ISBN: 978-628-96613-3-0. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2026.1.1.514

Desarrollo de Infraestructuras de Red Virtuales Basadas en Dispositivos con el Plano de Datos Programable Utilizando el Lenguaje P4 ejecutándose en un Testbed Básico

Fernando Becerra, M.Sc.[1], David Mejía, M.Sc.[2], and Iván Bernal, Ph.D. [3]
[1,2,3] Escuela Politécnica Nacional, Ecuador, fernando.becerrac@epn.edu.ec, david.mejia@epn.edu.ec, ivan.bernal@epn.edu.ec

***Abstract*– Software Defined Networking (SDN) decouples the control plane from the data plane, delegating control to a central controller that determines how network devices (switches) will operate. These devices perform only switching tasks and possess limited intelligence. As network architecture evolves, the data plane should also be programmable, adding flexibility for changes and improving performance. The P4 programming language is used to define how to customize the data plane of such devices. P4 is a high-level language for programming packet processors, independent of the protocol. General aspects of P4 version 16, the structure of a P4 program, and its correspondence with the processing pipeline in the programmable device are presented. A testbed using open-source software is implemented for compiling, testing, and debugging P4 programs. Additionally, several applications are developed using the testbed infrastructure. The general characteristics of their development and functionality are described.**

Keywords-- P4, SDN, P4 testbed.

I. Introducción

Las SDN (Software Defined Networking) constituyen una arquitectura de red cuya característica fundamental es desacoplar el plano de control (inteligencia) del plano de datos, derivando el control a una computadora (controlador) desde la cual se determinará cómo operarán los dispositivos de red (switches), como se indica en la Fig. 1.

El control sobre la red se lo realiza a través de la instalación de reglas en los dispositivos de red. Dichas reglas se programan en el controlador empleando lenguajes de alto nivel; el dispositivo envía al controlador los paquetes que no sabe cómo procesar, luego de que el controlador los procesa, establecerá y enviará a los dispositivos las reglas que deberán aplicar sobre el tráfico que circula por ellos.

A. Protocolo OpenFlow en las SDN

En las SDN, Openflow especifica el formato y el significado de los mensajes que circulan entre un controlador y un dispositivo de red; por lo tanto, un dispositivo de red también necesita un módulo OpenFlow. El controlador instala un conjunto de reglas de reenvío en cada dispositivo. Cada regla describe un tipo particular de paquete y especifica el puerto de salida a través del cual se debe enviar el paquete. Cuando un paquete llega al dispositivo, su hardware comprueba cada una de las reglas, encuentra una que coincida con el paquete y envía el paquete a través del puerto especificado [2].

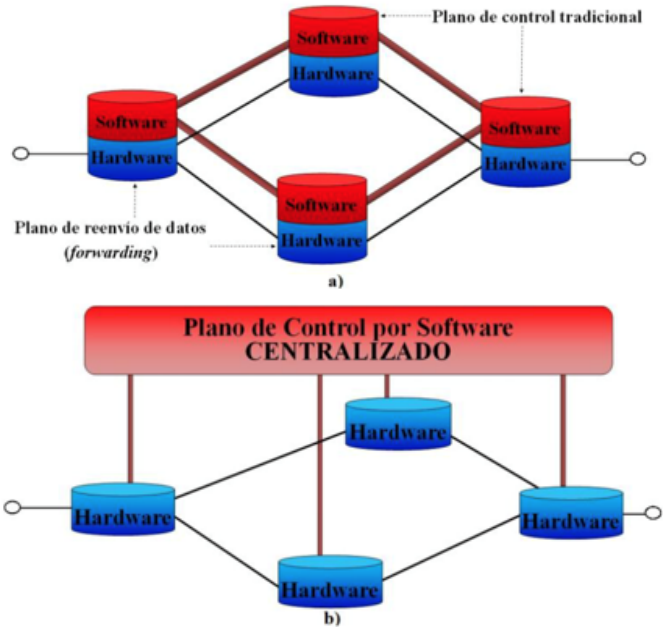

Fig. 1 a) Plano de datos y control distribuidos y co-localizados en una red tradicional; b) Plano de control centralizado en una SDN [1]

Para mejorar los tiempos de respuesta, se requiere evitar que el controlador maneje cada excepción. Una solución es que el controlador instale una aplicación en cada dispositivo. En lugar de un lenguaje de programación convencional, esta nueva generación de SDN utiliza el lenguaje especial llamado P4 (Programming Protocol-independent Packet Processors) [2]. El uso de un programa significa que un dispositivo puede manejar excepciones localmente, sin enviar cada excepción al controlador [3].

B. El Lenguaje de Programación P4

En marzo de 2015, se creó el consorcio del lenguaje P4 y se publicó la variante 1.0.2 denominada versión 14. Actualmente, Open Networking Foundation [4] gestiona el proyecto P4. En mayo de 2017 se publicó la versión 16 de P4, con mejoras significativas con respecto a la anterior [5].

P4 es un lenguaje de alto nivel para programar "procesadores de paquetes" que sean independientes del protocolo. P4 se puede utilizar para configurar dispositivos de red expresando cómo analizar, procesar y reenviar paquetes [7]. En definitiva, añade programabilidad al plano de datos.

Un programa P4 contiene: encabezados, analizadores (*parsers*), tablas, acciones y programas de control. En P4, un *pipeline* o línea es la secuencia de etapas por las que atraviesa un paquete dentro del dispositivo de red mientras se analiza, procesa y reenvía, como se representa en la Fig. 2.

Un programa P4 se compila y ejecuta en dispositivos basados en modelos de reenvío abstractos comunes, lo que significa que el programa no se escribe para un hardware específico, sino para una arquitectura lógica estandarizada que representa cómo un dispositivo procesa paquetes.

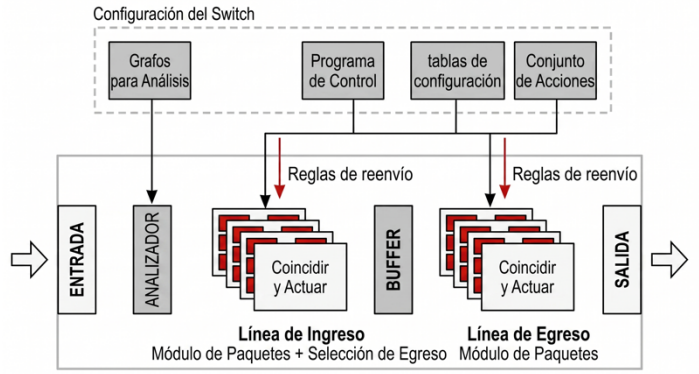

Fig. 2 Modelo de reenvío de paquetes [6]

P4 permite configurar y programar la manera en la que un dispositivo procesa los paquetes, incluso después de que el dispositivo sea desplegado, eliminando así la necesidad de comprar nuevos dispositivos para soportar nuevas funciones. Esta innovación soluciona la insuficiente programabilidad de OpenFlow; además, P4 permite que los dispositivos soporten el reenvío de paquetes de forma independiente del protocolo, lo que significa que puede soportar diferentes protocolos con diferentes programas P4 [6].

