_hybrid.p4* incluye todos los elementos existentes en *hybrid.p4*, extendido con los siguientes elementos:

- Un Parser y Deparser que incluyen la desencapsulación y encapsulación de los protocolos IP, TCP y UDP.
- Una tabla *acl* destinada a bloquear o permitir el acceso a paquetes en base a campos de cabecera IP, TCP, y UDP.
- Una tabla *ecmp*, implementada mediante el extern *Action Selector*, que asocia un grupo ECMP de 16 bits a un conjunto de posibles caminos, de las cuales se seleccionará uno de forma dinámica mediante hash.

4.5.1.1 Soporte de cabeceras IP, TCP y UDP

Para implementar el Parser y Deparser, se deberán definir las cabeceras correspondientes a los protocolos IP, TCP y UDP. La cabecera IP definida en el programa es la siguiente (*acl_hybrid.p4*):

```
header ipv4_t {
    bit<4> version;
        bit<4> ihl;
        bit<8> diffserv;
        bit<16> totalLen;
        bit<16> identification;
        bit<3> flags;
        bit<13> fragOffset;
        bit<8> ttl;
        bit<8> protocol;
        bit<16> hdrChecksum;
        bit<32> srcAddr;
        bit<32> dstAddr;
}
```

De todos los campos anteriores, únicamente se utilizarán en el programa P4, los campos *protocol* (siguiente protocolo), *srcAddr* (dirección IP origen) y *dstAddr* (dirección IP destino). Para este programa, la especificación del resto de campos es opcional, y podrían haberse agrupado en dos campos, uno de 60 bits situado al inicio de la cabecera, y *hdrChecksum*.

Para el protocolo TCP, la definición de la cabecera P4 es la siguiente (*acl_hybrid.p4*):

```
header tcp_t {
        bit<16> srcPort;
        bit<16> dstPort;
        bit<32> seqNo;
        bit<32> ackNo;
        bit<4> dataOffset;
        bit<3> res;
        bit<3> ecn;
        bit<6> ctrl;
        bit<16> window;
        bit<16> checksum;
        bit<16> urgentPtr;
}
```

De los campos definidos, se utilizarán *srcPort* y *dstPort* (puertos origen y destino), y *ctrl* para poder observar flags como SYN, ACK, FIN, etc… Al igual que con IP, existen campos cuya especificación podría haberse omitido.

Por último, la definición correspondiente a la cabecera UDP se muestra a continuación (*acl_hybrid.p4*):

```
header udp_t {
        bit<16> srcPort;
        bit<16> dstPort;
        bit<16> length;
        bit<16> checksum;
}
```

De los campos anteriores, se utilizarán *srcPort* y *dstPort* (puertos origen y destino).

Un detalle relevante acerca de estos protocolos es que los tres implementan un campo Checksum para comprobación de errores. Como en el programa P4 no se realiza ninguna modificación sobre estas cabeceras no es necesario recalcularlos, pero si se modificara el paquete habría que recalcular los Checksum correspondientes para que un host o switch (no P4) no rechazase el paquete.

4.5.1.2 Implementación de control de acceso

Con los campos se señalados, se plantea la implementación de una tabla P4 que permita filtrar flujos identificados por sus direcciones IP origen y destino, el protocolo a continuación de IP, los puertos de capa 4 (TCP o UDP), el campo *ctrl* de TCP, y el puerto de entrada del paquete. La tabla únicamente se aplicará a datagramas IP, por lo que previamente a su ejecución, se comprobará que el paquete es IP mediante el bit de validez de la cabecera. A su vez, la tabla puede contener campos no contenidos en el paquete (por ejemplo, campo TCP ctrl aplicado a datagramas UDP), por lo que en este caso se ignorarán en la búsqueda.

La tabla se ha modelado con match fields de tipo ternario, lo que permite aplicar una entrada a una subred, y no especificar todos los match fields para una entrada. Por ejemplo, para bloquear

el tráfico ICMP proveniente de la subred 10.0.1.0/24, el mensaje P4Runtime especificaría en el campo de dirección origen el valor 10.0.1.0 con máscara de 24 bits a 1, y el valor 0x1 con máscara todo a 1s en el campo *protocol* de la cabecera IP.

Dado que TCP y UDP comparten mismo formato de puertos, en vez de especificar como match fields los correspondientes puertos TCP y UDP, y dado que ambas cabeceras son excluyentes, se han usado dos metadatos de 16 bits como match fields, que identifican de forma genérica los puertos de capa 4. Estos metadatos se han denominado *srcTpPort* y *dstTpPort*. El Parser asigna en estos metadatos el valor de los puertos origen y destino de los protocolos TCP o UDP, según corresponda, de forma que lo que se consigue es reducir en 32 bits el tamaño de la clave de la tabla, al pasar de 4 campos (puertos origen y destino TCP y UDP) a 2 (puertos de capa 4).

En cuanto a las acciones permitidas en la tabla, se han definido cuatro:

- *_nop*: Permitir el paso del paquete (ninguna acción).
- *_drop*: Descartar el paquete.
- *send_to_cpu*: Envíar paquete al cliente P4Runtime.
- *apply_meter*: Aplica un meter para realizar un conformado de tráfico.

La tabla incluye contadores directos y meters directos, explicados en el apartado 3.3.4. El uso de contadores directos permite conocer el volumen de tráfico cursado por entrada, y recoger estadísticas de flujos. Los meters directos utilizados son de tipo BYTES. La acción *apply_meter* realiza la lectura del meter asociado a la entrada que ha producido el hit. El valor leído será un identificador de color (verde, amarillo, o rojo), en función de la clasificación del paquete de acuerdo a los parámetros CIR, CBS, PIR y PBS previamente configurados por el cliente P4Runtime. En *acl_hybrid.p4*, el meter directo se emplea para descartar el paquete si el valor del meter es rojo. Una aplicación de los meter en el campo del control de acceso podría ser la detección y bloqueo de ataques DDoS. Por ejemplo, con la implementación realizada, el controlador ONOS podría detectar un volumen de tráfico anómalo dirigido hacia un servidor mediante la lectura de contadores, y configurar una o varias entradas con meters para disminuir la carga soportada sin necesidad de interrumpir el servicio bloqueando todo el tráfico.

En base a la información anterior, la tabla implementada en P4 es la siguiente (*acl_hybrid.p4*):

```
table acl {
        key = {
                hdr.ipv4.srcAddr: ternary;
                hdr.ipv4.dstAddr: ternary;
                hdr.ipv4.protocol: ternary;
                intrinsic_metadata.dstTpPort: ternary;
                intrinsic_metadata.srcTpPort: ternary;
                hdr.tcp.ctrl: ternary;
                standard_metadata.ingress_port: ternary;
        }
        actions = {
                _nop;
                send_to_cpu;
                _drop;
                apply_meter;
        }
        default_action = _nop;
        counters = acl_counter;
        meters = acl_meter;
```

```
        }
```

La acción por defecto es permitir, que siendo la más adecuada para un entorno de experimentación, puede no ser apta para un despliegue comercial. Por ello, en una implementación real, la acción por defecto podría ser denegar *(_drop*), teniendo en este caso el controlador que especificar de forma explícita los flujos de tráfico permitidos.

4.5.1.3 Soporte para ECMP

ECMP es una técnica que posibilita la distribución del tráfico entre dos hosts entre varios caminos de igual coste. WCMP es una variante de ECMP en el que cada posible camino tiene asignado un peso que determina su probabilidad de ser seleccionado. En el programa *acl_hybrid.p4* se ha implementado soporte para ECMP/WCMP por flujos. A nivel de programa P4, no existe diferencia alguna entre ECMP y WCMP, por ello se hablará de forma genérica de ECMP al analizar el código P4.

El soporte para ECMP está basado en el extern *Action Selector*. Recordando lo afirmado en el apartado 3.3.4., un *Actor Selector* puede asociar una entrada de una tabla a un conjunto de acciones (grupo de acciones). Cuando un paquete provoca un hit con una entrada asociada a un grupo, se seleccionará una acción del grupo mediante una operación hash de los campos especificados en la definición de la tabla. Para ECMP, cada acción indicaría el reenvío del paquete por un puerto distinto, quedando asociado cada puerto al siguiente nodo de un camino configurado por el cliente P4Runtime. Para que el tráfico de un mismo flujo de capa 4 siempre vaya por el mismo camino, el hash del Action Selector debe aplicarse sobre los 5 campos que lo identifican de forma unívoca: direcciones IP, puertos origen y destino, y campo de protocolo en la cabecera IP.

La instanciación del extern Action Selector es la siguiente:

```
//Action Selector para ECMP
action_selector(HashAlgorithm.crc32,16,8) ecmp_selector;
```

El código anterior especifica que se emplee un CRC de 32 bits como hash, con hasta 64 entradas, y 8 bits como índice de la salida del hash.

En el programa P4, un grupo de caminos está asociado a un identificador de 16 bits, *ecmp_group*, que se emplea como match key en la tabla que implementa ECMP. De esta forma, la definición de la tabla queda (*acl_hybrid.p4*):

```
table ecmp {
        key = {
                intrinsic_metadata.ecmp_group: exact;
                hdr.ipv4.srcAddr: selector;
                hdr.ipv4.dstAddr: selector;
                hdr.ipv4.protocol: selector;
                intrinsic_metadata.srcTpPort: selector;
                intrinsic_metadata.dstTpPort: selector;
        }
        actions = {
                _nop;
                forward;
        }
        implementation = ecmp_selector;
        size = 64;
}
```

La especificación de los match fields de tipo selector indica que son los campos a utilizar para el cálculo del hash en el *Action Selector*, no como match fields de una tabla. La match key queda determinada exclusivamente por el campo *intrinsic_metadata.ecmp_group*. Por último, se hace referencia en el apartado de propiedades adicionales al Action Selector declarado, *ecmp_selector*.

Como se observa, la creación de tablas P4 con selección de acción dinámica es bastante simple, recayendo la complejidad de su uso en el plano de control, pues es éste quien debe decidir qué grupos crear, qué miembros incluir en cada grupo, etc… A nivel de P4, la tabla anterior puede emplearse tanto para ECMP como WCMP, radicando la diferencia en este último caso en los pesos asignados por el plano de control para los distintos miembros de un grupo.

Para que a un flujo se le aplique ECMP, esto es, que atraviese la tabla *ecmp* definida en el programa P4, debe ser previamente asignado a un grupo *ecmp* (metadato *ecmp_group*). Esta asignación se realiza en la tabla *l2_fwd* ya existente en *hybrid.p4*, a la que ahora se añade una nueva acción: *set_ecmp_group*, que posee un parámetro de 16 bits que se asigna al metadato *ecmp_group*.

Por tanto, cuando el controlador ONOS desee aplicar ECMP a una comunicación entre dos hosts:

1. Calculará un conjunto de caminos de igual coste.
2. Creará grupos de Action Selector en los switches que pertenecen a cada camino, estableciendo como miembros de cada grupo las acciones de reenvío por puertos correspondientes a cada camino. Cada grupo de Action Selector tendrá un identificador que se corresponderá con el grupo ECMP.
3. Por cada switch de cada camino, ONOS establecerá una entrada en la tabla *l2_fwd* que asigne a un flujo el grupo ECMP deseado, y otra entrada en la tabla *ecmp*, que asocie el grupo ECMP a su grupo de Action Selector correspondiente.

Tras realizar los procedimientos anteriores para un origen y destino considerados, los posibles flujos (por ejemplo, conexiones TCP) entre dichos hosts se repartirán entre los caminos seleccionados, distribuyendo de esta manera el tráfico sin posteriores intervenciones del controlador.

La Figura 30 muestra la disposición de las tablas del programa *acl_hybrid.p4*. Las tablas P4 están coloreadas de azul, mientras que el extern ARP-Path lo está en verde:

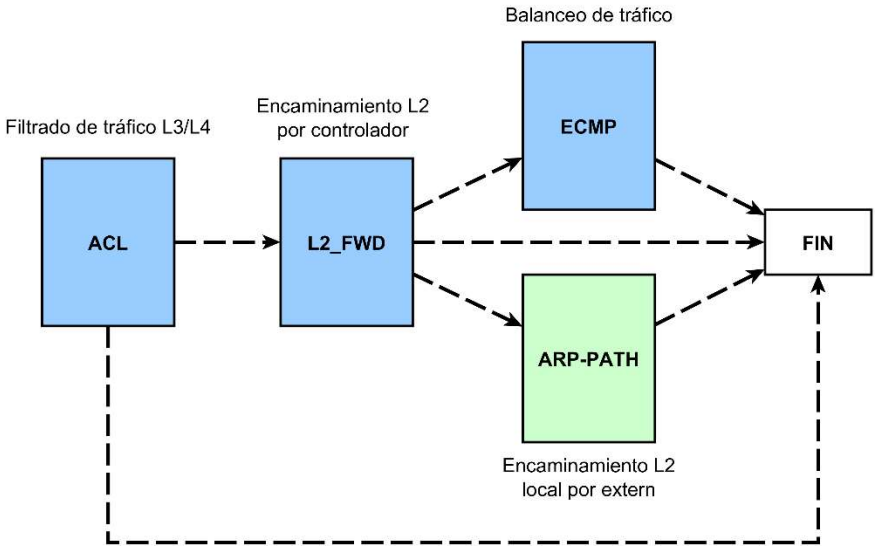


Figura 30 Disposición de tablas en acl_hybrid.p4

4.5.2 Operación SWAP en BMv2

La provisión dinámica de programa P4 puede ser identificada por varios términos, según el contexto: en P4Runtime queda asociada a la operación *SetForwardingPipelineConfig*, mientras que en BMv2 se conoce con el nombre de *SWAP*. Es decir, ambos conceptos son el mismo. Como ya se ha analizado a lo largo de este documento, la posibilidad de modificar sin interrupción de servicio el plano de datos abre las puertas a planteamientos innovadores en la operación de servicios de red.

Generalizando, podrían existir dos tipos de cambios sobre el programa P4. Uno sería una sustitución del programa existente por otro con el que no comparte ningún objeto en común. Para este caso, el estado de reenvío asociado al programa anterior se perdería, lo cual es lógico, ya que desaparecen los recursos que mantenían dicho estado de reenvío. El otro tipo sería una sustitución parcial, donde el programa antiguo y el nuevo comparten objetos P4, por lo que en este caso sí podría mantenerse el estado de reenvío, manteniendo las entradas de tablas y configuración de los objetos P4 comunes entre ambos programas.

4.5.2.1 Conservación del estado de reenvío de P4Runtime en operación SWAP

En el conjunto formado por BMv2 y la librería PI, el estado de reenvío se mantiene en dos lugares, en el propio servidor P4Runtime, y en los objetos de BMv2 que implementan la funcionalidad del programa P4. Existe una diferencia conceptual entre ambas entidades: el estado almacenado en el servidor P4Runtime sirve para registrar el estado de las interacciones entre el cliente y el servidor P4Runtime, mientras que el estado residente en los objetos BMv2 sirve para implementar a nivel lógico las operaciones deseadas. Una analogía sería una cuenta en una sucursal bancaria: el estado de la cuenta está representado por las anotaciones contables y por el dinero almacenado físicamente en el banco.

Ambos estados se interrelacionan en las operaciones *Write*, puesto que el servidor P4Runtime actualiza su estado interno (equivaldría a anotar los cambios solicitados por *Write*), e implementa en el plano de datos la configuración solicitada mediante la interacción con los objetos BMv2, empleando la API interna. Continuando con la analogía bancaria, un ingreso de

dinero se indica en la contabilidad, e implica también un ingreso físico de dinero en la cámara de la sucursal.

Sin embargo, las operaciones que no implican cambio de estado, no requieren ninguna consulta sobre el estado de los objetos BMv2, ya que el servidor P4Runtime puede responder a partir de su estado (en PI, técnicamente se realiza una consulta mínima sobre el estado de BMv2, orientada a comprobar que ambos estados, BMv2 y P4Runtime permanecen sincronizados). Para consultar el estado de una cuenta bancaria, bastaría con revisar los apuntes contables.

El modo *RECONCILE_AND_COMMIT* de la operación SetForwardingPipelineConfig, referenciado en el apartado 3.4.5 intenta preservar el estado de reenvío en los casos en los que sea posible.

La clase *bm::P4Objects* de BMv2 modela un contenedor que almacena la lógica del programa P4, implementada mediante una serie de objetos C++ ordenados según un flujo de ejecución. Por tanto, en BMv2, un objeto *P4Objects* es un programa P4. También se ha afirmado en el apartado 4.2 que un target BMv2 es una clase derivada de *bm::Switch*, y por ello, hereda los atributos de esta clase. *bm::Switch* contiene dos atributos clave para entender la implementación de la operación SWAP: *p4objects y p4objects_rt*. Estos dos atributos son punteros a un objeto *P4Objects*, y, con un programa P4 desplegado, apuntan al mismo objeto. El plano de datos de BMv2 referencia al programa P4 a través de *p4objects*, mientras que la interfaz del plano de control lo hace a través de *p4objects_rt*. Se trata, por tanto, de acceder al mismo programa P4 a través de punteros distintos. Sin embargo, durante un SWAP temporalmente apuntan a objetos distintos.

La operación SWAP no se diferencia técnicamente de la carga inicial de un programa P4, resumiéndose en la construcción de un objeto *P4Objects* que modele un programa P4 expresado en un JSON consumido, y su posterior asignación al atributo *p4objects* de *bm::Switch* de forma que el plano de datos quede implementado por dicho objeto. Está estructurada en dos fases, identificadas por dos métodos:

- ***bm::Switch::load_new_config()***: Se encarga de leer el fichero JSON del programa P4, crear un objeto *P4Objects* a partir de la lógica del fichero, y asignar este objeto *P4Objects* al puntero *p4objects_rt*.
- ***bm::Switch::swap_configs():*** Tras procesar todos los paquetes existentes en el sistema y bloquear la recepción de nuevos paquetes, asigna el objeto apuntado por *p4objects_rt* a al puntero *p4objects*, completando el procedimiento del programa P4.

En el lapso de tiempo existente entre la ejecución de ambos métodos se tiene que el puntero *p4objects_rt* apunta al nuevo programa P4, mientras que *p4objects* apunta al antiguo. En este intervalo temporal coexistirían dos programas P4: el nuevo, apuntado por *p4objects_rt*, y el antiguo, apuntado por *p4objects*. Esto implica que las operaciones del plano de control actuarían sobre el nuevo programa, a la vez que los paquetes se siguen conmutando a través del antiguo, por lo que se puede copiar el estado de reenvío durante este intervalo de tiempo del antiguo programa al nuevo sin interrumpir el servicio de forma brusca.

