TELEMÁTICA
REVISTA DIGITAL de las Tecnologías de la Información y las Telecomunicaciones
Revista Telemática. Vol. 16. No. 2, mayo-agosto, 2017, p.1- 11 ISSN 1729-3804

LENGUAJES DE ALTO NIVEL DE ABSTRACCIÓN PARA EL DESARROLLO DE APLICACIONES SDN

Julio Acosta Rams[1], Caridad Anías Calderón[2], Dayrene Frómeta Fonseca[3]

[1] Departamento de Telecomunicaciones y Telemática, CUJAE
[2] Departamento de Telecomunicaciones y Telemática, CUJAE
[3] Departamento de Telecomunicaciones y Telemática, CUJAE
[1]e-mail: jrams@tele.cujae.edu.cu
[2]e-mail: cacha@tesla.cujae.edu.cu
[3]e-mail: dayrene@tele.cujae.edu.cu

RESUMEN

Las Redes Definidas por Software (SDN) han sido propuestas como una alternativa para simplificar la gestión y administración de las redes. Nuevas características y servicios pueden ser añadidos dinámicamente en forma de aplicaciones, las cuales determinan el comportamiento de la red, y abren las puertas a la programabilidad e innovaciones. Nuevos *frameworks* para la programación SDN, en la forma de lenguajes de alto nivel de abstracción, han surgido como una alternativa para facilitar y acelerar el proceso de desarrollo de aplicaciones, brindándoles a los programadores las herramientas necesarias para alcanzar todo el potencial que prometen las SDN. En este artículo se presenta la arquitectura SDN, exponiendo las principales características de sus componentes y haciendo énfasis en las interfaces que brindan las SDN para el desarrollo de las aplicaciones. Luego se analizan las dificultades que impone OpenFlow para el desarrollo de aplicaciones SDN que han condicionado el desarrollo de lenguajes de alto nivel de abstracción. Finalmente se proponen un conjunto de elementos a tener en cuenta a la hora de escoger el lenguaje de programación más adecuado en función de la aplicación SDN a desarrollar.

PALABRAS CLAVE: Redes Definidas por Software (SDN), aplicaciones SDN, Interfaces de Programación de Aplicaciones (API), lenguajes de programación SDN de alto nivel de abstracción.

ABSTRACT

The Software Defined Networks (SDN) have been proposed as an alternative to simplify networks's management and administration. New features and services can be added dynamically in the form of applications, which determine the behavior of the network, and allow programmability and innovations. New SDN programming frameworks, in the form of high-level abstraction languages, have emerged as an alternative to facilitate and accelerate the application development process, giving programmers the tools they need to reach the full potential promised by SDN. This article presents the SDN architecture, exposing the main characteristics of its components and emphasizing the interfaces provided by the SDN for the development of applications. Later are analyzed the difficulties that OpenFlow imposes for the

development of SDN applications that have conditioned the development of high-level abstraction languages. Finally a set of elements are proposed to take into account when choosing the most appropriate programming language depending on the SDN application to be developed.

KEYWORDS: Software Defined Networks (SDN), SDN applications, Application Programming Interface (API), high-level abstraction languages.

INTRODUCCIÓN

Las últimas décadas se han caracterizado por un incremento alarmante del uso de Internet y las computadoras, donde el despliegue de un gran número de aplicaciones y servicios demanda a las redes subyacentes mayores velocidades, altas capacidades de procesamiento y el soporte de grandes cantidades de tráfico. Sin embargo, las arquitecturas de redes se han mantenido inalteradas, lo que ha contribuido a la complejidad en la administración y automatización de las mismas en las condiciones actuales.

La necesidad de dinamizar, hacer más robustas y permitir la experimentación con nuevas ideas y protocolos en escenarios más realísticos ha conducido al desarrollo de un nuevo paradigma, llamado SDN, el cual abre las puertas a la programabilidad y a las innovaciones de forma rápida y sencilla de las redes de computadoras. SDN se ha convertido en la nueva norma de redes, garantizando la simplicidad, dinamismo, control y reducción de costos operativos que requieren los servicios y aplicaciones actuales. Nuevas características y servicios pueden ser añadidos dinámicamente en forma de aplicaciones, lo cual aporta niveles de programabilidad nunca vistos, siendo esta una de las características más atractivas de las SDN.

