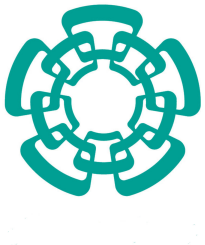

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Cinvestav Tamaulipas

Gestión de reglas en ambientes de Redes Definidas por Software para aprovisionamiento y control de servicios entre centros de datos

Tesis que presenta:

Francisco Javier Aguirre Gracia

Para obtener el grado de:

**Maestro en Ciencias
en Ingeniería y Tecnologías
Computacionales**

Director de la Tesis:
Dr. Javier Rubio Loyola

Cd. Victoria, Tamaulipas, México. Febrero, 2020

La tesis presentada por Francisco Javier Aguirre Gracia fue aprobada por:

_____

Dr. José Luis González Compeán

Dr. Hiram Galeana Zapién

Dr. Javier Rubio Loyola, Director

Cd. Victoria, Tamaulipas, México., 28 de Febrero de 2020

A mi familia

Agradecimientos

- A mis padres Amalia y Javier, que a lo largo de mi vida me han apoyado incondicionalmente sin importar el tamaño de los obstáculos para ser un hombre de bien.
- A mi esposa Cinthya, por la comprensión y apoyo total durante mis estudios. Este logro es de ambos.
- A mi hijo Javier, por sacarme sonrisas en momentos difíciles y crecer junto con este trabajo, al punto de cumplir sus 2 años de vida el día de la presentación de este trabajo. ¡Felicidades!
- A mi asesor de estudios, el Dr. Javier Rubio Loyola, por el gran apoyo, paciencia, comprensión y consejos que me dio, y por brindarme la oportunidad de trabajar con él en proyectos que me permitieron crecer profesionalmente, pero sobre todo me permitieron aprender.
- A mis revisores de tesis, por dar seguimiento a mi trabajo y los consejos y conocimiento compartido.
- A los profesores de la Unidad, que compartieron su conocimiento y experiencias, tanto dentro como fuera de las aulas.
- A mis compañeros de generación, juntos logramos formar un grupo unido y de apoyo en los momentos difíciles.
- A las maravillosas personas que pude conocer en los proyectos de RETRACT y NECOS, en especial a Celso por su hospitalidad durante mi estancia en un país desconocido y brindarme su amistad.
- Al personal administrativo del CINVESTAV - Unidad Tamaulipas por las facilidades y apoyo otorgado durante mi estancia en la institución.
- A CONACYT, por el apoyo económico que me permitió concentrarme en mis estudios.

Índice General

Índice de figuras

Índice de tablas

Índice de algoritmos

Resumen

Gestión de reglas en ambientes de Redes Definidas por Software para aprovisionamiento y control de servicios entre centros de datos

por

Francisco Javier Aguirre Gracia
Unidad Cinvestav Tamaulipas
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2020
Dr. Javier Rubio Loyola, Director

El concepto *Edge Computing* hace referencia a un modelo de cómputo en la nube para favorecer los servicios sensibles a latencia, para lo cual los proveedores de servicios requieren alojar sus servicios en diferentes zonas geográficas de acuerdo a la demanda temporal como, por ejemplo, videos en tendencia para los proveedores de servicios multimedia, componentes de monitoreo de condiciones marítimas para localización de puntos de pesca, o aprovechar las redes vehiculares como soporte para otras redes en situaciones de emergencia. Dado que los servicios necesitan ciertos requerimientos de red, como latencia o ancho de banda, es difícil para un proveedor de servicios asegurar estos requerimientos en los enlaces de la red de transporte que se encuentran entre sus centros de datos y los recursos virtuales cercanos al usuario final, así como el alojamiento mismo de estos recursos virtuales. Si bien el concepto de *Network Slicing* hace posible generar varias redes lógicas aisladas con sus propias características (conocidas como *slices*) sobre una red física, esta segmentación se hace en alguno o ambos extremos del slice, sin involucrar la red de transporte que influye significativamente en la calidad de servicio percibida por el consumidor final, por lo que son necesarios mecanismos más sofisticados para reconfigurar la red subyacente para slices que requieren elasticidad y que puedan suspenderse y reactivarse de acuerdo a la demanda geográfica de los proveedores de servicios, además de poder desplegar y gestionar recursos virtuales de diferentes infraestructuras como entidades aisladas.

En este trabajo de tesis se presenta una solución de Proveedor de Slices multi-cloud para el

control de calidad de servicio entre centros de datos, el cual ejecuta un control de admisión mediante subscripción, mecanismos de colocación de slices mediante invocación y orquestación de los recursos virtuales durante el ciclo de vida de los slices, de manera que se satisfagan los requerimientos de calidad de servicio (e.g. ancho de banda, latencia), aprovechando las redes físicas de los proveedores de infraestructura y el concepto Network Slicing con el paradigma de *Redes Definidas por Software* para la gestión dinámica de reglas de flujos que permitan controlar y segmentar el tráfico en las redes. La experimentación se realizó con servicios que demandan diferentes requerimientos y los resultados comprueban que el Proveedor de Slices es capaz de controlar la calidad de servicio de los slices mediante la gestión automatizada de reglas de flujos en el plano de datos desencadenada al detectar subutilización o sobreutilización de los recursos de acuerdo a sus requerimientos.

Abstract

Rules management in environments of Software Defined Networking for provisioning and control of services between data centers

by

Francisco Javier Aguirre Gracia
Cinvestav Tamaulipas
Center for Research and Advanced Studies of the National Polytechnic Institute, 2020
Dr. Javier Rubio Loyola, Advisor

The *Edge Computing* concept refers to a cloud computing model to favor latency-sensitive services, for which service providers need to host their services in different geographical areas according to temporary demand, such as trending videos for multimedia service providers, monitoring components of maritime conditions for location of fishing points, or taking advantage of vehicle networks as support for other networks in emergency situations. Since services require certain network requirements, such as latency or bandwidth, it is difficult for a service provider to secure these requirements in the links of the transport network between its data centers and virtual resources close to the end user, as well as hosting these virtual resources. While the Network Slicing concept makes it possible to generate several isolated logical networks with their own characteristics (known as slices) over a physical network, this segmentation is done at one or both ends of the slice, without involving the transport network that significantly influences the quality of service perceived by the final consumer, so more sophisticated mechanisms are necessary to reconfigure the underlying network for slices that require elasticity and that can be suspended and reactivated according to the geographical demand of the service providers, in addition to being able to deploy and manage virtual resources of different infrastructures as isolated entities.

In this thesis, a multi-cloud Slice Provider solution is presented to control quality of service between data centers, which performs admission control through subscription, slices placement mechanisms through an invocation process and orchestration of virtual resources during the slices life

cycle, so that the quality of service requirements (e.g. bandwidth, latency) are met, taking advantage of the physical networks of the infrastructure providers and the Network Slicing concept with the *Software Defined Networking* paradigm for the dynamic management of flow rules that allow to control and segment traffic in networks. The experimentation was carried out with services that demand different requirements and the results prove that the Slice Provider is able to control the quality of service of the slices through the automated management of flow rules in the data plane triggered by detecting underutilization or overuse of resources according to their requirements.

Nomenclatura

AGC	Algoritmo Genético Canónico
ARP	Address Resolution Protocol
CDC	Core Data Center
DiffServ	Differentiated Services
EC	Edge Computing
EDC	Edge Data Center
GRE	Generic Routing Encapsulation
HTB	Hierarchical Token Bucket
IaaS	Infrastructure as a Service
InP	Infrastructure Provider
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
LSDC	Lightweight Slice Defined Cloud
MAC	Media Access Control
MPLS	Multi-Protocol Label Switching
NCL	Network Control Layer
NECOS	Novel Enablers for Cloud Slicing
NF	Network Function
NFV	Network Functions Virtualization
PaaS	Platform as a Service
QoS	Quality of Service
REST	Representational State Transfer
RSU	Road Side Unit
SaaS	Software as a Service
SDN	Software Defined Networking
SLA	Service Level Agreement
SlaaS	Slice as a Service
SP	Service Provider
TCP	Transmission Control Protocol
TEQUILA	Traffic Engineering for Quality of Service in the Internet at Large scale
VANET	Vehicular Ad-hoc Network
VLAN	Virtual Local Area Network
YAML	YAML Ain't Markup Language

1

Introducción

En este capítulo se presenta la importancia de abordar el tema de aprovisionamiento de slices con calidad de servicio mediante una revisión de los antecedentes, la motivación que existe por abordar este problema y la definición de la hipótesis para solucionarlo. También se describen los objetivos de este trabajo y la metodología con las actividades realizadas.

1.1 Antecedentes

La cantidad y diversidad de servicios que se consumen a través de Internet ha crecido de manera importante en los últimos años, superando su crecimiento por mucho a otros sectores como

manufactura o información, creciendo el doble entre 2014 y 2018 sólo en los Estados Unidos [5], lo cuál hace que la cantidad de empresas que los proveen tambien aumente. Por esta razón, los proveedores de servicios (SPs, por sus siglas en inglés) buscan sacar ventaja de sus competidores al ofrecer calidad en ellos, ya sea mediante la provisión rápida de las solicitudes de servicio, realizar seguimiento del usuario y minería de datos para conocer al usuario y ofrecer contenido relevante para mejorar la experiencia de los usuarios [6]. Para cumplir con estos requerimientos los SPs necesitarían ubicar sus recursos de cómputo cerca del usuario final, lo cual resultaría en altos costos operativos y servicios aislados con limitaciones para ofrecer todas las ventajas mencionadas anteriormente. Para abordar esta problemática, un enfoque que se ha propuesto recientemente es el *Edge Computing* (EC) [7], el cual consiste en llevar los servicios más cerca del usuario final, manteniéndose conectado a una fuente de procesamiento y/o almacenamiento centralizada, de esta manera el proveedor de servicios mantiene contenido relevante o herramientas de procesamiento cerca del usuario y favorece los servicios sensibles a latencia, enriqueciendo su servicio con sus grandes volúmenes de datos o poder de cómputo en sus centros de datos.

No obstante, bajo el enfoque del EC hay tres niveles que afectan la calidad del servicio (QoS, por sus siglas en inglés) percibida por el usuario final: el nivel de servicio contratado al SP, el nivel de servicio contratado al proveedor de servicio de Internet(ISP, por sus siglas en inglés) y el nivel intermedio de la red de transporte entre los dos anteriores, el cuál es variable al ser puntos de interconexión de diferentes proveedores de infraestructura (InPs, por sus siglas en inglés). En sus inicios, los SPs controlaban la QoS en la infraestructura de sus centros de datos, por lo que los otros dos niveles a veces influían de manera negativa la calidad percibida por el usuario. Recientemente se incluyó a los ISPs como parte de esta cadena para permitir un control de la QoS mediante acuerdos entre el SP y el ISP del usuario.

Para agilizar la toma de decisiones en la gestión de redes en diversos ámbitos, se ha analizado

la forma en cómo aprovechar técnicas de inteligencia computacional como aprendizaje máquina o inteligencia artificial [8, 9] para resolver problemas, mediante el aprovechamiento de la información generada por la misma red. En el trabajo [10], por ejemplo, se propone un algoritmo genético para aprovisionamiento de recursos en un ambiente de *Network Slicing*, de forma que la configuración de slices en una red permitan maximizar la ganancia del operador de red.

Este trabajo de tesis se desarrolla en el contexto del Cloud Slicing [11], un concepto que toma en consideración los dominios de infraestructura de centros de datos tanto centrales como al borde y la red de transporte para crear, proveer y gestionar conjuntos de recursos virtuales vistos como infraestructuras aisladas (conocidas como *slices*). En este sentido, se propone una arquitectura de un Proveedor de Slices que controla la QoS de la red de transporte, orquestrando infraestructura de diferentes InPs para crear enlaces lógicos entre recursos virtuales de centros de datos mediante la gestión de reglas de flujos. Esta importante problemática ha recibido poca atención de la comunidad investigadora, dado que la mayoría de trabajos se centra en la gestión de recursos virtuales en alguno o ambos extremos de los slices.

1.2 Motivación

El Cloud Slicing es un modelo reciente de cómputo en la nube que tiene como objetivo proveer segmentos de la nube (conocidos como *slices*) para desplegar servicios entre centros de datos. En el Cloud Slicing, la nube que se segmenta está compuesta por múltiples centros de datos (nubes pequeñas) y redes de transporte mediante las cuáles se comunican los centros de datos, por lo que un slice es multi-cloud al involucrar recursos virtuales de varias nubes y enlaces lógicos, esto se ilustra en la Figura 1.1.

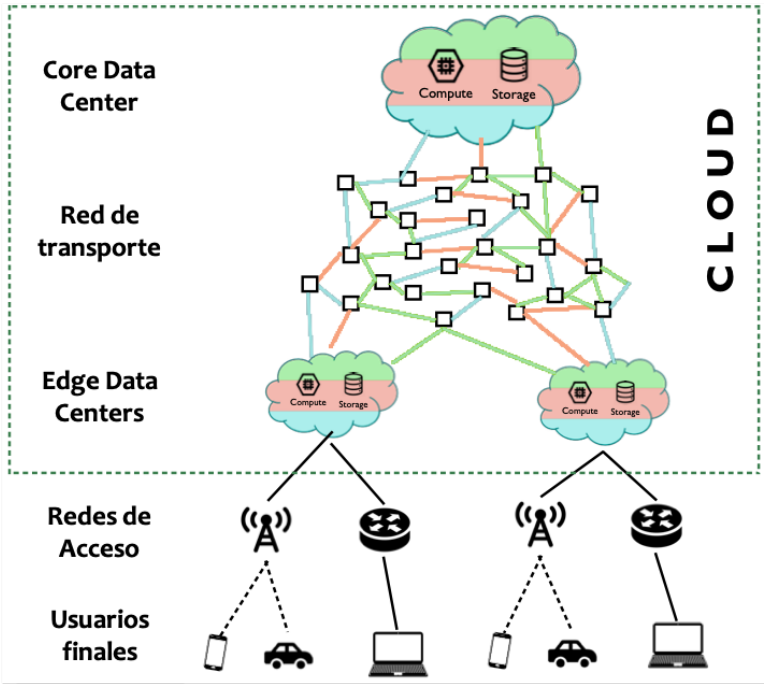

Figura 1.1: Representación del Cloud Slicing, donde la nube a segmentar se compone de centros de datos y redes de transporte.

Dada la naturaleza del Cloud Slicing que explota la virtualización de los recursos de extremo a extremo en ambientes multi-cloud resulta difícil asegurar el cumplimiento de las políticas de QoS, ya que es necesario llevar a cabo varias acciones de gestión de reglas orientadas a eficientar la utilización de recursos de redes compartidas y proveerlos como múltiples redes que funcionan de forma aislada. Sin embargo, lo anterior representa una dificultad muy elevada dado que se debe orquestrar infraestructura de diferentes dominios para múltiples solicitudes y satisfacer la QoS de cada slice sin que el servicio se vea interrumpido, pero la mayoría del trabajo relacionado se enfoca en redes de acceso o redes en centros de datos.

Definir una red aislada y gestionarla incluyendo los recursos de la red de transporte es un trabajo que no se ha abordado ya que se ha mantenido aún la atención en el Network Slicing, solo el trabajo

en NECOS [11] se define teóricamente, pero está limitado en su implementación en [12] a una configuración inicial de un slice completo y una reconfiguración sólo en los centros de datos para mantener un desempeño que mejora de forma reactiva.

1.3 Planteamiento del problema

Aunado a que el trabajo en Cloud Slicing aún es poco, no se ha demostrado la gestión de slices multi-cloud en su totalidad. Aunque en NECOS se presenta una arquitectura funcional [11, 12] que permite creación y gestión de slices en cuanto al aumento o decremento de recursos virtuales en los centros de datos, no ha abordado el problema de la gestión de la QoS en los enlaces lógicos sobre la red de transporte como una sola red aislada y sugiere hasta el momento los conceptos de Network Slicing en el control de QoS en los centros de datos en los extremos, volviendo al hecho de gestionar dos redes de forma aislada y depender de la red intermedia bajo la premisa de *Best Effort*. En algunos trabajos se han estudiado las posibilidades de provisión de slices bajo demanda a SPs mediante el acceso a la infraestructura de diferentes InPs a través de un proveedor de slices que localiza lugares idóneos para alojar los recursos virtuales [13, 14], crearlos con los requerimientos establecidos por el SP, conectarlos, dar acceso al SP y gestionar el slice durante su ejecución, de manera que el slice pueda crecer o disminuir en recursos para satisfacer el acuerdo de nivel de servicio (SLA, por sus siglas en inglés).

Sin embargo, las arquitecturas presentadas no incluyen aún mecanismos de control de QoS que involucren a los recursos del slice de red de transporte como recursos a gestionar. Dado que la demanda de los servicios que se despliegan en los slices cambia constantemente y requieren más o menos recursos durante su ciclo de vida, además de reubicación para poder alojar nuevos slices

invocados en la misma red subyacente, es necesaria una arquitectura que gestione infraestructura multi-cloud con mecanismos que realizen dichos cambios de manera dinámica y hagan un seguimiento de los requerimientos de cada slice para garantizar que sus SLAs no se violen.

1.4 Hipótesis

Con base en la problemática descrita anteriormente se plantea la siguiente hipótesis:

Mediante una arquitectura basada en miroservicios y el monitoreo de la utilización de recursos es posible definir reglas de flujos de manera dinámica para controlar la calidad de servicio de slices en ambientes multi-cloud y cumplir con el Acuerdo de Nivel de Servicio de cada slice.

1.5 Objetivos

Para el cumplimiento de la hipótesis se plantean los siguientes objetivos:

Objetivo general

Proponer una arquitectura de aprovisionamiento de slices autoconfigurable que gestione las reglas de flujos para controlar la calidad de servicio de los slices en ambientes multi-cloud.

Objetivos particulares

- Definir una arquitectura de gestión de reglas de flujos en ambientes de redes definidas por software que sea auto-configurable de acuerdo a la utilización de recursos.
- Establecer un mecanismo de gestión de slices para satisfacer la calidad de servicio de todos los slices en ejecución en un ecosistema global de slices.
- Definir un diseño de arquitectura modular para permitir la experimentación con diversos mecanismos de inteligencia computacional para asignación y optimización de recursos en ambientes multi-cloud

1.6 Metodología de desarrollo

A continuación se enlistan las etapas que fueron necesarias para implementar y operar el proveedor de slices propuesto, así como las actividades realizadas en cada una de las etapas.

Etapa 1. Diseño de la arquitectura del proveedor de slices. En esta etapa se realizó un estudio del estado actual del aprovisionamiento de recursos en la nube de forma automatizada y bajo demanda, además de implementar un prototipo funcional de un proveedor de slices multi-cloud del estado del arte más reciente.

Actividades:

- Analizar los conceptos network slicing y cloud slicing en el estado del arte.
- Analizar los resultados de los trabajos más representativos del control de QoS en redes MPLS, DiffServ y SDN.

- Desarrollar el prototipo de proveedor de slices de la solución NECOS y análisis de las áreas de oportunidad en la red.
- Definir los escenarios de operación del proveedor de slices.
- Diseñar el proveedor de slices basado en las etapas de subscripción e invocación basado en microservicios.

Etapa 2. Desarrollar e integrar los módulos y aplicaciones del proveedor de slices. En esta etapa se programó la plataforma para el proveedor de slices, la cuál consta de los módulos de subscripción e invocación y la aplicación de seguimineto de topología y monitoreo.

Actividades:

- Programar la aplicación REST de seguimiento de topología y monitoreo para mantener información de los dispositivos en el plano de datos y sus métricas de utilización.
- Programar el módulo de subscripción utilizando un algoritmo genético para maximizar la cantidad de slices en ejecución mientras se respetan sus requerimientos.
- Programar el módulo de invocación para buscar una configuración en la red que permita cumplir los requerimientos del slice invocado y de los slices en ejecución.
- Adaptar el módulo de invocación para incluir estrategias de búsqueda con clasificadores.
- Desarrollar de un orquestador de slices para el módulo de invocación que hace seguimiento a la QoS de cada slice y reconfigura la red en situaciones de degradación.

Etapa 3. Validación del proveedor de slices propuesto mediante pruebas de

subscripción, invocación y control de QoS. En esta etapa se configuró un banco de pruebas para validar el proveedor de slices propuesto para el control de QoS de slices en ambientes multi-cloud.

Actividades:

- Diseñar el ambiente de pruebas de la infraestructura de red y centros de datos.
- Configurar de una red definida por software mediante switches virtuales y túneles GRE (por sus siglas en inglés) para emulación de enlaces sobre enlaces físicos.
- Configurar de ambientes de centros de datos basados en contenedores para provisión de servicios.
- Ejecutar repetidamente las pruebas para obtener resultados sobre la eficiencia en los módulos de subscripción e invocación y el correcto control de la QoS de slices mediante elasticidad.

1.7 Estructura del documento de tesis

Este documento está compuesto por seis capítulos. El Capítulo 2 presenta el marco teórico, donde se explican los conceptos clave del tema de investigación. En el Capítulo 3 se muestran los trabajos relacionados más representativos revisados en la literatura. El Capítulo 4 describe la solución propuesta para lograr los objetivos. En el Capítulo 5 se describen las pruebas realizadas y los escenarios de validación, así como los resultados obtenidos. Finalmente, el Capítulo 6 concluye este trabajo de investigación destacando las contribuciones y exponiendo los retos presentados a lo largo del desarrollo de este proyecto.

2

Marco Teórico

En este capítulo se describen los conceptos teóricos para comprender el contexto en el que se desarrolla este trabajo de investigación. En concreto, se describe los siguientes conceptos; cómputo en la nube, la necesidad del Cloud Slicing y la gestión de la QoS, así como tecnologías como la virtualización y las Redes Definidas por Software (SDN, por sus siglas en inglés).

2.1 Redes Definidas por Software

El concepto SDN nace de la necesidad de centralizar (lógicamente) la toma de decisiones y automatización del funcionamiento de las redes, ya que en las redes IP convencionales como

Best Effort, *Multi-Protocol Label Switching* (MPLS) y servicios diferenciados (DiffServ, del inglés Differentiated Services), es difícil convertir políticas (conjunto de reglas que dictan cómo se deben repartir los recursos entre los clientes) en comandos de bajo nivel para dispositivos de red (e.g., switches y routers) y hacer que respondan ante cambios, fallas y necesidades emergentes de los centros de datos [15]. Una forma de afrontar estas dificultades es abstraer el funcionamiento de la red y tratar cada función de forma separada, lo cual es posible gracias al paradigma de SDN. Como se ilustra en la Figura 2.1, SDN separa la lógica de la red, manteniendo el plano de datos en los enrutadores pero centralizando el plano de control [16], permitiendo de esta forma la capacidad de programar las redes. Al utilizar SDN junto con las funciones de virtualización de red (NFV, por sus siglas en inglés) es posible aumentar las posibilidades de gestión de la red, ya que es posible desplegar redes virtuales sobre redes físicas.

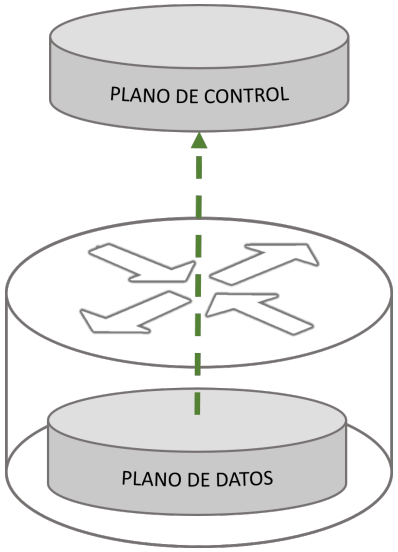

Figura 2.1: Separación de los planos de control y datos.

En SDN se conocen tres planos: 1) en el plano de datos fluye toda la información que transmiten y reciben los usuarios, la cuál se reenvía entre los dispositivos de la red, 2) en el plano de control se utilizan equipos inteligentes centralizados que llenan las tablas de reenvío de los dispositivos del

plano de datos de acuerdo al conocimiento global de la red, y 3) el plano de gestión, el cuál está conformado por las aplicaciones que monitorean y configuran los dispositivos del plano de control. Visto de otra forma, las políticas de red se definen en el plano de gestión, el plano de control crea las reglas de cómo deberán reenviarse los datos a través de la infraestructura de la SDN y el plano de datos ejecuta las acciones recibidas del plano de control [17].