La Figura 31 representa el proceso de SWAP referenciado anteriormente en BMv2:

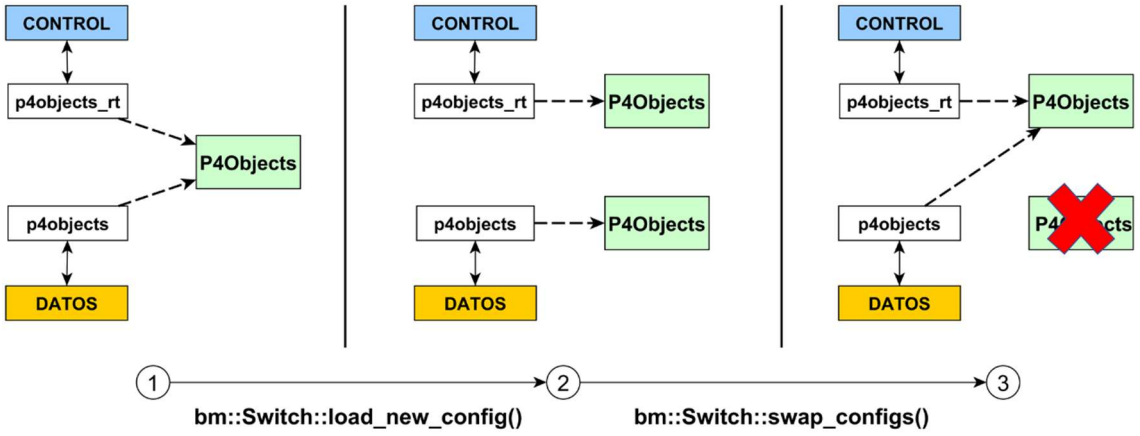


Figura 31 Proceso de SWAP en BMv2

Un paquete en BMv2 está asociado a dos clases: *bm::Packet*, que contiene el contenido original del paquete, y *bm::PHV*, que es una clase que contiene una representación estructurada del paquete de acuerdo a las cabeceras y metadatos definidos en el programa P4. Debido a que las cabeceras y metadatos entre dos programas P4 pueden ser distintos, BMv2 genera un nuevo modelo de PHV por programa P4, no pudiendo existir dos modelos PHV distintos a la vez. Por ello, BMv2 necesita procesar todos los paquetes existentes para poder destruir los PHVs correspondientes al programa P4 antiguo, y así poder generar un formato adecuado al nuevo programa.

Como entre las ejecuciones entre *load_new_config()* y *swap_configs()*, *p4objects_rt* y *p4objects* apuntan a objetos distintos, el servidor P4Runtime puede implementar la sustitución del programa P4 conservando el estado de reenvío. El servidor P4Runtime, tras ordenar la generación del nuevo programa P4 (*load_new_config()*), puede establecer el estado de reenvío del nuevo programa (*RECONCILE_AND_COMMIT*), leyendo su configuración de estado, e implementándola en el objeto *P4Objects* del nuevo programa, mediante su interacción a través del puntero *p4objects_rt*, al que se refieren todas las operaciones del plano de control como ya se ha explicado.

De esta forma cuando *swap_configs()* direccione *p4objects* al objeto apuntado con *p4objects_rt*, el plano de datos lo implementará el nuevo programa P4, conservándose la configuración de los objetos P4 comunes entre ambos programas.

4.5.2.2 Conservación del estado de reenvío ARP-Path en operación SWAP

Sin embargo, todo lo anterior es solamente es aplicable al estado de reenvío gestionado por el servidor P4Runtime, lo que excluye al estado de ARP-Path gestionado por el extern. Cuando se genera un nuevo programa P4 que hace uso del extern ARP-Path, se crea un nuevo objeto extern, destruyéndose el antiguo. La clave para conservar el estado de ARP-Path consiste en la compartición de un mismo mapa de memoria entre ambos objetos extern. Para ello se puede hacer uso de los atributos *static* de C++. La calificación de un atributo de una clase como *static* permite que el atributo pase a depender de la clase, y no del objeto, de forma que todos los posibles objetos de esta clase compartan el atributo.

Calificando el mapa de memoria de ARP-Path como atributo estático hace que quede asociado a la clase *ArpPath* que implementa el extern, y no al objeto extern de un programa específico.

Así, al generarse un nuevo objeto extern con el nuevo programa P4, el mapa de memoria estará compartido con el antiguo, conservándose tras la destrucción del extern antiguo. Una restricción de esta solución es que solamente puede haber una única instancia del extern *ArpPath* en un programa P4, ya que múltiples instancias compartirían estado. Sin embargo, esto no es un problema en la práctica, al no tener sentido la existencia en un programa de múltiples instancias del extern *ArpPath*.

Es probable que cuando se implemente el uso de externs en P4Runtime, se proporcionen primitivas para conservar la configuración de objetos extern no estándar, como es el caso de ArpPath. Posiblemente habría que implementar algún método de copia de estado entre externs del mismo tipo, que pueda ser llamado por el servidor P4Runtime de forma automática cuando se desee cambiar el programa P4 conservando el estado de reenvío (RECONCILE_AND_COMMIT).

4.5.3 Desarrollo de Pipeconf para *acl_hybrid.p4*

Al igual que con el programa *hybrid.p4*, es necesario desarrollar un nuevo Pipeconf en ONOS para que el controlador pueda interaccionar con los recursos expuestos por *acl_hybrid.p4*. No se puede reutilizar el Pipeconf ya desarrollado, puesto que no expone los nuevos recursos proporcionados, como por ejemplo el control de acceso implementado en la tabla *acl*.

Al suponer *acl_hybrid.p4* una extensión del programa *hybrid.p4*, gran parte del mapeo ya estará realizado, de forma que únicamente habrá que mapear las entidades correspondientes a los nuevos recursos. De las dos nuevas tablas desarrolladas, *acl* y *ecmp*, únicamente se mapeará *acl* para compatibilizar el control de acceso con las entidades propias de ONOS, de cara a proporcionar a las aplicaciones *Pipeline-Agnostic* una interfaz mediante la operación *filter* framework *FlowObjective*, para lo cual es necesario implementar un Pipeliner específico.

La Tabla 2 ilustra los mapeos realizados para el nuevo Pipeconf:

Tabla 2 Mapeo entre P4 y ONOS para acl_hybrid.p4

ONOS	***P4***
ID DE TABLA	**TABLA P4**
0	IngressImpl.l2_fwd
1	IngressImpl.ACLPipeline.acl
CRITERION TYPE	**P4 MATCH FIELD**
IN_PORT	standard_metadata.ingress_port
ETH_DST	hdr.ethernet.dstAddr
ETH_SRC	hdr.ethernet.srcAddr
ETH_TYPE	intrinsic_metadata.l2_next_header
IPV4_SRC	hdr.ipv4.srcAddr
IPV4_DST	hdr.ipv4.dstAddr
IP_PROTO	hdr.ipv4.proto
TCP_FLAGS	hdr.tcp.ctrl
INSTRUCTION	**ACCIÓN P4**
-	IngressImpl.drop_
OUTPUT(CONTROLLER)	IngressImpl.send_to_cpu
OUTPUT(puerto físico)	IngressImpl.forward(puerto físico)
NOACTION	IngressImpl.drop_

Nótese como los puertos TCP y UDP no se han mapeado, ya que para mapear debe existir una correspondencia 1:1, que no existe para los metadatos *srcTpPor*t y *dstTpPort* (sería una correspondencia 1:2).

Por otra parte, los recursos asociados a ECMP no se mapean, requiriendo el programador de las aplicaciones un conocimiento del programa *acl_hybrid.p4* para gestionar de forma correcta los recursos. De esta forma, la implementación de ECMP en *acl_hybrid.p4* solamente sería compatible con aplicaciones *Pipeline-Aware*.

Este apartado se estructura de la siguiente forma: primero se aborda el diseño del Pipeliner para *acl_hybrid.p4*, posteriormente la provisión dinámica del *acl_hybrid.p4*, el diseño de la REST API para control de acceso, y por último el análisis de la creación de grupos en *Action Selectors* mediante el servicio de grupos de ONOS.

4.5.3.1 Diseño del Pipeliner

El Pipeliner es un componente asociado a un driver concreto, que se encarga de convertir *Flow Objectives* en *Flow Rules* asociados a tablas concretas. Se podría resumir su funcionalidad en traducir una especificación genérica de comportamiento de la red para un switch en reglas de flujo de tablas concretas en un dispositivo. Un *Pipeliner* es específico de un driver porque su misión es proporcionar una interfaz genérica (*Pipeline-Agnostic*) que abstraiga respecto de las posibles variantes de Pipeline específicos de los dispositivos, de forma que todos ellos puedan ser controlados de la misma forma mediante el framework *flowObjective*.

Dado que un programa P4 puede definir un conjunto de tablas arbitrarias, cada una con su propósito y con match fields específicos, un componente *Pipeliner* puede abstraer a aplicaciones del conocimiento específico del programa P4.

Para el caso en estudio, *acl_hybrid.p4*, se tiene que el programa tiene tres tablas P4: *acl* (control de acceso a partir de capa 3), *l2_fwd* (encaminamiento en capa 2), y *ecmp* (distribución de tráfico entre caminos). Como la tabla *ecmp* solamente se plantea para su uso con aplicaciones *Pipeline-Aware*, las aplicaciones *Pipeline-Agnostic* operarán sobre las tablas *acl* y *l2_fwd* .

El Pipeliner para *acl_hybrid.p4* debe cumplir con los siguientes requisitos:

- Para encaminamiento en capa 2, redirigir a la tabla *l2_fwd* .
- Para filtrar o permitir tráfico en capa 2, redirigir a la tabla *l2_fwd* .
- Para filtrar o permitir tráfico en capa 3 o capa 4, redirigir a la tabla *acl*.

Las operaciones disponibles en *Flow Objective* son dos: *forward* (encaminar tráfico) y *filter* (denegar o permitir tráfico). El soporte para la operación *forward* debe dirigirse incondicionalmente a la tabla *l2_fwd* , dado que la tabla *acl* no tiene asociada ninguna acción de reenvío.

Para la operación *filter*, existen tres casos en función de los criterios selectores de tráfico (match fields en terminología P4):

- Si los campos selectores se tráfico se refieren a capa 2, se redirige a la tabla *l2_fwd*.
- Si los campos selectores de tráfico se refieren a capa 3 o 4, se redirige a la tabla *acl*.
- Si los campos selectores combinan campos de capa 2 y campos de capa 3 o 4, la operación no está soportada.

Por ello, la tabla sobre la que se instalará la entrada de flujo se seleccionará en base a los criterios seleccionados.

Por otra parte, no existe correspondencia directa entre los match fields *srcTpPort* y *dstTpPort* (puertos de capa 4) y criterios selectores en ONOS, ya que cada uno de los match fields se correspondería con dos posibles alternativas, el puerto UDP, o el puerto TCP.

Por ello, criterio selector de ONOS para un puerto TCP/UDP origen/destino debe traducirse en dos match fields de P4, el puerto genérico de capa 4 origen/destino, y el match field hdr.ipv4.protocol con el valor del protocolo que corresponda (TCP o UDP).

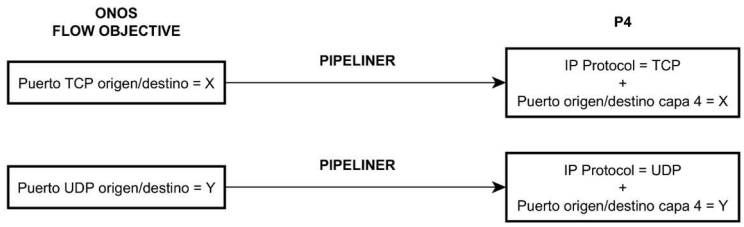


Figura 32 Mapeo en puertos de capa 4 en el Pipeliner para acl_hybrid.p4

Con el Pipeliner implementando estas funcionalidades, las aplicaciones *Pipeline-Agnostic* pueden filtrar tráfico con el programa *acl_hybrid.p4*, sin necesidad de conocer en qué tabla deben instalar las reglas de flujo que componen la política a implementar.

4.5.3.2 Provisión dinámica de programa P4

En este proyecto se ha diseñado un sistema capaz de cambiar el programa P4 del target *ARP-P4* entre *hybrid.p4* y *acl_hybrid.p4*. Sin embargo, la capacidad de provisionar un programa P4 una vez se ha desplegado previamente un programa inicial no está soportada actualmente en ONOS, aunque se plantea su inclusión a medio plazo. Por ello, ONOS asume que el programa P4 desplegado en un dispositivo se mantendrá invariante.

A pesar de ello, ONOS dispone de las capacidades para provisionar un programa P4 para sustituir uno previo, ya que en P4Runtime no hay diferencia entre la provisión inicial de un programa, y la sustitución de otro, empleándose en ambos casos la operación SetForwardingPipelineConfig.

ONOS, tras establecer la conexión gRPC con un switch, provisiona el programa P4 en dos fases:

1. A nivel del dispositivo mediante la operación *SetForwardingPipelineConfig*, a través una llamada al servicio PiPipeconfService.
2. A nivel interno, para asignar un Pipeconf a un dispositivo, y asociar de esa forma todos los comportamientos definidos en el Pipeconf al dispositivo sobre el que se ha provisionado el programa P4, también a través de PiPipeconfService.

Mediante las llamadas a los métodos de PiPipeconfService que implementan los dos procedimientos anteriores, puede desplegarse un programa P4 sobre un switch que ya disponga de uno. El mecanismo no es directo, puesto que se requiere una modificación sobre el driver asignado sobre el switch para eliminar el Pipeconf previo, pero es completamente funcional.

4.5.3.3 REST API para implementar control de acceso

ONOS ofrece facilidades para crear aplicaciones con interfaces REST, de forma que puedan ser controladas por componentes externos. Se ha creado una REST API específica para el Pipeconf

de *acl_hybrid.p4*, que posee dos funcionalidades: provisionar o eliminar *acl_hybrid.p4* para un dispositivo, e instalar reglas de filtrado en todos los switches de la red que ejecuten el programa acl_hybrid.p4.

La aplicación REST escucha en la ruta /onos/v1/p4acl y responde a métodos HTTP POST y DELETE. Existen dos rutas derivadas de la anterior, cada una dedicada a un propósito distinto.

La ruta /onos/v1/p4fw/deploy se emplea para la sustitución del programa P4. Un POST a esta ruta sirve para desplegar el programa *acl_hybrid.p4* en el switch cuyo identificador esté contenido en el cuerpo del mensaje. Analógicamente, un método DELETE despliega el programa *hybrid.p4* en el switch indicado.

Por otra parte, la ruta /onos/v1/p4fw/rules permite instalar reglas de filtrado de tráfico (aceptar/rechazar) en todos los switches de la red que tengan instalado *acl_hybrid.p4*. Un método POST instala la regla de filtrado contenida en el cuerpo del mensaje HTTP, mientras que un método DELETE elimina la regla contenida en su cuerpo. El formato de la regla de filtrado es un JSON con los siguientes campos:

- ***srcIp***: Dirección IP origen.
- ***dstIp***: Dirección IP destino.
- ***ipProto***: Siguiente protocolo tras IP.
- ***dstTpPort***: Puerto destino de capa de transporte.
- ***srcTpPort***: Puerto origen de capa de transporte.
- ***srcMac***: Dirección MAC origen.
- ***dstMac***: Dirección MAC destino.
- ***srcMac***: Dirección MAC origen.
- ***ethType***: Cabecera a continuación de Ethernet.
- ***inPort***: Puerto de entrada del paquete.
- ***prio***: Prioridad de la entrada

Todos los campos anteriores son opcionales, con la restricción de que no se pueden combinar campos referidos a capas 3 y 4, con campos referidos a capa 2, por el propio diseño del programa P4.

Se han desarrollado sendos scripts bash, *p4fw-deploy.sh* y *p4fw-acl.sh* que emplean la REST API para provisionar programas P4 e introducir reglas de filtrado, respectivamente. De esta forma, por ejemplo, se puede emplear el script *p4fw-deploy.sh* para sustituir el programa P4 de un switch, indicando al script la dirección IP de la instancia de ONOS maestro del switch, y el identificador del switch.

Sin embargo, la REST API y los scripts que hacen uso de ella se han planteado como una mera prueba de concepto para ilustrar como desde un agente externo a ONOS se puede modificar el estado de reenvío de un switch P4. A este respecto, REST API es solo una de las interfaces Northbound de las que dispone ONOS, entre las cuales debe destacarse gRPC, que a día de hoy cuenta con soporte parcial. El concepto de gRPC como API Northbound es completamente distinto al de interfaz Southbound. En el primer caso representa como interfaz de comunicación entre una aplicación ONOS y un componente externo, mientras que en el segundo caso representa una interfaz entre la capa de infraestructura (switches) y el controlador.

4.5.3.4 Gestión de grupos de Action Selectors en ONOS

ONOS posee una API para la gestión de grupos de Action Selectors mediante el servicio de grupos, Group Service. Group Service permite implementar acciones que afectan a un conjunto de grupos, modelado de forma independiente a los protocolos Southbound.

Un grupo en ONOS viene definido por:

- El identificador del dispositivo donde se instale (DeviceId).
- El tipo de grupo: *SELECT, INDIRECT, ALL, FAILOVER*. Para modelar un Action Selector de P4Runtime el tipo a utilizar es *SELECT*.
- Un GroupBucket compuesto por las acciones miembros del grupo.
- Un identificador de grupo (GroupId), que identifica un grupo de forma unívoca dentro de los grupos de un Action Selector de un switch.
- Una cookie (GroupKey), que para P4Runtime se determina a partir del identificador del Action Selector, la tabla donde se utilice, y el identificador de grupo.
- El ID de aplicación ONOS que instale el grupo.

Para el ejemplo ECMP contemplado, una vez se han calculado los caminos de igual coste sobre los que balancear el tráfico, por cada switch distinto que forme parte del conjunto de caminos, se tendrá que configurar un grupo, e instalar una regla de flujo. Por ejemplo, supongamos que por un switch pasan cuatro caminos, de los cuales el siguente salto es distinto entre ellos, y consecuentemente, en este switch concreto, cada posible camino queda identificado por un puerto de salida. Para implementar ECMP según el programa *acl_hybrid.p4* habría que:

1. Crear un grupo cuyos miembros sean acciones *forward* con cada uno de los posibles puertos de salida. Como hay cuatro puertos, el grupo tendrá cuatro miembros. El ID de grupo asignado al crearlo será el identificador de grupo ECMP (*ecmp_group* en el programa P4).
2. Instalar una entrada para la tabla *ecmp,* especificando como match field el identificador de grupo ECMP, y como acción, la redirección al grupo Action Selector identificado por el grupo ECMP. Nótese como el valor a introducir tanto en la clave de entrada como en la acción es el grupo ECMP.
3. Instalar una entrada en la tabla *l2_fwd*, especificando los match fields a utilizar (direcciones MAC origen y destino, Ethernet Type, o puerto de entrada) y asignando la acción *set_ecmp_group*, pasando como parámetros el identificador de grupo ECMP.

4.6 Visualizador de caminos implementado sobre header-stacks

P4 ofrece una programabilidad nunca antes vista en SDN, pero, sin embargo, el trabajo explicado hasta ahora no representa nada que no se pueda implementar con otras tecnologías más asentadas, como, por ejemplo, OpenFlow. Por ello, el propósito de este apartado es presentar una aplicación que haga uso de funcionalidades de P4 no implementables por tecnologías actuales.