El resto de este trabajo se organiza de la siguiente manera: La sección II presenta generalidades sobre P4. La sección III describe el *pipeline* de procesamiento, la sección IV aborda la estructura de un programa P4, incluyendo un ejemplo básico con explicación del código y su semántica, así como su relación con la estructura física del dispositivo a generarse. La sección V presenta la ejecución de un programa P4 y la interacción del dispositivo con el plano de control. La sección VI expone las aplicaciones desarrolladas. Finalmente, la Sección VII presenta las conclusiones del trabajo realizado.

II. Generalidades sobre P4

A. Especificación de P4 versión 16

La Fig. 3 presenta dos grupos de construcciones: a) Sobre la línea entrecortada hay conceptos generales que son comunes a todas las implementaciones de P4; b) Bajo la línea entrecortada se encuentran definiciones y componentes específicos de la arquitectura.

B. Elementos fundamentales del lenguaje de programación P4

P4 es un lenguaje de dominio específico (DSL), así tiene conceptos asociados como búsquedas de tablas que coinciden con una búsqueda de tablas en hardware, pero, por ejemplo, una búsqueda de éstas no puede invocarse más de una vez para mantener el rendimiento de manera predecible.

Los elementos centrales del lenguaje P4 se muestran en la Fig. 3 y son [5]:

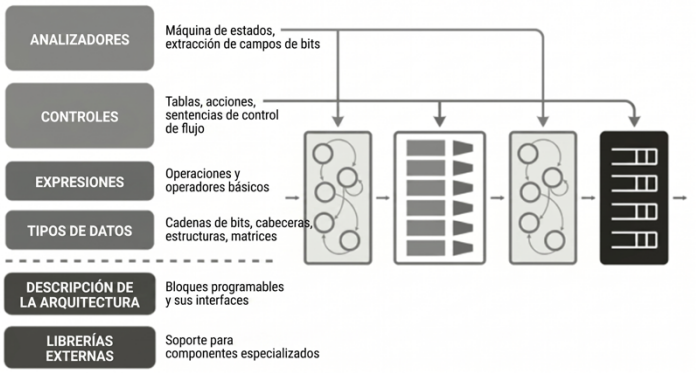

Fig. 3 Componentes del Lenguaje P4 [5]

1) Analizadores (parsers): Los *parsers* describen cómo extraer los encabezados de un paquete. Los *parsers* se modelan como máquinas de estados finitos que leen campos de un paquete de forma secuencial. Cada estado puede extraer uno o más encabezados y decidir la transición al siguiente estado en función de los valores de los campos extraídos.

2) Controles: Los controles control definen la lógica de procesamiento de paquetes luego de lo realizado por el *parser*. Incluyen llamadas a acciones, aplicación de tablas y lógica condicional. Las acciones encapsulan operaciones que se ejecutan sobre los paquetes o metadatos. Las tablas definen la relación entre llaves (keys) de búsqueda y acciones, permitiendo el control dinámico del comportamiento del plano de datos mediante reglas instaladas desde el plano de control. Así, en un contexto SDN, el controlador instala entradas en las tablas P4 mediante una API como P4Runtime. Las tablas actúan como puntos donde el plano de control decide el comportamiento del dispositivo (forwarding, mirroring, QoS, etc.).

3) Expresiones: Los *parsers* y controles se escriben mediante expresiones. Las expresiones incluyen sentencias estándar con operadores booleanos de campos a nivel de bit; comparación aritmética simple con números enteros y operadores condicionales; operaciones sobre encabezados; etc. P4 soporta estructuras condicionales if-then-else dentro de los bloques de control. Estas estructuras permiten tomar decisiones basadas en campos de encabezados, metadatos o resultados de tablas, facilitando la implementación de políticas de reenvío y filtrado.

4) Tipos de datos: P4 tiene un sistema base de tipos estáticos. Los tipos de datos incluyen nulos, booleanos, cadenas de bits de tamaño fijo, enteros con signo, el tipo match_kind para describir la implementación de tablas de búsquedas, tipos estructurados para gestionar cabeceras y

estructuras, etc. Este sistema de tipos permite un control sobre el tamaño en bits y la semántica de los datos. Se debe especificar los formatos de cabeceras que se van a reconocer y procesar.

5) Descripciones de la arquitectura: Es la descripción de la arquitectura del dispositivo de red siendo la arquitectura una abstracción del dispositivo subyacente, constituye la forma en la que los programadores de P4 piensan sobre el plano de datos del dispositivo. La arquitectura identifica los bloques programables; por ejemplo: cuántos parsers, deparsers, líneas de Coincidir + Acción y su relación. P4 es agnóstico de la arquitectura y es el vendedor del hardware quien define la arquitectura. Con base en la arquitectura se tienen los dispositivos objetivos capaces de ejecutar un programa P4. En general, son dispositivos de hardware, pero la versión 16 de P4 también admite switches en software como OVS (Open vSwitch). En el caso de hardware, el dispositivo puede ser estructurado con base en los FPGA o los ASIC.

6) Librerías Externas: Los programas P4 pueden invocar servicios implementados por objetos y funciones externas. La construcción externa describe las interfaces que expone un objeto al plano de datos. Por ejemplo, funciones CRC (Cyclic Redundancy Check*)*, marcas de tiempo (timestamp) y temporizadores.

El vendedor de hardware proporciona el compilador P4 que produce archivos de configuración binarios para el hardware. Los binarios asignan y particionan los recursos de los dispositivos. Estos recursos pueden ser manipulados en tiempo de ejecución mediante la runtime API de P4. Esta API permite, por ejemplo, agregar y remover entradas de tablas, leer contadores, programar medidores, inyectar y recibir paquetes de control, etc.