Para la comunicación entre el plano de control y el plano de datos se utiliza la interfaz *Southbound*, que comúnmente utiliza el protocolo *OpenFlow*. Para la comunicación entre el plano de control y el plano de gestión se utiliza la interfaz *Northbound* [18], por la cual los controladores obtienen información del estado de la red y envían reglas a las tablas de reenvío de los switches para controlar el tráfico, permitiendo la programación de las redes. Para la comunicación entre los controladores del plano de control se utilizan las interfaces *Eastbound* y *Westbound*, mediante las cuales los controladores intercambian información de los dispositivos que controlan, pudiendo conocer entre todos el estado y la topología de la red SDN. La Figura 2.2 muestra el esquema de la arquitectura SDN.

Como se mencionó anteriormente, *OpenFlow* es el protocolo más utilizado en SDN y cuenta con una especificación amplia [1], por lo que es utilizado en este trabajo de tesis. *OpenFlow* es una tecnología habilitada en switches y controladores, y es necesaria una negociación de versión del protocolo *OpenFlow* entre ambos para establecer comunicación mediante un canal *OpenFlow* e intercambiar información.

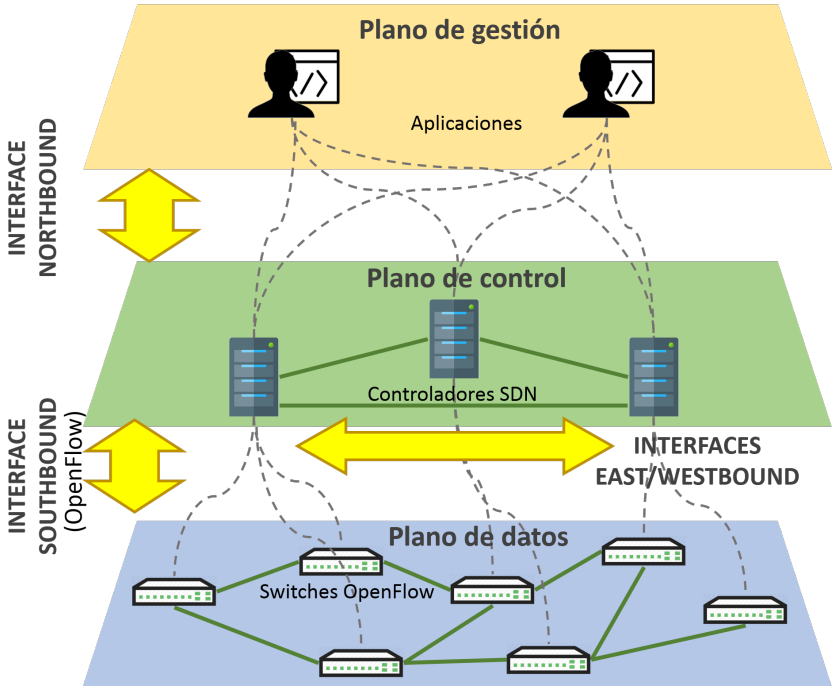

Figura 2.2: Arquitectura de la tecnología SDN.

Los switches *OpenFlow* tienen tablas de reenvío, las cuales son llenadas con reglas de flujos creadas por el controlador con base a cualquier criterio que se requiera, ya que son programados para reaccionar, por ejemplo, a congestión en enlaces, para separar flujos de diferentes usuarios, etc. Cada regla de flujo realiza una acción, como eliminar el paquete, reenviarlo a un switch en específico, modificarlo, o enviarlo a otra tabla o al controlador para definir qué hacer con él en caso de que el switch no tenga una regla para ello, por lo que el switch puede comportarse como router, switch, firewall, balanceador de carga, etc. Un flujo es un conjunto de paquetes entre un origen y un destino, y los paquetes de un flujo reciben el mismo tratamiento de acuerdo a la tabla de reenvío que indica las acciones a realizar para cada flujo. La Figura 2.3 muestra los campos escenciales para generar una regla de flujo. Para identificar flujos se utilizan campos coincidentes, como el puerto por el que entran los paquetes, la dirección MAC o IP de origen o destino de los paquetes. Los campos de instrucciones dictan lo que se debe hacer con los paquetes del flujo coincidente, como por ejemplo,

enviar todos los paquetes de ese flujo por un puerto, enviarlos al controlador para hacer algún análisis o modificación, destruirlo o enviarlo a otra tabla para que se darle otro tratamiento. El campo de contadores se utiliza simplemente para contabilizar las veces que la regla se ha implementado.

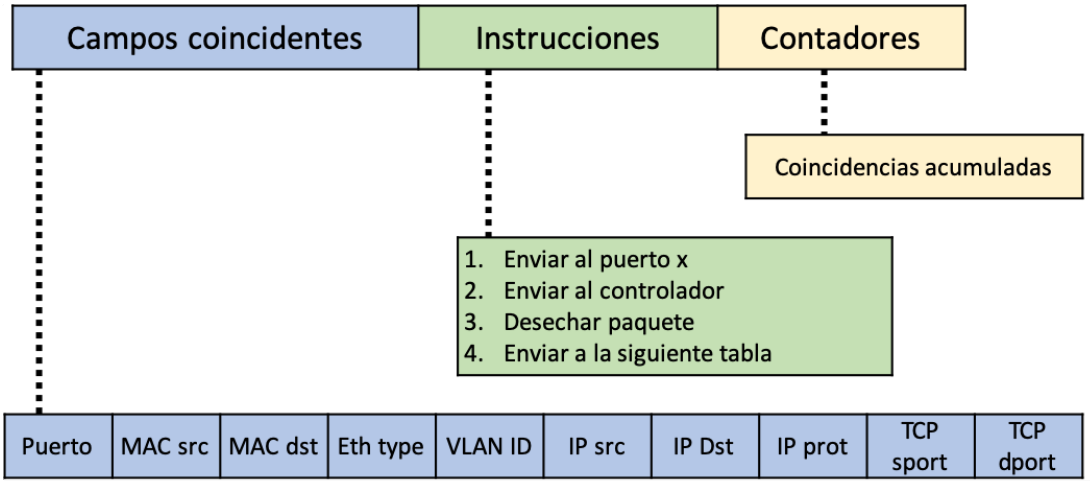

Figura 2.3: Composición de una regla de flujo OpenFlow [1]

SDN provee la capacidad de gestionar la QoS a nivel global de red, siendo la gran ventaja ante arquitecturas como DiffServ y MPLS. En DiffServ, las funcionalidades para proveer QoS están delegadas en los elementos enrutadores del borde y los elementos enrutadores centrales, los cuales se encuentran distribuidos en toda la infraestructura de red. De igual manera, las tareas de etiquetado y enrutamiento en la tecnología MPLS se llevan a cabo en cada uno de los elementos de la infraestructura de red. Esta distribución de las tareas de control demanda altos niveles de coordinación y manejo de consistencia y se caracterizan por su rigidez en las tareas de recofiguración.

2.2 Gestión de la calidad de servicio en la red

La calidad de servicio (QoS) es una métrica que indica qué tan bien se provee un servicio a un consumidor [19] (i.e. ancho de banda, retraso, pérdida, jitter). En el área de redes, el término QoS se refiere al uso de técnicas para que las aplicaciones que funcionan sobre la red puedan cumplir con sus requerimientos de desempeño, aprovechando eficientemente los recursos para evitar expandir o saturar la red [20].

La gestión de la QoS tiene como objetivo modificar el comportamiento de la red para cumplir con el acuerdo de nivel de servicio (SLA, por sus siglas en inglés), el cual es un contrato del proveedor del servicio con el consumidor y especifíca los niveles de QoS que el consumidor debe recibir. Garantizar valores específicos de QoS es una tarea difícil y pueden no cumplirse al momento de la petición de servicio, por lo que las técnicas de gestión de QoS en redes controlan rangos de valores aceptables dentro de los niveles de QoS.

Proveer QoS no sólo es limitar el ancho de banda de acuerdo a lo establecido en el SLA, sino también diferenciar el orden en que se tratan los paquetes en la red basandose en prioridades [21] (encolamiento), en las rutas en que se pueden enviar los paquetes para evitar congestión o para cumplir con criterios de retardo (enrutamiento) o ser una gestión basada en políticas.

Utilizar el tipo de provisión de QoS basado en encolamiento resulta costoso. En el trabajo reportado en [22], se probaron diferentes técnicas de encolamiento y diferentes tipos de switches OpenFlow y los resultados muestran un sobreuso de los recursos al tener que duplicar paquetes. En los trabajos [23, 24, 25] se ha explorado esta forma de gestionar la QoS en redes SDN.

Mediante técnicas de gestión de QoS basada en enrutamiento las redes SDN se configuran dinámicamente para cumplir con uno de dos objetivos: evitar el congestionamiento [26, 27, 28, 29, 30, 31] o cumplir con requerimientos de retardo para las aplicaciones [32].

La gestión basada en políticas ha sido muy utilizada para gestionar redes DiffServ o MPLS para proveer QoS con resultados aceptables, sin embargo hay poca exploración en la literatura de la utilización de ésta técnica en redes definidas por software, donde la mayoría de trabajos se limita a utilizar los métodos de encolamiento o enrutamiento. Este tipo de gestión se presenta como un habilitador de la configuración dinámica de la red de forma que se cumplan las políticas que se extraen del SLA [33]. El proceso de transformar políticas de alto nivel del SLA (i.e. otorgar calidad premium al servicio de VoIP) a políticas de bajo nivel que serán ejecutadas en los dispositivos de la red (i.e. ancho de banda >20 Mbps) se conoce como *refinamiento de políticas* y es uno de los retos más difíciles de la gestión basada en políticas.

En el trabajo de tesis que se presenta, al tratarse de un trabajo en el contexto multi-cloud en donde el cumplimiento de la QoS pasa por el aseguramiento de cumplir con los SLAs, la gestión basada en políticas y la gestión basada en enrutamiento resultan ser las opciones de gestión más adecuadas para la problemática a resolver.

2.3 Network Slicing

El concepto *Network Slicing* permite que una red física pueda ser segmentada en varias redes lógicas independientes, de manera que los SPs puedan gestionar de forma individual sus servicios sin que el consumo de recursos de un slice afecte el desempeño del resto aún y cuando se comparte

la misma red física. Este modelo se ha vuelto muy popular en la industria, específicamente en el modelo IaaS, donde se provee infraestructura a través de Internet mediante la creación de recursos virtualizados y enlaces lógicos sobre una infraestructura física de red [34].

La Figura 2.4 ejemplifica el concepto de Network Slicing, donde en una red física hay recursos o funciones de red (NFs, por sus siglas en inglés), los cuales pueden ser ocupados por diferentes slices, pero cada slice se gestiona de forma independiente.

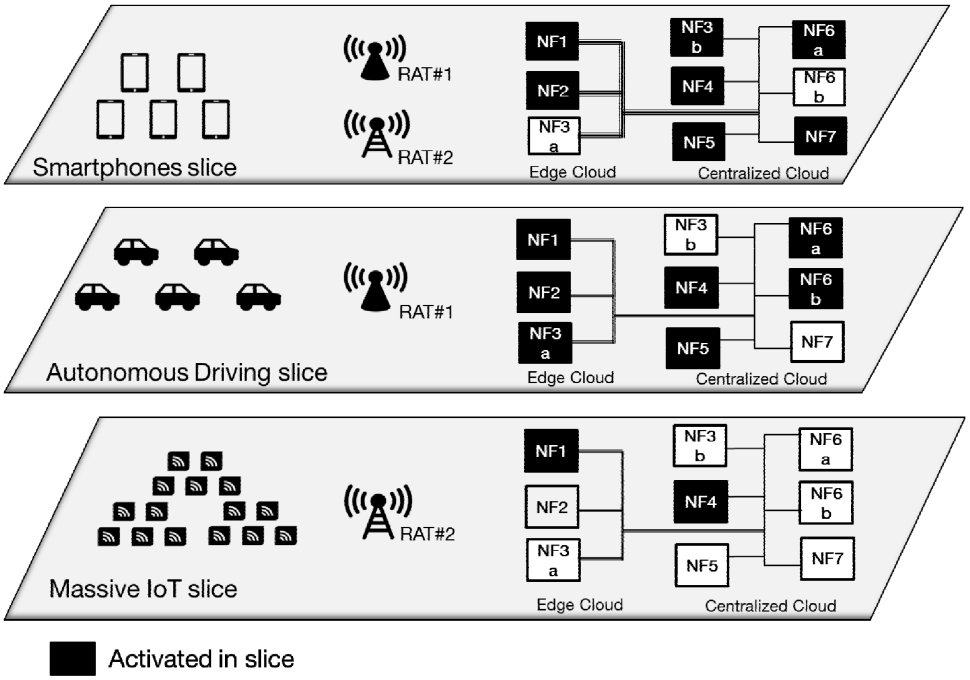

Figura 2.4: Ejemplo de slices sobre una misma red física [2]

Para hacer posible el Network Slicing son necesarias las tecnologías SDN y las Funciones de Virtualización de Red (NFV, por sus siglas en inglés). Las NFVs separan la lógica del hardware dedicado a la gestión de la red (i.e. firewalls, switches, balanceadores de carga) y lo empaqueta como software en máquinas o contenedores virtuales [35], de manera que se pueden crear y gestionar bajo demanda tantas NFVs como sean necesarias, ahorrando en costos y tiempo de despliegue.

2.4 Cómputo en la Nube

El cómputo en la nube es un modelo de provisión de centros de datos privados bajo demanda a través de Internet [36] (tal como se muestra en la Figura 2.5), los cuales se basan en tecnologías de virtualización para administrar eficientemente los recursos de cómputo [37]. Dado que la tendencia en el cómputo en la nube es acercar los servicios a los usuarios finales para favorecer tiempos de respuesta, se han establecido los términos *Fog Computing* o *Edge Computing* para este propósito. En este contexto, se contemplan dos tipos de centros de datos: *Core Data Center* (CDC) y *Edge Data Center* (EDC).

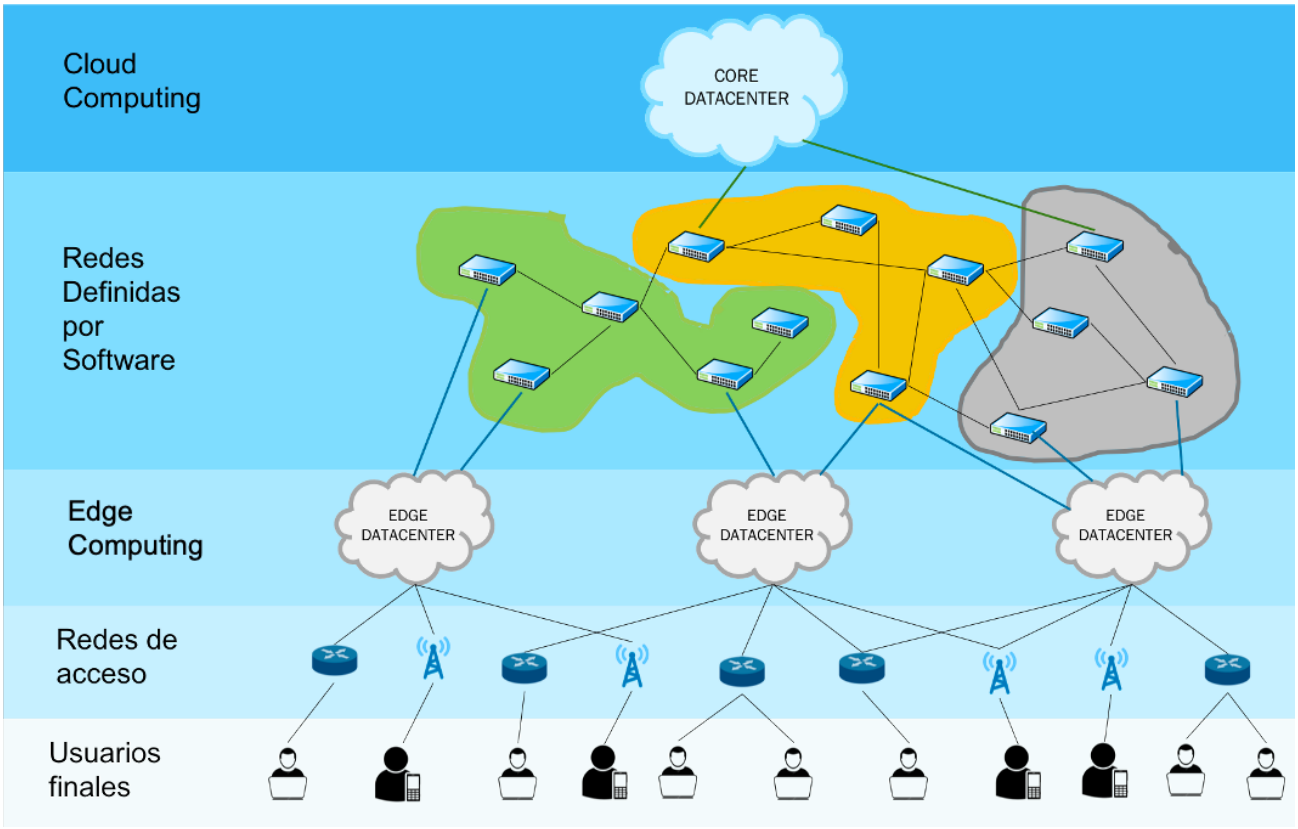

Figura 2.5: Arquitectura del cómputo en la nube a utilizar en este trabajo de investigación

Los CDCs han extendido la virtualización a todos sus recursos, como lo son el cómputo, el

almacenamiento y las redes, aprovechando tecnologías como SDN y NFV para habilitar las nubes definidas por software (SDC, por sus siglas en inglés) [37] para la automatización y gestión de recursos. Por su parte, en los EDCs se colocan los servicios disponibles en los CDCs, de manera que el acceso por parte del usuario final sea más rápido para favorecer a los servicios sensibles a latencia y servir a aplicaciones de tiempo real, haciendo que este acercamiento de los servicios provea una mejor QoS percibida por el usuario [36]. Para realizar tal acercamiento de servicios con QoS, los CDC y los EDC deben estar interconectados mediante redes que implementen QoS, entre las que se pueden mencionar aquellas que implementan *Multiprotocol Label Switching* (MPLS) [38], servicios diferenciados (DiffServ, del inglés *Differentiated services*) [39] o el reciente paradigma SDN, el cuál representa una evolución en cuanto a infraestructuras de red al proponer una abstracción de la lógica de su funcionamiento y de esta manera poder tratar las funciones de control y reenvío de flujos de forma separada, manteniendo el plano de datos en los switches de la red y centralizando el plano de control, permitiendo de esta forma programar las redes.

El cómputo en la nube permite la virtualización de recursos de cómputo y ofrecerlos como servicios con características de elasticidad, que permite a los consumidores hacer crecer o decrementar la capacidad de los recursos virtualizados de acuerdo a sus necesidades y sin interrupciones de servicio [40] y hecerlos accesibles a través de Internet. Tradicionalmente, el cómputo en la nube se ofrece en tres modalidades (ver Figura 2.6): 1) Infraestructura como Servicio (IaaS, por sus siglas en inglés), donde el cliente adquiere recursos de cómputo o de red virtualizados y es responsable de su administración [41], 2) Plataforma como Servicio (PaaS, por sus siglas en inglés) mediante servidores donde el cliente puede instalar, gestionar y mantener su software de acuerdo a sus necesidades, ignorando lo que suceda con el equipo que las aloja [42], y 3) Software como Servicio (SaaS, por sus siglas en inglés), que son aplicaciones disponibles a través de Internet como el correo electrónico o procesadores de texto [43].

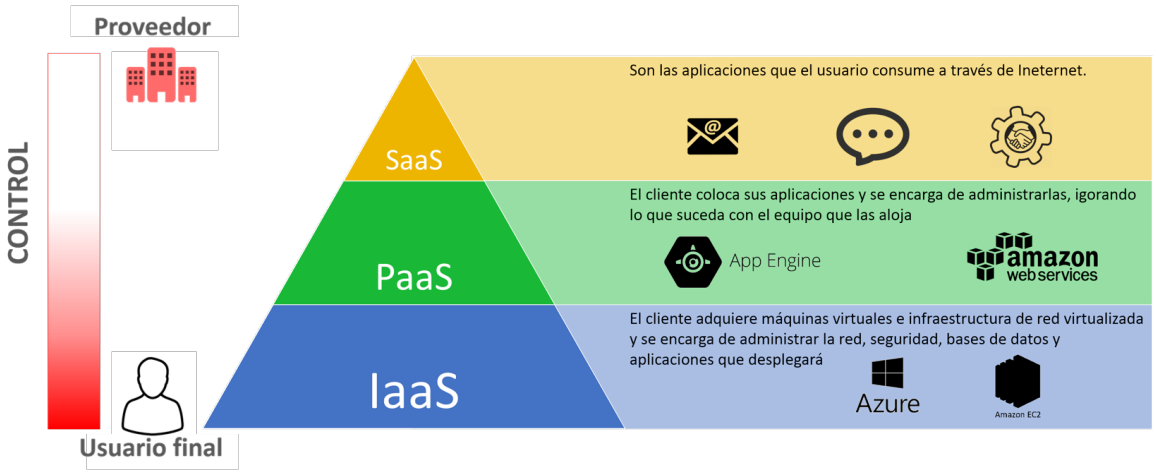

Figura 2.6: Modalidades principales del cómputo en la nube.

2.5 Cloud Slicing

Basado en la escencia del Network Slicing, el paradigma Cloud Slicing pretende cubrir los tres niveles de infraestructura por las que se transmite un servicio. Para esto se presenta el modelo de negocio "Slice como servicio" (SlaaS, del inglés *Slice as a service*), donde un proveedor de slices tiene a su alcance infraestructura de diferentes InPs y automatiza el proceso de provisión de slices, donde se configuran recursos virtuales entre dos centros de datos sobre la infraestructura disponible y se provee el acceso y la gestión al SP, de forma que pueda ser visto como un centro de datos aislado [11].

La Figura 2.7 ilustra el aprovisionamiento de recursos entre CDC y EDC para un slice, donde se obtienen recursos virtuales de diferentes InPs, tanto de cómputo como de red.

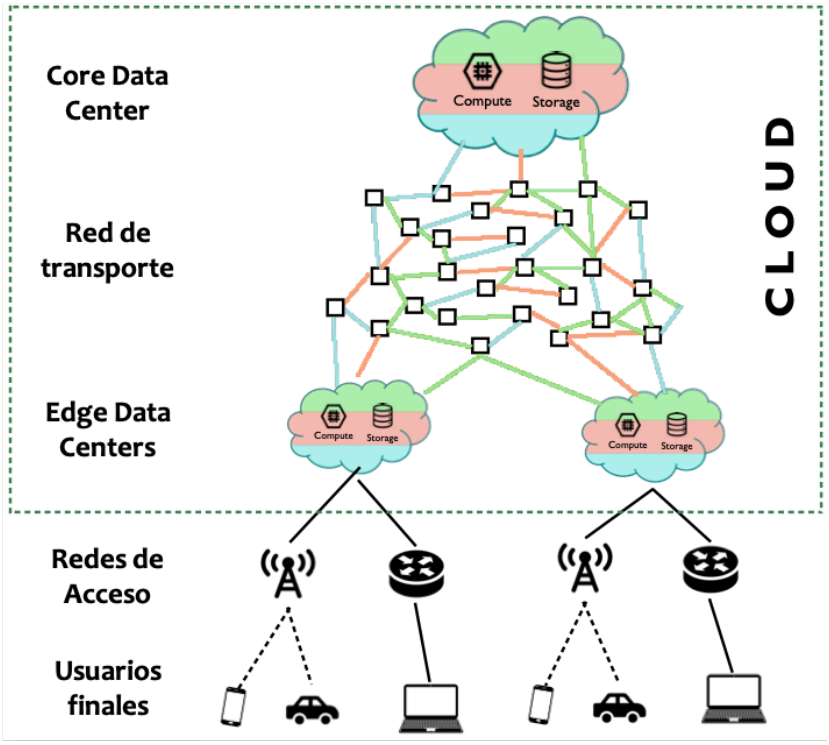

Figura 2.7: Ejemplo de slices sobre la nube, asumiendo que la nube está compuesta por infraestructura de red y cómputo de un gran número de InPs.

Como se puede observar, el Cloud Slicing está ligado al concepto Edge Computing tomando en cuenta todos los participantes del servicio de extremo a extremo y no sólo en el dominio del SP y/o el ISP. Siendo un tema técnicamente poco explorado, aún no existe un proveedor de slices que integre la red de transporte al slice y lo gestione como una entidad unificada.