En este apartado se va a presentar el diseño de una aplicación de visualización de caminos entre hosts implementada sobre la GUI de ONOS. La GUI de ONOS es capaz de señalar de forma gráfica, sobre un dibujo de la topología de la red, los enlaces sobre los que discurre tráfico (medido en bytes o paquetes), y los caminos asociados a Intents. Asimismo, es posible inferir el camino que atraviesan los paquetes de un determinado flujo que cuente con contadores de bytes o paquetes. De esta forma, si en un switch el contador de bytes o paquetes para una regla

de flujo instalada se incrementa, se puede deducir que el flujo considerado ha atravesado el switch durante el periodo entre lecturas del contador.

Por tanto, si para una tabla P4 destinada a encaminar paquetes se especifican contadores directos que puedan ser leídos por ONOS, se podría visualizar sobre la GUI el camino que sigue el flujo.

Sin embargo, ONOS no conoce el estado de reenvío asociado a ARP-Path en el switch desarrollado, al no tener posibilidad de acceder a él. Por esta razón, los caminos ARP-Path son desconocidos para ONOS, y no se pueden representar sobre la GUI a partir de datos estadísticos. Esto plantea un problema para la verificación del funcionamiento del extern ARP-Path, puesto que, en principio, para identificar los caminos que se forman en la red habría que examinar la información de depuración en todos los switches de la red y trazar los caminos a partir de ella.

Por esta razón, se plantea desarrollar una aplicación que pueda dibujar sobre la GUI de ONOS el camino que sigue un flujo en tiempo real, independientemente de si ese a ese flujo se aplican entradas P4Runtime o ARP-Path. El modelo sobre el que se ha implementado la aplicación está basado en header-stacks.

El uso de header-stacks, o pila de cabeceras, se emplea para etiquetar la ruta sobre el paquete, haciendo que cada switch en el camino inserte una etiqueta con su identificador de dispositivo. De esta forma, ONOS recibirá un paquete que contendrá las etiquetas de todos los switches que ha atravesado. Esta información será consumida por una aplicación que resalte sobre la topología representada en la GUI de ONOS los enlaces correspondientes a la ruta seguida.

Sin embargo, cada switch puede incluir más información aparte de su identificador en la cabecera que inserta en cada paquete, como, por ejemplo, datos acerca de su estado (longitud de colas, marca de tiempo en que se recibió el paquete, etc…). In-Band Telemetry (INT) [64] es un framework P4, actualmente en desarrollo por el P4 Applications Working Group [65], que implementa un sistema de recolección y reporte del estado de la red a través del plano de datos, mediante el uso de cabeceras. En este trabajo se ha implementado un sistema inspirado en INT, a mucho menor nivel, para recolectar datos acerca de la red no accesibles por ONOS. MRI [66] es otra aplicación P4 basada en la misma filosofía de INT, planteada como prueba de concepto, con la que existen similitudes respecto a la aplicación desarrollada, pero de una complejidad considerablemente menor al no ser transparente a los hosts.

Por tanto, la aplicación de visualización de caminos está formada por dos componentes:

- Una extensión de *hybrid.p4*, implementada sobre el bloque de control de egreso, que realice el etiquetado de paquetes mediante header-stacks. La funcionalidad desarrollada se denominará *ARP-Path Telemetry* (APT).
- Una aplicación ONOS que reciba paquetes etiquetados, los desencapsule, e interaccione con la GUI de ONOS para resaltar el camino que siguen los paquetes del flujo examinado.

4.6.1 Etiquetado de paquetes en P4 mediante header-stacks

La lógica P4 correspondiente al etiquetado de paquetes se ha implementado tanto en *hybrid.p4*, como en *acl_hybrid.p4*, en el bloque de control de egreso. La razón para ello es la funcionalidad genérica de este bloque, pensada para realizar operaciones adicionales sobre el paquete tras haberse fijado el puerto de salida en el bloque de control de ingreso.

El sistema implementado consiste en el etiquetado de paquetes de un flujo definido en capa 2 por cada switch a lo largo de su ruta entre los hosts origen y destino. El etiquetado consiste en la inclusión de cabeceras adicionales entre Ethernet y su carga útil (datagrama IP, trama ARP, etc…). Estas cabeceras adicionales contienen datos acerca del estado de los switches que ha atravesado el paquete. Esta información adicional será enviada a un cliente P4Runtime para su consumo, que en este caso sería un controlador ONOS. El host destino no puede recibir un paquete etiquetado, ya que no sabría extraer la carga útil (las etiquetas serían un protocolo desconocido).

Por ello, se ha diseñado un sistema de etiquetado transparente a los hosts mediante la eliminación del etiquetado del paquete reenviado desde el switch frontera destino hacia el host. Es decir, el host destino recibe el paquete tal cual fue enviado por el host origen, de ahí el calificativo de transparente.

Por otra parte, el etiquetado no se aplica a todo paquete, sino que únicamente se aplica sobre los flujos indicados explícitamente por el controlador, ya que el uso de esta funcionalidad se traduce en mayor tiempo de procesamiento y longitud de paquetes.

Conceptualmente, existen tres tipos de switches en la red, atendiendo a la funcionalidad en su etiquetado:

- **Switch frontera origen**: Marca el paquete para ser etiquetado por los subsecuentes switches e introduce la primera etiqueta.
- **Switch intermedio**: Añade una etiqueta si el paquete ha sido marcado para ser etiquetado.
- **Switch frontera destino**: Etiqueta el paquete, y realiza una copia de él. El paquete original se reenviará al host destino sin las etiquetas. El paquete copiado se reenviará al cliente P4Runtime (ONOS), eliminando la carga útil del paquete original, pero conservando las etiquetas.

El etiquetado se implementa mediante la definición de dos cabeceras: *etiqueta APT* y *preámbulo APT*. La *etiqueta APT* modela la etiqueta añadida por un switch a un paquete. Las *etiquetas APT* siempre van precedidas de una cabecera de tipo *Preámbulo APT,* existiendo una única instancia de este tipo de cabecera en el paquete. El funcionamiento del etiquetado de un paquete a lo largo de la ruta entre origen y destino se muestra en la Figura 33:

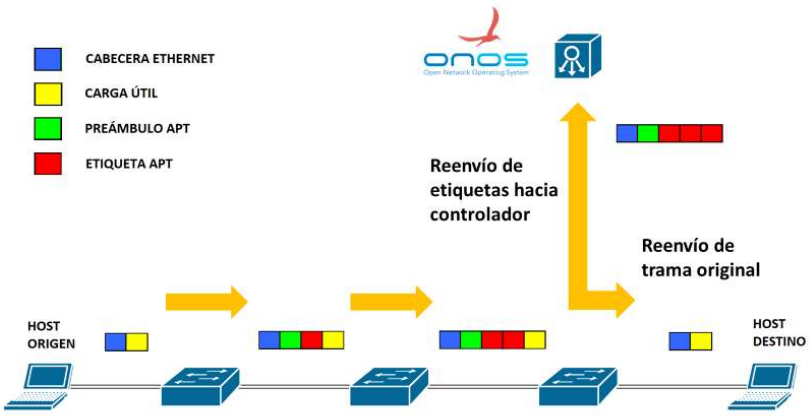


Figura 33 Funcionamiento de ARP-Path Telemetry con un controlador ONOS

Para discriminar entre los flujos que la red debe etiquetar y los que no, se va a emplear una nueva tabla P4, *tag_packet*, donde los flujos se identificarán por las direcciones MAC origen y destino. Esta tabla únicamente contendrá entradas en los switches frontera, y tendrá dos acciones permitidas: etiquetar paquete, y desetiquetar paquete. Para un flujo entre dos hosts que se desee etiquetar, se tendría que en el switch frontera origen existiría una entrada con la acción de etiquetar, y en el switch frontera destino, otra entrada con la acción de desetiquetar, haciendo referencia ambas a las mismas direcciones MAC.

En principio, el cliente P4Runtime tiene la capacidad de recibir las etiquetas de todos los paquetes enviados por el host origen. Sin embargo, esto puede no ser deseable, ya que el cliente P4Runtime podría no tener la capacidad o la necesidad de procesar todos y cada uno de los paquetes de un flujo, además de la pérdida de rendimiento debida a sobrecarga introducida en la red. Por ejemplo, el cliente P4Runtime podría necesitar solamente una etiqueta cada segundo, y por ello, debe ser capaz de configurar cuántos paquetes desea recibir en un periodo de tiempo (frecuencia de muestreo). El extern de PSA *Packet Digest* (véase apartado 3.3.4) representa la solución más óptima, porque permite configurar la frecuencia y número de notificaciones a enviar al plano de control.

Sin embargo, nuevamente es una funcionalidad sin implementación en P4Runtime ni en ONOS a día de hoy, por lo que se ha recurrido a una solución basada en mensajes Packet Out, meters directos, y los extern *clone* y *truncate* de *V1Switch*. La idea sería configurar la frecuencia de muestreo mediante un meter inddirecto de paquetes en el switch frontera origen, a través de los parámetros CBS, PBS, PIR y CIR. Si el color resultante de la lectura del meter fuera rojo, el paquete no se marcaría para ser etiquetado. Por otra parte, en el switch frontera destino, la acción de desetiquetar incluiría la clonación del paquete original, el truncamiento de la longitud de la copia dejando únicamente la cabecera Ethernet y las etiquetas, y el reenvío de ésta al plano de control mediante un mensaje Packet-Out.

4.6.1.1 Definición de cabeceras de etiquetado en P4

La etiqueta APT se define en P4 mediante una cabecera de tipo *apt_header*. Se emplea un sistema de header-stacks o pila de cabeceras para la adición de sucesivas cabeceras a lo largo de la ruta. Es decir, un paquete de un flujo monitorizado tendrá tantas cabeceras *apt_header* como switches haya atravesado, con un máximo de 10. En P4, un header-stack puede entenderse como un vector de cabeceras del mismo tipo que proporciona facilidades para la gestión del conjunto tanto en los Parsers como en los bloques de control.

La cabecera *apt_header* presenta la siguiente definición en P4 (*hybrid.p4*/*acl_hybrid.p4*):

```
header apt_t {
   bit<32>device_id;
   bit<8> in_queue_ocup;
   bit<8> eg_queue_ocup;
   bit<32> timedelta;
   bit<32> num_entries;
}
```

El significado de cada uno de sus campos es el siguiente:

- ***Device_id***: Identificador del dispositivo. En BMv2 es un dato de hasta 64 bits, pero en la implementación realizada se reduce a 32.

- ***In_queue_ocup***: Porcentaje de ocupación del buffer de procesado (cola previa al bloque de Ingreso) (Figura 25).
- ***Eg_queue_ocup***: Porcentaje de ocupación del buffer de transmisión (cola posterior al bloque de Egreso) (Figura 25).
- ***Timedelta***: Latencia en nanosegundos del bloque de control de Ingreso.
- ***Num_entries***: Número de entradas ARP-Path existentes en el mapa de memoria del extern ARP-Path, incluyendo tanto las activas como aquellas caducadas (que eventualmente serán eliminadas).

La Figura 34 muestra gráficamente el formato de la cabecera propuesta:

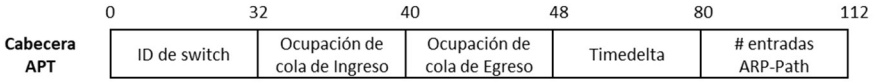


Figura 34 Formato de cabecera APT

Por otra parte, la definición en P4 del *Preámbulo APT* se realiza mediante la cabecera *apt_preamble* (*hybrid.p4/acl_hybrid.p4*):

```
header apt_preamble_t {
   bit<16> next_hdr;
   bit<8> hdr_count;
}
```

Por un lado, en el campo *next_hdr* almacena el Ethernet Type referido a la carga útil del paquete, ya que el Ethernet Type de la cabecera original se establece con el valor 0xDEAD para indicar que va etiquetado, de forma que el Parser desencapsule la cabecera *Preámbulo APT* y el header-stack de cabeceras *APT*.

Por otra parte, en el campo *hdr_count* se almacena el número de cabeceras *APT* existentes en el header stack (cuántas veces ha sido etiquetado el paquete).

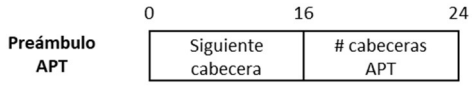


Figura 35 Formato de cabecera del preámbulo APT

Una representación visual del orden de las cabeceras para una trama Ethernet etiquetada por N switches es la siguiente:

Figura 36 Trama Ethernet etiquetada por N switches

4.6.1.2 Obtención de los valores a incluir en la etiqueta

Los valores a incluir en los campos de la etiqueta APT se obtienen de metadatos definidos en el programa P4. Ahora bien, desde el programa P4 no se pueden obtener algunos de dichos valores, por lo que es preciso obtenerlos a partir de código implementado en el target (C++) y cargarlos en la estructura PHV del paquete (véase apartado 4.5.2) para que en P4 puedan visualizarse como metadatos. Los procedimientos utilizados para obtener los valores son:

- **ID de dispositivo**: Llamada al método *get_device_id()* de bm::Switch.

- **Ocupación de colas**: Obteniendo el número de elementos en contenidos en los objetos bm::Queue que modelan las colas en tanto por cien.
- **Timedelta**: Calculando el tiempo de ejecución del bloque de Ingreso para un paquete.
- **Número de entradas ARP-Path**: Mediante una llamada al método num_entries() del extern dentro del programa P4.

4.6.1.3 Parser para procesamiento de etiquetas

Es necesario realizar una modificación del Parser en ambos programas, *hybrid.p4* y *acl_hybrid.p4*, para proporcionarle la capacidad de desencapsulación de las cabeceras *Preámbulo APT* y *APT*. Los cambios son los siguientes:

- Tras desencapsular la cabecera Ethernet, se desencapsula el *Preámbulo APT* si el valor de Ethernet Type es 0xDEAD.
- Al desencapsular el *Preámbulo APT*, se almacena en el metadato *l2_next_header* su campo next_header, y en un metadato de uso temporal, *hdr_count*, el número de header-stacks referenciado en su campo *hdr_count*.
- Mientras el metadato *hdr_count* sea distinto de 0, se desencapsula una cabecera *APT*, decrementándose el valor del metadato por cada cabecera *APT* procesada. De esta forma, cuando tome el valor de 0, se habrán procesado todas las etiquetas presentes en el paquete.

Lógicamente, también habrá que realizar cambios en el Deparser para incluir la serialización de las cabeceras *Preámbulo APT* y los headers-stack *APT*.

4.6.1.4 Flujo de ejecución del proceso de etiquetado

Tal como se ha expuesto previamente, existen tres tipos de switches según su funcionalidad en el proceso de etiquetado: switch frontera origen, switch intermedio, y switch frontera destino. Como la finalidad de la tabla *tag_packet* es en esencia gestionar el etiquetado en las fronteras de la red, solamente los switches frontera deberán tener entradas en la tabla *tag_packet,* aunque es técnicamente posible realizar el etiquetado a partir de un switch intermedio (caso que es posible pero no se contempla en este documento). Un switch intermedio añadirá automáticamente una cabecera *APT* al header-stack si el paquete viene previamente etiquetado (bit de validez de *apt_preamble* verdadero).

Los switches frontera inician o eliminan el etiquetado y por ello, la tabla *tag_packet* posee acciones para implementar tales operaciones. La definición en P4 de esta tabla se presenta a continuación (*hybrid.p4/acl_hybrid.p4*):

```
table tag_packet {
        key = {
                hdr.ethernet.dstAddr: exact;
                hdr.ethernet.srcAddr: exact;
        }
        actions = {
                tag;
                untag;
        }
}
```

Como match key se emplean las direcciones MAC origen y destino, lo que implica que la monitorización de flujos se realiza a nivel de hosts. Podría implementarse un nivel mayor de

granularidad permitiendo discriminar, por ejemplo, a nivel de conexión TCP, simplemente cambiando los match fields utilizados.

Por otra parte, la tabla especifica dos posibles acciones:

- ***Tag***: Esta acción está pensada para ser especificada únicamente en el switch frontera origen, y desencadena la adición de una cabecera *Preámbulo APT* y la primera etiqueta APT del header-stack, es decir, marca el paquete para ser etiquetado y añade el preámbulo y la primera etiqueta del camino.
- ***Untag***: Esta acción está pensada para ser especificada en el switch frontera destino. Realiza una clonación del paquete, especificando como puerto de salida de la copia CPU_PORT (cliente P4Runtime), copia las cabeceras Preámbulo APT y etiquetas APT en metadatos de uso temporal, y elimina las cabeceras anteriores del paquete original.

El paquete clonado se copia a partir del paquete original tras salir del bloque de control de egreso, por lo que inicialmente el paquete clonado no dispone de las cabeceras de etiquetado. Por ello, la acción *untag* las almacena temporalmente como metadatos. Este paquete clonado se introduce en el bloque de control de egreso, cuya ejecución es distinta para paquetes clonados (el metadato *instance_type* permite conocer si el paquete es original o una copia), lo que permite darles un tratamiento diferenciado sobre las copias originales. Sobre un paquete clonado:

- Se añaden las cabeceras de etiquetado extraídas del paquete original.
- Se trunca el contenido del paquete, eliminando la carga útil, de forma que se conserven exclusivamente las cabeceras Ethernet y de etiquetado (preámbulo y etiquetas).
- Se añade la cabecera Packet-In.

El proceso de desetiquetado, clonación y reenvío de etiquetas se muestra en la Figura 37:

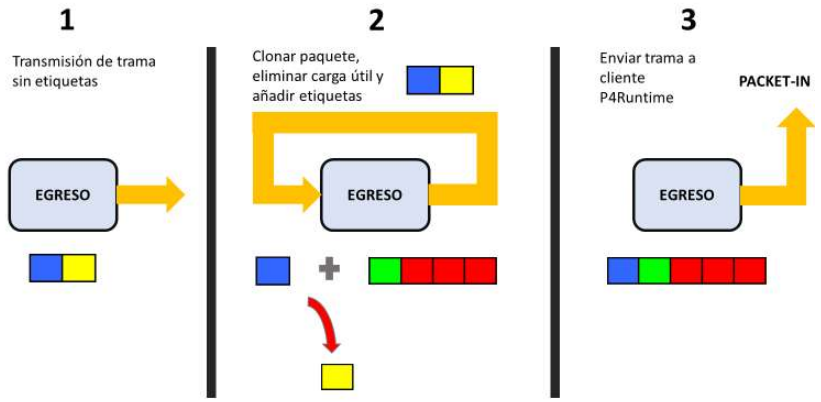


Figura 37 Desetiquetado en el switch frontera destino

De esta forma, el paquete clonado se envía al cliente P4Runtime (ONOS en nuestro caso) para que sea este quien procese los datos contenidos en las etiquetas.

Cada switch posee un meter indirecto de tipo paquete. Éste se emplea para configurar la tasa de etiquetado de paquetes, limitando el etiquetado al número de etiquetas que desee recibir el

cliente P4Runtime por unidad de tiempo. Si no se configura el meter señalado, todos los paquetes de un flujo introducido en la tabla *tag_packet* serán etiquetados. Si se configura una tasa de etiquetado, los paquetes que la excedan (color rojo) no serán etiquetados, aunque la entrada *tag_packet* especifique lo contrario. De esta forma, un cliente P4Runtime puede monitorizar un flujo de alto volumen de tráfico muestreando un paquete por segundo, sin necesidad de recibir todos y cada uno de los paquetes.