III. PIPELINE DE PROCESAMIENTO

Un programa en P4 tiene una estructura que refleja el *pipeline* de procesamiento en un dispositivo programable (Fig.4). La idea es que el desarrollador defina cómo se realiza el parsing, pipeline y *deparsing* de los paquetes, y qué acciones se ejecutan sobre ellos. A continuación, se describen los bloques más comunes del *pipeline*:

1) Parser: Primera etapa; extrae los encabezados del paquete y obtiene campos estructurados. Ejemplo: Ethernet → IPv4 → TCP. Salida: encabezados y metadatos.

2) Verificar Checksum: Etapa opcional. Valida los checksums de los encabezados (ejemplo: checksum de IPv4).

3) Control de entrada (ingress): Etapa inmediatamente posterior al *parsing* y que es el principal bloque de procesamiento. Aquí se utilizan tablas de Coincidir+Acción (*match–action*). Entre las tareas que puede realizarse están: clasificación, filtrado, decisiones de reenvío, configuración de metadatos (como el puerto de salida). Ejemplos: tabla de conmutación L2, tabla LPM (Longest Prefix Match) de IPv4.

4) Control de salida (egress): Puede modificar los encabezados nuevamente. Por ejemplo: insertar etiquetas VLAN, reducir el TTL (*Time To Live*), añadir telemetría. Se aplican las modificaciones finales a los encabezados, así como políticas. Las copias de *multicast* se procesan individualmente. Los paquetes se clonan si es necesario.

5) Calcular checksums: Recalcula los *checksums* si se modificaron los encabezados.

6) Deparser: especifica cómo se reconstruye o reensambla el paquete antes de ser reenviado. Define el orden y la inclusión de los encabezados que serán serializados nuevamente en el paquete para finalmente ser transmitidos.

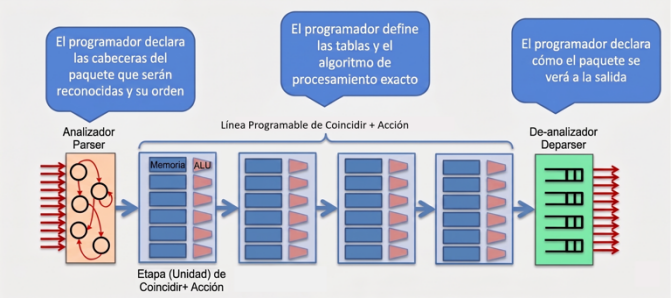

Fig. 4 Pipeline de procesamiento [5]

IV. ESTRUCTURA DE UN PROGRAMA EN P4

Un programa en P4 está conformado por definiciones y bloques específicos.

A. Encabezados (headers)

Una ventaja de P4 es que permite procesar no solo encabezados (headers) de protocolos estandarizados (Ethernet, IP, TCP, etc.), como los switches o enrutadores tradicionales, sino también permite procesar encabezados totalmente personalizados (customized).

En P4 no hay variables independientes. Todo está organizado en estructuras de tipo header que incluyen a los encabezados en sí mismo y metadatos. Las estructuras están alineadas a nivel de bytes y ofrece operaciones para probar y asignar valor al bit de validez: isValid(), setValid() y setInvalid(). Todos los encabezados (tanto conocidos como personalizados), así como su análisis, deben definirse explícitamente en el programa P4.

B. Componentes de un Programa P4 y ejemplo básico desarrollado

Un programa P4 incluye los siguientes bloques:

1) Librerías y definición de constantes
2) Definición de headers y metadata
3) Agrupación de headers
4) Parser
5) Acciones y tablas
6) Controles (ingress/egress)
7) Deparser
8) *Main* (*pipeline*)

Estos componentes trabajan conjuntamente para describir el procesamiento de los paquetes.

Se presenta a continuación un ejemplo de un enrutador, que permite visualizar cómo se usan los conceptos presentados

y se proporcionan ideas básicas sobre construcciones del lenguaje P4; el ejemplo usa como objetivo el switch BMv2 que puede implementar la arquitectura "V1 model" o PSA (Portable Switch Architecture).

1) Librerías y constantes

#include <core.p4>

- Tipos fundamentales (bit<>, int<>)
- Construcciones del lenguaje (header, struct, parser, control)
- Operaciones básicas y semántica del lenguaje

#include <v1model.p4>

- standard_metadata_t
- packet_in, packet_out
- Firmas obligatorias de: Parser; Ingress; Egress; Deparser.

2) Definición de headers

```
header ethernet_t {
   bit<48> dstAddr;
   bit<48> srcAddr;
   bit<16> etherType;
}
header ipv4_t {
   bit<4>  version;
   bit<4>  ihl;  // Internet header length
   bit<8>  diffserv;
   bit<16> totalLen;
   .....
   bit<32> srcAddr;
   bit<32> dstAddr;
}
```

3) Parser

```
parser MyParser(packet_in pkt, out headers hdr) {
   state start {
      pkt.extract(hdr.ethernet);
      transition accept;
   }
}
```

Los *parsers* reciben paquetes (packet_in) y entregan encabezados (*headers*) y metadatos, y están escritos como una máquina de estados (FSM), como se presenta en la Fig.5.

- «*state*» declara un estado del *parser*, es decir, define un nodo de una FSM.
- Todo parser tiene tres estados predefinidos: «*start*»; «*accept*»; y, «*reject*».
- Todo *parser* debe tener e iniciar en el "estado de entrada" «*start*».
- El programador puede definir otros estados.
- En cada estado, se ejecutan cero o más sentencias y luego se pasa a otro estado (se permiten lazos).
- packet_in.extract es una de las funciones externas proporcionadas por el modelo v1.

Semántica de «*start*»

- «*start*» es el primer estado que se ejecuta cuando el *parser* empieza a procesar un paquete entrante.
- A partir de aquí, el *parser* decide qué hacer: normalmente extrae el primer encabezado (p. ej., Ethernet) o pasa directamente a «*accept*» en caso de no requerirse ningún análisis.
- Se puede pasar a otro estado (transition).

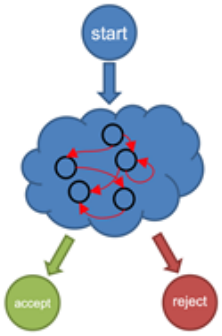

Fig. 5 FSM para un parser [2]

Semántica de «*transition accept*» y «*transition reject*»

- Indican al *parser* que el análisis ha finalizado correctamente y el control sale del *parser* y pasa al «*ingress pipeline*».
- Los «*headers*» extraídos quedan disponibles para el procesamiento en «*ingress pipeline*».
- El paquete se considera válido con «*transition accept*», a diferencia de «*transition reject*» que indica un error de análisis y normalmente se procede al descarte del paquete o se envía al controlador SDN. «*accept*» y «*reject*» son estados terminales.