2.6 Conclusiones parciales

Los conceptos revisados en este capítulo sirven para comprender el contexto en el que se desarrolla este trabajo de tesis. Como se puede observar, todos son conceptos relacionados, ya que el Clloud Slicing extiende el alcance del Network slicing, y este a su vez es posible gracias al paradigma SDN y las NFV. El slicing se lleva a la red de transporte y a los centros de datos que se ubican en los extremos de los slices para favorecer servicios sensibles a latencia. Se revisa la gestión de calidad de serivicio dado que es parte escencial de la propuesta para incorporar de manera funcional en una arquitectura de Cloud Slicing.

3

Estado del Arte

En este capítulo se presenta la revisión de los trabajos relacionados más representativos sobre arquitecturas de gestión de la QoS, así como arquitecturas de provisión de slices. La primera parte presenta arquitecturas robustas para proveer QoS a múltiples servicios en redes MPLS y DiffServ, así como arquitecturas simples para ambientes de redes definidas por software probadas con una cantidad de servicios reducida. La segunda parte presenta una revisión de la arquitectura de la solución *Novel Enablers for Cloud Slicing* (NECOS) y su proveedor de slices *Lightweight Slice Defined Cloud* (LSDC), la cuál es la única propuesta que ofrece cloud slices para gestionar la QoS de extremo a extremo en servicios entre centros de datos.

3.1 Arquitecturas de gestión de QoS basada en políticas en redes

El trabajo de tesis se desarrolla tomando en consideración una arquitectura que gestiona la calidad de servicio de slices para satisfacer sus requerimientos de SLA. En esta sección se describen las arquitecturas más representativas en el estado del arte para controlar la QoS de redes gestionables mediante automatización, como MPLS, DiffServ y SDN.

3.1.1 Traffic Engineering for Quality of Service in the Internet at Large scale: TEQUILA

Una solución muy popular en redes Multi-Protocol Label Switching (MPLS) y con Servicios Diferenciados (DiffServ) fue TEQUILA, una arquitectura modular para la gestión del plano de control como, por ejemplo, ejecutar mecanismos de control de subscripciones y gestión de los recursos de la red. Diversos trabajos fueron presentandos utilizando esta arquitectura, por lo que se puede representar de forma genérica como aparece en la Figura 3.1. Esta arquitectura funciona gracias a cuatro componentes principales: en la gestión de servicios se realizan tareas de control de admisión mediante subscripción de usuarios tomando en consideración la disponibilidad de recursos de la red, también se gestiona la invocación de servicios mediante el aprovisionamiento de recursos en la red. El componente de ingeniería de tráfico reconfigura la red con base en la utilización de los recursos para controlar la QoS. El componente de gestión de políticas controla los dos componentes anteriores mediante reglas de alto nivel que son descompuestas en mensajes de operación para el plano de datos. Por último, el componente de monitoreo envía notificaciones a los bloques de gestión de servicios e

ingeniería de tráfico para la toma de decisiones para planear la asignación de recursos.

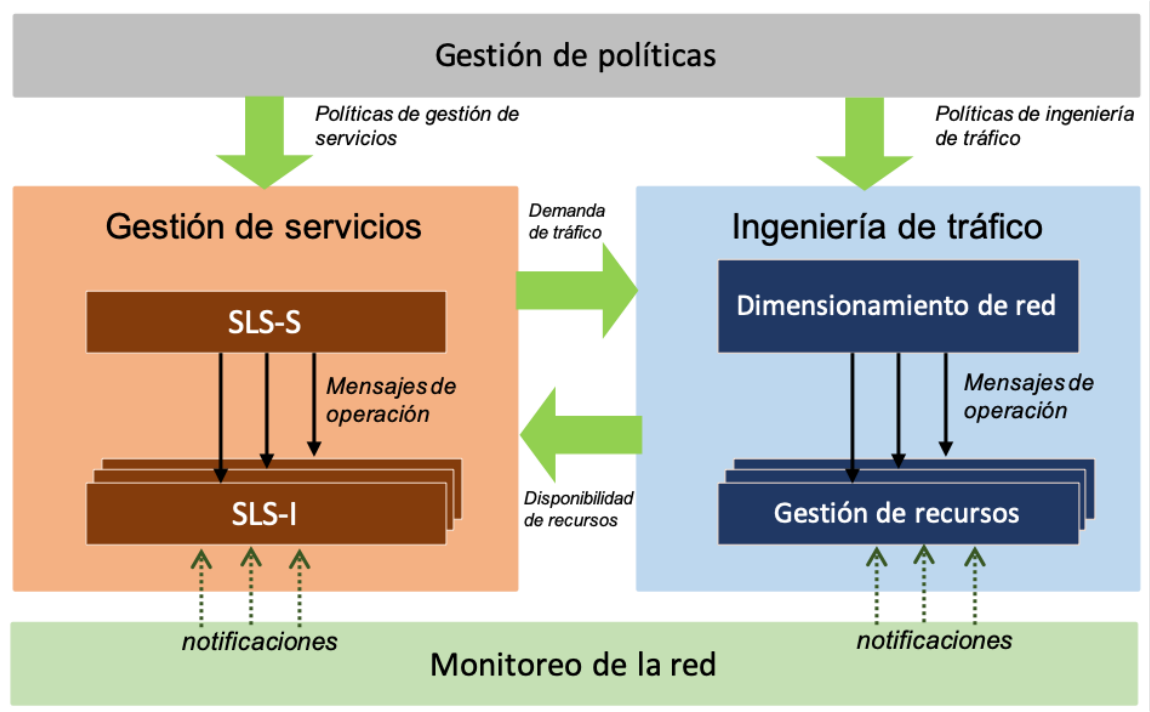

Figura 3.1: Arquitectura genérica del proyecto TEQUILA [3].

En [44], Trimintzios et al. proponen una implementación de TEQUILA en redes de servicios diferenciados sobre Internet, tomando en consideración una red MPLS subyacente con una configuración de ruteo preestablecida. El objetivo de este trabajo es ofrecer servicios servicios tanto cualitativos como cuantitativos.

El trabajo de Flegkas *et al.* [45] hace una revisión de la implementación de TEQUILA utilizando gestión basada en políticas para los procesos de orquestración de recursos.

En [46], Mykoniati *et al.* proponen modificar TEQUILA como un servicio holistico jerárquico, de manera que el funcionamiento de la plataforma funciona por etapas y el proceso de control de

admisión funciona mediante un modelo de retroalimentación.

El refinamiento de políticas de TEQUILA es abordado con un enfoque diferente en [33], donde Bandara *et al.* plantean el refinamiento de políticas elaborando objetivos y razonamiento abductivo para crear estrategias que cumplan con objetivos de alto nivel.

El trabajo propuesto por Rubio-Loyola *et al.* en [3] identifica indicadores de negocio en TEQUILA para configurar los recursos de la red y cumplir con una gestión del plano de control que logre controlar la calidad de servicio y maximizar las ganancias de los InPs.

3.1.2 Network Control Layer: NCL

En [47] Bueno et al. proponen NLC, un framework que utiliza monitorización para el estado de la red y, basado en políticas de QoS, reconfigura la red para transmitir los diferentes tipos de tráfico, seleccionando entre un conjunto de algoritmos de ingeniería de tráfico que calculan caminos para aproximar una QoS deseable para cada servicio de extremo a extremo.

El diseño de la arquitectura del NCL se observa en la Figura 3.2, donde el módulo SDNapp ejecuta una serie de procesos utilizando OpenFlow, como asignación de recursos, seguimiento de QoS y gestión de peticiones. El controlador SDN es una aplicación que genera los mensajes para configurar los dispositivos del plano de datos. El monitor SDN constantemente recaba información de la utilización de los recursos de la red.

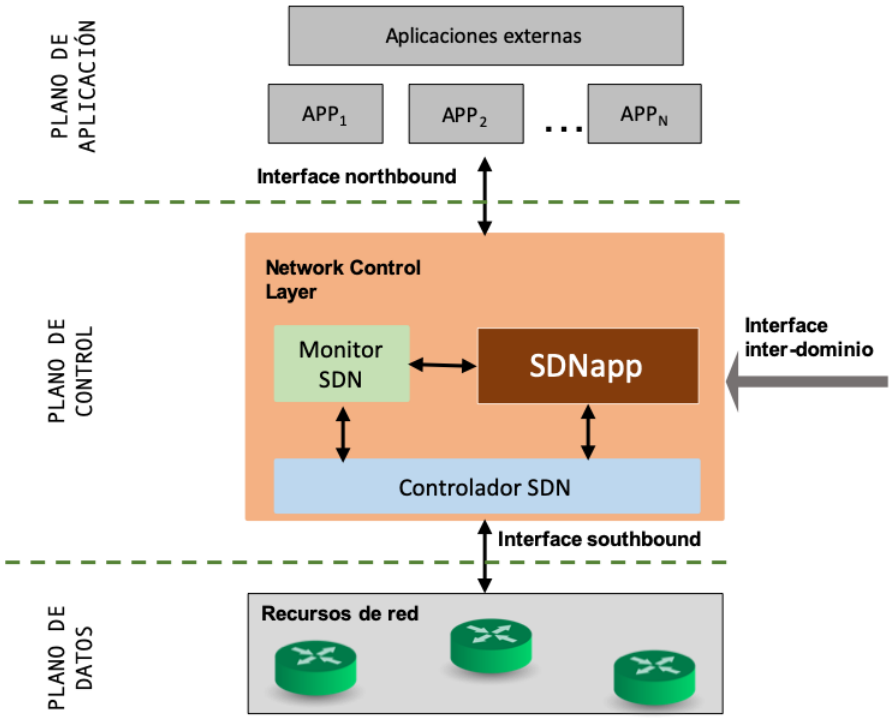

Figura 3.2: Arquitectura del controlador NCL.

NCL soporta las interfaces northbound y southbound, así como un interface inter-dominio para la comunicación con otras instancias de NCL en otros dominios. Esta arquitectura se basa en el paradigma SDN y en la plataforma OpenNaas y fue la primer arquitectura SDN que permitió la gestión de la QoS y la gestión de los recursos de acuerdo a su utilización y demanda.

3.1.3 PolicyCop

El trabajo [48] de Faizul Bari *et al.* presenta *PolicyCop*, un gestor de QoS para redes definidas por software que toma ventaja del monitoreo que se puede realizar continuamente de los recursos

de la red para ajustar el plano de control y cambiar en tráfico de la red. Lo que hace es crear colas en los enlaces de la red para proveer diferente tratamiento al tráfico mediante limitadores de tráfico. Las funciones de la red (i.e. control de admisión) funcionan como aplicaciones en el plano de control. Los resultados muestran que es efectivo al encontrar violaciones en las políticas, aunque el tiempo de respuesta depende directamente del intervalo de monitoreo.

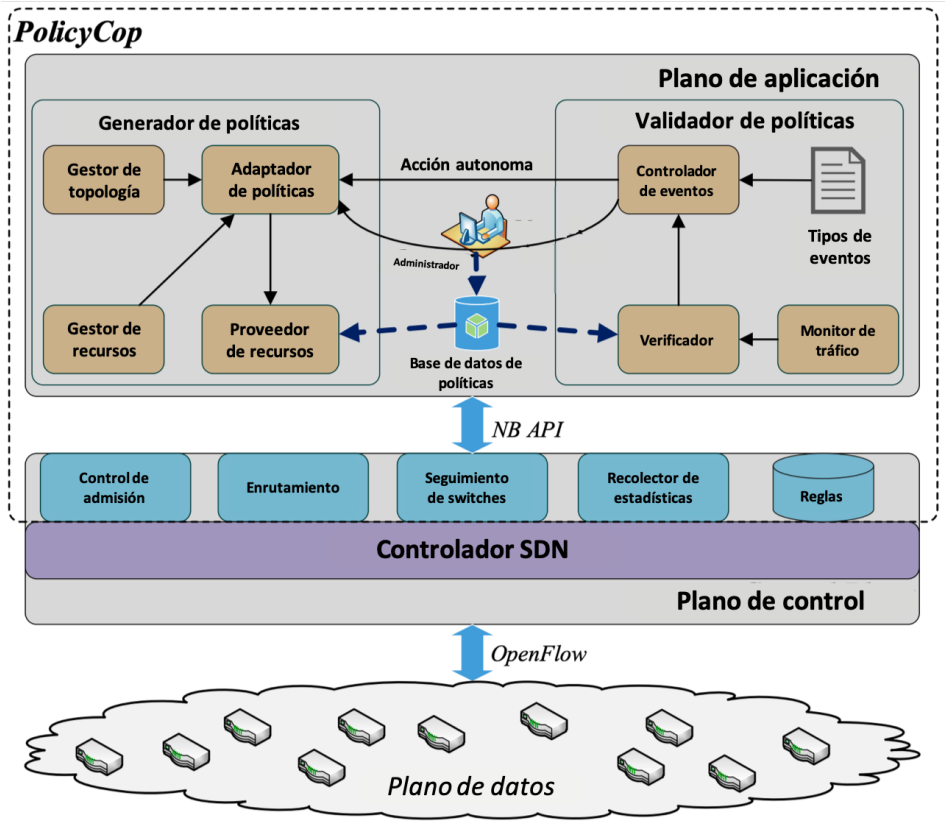

Figura 3.3: Arquitectura de PolicyCop.

La Figura 3.3 muestra el diseño modular de PolicyCop, donde se observan módulos comúnes en los gestores de QoS, como control de admisión, gestor de recursos y monitores. Aunque el diseño de PolicyCop puede parecer robusto en cuanto al alcance de su ejecución, su funcionamiento se limita a establecer límites en los enlaces de forma individual y depende del proceso de monitoreo para dar

una respuesta oportuna de configuración.

3.1.4 SDN-CADTR

Adami et al. presentan en [49] una aplicación para controlar tráfico diferenciado en una red simulada de switches OpenFlow y el controlador POX mediante la priorización de servicios diferenciados utilizando enrutamiento.

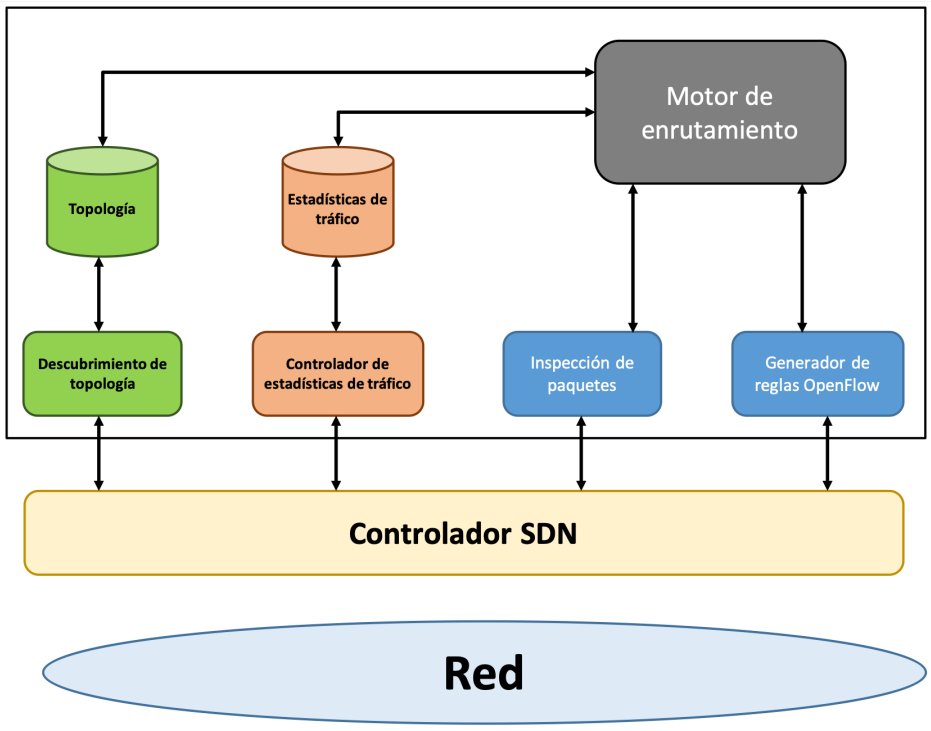

Figura 3.4: Arquitectura de SDN-CADTR.

La figura 3.4 muestra el diseño modular de la arquitectura, donde se muestran los procesos de descubrimiento de topología, el almacenamiento de estadísticas de tráfico y la inspección de paquetes, datos que sirven de entrada al motor de enrutamiento para gestionar la ingeniería de tráfico. El motor

de enrutamiento se comunica con el generador de reglas OpenFlow para instalar las reglas de flujos en los switches que integran las rutas seleccionadas. Esta propuesta permite tomar decisiones en tiempo real mediante el análisis de información del plano de datos.

3.2 Arquitecturas de provisión de slices

La arquitectura del Proveedor de Slices que se propone en este trabajo de tesis funciona en ambientes con múltiples infraestructuras a disposición para proveer segmentos de la nube (por nube nos referimos al conjunto de infraestructuras a disposición del Proveedor de Slices). Dado que el concepto de cloud slicing es nuevo y a la fecha sólo existe una solución que ha propuesto avances en esta área, en esta sección se describe la solución de la arquitectura NECOS (Novel Enablers for Cloud Slicing).

3.2.1 Novel Enablers for Cloud Slicing: NECOS

El concepto de cloud slicnig fue propuesto por el proyecto de investigación NECOS, el cuál nace de la necesidad de poder gestionar conjuntos completos de los recursos en servicios entre centros de datos de una manera inteligente y automatizada. Para esto se creó el proveedor de slices LSDC, que además fue diseñado con el objetivo de ser capaz de desplegarse en ambientes de recursos de cómputo de bajas prestaciones y sobre una cantidad elevada de servidores.

Como se muestra en la Figura 3.5, la arquitectura de NECOS se compone de varios dominios: a) el dominio del Tenant o SP, donde se solicitan los slices y puede hacer la gestión por su propia

cuenta, b) el dominio del proveedor de slices LSDC, la plataforma que permite orquestrar los slices y ejecutar servicios de telemetría remotos en los dominios donde se encuetren los recursos virtuales, c) y el dominio de recursos, que son los centros de datos y las infraestructuras de red sobre las que se puenden desplegar componentes por el LSDC.

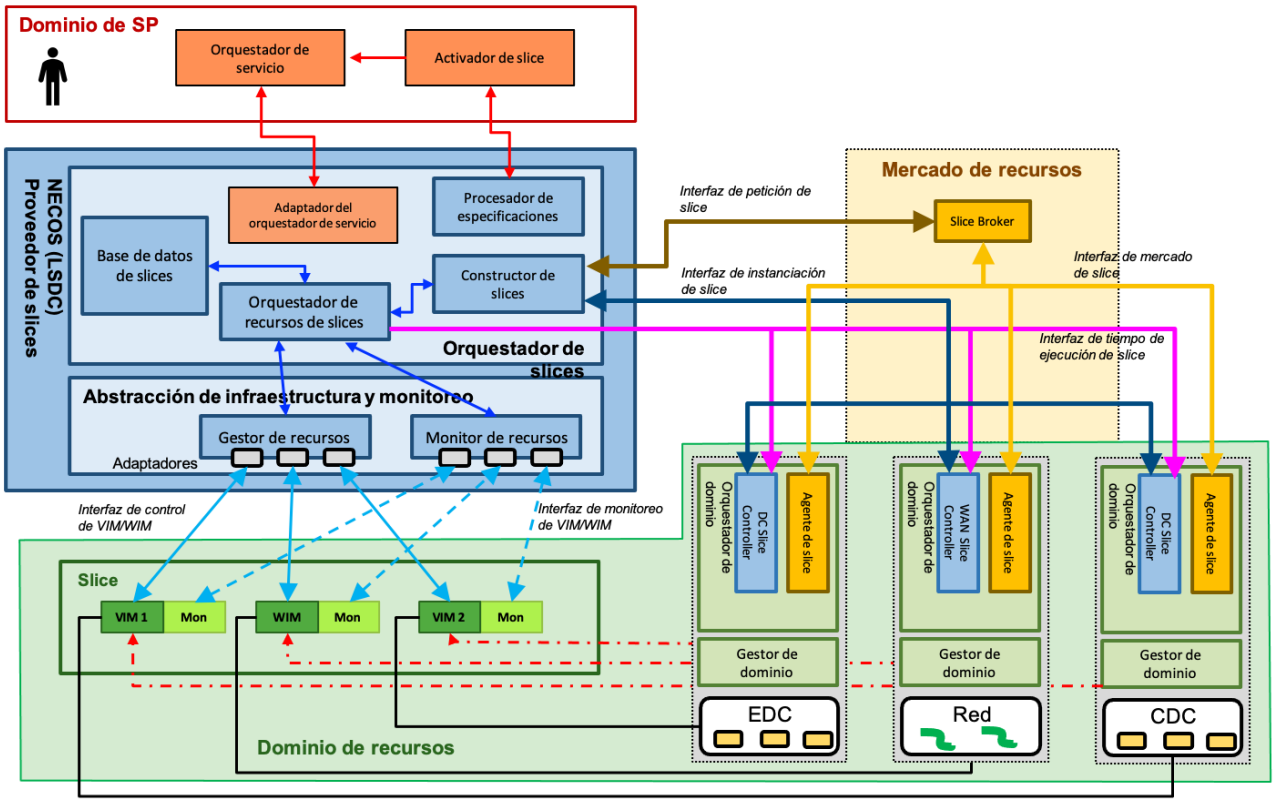

Figura 3.5: Arquitectura funcional de NECOS [4].

La implementación de NECOS se hizo mediante microservicios, aplicaciones aisladas que se pueden comunicar entre sí realizando tareas específicas. La comunicación entre los microservicios es mediante REST y utilizando archivos YAML para establecer las configuraciones de las solicitudes del slice, además de que los componentes tienen la capacidad de crear tales archivos para comunicarse con otros microservicios.

Aunque teóricamente se estableció el concepto de cloud slicing, una importante limitante de NECOS es que no ha sido posible comprobar la gestión de extremo a extremo de la QoS en el slice en términos de parámetros de red, ya que la implementación de su plataforma sólo define la creación de las partes de red al inicializar el slice, no hace un seguimiento de los recursos virtuales en la red durante el ciclo de vida del slice.

3.3 Conclusiones parciales

El control de la QoS basado en políticas ha sido poco explorado en ambientes de redes definidas por software, limitándose a arquitecturas que controlan situaciónes de congestión y redireccionamiento. Los trabajos más representativos en redes tradicionales tienen limitantes en cuanto al tiempo de reconfiguración debido al difícil monitoreo y la reconfiguración de los dispositivos de la red, además de que el refinamiento de políticas es una de las tareas más costosas.

De acuerdo a la mayoría del trabajo relacionado, en una arquitectura para la gestión basada en políticas existen algunos componentes comúnes: un módulo centralizado de inteligencia que decide la reconfiguración de la infraestrucutra de acuerdo a la información provista por dos componentes de bajo nivel, de los cuáles uno se encarga de monitorear la infraestructura y proveer información sobre la capacidad de la misma al módulo de inteligencia, mientras que el otro módulo gestiona las reglas sobre el funcionamiento de los dispositivos de la red, de manera que está en constante cambio (de acuerdo a la decisión del módulo de inteligencia) para gestionar la calidad de los servicios, de manera que estos cumplan con el SLA contratado.

Por parte de las arquitecturas de provisión de slices sólo hay una propuesta de cloud slicing, y

aunque tiene nociones de cómo se podría gestionar la QoS del slice completo, no se ha comprobado ni se han desarrollado los componentes necesarios para controlar la QoS en las infraestructuras de red que conectan los centros de datos.

4

Solución Propuesta

En este capítulo se describe la solución propuesta para el aprovisionamiento de slices para ambientes multi-cloud, con énfasis en los procesos de subscripción, invocación y control de QoS. El primero busca maximizar la cantidad de slices que puede alojar el Proveedor de Slices con la infraestructura disponible, el segundo se encarga de configurar la red para alojar recursos entre los centros de datos de un slice y el tercero se encarga de ejecutar reconfiguraciones en la red mediante la generación y administración dinámica de reglas de flujos en una Red Definida por Software (SDN, por sus siglas en inglés). El módulo de subscripción se valida mediante la inclusión de un algoritmo genético para aceptar o rechazar un slice al analizar la posibilidad de ejecución del mismo al ejecutar los slices ya subscritos. Para los módulos de invocación y control se prueban tres estrategias diferentes para validar el uso de técnicas como *plug-ins*.