Por último, el programa P4 elimina el etiquetado si un paquete original tiene su puerto fijado en CPU_PORT, ya que, en este caso, el envío del paquete al plano de control habrá sido solicitado por el bloque de control de ingreso para un fin distinto al procesado de etiquetas. Si en este caso no se eliminasen las etiquetas, puede darse el caso de que el plano de control no entienda el contenido del paquete. Por ejemplo, una aplicación de ONOS que trabaje en capa 3 no sería capaz de extraer el datagrama IP a partir de la trama Ethernet, ya que las cabeceras serían un protocolo desconocido.

4.6.2 Aplicación ONOS para visualización de caminos

La visualización de caminos a partir de la recepción de paquetes etiquetados se ha implementado mediante el desarrollo de una aplicación ONOS que interactúa con la interfaz Web GUI de ONOS (ruta /onos/ui), resaltando sobre una representación de la topología de la red los enlaces sobre los que discurren los paquetes muestreados de un flujo monitorizado.

Por ello, la aplicación está compuesta de dos partes:

- **Extracción y procesado de etiquetas**: Procesa los Packet-Ins que contienen las etiquetas correspondientes a un paquete de un flujo, extrayendo sus valores y almacenándolos de forma estructurada.
- **Extensión UI**: Interactúa con la GUI de ONOS para resaltar sobre la topología el camino y visualizar datos relacionados en base a las etiquetas procesadas, y proporcionar controles al usuario para añadir, modificar o borrar flujos para monitorizar.

En resumen, el procesado de cabeceras extrae información de las etiquetas, y la Extensión UI consume dicha información, siendo procesos asíncronos entre ellos.

La aplicación permite especificar al usuario varios flujos a monitorizar, pero únicamente se recogerán las etiquetas de un único flujo, el que se conoce como *flujo activo*, lo que se traduce en que únicamente se podrá visualizar el camino de un flujo. Esta restricción no es del programa P4, sino del diseño de la aplicación, puesto que la visualización de varios flujos a la vez puede ser confusa. La aplicación permite cambiar el flujo activo de una forma rápida, remitiendo al manual de usuario contenido en el anexo. El cambio de flujo activo implica el borrado de las entradas correspondientes de la tabla *tag_packet* en los switches frontera del flujo antiguo, y la instalación de nuevas para el nuevo flujo activo.

4.6.2.1 Extracción y procesado de etiquetas

Este componente de la aplicación ONOS procesa un Packet-In con etiquetas cada segundo, ignorando el resto de paquetes que pueda recibir durante este periodo de tiempo. Como solamente existirá un flujo activo, solamente se recibirán Packet-Ins con paquetes etiquetados pertenecientes a dicho flujo.

El procesado del Packet-In consiste en la desencapsulación de las etiquetas, la extracción de los valores de sus campos, y la inserción de ellos en un objeto que permite acceder a la información de forma estructurada.

4.6.2.2 Extensión de la UI

En ONOS una extensión de la UI puede ser de tres tipos [67]: *Custom View* (extensión personalizable), *Tabular View* (extensión basada en vista de tipo tabla) , o *Topology Overlay* (extensión que contiene una representación gráfica de la topología de la red).

El tipo de la extensión desarrollada es *Topology Overlay*, dado que se necesita visualizar la topología para representar caminos. Una extensión se compone de código Java para el servidor, y código Angular.js [68] (framework JavaScript) para la ejecución en el navegador del cliente. Se ha implementado una API de comunicación entre estos dos componentes que permite al navegador interactuar con el servidor (ONOS en este caso), para añadir un nuevo flujo, cambiar el flujo activo, o eliminar uno o todos los flujos. Por otra parte, para el dibujo del camino no ha sido necesario implementar ninguna funcionalidad en la parte del cliente, puesto que, al ser una Extensión UI, se han podido reutilizar módulos ya implementados en Angular.js relativos a la representación de la topología.

A nivel funcional, cuando exista un flujo activo, se ejecuta una tarea cada segundo, de forma periódica, para dibujar el camino. Para ello, si existe información relativa a etiquetas almacenada en la aplicación, se obtienen los enlaces de la topología a resaltar a partir de dicha información. Adicionalmente, sobre cada switch en el camino se representa bien su respectivo valor de *timedelta* en una etiqueta, especificado en microsegundos, o bien el número de entradas ARP-Path que tiene almacenadas.

Por último, al hacer click en el navegador sobre un switch perteneciente al camino representado, se muestran los valores de los campos correspondientes a la etiqueta procesada.

En el anexo de este documento se detallan las instrucciones de uso, no obstante, la Figura 38 muestra un ejemplo de la visualización de un camino entre dos hosts:

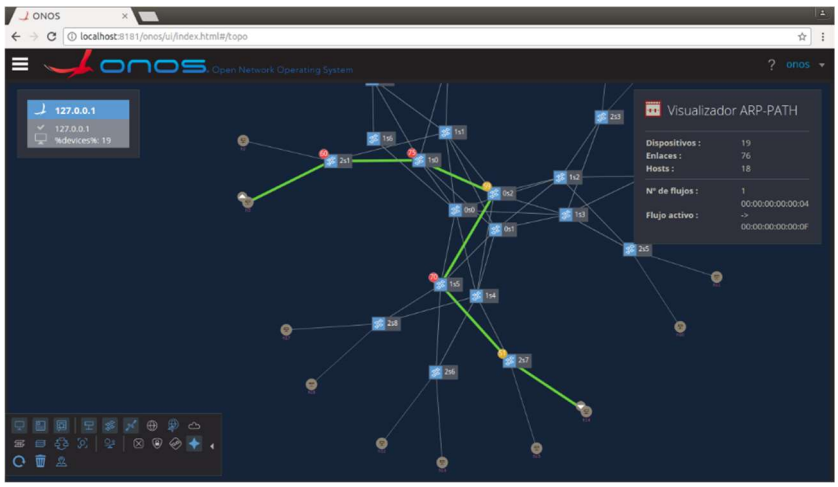


Figura 38 Camino visualizado en la GUI de ONOS

4.7 Consideraciones de rendimiento

BMv2 es un target pensado para un uso académico o experimental, debido principalmente a tres limitaciones en su diseño:

- Uso de una librería ineficiente para la transmisión y recepción de paquetes (*Libpcap* [30]).
- Sistema de colas con alta sobrecarga computacional por sincronización.
- Copias de memoria innecesarias.

Las ineficiencias anteriores son, en gran medida, consecuencia del diseño modular y reconfigurable del target. Por estos motivos, es posible realizar implementaciones con un rendimiento mayor sin excesivas modificaciones sobre el código base. La sustitución de la librería Libpcap es sencilla debido al carácter modular del componente *DevMgr*. De esta forma se puede implementar un *DevMgr* basado en otras librerías de entrada/salida de paquetes.

4.7.1 Problemas en la recepción de paquetes

En el apartado 3.6.1 se señaló que BMv2 emplea el *DevMgr* BMI por defecto, que es el único de los tres que permite conmutar paquetes de forma clásica. BMI es un sistema ineficiente porque emplea la librería Libpcap, inicialmente diseñada para capturar paquetes. Además, se realizan copias de memoria innecesarias.

Pero existe un problema de diseño en BMI que tiene una incidencia sobre el rendimiento del protocolo ARP-Path: el sistema de recepción de paquetes. Para el rendimiento óptimo del protocolo (escoger el camino de menor latencia) es preciso que un switch procese los paquetes en el orden temporal en que fueron recibidos. Sin embargo, el diseño de BMI hace que no se garantice el procesado de paquetes según el orden de llegada, sino que el orden de procesado depende esencialmente del orden de sondeo de las interfaces. Es decir, dados un paquete A recibido en un instante t=0 en la interfaz *eth1*, y un paquete B recibido en el instante t=1 en la interfaz *eth2*, BMI no garantiza que A se procese antes que B. Para ARP-Path esto podría implicar que mensajes ARP-Request tardíos se procesen antes que aquellos que fueron recibidos primero, alterando sensiblemente la formación de caminos respecto al funcionamiento teórico.

Entrando más en detalle, BMI asocia un descriptor de fichero a cada interfaz, y los almacena en un vector. BMI dispone de un hilo que realiza el sondeo sobre las interfaces del switch mediante el uso de la función *select()*, que permite realizar una espera pasiva sobre una serie de descriptores, a la espera de que alguno de ellos disponga de nueva información. Cuando se recibe un paquete por una interfaz, la función *select()* retorna con el número de descriptores que disponen de nueva información.

A continuación, BMI requiere identificar cuál es el descriptor que dispone de un nuevo paquete de la lista de vectores. En este punto se encuentra el problema, puesto que se itera en orden de índice creciente sobre el vector de descriptores para consultar a cada descriptor si dispone de nueva información, y recoger el paquete en caso afirmativo. Por tanto, el orden de sondeo de las interfaces depende del orden en que los descriptores estén almacenados en el vector.

Si en el retorno de la función *select()* solamente existe un descriptor con información, este modelo no supone un problema. Pero si existe más de un paquete, se procesará primero aquél cuyo índice de interfaz se encuentre situado antes en el vector de descriptores.

Para complicar más lo anterior, en cada iteración sobre el vector de descriptores, solamente se puede extraer un paquete de una interfaz, aunque se hayan recibido varios paquetes por la misma.

A continuación, se presenta un fragmento de código C simplificado de BMI, que realiza la funcionalidad descrita:

```
//fds es el vector de descriptores
 n = select(max_fd + 1, &fds, NULL, NULL, &timeout);

 for(i = 0; n && i < PORT_COUNT_MAX; i++) {
   port_info = get_port(port_mgr, i);
   //Sondear descriptor por si tiene paquetes disponibles
   if(!FD_ISSET(port_info->fd, &fds)) continue;
   --n;
   if(!port_info->bmi) continue;
   //Extraer paquete de la interfaz
   pkt_len = bmi_interface_recv(port_info->bmi, &pkt_data);
   if(pkt_len < 0 || !port_mgr->packet_handler) continue;
   //Llamar a metodo receive() del target para procesar paquete
   port_mgr->packet_handler(i, pkt_data,
              pkt_len, port_mgr->cookie);
 }
```

Este diseño de BMI puede presentar problemas en redes con alta carga de tráfico, o que presenten patrones basados en ráfagas de paquetes. A continuación, se presenta un ejemplo hipotético de la desordenación de los paquetes en BMI:

Se parte de 4 interfaces A,B,C,D cuyo orden en el vector de descriptores coincide con su orden de enumeración. En un instante determinado, se ejecuta el hilo de recepción con varios paquetes disponibles en las interfaces, y para simplificar el caso, no se reciben nuevos paquetes hasta que se han terminado de procesar los señalados.

La Figura 42 representa el orden de procesamiento de los paquetes según BMI. Cada paquete se representa con un cuadrado cuyo color representa la interfaz por la que se recibió, y el número, el instante de tiempo en que fue recibido.

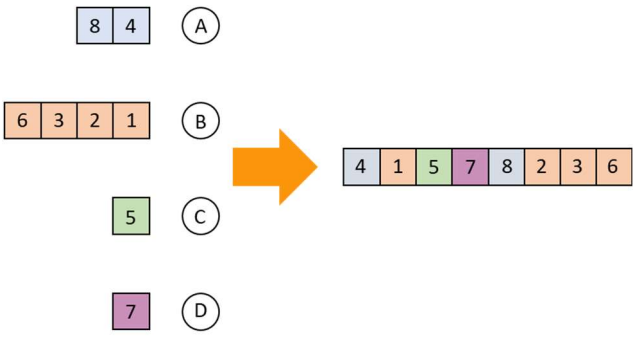


Figura 39 Ejemplo de gestión de colas en BMI

4.7.2 Mejoras propuestas sobre BMv2

En [26] se realiza un estudio sobre las ineficiencias de BMv2, y se propone una solución por cada uno de los tres problemas mencionados, destacándose la sustitución de la librería Libpcap por

NETMAP[69]. Por otra parte, BMAcc [70] es un target BMv2 que emplea el framework *DPDK* [37] y procesamiento multihilo para alcanzar velocidades de hasta 10 Gbps.

Alternativas para solucionar los problemas de colas y disminuir el uso de memoria aparecen referenciados en [26], donde se propone un nuevo modelo de cola basado en histéresis, y el uso de la librería *tcmalloc* [71], pensada para incrementar el rendimiento de reservas de memoria dinámicas en aplicaciones multihilo.

De las mejoras propuestas en [26], se ha implementado el uso de *tcmalloc*, por su sencillez, ya que únicamente es necesario enlazar dicha librería en la compilación del target BMv2. *Tcmalloc* sustituye la función *malloc* estándar de C por una implementación propia, de forma totalmente transparente al programador.

5 Evaluación y resultados

5.1 Pruebas de rendimiento

5.1.1 Medidas de ancho de banda

En este apartado se va a a medir la máxima tasa de tráfico que es capaz de proporcionar el target *ARP-P4* con el programa *hybrid.p4*. Para ello, se va a emplear una topología que incluye un switch al que se conectan dos hosts. Se va a generar un flujo con la máxima tasa de tráfico posible para medir el ancho de banda del switch, mediante la herramienta IPerf [72].

La prueba se ha realizado en un equipo con procesadores Intel® Core™ i7 con 24 GB de RAM y la media de los resultados arroja una tasa de 2 Gbps.

Dado que BMv2 es un switch software que se ejecuta en espacio de usuario, debe compartir su ejecución con otros procesos del sistema operativo. La segunda prueba a realizar mide cómo afecta el número de switches BMv2 instanciados en una única máquina al ancho de banda de un flujo, dado que no todos los switches podrán estar en ejecución a la vez.

Para ello, se ha empleado una topología con dos hosts conectados por cuatro switches en cascada. La media de los resultados obtenidos se sitúa en 600 Mbps, que es un resultado algo superior a la división entre los 2 Gbps de la prueba anterior y los cuatro switches instanciados. Por ello, se puede asumir que el ancho de banda se divide entre el número de switches que estén conmutando paquetes.

5.1.2 Evaluación del ancho de banda en topología Spine-Leaf

Para evaluar el ancho de banda proporcionado se va a emplear el conjunto de pruebas utilizados en [73]. La topología empleada es una Spine-Leaf 4-4-20 (4 Spines conectados a 4 Leafs, con 20 hosts por Leaf) con enlaces a 10 Mbps, que se muestra en la Figura 40:

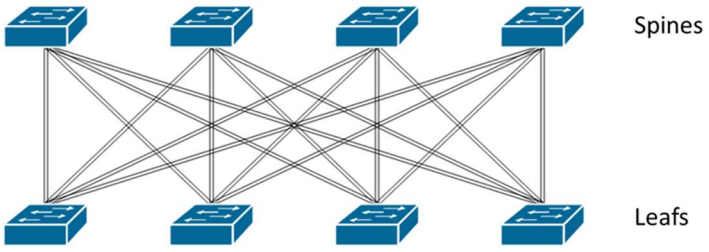


Figura 40 Topología Spine-Leaf empleada

El hardware empleado son 4 equipos con procesadores Intel® Core™ i7 con 24 GB de RAM interconectados por un switch GbE Netgear GS116. Se ha empleado la herramienta Mininet para emular la citada topología con switches ARP-P4 con el programa *hybrid.p4*., distribuyéndose 2 switches ARP-P4 por equipo hardware.

Se generarán flujos de tráfico disitribuidos aleatoriamente entre cualquier par de hosts situados en un lead disintos. Los flujos se clasifican en tres categorías, atendiendo a su tamaño:

- Un flujo ratón es un flujo de menos de 10 KB.
- Un flujo elefante es un flujo de más de 10 MB.
- El resto de flujos se consideran de tipo conejo.

Se emplearán dos distribuciones de tráfico, *Data Mining* y *Web Search*, basadas en medidas de tráfico real en centros de datos. La principal diferencia entre ambas distribuciones es el tipo predominante de tráfico: en *Data Mining* predominan los flujos de tipo ratón, mientras que en *Web Search* lo hacen los de tipo conejo.

Se han realizado 10 experimentos de 1800 segundos de duración, de los cuales 800 segundos se consideran tiempo de calentamiento para reducir al máximo posible el impacto de efectos transitorios en las medidas.

En la Figura 41 se muestra una comparativa de los resultados obtenidos entre ECMP, la implementación estándar de ARP-Path, y ARP-P4. En la columna de la izquierda, se muestran los resultados para *Web Search*, mientras que en la de la derecha, los resultados para *Data Mining*. Cada fila representa un tipo de tráfico (elefante, conejo o ratón).

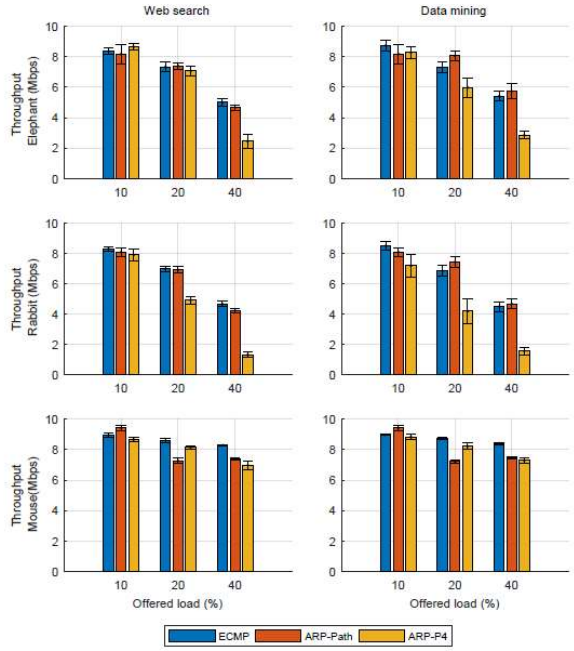


Figura 41 Comparativa de ancho de banda en topología Spine-Leaf

Observando la comparativa, se aprecia una degradación del rendimiento en función de la carga de la red de ARP-P4 mucho mayor que para ECMP y ARP-Path. Es especialmente relevante esta pérdida de rendimiento en flujos tipo conejo y elefante.

Los motivos de está degradación se derivan del diseño del sistema de recepción de paquetes (véase 4.7.1), y se manifiestan en dos problemas:

- Como consecuencia del desorden en el procesamiento de paquetes y la subsecuente variabilidad en la latencia para los paquetes dentro de un mismo flujo, los flujos tipo conejo y elefante se ven afectados por el control de congestión de TCP, de forma que experimentan variaciones en la carga de tráfico enviado. Por el contrario, los flujos tipo ratón apenas se ven afectados por el control de congestión, al ser un número reducido de paquetes, de forma que poseen un rendimiento comparable a ARP-Path.

- Por otra parte, el desorden de las colas impide que el balanceo de carga intrínseco a ARP-Path no funcione debidamente, ya que al formar caminos no se garantiza que los switches procesen en primer lugar la trama ARP-Request recibida. Así, se tiene que los caminos se distribuyen aleatoriamente, pero sin tener en cuenta la carga de la red.

5.2 Verificación de una lista de control de acceso

Dada la funcionalidad de seguridad perimetral desarrollada en este trabajo, se plantea verificar formalmente su correcto funcionamiento. La herramienta más adecuada para ello es *Packet Test Framework* (PTF) [74]. PTF permite diseñar tests unitarios para programas P4 que se ejecutan sobre targets BMv2, inyectando paquetes en el switch y analizando los paquetes reenviados por él. PTF está basado internamente en la librería Scapy [75], y es empleado en entornos de desarrollo profesionales, como el ya mencionado Barefoot Capilano.