4) Actions y tables

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

«*action*» declara una unidad de comportamiento que se invoca desde una tabla, un «*apply block*» o una acción por defecto.

- «*action*» está asociada a una key (llave) en una tabla.
- standard_metadata está definida en v1model.p4.

```
table ipv4_lpm {
   key = { hdr.ipv4.dstAddr : lpm; }
   actions = { forward; drop; }
   size = 1024;
}
```

Una tabla en P4 es una abstracción lógica que representa una estructura de decisión programable, cuyo contenido es gestionado por el plano de control.

Declaración de la tabla

- «*table ipv4_lpm*» declara una tabla con el nombre *ipv4_lpm*.
- La tabla no se ejecuta automáticamente; debe ser invocada explícitamente mediante «*apply( )*».

Definición de la clave

- «*key*» define el campo para búsqueda en la tabla.

- «*hdr.ipv4.dstAddr*» corresponde a la dirección IPv4 de destino, previamente extraída por el *parser*.
- El tipo de coincidencia «*lpm*» indica que se seleccionará la entrada de la tabla con el prefijo más largo que coincida, lo cual es característico de las tablas de enrutamiento IP.

Acciones permitidas

- «*actions*» define el conjunto de acciones que la tabla puede ejecutar. En el ejemplo: «*forward*» y «*drop*».
- El plano de control solo puede asociar entradas de la tabla a este conjunto de acciones.
- Las acciones pueden provocar modificaciones sobre encabezados, metadatos o algún estado del sistema.

Tamaño de la tabla

- «*size = 1024*» especifica el número máximo de entradas que la tabla puede contener.
- Este valor sirve como guía para el compilador y para la asignación de recursos en el dispositivo objetivo. En el hardware real, este tamaño puede estar sujeto a limitaciones adicionales.
- Si un controlador SDN intenta agregar más entradas que el tamaño declarado, la operación fallará.

Semántica de Ejecución

- Para que la tabla tenga efecto, debe invocarse dentro de un bloque de control usando «*ipv4_lpm.apply( )*».
- Durante la ejecución, si el encabezado IPv4 es válido, se evalúa la clave, se realiza la búsqueda LPM y, en caso de coincidencia, se ejecuta la acción asociada.
- Si no existe coincidencia, se ejecuta una acción definida por defecto o ninguna operación.

Dependencia de la Arquitectura

- Aunque la definición de la tabla es independiente del hardware, su implementación concreta depende de la arquitectura P4 utilizada, como v1model o PSA.
- El tipo de memoria, el soporte para LPM y el tamaño efectivo de la tabla son determinados por el compilador y el dispositivo objetivo.

5) Control ingress

```
control MyIngress(inout headers hdr, inout metadata meta) {
   apply {
      if (hdr.ipv4.isValid()) {
         ipv4_lpm.apply();
      }
   }
}
```

Este código implementa un patrón clásico en P4 que consiste en aplicar lógica de reenvío de IP únicamente cuando el paquete contiene un encabezado IPv4 válido. En este ejemplo no se procesa el header de Ethernet.

Definición del bloque de control

- La sentencia «*control MyIngress(...)*» define un bloque de control denominado MyIngress.
- «*control*» representa una unidad de procesamiento que se ejecuta en una etapa específica del pipeline, típicamente en el ingreso.
- Los parámetros «*hdr*» y «*meta*» se pasan con el modificador «*inout*», indicando que en el control se puede leer, así como modificar su contenido.

Bloque «*apply*»

- «*apply*» es obligatorio en toda sentencia «*control*» y define el cuerpo ejecutable del control.
- Sobre cada paquete que atraviesa esta etapa del *pipeline* se ejecuta exactamente una vez su contenido.
- «*if (hdr.ipv4.isValid())*» verifica si el encabezado IPv4 ha sido marcado como válido por el *parser*, es decir, si el encabezado fue extraído correctamente del paquete.

Aplicación de la tabla ipv4_lpm

- «*ipv4_lpm.apply()*» invoca la evaluación de la tabla *ipv4_lpm*.
- Durante la invocación, las llaves de la tabla se comparan con las entradas instaladas por el plano de control y si hay coincidencia, se ejecuta la acción asociada; en caso contrario, se ejecuta la acción por defecto.

6) Control egress

```
control MyEgress(inout headers hdr,
                 inout metadata_t meta,
                 inout standard_metadata_t standard_meta) {
   apply { }
}
```

En esta etapa se puede realizar tareas como: reescribir MAC origen/destino; cambiar VLAN ID; modificar DSCP/ToS; agregar una VLAN tag; crear copias para monitoreo.

- En el ejemplo, «*control MyEgress*» es semánticamente válido, pero funcionalmente nulo.
- Lo presentado se utiliza típicamente como plantilla inicial para extensión futura o simplemente cuando la lógica egress no es necesaria.

7) Deparser

```
control MyDeparser(packet_out pkt, in headers hdr) {
   apply {
      pkt.emit(hdr.ethernet);
      pkt.emit(hdr.ipv4);
   }
}
```

El código presentado para el *deparser* implementa una reconstrucción mínima y explícita del paquete, limitándose a los encabezados Ethernet e IPv4. No se incluyen trailers, payloads explícitos ni encabezados de capas superiores, lo que implica que el payload original será agregado automáticamente después de los encabezados emitidos.

Definición del bloque de control

- «*control MyDeparser (...)*» define un bloque de control llamado MyDeparser.
- En este caso, «*control*» está usado para funcionar como un *deparser*.
- packet_out representa el paquete de salida.
- headers es una estructura de encabezados de solo lectura (in) y contiene los encabezados previamente sometidos al parsing y posiblemente modificados en las etapas de *parser* y control de procesamiento.

Bloque «*apply*»

- «*apply*» define el bloque de instrucciones que se ejecutan cuando se invoca al control en el pipeline.
- En un *deparser*, «*apply*» especifica explícitamente el orden y la selección de encabezados que serán serializados en el paquete de salida.
- pkt.emit(hdr.ethernet) serializa el encabezado Ethernet contenido en hdr en un paquete de salida. La posición del encabezado Ethernet en el paquete de salida depende del orden de las llamadas emit.
- pkt.emit(hdr.ipv4) serializa el encabezado IPv4 inmediatamente después del encabezado Ethernet. El orden de emisión garantiza que el paquete final tenga la estructura Ethernet + IPv4.
- Si existieran otros encabezados (por ejemplo, TCP o UDP), deberían ser emitidos explícitamente para formar el paquete completo.