La solución propuesta representa una prueba de concepto para el problema de gestión de slices entre centros de datos, mediante la orquestración de recursos virtualizados de extremo a extremo incluyendo las redes de transporte.

4.1 Arquitectura del Proveedor de Slices multi-cloud

El Proveedor de Slices multi-cloud es una plataforma de alto nivel para la gestión de slices en un ambiente que incluye infraestructura de diversos InPs tanto de red como de centros de datos. Esta plataforma recibe solicitudes de subscripción de slices para ser invocados bajo demanda por parte de los SPs para desplegar servicios de Edge Computing.

Esta plataforma se comunica con controladores de centros de datos para aprovisionar recursos virtualizados para que los SPs desplieguen sus servicios. Para comunicarse con controladores de SDNs necesita una aplicación intermedia de bajo nivel que obtiene información del plano de datos e instala reglas de flujos en el mismo. Esta aplicación fue llamada *"Topología, monitoreo y gestión de reglas"*, la cual provee datos al Proveedor de Slices para la toma de decisiones e instala las reglas de flujos dependiendo de dichas decisiones.

El Proveedor de Slices fue desarrollado en módulos en forma de microservicios, comunicando mediante REST, así como también la comunicación con las aplicaciones externas. A continuación se describen los componentes y entidades mostrados en la arquitectura del proveedor de slices mostrado en la Figura 4.1.

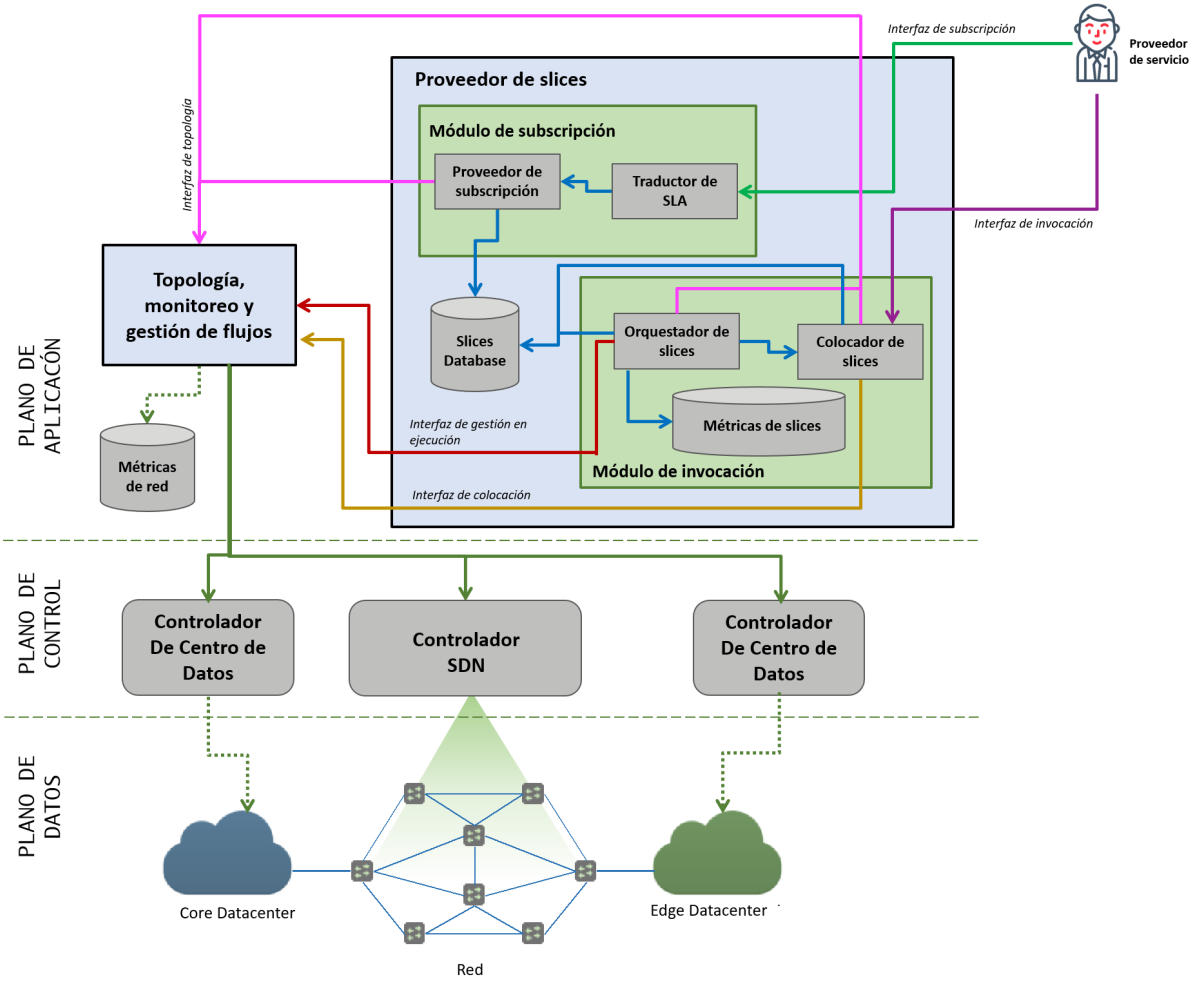

Figura 4.1: Arquitectura de la propuesta del Proveedor de slices.

4.1.1 El proveedor de servicios

El proveedor de servicios (SP, por sus siglas en inglés) puede ser una organización o individuo que requiere desplegar servicios de *Edge Computing*, pero que requiere de infraestructura de InPs para ubicar sus servicios en zonas donde no cuenta con recursos de cómputo. Para solucionar su problema, el SP elabora un SLA con los requerimientos del servicio a desplegar y solicita al Proveedor

de Slices una subscripción para dicho servicio y poder invocarlo en distintos momentos de acuerdo a la demanda.

4.1.2 El Proveedor de Slices

Como se mencionó anteriormente, esta plataforma facilita la localización, despliegue y gestión de recursos sobre distintas infraestructuras para proveer slices que funcionan como redes aisladas para el SP. Con esto se facilita el trabajo de los SPs, los cuales pueden centrarse en sus servicios y delegar el resto de trabajo de orquestración de recursos al Proveedor de Slices. Para realizar este trabajo, el Proveedor de Slices integra dos módulos críticos: el módulo de subscripción y el módulo de invocación.

4.1.2.1. Módulo de subscripción

Este módulo se encarga de la decisión de aceptar o rechazar slices para subscribirse de acuerdo a la capacidad de infraestructura y de los requerimientos de los slices subscritos. El objetivo de este componente es maximizar el número de slices subscritos y que se pueda asegurar el cumplimiento de sus SLAs. El proceso de subscripción puede observarse en la Figura 4.2.

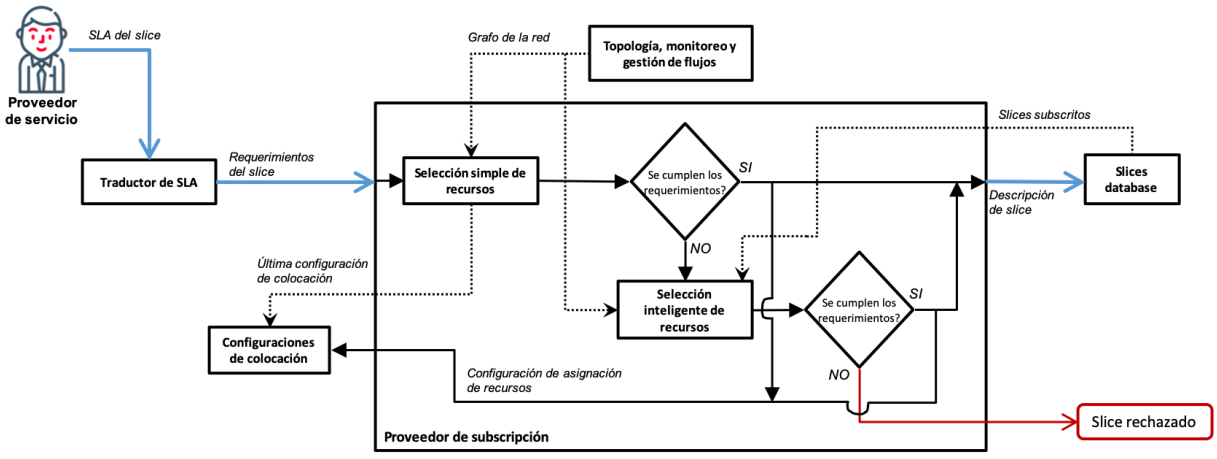

Figura 4.2: Proceso de subscripción de un slice.

El proceso de subscripción inicia con la traducción del SLA a requerimientos del slice, mediante la validación del tipo de membresía del SP y del tipo de servicio solicitado. Con base en estos dos parámetros se obtienen los requerimientos de prioridad, ancho de banda y enlaces disponibles mediante un diccionario de equivalencias. Un ejemplo de esta traducción se observa en la Figura 4.3, donde hay un mínimo y un máximo de ancho de banda que se traduce de acuerdo al tipo de servicio y se ajusta durante el ciclo de vida del slice para el control de calidad de servicio.

Muestra del SLA de un slice

slice_id	101
membership	gold
type_of_service	multimedia

Traducción

Requerimientos del slice

slice_id	101	priority	1
min_bw	8 Mbps	max_bw	12 Mbps
min_paths	1	max_paths	3

Figura 4.3: Ejemplo de obtención de los requerimientos de un slice mediante la traducción de un SLA.

Con los requerimientos del slice se procede a realizar una selección simple de recursos para el

slice, que es una búsqueda de una ruta que cumpla con los requerimientos del slice solicitante si se restan a los recursos disponibles de la red los recursos que necesitan los slices actualmente subscritos. Debido a su diseño modular, esta selección simple puede implementarse de diversas formas, en este caso se optó por utilizar el algoritmo de Dijkstra para obtener una respuesta en un lapso de tiempo corto si se compara con realizar cada combinación de slices en todas las configuraciones posibles. En esta etapa se consulta la disponibilidad y conectividad de switches OpenFlow en el plano de datos y se genera un grafo. Así mismo, se solicita la última configuración de colocación de slices subscritos en la red y de ahí obtener un subgrafo para ejecutar Dijkstra y decidir si existe una ruta con el ancho de banda disponible para el slice. En caso de que exista una ruta para el slice se acepta su subscripción y se agrega dicha ruta a la última configuración de colocación, se guarda en el repositorio de configuraciones de colocación y se guarda el slice en la base de datos *Slices Database*.

El algoritmo 1 detalla el proceso de selección simple para determinar si un slice solicitante puede subscribirse, de manera que se pueda satisfacer su SLA y el del resto de slices subscritos. Como datos de entrada se requiere un grafo R, el cuál se genera mediante la consulta al controlador de los switches y conectividad disponible, asumiendo que todos los enlaces tienen la misma capacidad de ancho de banda. El parámetro $config$ es la última configuración guardada en la base de datos *Configuración de colocaciones* y $slice$ es el requerimiento a subscribir. La Figura 4.4 muestra un ejemplo donde, recibiendo la última configuración de subscripción guardada y R, se crea el grafo R' para buscar una ruta para $slice$.

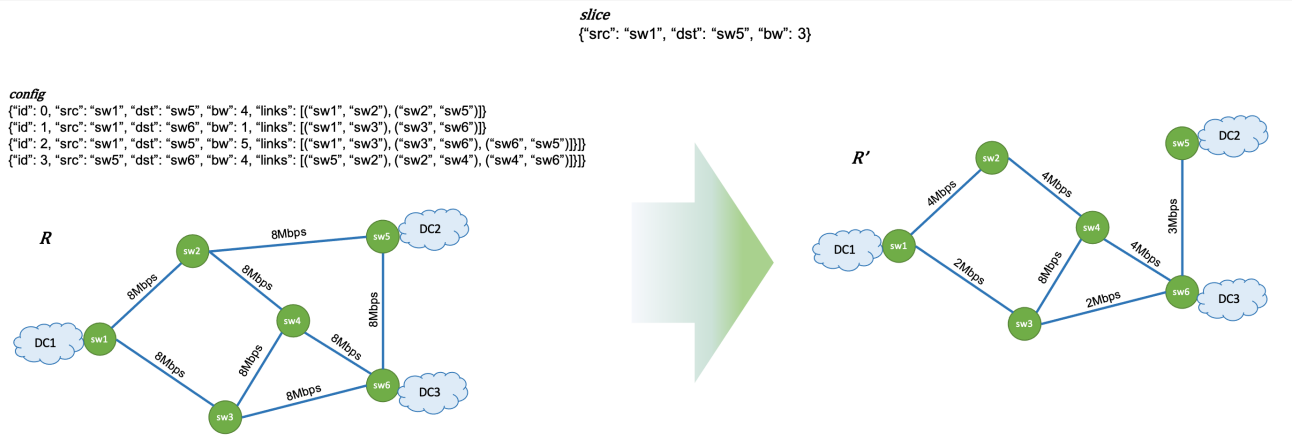

Figura 4.4: Ejemplo de creación de R' a partir de R y $config$, donde se busca subscribir a $slice$ buscando una ruta en R'.

Una vez obtenido R' se inicializa una cola de prioridad Q, así como diccionarios para guardar la $distancia$ de cada nodo al nodo inicial y los nodos visitados en $visitado$. Se agrega el nodo inicial $slice.src$ a la cola de prioridad, el cuál se obtiene de los requerimientos del slice al conocer los centros de datos de los extremos, así como el peso del nodo, el cuál es igual ala distancia al mismo nodo (0). Después se extraerá el nodo u con menor peso de la cola de prioridad hasta que se agoten, si el nodo ya ha sido visitado, se continúa con el siguiente nodo, en caso contrario el nodo se marca como visitado y se consulta cada nodo vecino v, obteniendo el ancho de banda bw del enlace entre los dos nodos. Si el nodo vecino v no ha sido visitado y el peso de $distancia[u] + bw$ es menor al peso de $distancia[v]$, se actualiza el peso de v respecto al nodo inicial en $distancia$ con el valor de $distancia[u] + bw$, agrgando el nodo v y su peso en $distancia$.

Algoritmo 1 Algoritmo Dijkstra para determinar si un slice puede ser subscrito.

Entrada: R, $config$, $slice$

por cada $slice\_s \; \epsilon \; config$ **hacer**

$R \leftarrow R - slice\_s$

fin por

$R' \leftarrow R$

$Q \leftarrow []$

$distancia \leftarrow \{\}$

$visitado \leftarrow \{\}$

$anterior \leftarrow \{\}$

por cada n en R **hacer**

$distancia[n] \leftarrow \infty$

$visitado[n] \leftarrow Falso$

$anterior[n] \leftarrow -1$

fin por

$distancia[slice.src] \leftarrow 0$

$Q \cup (slice.src, distancia[slice.src])$

mientras $Q \neq []$ **hacer**

$u \leftarrow min(Q)$

$Q.eliminar(min(Q))$

si $visitado[u]$ **entonces**

Continuar

fin si

$visitado[u] \leftarrow Verdadero$

por cada v en $R[u]$ **hacer**

$bw \leftarrow R[u][v]$

si $!(v.visitado)$ **entonces**

si $(distancia[u] + bw) < distancia[v]$ **entonces**

$distancia[v] \leftarrow distancia[u] + bw$

$Q \cup (v, distancia[v])$

$anterior[v] = u$

fin si

fin si

fin por

fin mientras

$ruta = [slice.dst]$

$destino = slice.dst$

mientras $anterior[destino] \neq -1$ **hacer**

$ruta \cup anterior[destino]$

$destino = anterior[destino]$

fin mientras

Salida: $invertir(ruta)$

La ruta se obtiene al revisar el diccionario $anterior$, donde se consulta el nodo destino $slice.dst$ y devuelve el nodo anterior de la ruta, debiendo hacer la consulta de forma recursiva hasta llegar al nodo inicial $slice.src$ que apunta a -1.

En caso de que no se obtenga una ruta con Dijkstra, se procede a realizar una selección inteligente de recursos, y en este caso se optó por utilizar un algoritmo genético canónico (AGC) [50] debido a la experiencia del autor y la facilidad de adaptación a este problema. Lo que hace el AGC es crear configuraciones aleatorias de colocación de los slices subscritos y el solicitante en la red. El AGC comprobará que las colocaciones de cada configuración sean válidas, y se contabilizará el número de colocaciones. El objetivo es encontrar que la configuración con mayor número de colocaciones sea igual a la cantidad de slices subscritos y el slice solicitante. Si dicha condición se cumple, el slice se acepta como subscrito, en caso contrario el slice se rechaza.

Los parámetros y el AGC se detallan en el Algoritmo 2. Como entrada se recibe una lista $slices$ y un diccionario R, la primera incluye los requerimientos de ancho de banda de los slices subscritos y el slice solicitante, mientras que el segundo representa un grafo de la red con los switches disponibles como nodos y los enlaces como aristas, cada arista con un peso que indica el ancho de banda total de dicho enlace. El parámetro t inicializa el contador de generaciones que el AGC tendrá que recorrer hasta alcanzar el máximo de generaciones G, mientras no se encuentre una configuración que permita subscribir a la lista $slices$. El parámetro μ indica la cantidad de individuos de cada generación.

La primera instrucción es generar una población inicial $\mathcal{P}(0)$, donde cada individuo x_i se genera de manera aleatoria y representa una configuración de asignación de recursos de R. Cada x_i está compuesto por $genes$, donde la cantidad de estos es igual al tamaño de la lista $slices$ y cada gen es una ruta seleccionada de forma aleatoria entre todas las rutas posibles que hay entre los dos servidores del $slice$, de forma que un individuo es una configuración de asignación de recursos para

cada slice. La configuración del individuo puede o no ser correcta, para eso se verifica que al terminar con las asignaciones de recursos por cada individuo, los enlaces de R son capaces de alojar el ancho de banda de los slices que le fueron asignados, por lo que se contabiliza la cantidad de slices cuyo ancho de banda se puede satisfacer (se evalúa x_i). Como se busca garantizar que el slice solicitante puede ser alojado en R cumpliendo con el requerimiento de ancho de banda y también cumplir con los demás slices subscritos, el AGC sólo considera una configuración donde la cantidad de slices colocados sea igual al tamaño de la lista $slices$.

La selección de padres se realiza en forma de ruleta, esto es, se seleccionan el conjunto de parejas $((x_0, x_1), (x_1, x_2), (x_2, x_3), ..., (x_{\mu-1}, x_\mu), (x_\mu, x_0))$. En la etapa de cruza, se definió en un 90 % de probabilidad de cruza y, si una pareja de padres es cruzada, de manera aleatoria se elige un punto entre los genes de los individuos, de forma que el nuevo individuo sea una configuración con colocaciones de los individuos anteriores, como se muestra en la figura 4.5.

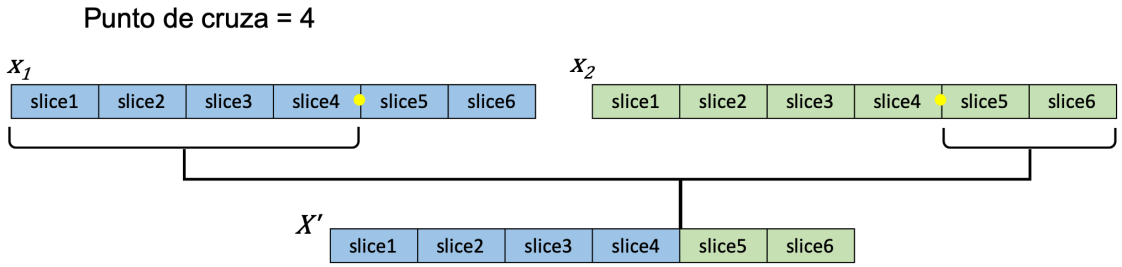

Figura 4.5: Ejemplo de la cruza de dos individuos que representan configuraciones de colocación de slices.

En la etapa de mutación, se definió una probabilidad de mutación de 10 % para cada gen del individuo, y en caso de que un gen sea mutado, se obtiene una nueva ruta para el $slice$ asignado a ese gen. Cada nuevo individuo se agrega a la población de la siguiente generación ($\mathcal{P}'(t)$), y al terminar se agrega el individuo con el mayor número de colocaciones a la nueva población, por lo

que sobrevive el más fuerte.

Algoritmo 2 Algoritmo genético para búsqueda de recursos de un nuevo slice.

Entrada: $slices$, R

$t \leftarrow 0$

$\mu \leftarrow 10$

$G \leftarrow 100$

Generar $\mathcal{P}(0)$ con μ individuos

mientras $t < G$ **hacer**

  por cada $x_i \; \epsilon \; \mathcal{P}(t)$ **hacer**

    evaluar x_i por cantidad de colocaciones

    si tamaño($slices$) == $f(x_i)$ **entonces**

      regresar x_i

      Fin

    fin si

  fin por

  $\mathcal{P}'(t) = \emptyset$

  por i = 1 hasta i = μ **hacer**

    $padres = seleccionarPadres(\mathcal{P}(t))$

    $x_i' = cruza(padres)$

    $x_i' = mutar(x_i')$

    $\mathcal{P}'(t) = \mathcal{P}'(t) \cup \{x_i'\}$

  fin por

  $x^* \leftarrow max(\mathcal{P}(t))$

  Insertar x^* en $\mathcal{P}'(t)$

  $t = t + 1$

fin mientras

4.1.2.2. *Slices Database*

La base de datos *Slices Database* almacena los requerimientos de cada slice subscrito y el estado en que se encuentra (e.g. en espera, en ejecución). Esta base de datos es consultada por los módulos de subscripción e invocación para guardar nuevos slices disponibles o para determinar los valores de ancho de banda para realizar tareas de reconfiguración durante el ciclo de vida de los slices.

4.1.2.3. *Módulo de invocación*

El proceso de invocación se inicia mediante la solicitud del SP para desplegar sus servicios subscritos, por lo que la tarea del Proveedor de Slices es gestionar los recursos virtuales para instalar las reglas de flujo que permitan al slice proveer un servicio dentro de sus requerimientos.

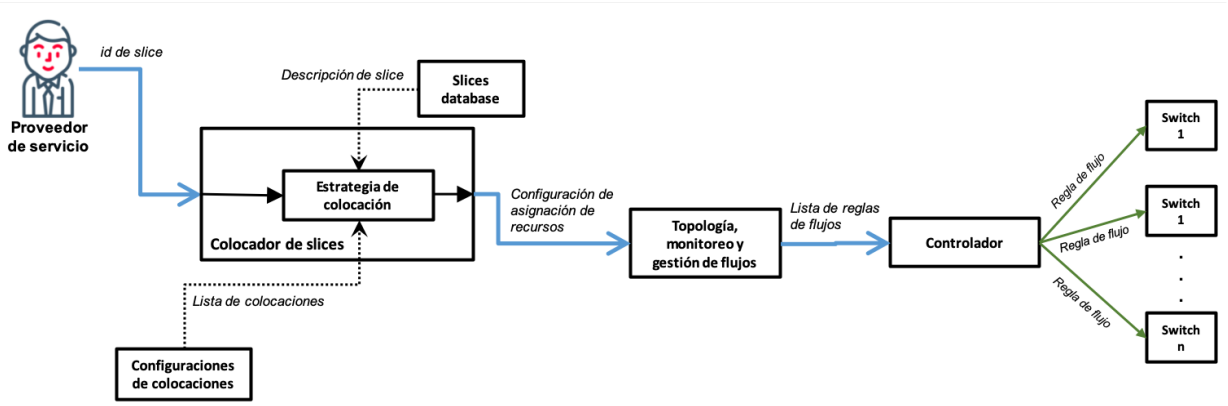

Figura 4.6: Proceso de invocación de un slice.

El proveedor de slices ordena inicializar su slice, por lo que la estrategia de colocación a seguir obtiene la topología de la red y las configuraciones de colocación almacenadas, las cuáles servirán como conjunto de datos. Una vez encontrada la configuración, se envía la lista de slices colocadas a

la aplicación de gestión de flujos para crear e instalar las reglas de flujos y asignar el ancho de banda indicado en sus requerimientos. La Figura 4.6 muestra el flujo del proceso de invocación.

Como estrategias de colocación se probaró una búsqueda voraz, un clasificador Bayesiano y el clasificador de bosques aleatorios.