PTF dispone de soporte para P4Runtime a través de un módulo disponible en el repositorio de la librería PI, pudiendo provisionar programas P4, modificar tablas y gestionar Packet-Ins/Outs.

Para verificar el correcto funcionamiento de una lista de control de acceso, se va a emplear PTF para instalar unas determinadas reglas de filtrado, y posteriormente inyectar una serie de paquetes específicamente escogidos para intentar verificar o vulnerar dichas reglas. Por cada paquete inyectado, el programa P4 podrá desecharlo, filtrarlo, o enviarlo a un cliente P4Runtime. PTF dispone de herramientas para verificar la recepción o no de un determinado paquete, así como para recibir Packet-Ins, comprobando en todos casos que la salida derivada de la ejecución es la esperada.

La prueba de funcionamiento puede representarse gráficamente con la Figura 42:

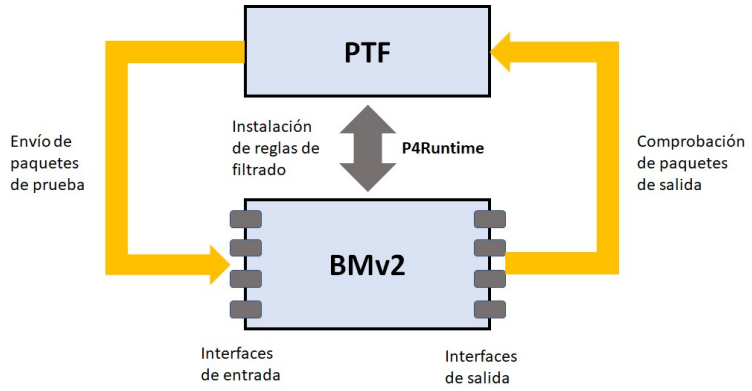


Figura 42 Verificación de reglas de filtrado con PTF

En la figura se observa que el target BMv2 contiene un conjunto de interfaces de entrada y otro conjunto de interfaces de salida. Está distinción es puramente teórica, ya que una interfaz puede pertenecer a uno, o a ambos grupos, en función de la respuesta esperada por PTF. Una interfaz de entrada es aquel puerto por el que PTF inyecta un paquete, mientras que una interfaz de salida es un puerto por el que puede reenviarse un paquete previamente inyectado por PTF. Nótese que una misma interfaz puede pertenecer a ambos grupos si BMv2 reenvia un paquete por el mismo puerto por el que fue recibido.

Paralelamente, PTF actuará como cliente P4Runtime para instalar reglas de filtrado, y verificar la recepción de Packet-Ins.

La lógica del test de verificación es la siguiente:

1. PTF crea un paquete con unas determinadas cabeceras, del que se conoce el comportamiento esperado en función de un puerto de entrada.
2. PTF inyecta dicho paquete por un puerto de entrada específico.
3. PTF verifica si el switch proporciona la respuesta esperada: filtrado, bloqueo o recepción de Packet-In, comprobando si se recibe o no algún paquete, y en su caso, si se recibe por la interfaz esperada.

Una vez se ha conocido el funcionamiento del test unitario a realizar, se procede a analizar la política de seguridad a implementar. Para ello, se toma como referencia una política estándar para cualquier sede empresarial, basada principalmente en el filtrado exhaustivo procedente del exterior, y en la interconxexión entre sedes. Una política basada en el filtrado de tráfico de un centro de datos hubiera sido más adecuada, pero ello implicaría añadir un soporte para VLANs que actualmente P4Runtime no proporciona (no está soportada aún la configuración de sesiones Multicast).

Se parte de una sede empresarial que posee una red interna 10.0.0.0/16, una red DMZ 160.88.82.0/24 con servidores DNS y HTTP, y se desea una interconxexión con otras sedes situadas en la subred 160.20.0.0/16. Asimismo, se implementa una política restrictiva respecto a las conexiones a Internet que realizan los empleados con el exterior, y para ello se dispone de un controlador SDN que impide la visita a webs no relacionadas con el desempeño profesional de los empleados de forma dinámica.

Por tanto, la política a implementar puede resumirse en los siguientes puntos:

- Bloqueo de todo tráfico por defecto.
- Permitir todo tráfico con origen y destino en la red interna (10.0.0.0/16).
- Permitir tráfico TCP/UDP desde el interior hacia el exterior.
- Permitir tráfico DNS y HTTP hacia la DMZ desde el exterior.
- Reenviar segmentos TCP SYN recibidos desde el interior hacia el controlador SDN, para que éste decida si permitir o no la conexión TCP, mediante la subsecuente instalación de reglas de filtrado asociadas a la conexión permitida.
- Permitir tráfico entre sedes (sin necesidad de filtrar segmentos TCP SYN a través del controlador).

Para ello, el test unitario diseñado, tras instalar las reglas que implementan la política anterior de seguridad, inyecta los siguientes paquetes:

1. TCP SYN desde el exterior hacia la red interna -> BLOQUEADO.
2. Datagrama UDP desde el exterior hacia la red interna -> BLOQUEADO.
3. Consulta DNS desde el exterior hacia la subred 160.88.82.0/24 -> PERMITIDO.
4. Acceso HTTP desde el exterior hacia la subred 160.88.82.0/24 -> PERMITIDO.
5. Segmento TCP SYN desde el interior -> PACKET-IN.
    - Controlador permite conexión TCP (nueva regla) -> TCP SYNACK desde el exterior -> PERMITIDO.
6. Segmento TCP SYN desde una sede remota a la red interna -> PERMITIDO.
7. Segmento TCP SYN desde la red interna hacia una sede remota -> PERMITIDO.
8. Datagrama UDP desde una sede remota a la red interna -> PERMITIDO.

9. Datagrama UDP desde la red interna hacia una sede remota -> PERMITIDO.
10. Paquete ICMP desde interfaz externa con IP origen perteneciente a la red interna (IP Spoofing) -> BLOQUEADO

5.3 Verificación de ARP-Path Telemetry

En este apartado se realiza una inspección sobre los paquetes transmitidos sobre la red para comprobar el correcto funcionamiento del sistema de etiquetas señalado en el apartado 4.6.1. Para ello, se va a analizar el transcurso de una trama Ethernet correspondiente a un Ping Request, enviado desde un host H1 a un host H2, a lo largo de su ruta por la red, observando los cambios introducidos por la adición de etiquetas.

La topología utilizada se corresponde con dos hosts conectados en cascada por cuatro switches, tal como aparece en la Figura 43:

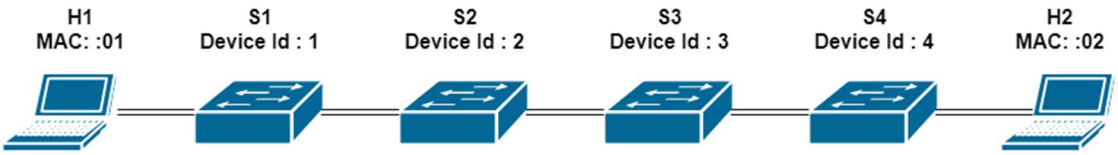


Figura 43 Topología para verificación de APT

Para mostrar el contenido de las tramas se emplea la herramienta Wireshark [76]. La primera trama a analizar es la transmitida desde el host H1 hacia S1, representada en la Figura 44:

```
Frame 60: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
  Interface id: 0 (s1-eth2)
  Encapsulation type: Ethernet (1)
  Arrival Time: Jun  6, 2018 10:01:17.707074223 Hora de verano romance

0000  00 00 00 00 00 02 00 00  00 00 00 01 08 00 45 00   ........ ......E.
0010  00 54 23 c7 40 00 40 01  02 e0 0a 00 00 01 0a 00   .T#.@.@. ........
0020  00 02 08 00 21 ed 12 88  00 01 cd 94 17 5b 00 00   ....!... .....[..
0030  00 00 15 c7 0a 00 00 00  00 00 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                             67
```

Figura 44 Trama entre H1 y S1

Se indican sobre la imagen los campos de dirección MAC destino, dirección MAC origen y Ethernet Type (0x800, IP). La carga útil se corresponde con un datagrama IP en el que va encapsulado un mensaje ICMP Ping Request.

La siguiente trama a analizar es la transmitida desde S1 hacia S2. S1 habrá introducido la primera etiqueta y el Preámbulo APT. El contenido de la trama se muestra en la Figura 45:

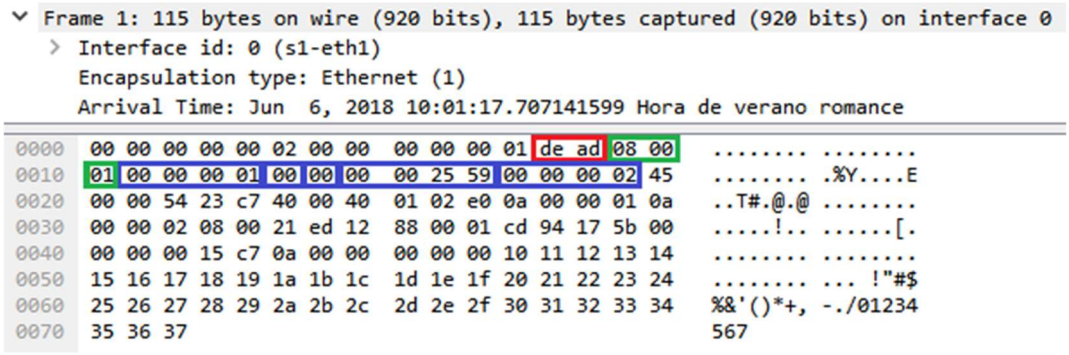


Figura 45 Trama entre S1 y S2

En rojo, se muestra el nuevo valor de Ethernet Type de la cabecera Ethernet, que indica al Parser del siguiente switch que la siguiente cabecera es de tipo Preámbulo APT. Los campos del Preámbulo APT se muestran en verde: los dos primeros bytes indican el Ethernet Type de la trama original (IP), y el tercer byte indica el número de etiquetas presentes en la trama, que en este caso es una. Por último, los distintos campos de la etiqueta APT se muestran en azul: 4 bytes para el Device Id (1, correspondiente a S1), 1 byte para la ocupación del búfer de ingreso, 1 byte para la ocupación del búfer de egreso, 4 bytes para el Timedelta, y 4 bytes para indicar el número de entradas ARP-Path existentes en la tabla (2, correspondientes a las MAC de H1 y H2).

La trama entre S2 y S3 se muestra en la Figura 46. S2 habrá apilado una nueva etiqueta, y habrá modificado el Preámbulo APT para indicar que la trama contiene dos etiquetas APT:

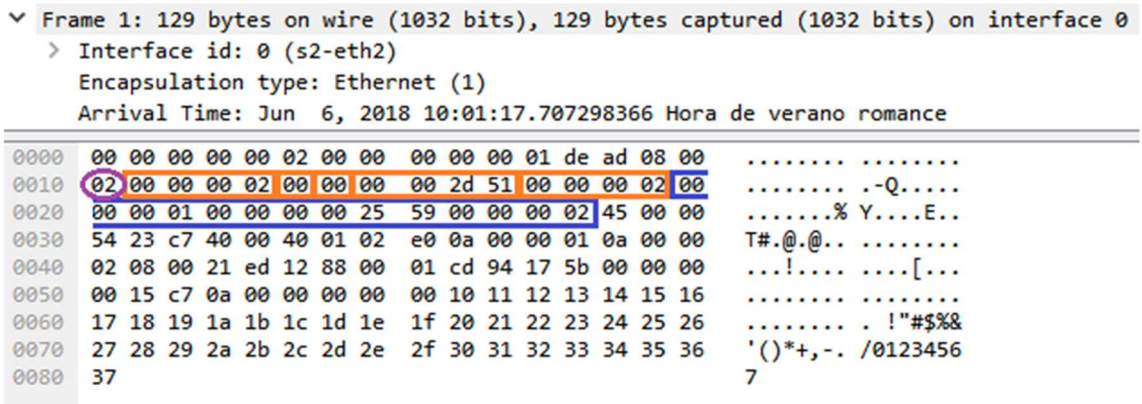


Figura 46 Trama entre S2 y S3

Redondeado en morado, aparece el byte del Preámbulo APT que indica el número de etiquetas apiladas: 2, una de S2, y otra de S1. En naranja, está recuadrada la etiqueta apilada por S2, con separados sus distintos campos. En azul, está recuadrada la etiqueta de S1. El campo de *Device Id* de la etiqueta de S2 se corresponde con su identificador, 2. Asimismo, se observa que el número de MACs aprendidas (últimos 4 bytes de las etiquetas APT) es idéntico en ambos switches, lo cual es coherente con la topología utilizada.

La trama entre S3 y S4 aparece representada en la Figura 47:

```
Frame 1: 143 bytes on wire (1144 bits), 143 bytes captured (1144 bits) on interface 0
  > Interface id: 0 (s3-eth2)
    Encapsulation type: Ethernet (1)
    Arrival Time: Jun  6, 2018 10:01:17.707415801 Hora de verano romance

0000  00 00 00 00 00 02 00 00  00 00 00 01 de ad 08 00   ........ ........
0010  03 00 00 00 03 00 00 00  00 44 95 00 00 00 02 00   ........ .D......
0020  00 00 02 00 00 00 00 2d  51 00 00 00 02 00 00 00   .......- Q.......
0030  01 00 00 00 00 25 59 00  00 00 02 45 00 00 54 23   .....%Y. ...E..T#
0040  c7 40 00 40 01 02 e0 0a  00 00 01 0a 00 00 02 08   .@.@.... ........
0050  00 21 ed 12 88 00 01 cd  94 17 5b 00 00 00 00 15   .!...... ..[.....
0060  c7 0a 00 00 00 00 00 10  11 12 13 14 15 16 17 18   ........ ........
0070  19 1a 1b 1c 1d 1e 1f 20  21 22 23 24 25 26 27 28   ....... !"#$%&'(
0080  29 2a 2b 2c 2d 2e 2f 30  31 32 33 34 35 36 37      )*+,-./0 1234567
```

Figura 47 Trama entre S3 y S4

En el Preámbulo APT se indica que la trama contiene 3 etiquetas (morado). En verde, aparece recuadrada la etiqueta apilada por S3, separada por campos. En naranja y azul, las etiquetas apiladas por S2 y S1, respectivamente, y ya analizadas en las figuras anteriores.

Por último, al ser S4 el switch frontera destino, realizará las operaciones mostradas en la Figura 37. Por tanto, S4 añade su etiqueta, reenvía la trama sin etiquetas a H2, y envía un Packet-In con las etiquetas al controlador ONOS.

La trama reenviada por S4 hacia H2 se representa en la Figura 48.

```
Frame 7: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
  > Interface id: 0 (s4-eth2)
    Encapsulation type: Ethernet (1)
    Arrival Time: Jun  6, 2018 10:01:17.707564930 Hora de verano romance

0000  00 00 00 00 00 02 00 00  00 00 00 01 08 00 45 00   ........ ......E.
0010  00 54 23 c7 40 00 40 01  02 e0 0a 00 00 01 0a 00   .T#.@.@. ........
0020  00 02 08 00 21 ed 12 88  00 01 cd 94 17 5b 00 00   ....!... .....[..
0030  00 00 15 c7 0a 00 00 00  00 00 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                             67
```

Figura 48 Trama entre S4 y H2

Esta trama es idéntica a la trama original reenviada por H1. El Packet-In se reenviará hacia el controlador ONOS a través del canal gRPC establecido entre S4 y ONOS, que lo procesará y mostrará en la GUI un resultado análogo al de la Figura 49, donde el indicador situado encima de cada switch hace referencia al campo Timedelta de cada etiqueta, expresado en microsegundos:

Figura 49 Camino representado por ONOS entre H1 y H2

6 Conclusiones

A la hora de realizar las conclusiones sobre el trabajo realizado, es preciso revisar el cumplimiento de los objetivos planteados al inicio del mismo.

En primer lugar, se ha realizado un estudio en profundidad acerca de una nueva tecnología que supone una evolución de las redes definidas por software: el lenguaje P4. Como ya se ha expuesto a lo largo de este documento, el lenguaje P4 permite describir planos de datos en dispositivos procesadores de paquetes, siendo especialmente relevante en dispositivos reprogramables. Además del lenguaje P4, se ha analizado la propuesta de arquitectura estándar para targets P4, la Portable Switch Architecture (PSA), y el protocolo P4Runtime como interfaz de control estandarizada para todo tipo de dispositivos P4.

La combinación de estas tres tecnologías señaladas puede suponer un avance muy importante en la programabilidad de las redes, y en el desarrollo y despliegue de nuevos protocolos y servicios. Asimismo, las tecnologías citadas contribuirán enormemente a la personalización de las capacidades de los equipos de red por parte de operadores o proveedores de servicios, logrando una mayor independencia de los fabricantes de hardware.

La adopción de la tecnología P4 está promovida por la Open Networking Foundation, que agrupa a los principales fabricantes de hardware, operadores, y proveedores de servicio. En este punto, hay que resaltar el proyecto Stratum, que pretende estandarizar un Sistema Operativo para dispositivos de red que emplea P4 y P4Runtime para la especificación del plano de datos, y el control del estado de reenvío.

En cuanto al trabajo desarrollado, se ha realizado una implementación del protocolo ARP-Path en un target BMv2 empleando código P4 y un objeto extern diseñado específicamente para tal fin. El extern desarrollado ha sido necesario para proporcionar primitivas de modificación del estado de reenvío desde un programa P4, algo que no es posible utilizando tablas P4 estándar. Además, el uso de un extern ha permitido optimizar la estructura de información y procesado respecto a una lógica expresada en lenguaje P4.

La integración realizada del protocolo ARP-Path ha definido un concepto de switch BMv2 plenamente autónomo, para el que no es necesario instanciar un proceso de plano de control externo. El switch implementado puede funcionar autónomamente en cualquier tipo de topología, en contraste con los switches BMv2 estándar, que requieren necesariamente de un plano de control externo (script ad-hoc, comandos CLI, controlador SDN,…) para poder encaminar paquetes.

Es esencial resaltar que la implementación de ARP-Path se adecúa completamente al estándar P4-16, y que no ha sido necesaria ninguna modificación sobre el código base de BMv2.

Se han desarrollado dos programas P4, *hybrid.p4* y *acl_hybrid.p4* que integran el protocolo ARP-Path con tablas cuyo estado de reenvío sea especificado por un plano de control externo. Estos programas permiten que el procesado a realizar sobre un paquete pueda ser definido a nivel local por el protocolo ARP-Path o por un controlador SDN.

Se ha realizado la integración de los switches desarrollados con el controlador ONOS mediante el protocolo P4Runtime. ONOS es un controlador SDN utilizado en despliegues reales de redes definidas por software, dado que sus características se adecúan a los requerimientos de

operadores y proveedores de servicios, y en ese sentido, hay que poner en valor la familiarización con un software que representa el presente y el futuro en cuanto a la adopción de la tecnología SDN.