8) Main (pipeline)

- Es necesario instanciar y conectar el *pipeline*.

```
V1Switch(
   MyParser(),
   MyIngress(),
   MyEgress(),
   MyDeparser())   main;
```

V. EJECUCIÓN DE UN PROGRAMA P4 E INTERACCIÓN CON EL PLANO DE CONTROL

Una vez compilado y ejecutado un programa en P4, las tablas y otros objetos definidos en el código están presentes en el plano de datos. La única entidad que aún falta por incorporarse es el plano de control. El usuario del dispositivo definido en P4 puede crear el plano de control, que no es práctico en todos los casos, o se puede usar software de una infraestructura existente (como un controlador SDN), extendiéndolo con un conjunto de funciones que permitan una comunicación efectiva con el plano de datos creado con base a P4. El plano de datos ahora puede ser manipulado por el plano de control en tiempo de ejecución, como se indica en la Fig. 6.

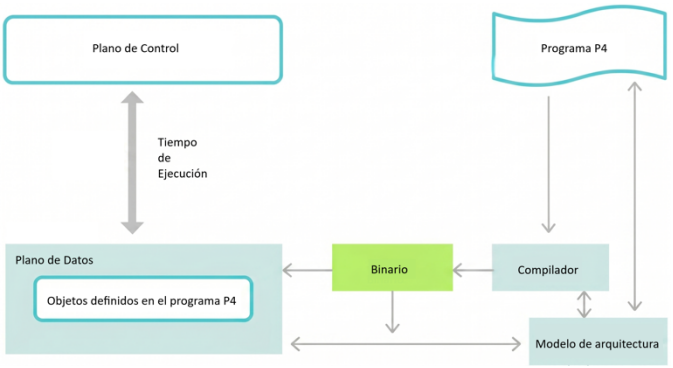

Fig. 6 Interacción de plano de datos y control con dispositivo programable [8]

El término "tiempo de ejecución" (runtime) se refiere a las operaciones que tienen lugar luego de la configuración inicial del dispositivo objetivo. Estas operaciones incluyen la actualización de la configuración del dispositivo para el reenvío de paquetes, el llenado de tablas para el posterior procesamiento de paquetes, etc.

P4Runtime es una especificación de tiempo de ejecución que ofrece una API que puede usarse desde el plano de control justamente para el control estándar del plano de datos definido por P4 e implementado en el dispositivo. P4Runtime es una API basada en gRPC/protobuf [1] que permite la generación automática de código cliente/servidor para numerosos lenguajes de programación y que puede utilizarse tanto para aplicaciones de plano de control local (usando la línea de comandos), así como remoto (con un controlador SDN).

A. Testbed para crear redes virtuales y desarrollar aplicaciones con P4

Se implementó un *testbed* totalmente virtualizado requerido para desarrollar y probar los programas escritos en P4, que incluye con una arquitectura SDN con plano de control programable y un plano de datos programable. Esto se realizó con base al análisis realizado de las características de los componentes de software de libre distribución que se tiene a disposición, en particular en lo relativo a switches virtuales. A más del sistema operativo base, una distribución de Linux, se instaló el software del dispositivo virtual BMv2 que, si bien no es un switch de producción, es un modelo de comportamiento cuyo objetivo principal es el desarrollo, pruebas, depuración y experimentación. BMv2 ejecuta programas P4 ligados a una arquitectura (modelo V1 o PSA). Con opennetworking/p4mn se instancia switches BMv2 con soporte P4Runtime y se genera automáticamente artefactos de ejecución (logs, puertos gRPC y archivos de configuración netcfg) en el directorio temporal (/tmp).

El *testbed* además incluye, entre los elementos principales, la herramienta de compilación p4c del lenguaje P4; la interfaz de control P4Runtime (gRPC); Mininet para emular redes SDN, permitiendo la creación de redes virtuales; y, el controlador ONOS para el plano de control de una SDN. Finalmente, para algunas pruebas ha resultado de suma utilidad el paquete P4-Utils [9] de Python que permite crear y probar redes virtuales que incluyan switches P4, como extensión de Mininet.

Esta automatización permite reproducibilidad y trazabilidad de las ejecuciones, al conservar de forma persistente los puertos gRPC asignados, los logs de BMv2 y el archivo bmv2-<switch>-netcfg.json, utilizado por el controlador. Para cada programa en P4_16 se siguió un ciclo de ingeniería controlado: compilación, despliegue del pipeline, programación del runtime y validación con tráfico de prueba y lectura de contadores.

El *testbed* permite, por lo indicado, disponer de un ambiente para realizar las pruebas necesarias para que los elementos de software puedan interactuar de forma adecuada, así también permite usarlo para ejecutar, probar y depurar los programas escritos en P4.

B. Algunos aspectos relevantes sobre la interacción de dispositivos con soporte P4 y el controlador ONOS

A partir de un programa P4, el compilador p4c genera dos artefactos principales: P4Info y BMv2 JSON. El archivo P4Info describe la interfaz lógica visible para el plano de control, incluyendo metadatos, identificadores de tablas, acciones, parámetros y tipos de coincidencia definidos en el programa P4. Por ejemplo, tablas match-action como l2_forwarding_table, acciones como set_egress_port(port), send_to_cpu() y standard_ metadata.egress_spec quedan expuestos para que un controlador como ONOS pueda referenciarlos mediante P4Runtime. En cambio, el archivo BMv2 JSON contiene la representación compilada del plano de datos que ejecuta BMv2, incluyendo la lógica del parser, los controles Ingress y Egress, las tablas, las acciones y el Deparser. Por esta razón, BMv2 no ejecuta directamente el código fuente P4, sino su descripción compilada en JSON, mientras que ONOS utiliza P4Info como contrato de programación del pipeline.

El mecanismo de pipeconf en ONOS permite asociar a cada dispositivo el archivo P4Info y el archivo BMv2 JSON. Adicionalmente, se debe generar manualmente o mediante un script la configuración Netcfg que declara por cada dispositivo la managementAddress, el p4DeviceId, el driver bmv2 y el identificador del pipeconf a cargar. La arquitectura de integración del testbed se muestra en la Fig. 7.

Con la información indicada, ONOS ejecuta SetForwardingPipelineConfig y establece la sesión P4Runtime con cada instancia de BMv2. Una vez activa la sesión, el controlador puede programar dinámicamente el plano de datos mediante mensajes WriteRequest, insertar entradas TableEntry en las tablas definidas en P4, consultar objetos mediante ReadRequest, leer contadores y gestionar eventos PacketIn/PacketOut.