Con la estrategia de búsqueda voraz se identifican los slices actualmente invocados y el slice que solicita invocación y se realiza una búsqueda secuencial donde se toma en cuenta la primera coincidencia encontrada en la base de datos de *Configuraciones de colocación*. Esta configuración encontrada implica hacer una reconfiguración total al plano de datos, por lo que tanto para búsqueda como para instalación necesita un lapso de tiempo considerable.

Para el caso del clasificador Bayesiano, de acuerdo a los slices actualmente invocados y del slice solicitante se realiza una búsqueda secuencial en la base de datos de *Configuraciones de colocación* donde coincidan los slices involucrados, se asigna una clase a cada configuración (en este caso el nivel de ajuste de ancho de banda del slice solicitante) y con base en los niveles del ajuste de ancho de banda de los slices actualmente invocados se calcula qué tan probable es la pertenencia de ajuste de ancho de banda del slice solicitante en el escenario actual. Un ejemplo de coincidencias encontradas se muestra en la Tabla 4.1, donde hay doce coincidencias con slices que han sido invocados (4, 7, 1, 9) y un slice que solicita subscripción (2). Las coincidencias son escenarios en donde se han ejecutado esos slices con diferente asignación de ancho de banda. En la tabla se muestra sólo el ancho de banda, que es el valor utilizado por el clasificador, pero una vez que se obtenga la clase (nivel de ancho de banda) se configurará la ruta del slice en esa configuración.

Tabla 4.1: Ejemplo de escenarios donde coinciden ciertos slices invocados y un slice que solicita invocación.

slice_4	slice_7	slice_1	slice_9	slice_2
1.75	2	5	2.5	6
1.75	2.5	5	2	6
1.5	2.5	6	2.5	5
1	2	7	2.5	6
1.75	2.5	5	2	5
1.5	2.5	5	2	6
1.75	2	6	2.5	6
1.75	2	6	2	6
1.5	3	5	2.5	6
1	2.5	5	3	5
1.75	2	5	2.5	5
1.75	2	6	2.5	6

Dado que el clasificador Bayesiano se basa en las probabilidades de que ocurra un suceso habiendo sucedido otro que influyera en esto, por cada nivel de ancho de banda para el slice solicitante en la Tabla 4.1 se calcula la probabilidad de que el ancho de banda para el slice solicitante sea igual a ese valor tomando en consideración las coincidencias de los demás slices. Por ejemplo, si en el escenario actual se requiere invocar al $slice\_2$, cuyo requerimiento de ancho de banda está entre 4 y 8 Mbps, y se tienen los siguientes slices:

$$\{slice\_4 : 1{,}75, slice\_7 : 2{,}5, slice\_1 : 6, slice\_9 : 2{,}5\}$$

la probabilidad de cada ancho de banda encontrado para el $slice\_2$ se calcula como:

$$P(x|c) = P(slice\_4 = 1{,}75) * P(slice\_7 = 2{,}5) * P(slice\_1 = 6) * P(slice\_9 = 2{,}5)$$

$$P(x|5) = \frac{2}{4} * \frac{3}{4} * \frac{1}{4} * \frac{3}{4} = 0{,}0703$$

$$P(x|6) = \frac{5}{8} * \frac{2}{8} * \frac{3}{8} * \frac{5}{8} = 0{,}0366$$

$$P(x) = \sum_{i=0}^{n} P(c_i|x)P(C_i)$$

$$= P(5|x)P(5) + P(6|x)P(6)$$

$$= 0{,}0703 + 0{,}0366 = 0{,}1069$$

$$P(5|x) = \frac{0{,}0703}{0{,}1069} = 0{,}657$$

$$P(6|x) = \frac{0{,}0366}{0{,}1069} = 0{,}342$$

por lo que al $slice\_2$ se le asigna un de ancho de banda de 5 Mbps y la ruta establecida en la primera configuración.

Para el clasificador de bosques aleatorios se siguió la misma estrategia inicial que en el clasificador Bayesiano, de acuerdo a los slices actualmente invocados y del slice solicitante se realiza una búsqueda secuencial en la base de datos de *Configuraciones de colocación* donde coincidan los slices involucrados, se asigna una clase a cada configuración (en este caso el nivel de ajuste de ancho de banda del slice solicitante) y con base en los niveles del ajuste de ancho de banda de los slices actualmente invocados se generan 100 árboles de decisión, sin establecer un máximo de

profundidad, y se toma un 40 % de muestras de las coincidencias totales. Cada árbol se evalúa de forma independiente y se toma como predicción el ancho de banda más evaluado por los árboles. Para este caso se utilizó la librería *scikit-learn* de Python.

La Figura 4.7 muestra el ejemplo de los campos utilizados por el Proveedor de Slices para generar una regla de flujo. El campo *datapath* indica el identificador del switch al que se le instalará esta regla, la cual es enviada como un mensaje del controlador al switch. Por cada invocación o reconfigurción debido al control de QoS, se realiza la búsqueda de configurciones detallas anteriormente, y por cada ruta de un slice, se identifican los switches mediante su datapath y el puerto al que está conectado cada switch vecino, por lo que de esta manera se genera una regla de flujo para cada switch. El campo *priority* indica la prioridad de esta regla con relación a otras, en caso de un conflicto se tomará en cuenta aquella regla con una mayor prioridad. Al ser reglas dinámicas, al instalar una nueva regla con las mismas coincidencias, se tomará en cuenta la nueva regla, mientras que la anterior será eliminada. Este mecanismo se utiliza para evitar pérdidas de paquetes durante el cambio de configuración, donde mientras algunas reglas se instalan otras son elimindas. Cabe resaltar que este campo de prioridad es diferente al parámetro de prioridad del Proveedor de Slices, donde en ese caso, el campo de prioridad indica el orden de preferencia al asignar cambios en el ancho de banda (e.g. un slice con prioridad 2 tendrá preferencia al requerir más ancho de banda que otro slice con prioridad 3). El campo *match* indica las coincidencias en los campos a tomar en cuenta para aplicar las acciones de la regla, para este caso se utilizan el puerto de entrada, la dirección MAC de origen y la dirección MAC de destino (para identificar el slice). Todas las acciones del gestor de reglas son el tráfico los servidores del slice, por lo que las acciones siempre son *out_port*, que indica el puerto de salida del flujo. Al hacer cambios en un slice, se ejecuta una de estas reglas por cada switch que compone el slice.

datapath	priority	match		actions	
3	1	in_port	2	out_port	5
		eth_src	de:00:80:52:4a:00		
		eth_dst	78:7b:8a:bb:f5:fc		

Figura 4.7: Ejemplo de una regla de flujo generada por el Proveedor de Slices.

Cuando un slice es invocado, se genera un hilo para monitorear cada slice, de forma que se detecte cuando el parámetro establecido de ancho de banda se está utilizando por encima o debajo de determinados umbrales. Cuando esto ocurre durante un periodo de tiempo, el monitor de slice repite la estrategia de colocación utilizada en invocación, pero en esta ocasión se solicita un mayor o menor ancho de banda (dentro de lo establecido en los requerimientos) y se busca una configuración que permita mantener la utilización de ancho de banda fuera de esos umbrales, es decir, se gestiona la QoS basándose en la elasticidad de los slices. El ancho de banda fue el parámetro seleccionado para la gestión, debido a que es más fácil controlarse y tiene menos restricciones que, por ejemplo, el retardo. En esta implementación se utiliza un sólo controlador de la red, por lo que para escenarios de producción es necesario utilizar varios controladores en cooperación para mantener centralizado lógicamente el plano de control.

La Figura 4.8 muestra el flujo del proceso de gestión de reglas de flujos. Durante la búsqueda de configuración de colocaciones se respetan los niveles de prioridad en caso de encontrarse conflictos entre slices. Por cada slice invocado se genera un hilo que realiza el seguimiento del porcentaje de utilización den ancho de banda del slice respecto a sus requerimientos y asignación en un momento dado, si dicho porcentaje está por encima o por debajo de un umbral, entonces se desencadena la estrategia de colocación para ese slice, siguiendo las mismas estrategias utilizadas en el proceso de invocación.

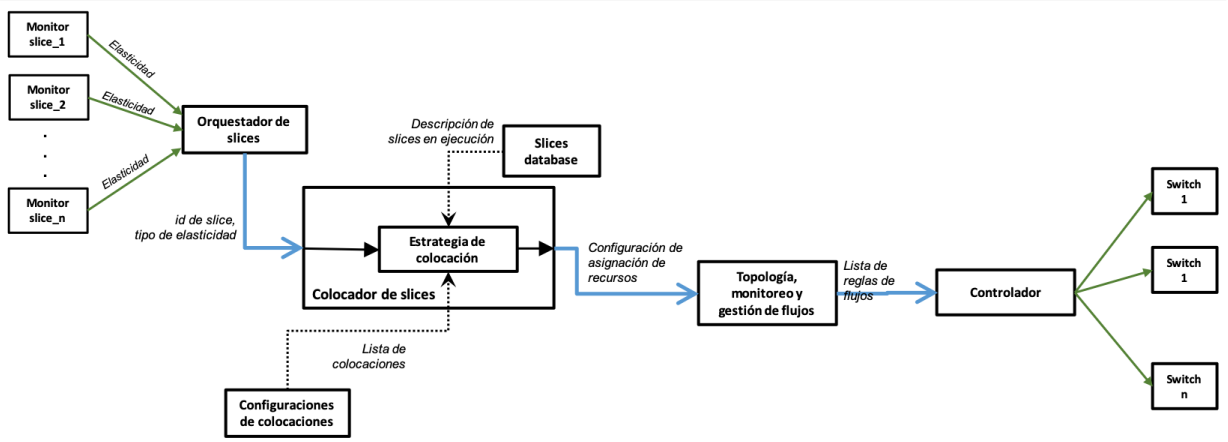

Figura 4.8: Proceso de gestión de reglas de flujos para el control de la QoS de slices durante su ciclo de vida.

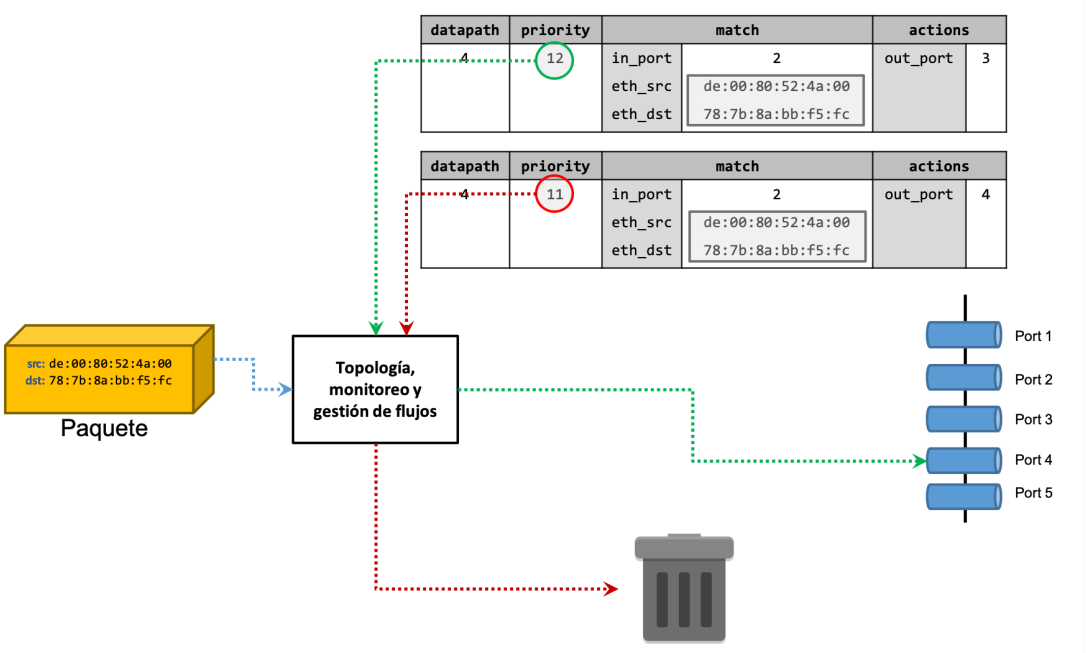

Figura 4.9: Ejemplo de un conflicto de reglas de flujos.

En la Figura 4.9 se observa cómo dos reglas de flujos tienen la misma coincidencia para un paquete entrante. Debido al control de la QoS de los slices es necesario reconfigurar el plano de

datos constantemente, por lo que la ruta del enlace lógico del slice cambia de la misma forma. Para identificar las rutas nuevas, se incrementa el valor del parámetro de prioridad, por lo que al encontrar conflictos se toma en consideración la regla con el valor más grande de prioridad, mientras que la regla anterior es eliminada. En este ejemplo, los paquetes entrantes se enviarán ahora por el puerto 4, dado que el switch conectado en el puerto 3 ya no forma parte del slice.

4.1.3 Topología, monitoreo y gestión de reglas

Esta aplicación de propósito de bajo nivel se comunica con el controlador SDN para recolectar información de utilización y del estado actual de los switches, lo cuál es importante para mantener una topología actualizada para la toma de decisiones.

El Proveedor de Slices utiliza las direcciones MAC de origen y destino para identificar el tratamiento a los flujos por los servidores que componen el slice. Otros valores para detectar coincidencias pueden ser el puerto de entrada del paquete o las direcciones IP, aunque para esto se requiere previamente una etapa de descubrimiento entre los servidores para llenar los registros en sus tablas del protocolo de resolución de direcciones (ARP, por sus siglas en inglés). Otras acciones que se pueden realizar con una regla de flujo, además de enviarlos por un puerto específico es eliminar el paquete, esto se hace cuando por ejemplo se detectan usuarios maliciosos en la red, o puede enviarse al controlador para definir qué hacer con él en caso de que el switch no tenga una regla para ello o para modificar el contenido de los paquetes. Las reglas generadas se envían del controlador al switch que implementará dicha regla mediante el protocolo OpenFlow, como se indica en la sección de *Redes Definidas por Software* en el capítulo 2.

El Algoritmo 3 muestra la manera en que se gestionan las reglas de flujos de acuerdo a la

configuración recibida del Proveedor de Slices. Como entrada se toma en cuenta un grafo G, el cuál es construido mediante la información de monitoreo del plano de datos. Se recibe una ruta R del Proveedor de Slices y un ancho de banda cada vez que se configura una nueva ruta para un slice. Se toman el inicio y el fin de R como las direcciones MAC de cada servidor, mientras que el resto son los switches que conectan tales servidores. Después, por cada switch, se identifican los switches de inicio y fin, los cuáles están conectados a los servidores, y se identifica el puerto al que está conectado el servidor correspondiente para guardar como puerto de salida $outPort$ o puerto de entrada $inPort$ según el orden de la lista. También se identifica el switch del cuál es vecino para guardar el puerto de la misma manera, por lo que cada switch genera una coincidencia de un origen-destino y puerto de entrada y se toma la acción de enviar por un puerto para definir la ruta al recorrer todos los switches del slice. Finalmente se instala la regla en el switch correspondiente, agregando la acción a tomar si se coincide con el par origen-destino y el puerto de entrada.

Algoritmo 3 Gestión de reglas de flujos de acuerdo a la configuración recibida del Proveedor de Slices.

Entrada: G, R, BW

$src \leftarrow R[0]$

$dst \leftarrow R[-1]$

$R \leftarrow R[1:-1]$

por cada $switch\ \epsilon\ R$ **hacer**

  si $switch == R[0]$ **entonces**

    $inPort \leftarrow switch.getPort(src)$

  si no

    $inPort \leftarrow switch.getPort(prevSwitch)$

  fin si

  si $switch == R[-1]$ **entonces**

    $outPort \leftarrow switch.getPort(dst)$

  si no

    $outPort \leftarrow switch.getPort(nextSwitch)$

  fin si

  $accion \leftarrow enviarPor(outPort)$

  $coincidencia \leftarrow [puertoEntrada = inPort, ethDst = dst, ethSrc = src]$

  $agregarRegla(switch, coincidencia, accion)$

fin por

4.1.4 Controladores SDN y de centros de datos

Estos componentes se encargan de gestionar los recursos virtuales en infraestructuras físicas. Los controladores de centros de datos se encargan de proveer máquinas virtuales bajo demanda con tecnología de contenedores y establecer los servicios a ejecutar mediante archivos de configuración. Por su parte, los controladores SDN permiten configurar los dispositivos del plano de control de la

red para controlar el tráfico de acuerdo a decisiones de alto nivel.

En esta implementación se utilizó el *DC Slice Controller* de NECOS [12], el cual es un servicio REST empaquetado en un contenedor que requiere de servidores de virtualización para desplegar máquinas virtuales y alojar los servicios requeridos por los SPs en contenedores. En el caso del controlador SDN, el controlador pertenece al Proveedor de Slices, siendo necesario que los switches de las redes de los InPs establezcan la dirección IP del controlador.

Para aprovisionar los recursos virtules el SP envía al Proveedor de Slices durante la subscripción un archivo YAML con las especificaciones que debe tener el slice. Después de que el slice es subscrito, el Proveedor de slices envía a la aplicación de *Topología, monitoreo y gestión de reglas* las líneas a ejecutar en las máquinas virtuales a desplegar. Una vez recibidas las líneas, la aplicación se comunica vía REST con los controladores para desplegar las máquinas virtuales con la cantidad de contenedores especificada e instalar las herramientas y configuraciones de las líneas de configuración recibidas del SP.

En la red, el aprovisionamiento de recursos se realiza en tiempo de invocación y durante el ciclo de vida del slice, haciendo la reconfiguración dinámica descrita en las secciones anteriores mediante los datos de utilización del plano de datos y de los slices. Durante el proceso de subscripción, la red no se configura para el slice subscrito, ya que las máquinas virtuales no están ejecutando el servicio requerido, sólo se instalan las herramientas necesarias y se conectar cuando el slice es invocado, cambiando dinámicamente los switches que componen el slice pero manteniendo el mismo origen y destino en los centros de datos.

4.2 Conclusiones parciales

La solución propuesta sirve como intermediario entre los SPs y los InPs, de manera que se provean slices a los SPs utilizando técnicas de virtualización y reglas de flujos sobre los recursos físicos de los InPs. Para tal efecto, la arquitectura presentada en este trabajo de tesis fue diseñada para cumplir con tres principales objetivos: 1) gestionar infraestructuras de centros de datos y redes de transporte de diferentes InPs, 2) crear y gestionar la colocación de slices mediante la asignación de recursos, y 3) gestionar reglas de flujos de manera dinámica para satisfacer los requerimientos de QoS de los slices.

El desarrollo del Proveedor de Slices siguió un diseño modular con microservicios para su facilidad de implementación en contenedores y con comunicación REST, lo que lo hace flexible para adaptar los módulos de esta arquitectura a otros mecanismos de gestión de infraestructura. Como requerimientos para los InPs de centros de datos es necesario utilizar el *DC Slice Controller* de NECOS, ya que es el controlador compatible con la arquitectura de este trabajo, mientras que los InPs de redes requieren configurar el parámetro de controlador en sus switches OpenFlow para apuntar al controlador de la arquitectura.

La presente arquitectura tiene algunos beneficios respecto al trabajo de NECOS, destacando la capacidad de gestionar efectivamente la totalidad del slice al incluir un gestor de reglas de flujos para las redes de transporte. Otro de los beneficios es la capacidad de permitir la inclusión de mecanismos de gestión como *plug-ins*, lo que permite probar diferentes estrategias de colocación al apuntar a la dirección IP del servicio ejecutando de una estrategia determinada, algo que el diseño de NECOS tiene, pero la implementación se realizó con diferentes demostraciones [51]. Entre las desventajas que la propuesta tiene es que por el momento soporta slices con sólo dos extremos, por lo que en algunos

servicios será necesario utilizar algunos centros de datos como puntos de unión. Otra desventaja que se planea abordar a futuro es la utilización de un sólo controlador SDN para todas las redes de transporte, lo cuál crearía retrasos importantes en ambientes de redes a gran escala, haciendo irrelevantes los cambios de reglas de flujos en las diferencias del tiempo de detección de cambio y el tiempo de instalación de reglas.

5

Pruebas de validación y resultados

En este capítulo se describen las pruebas experimentales para validar la arquitectura propuesta del Proveedor de Slices en ambientes multi-cloud. Se comienza con una revisión de los tipos de servicios utilizados en este trabajo para validación, los cuáles son tomados de la literatura y se basan en el concepto de *Edge Computing*. Se describe el testbed de emulación desarrollado para la ejecución de los experimentos y se presentan los resultados que demuestran las capacidades del Proveedor de Slices para asignar recursos a slices, así como la capacidad de orquestrar tales recursos durante el ciclo de vida de los slices para controlar su QoS mediante el ajuste de elasticidad de acuerdo a su demanda.

5.1 Servicios de Edge Computing utilizados

Para la validación del Proveedor de Slices se seleccionaron tres tipos de servicios basados en Edge Computing para desplegar slices bajo demanda, debido a que se necesita validar que el Proveedor de Slices controle la calidad de servicio de slices con distintos requerimientos. Los tres servicios tienen rangos de valores de requerimientos distintos, y se muestran en la Tabla 5.1.

Tabla 5.1: Tipos de servicios y sus requerimientos

Servicio	**Requerimientos**	
	Ancho de banda (Mbps)	Confiabilidad (Rutas dedicadas)
Multimedia	4 - 8	1 - 3
Redes vehiculares	2 - 4	1 - 2
Redes de dispositivos IoT	1 - 2	1

El tipo de servicio con requerimientos más demandantes es el multimedia, dado que realiza transferencia de videos, documentos, imágenes, etc., además de una confiabilidad variable dependiendo del SP para utilizar enlaces alternos en caso de que ocurran fallas en los enlaces principales. Los servicios de redes vehiculares requieren un menor ancho de banda en comparación a los servicios multimedia debido a que para estas pruebas se utilizan aplicaciones de seguridad que envían y reciben datos de ubicación de vehículos cercanos. Para estas pruebas, el tipo de servicio menos demandante es el de redes de dispositivos de Internet de las cosas (IoT, por sus siglas en inglés), ya que la comunicación constante se da entre los servidores en los EDC y los dispositivos de sensado, utilizando la comunicación entre CDC y EDC para transmitir bloques de datos de manera

periódica. Los requerimientos se determinaron de acuerdo a la carga de trabajo que se aplica mediante un generador de tráfico.

A continuación se detallan los servicios que se utilizan en este trabajo de tesis como servicios para validación.

5.1.1 Servicio multimedia: provisión de réplicas de servidores multimedia en redes de entrega de contenidos

Este servicio consiste en crear réplicas de servidores que alojan contenido multimedia al hacerse virales. Jahromi et al. propusieron en [52] una arquitectura para una plataforma basada en microservicios y NFVs que detecta un gran número de solicitudes para algún contenido multimedia en una diferente zona geográfica y, para aliviar el congestionamiento del servidor demandado y reducir la latencia de los usuarios, el SP contacta con un ISP en dicha zona geográfica para desplegar un servidor con el contenido demandado.

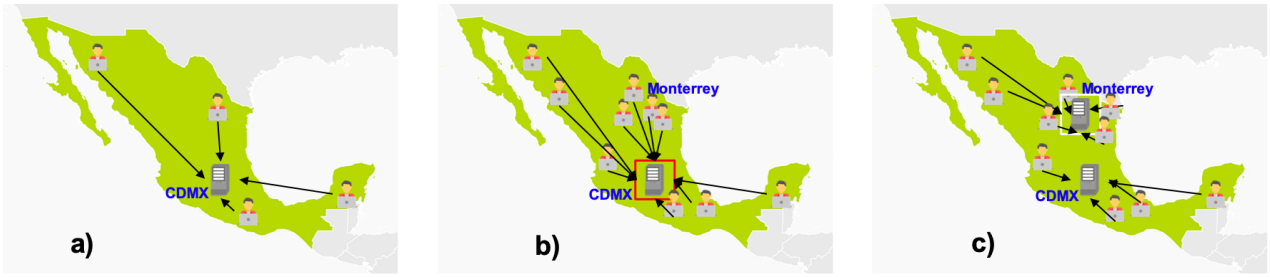

Figura 5.1: Provisión de contenido multimedia bajo demanda. **a)** Un contenido multimedia está alojado en un centro de datos, en **b)** el contenido comienza a hacerse viral y en **c)** el contenido multimedia se copia a otro centro de datos en una zona de demanda para aliviar la congestión del primer centro de datos y reducir la latencia percibida por los usuarios.