Tanto la industria como la academia han prestado gran atención a las SDN, ya que permiten a los investigadores y desarrolladores de software crear y desarrollar aplicaciones para administrar servicios de red, incluyendo enrutamiento, control de acceso, *multicast*, y otras tareas de ingeniería de tráfico [1], mediante la abstracción de la infraestructura subyacente, e incluso de los complejos protocolos existentes en las redes tradicionales.

El presente artículo posee la estructura siguiente: en la Sección II se abordará lo referente a la Arquitectura SDN, haciendo énfasis en las *northbound APIs*, se analizará también la programación SDN empleando las herramientas que brinda OpenFlow y los lenguajes de programación de alto nivel de abstracción. En la sección III, se presenta los elementos a tener en cuenta para optar por un lenguaje y u otro en función de las características de la aplicación a desarrollar.

ARQUITECTURA SDN

Mientras que en las redes tradicionales la inteligencia de la red estaba acoplada al plano de reenvío en cada uno de los dispositivos, SDN desacopla toda la lógica de control embebida en los mismos, y la ubica en un nuevo dispositivo, denominado controlador SDN, quien se convierte en el componente responsable de la administración de la red, brindándole a los administradores una visibilidad completa y el control directo sobre cada dispositivo subyacente desde un único y centralizado controlador lógico [2].

La Figura 1 muestra la arquitectura general de SDN, la cual está compuesta por tres capas o planos: plano de reenvío, plano de control y plano de aplicaciones; con sus respectivas interfaces: *southbound* API y *northbound* API.

El *Plano de reenvío* es similar a las redes tradicionales, pues contempla a todo un conjunto de equipos de red, los cuales pueden adoptar la forma de software o hardware. Sin embargo, la infraestructura subyacente de red SDN está formada por simples elementos de reenvío sin mecanismos que les permitan tomar decisiones de forma autónoma. Estos dispositivos se caracterizan por realizar un conjunto de operaciones elementales, tales como en el reenvío de paquetes, la monitorización de información y recolección de estadísticas. Reciben instrucciones del controlador SDN que determinan su comportamiento, el cual está directamente vinculado a lo establecido por las aplicaciones.

En el *Plano de control* se ubica el controlador SDN, "el cerebro de la red", quien se encarga de realizar todas las funciones complejas de la misma. Este plano posee interfaces para la comunicación e interacción con el plano de reenvío y el plano de aplicaciones, denominadas *southbound* APIs y *northbound* APIs, respectivamente. Varios dispositivos de red pueden ser conectados a un único controlador y tomarse las decisiones de forma centralizada, en lugar de tener varios dispositivos de red con un conocimiento limitado de la red.

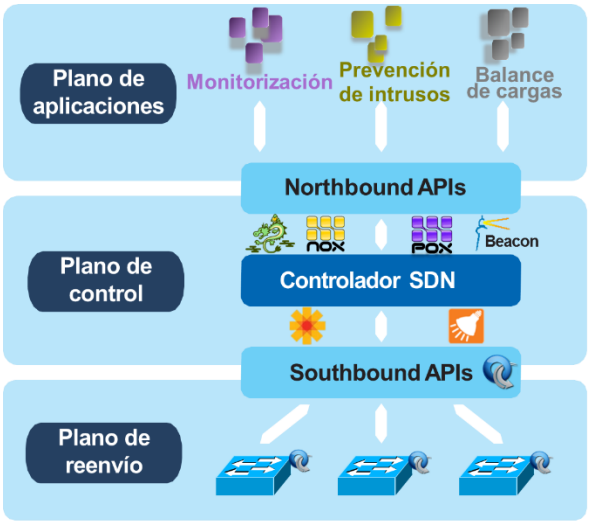

Figura 1. Arquitectura SDN

El *Plano de aplicaciones* incluye a los servicios de red, herramientas de orquestación y aplicaciones de negocio que determinan el comportamiento y características de la red, como aplicaciones de seguridad, gestión, enrutamiento, políticas de control de acceso, implementación de QoS (Calidad de Servicio, por su sigla en inglés), detección de intrusos, entre otras. Este plano recibe la información proveniente de los dispositivos de reenvío a través del controlador SDN, obteniendo una visión abstracta y global del estado de la red.