VI. APLICACIONES DESARROLLADAS CON P4

A. Funcionalidades básicas desarrolladas sobre el Testbed

Se describe a continuación el desarrollo de tres programas escritos en P4, relativos a funcionalidades básicas que permiten demostrar las características ofrecidas por P4. Todos los programas incluyen: la sección de inclusión de librerías; definición de headers; la lógica en el control de egreso no se emplea en los ejemplos; finalmente, en el deparser se emiten los headers para la reconstrucción de los paquetes. En la descripción de las aplicaciones se hace referencia únicamente a aspectos relevantes de uno o varios de los siguientes elementos: a) Parsers; b) Lo relativo a las tablas y acciones; c) El control de ingreso, en particular en el bloque «apply», se usan las tablas definidas empleando condiciones sobre campos de los headers entregados por el parser; d) aspectos correspondientes a ONOS; y, e) las pruebas realizadas.

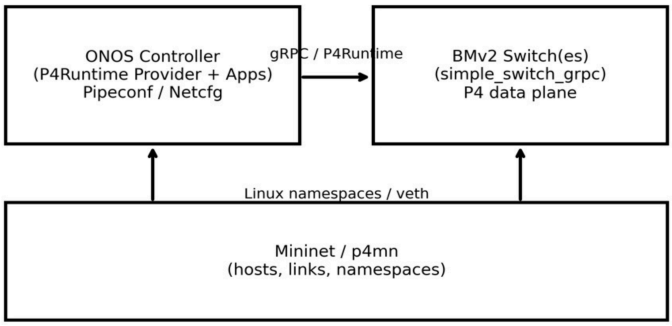

Fig. 7. Arquitectura de integración del testbed: ONOS

1) Programa P4 #1: Descubrimiento y host learning. En este ejemplo, los servicios de ONOS utilizan la interacción con el dispositivo P4 descrita en la sección anterior para detectar dispositivos, enlaces y hosts; mientras que el pipeline P4 define cómo se procesan las tramas dentro de BMv2, cómo se identifican los campos relevantes de la cabecera Ethernet y cómo se exponen las tablas y metadatos requeridos para la programación del plano de datos.

El despliegue del pipeline P4 con ONOS se estructuró en tres etapas: generación de artefactos, configuración pipeconf/netcfg y establecimiento de la sesión P4Runtime.

El bloque MyParser extrae la cabecera Ethernet desde el paquete de entrada mediante pkt.extract(hdr.ethernet) y transfiere el procesamiento a la etapa de ingreso cuando el parsing finaliza correctamente.

Con los dispositivos en estado AVAILABLE se activaron servicios de ONOS para descubrimiento y host learning; en particular, lldpprovider y hostprovider permiten construir el grafo lógico (*switches*, enlaces y *hosts*) a partir de LLDP y tramas de host, mientras que proxyarp reduce el flooding al resolver ARP desde el controlador. Sobre esta base, se habilitó forwarding reactivo (instalación de reglas a demanda) de forma que el primer paquete de un flujo genere packet-in hacia

ONOS y el controlador programe el plano de datos mediante WriteRequest. Este esquema demuestra control efectivo del plano de datos y trazabilidad de decisiones de forwarding. La topología usada se presenta en la Fig. 8.

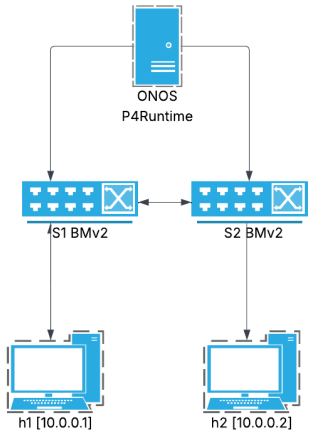

Fig. 8. Topología usada en pruebas con ONOS

Se definieron pruebas de caja negra (desde hosts) y de caja blanca (desde el controlador) con criterios de aceptación explícitos. Las pruebas incluyeron:

1) *Descubrimiento:* para confirmar visibilidad completa*;*
2) *Programación del plano de datos (flows -s):* para evidenciar instalación de reglas base y reglas de forwarding; conectividad end-to-end (ping, pruebas TCP/UDP) para comprobar el comportamiento bajo políticas ACL; y
3) *Repetibilidad:* limpiando reglas e intentos antes de re-ejecutar los casos.

El mecanismo de Pipeconf en ONOS fue configurado para desplegar el archivo P4Info y el binario JSON de BMv2, estableciendo una sesión de arbitraje maestro/esclavo, necesaria para la gestión persistente del plano de datos

2) Programa P4 #2: Conmutación L2 basada en tablas match-action. Se implementó un conmutador L2 (l2switch); el parser extrae la cabecera Ethernet y habilita el acceso a los campos en hdr.ethernet (srcAddr, dstAddr y etherType), lo que permite identificar tramas ARP e IPv4 dentro del pipeline. En la sección de tablas y acciones, se define la tabla l2_forwarding_table, configurada con coincidencia exacta sobre hdr.ethernet.dstAddr. Esta tabla asocia cada dirección MAC de destino con una acción de reenvío, por ejemplo, set_egress_port(port), que asigna el puerto de salida mediante standard_metadata.egress_spec. Cuando no hay un match, se ejecuta como acción por defecto send_to_cpu(), para derivar la trama al plano de control para su procesamiento reactivo.

En el bloque Ingress, la lógica de conmutación requiere verificar la validez de la cabecera Ethernet y aplicar la tabla mediante l2_forwarding_table.apply(). Si la búsqueda produce un match, el paquete es reenviado por el plano de datos; caso contrario, el switch genera un mensaje PacketIn hacia ONOS por el canal P4Runtime/gRPC, incluyendo la trama original y metadatos asociados al puerto de ingreso. El controlador procesa el payload recibido, aprende la asociación entre la dirección MAC de origen y el puerto de entrada, e instala nuevas reglas en el plano de datos con mensajes WriteRequest que contienen entradas TableEntry para l2_forwarding_table. Cuando es necesario reenviar la primera trama que originó el evento, ONOS emite un mensaje PacketOut hacia el switch. Este mecanismo separa la lógica de aprendizaje, ejecutada en el plano de control, del reenvío de tramas conocidas, ejecutado en el plano de datos con tablas match-action. La topología utilizada se presenta en la Fig. 9.