La Figura 5.1 muestra el ejemplo en donde un servidor en CDMX recibe solicitudes por contenido, después dicho contenido se vuelve viral y comienza a recibir grandes solicitudes de zonas geográficas particulares. Para evitar la degradación de la QoS, la plataforma solicita de forma automatizada a un ISP en Monterrey, la cuál es la zona con más demanda, la creación de un nuevo servidor para alojar el contenido viral y establece redireccionamiento a los servidores dependiendo de la zona geográfica de la solicitud para balancear la carga de trabajo.

En las pruebas de validación de este trabajo de tesis el Proveedor de Slices es quien realiza las tareas de comunicación con otros InPs para desplegar un nuevo servidor y además crear los enlaces lógicos entre los servidores para la transferencia de contenido y datos de solicitudes, es decir, crea un slice en la nube en donde los dos servidores se encuentren comunicados en su propia red aislada.

Este trabajo de réplicas de contenido multimedia ha sido reproducido para las pruebas del Proveedor de Slices utilizando contenedores virtuales y ha sido ligeramente modificado, de modo que las tareas de localización son sustituidas por la ubicación explícita en el archivo de configuración del slice, mientras que el contacto para despliegue de infraestructura se delega al Proveedor de Slices.

5.1.2 Servicio de redes vehiculares: provisión de slices para control de tráfico vehicular.

El servicio de control de tráfico vehicular permite aumentar el conocimiento de los vehículos en una ciudad inteligente mediante la comunicación de redes vehiculares ad-hoc (VANETs, por sus siglas en inglés) aisladas y la diseminación de información de vehículos no alcanzables para aumentar la calidad de la información de seguridad disponible para los conductores o vehículos inteligentes.

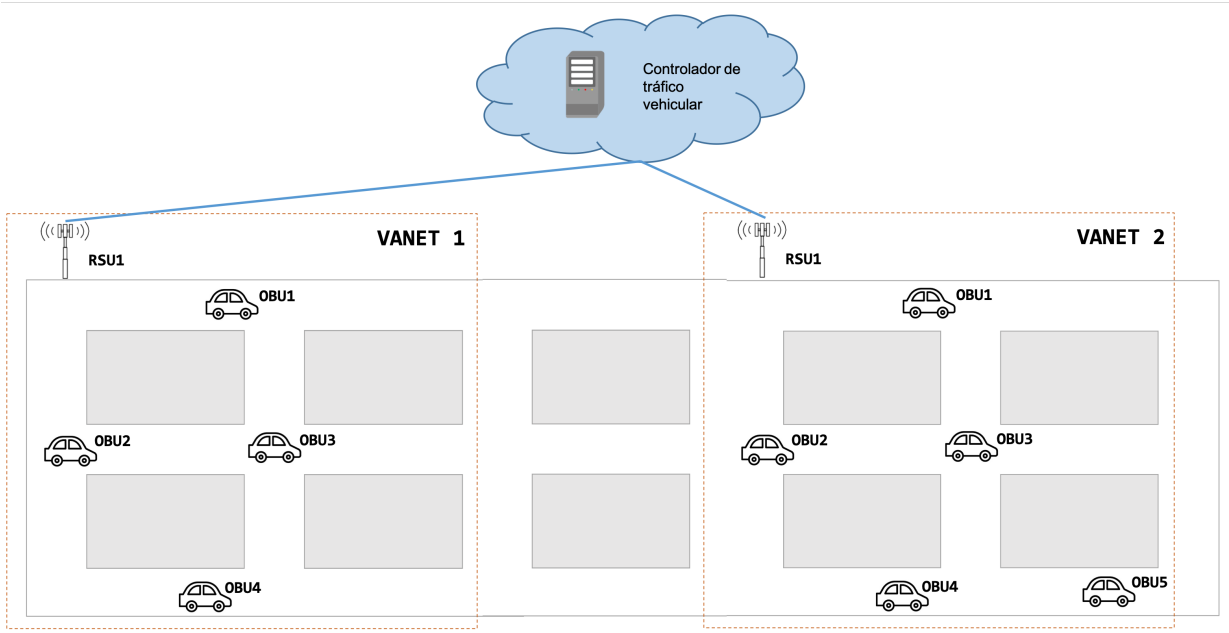

Figura 5.2: Creación de slices para controlar el tráfico vehicular a incrementar el conocimiento de los vehículos sobre su vecindario al incluir datos de posicionamiento de otras redes vehiculares.

La Figura 5.2 ejemplifica el uso de este servicio mediante la creación de un slice consistente en dos servidores en el borde alojados en las RSUs (del inglés *Road Side Units*) y un servidor central que funciona como controlador de tráfico. Dado que hay áreas en una zona geográfica donde no hay vehículos, algunos vehículos inalcanzables en un momento dado no pueden preveer que en unos instantes puede congestionarse alguna calle por la que han planeado transitar, pero gracias a la existencia del slice los vehículos de ambas VANETs pueden recibir información de otros vehículos fuera de su rango de comunicación. El slice permite tambien planear los tiempos de duración de semáforos para un eficiente control de tráfico vehicular.

Para las pruebas de validación del proveedor de slices, este servicio se limita al intercambio de datos de localización y estado de los vehículos.

5.1.3 Servicio de dispositivos de IoT: monitoreo de condiciones marítimas.

Este servicio hace uso de la plataforma Dojot [53], la cuál es un proyecto de código abierto para proveer servicios públicos y privados para ambientes de IoT. Se compone de microservicios basados en contenedores para un despliegue rápido de los componentes de forma distribuida en ambientes multi-cloud. Como caso de uso se toma como referencia un sistema de monitoreo de las condiciones marítimas para ayudar a empresas pesqueras en el proceso de toma de decisiones para despliegue de puntos de pesca.

La Figura 5.3 muestra el despliegue de la plataforma Dojot bajo el concepto de slicing, utilizado como caso de uso en la solución NECOS [11] y se ha reproducido en este trabajo de tesis. El *slice part 1* se aloja en un CDC, dado que es el punto central para la toma de decisiones a nivel global para ubicar zonas de pesca y almacenar datos históricos. El resto de *slice parts* se alojan en EDCs cercanos a las costas para recibir datos de ubicación de barcos y datos de las condiciones marítimas de dicha ubicación. Para esto cada barco incluye sensores de monitoreo y envían dichos datos de forma asíncrona al *slice part* más cercano.

Para las pruebas de este trabajo de tesis se generó el tráfico de los barcos hacia el *slice part* más cercano utilizando la herramienta Locust [54], una aplicación con interfaz web para la simulación de múltiples usuarios para probar la capacidad de carga de trabajo de servidores o centros de datos.

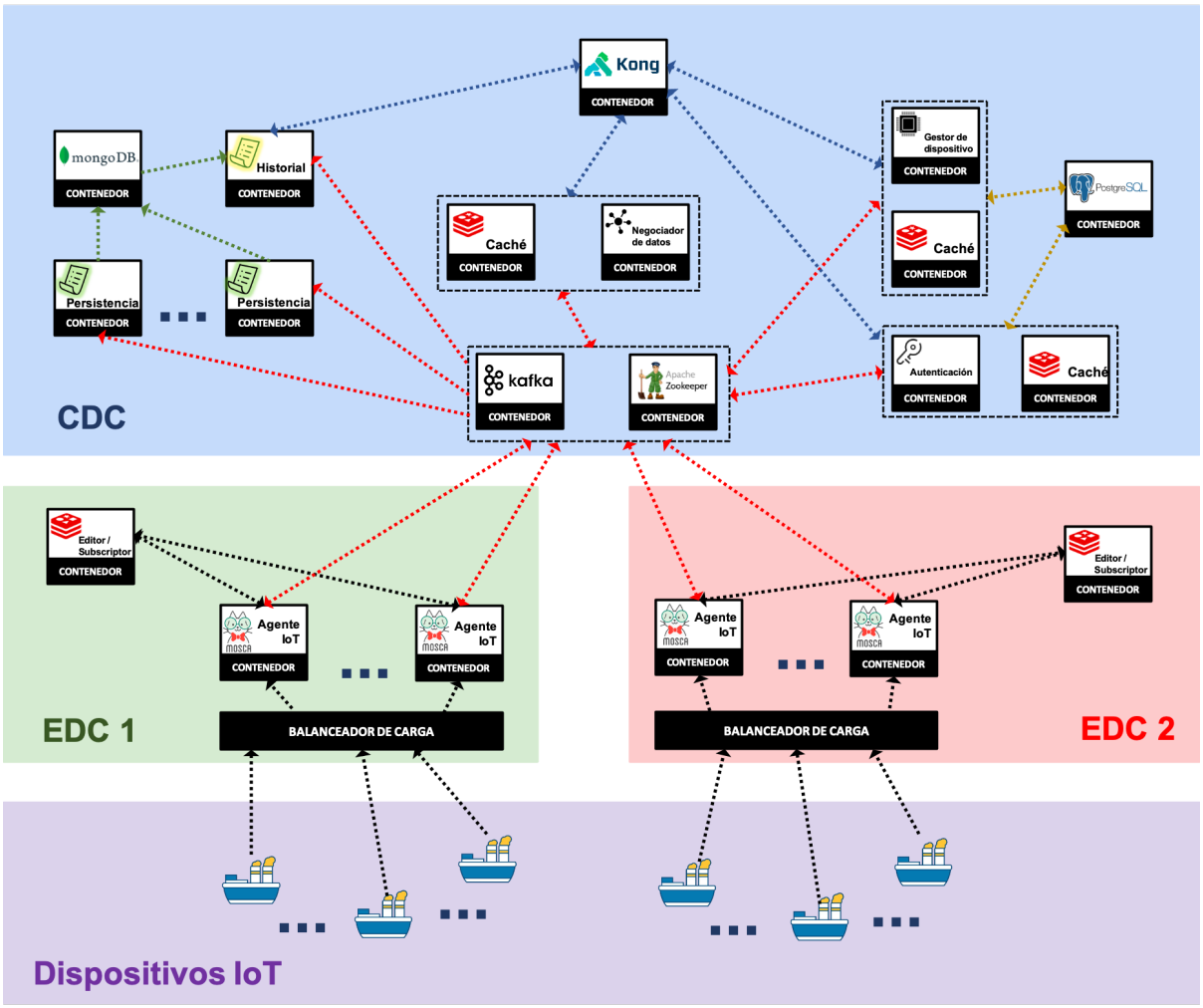

Figura 5.3: Servicio de monitoreo de condiciones marítimas, donde los barcos en el mar envían datos como ubicación y temperatura al centro de datos más cercano, a partir de ahí los datos se envían en bloques a un CDC en periodos de 5 minutos para su análisis y toma de decisiones.

5.2 Configuración del testbed de validación

Para validar la arquitectura propuesta fue necesario diseñar una topología que incluyera InPs tanto de redes de transporte como de centros de datos. La tarea del Proveedor de Slices es gestionar el conjunto de infraestructuras como una entidad y proveer segmentos de ella a los SPs. La topología utilizada para las pruebas de este trabajo se muestra en la Figura 5.4.

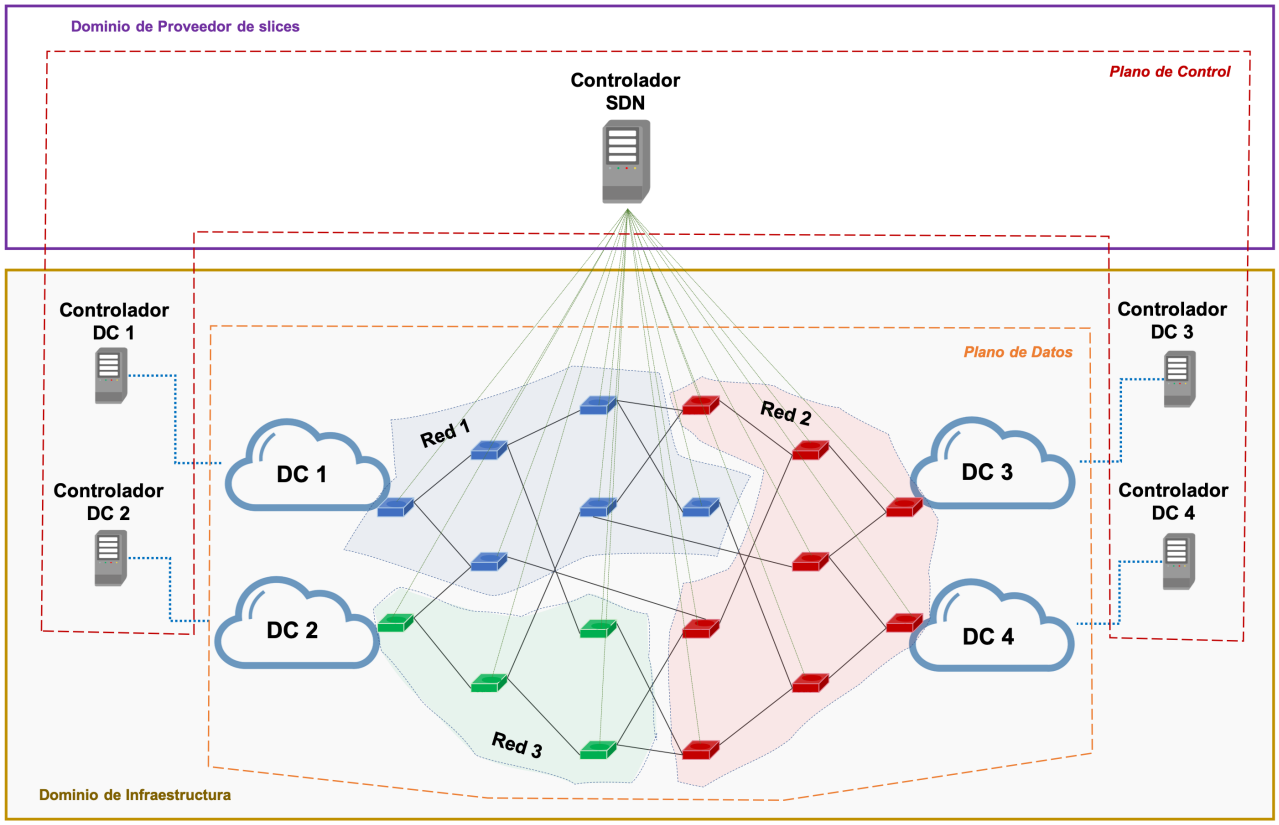

Figura 5.4: Topología para la plataforma del Proveedor de slices, donde tiene a disposición cuatro centros de datos y tres redes de transporte.

Como se puede observar, se tienen tres redes de transporte y cuatro centros de datos conectados

entre sí, teniendo switches OpenFlow en lugar de enrutadores en las redes para una gestión de reglas eficiente.

Para los centros de datos se utilizó una variante del *DC Slice Controller* de NECOS [12] para crear y configurar los servidores requeridos por el SP con contenedores Docker [55]. Este controlador está empaquetado en un contenedor y recibe peticiones del Proveedor de slices mediante una aplicación REST. El controlador tiene a su disposición servidores de virtualización Xen [56], en donde se crean máquinas virtuales para alojar los contenedores requeridos por el servicio.

Para la creación de las redes se utilizó Open vSwitch [57], una herramienta que permite crear switches OpenFlow virtuales. Dado que en la topología se necesitan varios switches fue necesario distribuir los switches virtuales en diferentes computadoras y, para crear enlaces que conectaran los switches alojados en diferentes computadoras se crearon túneles con el protocolo *Generic Routing Encapsulation* (GRE), asignando un ancho de banda de 20Gbps a cada túnel.

5.3 Resultados

A continuación se muestran los resultados obtenidos de las pruebas preparadas para validar el correcto funcionamiento de la arquitectura propuesta en los procesos de subscripción, invocación y control de QoS. Los servicios revisados en la sección anterior fueron utilizados mediante despliegue en contenedores a solicitud de SPs con diferentes membresías, las cuales dictan la prioridad a los servicios.

5.3.1 Proceso de subscripción

En esta prueba se valida el objetivo del proceso de subscripción, el cuál es maximizar el número de subscripciones de slices cuyos requerimientos de SLA puedan satisfacerse. Para lograrlo, se sigue el proceso de subscripción descrito en la sección anterior, de forma que se intenta encontrar una ruta disponible entre los centros de datos con la capacidad suficiente para cumplir los requerimientos mínimos del slice solicitante. Primero se utiliza el algoritmo de Dijkstra para tal objetivo, tomando en cuenta el grafo de la red y la última configuración encontrada en el módulo de subscripción para trazar un nuevo grafo y aplicar el algoritmo sobre este. En caso que no sea posible ubicar el slice, se ejecuta el AGC para generar configuraciones aleatorias con los slices suscritos y el solicitante, validando cada configuración y maximizando la cantidad de slices colocados. El AGC terminará cuando se encuentre una configuración con la misma cantidad de slices colocados que los slices actualmente subscritos y el solicitante, en caso de no cumplirse hasta el final de las generaciones, se asume que no puede ser posible colocarlo y el slice es rechazado.

Para realizar esta prueba se generó una traza con 100 solicitudes, donde cada registro contiene un SLA de un SP para solicitar su subscripción. Los valores de esta traza fueron generados de manera aleatorioa, donde se obtubieron 34 solicitudes para de servicio multimedia, 29 solicitudes para servicios de redes vehiculares y 37 solicitudes para servicios de IoT. El tiempo de interarribo de cada solicitud está definido por la distribución de Poisson, con una media de 30 segundos. Estos valores se definieron de manera experimental en la búsqueda de una configuración de prueba que permitiera estresar los módulos de la solución propuesta y analizar sus capacidades de gestión de reglas para aprovisionamiento y control de servicios entre centros de datos bajo condiciones de saturación de solicitudes. Para efectos de validación estadística, se repitió la prueba 50 veces con la misma traza, cambiando la semilla del generador de números aleatorios, que impacta el desempeño

del AGC.

La Figura 5.5 muestra la cantidad de slices que fueron subscritos en cada repetición, donde **a)** muestra los totales de subscripciones de todas las solicitudes, observándose que el mínimo alcanzado fue de 24 subscripciones y un máximo de 30. En **b)** se observa la cantidad de slices definidas mediante los algoritmos de Dijkstra y AGC, así como la cantidad de slices rechazados de la totalidad de solicitudes. Las gráficas **c)** y **d)** muestran el mismo análisis pero concentrándose en las solicitudes del tipo de servicios multimedi, donde se observa que mediante Dijkstra se pudo subscribir un sólo slice en todas las pruebas, con excepción a una que logró subscribir dos. El mismo análisis se realiza con las gráficas **e)** y **f)** para el tipo de servicio de redes vehiculares, donde se pueden subscribir entre cinco y diez slices, donde Dijkstra soluciona su colocación entre tres y cuatro casos por prueba. Finalmente, en el par de gráficas **g)** y **h)** se observan los resultados de subscripción para el tipo de servicio de IoT, donde se muestra que en más del 60 % de los casos las subscripciones se solucionaron utilizando el AGC. Al comparar los resultados entre los tipos de servicios se observa una mayoría de subscripciones del tipo de servicio IoT (**g)**), dado que es el servicio que menores requerimientos demanda y es más probable encontrar recursos disponibles para colocar slices al hacer una reconfiguración aún cuando la red ha rechazado varias solicitudes de tipos de servicios más demandantes. Esto contrasta con el tipo de servicio multimedia (**c)**), donde la cantidad de slices subscritos está por debajo del resto de servicios.

Con los resultados de estas pruebas se comprueba que la solución propuesta es efectiva en encontrar soluciones basadas en una búsqueda simple (utilizando Dijkstra) en lapsos de tiempo reducidos, así como soluciones que necesitan búsquedas complejas basadas en inteligencia computacional (con el AGC). De igual manera, se comprueba la efectividad que tiene la solución en admitir las subscripciones para las que ciertamente existen recursos para poder satisfacer su calidad de servicio (QoS), el cual es un objetivo importante de este trabajo de investigación.

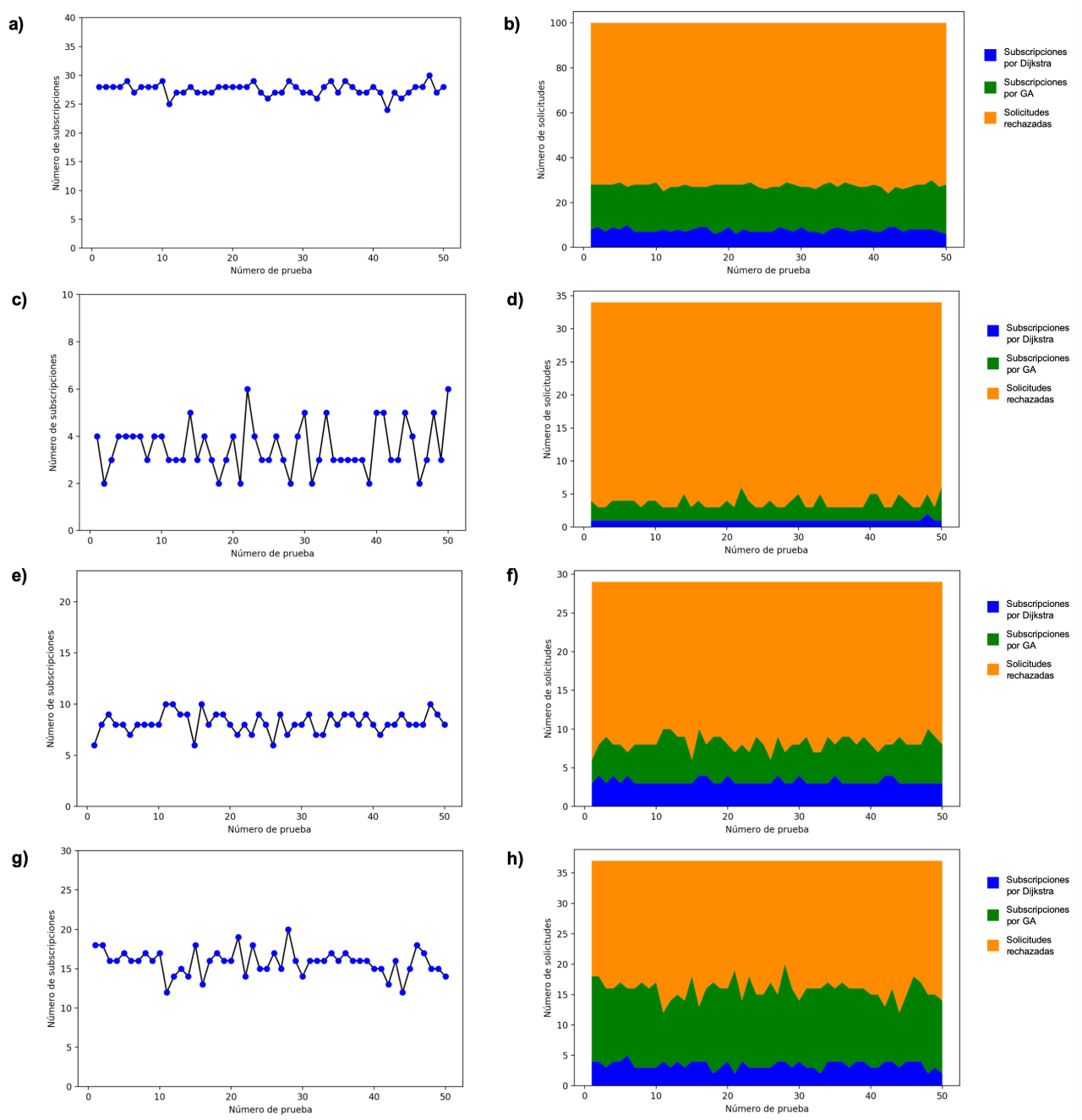

Figura 5.5: Cantidad de subscripciones en cada repetición del experimento, donde se muestra la cantidad total de subscripciones y se detallan las subscripciones encontradas por Dijkstra, por el GA y las solicitudes rechazadas.

Para medir el tiempo de espera de las solicitudes de subscripción se calcularon los promedios de cada solicitud. El proveedor de subscripciones funciona atendiendo una solicitud a la vez, creando una cola de solicitudes de tipo *first-in, first-out* para evitar conflictos en estimación de reparto de recursos al atender más de una solicitud a la vez.

La Figura 5.6 muestra los promedios de tiempo de espera de las solicitudes de subscripción, donde se observa que al inicio los tiempos de espera crecen en forma lineal y después cambia en forma exponencial, esto debido a las condiciones de saturación definidas en el escenario de prueba, en donde el máximo de subscripciones es de 30, por lo que conforme se incrementa ese número es menos probable encontrar recursos disponibles para un nuevo slice, debiendo ejecutar el AGC de forma completa por cada solicitud e incrementar el tamaño de la cola de espera.