Las *Southbound APIs* permiten al controlador comunicarse y programar dinámicamente dispositivos heterogéneos de reenvío, donde OpenFlow se ha convertido en la interfaz estándar por excelencia para SDN, siendo soportada por una gran gama de controladores y otros dispositivos de red.

Las *Northbound APIs* permiten al controlador interactuar con las aplicaciones y servicios ubicados en el Plano de Aplicaciones, dándole la posibilidad a éstas de obtener información sobre la infraestructura y controlar las políticas y el comportamiento de la red de forma granular. Esta interfaz comúnmente abstrae el bajo nivel de instrucciones que implementan las interfaces *southbound* para programar los dispositivos de reenvío, constituyendo una potente herramienta para el desarrollo de aplicaciones. Los controladores actuales ofrecen una gran variedad de *northbound* APIs, como APIs ad-hoc, RESTful APIs, interfaces de programación de multinivel, entre otras. Controladores como Floodlight, Trema, NOX, Onix y OpenDaylight proponen y definen su propia northbound API, donde cada una tiene específicas definiciones en correspondencias con los intereses de los fabricantes y comunidades de desarrollo. Otro conjunto de Northbound APIs ha surgido como lenguajes de programación SDN, entre los que se destacan Nettle, NetCore, Frenetic, Procera, Pyretic, y otros.

DESARROLLO DE APLICACIONES CON OPENFLOW

OpenFlow brinda a las redes la posibilidad de una visión global desde un único punto (el controlador SDN), así como el control directo sobre el plano de datos. Sin embargo, esta API no hace sencilla la tarea a los programadores, ya que el lenguaje de configuración solo ofrece abstracciones primitivas derivadas de las capacidades básicas del hardware, conduciendo a complicadas aplicaciones propensas a errores.

Los lenguajes de configuración actuales, tanto en arquitecturas de redes tradicionales como en sistemas OpenFlow no permiten de una forma intuitiva declarar las políticas de red. Como resultado la mayoría de los prototipos de los sistemas OpenFlow carecen de interfaces configurables, por lo que requieren operadores para programar en el lenguaje de implementación de dichos sistemas.

Las implementaciones de OpenFlow en los controladores como Beacon, Floodlight, Trema, ONIX, POX, y otros basados en NOX no son eficientes y plantean muchos retos para los programadores en el proceso de desarrollo de aplicaciones. [3]

El primer reto se produce debido a que los administradores de red tienen que configurar los dispositivos de red para de forma simultánea proveer muchos servicios interrelacionados como enrutadores, balanceo de cargas, monitorización de tráfico y control de acceso. Desafortunadamente, desacoplar estas tareas e implementarlas independientemente en módulos separados es prácticamente imposible. De esta forma, OpenFlow no permite la composición de aplicaciones, ya que las reglas instaladas en cada uno de los módulos pueden solaparse e interferirse mutuamente. Para obtener el comportamiento deseado, su diseño requiere mucho cuidado del programador, y planificar la programación de la interacción de los módulos desde el principio, para evitar que las reglas instaladas se solapen, mientras se mantiene la semántica de las aplicaciones originales. Esto provoca que sea prácticamente imposible encontrar funcionalidades comunes dentro de bibliotecas reutilizables y además previene un razonamiento composicional de las aplicaciones desarrolladas.

Otra de las limitaciones de OpenFlow recae en que la interfaz que ésta brinda para la programación de aplicaciones está definida en un nivel de abstracción muy bajo, donde las reglas directamente reflejan las capacidades del hardware del switch, lo que impide a los programadores abstraer y crear grandes y complejas aplicaciones de red. Esto trae consigo que las aplicaciones se tornen innecesariamente complicadas y propensas a errores en la programación. Simples conceptos como complemento o diferencia obligan a los programadores a utilizar múltiples reglas y prioridades; y a codificar patrones que pudieran ser expresados de forma sencilla con operadores naturales de negación, diferencia y unión. OpenFlow añade un desorden innecesario a los programas, además de complicar el razonamiento de los comportamientos de los mismos [4].