Se han desarrollado dos Pipeconfs para los dos programas P4 desarrollados que permiten a aplicaciones ONOS disponer de una interfaz de control para los switches que ejecuten los programas P4 citados. Se han explorado las posibilidades que ofrece la provisión dinámica de programas P4 desde ONOS, y específicamente, el cambio dinámico de programa sin interrupción de servicio y conservando el estado de reenvío. Esta posibilidad es prometedora, puesto que permite variar dinámicamente las capacidades de las capacidades de los dispositivos de red, y puede suponer el desarrollo de funciones de red virtuales (VNFs) que se ejecuten directamente sobre un hardware, aumentando su eficiencia.

El diseño de ONOS permite aislar a las aplicaciones que emplean la API Northbound de los servicios utilizados por la interfaz Southbound, lo que se traduce en que se pueden desarrollar aplicaciones agnósticas respecto de los dispositivos de red utilizados. En este sentido, los Pipeconfs desarrollados permiten la interacción de una serie de aplicaciones ONOS (LLDP, Reactive Forwarding, Proxy-ARP, etc…) con los switches que ejecutan los programas P4 ya señalados, de la misma forma que podrían hacerlo con dispositivos OpenFlow. La principal consecuencia es que se pueden reutilizar aplicaciones de ONOS inicialmente pensadas para dispositivos OpenFlow, con dispositivos P4; no es necesario volver a reescribirlas, lo que sin duda es un valor añadido tanto para ONOS como para la adopción de la tecnología P4.

Para proporcionar capacidades de seguridad perimetral, se ha desarrollado una aplicación de ONOS que expone un servicio de filtrado de paquetes a través de la provisión del programa *acl_hybrid.p4*, y la interacción con él a través del framework *Flow Objetive*, gracias al desarrollo de un Pipeliner específco. Asimismo, se expone una interfaz REST para la provisión de los programas P4 citados, y la instalación de reglas de filtrado.

Por último, se ha desarrollado una aplicación de visualización de caminos sobre la GUI de ONOS, que permite comprobar de forma directa el camino que sigue un paquete a la vez que permite recoger estadísticas en cada switch que atraviesa. El desarrollo de esta aplicación ha sido posible gracias a la flexibilidad que proporciona P4 en cuanto al tratamiento de cabeceras, y supone una aplicación imposible de desarrollar con OpenFlow. Para su implementación, ha sido necesario desarrollar la lógica de recolección de estadísticas en P4, y procesar la información en ONOS de forma que se pueda visualizar. La utilidad de esta aplicación es útil para poder recoger información interna acerca del estado de los switches que atraviesa un paquete, siendo de especial relevancia para la verificación del correcto funcionamiento de ARP-Path, ya que ONOS desconoce por completo el estado de reenvío establecido por este protocolo.

La evaluación realizada muestra que la implementación realizada de ARP-Path sufre una degradación muy superior respecto a la implementación estándar a altas cargas de tráfico. Ello es consecuencia del sistema de recepción de paquetes integrado en BMv2, que no garantiza el procesamiento de paquetes en orden de llegada, y evita que ARP-Path genere caminos de latencia mínima. Al tratarse de un fallo a nivel de target, es probable que en un futuro pueda ser corregido.

Como consideración final, hay que señalar que el trabajo desarrollado supone el primer paso en cuanto al desarrollo de switches híbridos con P4, entendiendo por tal un concepto de switch

que es capaz de funcionar de forma autónoma, pero que a la vez puede ser configurable por un controlador SDN. El diseño de la implementación de ARP-Path permite la inclusión del protocolo en cualquier pipeline definido en P4, y además ofrece posibilidades de interacción más alla de la lógica estricta de ARP-Path: consulta de número de entradas, inserción de nuevas entradas, difusión controlada de tramas broadcast, etc… Es decir, el extern desarrollado no solo se circunscribe al propio protocolo ARP-Path, sino que puede exponer servicios del mismo a otras partes del programa P4.

6.1 Líneas de investigación futuras

La tecnología P4 todavía se encuentra en fase de desarrollo: la especificación del lenguaje P4-16 data de 2016, la arquitectura PSA de marzo de 2018 y P4Runtime de junio de 2018. Además, la implementación del compilador p4c y de P4Runtime sobre la librería PI todavía es parcial, lo que implica que a día de hoy no se pueden utilizar todas las posibilidades que ofrece P4, algo que cambiará a corto-medio plazo. Por ello, a lo largo del desarrollo del TFM se ha tenido que lidiar con las inevitables limitaciones de las implementaciones, y los continuos cambios de versiones de compilador y protocolo.

Sin embargo, una vez que se desarrollen por completo las implementaciones de p4c, PSA y P4Runtime, el abanico de posibilidades en cuanto a la investigación de la tecnología P4 se extenderá considerablemente. En este punto, es preciso hacer una introducción somera a algunas de las posibles líneas de investigación derivadas de este TFM:

- BMv2 fue desarrollado como plataforma de verificación de programas P4, de forma que el énfasis se puso en la modularidad y facilidad de uso, y no en el rendimiento. En el apartado 4.7 se señalaron algunas implementaciones de BMv2 que proporcionan mayor rendimiento respecto al estándar. Asimismo, se ha reseñado cómo el sistema de recepción de paquetes de BMv2 tiene un impacto negativo en el rendimiento de ARP-Path. Sería interesante ejecutar en dichas implementaciones los programas P4 desarrollados para observar el posible aumento de rendimiento de los mismos.
- Como consecuencia del soporte parcial de gNMI tanto en BMv2 como en ONOS, la implementación desarrollada del procotolo ARP-Path es básica, dado que carece de la imprescindible funcionalidad de recuperación de caminos mediante controlador SDN. Se plantea en este punto el uso de gNMI para monitorizar el estado de las interfaces del switch y poder gestionar la recuperación de caminos que atraviesen interfaces caídas cuandose notifiquen cambios en el estado de dichas interfaces (UP -> DOWN). Adicionalmente, podría explorarse la posibilidad de implementar los mecanismos de recuperación señalados en el apartado 3.9.3, o el desarrollo de nuevos externs para implementar otros protocolos de la familia All-Path.
- Una línea de investigación muy prometedora es la que surge de la interacción del extern ARP-Path con un controlador SDN mediante P4Runtime. Cuando se implemente en P4Runtime la interfaz para control de externs, sería posible extender el extern para poder ser configurado y controlado desde ONOS. Las posibilidades incluyen, entre otras muchas: lectura de la tabla de entradas ARP-Path y visualización de la misma en la GUI de ONOS, posibilidad de configurar los tiempos BT y LT de forma dinámica, o la modificación de la propia tabla ARP-Path. La integración del extern con ONOS significaría, por tanto, conocer el estado de reenvío ARP-Path, lo que permitiría a ONOS combinar la información propia con la proporcionada por ARP-Path de cara a optimizar

recursos. Por ejemplo, conocer la carga de tráfico asociada a los caminos ARP-Path, y en base a ello, calcular nuevos caminos intentando minimizar la carga de cada uno de ellos, lo que a día de hoy no es posible porque ONOS desconoce qué caminos ARP-Path existen.

- Por último, se sugiere la implementación de ARP-Path en otro target distinto a BMv2, preferiblemente hardware. En un target hardware la dificultad de diseñar un extern posiblemente sea mucho mayor respecto a BMv2, por lo que convendría explorar una solución basada íntegramente en P4 y P4Runtime, sin utilizar externs. Para ello, habría que utilizar un cliente P4Runtime por cada plano de datos, dedicado exclusivamente a implementar la lógica de control del protocolo (inserción de entradas, barrido de entradas caducadas, etc…). A día de hoy, esta implementación no es posible por la funcionalidad parcial de P4Runtime, pero si la implementación se realiza con éxito en un futuro, podría ser exportable a entornos comerciales (switches basados en el chip Barefoot Tofino, por ejemplo), y podría suponer el primer paso en la adopción del protocolo ARP-Path, dado que el programa P4 sería común a todos los targets.

7 Referencias

[1] Y. Jarraya, T. Madi, y M. Debbabi, «A survey and a layered taxonomy of software-defined networking», *IEEE Commun. Surv. Tutor.*, vol. 16, n.º 4, pp. 1955–1980, 2014.

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

[3] W. Stallings, «Software-Defined Networks and OpenFlow», *Internet Protoc. J.*, vol. 16, n.º 1.

[4] P. Bosshart *et al.*, «P4: programming protocol-independent packet processors», *SIGCOMM Comput Commun Rev*, vol. 44, n.º 3, pp. 87–95, 2014.

[5] «Sitio web del P4 Language Consortium». [En línea]. Disponible en: https://p4.org/. [Accedido: 03-abr-2018].

[6] «Sitio web de la Open Networking Foundation». [En línea]. Disponible en: https://www.opennetworking.org/. [Accedido: 03-abr-2018].

[7] «Repositorio Github de P4Runtime». [En línea]. Disponible en: https://github.com/p4lang/PI/tree/master/proto. [Accedido: 12-jun-2018].

[8] «Especificación de Portable Switch Architecture». [En línea]. Disponible en: https://p4.org/p4-spec/docs/PSA-v1.0.0.html. [Accedido: 12-jun-2018].

[9] «Especificación de P4-16 versión 1.0.0». [En línea]. Disponible en: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html. [Accedido: 03-abr-2018].

[10] «A Two Rate Three Color Marker». [En línea]. Disponible en: https://tools.ietf.org/html/rfc2698. [Accedido: 07-may-2018].

[11] «Repositorio Github de Thrift-BMv2». [En línea]. Disponible en: https://github.com/p4lang/behavioral-model/tree/master/thrift_src. [Accedido: 12-jun-2018].

[12] «P4 OpenFlow Agent». [En línea]. Disponible en: https://github.com/p4lang/p4ofagent. [Accedido: 04-abr-2018].

[13] «gRPC». [En línea]. Disponible en: https://grpc.io/. [Accedido: 04-abr-2018].

[14] «Protocol Buffers». [En línea]. Disponible en: https://developers.google.com/protocol-buffers/. [Accedido: 04-abr-2018].

[15] «HTTP/2». [En línea]. Disponible en: https://http2.github.io/. [Accedido: 05-abr-2018].

[16] «Cloud Native Computing Foundation». [En línea]. Disponible en: https://www.cncf.io/. [Accedido: 05-abr-2018].

[17] S. Abdi, W. Mohsin, Y. Yavuz, y A. Ghaffarkhah, «P4 Program-Dependent Controller Interface for SDN Applications». .

[18] «P4 Tutorial». [En línea]. Disponible en: https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf. [Accedido: 12-jun-2018].

[19] «gRPC Network Management Interface». [En línea]. Disponible en: https://github.com/openconfig/gnmi. [Accedido: 05-abr-2018].

[20] «OpenConfig». [En línea]. Disponible en: http://www.openconfig.net/. [Accedido: 05-abr-2018].

[21] «Network Configuration Protocol (NETCONF)». [En línea]. Disponible en: https://tools.ietf.org/html/rfc6241. [Accedido: 12-jun-2018].

[22] «RESTCONF». [En línea]. Disponible en: https://tools.ietf.org/html/rfc8040. [Accedido: 06-abr-2018].

[23] «Definición de streaming telemetry». [En línea]. Disponible en: http://www.openconfig.net/projects/telemetry/. [Accedido: 06-abr-2018].

[24] «Repositorio Github de p4c». [En línea]. Disponible en: https://github.com/p4lang/p4c. [Accedido: 12-jun-2018].

[25] «Repositorio Github BMv2». [En línea]. Disponible en: https://github.com/p4lang/behavioral-model. [Accedido: 12-jun-2018].

[26] Iozelli, Yuri, «Performance Improvements on the P4 Software Switch». .

[27] «Repositorio GitHub de p4c-behavioral». [En línea]. Disponible en: https://github.com/p4lang/p4c-behavioral. [Accedido: 06-abr-2018].

[28] «Apache Thrift». [En línea]. Disponible en: https://thrift.apache.org/. [Accedido: 06-abr-2018].

[29] «Nanomsg». [En línea]. Disponible en: http://nanomsg.org/. [Accedido: 06-abr-2018].

[30] «Librería Libpcap». [En línea]. Disponible en: https://github.com/the-tcpdump-group/libpcap. [Accedido: 20-abr-2018].

[31] «Librería PI». [En línea]. Disponible en: https://github.com/p4lang/PI. [Accedido: 06-abr-2018].

[32] «Sysrepo». [En línea]. Disponible en: http://www.sysrepo.org/. [Accedido: 06-abr-2018].

[33] «Barefoot Tofino». [En línea]. Disponible en: https://barefootnetworks.com/products/brief-tofino/. [Accedido: 12-jun-2018].

[34] «Open Compute Project». [En línea]. Disponible en: http://www.opencompute.org/. [Accedido: 13-jun-2018].

[35] «Barefoot Capilano». [En línea]. Disponible en: https://www.barefootnetworks.com/products/brief-capilano/. [Accedido: 06-abr-2018].

[36] «Repositorio Github T4P4S». [En línea]. Disponible en: https://github.com/P4ELTE/t4p4s. [Accedido: 12-jun-2018].

[37] «DPDK». [En línea]. Disponible en: http://dpdk.org/. [Accedido: 21-may-2015].

[38] M. Shahbaz *et al.*, «Pisces: A programmable, protocol-independent software switch», en *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, 2016, pp. 525–538.

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

[40] «NetFPGA». [En línea]. Disponible en: http://netfpga.org/site/#/. [Accedido: 12-jun-2018].

[41] «Arquitectura SimpleSumeSwitch». [En línea]. Disponible en: https://github.com/NetFPGA/P4-NetFPGA-public/wiki/Workflow-Overview. [Accedido: 06-abr-2018].

[42] «Arquitectura de ONOS». [En línea]. Disponible en: https://wiki.onosproject.org/display/ONOS/System+Components?preview=/1048836/14 41810/onos-stack.png#SystemComponents-SystemTiers. [Accedido: 06-abr-2018].

[43] C. Cascone, «P4 Support in ONOS». Open Networking Foundation.

[44] «Sitio web de Stratum». [En línea]. Disponible en: https://stratumproject.org/. [Accedido: 03-abr-2018].

[45] «Open Network Linux». [En línea]. Disponible en: https://opennetlinux.org/. [Accedido: 09-abr-2018].

[46] «gRPC Network Operations Interface». [En línea]. Disponible en: https://github.com/openconfig/gnoi. [Accedido: 09-abr-2018].

[47] «Trellis». [En línea]. Disponible en: https://www.opennetworking.org/projects/trellis/. [Accedido: 12-jun-2018].

[48] «CORD». [En línea]. Disponible en: https://opencord.org/. [Accedido: 09-abr-2018].

[49] «Introducción al Rapid Spanning Tree Protocol [Spanning Tree Protocol rápida] (802.1w)». [En línea]. Disponible en: http://www.cisco.com/cisco/web/support/LA/7/75/75983_146.html. [Accedido: 14-jun-2018].

[50] «Understanding Multiple Spanning Tree Protocol (802.1s)». [En línea]. Disponible en: http://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/24248-147.html. [Accedido: 14-jun-2018].

[51] «IEEE 802.1: 802.1aq - Shortest Path Bridging», 2012. [En línea]. Disponible en: http://www.ieee802.org/1/pages/802.1aq.html. [Accedido: 14-jun-2018].

[52] «Transparent Interconnection of Lots of Links (trill) -». [En línea]. Disponible en: https://datatracker.ietf.org/wg/trill/charter/. [Accedido: 14-jun-2018].

[53] E. Rojas *et al.*, «All-Path bridging: Path exploration protocols for data center and campus networks», *Comput. Netw.*, vol. 79, n.º 0, pp. 120 – 132, 2015.

[54] Y. K. Dalal y R. M. Metcalfe, «Reverse Path Forwarding of Broadcast Packets», *Commun ACM*, vol. 21, n.º 12, pp. 1040–1048, dic. 1978.

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

[56] M. Stamp, *Information security: principles and practice*. John Wiley & Sons, 2011.

[57] «Network Configuration Service (ONOS)». [En línea]. Disponible en: https://wiki.onosproject.org/display/ONOS/The+Network+Configuration+Service. [Accedido: 16-abr-2018].

[58] *Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet*. .

[59] «802.1AB: Link Layer Discovery Protocol (LLDP)». [En línea]. Disponible en: http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf.

[60] «LLDP Provider (ONOS)». [En línea]. Disponible en: https://github.com/opennetworkinglab/onos/tree/master/providers/lldp. [Accedido: 16-abr-2018].

[61] «Host Provider (ONOS)». [En línea]. Disponible en: https://github.com/opennetworkinglab/onos/blob/master/core/api/src/main/java/org/onosproject/net/host/HostProvider.java. [Accedido: 16-abr-2018].

[62] «Network Config Host Provider (ONOS)». [En línea]. Disponible en: https://wiki.onosproject.org/display/ONOS/Network+Config+Host+Provider. [Accedido: 16-abr-2018].

[63] «Aplicación Reactive Forwarding de ONOS». [En línea]. Disponible en: https://github.com/opennetworkinglab/onos/blob/master/apps/fwd/src/main/java/org/onosproject/fwd/ReactiveForwarding.java. [Accedido: 16-abr-2018].

[64] «In-Band Network Telemetry Dataplane Specification». [En línea]. Disponible en: https://github.com/p4lang/p4-applications/blob/master/docs/INT.pdf. [Accedido: 12-jun-2018].

[65] «Repositorio del P4 Applications WG». [En línea]. Disponible en: https://github.com/p4lang/p4-applications. [Accedido: 18-abr-2018].

[66] «Multi-Hop Route Inspection». [En línea]. Disponible en: https://github.com/p4lang/tutorials/tree/master/P4D2_2018_East/exercises/mri. [Accedido: 18-abr-2018].

[67] «Tutoriales para extender la GUI de ONOS». [En línea]. Disponible en: https://wiki.onosproject.org/display/ONOS/Customizing+and+Extending+the+ONOS+GUI. [Accedido: 19-abr-2018].

[68] «Framework Angular.js». [En línea]. Disponible en: https://angularjs.org/. [Accedido: 19-abr-2018].

[69] L. Rizzo, «Netmap: A Novel Framework for Fast Packet I/O», en *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, Berkeley, CA, USA, 2012, pp. 9–9.

[70] «BMAcc: Accelerating P4-Based Data Plane with DPDK». [En línea]. Disponible en: https://dpdksummit.com/Archive/pdf/2017USA/Accelerating%20P4-based%20Dataplane%20with%20DPDK.pdf. [Accedido: 12-jun-2018].

[71] «Librería TCMalloc». [En línea]. Disponible en: http://goog-perftools.sourceforge.net/doc/tcmalloc.html. [Accedido: 09-abr-2018].

[72] *iPerf - The network bandwidth measurement tool*. .

[73] J. Alvarez-Horcajo, D. Lopez-Pajares, J. M. Arco, J. A. Carral, y I. Martinez-Yelmo, «TCP-path: Improving load balance by network exploration», en *Cloud Networking (CloudNet), 2017 IEEE 6th International Conference on*, 2017, pp. 1–6.

[74] «Packet Test Framework». [En línea]. Disponible en: https://github.com/p4lang/ptf. [Accedido: 04-jun-2018].