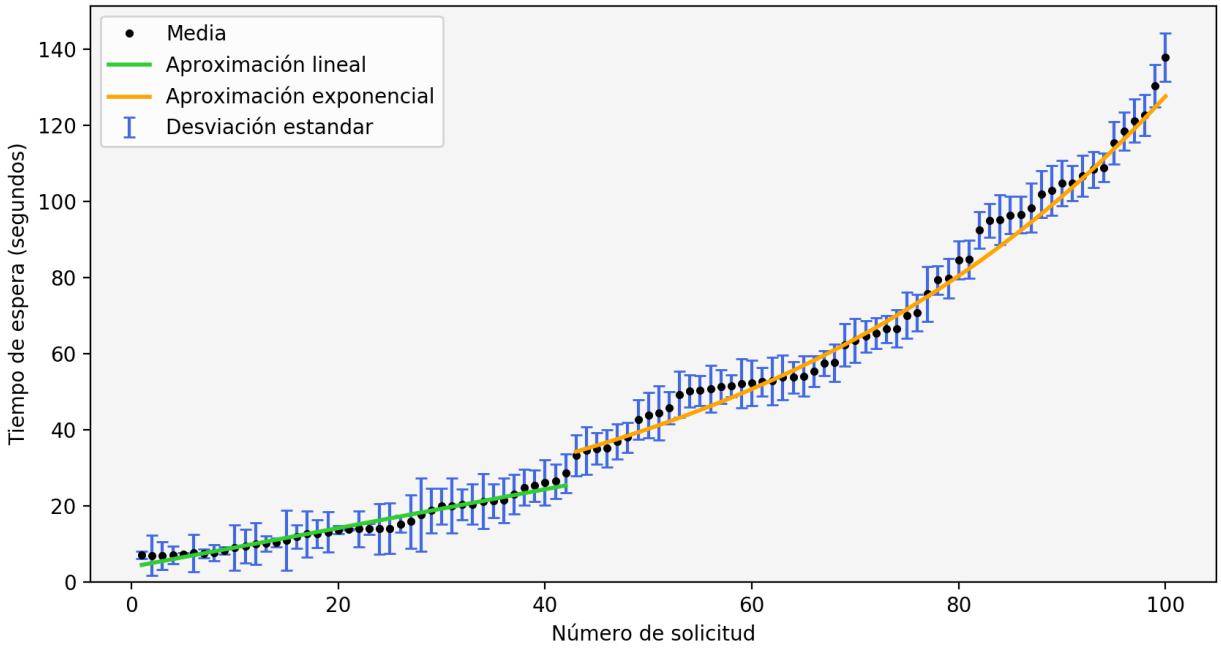

Figura 5.6: Promedio del tiempo de espera en cada solicitud. Se identifica el momento en que los tiempos de espera cambian de forma lineal a exponencial.

De acuerdo a los resultados observados del proceso de subscripción, el Proveedor de Slices de la solución propuesta es efectivo en garantizar la QoS del slice solicitante, garantizando que los slices subscritos no experimenten degradaciones de calidad de servicio en el futuro (i.e. durante su invocación). Es decir, en el caso que los slices subscritos invoquen sus servicios en su totalidad, los slices aceptados disfrutan de los servicios sin ninguna degradación. La solución también es eficiente en rechazar solicitudes de subscripción para los cuales no existen recursos, las cuales en caso de aceptarse, causarían degradación en los slices subscritos, y en particular aquellos para los cuales se han invocado sus servicios.

5.3.2 Proceso de invocación

El proceso de invocación de un slice (descrito en el Capítulo 4) consiste en encontrar una configuración de red que cumpla con los requerimientos del SLA del slice invocado y también de los slices actualmente en ejecución. El Proveedor de slices de la solución propuesta cuenta con tres estrategias de colocación del slice, las cuáles son basadas en tres tipos de estrategias de búsqueda: búsqueda voraz, un clasificador bayesiano y bosques aleatorios.

Para analizar los tiempos de obtención de configuraciones e instalación de reglas de flujos se identificó la semilla del generador de números aleatorios que permitió al módulo de subscripción aceptar el mayor número de subscripciones y se repitió 50 veces la prueba con la misma semilla.

La Figura 5.7 muestra el promedio del tiempo de obtención de configuración para satisfacer los requerimientos de SLA de los slices, donde se muestra que, a excepción de las primeras seis solicitudes, la estrategia de búsqueda voraz incrementa los tiempos de obtención de ruta de manera exponencial, mientras que los tiempos de los clasificadores responden de manera casi constante.

El comportamiento de las tres estrategias se debe a que, con una cantidad reducida de slices en búsqueda, el algoritmo voraz recorre una cantidad reducida de opciones, pero a medida de que la cantidad de slices a buscar aumenta, el problema de búsqueda se vuelve más complejo, ya que se revisan las configuraciones almacendas en forma secuencial hasta encontrar coincidencias. Por parte de los clasificadores, el tiempo de obtención de configuración se mantiene constante, ya que realizan la misma cantidad de operaciones sin importar la cantidad de slices involucrados en la búsqueda. Los tiempos obtenidos por esta gráfica sugieren la utilización de una búsqueda voraz al tener una cantidad de slices en ejecución baja para evitar retrasos en la configuración del plano de datos, pero cambiar a una estrategia de clasificación cuando la cantidad de restricciones aumente (e.g. cantidad de slices ejecutándose, cantidad de enlaces ocupados).

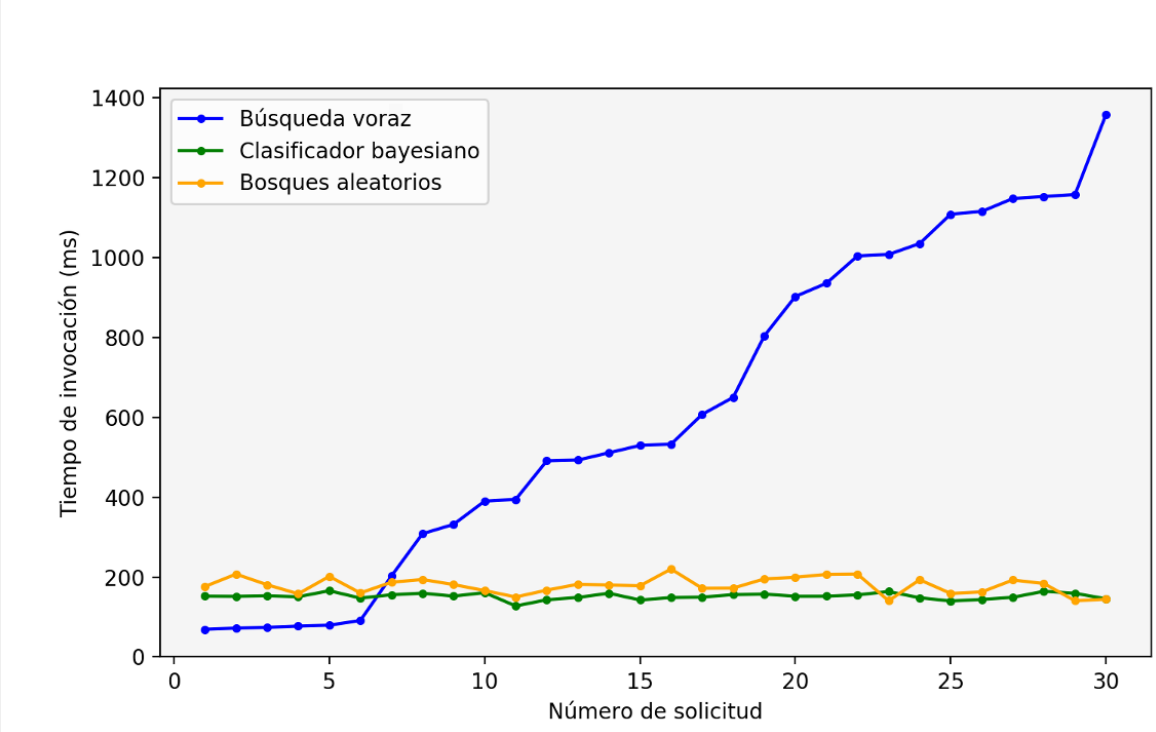

Figura 5.7: Promedio del tiempo de obtención de configuración de reglas de flujos que cumplen los requerimientos del slice invocado y los slices en ejecución.

Un dato importante de analizar es el tiempo de configuración del plano de datos durante la invocación de un slice, lo cual consiste en hacer la eliminación e instalación de reglas de flujos en los switches OpenFlow de acuerdo a la configuración obtenida. Entre más diferencia exista entre la configuración actual y la configuración obtenida, mayor será la cantidad de cambios en el plano de datos, por lo que tomará más tiempo reconfigurar las reglas de flujos para mantener un cumplimiento aceptable en el SLA de los slices.

Las Figuras 5.8 muestra en **a)**, **b)** y **c)** los tiempos promedio de eliminación e instalación de reglas de flujos por las estrategias de búsqueda voraz, clasificador bayesiano y bosques aleatorios respectivamente, para establecer la configuración de red obtenida. Se observa que con las estrategias de búsqueda voraz y bosques aleatorios los tiempos tienen un comportamiento lineal, mientras que con el clasificador bayesiano se observa un crecimiento exponencial, lo cual puede deberse a que el clasificador bayesiano encuentra configuraciones que requieren más cambios en la red en la última parte de las subscripciones debido a grandes diferencias entre la configuración actual y la configuración obtenida. En **d)** se comparan los tiempos de configuración totales (instalación y eliminación de reglas de flujos) mostrando una clara desventaja de la búsqueda voraz, mientras que los clasificadores tienen tiempos similares, siendo ligeramente mejores los tiempos de la estrategia de bosques aleatorios.

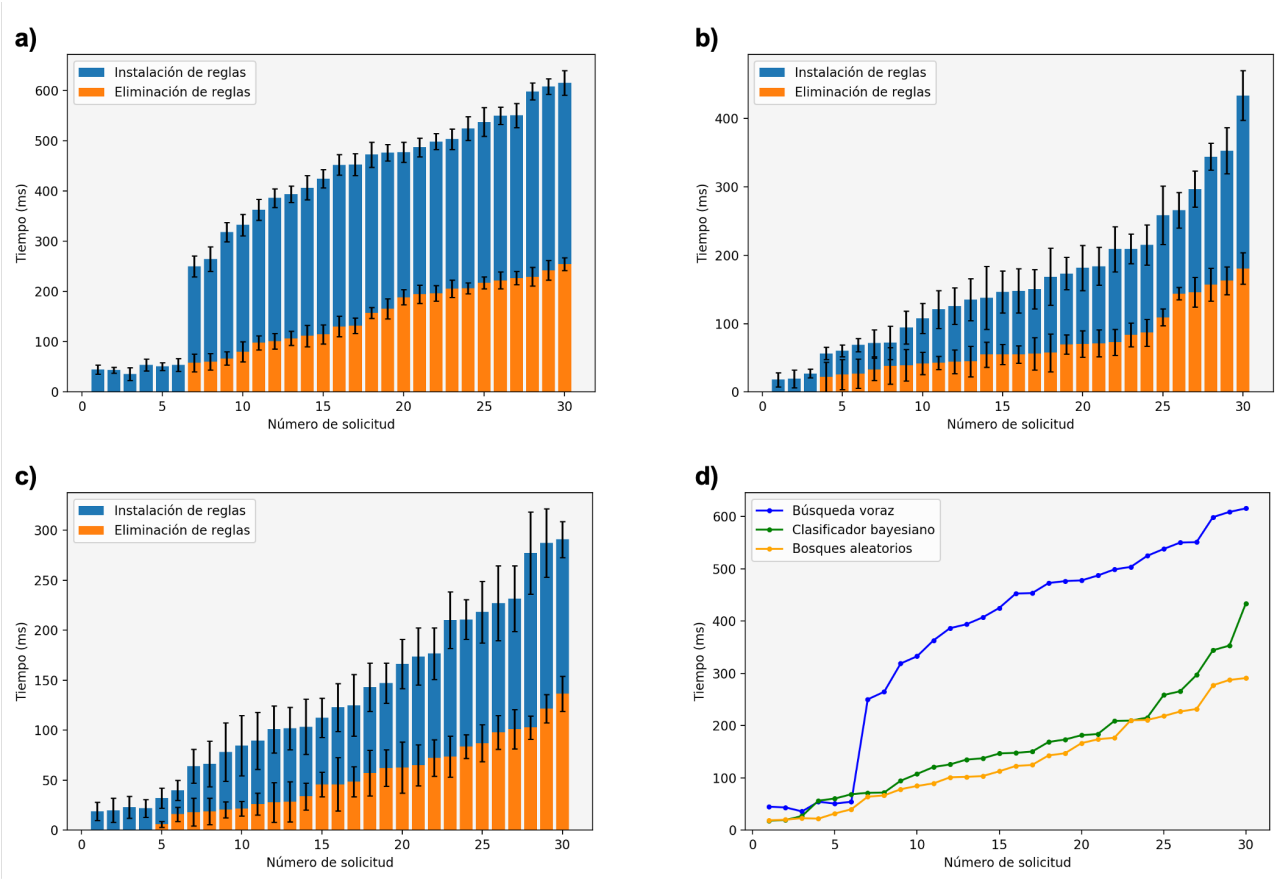

Figura 5.8: Tiempos promedio de instalación y eliminación de reglas de flujos para las estrategias **a)** búsqueda voraz, **b)** clasificador bayesiano y **c)** bosques aleatorios. En **d)** se muestra la comparación del total de tiempo de eliminación e instalación de reglas de flujos.

Los resultados del proceso de invocación muestran que el Proveedor de Slices es efectivo en utilizar técnicas simples (como una búsqueda voraz) y técnicas de aprendizaje máquina (como clasificadores) para encontrar configuraciones en el plano de datos que permitan cumplir los requerimientos de QoS de los slices, siendo este módulo de invocación flexible para implementar otras estrategias. Los tiempos de respuesta, tanto para obtención de configuraciones como instalación de reglas, sugieren la utilización de estrategias de clasificación para situaciones de saturación de la infraestructura subyacente, aunque es podible utilizar estrategias simples para situaciones de sub-utilización que

reaccionen dentro de rangos de tiempo aceptables para la configuración del plano de datos.

5.3.3 Proceso de gestión de QoS basada en elasticidad

El proceso de gestión de QoS consiste en la generación dinámica de reglas de flujos para el plano de datos. En la sección 4.1.2.3 se describen ejemplos de reglas de flujos y de la solución ante conflictos entre reglas, las cuáles se generar de forma intencionada para mantener funcionando el tráfico de datos en tiempo de reconfiguración. La instalación de reglas se realiza de manera dinámica de acuerdo a la utilización de los slices. En la Figura 5.9 se muestra un ejemplo de una situación que requiere reconfigurar la ruta de un slice debido a las necesidades de ancho de banda de otro slice, donde pasará de utilizar la ruta marcada en **a)** para utilizar la ruta marcada en **b)**.

La instalación de reglas se realiza de forma dinámica, mientras que la desinstalación se realiza de forma periódica para evitar pérdida de paquetes en tiempo de reconfiguración. La Figura 5.10 muestra las nuevas reglas generadas por la aplicación *Topología, monitoreo y gestión de reglas* al obtener la configuración del Proveedor de Slices, así como las reglas previamente funcionando para el slice que serán eliminadas después de un periodo de tiempo. Estas reglas dictan el tratamiento que se les dará a los paquetes de los flujos entre los servidores del slice, mientras que el ajuste de ancho de banda se realiza mediante la creación de colas mediante el método jerárquico de *Token Bucket* (HTB, por las siglas en inglés de *Hierarchical Token Bucket*) y reglas de reserva de ancho de banda permitidas por los switches virtuales *Open vSwitch*.

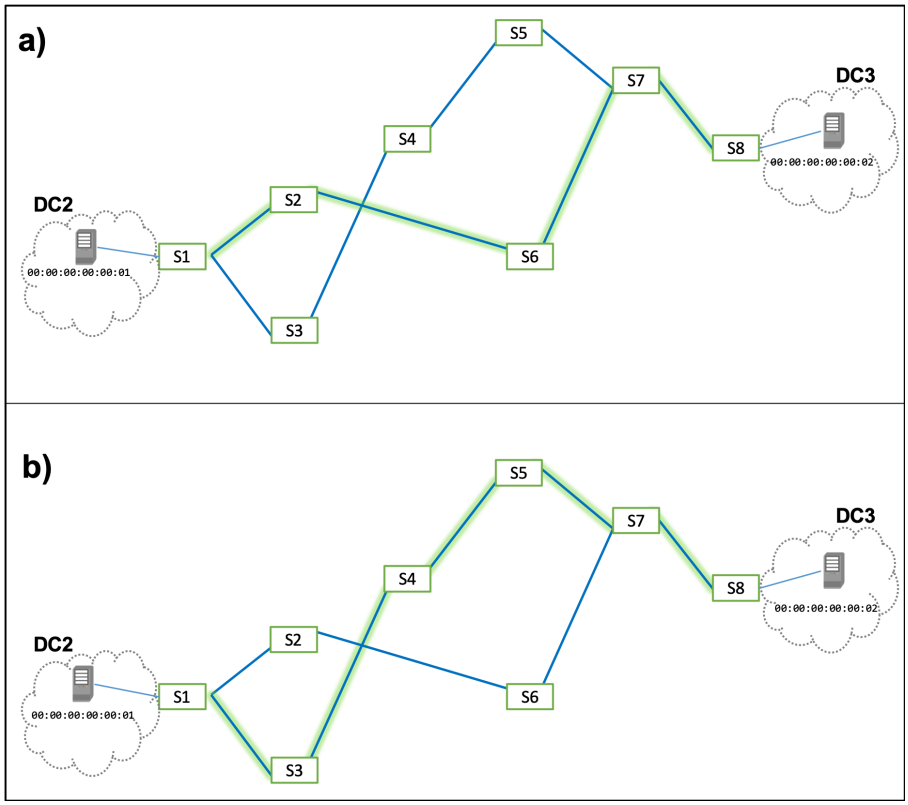

Figura 5.9: Ejemplo de cambio de ruta durante gestión de QoS. En **a)** se resalta la ruta actual en un slice, la cuál puede cambiar debido a que es necesario aplicar mayor ancho de banda y alguno de los enlaces no pueden asegurar más recursos. En **b)** se resaltauna nueva ruta para configurar el slice, donde los enlaces que lo componen soportan el nuevo requerimiento de ancho de banda.

Reglas de flujos a instalar


```
{“datapath”: 1, "priority": 8, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 3}}
{“datapath”: 1, "priority": 8, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
{“datapath”: 3, "priority": 8, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 2}}
{“datapath”: 3, "priority": 8, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
{“datapath": 4, "priority": 8, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 2}}
{“datapath": 4, "priority": 8, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
{“datapath": 5, "priority": 8, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 2}}
{“datapath": 5, "priority": 8, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
{“datapath": 7, "priority": 8, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 2}}
{“datapath": 7, "priority": 8, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
{“datapath": 8, "priority": 8, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 3}}
{“datapath": 8, "priority": 8, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
```


Reglas de flujos a eliminar


```
{"datapath": 1, "priority": 7, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 2}}
{"datapath": 1, "priority": 7, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
{"datapath": 2, "priority": 7, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 2}}
{"datapath": 2, "priority": 7, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
{"datapath": 6, "priority": 7, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 2}}
{"datapath": 6, "priority": 7, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
{"datapath": 7, "priority": 7, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 3}}
{"datapath": 7, "priority": 7, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 2}}
{"datapath": 8, "priority": 7, "match": {"eth_src": "00:00:00:00:00:01", "eth_dst": "00:00:00:00:00:02"}, "actions": {"out_port": 3}}
{"datapath": 8, "priority": 7, "match": {"eth_src": "00:00:00:00:00:02", "eth_dst": "00:00:00:00:00:01"}, "actions": {"out_port": 1}}
```


Figura 5.10: Reglas de flujos a instalar y eliminar del plano de datos.

Dado que el Proveedor de Slices tiene el compromiso de satisfacer los requerimientos de los slices, incluyendo situaciones en donde sean requeridos más o menos recursos, esta prueba permite demostrar la efectividad del control de la QoS de slices mediante elasticidad. Tal como se menciona en la sección 2.4, la elasticidad es la característica de un sistema para ajustar su desempeño mediante adición o remoción de recursos de acuerdo a la demanda sin la necesidad de interrumpir los servicios. Cuando un slice consume ancho de banda sobre o bajo un umbral durante un lapso de tiempo, será necesario hacer crecer o reducir los recursos asignados al slice de acuerdo a los parámetros con los que fue subscrito.

Los slices están diferenciados con un tipo de prioridad, por lo que aquellos slices con prioridades altas serán los primeros en satisfacerse por elasticidad o mantenerse con altos recursos. Para esta prueba se invocaron diez slices para validar el control de QoS del Proveedor de Slices. La Figura 5.11 muestra la utilización de ancho de banda y los eventos de elasticidad realizados para reconfigurar el

plano de datos y mantener los requerimientos de QoS de los slices entre los umbrales aceptables. Cuando la utilización instantánea de ancho de banda se encuentra por debajo del 10 % del establecido durante diez segundos se dispara el evento de decremento de recursos por elasticidad, que consiste en reducir el parámetro de ancho de banda dentro de lo establecido en el SLA. Cuando la utilización de ancho de banda se encuentra por encima del 90 % del establecido durante diez segundos se dispara el evento de aumento de recursos por elasticidad, que consiste en aumentar el parámetro de ancho de banda dentro de lo establecido en el SLA.

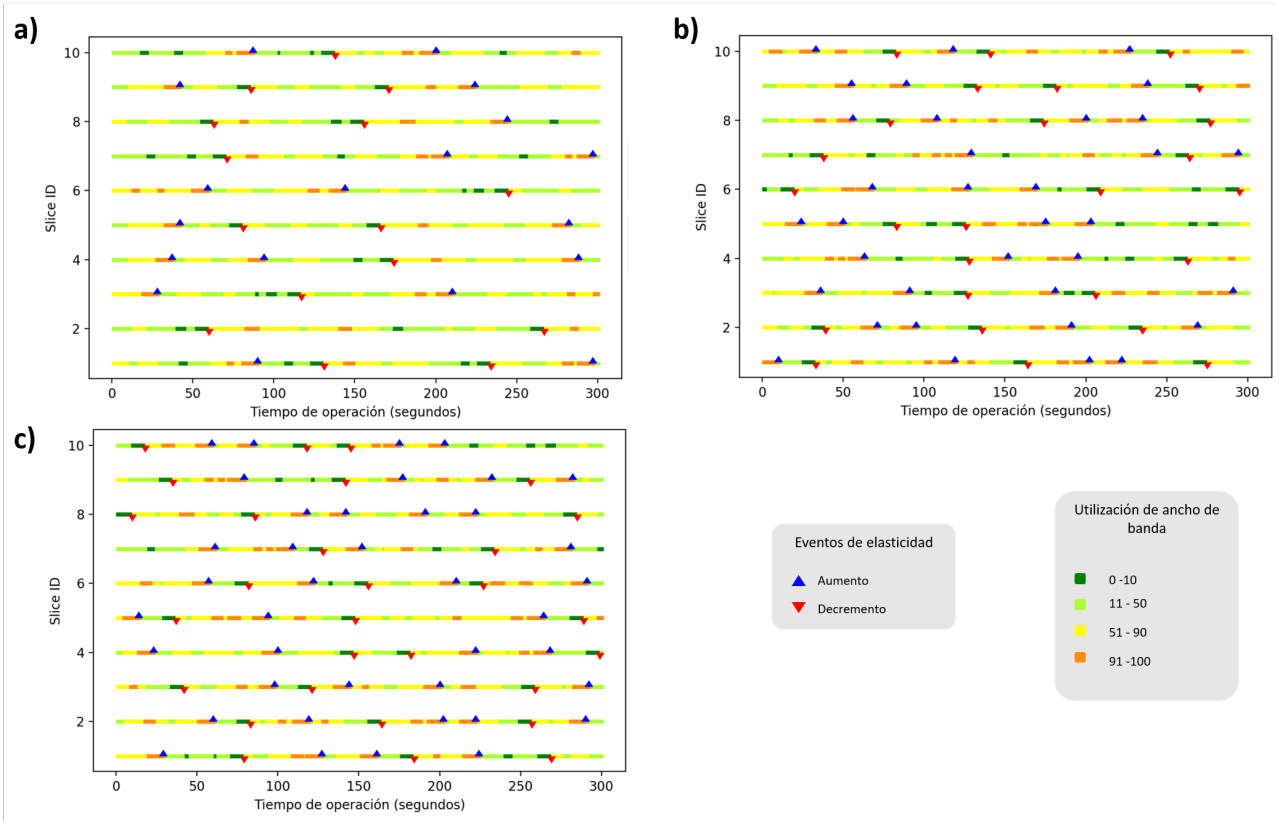

Figura 5.11: Utilización de ancho de banda y eventos de QoS mediante elasticidad, utilizando las estrategias de **a)** búsqueda voraz, **b)** clasificador bayesiano y **c)** bosques aleatorios.