Por último, el controlador sólo recibe eventos para paquetes que los *switches* no saben cómo manejar. Códigos que instalen una regla de reenvío pueden provocar ambigüedad y prevenir a otro evento diferente desencadenarse. Como resultado, escribir aplicaciones se torna un ejercicio difícil en la programación de dos-capas, donde los programadores deben simultáneamente razonar sobre los paquetes que serán procesados en los *switches* y aquellos que se procesarán en el controlador. Esto fuerza a los programadores a especificar patrones de comunicación entre el controlador y los *switches* y encargarse de asuntos difíciles como la coordinación de eventos asincrónicos. Para lograr un diseño adecuado de las aplicaciones, estas deben ser implementadas empleando dos sub-aplicaciones, una en el controlador y otra en el switch. Mientras este diseño es esencial para lograr eficiencia, la arquitectura de dos-capas hace a las aplicaciones difíciles de leer y razonar, debido a que el comportamiento de cada programa depende del otro, forzando a los programadores a especificar interacciones y eventos asincrónicos entre los programas corriendo en el controlador y los *switches* de la red.

Aunque OpenFlow hace posible implementar nuevos servicios de red, y logra la programabilidad prometida por SDN, esto no constituye una tarea sencilla para los programadores, donde prácticamente la programación se define con el nivel de abstracción soportado por el hardware subyacente, conduciendo así a complicados programas propensos a errores [4]. Todo esto hace que la programación de nuevas aplicaciones SDN con las plataformas que brindan los controladores OpenFlow sea todo un reto.

LENGUAJES DE PROGRAMACIÓN SDN

Las dificultades mencionadas de OpenFlow han conducido al desarrollo de lenguajes específicos para la programación SDN, cuyos objetivos principales son la simplificación y automatización de la configuración y gestión de las redes. Dichos lenguajes constituyen *northbound* APIs que brindan las herramientas necesarias para alcanzar todo el potencial que prometen las SDN mediante el desarrollo de nuevas aplicaciones con altos niveles de abstracción. Esto facilita a los programadores el control directo sobre la red, permitiéndoles especificar qué es lo que quieren que la red haga sin tener que preocuparse de la forma en que esto se implemente.

Los lenguajes específicos para la programación SDN han sido implementados y están disponibles como *frameworks*, los cuales al ejecutar aplicaciones desarrolladas con las primitivas y funciones que ofrecen, se convierten en los nuevos controladores SDN de la red. En la Figura 2 muestra algunos de los lenguajes de programación SDN.

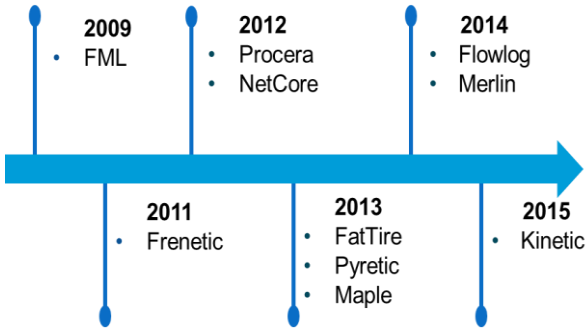

Figura 2. Evolución de los lenguajes de programación SDN.

FML (Lenguaje de Gestión basado en Flujos) [5]: es un lenguaje de alto nivel implementado en C++, que se basa en los paradigmas de programación declarativo y lógico. Este permite expresar políticas de red para diferentes tareas de gestión de configuración en redes empresariales dentro de un único y cohesivo framework. El objetivo principal de este lenguaje es reemplazar los variados mecanismos de configuración empleados tradicionalmente para aplicar políticas de red dentro de la empresa, ofreciendo una implementación adecuada de dichas políticas para grandes redes empresariales. El diseño FML fue dirigido específicamente al lenguaje de políticas para el controlador NOX.

Frenetic [3]: es un lenguaje declarativo de alto nivel para la programación SDN implementado en OCaml y en Python. A diferencia de la programación directa sobre Openflow, Frenetic brinda una colección de funciones simples para el filtraje, modificación, contabilidad y reenvío de los paquetes, así como varios operadores para combinar pequeñas funciones en una que las englobe. Este lenguaje dio origen a toda una familia de lenguajes, conocida como “Familia Frenetic”, cuyos miembros han sido desarrollados con el objetivo de proveer abstracciones de alto nivel que brinden a los programadores control directo sobre la red. Frenetic ofrece constructores modulares que facilitan el razonamiento composicional de los programas, así como rigurosas bases semánticas que documentan el significado del lenguaje. Aunque la implementación actual de Frenetic opera sobre el controlador NOX, el uso de éste es conveniente pero no imprescindible, pudiéndose adaptar fácilmente a otros controladores.

Procera [6]: es un lenguaje declarativo y funcional para la programación SDN. Procera ha sido implementado en el lenguaje Haskell, lo cual permite reutilizar varios tipos de datos y constructores de este lenguaje, como variables, listas y funciones, y emplearlos junto con los propios constructores de Procera para proveer expresiones que satisfagan los requerimientos de las aplicaciones a desarrollar. Además, los administradores pueden extender de forma sencilla el lenguaje añadiendo nuevos constructores y funciones. Procera emplea a NOX como controlador de red para tomar las decisiones y realizar las tareas relacionadas con el tráfico de la misma, como enrutamiento y descubrimiento de topología, así como de actualizar e instalar las reglas de bajo nivel en las entradas de las tablas de flujo de los *switches* de acuerdo a las aplicaciones que se ejecuten.

NetCore [7]: es definido como un lenguaje declarativo y funcional de alto nivel para expresar políticas de encaminamiento de paquetes sobre SDN. El objetivo es permitir a los programadores pensar y construir políticas de forma natural, de ahí que su sintaxis sea intuitiva, basada en operaciones teóricas familiares

a los programadores. NetCore es de código abierto, está compilado en Haskell e implementado con el controlador NOX.

FatTire [8]: es un lenguaje de programación declarativo y funcional de alto nivel, que ha sido implementado en OCaml para el desarrollo de aplicaciones SDN que puedan reaccionar ante fallos en el funcionamiento de la red, permitiendo especificar diferentes políticas de reenvío, desempeño y seguridad. FatTire ha sido implementado sobre NetCore, el cual se encarga de tomar como entrada las aplicaciones programadas en FatTire en términos de rutas para el reenvío de los paquetes y traducirlas en políticas de NetCore que, a través de protocolo OpenFlow, apliquen las configuraciones en los *switches*.

Pyretic [2]: es un lenguaje de programación de alto nivel de abstracción que permite la combinación automática de diferentes aplicaciones SDN mediante módulos independientes que puedan gestionar la red en conjunto.. Este lenguaje es de código abierto, se encuentra embebido e implementado en Python y sigue los principios del paradigma imperativo. Es un miembro la "Familia Frenetic" que se encuentra en activo desarrollo. Es un lenguaje imperativo y DSL (Lenguaje Específico de Dominio, por su sigla en inglés) embebido en Python que permite a los programadores especificar políticas de red a muy altos niveles de abstracción. Aunque Pyretic utiliza en su implementación al controlador POX para comunicarse con los switches Openflow, nuevas versiones permiten emplear también a Ryu.

Maple [9]: es un lenguaje de programación de alto nivel para las SDN basado en los paradigmas declarativo y funcional. Este lenguaje aporta simplicidad, flexibilidad y expresividad, permitiendo a los programadores desarrollar aplicaciones de forma intuitiva que alcanzan altos niveles de desempeño y escalabilidad. Ha sido desarrollado para operar con dos controladores SDN: McNettle [10], que eficientemente ejecuta los eventos OpenFlow escritos en Haskell; y OpenDaylight (ODL), cuyo framework está programado completamente en Java.

Flowlog [11]: es un lenguaje de programación de aplicaciones SDN que sigue los principios de los paradigmas lógico y declarativo, cuyo un diseño es semejante a SQL(Lenguaje de Consulta Estructurado, por su sigla en inglés) en su diseño. Provee una abstracción unificada para el plano de control y de datos, la cual simplifica el proceso de programación y, simultáneamente, permite la verificación entrecruzada de los planos de la arquitectura SDN. Puede ser empleado para descubrir errores en aplicaciones SDN, mientras también produce mínimas y eficientes reglas de red. La actual implementación de Flowlog está desarrollada en OCaml y emplea Openflow 1.0 para el manejo de paquetes, y puede implementarse tanto con el controlador Fenetic (OCaml) y utiliza a NetCore.

Merlin [12]: es un lenguaje para la gestión de redes SDN implementado en OCaml y en C, el cual se ubica encima de Frenetic y permite a los programadores expresar políticas en un lenguaje declarativo basado en predicados lógicos para identificar conjuntos de paquetes, expresiones regulares para codificar las rutas de reenvío y fórmulas aritméticas para identificar restricciones de ancho de banda. Este lenguaje pertenece a la Familia Frenetic. Merlin, sobre todo, simplifica la administración de la red, proveyendo abstracciones de alto nivel para especificar políticas que aprovisionen los recursos de la misma.

Kinetic [13]: es un lenguaje declarativo de alto nivel de abstracción para la programación de redes SDN desarrollado en Python y C++, que permite a los administradores expresar políticas dinámicas de forma concisa e intuitiva. Forma parte de la Familia Frenetic. Kinetic provee primitivas para automatizar cambios en las políticas de la red en respuesta a condiciones de red dinámicas. Además hace posible verificar si

dichos cambios satisfacen los requerimientos del administrador de la red y como debiera reaccionar para cambiar condiciones de la red. Tal verificación hace a Kinetic diferente de varios de los lenguajes previos.

PROCESO DE SELECCIÓN DEL LENGUAJE DE PROGRAMACIÓN SDN

En todo proceso de desarrollo de aplicaciones de software, es muy importante contar con orientaciones que permitan a los programadores organizarse. De acuerdo con esto, el desarrollo de aplicaciones SDN empleando lenguajes de alto nivel de abstracción, si pretende tener utilización dentro del contexto real, debe seguir un modo de proceder, donde un aspecto fundamental lo constituye la selección del lenguaje de programación a emplear.

Como parte de la investigación realizada y con el objetivo de facilitar esta selección, se realizó un resumen de las características más importantes de los lenguajes de programación SDN de alto nivel que fueron analizados en el apartado anterior, las cuales quedan resumidas en la tabla 1. Es importante especificar que se consideró como lenguajes que permiten desarrollar aplicaciones complejas a aquellos que brinden operadores composicionales eficientes y que no exigen que el programador desarrolle aplicaciones monolíticas y carentes de modularidad, propensas a interferencias entre las instrucciones declaradas. Por otro lado, por soporte de reglas se entiende la posibilidad que tenga el lenguaje de programación de especificar reglas cuyo comportamiento sea estático o dinámico, pues no todos los lenguajes brindan la posibilidad de desarrollar aplicaciones cuyas políticas cambien en función del tiempo u otros parámetros.

Tabla 1: Lenguajes de programación SDN.

Lenguaje	Lenguaje base	Controlador	Complejidad Aplicaciones	Soporte de Reglas	Bibliografía y soporte
FML	C++, Python	NOX	Simples	Estáticas	Baja
Frenetic	OCaml, Python	NOX	Complejas	Estáticas	Buena
Procera	Haskell	NOX	Complejas	Dinámicas, Estáticas	Baja
NetCore	Haskell	NOX	Complejas	Dinámicas, Estáticas	Baja
FatTire	OCaml	NetCore	Complejas	Dinámicas Estáticas	Muy Baja
Pyretic	Python	POX, Ryu	Complejas	Dinámicas, Estáticas	Buena
Maple	Haskell, Java	McNettle, OpenDaylight	Simples	Dinámicas	Muy Baja
Flowlog	OCaml	NetCore	Complejas	Dinámicas, Estáticas	Muy Baja
Merlin	C, OCaml	Frenetic	Complejas	Dinámicas, Estáticas	Media
Kinetic	C++, Python	Pyretic	Complejas	Dinámicas Estáticas	Baja

Cada lenguaje analizado tiene sus particularidades, por lo cual no se deben dejar de valorar algunos elementos claves para una apropiada elección, tales como el controlador SDN a emplear, pues un punto crucial de los lenguajes actuales de alto nivel de abstracción para la programación SDN es la dependencia del controlador, la cual, en determinadas circunstancias, obliga a los programadores a seleccionar previamente el controlador SDN antes del lenguaje de programación. Es importante destacar que cada lenguaje está estrechamente vinculado a la(s) versión(es) de OpenFlow soportada(s) por el controlador

que lo implementa, siendo ésta una de las particularidades que más limitantes impone al desarrollo de aplicaciones.

Por otro lado, resulta necesario analizar el lenguaje de programación en que se basa el lenguaje SDN, el cual constituye el aspecto más importante en cualquier proyecto de desarrollo de software. Cada lenguaje posee sus propias variables, funciones, librerías, sintaxis, métodos, semántica, Entornos de Desarrollo Integrado (*Integrated Development Enviroment*, IDE por sus siglas en inglés), plataformas, soporte, documentación y otras características que conducen a los programadores a decidirse por uno u otro. Además, generalmente los programadores se familiarizan con uno o varios lenguajes de programación y las habilidades que han obtenido influyen en la calidad y eficiencia de las aplicaciones desarrolladas.

Por otro lado, aunque la mayoría de los lenguajes que se analizan brindan funcionalidades para desarrollar un gran cúmulo de aplicaciones, algunos no son eficientes y requieren de otras herramientas que lo complementen, o simplemente no son recomendados para implementar ciertos tipos de aplicaciones. En la tabla 2 se muestra la relación entre algunas aplicaciones SDN y las posibilidades de su realización por los lenguajes considerados.

Otro aspecto importante a considerar en este proceso son las aplicaciones que se incluyen en las implementaciones de los lenguajes tratados, las cuales constituyen un punto de partida considerable para la creación de nuevas aplicaciones. En la tabla 3 se presentan las aplicaciones incluidas en el *framework* de los lenguajes SDN de alto nivel de abstracción que se han considerado en esta investigación. Es importante aclarar que no todos los lenguajes proveen aplicaciones ya desarrolladas, lo cual constituye una deficiencia.

Tabla 2: Relación entre las aplicaciones y lenguajes SDN.

Tipo	Aplicaciones	Lenguajes									
		FML	Frenetic	Procera	NetCore	FatTire	Pyretic	Maple	Flowlog	Merlin	Kinetic
Ingeniería de tráfico	Enrutamiento	X	X	X	X	X	X	X	X	X	X
	Balance de carga	----	----	----	X	O	X	X	X	X	X
	QoS	O	X	O	X	X	X	X	X	X	X
	Tolerancia a Fallos	----	----	----	----	X	----	----	----	----	----
Mediciones y monitorización	Recolección de Estadísticas	X	X	X	X	X	X	X	X	X	X
	Monitor de Red	X	X	X	X	X	X	X	X	O	O
	Inspección superficial de paquetes	X	X	X	X	O	X	X	O	X	X
Se gu ri	IDS (Sistema de Detección de Intrusiones, por su sigla en inglés)	X	X	X	X	X	X	X	X	X	X

	Inspección profunda de paquetes	X	X	O	X	O	X	X	O	X	X
	NAT (Traducción de Direcciones de Red, por su sigla en inglés)	X	X	O	X	O	X	X	X	O	X
	VLAN (Red de Área Local virtual, por su sigla en inglés)	X	X	X	X	X	X	X	X	X	X
	Firewall	O	X	O	X	X	X	X	X	X	X
	ACL (Listas de Control de Acceso, por su sigla en inglés)	X	X	X	X	O	X	X	X	X	X

(X-altamente recomendable, O-parcialmente recomendable, ---- no recomendable)

Tabla 3: Aplicaciones ya implementadas de los lenguajes SDN de alto nivel de abstracción.

Lenguaje	Aplicaciones incluidas en la implementación
Frenetic	Repetidor, *Learning Switch*, *Firewall*, Monitor
NetCore	Monitor, *Learning Switch*, NAT
Pyretic	Repetidor, *Firewall*, Descubrimiento de Topología, *Learning Switch*
Maple	*Learning Switch*, ACL
Flowlog	*Learning Switch*, NAT, VLAN, ACL
Kinetic	Balance de carga, IDS, DoS (Denegación de Servicios por su sigla en inglés)

Por último, como se muestra en la tabla 1, no todos los lenguajes poseen el mismo nivel de documentación, ni ofrecen a los programadores el soporte para lograr un completo dominio del mismo, por lo cual, la bibliografía y soporte debe tenerse en cuenta para optar por un lenguaje u otro.