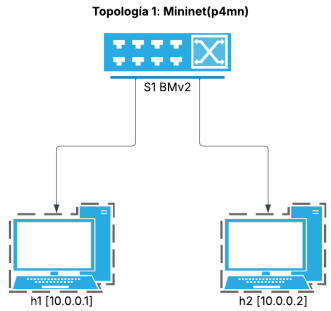

Fig. 9. Topología de prueba para validación de conmutación L2

La tabla MAC se aprendió dinámicamente desde el plano de control utilizando la API P4Runtime, mediante mensajes WriteRequest con entradas TableEntry que representan asociaciones del tipo MAC destino → puerto de salida. Para validar el funcionamiento del pipeline se utilizaron pruebas con tráfico ICMP entre hosts, verificando que las tramas con coincidencia en l2_forwarding_table fueran reenviadas directamente por el plano de datos, sin intervención adicional del controlador. Además, se inspeccionaron los contadores y registros de BMv2 para confirmar que los aciertos en la tabla correspondieran al tráfico generado. Este procedimiento permitió diferenciar fallas de compilación, despliegue del pipeline, programación del runtime y ejecución del reenvío. En la Tabla I se presenta un ejemplo de entradas de forwarding L2 para la topología de la Fig. 8.

Las pruebas cubrieron tanto casos positivos como condiciones que fuerzan la intervención del plano de control. En la topología de la Fig. 8 se realizaron: resolución ARP end-to-end, pruebas de conectividad mediante ICMP con los hosts 1 y 2 con éxito sostenido tras poblar la tabla, y validación de contadores de la tabla para confirmar que el forwarding ocurre por match-action y no por flooding. La evidencia de estas pruebas se complementa con los logs de BMv2 y el incremento de contadores asociados a la tabla de forwarding.

TABLA I
EJEMPLO DE ENTRADAS DE FORWARDING L2 PARA LA TOPOLOGÍA CON UN SOLO CONMUTADOR

Host	MAC	Puerto egress (s1)
h1	00:00:00:00:00:01	1
h2	00:00:00:00:00:02	2

3) Programa P4 #3: ACL L3/L4 con filtrado previo al forwarding. Este programa extiende el pipeline del anterior con una etapa de control de acceso (ACL) previa al forwarding El parser extrae inicialmente la cabecera Ethernet y, si el campo hdr.ethernet.etherType identifica tráfico IPv4, decodifica la cabecera hdr.ipv4. A partir del campo hdr.ipv4.protocol, el análisis continúa de forma selectiva hacia las cabeceras de capa de transporte, permitiendo extraer campos relevantes de hdr.tcp, hdr.udp o hdr.icmp. Esta lógica permite disponer, dentro del pipeline, de los campos necesarios para aplicar políticas de filtrado L3/L4, tales como direcciones IP de origen y destino, protocolo de transporte y puertos TCP/UDP. En el bloque de tablas y acciones, se define una tabla ACL basada en coincidencia ternaria, por ejemplo acl_table, que permite aplicar reglas con máscaras sobre múltiples campos del paquete. Las llaves de búsqueda incluyen campos como hdr.ipv4.srcAddr, hdr.ipv4.dstAddr, hdr.ipv4.protocol, hdr.tcp.srcPort, hdr.tcp.dstPort, hdr.udp.srcPort y hdr.udp.dstPort, dependiendo de la validez de las cabeceras extraídas. Esta tabla se asocia con acciones de seguridad como permit() y deny(). permit() permite que el paquete continúe hacia la etapa de forwarding, mientras que deny() invoca mark_to_drop(standard_metadata), modificando los metadatos del paquete para producir un descarte silencioso (silent drop) dentro del pipeline. Además, la tabla puede incorporar contadores directos, direct_counter, asociados a cada entrada, lo que permite medir cuántos paquetes coinciden con cada política ACL.

En el bloque Ingress, primero se aplica las políticas ACL y, únicamente si el paquete no es descartado, continúa con la tabla de reenvío. Esta secuencia garantiza que el filtrado L3/L4 se ejecute antes del forwarding, evitando que tráfico no autorizado alcance etapas posteriores del pipeline. Las reglas ACL son instaladas y actualizadas desde ONOS mediante P4Runtime, utilizando mensajes WriteRequest con entradas TableEntry asociadas a la tabla ACL. Asimismo, el controlador puede consultar la efectividad de las políticas mediante operaciones ReadRequest, recuperando los valores de los contadores asociados sin intervenir en el procesamiento de paquetes legítimos. En el entorno BMv2, este diseño permite validar la semántica de reglas ternarias y el comportamiento del filtrado antes de su eventual despliegue en objetivos de hardware que implementen este tipo de coincidencia. La topología empleada para la validación se presenta en la Fig. 10.

Para las pruebas, se definió una matriz que cubre: conectividad base (ICMP permitido), bloqueo selectivo de puertos TCP/UDP (p.ej., denegar TCP/80), y casos negativos para confirmar ausencia de bypass (paquetes bloqueados no generan forwarding en la tabla). La Tabla II resume casos representativos. En ejecución, cada caso se validó mediante tráfico de aplicación y verificación de contadores por tabla, garantizando la política aplicada.

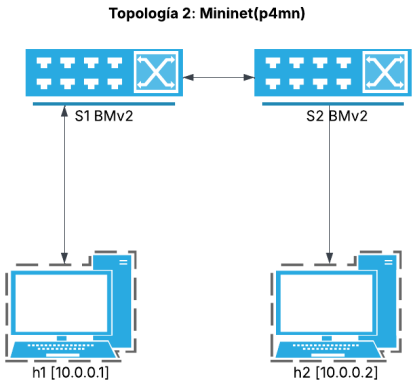

Fig. 10. Topología para validación de políticas ACL

TABLA II
MATRIZ DE PRUEBAS PARA EL MÓDULO ACL Y RESULTADOS ESPERADOS

Caso	Tráfico	Política ACL	Resultado
T1	ICMP h1->h2	permit ip proto=ICMP	Éxito (reply)
T2	TCP/80 h1->h2	deny tcp dstPort=80	Bloqueado (timeout/refused)
T3	UDP/5001 h1->h2	permit udp dstPort=5001	Éxito (recepción)
T4	ICMP h1->h2	deny ip src=10.0.0.1/32	Bloqueado

B. Aplicación básica que emplea conceptos de In-band Network Telemetry (INT)

La creciente complejidad de las redes modernas ha impulsado la necesidad de mecanismos de visibilidad más precisos y de baja latencia. In-band Network Telemetry (INT), representa un cambio de paradigma al permitir que los propios paquetes transporten información del estado de la red. La aparición de lenguajes como P4 ha facilitado la adopción de INT al permitir la programación explícita del plano de datos.

INT es una nueva abstracción que puede entenderse como que los paquetes de datos realizan consultas sobre el estado interno del switch, como la utilización del enlace y la latencia introducida en la cola de paquetes. Estas métricas pueden ser generadas por cada nodo de la red y enviadas al sistema de monitoreo en forma de un reporte. Otro método consiste en añadir las métricas en los paquetes en cada nodo que visitan en su ruta y, finalmente, eliminarlas en los nodos designados que las enviarán agregadas al sistema de monitoreo (Fig.11).

P4 proporciona los mecanismos necesarios para implementar INT mediante el uso de headers personalizados, metadatos, librerías externas y control explícito del parser y deparser.

La telemetría granular obtenida (latencia de cola y estampillas de tiempo de ingreso) proporciona la base de datos en tiempo real, permitiendo una monitorización proactiva del estado interno del dispositivo de red. Un ejemplo se presenta en la Fig. 11. A la salida del switch A se agregó headers asociados a INT. Durante el viaje del paquete por la red, cada

switch agregará su propia información (switch B), hasta llegar a un nodo (switch C) que retira todos los headers agregados por otros switches y envía la información recopilada, incluida la propia, a un sistema de monitoreo.

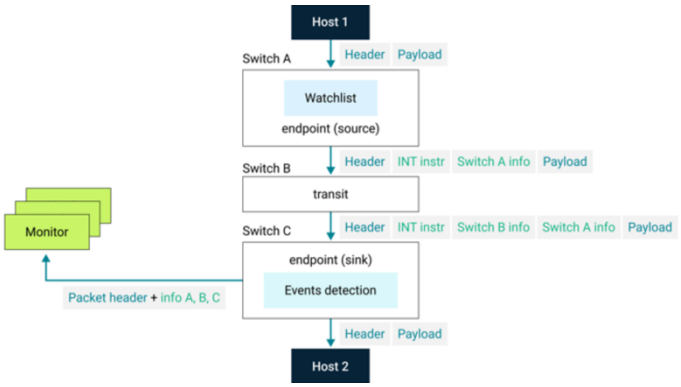

Fig. 11 Acumulación de Telemetría en cada nodo [9]

Para la implementación de INT de la aplicación básica se sigue la siguiente lógica: se define una estructura de encabezado INT, se las inserta condicionalmente en el pipeline y se controla su serialización en el deparser.

Se inicia con la definición de un encabezado personalizado; en este caso:

```
header int_header_t {
   bit<8>  int_type;
   bit<8>  length;
   bit<16> switch_id;
   bit<32> ingress_ts;  //estampilla de tiempo
}
```

Se define la pila de encabezados de interés y se incluye un encabezado para INT:

```
struct headers {
  ethernet_t ethernet;
  ipv4_t     ipv4;
  int_header_t int_header;
}
```

En la etapa de ingress de los switches se provee la información necesaria:

```
control Ingress (inout headers hdr, inout metadata meta, in
standard_metadata_t  smd) {
   apply {
      if (hdr.ipv4.isValid()) {
         hdr.int_header.setValid();
         hdr.int_header.switch_id = 1; //ejemplo
         hdr.int_header.ingress_ts =
                                 smd.ingress_global_timestamp;
      }
   }
}
```

El *deparser* controla explícitamente la inclusión del encabezado INT:

```
control Deparser(packet_out pkt, in headers hdr) {
   apply {
      pkt.emit(hdr.ethernet);
      pkt.emit(hdr.ipv4);
      pkt.emit(hdr.int_header); // incluye telemetría
   }
}
```

VII. CONCLUSIONES Y TRABAJO FUTURO

Con base a dos puntos fundamentales: a) El análisis del lenguaje P4 para conocer la lógica, semántica de las construcciones de P4, la estructura de los programas P4 y su relación con el *pipeline* del dispositivo; y, b) El *testbed* desarrollado, se consiguió recorrer el camino completo para el desarrollo de aplicaciones con P4 para un dispositivo virtual, el switch BMv2.

El desarrollo de las aplicaciones básicas con P4, y en particular las que incluyen INT (In-band Network Telemetry), permitieron aplicar lo analizado y aprendido respecto al lenguaje P4, pero, sobre todo, se pudo aplicar la idea principal de tener un plano de datos programable, junto con el plano de control también programable; es decir, los dos planos programables.

Se construyó el *testbed* con software libre, se desarrollaron aplicaciones para dispositivos programables y se aplicaron las ideas fundamentales sobre telemetría en banda, empleando el plano de datos y sin la necesidad de contactar el plano de control. Para las aplicaciones de P4 se pudo controlar el plano de datos con una interfaz de línea de comando o usando el controlador ONOS.

Todo el camino recorrido permitirá seguir trabajando en estas arquitecturas de red actuales, con los planos de control y de datos programables, desarrollando aplicaciones más complejas y buscando aplicar el conocimiento adquirido para dispositivos físicos implementados en hardware.

AGRADECIMIENTO

Los autores desean agradecer a la Escuela Politécnica Nacional por el apoyo en esta investigación.

REFERENCIAS

[1] N. McKeown, "Software Defined Networks and the maturing of the Internet," IET – The Institution of Engineering and Technology, Apr. 30, 2014. [Online]. Available: https://tv.theiet.org/?videoid=5447

[2] P4 Language Consortium, Jan, 2nd, 2026, [Online]. Available: https://p4.org/

[3] D. Comer, "The Cloud Computing Book: The Future of Computing Explained". Boca Raton, FL, USA: CRC Press, 2021.

[4] N. Foster, "Open Networking Foundation," 2020. [Online]. Available: https://opennetworking.org/wp-content/uploads/2020/12/Nate_Foster_v2.pdf.

[5] S. Gai, "Building a Future-Proof Cloud Infrastructure: A Unified Architecture for Network, Security, and Storage Services". Boston, MA, USA: Addison-Wesley, 2020.

[6] L. Zhenbin, H. Zhibo, and L. cheng, "SRv6 Network Programming: Ushering in a New Era of IP Networks". Boca Raton, FL, USA: CRC Press, 2021.

[7] A. 2.gRPC, "gRPC," 2024. [Online]. Available: https://grpc.io

[8] CodiLime, Jan, 2nd, 2026, [Online]. Available: https://codilime.com/

[9] P4-Utils, 2026. [Online]. Available: https://nsg-ethz.github.io/p4-utils/