La Tabla 5.2 muestra la cantidad de cambios debido a los eventos de elasticidad. Es de notar que, bajo la estrategia de búsqueda voraz se realiza casi la mitad de cambios en comparación con los

clasificadores, esto debido a que es una búsqueda fiel para encontrar exactamente los slices indicados, contrario a los clasificadores, que buscan similitudes con otros slices de requerimientos similares.

Tabla 5.2: Cambios durante la ejecución de slices debido al control de QoS mediante elasticidad.

Estrategia	Aumento de recursos por elasticidad	Decremento de recursos por elasticidad	Total de reconfiguraciones
Búsqueda voraz	18	15	33
Clasificador bayesiano	35	26	61
Bosques aleatorios	40	29	69

En la Figura 5.12 se muestra un ejemplo del mensaje de las llamadas que genera la solución propuesta para los eventos de aumento y reducción del ancho de banda de manera dinámica. Se muestran mensajes en pares, donde el primer mensaje establece una cola para tratar el tráfico del slice y en el segundo mensaje se asigna la cola al tráfico entre los servidores del slice. Se muestran dos pares de mensajes, donde el primero par es para configurar el ancho de banda de *switch1-switch2* y el segundo de *switch2-switch1*. Los mensajes para decremento son iguales, estableciendo el ancho de banda en los campos *min_rate* y *max_rate* (en el ejemplo se asigna el mismo valor, dado que quien gestionará el ancho de banda es el Proveedor de slices), en este ejemplo el aumento establece 1Mbps, mientras que el decremento se realiza en un 50 % (500Kbps).

Mensajes para aumento de recursos

{“port_name”: “3”, “type”: “linux-htb”, “queues”: [{“max_rate”: “1000000”}, {“min_rate”: “1000000”}]}
{“datapath”: 1, “priority”: 8, “match”: {“eth_src”: “00:00:00:00:00:01”, “eth_dst”: “00:00:00:00:00:02”}, “actions”: {“queue”: 5}}

{“port_name”: “1”, “type”: “linux-htb”, “queues”: [{“max_rate”: “1000000”}, {“min_rate”: “1000000”}]}
{“datapath”: 1, “priority”: 8, “match”: {“eth_src”: “00:00:00:00:00:02”, “eth_dst”: “00:00:00:00:00:01”}, “actions”: {“queue”: 6}}

Mensajes para reducción de recursos

{“port_name”: “3”, “type”: “linux-htb”, “queues”: [{“max_rate”: “500000”}, {“min_rate”: “500000”}]}
{“datapath”: 1, “priority”: 8, “match”: {“eth_src”: “00:00:00:00:00:01”, “eth_dst”: “00:00:00:00:00:02”}, “actions”: {“queue”: 5}}

{“port_name”: “1”, “type”: “linux-htb”, “queues”: [{“max_rate”: “500000”}, {“min_rate”: “500000”}]}
{“datapath”: 1, “priority”: 8, “match”: {“eth_src”: “00:00:00:00:00:02”, “eth_dst”: “00:00:00:00:00:01”}, “actions”: {“queue”: 6}}

Figura 5.12: Ejemplo de los mensajes para aumentar y decrementar el ancho de banda del slice para un switch OpenFlow.

5.4 Conclusiones parciales

Con los resultados de estas pruebas se comprueba que el Proveedor de Slices de la solución propuesta es efectivo para encontrar soluciones que permitan maximizar la cantidad de subscripciones garantizando la calidad de los servicios que se aceptan. Para ello la solución ha sido validada con la utilización de un algoritmo genético que genera soluciones en busca de encontrar recursos para slices solicitantes en tiempo reducido. Para el proceso de invocación se comprobó la flexibilidad de la arquitectura para adaptarse a diversos mecanismos de búsqueda de configuraciones simples o inteligente por su diseño en microservicios. Para la orquestación de QoS se comprueba que la gestión de reglas de flujos es efectiva para controlar la elasticidad de los servicios provistos a los centros de datos, mediante la manipulación del plano de datos de las redes de transporte en situaciones donde el servicio requiera más o menos recursos.

La arquitectura de la solución propuesta ha sido implementada para ser flexible en incluir diversas estrategias para búsqueda de soluciones, y las que han sido probadas en este trabajo de tesis

han demostrado ser eficientes en diferentes etapas, tales como el control de admisión mediante subscripción utilizando un algoritmo simple como Dijkstra y un algoritmo inteligente como el AGC, y técnicas de clasificación en las etapas de invocación y control de la calidad de servicio mediante un clasificador bayesiano y clasificación con bosques aleatorios.

6

Conclusiones

En este capítulo se describe la contribución principal de este trabajo, así como las limitaciones del mismo, las dificultades afrontadas durante su desarrollo y el trabajo a futuro sobre aprovisionamiento de slices entre centros de datos.

6.1 Contribuciones

La evolución de los servicios en Internet requiere de mejores plataformas de automatización para aprovisionamiento de recursos y despliegue de servicios en zonas geofráficas estratégicas. Este trabajo de tesis abordó el desafío de gestionar infraestructuras multidominio para aprovisionar, configurar y

gestionar cloud slices y su calidad de servicio mediante la gestión dinámica de reglas de flujos en redes definidas por software. Con la elaboración y validación de esta plataforma se ha contribuido al estado del arte con una arquitectura basada en microservicios para gestión dinámica de reglas de flujos en redes definidas por software para controlar la calidad de servicio de slices entre centros de datos.

6.2 Dificultades

Aunque durante el desarrollo de este trabajo hubo varias facilidades, como el aporte y contribución con el proyecto pionero en cloud slicing, NECOS, también hubo dificultades técnicas, por ejemplo, la imposibilidad de experimentar cloud slicing completo debido a que no hay posibilidad, en la actualidad, de desplegar redes definidas por software entre centros de datos intercontinentales, ya que sería necesario hacer uso de Internet, perdiendo el control de algunos dispositivos en el plano de datos. Por este motivo se optó por hacer una implementación emulada de redes de transporte bajo el enfoque SDN utilizando virtualización. Otra de las dificultades fue el poco trabajo en cloud slicing reportado en la literatura, si bien en network slicing hay bastantes trabajos destacados, el cloud slicing sólo ha sido revisado por un grupo de trabajo, encontrando difícil la posibilidad de comparar la propuesta presentada con la única plataforma disponible en la literatura, ya que esta última no aborda la totalidad del slice en su implementación disponible.

6.3 Limitaciones

El Proveedor de Slices multi-cloud aquí presentado se ha validado como una plataforma autoconfigurable para lograr objetivos de alto nivel, como satisfacer los requerimientos de calidad de servicio mediante el constante monitoreo del plano de datos y de los slices, resultando efectivo con las estrategias utilizadas, las cuales pueden ser cambiadas para mejorar el desempeño de los procesos de subscripción, invocación y control de la calidad de servicio mediante la implementación de mecanismos de elasticidad de recursos. Sin embargo, el Proveedor de Slices propuesto es una prueba de concepto para gestionar el control de la QoS de slices multi-cloud, el cuál no ha sido probado en escenarios a gran escala, para lo cuál será necesario utilizar varios controladores SDN en modo de cooperación para mantener centralizado lógicamente el plano de datos.

6.4 Trabajo futuro

Como trabajo futuro se espera abordar problemas de redes de cloud slicing a gran escala para provisión de servicios de precisión, tomando en consideración casos de uso con tecnologías más allá de 5G (del inglés Beyond 5G) mediante técnicas de monitoreo para aumentar la consciencia del estado de la red por parte del proveedor de slices y automatizar el control de las configuraciones de las infraestructuras de red y de los centros de datos, haciendo uso de grandes volúmenes de información.

Bibliografía

[1] Open Networking Foundation. OpenFlow Switch Specification. Version 1.5.1 (Protocol version 0x06). Specification ONF TS-025, Open Networking Foundation, marzo de 2015. URL `https://www.opennetworking.org`. En línea en https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[2] B. Chatras, U. S. Tsang Kwong, and N. Bihannic. NFV enabling network slicing for 5G. *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. pp. 219, marzo de 2017. doi: 10.1109/ICIN.2017.7899415, ISSN: 2472-8144.

[3] Javier Rubio-Loyola, Marinos Charalambides, Issam Aib, Joan Serrat, George Pavlou, and Raouf Boutaba. Business-driven Management of Differentiated Services. *2010 IEEE Network Operations and Management Symposium - NOMS 2010*. pp. 240-247, 17 de junio de 2010. doi: 10.1109/NOMS.2010.5488468, ISBN: 978-1-4244-5367-2.

[4] Stuart Clayman and Alex Galis. D3.2: NECOS System Architecture and Platform Specification. V2. *Novel Enablers for Cloud Slicing - Deliverables*. pp. 37, abril de 2019.

[5] Christopher Hooton. Measuring The U.S. Internet Sector: 2019. Report 2019, Internet Association, septiembre de 2019. URL `https://internetassociation.org`. En línea en https://internetassociation.org/wp-content/uploads/2019/09/I_Measuring-The-US-Internet-Sector-2019.pdf.

[6] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*. pp. 637-646, octubre de 2016. doi: 10.1109/JIOT.2016.2579198, ISSN: 2327-4662.

[7] Weisong Shi and Schahram Dustdar. The Promise of Edge Computing. *Computer*. Vol. 49, no. 5, pp. 78-81, mayo de 2016. doi: 10.1109/MC.2016.145, ISSN: 1558-0814.

[8] Mirza Golam Kibria, Kien Nguyen, Gabriel Porto Villardi, Ou Zhao, Kentaro Ishizu, and Fumihide Kojima. Big Data Analytics, Machine Learning, and Artificial Intelligence in Next-Generation Wireless Networks. *IEEE Access*. Vol. 6, pp. 32328-32338, mayo de 2018. doi: 10.1109/ACCESS.2018.2837692, ISSN: 2169-3536.

[9] Rongpeng Li, Zhifeng Zhao, Qi Sun, Chih-Lin I, Chenyang Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang. Deep Reinforcement Learning for Resource Management in Network Slicing. *IEEE Access*. Vol. 6, pp. 74429-74441, noviembre de 2018. doi: 10.1109/ACCESS.2018.2881964, ISSN: 2169-3536.

[10] Bin Han, Ji Lianghai, and Hans D. Schotten. Slice as an Evolutionary Service: Genetic Optimization for Inter-Slice Resource Management in 5G Networks. *IEEE Access*, . Vol. 6, pp. 33137-33147, junio de 2018. doi: 10.1109/ACCESS.2018.2846543, ISSN: 2169-3536.

[11] F. S. D. Silva, M. O. O. Lemos, A. Medeiros, A. V. Neto, R. Pasquini, D. Moura, C. Rothenberg, L. Mamatas, S. L. Correa, K. V. Cardoso, C. Marcondes, A. ABelem, M. Nascimento, A. Galis, L. Contreras, J. Serrat, and P. Papadimitriou. NECOS Project: Towards Lightweight Slicing of Cloud Federated Infrastructures. *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. pp. 406-414, junio de 2018. doi: 10.1109/NETSOFT.2018.8460008, ISBN: 978-1-5386-4633-5.

[12] NECOS - MUlti-Slice/Tenant/Service Demo. `https://gitlab.com/necos/demos/musts`. Consultado el 2020-01-10.

[13] Asma Islam Swapna, Raphael Vicente Rosa, Christian Esteve Rothenberg, Ilias Sakellariou, Lefteris Mamatas, and Panagiotis Papadimitriou. Towards A Marketplace for Multi-domain Cloud Network Slicing: Use Cases. *2019 ACM/IEEE Symposium on Architectures for*

Networking and Communications Systems (ANCS). pp. 1-4, septiembre de 2019. doi: 10.1109/ANCS.2019.8901876, ISBN: 978-1-7281-4387-3.

[14] Polychronis Valsamas, Sotiris Skaperas, George Violettas, T. Theodorou, S. Petridou, D. Vardalis, A. Tsioukas, and Lefteris Mamatas. Experimenting with Cloud and Network Orchestration for Multi-Access Edge Computing. *IEEE Wireless Communications and Networking Conference*. pp. 1-2, febrero de 2019.

[15] Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys & Tutorials*. Vol. 16, no. 3, pp. 1617-1634, febrero de 2014. doi: 10.1109/SURV.2014.012214.00180, ISSN: 1553-877X.

[16] Samaresh Bera, Sudip Misra, and Athanasios V. Vasilakos. Software-Defined Networking for Internet of Things: A Survey. *IEEE Internet of Things Journal*. Vol. 4, pp. 1994-2008, diciembre de 2017. doi: 10.1109/JIOT.2017.2746186, ISSN: 2327-4662.

[17] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*. Vol. 103, no. 1, pp. 14-76, enero de 2015. doi: 10.1109/JPROC.2014.2371999, ISSN: 1558-2256.

[18] Celio Trois, Marcos D. Del Fabro, Luis C. E. de Bona, and Magnos Martinello. A Survey on SDN Programming Languages: Toward a Taxonomy. *IEEE Communications Surveys & Tutorials*. Vol. 18, no. 4, pp. 2687-2712, abril de 2016. doi: 10.1109/COMST.2016.2553778, ISSN: 1553-877X.

[19] Anja Strunk. QoS-Aware Service Composition: A Survey. *2010 Eighth IEEE European Conference on Web Services*. pp. 67-74, 20 de enero de 2011. doi: 10.1109/ECOWS.2010.16, ISBN: 978-1-4244-9397-5.

[20] Cristian Cleder Machado, Lisandro Zambenedetti Granville, Alberto Schaeffer-Filho, and Juliano Araujo Wickboldt. Towards SLA Policy Refinement for QoS Management in Software-Defined Networking. *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. pp. 397-404, 13 al 16 de mayo de 2014. doi: 10.1109/AINA.2014.148, ISSN: 1550-445X.

[21] Airton Ishimori, Fernando Farias, Eduardo Cerqueira, and Antônio Abelém. Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking. *2013 Second European Workshop on Software Defined Networks*. pp. 81-86, 12 de diciembre de 2013. doi: 10.1109/EWSDN.2013.20, ISBN: 978-1-4799-2433-2.

[22] Raphael Durner, Andreas Blenk, and Wolfgang Kellererm. Performance study of dynamic QoS management for OpenFlow-enabled SDN switches. *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*. pp. 177-182, 11 de febrero de 2016. doi: 10.1109/IWQoS.2015.7404730, ISBN: 978-1-4673-7113-1.

[23] Wendong Wang, Qinglei Qi, Xiangyang Gong, Yannan Hu, and Xirong Que. Autonomic QoS management mechanism in Software Defined Network. *China Communications*. pp. 13-23, 11 de septiembre de 2014. doi: 10.1109/CC.2014.6895381, ISSN: 1673-5447.

[24] Jochen W. Guck and Wolfgang Kellerer. Achieving end-to-end real-time Quality of Service with Software Defined Networking. *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. pp. 70-76, 1 de diciembre de 2014. doi: 10.1109/CloudNet.2014.6968971, ISBN: 978-1-4799-2730-2.

[25] Slavica Tomovic, Neeli Prasad, and Igor Radusinovic. SDN control framework for QoS provisioning. *2014 22nd Telecommunications Forum Telfor (TELFOR)*. pp. 111-114, noviembre de 2014. doi: 10.1109/TELFOR.2014.7034369, ISBN: 978-1-4799-6191-7.

[26] C. N. Sminesh, E. Grace Mary Kanaga, and K. Ranjitha. Flow Monitoring Scheme for Reducing Congestion and Packet Loss in Software Defined Networks. *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*. 5 páginas, enero de 2017. doi: 10.1109/ICACCS.2017.8014667, ISBN: 978-1-5090-4559-4.

[27] Shadi Attarha, Koosha Haji Hosseiny, Ghasem Mirjalily, and Kiarash Mizanian. A Load Balanced Congestion Aware Routing Mechanism for Software Defined Networks. *2017 Iranian Conference on Electrical Engineering (ICEE)*. pp. 2206-2210, mayo de 2017. doi: 10.1109/IranianCEE.2017.7985428, ISBN: 978-1-5090-5963-8.

[28] Seungbeom Song, Jaiyong Lee, Kyuho Son, Hangyong Jung, and Jihoon Lee. A Congestion Avoidance Algorithm in SDN Environment. *2016 International Conference on Information Networking (ICOIN)*. pp. 420-423, enero de 2016. doi: 10.1109/ICOIN.2016.7427148, ISBN: 978-1-5090-1724-9.

[29] Ming-Tsung Kao, Bo-Xiang Huang, Shang-Juh Kao, and Hsueh-Wen Tseng. An Effective Routing Mechanism for Link Congestion Avoidance in Software-Defined Networking . *2016 International Computer Symposium (ICS)*. pp. 154-158, diciembre de 2016. doi: 10.1109/ICS.2016.0039, ISBN: 978-1-5090-3438-3.

[30] You-Chiun Wang and Siang-Yu You. An Efficient Route Management Framework for Load Balance and Overhead Reduction in SDN-based Data Center Networks. *IEEE Transactions on Network and Service Management*. 13 páginas, 24 de septiembre de 2018. doi: 10.1109/TNSM.2018.2872054, ISSN: 1932-4537.

[31] Umme Zakia and Hanene Ben Yedder. Dynamic Load Balancing in SDN-Based Data Center Networks. *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. pp. 242-247, octubre de 2017. doi: 10.1109/IEMCON.2017.8117206, ISBN: 978-1-5386-3371-7.

[32] Anand V Akella and Kaiqi Xiong. Quality of Service (QoS) Guaranteed Network Resource Allocation via Software Defined Networking (SDN). *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*. pp. 7-13, octubre de 2012. doi: 10.1109/DASC.2014.11, ISBN: 978-1-4799-5079-9.

[33] A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, and G. Pavlou. Policy refinement for DiffServ quality of service management. *2005 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005. IM 2005.* pp. 469-482, 13 de junio de 2015. doi: 10.1109/INM.2005.1440817, ISBN: 0-7803-9087-3.

[34] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck. Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions. *IEEE Communications Surveys Tutorials*. pp. 2429-2453, 2018. doi: 10.1109/COMST.2018.2815638, ISBN: 1553-877X.

[35] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network Function Virtualization: Challenges and Opportunities for Innovations. *IEEE Communications Magazine*, . Vol. 53, no. 2, pp. 90-97, febrero de 2015. doi: 10.1109/MCOM.2015.7045396, ISSN: 1558-1896.

[36] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. *MCC '12 Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. pp. 13-15, agosto de 2012. doi: 10.1145/2342509.2342513, ISBN: 978-1-4503-1519-7.

[37] Adel Nadjaran Toosi, Redowan Mahmud, Qinghua Chi, and Rajkumar Buyya. Management and Orchestration of Network Slices in 5G, Fog, Edge and Clouds. *Fog and Edge Computing: Principles and Paradigms, Wiley Press*. Capítulo 4, 31 páginas, 2018. ISBN: 978-1-119-52498-4.

[38] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031, RFC Editor, enero de 2001. URL `https://tools.ietf.org/html/rfc3031`. En línea en https://tools.ietf.org/html/rfc3031.

[39] D. Black and P. Jones. Differentiated Services (Diffserv) and Real-Time Communication. RFC 7657, RFC Editor, noviembre de 2015. URL `https://tools.ietf.org/html/rfc7657`. En línea en https://tools.ietf.org/html/rfc7657.

[40] Flávio R.C. Sousa and Javam C. Machado. Towards Elastic Multi-Tenant Database Replication with Quality of Service. *2012 IEEE Fifth International Conference on Utility and Cloud Computing*. pp. 168-175, noviembre de 2012. doi: 10.1109/UCC.2012.36, ISBN: 978-1-4673-4432-6.

[41] Pragati Chavan, Pradeep Patil, Gurudatt Kulkarni, Ramesh Sutar, and Shrikant Belsare. IaaS Cloud Security. *2013 International Conference on Machine Intelligence and Research Advancement*. pp. 549-553, diciembre de 2013. doi: 10.1109/ICMIRA.2013.115, ISBN: 978-0-7695-5013-8.

[42] Michael Boniface, Bassem Nasser, Juri Papay, Stephen C. Phillips, Arturo Servin, Xiaoyu Yang, Zlatko Zlatev, Spyridon V. Gogouvitis, Gregory Katsaros, Kleopatra Konstanteli, George Kousiouris, Andreas Menychtas, and Dimosthenis Kyriazis. Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds. *2010 Fifth International Conference on Internet and Web Applications and Services*. pp. 155-160, enero de 2010. doi: 10.1109/ICIW.2010.91, ISBN: 978-1-4244-6729-7.

[43] Vidyanand Choudhary. Software as a Service: Implications for Investment in Software Development. *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. enero de 2007. doi: 10.1109/HICSS.2007.493, ISBN: 0-7695-2755-8.

[44] P. Trimintzios, I. Andrikopoulos, G. Pavlou, P. Flegkas, D. Griffin, P. Georgatsos, D. Goderis, Y. T'Joens, L. Georgiadis, C. Jacquenet, and R. Egan. A Management and Control Architecture for Providing IP Differentiated Services in MPLS-based Networks. *IEEE Communications Magazine*. pp. 80-88, mayo de 2001. doi: 10.1109/35.920861, ISSN: 1558-1896.

[45] P. Flegkas, P. Trimintzios, and G. Pavlou. A Policy-Based Quality of Service Management System for IP DiffServ Networks. *IEEE Network*. pp. 50-56, 7 de agosto de 2002. doi: 10.1109/65.993223, ISSN: 1558-156X.

[46] E. Mykoniati, C. Charalampous, P. Georgatsos, T. Damilatis, D. Goderis, P. Trimintzios, G. Pavlou, and D. Griffin. Admission Control for Providing QoS in DiffServ IP Networks: The TEQUILA Approach. *IEEE Communications Magazine*. pp. 38-44, 22 de enero de 2003. doi: 10.1109/MCOM.2003.1166652, ISSN: 1558-1896.

[47] Iris Bueno, José Ignacio Aznar, Eduard Escalona, Jordi Ferrer, and Joan Antoni García-Espín. An OpenNaaS Based SDN Framework for Dynamic QoS Control. *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. 7 páginas, noviembre de 2013. doi: 10.1109/SDN4FNS.2013.6702533.

[48] Md. Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba. PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks. *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. pp. 1-7, 9 de enero de 2014. doi: 10.1109/SDN4FNS.2013.6702548, ISBN: 978-1-4799-2781-4.

[49] Davide Adami, Gianni Antichi, Rosario G. Garroppo, Stefano Giordano, and Andrew W. Moore. Towards an SDN network control application for differentiated traffic routing. *2015 IEEE International Conference on Communications (ICC)*. pp. 5827-5832, junio de 2015. doi: 10.1109/ICC.2015.7249251, ISSN: 1938-1883.

[50] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*. Vol. 4, pp. 65-85, 1994. doi: 10.1007/BF00175354.

[51] NECOS - Demos. `https://gitlab.com/necos/demos`. Consultado el 2020-01-10.

[52] N. T. Jahromi, R. H. Glitho, A. Larabi, and R. Brunner. An NFV and microservice based

architecture for on-the-fly component provisioning in content delivery networks. *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*. pp. 1-7, enero de 2018. doi: 10.1109/CCNC.2018.8319227, ISBN: 978-1-5386-4790-5.

[53] dojot Soluções para IOT. Plataforma de Desenvolvimento para IOT. `http://www.dojot.com.br`. Consultado el 2020-01-10.

[54] Locust - A modern load testing framework. `https://locust.io/`. Consultado el 2020-01-10.

[55] Docker - Empowering App Development for Developers. `https://www.docker.com/`. Consultado el 2020-01-10.

[56] Xen Project. `https://xenproject.org`. Consultado el 2020-01-10.

[57] Open vSwitch. `https://www.openvswitch.org`. Consultado el 2020-01-10.