Una vez identificado el lenguaje de programación que más se ajuste a las características de la aplicación y a las habilidades del programador, se recomienda proceder al estudio de su sintaxis, variables, operadores, primitivas y módulos que lo componen. Esto permite simplificar mucho el código a desarrollar e identificar si el lenguaje de programación identificado se adecúa a los requerimientos de la aplicación. Por ejemplo, algunos lenguajes proveen primitivas e instrucciones para, de forma intuitiva, modificar algunos parámetros de las cabeceras de un paquete, lo que se debe considerar si la aplicación a desarrollar requiere el uso de *firewalls* o NATs. Otros lenguajes proveen abstracciones para especificar las políticas en términos de ancho de banda o latencia, lo que los hace más eficientes en aplicaciones como balanceadores de carga.

Como se muestra en la tabla 3 no todos los lenguajes proveen aplicaciones en su implementación y, las existentes en éstos, no siempre dan solución a todos los requerimientos que puede demandar una nueva aplicación. No obstante, resulta necesario analizar las aplicaciones que los lenguajes incluyen en su framework, pues brindan funciones y métodos que pudiesen reutilizarse para el desarrollo de otras nuevas.

Si el lenguaje identificado satisface los requerimientos de la aplicación a desarrollar, se puede proceder a la programación de la aplicación.

CONCLUSIONES

Una de las oportunidades más interesantes que ofrecen las SDN es el desarrollo de aplicaciones, las cuales permiten introducir en las redes, a través de *software,* nuevas funcionalidades, herramientas y servicios. A pesar de esto, programar las SDN empleando las herramientas que brinda OpenFlow no es una tarea fácil, por lo cual han surgido varios lenguajes de alto nivel de abstracción que hacen esta tarea más factible e intuitiva.

Debido a las deficiencias de la programación con OpenFlow, los autores de esta investigación han analizado los principales lenguajes de programación de alto nivel de abstracción para la programación de aplicaciones SDN, y fue posible identificar un conjunto de elementos a tener en cuenta que puedan conducir a optar por un lenguaje u otro cuando los programadores se enfrenten a la tarea de desarrollar nuevas aplicaciones. Estos elementos se enfocan principalmente en las características de la aplicación, en las habilidades de los programadores y en las posibilidades de su implementación.

REFERENCIAS

1. Jehn-Ruey Jiang, H.-W.H., Ji-Hau Liao, and Szu-Yuan Chen, *Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking.* 2014.
2. Christopher Monsanto, J.R., Nate Foster, Jennifer Rexford, David Walker, *Composing Software-Defined Networks.* 2014.
3. Nate Foster, R.H., Michael J. Freedman, Jennifer Rexford, Christopher Monsanto, Alec Story and David Walker *Frenetic: A Network Programming Language*. 2012.
4. Nate Foster, R.H., Michael J Freedman, Jennifer Rexford and David Walker *Frenetic: A High-Level Language for OpenFlow Networks*. 2012.
5. Timothy L. Hinrichs, N.S.G., Martin Casado, John C. Mitchell, Scott Shenker, *Practical Declarative Network Management.* 2009.
6. Andreas Voellmy, H.K., Nick Feamster *Procera: A Language for High-Level Reactive Network Control*. 2012. 6.
7. Christopher Monsanto, N.F., Rob Harrison, David Walker *A Compiler and Run-time System for Network Programming Languages*. 2012. 14.
8. Mark Reitblatt, M.C., Arjun Guha, Nate Foster, *FatTire: Declarative Fault Tolerance for Software-Defined Networks.* 2013.
9. Andreas Voellmy, J.W., Y. Richard Yang, Bryan Ford, Paul Hudak *Maple: Simplifying SDN Programming Using Algorithmic Policies*. 2013.
10. Andreas Voellmy, J.W. *Scalable Software Defined Network Controllers*. 2012.
11. Tim Nelson, A.D.F., Michael J.G. Scheer, Shriram Krishnamurthi, *Tierless Programming and Reasoning for Software-Defined Networks.* 2014.
12. Robert Soulé, S.B., Parisa Jalili Marandi, Fernando Pedone, Robert Kleinberg, Emin Gün Sirer, Nate Foster, *Merlin: A Language for Provisioning Network Resources.* 2014.
13. Hyojoon Kim, J.R., Arpit Gupta, Muhammad Shahbaz, Nick Feamster, Russ Clark, *Kinetic: Verifiable Dynamic Network Control.* 2015.