[75] «Scapy». [En línea]. Disponible en: https://github.com/secdev/scapy. [Accedido: 04-jun-2018].

[76] «Wireshark». [En línea]. Disponible en: https://www.wireshark.org/. [Accedido: 06-jun-2018].

[77] «Librerías Boost». [En línea]. Disponible en: https://www.boost.org/. [Accedido: 08-jun-2010].

8 Anexos

8.1 Duración y costes del proyecto

8.1.1 Planificación Temporal

La relación de tareas concretas a desarrollar son las siguientes:

- Estudio de P4 y de los compiladores disponibles
- Estudio de BMv2
- Desarrollo de extern ARP-Path
- Estudio de P4Runtime
- Integración de BMv2 con P4Runtime
- Estudio de ONOS
- Desarrollo de aplicaciones ONOS
- Evaluación

8.1.2 Diagrama de Gantt

La Tabla 3 muestra el tiempo en semanas en realizar cada tarea:

Tabla 3 Distribución temporal del trabajo realizado

TAREA	SEMANA															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Estudio de P4 y de compiladores disponibles																
Estudio de BMv2																
Implementación del extern ARP-Path																
Estudio de P4Runtime																
Integración del switch con P4Runtime																
Estudio de ONOS																
Desarrollo de aplicaciones ONOS																
Evaluación																

8.1.3 Medios

Los recursos utilizados en el desarrollo del trabajo son los siguientes:

- Hardware:
  - Ordenador portátil para desarrollo
  - 5 equipos para evaluación de rendimiento Intel Core i7 24 GB RAM
  - Switch NetGear GbE
- Software de código abierto gratuito:
  - BMv2
  - p4c
  - Mininet
  - ONOS
- Recursos bibliográficos:
  - Acceso a internet

8.1.4 Costes

La Tabla 4 presenta los costes asociados al desarrollo del TFM:

Tabla 4 Costes del proyecto

Recurso	Unidades	Coste unitario	Coste
Hora trabajo Ingeniero Junior	300	50 €	15000 €
Equipo de evaluación	5	1000 €	5000 €
Ordenador portátil	1	800 €	800 €
Switch NetGear	1	125 €	125 €
Conexión a Internet menusal	4	50 €	200 €
TOTAL			21125 €

8.2 Instalación y configuración de switch BMv2

8.2.1 Dependencias necesarias

El primer paso es instalar todas las dependencias necesarias:

```
sudo apt update
sudo apt-get install -y --no-install-recommends \
        autoconf \
        automake \
        bison \
        build-essential \
        cmake \
        cpp \
        curl \
        flex \
        git \
        libavl-dev \
        libboost-dev \
        libboost-program-options-dev \
        libboost-system-dev \
        libboost-filesystem-dev \
        libboost-thread-dev \
        libboost-filesystem-dev \
        libboost-program-options-dev \
        libboost-system-dev \
        libboost-test-dev \
        libboost-thread-dev \
        libc6-dev \
        libev-dev \
        libevent-dev \
        libffi-dev \
        libfl-dev \
        libgc-dev \
        libgc1c2 \
        libgflags-dev \
        libgmp-dev \
        libgmp10 \
        libgmpxx4ldbl \
        libjudy-dev \
        libpcap-dev \
        libpcre3-dev \
        libreadline6 \
        libreadline6-dev \
        libssl-dev \
        libtool \
        make \
        mktemp \
        pkg-config \
        protobuf-c-compiler \
        python \
        python-dev \
        python-ipaddr \
        python-pip \
        python-scapy \
        python-setuptools \
        tcpdump \
        wget \
        unzip
```

Para una versión de Ubuntu 14.04, se requieren además las siguientes dependencias:

```
sudo apt install -y python-software-properties software-properties-
common
sudo add-apt-repository -y ppa:ubuntu-toolchain-r/test
sudo add-apt-repository -y ppa:george-edison55/cmake-3.x
sudo apt update
sudo apt install -y \
        dpkg-dev \
        g++-4.9 \
        gcc-4.9 \
        cmake \
        libbz2-dev

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.9
50
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.9
50
```

También es necesaria la instalación de las librerías Boost[77] en el caso de que no estén instaladas:

```
wget
https://sourceforge.net/projects/boost/files/boost/1.58.0/boost_1_58_0
.tar.bz2/download -O boost_1_58_0.tar.bz2
tar --bzip2 -xf boost_1_58_0.tar.bz2
cd boost_1_58_0

./bootstrap.sh --with-libraries=iostreams
sudo ./b2 install
sudo ldconfig
```

Por el contrario, si la versión de Ubuntu es 16.04, se requieren las siguientes dependencias:

```
sudo apt-get update
sudo apt-get install -y --no-install-recommends \
        ca-certificates \
        g++ \
        libboost-iostreams1.58-dev \
        libprotobuf-c-dev
```

Con las dependencias instaladas, se puede proceder a instalar gRPC y Protobuf. Para instalar Protobuf:

```
PROTOBUF_COMMIT="tags/v3.2.0"
git clone https://github.com/google/protobuf.git
cd protobuf
git fetch
git checkout ${PROTOBUF_COMMIT}

export CFLAGS="-Os"
export CXXFLAGS="-Os"
export LDFLAGS="-Wl,-s"
./autogen.sh
./configure --prefix=/usr
make
sudo make install
sudo ldconfig
unset CFLAGS CXXFLAGS LDFLAGS
```

Para instalar gRPC:

```
GRPC_COMMIT="tags/v1.3.2"
git clone https://github.com/grpc/grpc.git
cd grpc
git fetch
git checkout ${GRPC_COMMIT}
git submodule update --init

export LDFLAGS="-Wl,-s"
make
sudo make install
sudo ldconfig
unset LDFLAGS
```

8.2.2 Instalación del compilador p4c

Para instalar el compilador de programas P4, p4c, se pueden seguir las siguientes instrucciones:

```
git clone https://github.com/p4lang/p4c.git
cd p4c
git checkout "4c0d629ce2492294ff4108c910f8e6be44112c68"
git submodule update --init --recursive

mkdir -p build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

NOTA: La versión que genera un archivo P4INFO adecuado a la versión de P4Runtime de ONOS 1.13 NO es compatible con la última versión de p4c. La versión de p4c compatible con ONOS 1.13 se corresponde con el commit de p4c *4c0d629ce2492294ff4108c910f8e6be44112c68*. Sin embargo, esa versión del compilador no posee soporte para externs arbitrarios, por lo que no se podría compilar el extern ArpPath. El fichero P4Info debe generarse a partir de la versión del compilador asociada al commit anterior.

8.2.3 Instalación de la librería PI (P4Runtime)

La versión de P4Runtime a instalar es la soportada por ONOS 1.13, que data de principios de Abril de 2018. Por tanto, se corresponde con una implementación de P4Runtime anterior a la versión 1.0. La versión de P4Runtime soportada por ONOS se indica en el fichero protocols/p4runtime/BUCK, en la variable PI_COMMIT.

```
PI_COMMIT="0325da7746efe192935e8969fd08eed68d654c98"
git clone https://github.com/p4lang/PI.git p4runtime
cd p4runtime
git fetch
git checkout ${PI_COMMIT}
git submodule update --init --recursive

./autogen.sh
./configure --with-proto --without-internal-rpc --without-cli 'CFLAGS=
-O3' 'CXXFLAGS= -O3'
make
```

```
sudo make install
sudo ldconfig
```

8.2.4 Instalación de BMv2

8.2.4.1 Instalación de la librería tcmalloc

En el apartado 4.7 se habló de una mejora de rendimiento propuesta en [26] basada en la inclusión de la librería TCmalloc. Para instalarla:

```
git clone https://github.com/gperftools/gperftools.git
cd gperftools
./autogen.sh
./configure
make
sudo make install
```

8.2.4.2 Compilación e instalación de BMv2

El repositorio de BMv2 con el target ARP-P4 y el extern ArpPath se proporciona en el cd adjunto. El primer paso es instalar las dependencias que requiere BMv2. Para ello se dispone de varios scripts que automatizan los procesos:

```
tmpdir=`mktemp -d -p .`
cd ${tmpdir}
bash ../travis/install-thrift.sh
bash ../travis/install-nanomsg.sh
sudo ldconfig
bash ../travis/install-nnpy.sh
cd ..
sudo rm -rf $tmpdir
```

A continuación, se puede proceder a instalar BMv2:

```
./autogen.sh
./configure --with-pi --without-nanomsg --disable-elogger --disable-
logging-macros\
 'CFLAGS= -O3 -ltcmalloc_minimal' 'CXXFLAGS= -O3 -ltcmalloc_minimal'
make
sudo make install
```

El siguiente paso es compilar el ejecutable *arp4th* que cuenta con soporte para la API Thrift, y el ejecutable *grpc_arp4th*, que dispone de soporte para P4Runtime, y es el target en el que se han implementado todas las funcionalidades presentadas en este documento. En el mismo directorio en el que se ha compilado BMv2:

```
cd targets/arp4th
make
cd ../grpc_arp4th
make
```

El ejecutable *grpc_arp4th* es el target BMv2 que incluye soporte para el extern ArpPath y P4Runtime.

8.2.5 Instalación de Packet Test Framework

Para poder ejecutar el test unitario referenciado en el apartado 5.2 es preciso instalar Packet Test Framework (PTF).

PTF requiere de la librería Scapy:

```
git clone https://github.com/p4lang/scapy-vxlan.git
cd scapy-vxlan
sudo python setup.py install
```

A continuación, se clona el repositorio de PTF y se instala:

```
git clone https://github.com/p4lang/ptf.git
cd ptf
sudo python setup.py install
```

8.3 Compilación de programas P4

Para compilar programas P4, se debe disponer del compilador p4c, cuya instalación se ha documentado en el apartado anterior. En el apartado 3.5 se presentaron los backends que incluye el respositorio por defecto; solamente se abordará la compilación para el backend BMv2.

La orden que identifica el compilador p4c con backend BMv2 es : p4c-bm2-ss. Para generar un programa P4 con externs no estándar se puede emplear la siguiente orden:

```
p4c-bm2-ss --emit-externs programa.p4 -o programa.json
```

donde programa.p4 hace referencia al programa a compilar, y programa.json al fichero JSON que debe ser consumido por un target BMv2.

La implementación actual de la opción *--emit-externs* incluye como instancia extern en el JSON tanto los externs arbitrarios como los de la arquitectura V1Switch. El problema es que para BMv2, los externs de V1Switch no están implementados internamente como clases derivadas de *ExternType* (a diferencia del extern *ArpPath*), sino como clases plenamente integradas en el diseño del target. Ello provoca que BMv2 no pueda procesar el fichero JSON, por lo que la solución actual es modificar el JSON, eliminando en el campo *extern_instances* todos aquellos externs que pertenezcan a la arquitectura V1Switch.

Si el programa P4 solamente incluye un extern *ArpPath,* el campo *extern_instances* del fichero JSON debería ser similar al mostrado a continuación:

```
"extern_instances" : [
    {
      "name" : "extern_arp_path",
      "id" : 0,
      "type" : "ArpPath",
      "attribute_values" : [
        {
          "name" : "size",
          "type" : "hexstr",
          "value" : "0x400"
        }
      ]
    }
  ]
```

Paralelamente, debe generarse el fichero P4Info, que recordemos, describe la API P4Runtime a utilizar por el plano de control para un programa concreto. Para generarlo, se puede utilizar una orden análoga a la siguiente:

```
p4c --p4runtime-format text --p4runtime-file programa.p4info
                            programa.p4
```

donde se especifica que el formato del fichero p4info sea texto (adaptado a BMv2), y que el nombre del fichero generado sea programa.p4info.

NOTAS: Puede que la versión del compilador no genere el fichero P4Info con el nombre deseado, sino que se emplee el nombre *<nombre-programa-p4>.p4rt*. El P4Info debe generarse con la versión del compilador adecuada a ONOS, pero se plantea el problema de que dicha versión no

dispone de soporte para externs. La solución provisional consiste en eliminar las referencias al extern *ArpPath* en el programa P4 para que sea de esta forma aceptado por el compilador, y genere el P4Info.

8.4 Ejecución de un target BMv2

Todo target BMv2 por defecto tiene una serie de argumentos comunes:

- **`--device-id`** permite asociar un identificador de 64 bits al switch, imprescindible en entornos SDN. Si no se especifica ningún valor, el valor por defecto es 0.
- **`--log-console`** permite mostrar información de depuración, que con la configuración de instalación presentada, se encuentra deshabilitada por motivos de rendimiento.
- **`-i [Número de puerto]@[Número de interfaz]`** añade un puerto conectado a una interfaz, ya sea física o virtual. Se debe emplear este argumento tantas veces como interfaces se desee introducir en el switch.
- **`--no-p4`** permite arrancar el target sin consumir un programa P4. Sin programa P4 el target no es capaz de conmutar paquetes.
- Para arrancar el switch con un programa P4, basta con especificar la ruta al fichero JSON correspondiente, sin ningún tipo de carácter. La especificación de un fichero JSON y la opción `--no-p4` son lógicamente incompatibles entre sí.

BMv2 dispone de otras muchas opciones, en general relacionadas con la depuración de programas. Para mostrar la ayuda relacionada con la ejecución de BMv2 puede emplearse el argumento -h, o simplemente lanzar el target BMv2 sin ningún argumento.

Los targets compatibles con P4Runtime (*simple_switch_grpc* y *grpc_ARP-P4*) pueden requerir del uso de más argumentos respecto a los targets estándar. Para especificar argumentos propios asociados a estos targets, hay que añadir los caracteres -- después de los argumentos *estándar*, para a continuación introducir los específicos de P4Runtime:

- **`--cpu-port [número]`** indica el identificador de puerto asociado al cliente P4Runtime. No se puede emplear un identificador asociado a una interfaz.
- **`--grpc-server-addr [IP]:[Puerto TCP]`** permite especificar la IP y puerto TCP donde escuchará el servidor gRPC. Si no se especifica esta opción, su valor por defecto es *0.0.0.0:50051*.

Un ejemplo para el arranque del target grpc_ARP-P4 puede ser la siguiente orden:

```
[Ruta ejecutable grpc_ARP-P4] --device-id 1 -i 1@eth1 -i 2@eth2
hybrid.json -- --grpc-server-addr 0.0.0.0:49090 --cpu-port 255
```

La orden anterior arranca el target *grpc_ARP-P4* indicando un identificador de switch, dos interfaces y la carga del programa *hybrid.json*. En cuanto a las opciones específicas para P4Runtime, se indica que el servidor gRPC escuche en el puerto 49090, y que el identificador de puerto asociado al cliente P4Runtime sea 255.

NOTA: El uso del valor 0 para identificar un puerto está permitido en BMv2 pero no se acepta para los programas *hybrid.p4* y *acl_hybrid.p4*.

8.5 Instalación y configuración de ONOS

8.5.1 Instalación y compilación de ONOS con los Pipeconfs desarrollados

Se han desarrollado tres aplicaciones para ONOS, de las cuales los dos Pipeconfs se compilan con BUCK dentro de la compilación de ONOS, mientras que el visualizador de caminos, se compila en un proyecto aparte con la herramienta Maven. La razón por la cual se utiliza BUCK para compilar los Pipeconfs reside en que, en el momento de su desarrollo, Maven no disponía de las dependencias de P4Runtime, y la única forma de referenciarlas era mediante BUCK.

El código asociado a los Pipeconfs está en el cd adjunto. Para compilarlo, habría que descargar del repositorio de ONOS la última versión de ONOS Nightingale (1.13), y copiar los proyectos asociados a los dos Pipeconfs. Por último, incluir en el fichero modules.defs la ruta a los proyectos señalados para que BUCK los incluya en la compilación de ONOS.

Hechas las operaciones anteriores, se puede compilar ONOS con la orden siguiente, ejecutada desde el directorio raíz del repositorio:

```
tools/build/onos-buck build onos --show-output
```

En buck-out/gen/tools/package/onos-package se encuentra una carpeta comprimida que contiene ONOS ya compilado.

Una vez descomprimido, en el directorio bin/tools se puede ejecutar la orden siguiente para iniciar ONOS:

```
onos-service start
```

8.5.2 Instalación y activación del visualizador de caminos

La aplicación ONOS de visualización de caminos se proporciona como un proyecto que contiene tanto el código, como la aplicación compilada. Si se desea compilarla, primero hay que instalar la herramienta Maven si no lo está, y después basta ejecutar la siguiente orden dentro del directorio raíz:

```
mvn clean install
```

Ello generara un fichero .oar dentro de la carpeta targets. Este es el archivo que se debe cargar en ONOS para instalar y activar la aplicación. Un método muy intuitivo es el disponible a través de la Web GUI (:8181/onos/ui).

Tras acceder con nuestro usuario y contraseña, pinchar en el menú desplegable a la izquierda, y despúes en *Aplicaciones*:

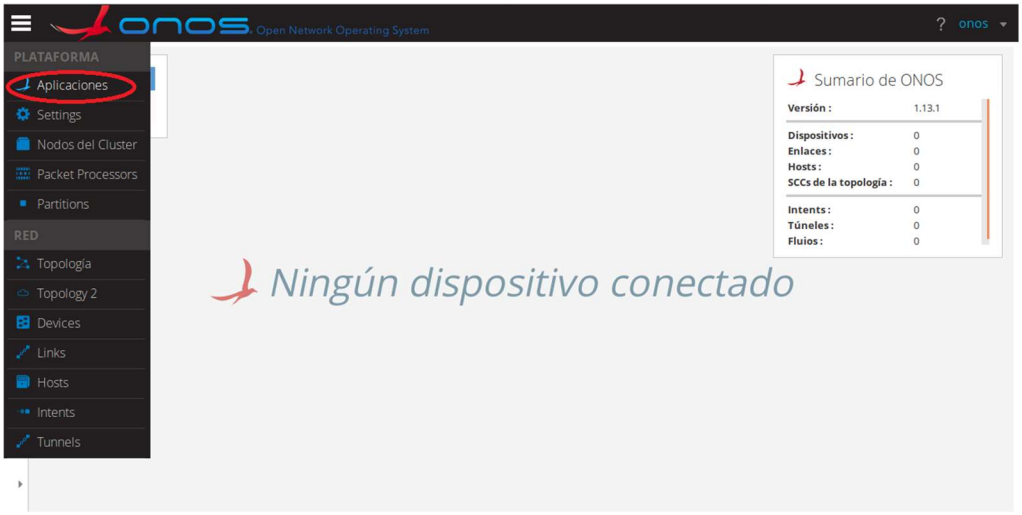


Figura 50 GUI de ONOS

Pinchar en el icono indicado a continuación, y seleccionar el archivo .oar deseado:

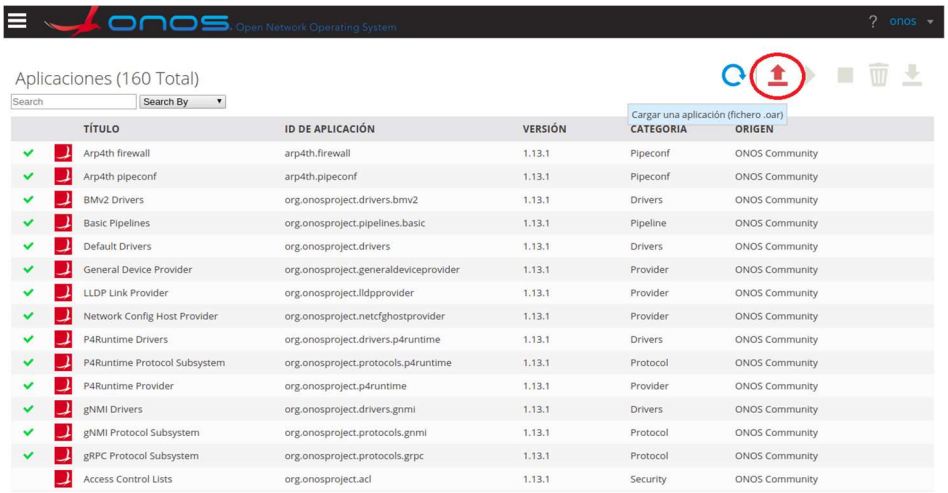


Figura 51 Carga de apliación ONOS

Por último, pinchar en el icono resaltado para activar la aplicación seleccionada:

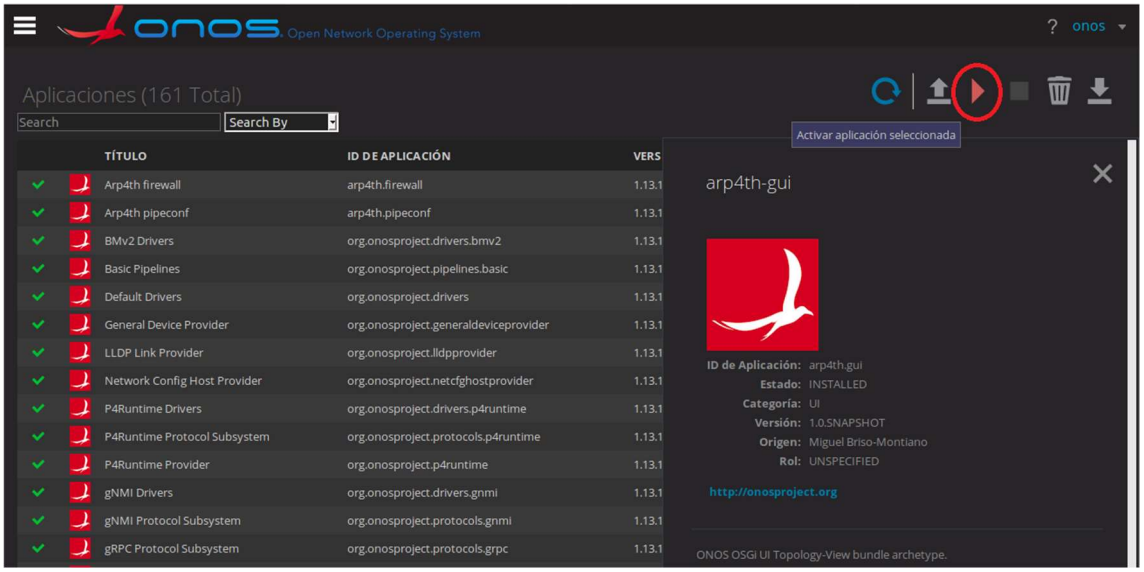


Figura 52 Activación de aplicación ONOS

8.5.3 Uso de la aplicación de visualización de caminos

La vista que proporciona la visualización de caminos se activa pinchando en el icono indicado en la parte inferior izquierda de la imagen:

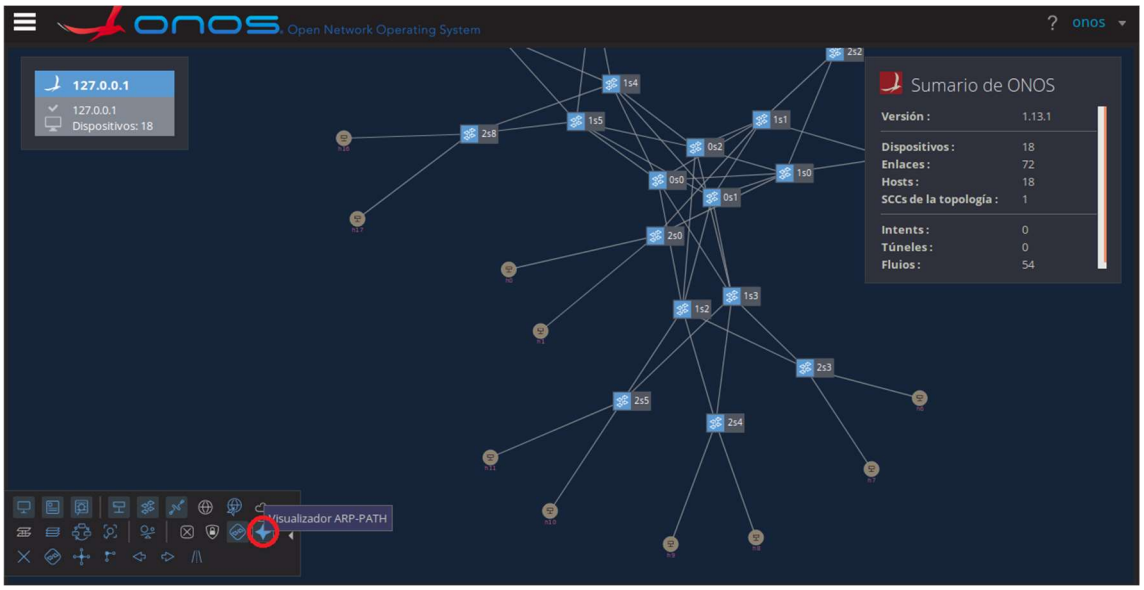


Figura 53 Activación de vista en ONOS

A partir de este momento, se pueden añadir flujos para ser visualizados sobre la topología. Por flujo se entiende un flujo de tráfico unidireccional a nivel de capa 2 entre dos hosts. Para añadir un flujo para monitorizar al visualizador, hay que seleccionar dos hosts (se muestran en la GUI con la tecla h), y posteriormente pinchar en un icono que saldrá en la ventana de *Ítems Seleccionados*, tal como muestra la imagen siguiente:

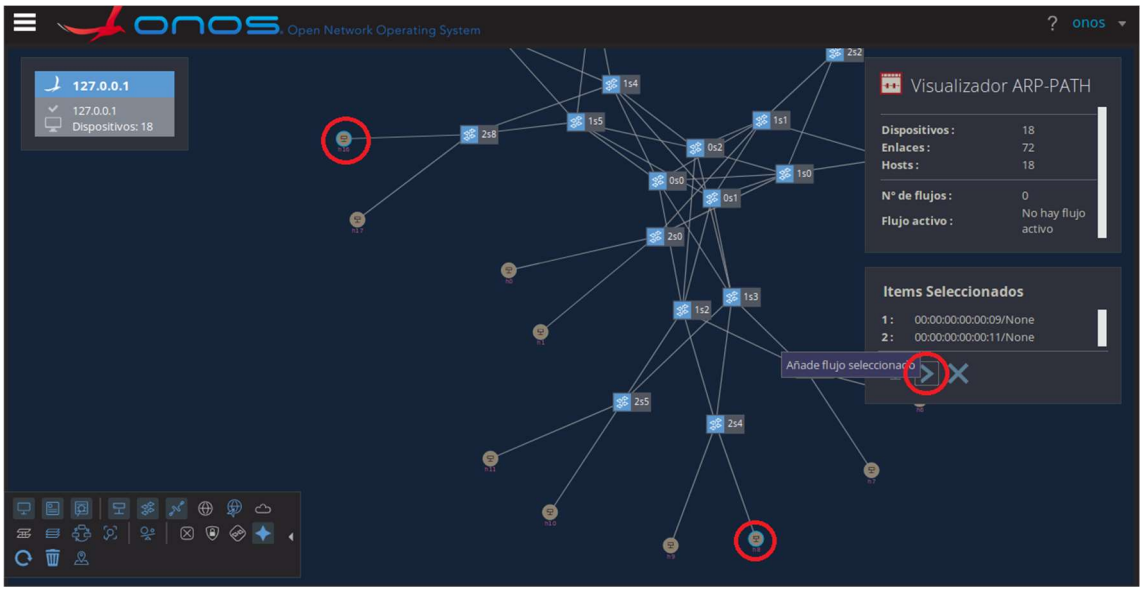


Figura 54 Selección de flujo en arp4th-gui

El navegador mostrará una ventana emergente para confirmar la operación.

El orden en que se seleccionan los hosts es importante, puesto que el primero en serlo es el origen, y el segundo el destino. Para cambiar el flujo activo se pulsa la tecla Enter. Un flujo específico se puede borrar de una forma análoga a su adición, pinchando en la cruz de la ventana de *Ítems Seleccionados*. Se pueden añadir tantos flujos como se deseen, pero únicamente se visualizará uno de ellos, el denominado flujo activo. Los hosts que identifican el flujo activo pueden consultarse en la ventana *Visualizador ARP-PATH*. El resultado de la visualización de un flujo es análogo al siguiente:

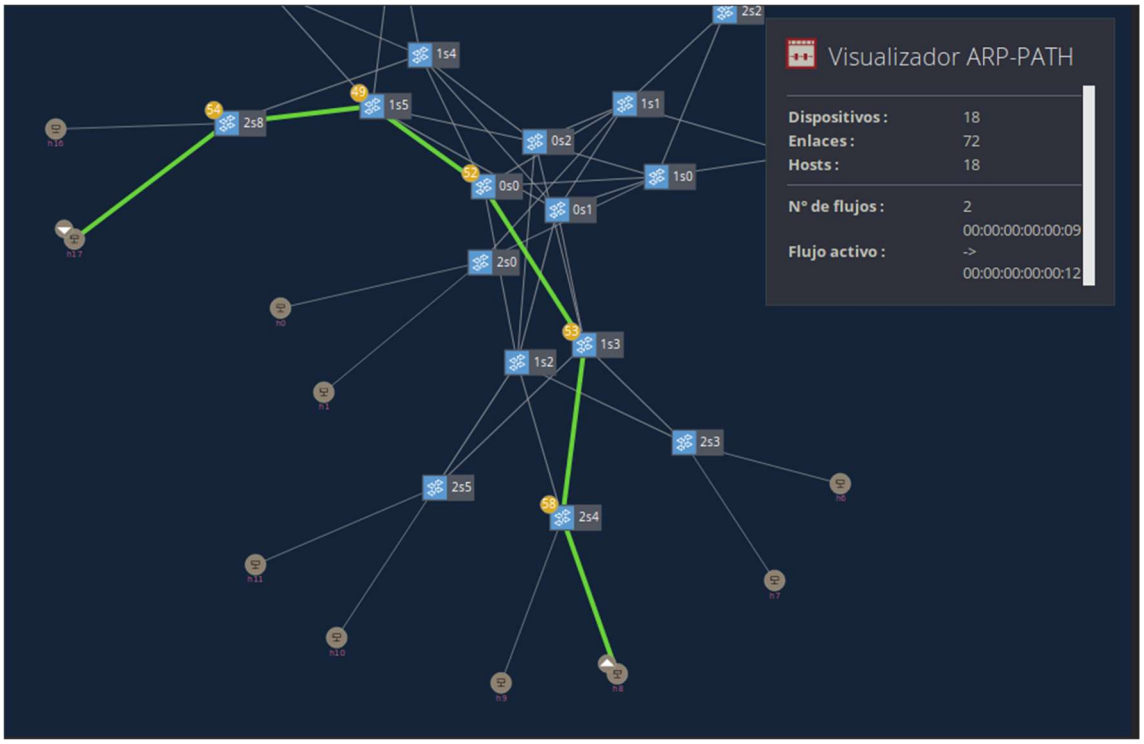


Figura 55 Flujo visualizado en arp4th-gui

La información se recarga periódicamente cada segundo, en función del contenido de las etiquetas recibidas. Si se pincha sobre un switch perteneciente al camino visualizado, se muestran los datos presentes en la etiqueta que insertó en el paquete procesado por ONOS:

Figura 56 Estadísticas en arp4th-gui

Sobre cada switch se muestra un indicador numérico. Este indicador numérico puede mostrar bien el valor de Timedelta en microsegundos, o bien el número de entradas ARP-Path en el switch. Se conmuta entre una u otra opción con la tecla Espacio, mostrándose por pantalla un mensaje con la opción escogida en ese momento:

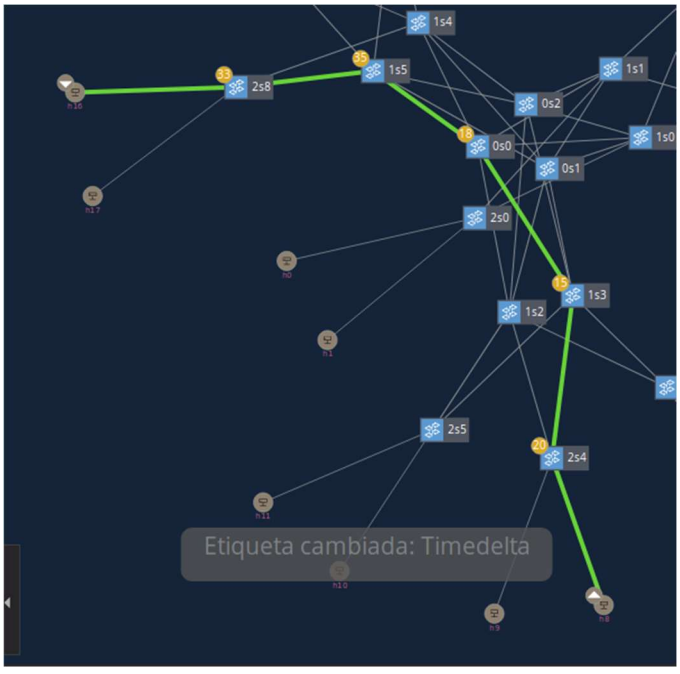


Figura 57 Selección de indicador en arp4th-gui

Por último, pulsando la tecla de Retroceso, se pueden borrar todos los flujos:

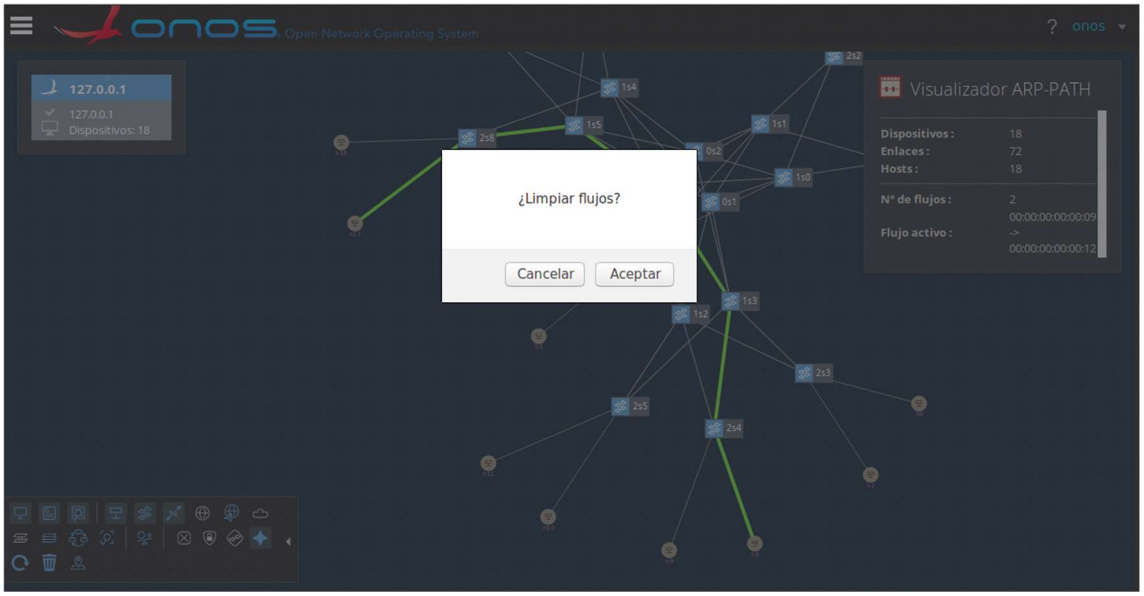


Figura 58 Borrado de flujos en arp4th-gui

8.5.4 Acerca de la versión de P4Runtime de ONOS

Para que el switch BMv2 pueda comunicarse con ONOS, ambos deben utilizar la misma versión de P4Runtime. Es por ello que la versión de P4Runtime con la que se entrega este trabajo no es la versión 1.0, sino la correspondiente a un draft que sí está soportada en ONOS. Se espera que la próxima versión de ONOS, Owl, soporte la versión 1.0 de P4Runtime.

8.5.5 Acerca del número de paquetes etiquetados en el visualizador de caminos

La API de ONOS 1.13 todavía no permite configurar Meters P4 a través del servicio Meter Service. Por ello, los switches frontera origen no tienen limitado el tráfico a etiquetar, por lo que todos los paquetes del flujo afectado se duplican, lo que puede ser indeseable para altas tasas de tráfico. Un parche temporal utilizado es la configuración a nivel de target del Meter para configurar explícitamente que se etiquete 1 o 2 paquetes cada segundo.

Cuando la API de ONOS disponga de las capacidades indicadas, la aplicación de visualización de caminos debería configurar el Meter en el switch frontera origen correspondiente.

8.5.6 Acerca del uso de Mininet con BMv2 y ONOS

Para emular una red de switches BMv2 en Mininet con conectividad con ONOS se deben emplear las clases *ONOSBmv2Switch* y *ONOSHost* a la hora de crear los switches y hosts de la red, respectivamente. Estas clases están definidas en el script adjunto bmv2_ARP-P4.py. La clase *ONOSBmv2Switch* genera un HTTP POST con el cuerpo indicado en el apartado 4.4.3.1, mientras que la clase *ONOSHost* realiza el mismo procedimiento, pero con el cuerpo analizado en el apartado 4.4.3.3.

8.6 Código fuente desarrollado

En el cd adjunto se proporcionan varias carpetas:

- BMv2: contiene el repositorio behavioral-model con los targets *ARP-P4* y *grpc_ARP-P4*.
- ONOS: contiene ONOS compilado con los Pipeconfs desarrollados, listo para ser iniciado.
- Apps_onos: contiene el código fuente de los dos Pipeconfs y la aplicación de visualización de caminos. La aplicación de visualización de caminos se proporciona ya compilada, solamente es preciso cargar el fichero .oar para integrarla en ONOS.
- PTF: Contiene el test unitario referenciado en el apartado 5.2, requiere de la instalación de Packet Test Framework para ser ejecutado.
- Firewall: Contiene dos scripts *p4fw-deploy.sh*, y *p4fw-acl.sh*, cuya utilidad es alternar entre *hybrid.p4* y *acl_hybrid.p4*, e instalar reglas de filtrado para este último programa. Emplean la interfaz REST que expone ONOS.
- Programas P4: Se incluye el código fuente de *hybrid.p4* y *acl_hybrid.p4*, así como los ficheros generados de su compilación.
- Mininet: Se incluyen diversos scripts para la emulación de topologías con la herramienta Mininet y su integración con ONOS.

En cada directorio existe un fichero README en el que se especifica más información del uso de las herramientas disponibles en cada directorio

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá