# Enrutamiento Inteligente y Consciente de la Calidad del Servicio en Redes Jerárquicas Definidas por Software


Trabajo de Grado

**Juan José Carvajal Barrios**
**Carlos Daniel Hernández Muñoz**

Director:
Oscar Mauricio Caicedo Rendón
Ph.D. en Ciencias de la Computación

Co-Director:
Carlos Felipe Estrada Solano
Ph.D. en Ciencias de la Computación

*Universidad del Cauca*
*Facultad de Ingeniería Electrónica y Telecomunicaciones*
*Departamento de Telemática*
*Popayán, Cauca, 2023*

# Intelligent and Quality of Service-Aware Routing in Hierarchical Software-Defined Networking


Trabajo de grado presentado a la Facultad de Ingeniería
Electrónica y Telecomunicaciones de la
Universidad del Cauca para obtener el título de:

Ingeniero en Electrónica y Telecomunicaciones

Juan José Carvajal Barrios
Carlos Daniel Hernández Muñoz

Director:
PhD. Oscar Mauricio Caicedo Rendón

Co-Director:
PhD. Carlos Felipe Estrada Solano

*Popayan*
*2023*

*Dedico este logro a mis padres quienes fueron mi más grande apoyo y motivación para cumplir con esta meta de mi vida.*
*A mi familia por siempre apoyarme incondicionalmente. A mi compañera de vida, por su amor y por estar a mi lado en los momentos difíciles.*
*A mis amigos por ayudarme y facilitarme el proceso para culminar esta etapa de mi vida. Gracias a todos de corazón.*
*Juan José Carvajal Barrios*

*A mis padres y hermana, quienes siempre han estado ahí con su apoyo incondicional. Agradezco sinceramente por enseñarme el genuino significado del esfuerzo y la persistencia para alcanzar nuestros sueños.*
*Asimismo, a mi círculo cercano de familiares y amigos por su respaldo y valiosos consejos durante esta fase crucial de mi vida. Su apoyo ha sido fundamental y valoro profundamente cada uno de sus gestos.*
*Carlos Daniel Hernández Muñoz*

# Agradecimientos

Expresamos un profundo y sincero agradecimiento a nuestros padres quienes brindaron un apoyo y acompañamiento incondicional a lo largo de esta meta, agradecemos la motivación que nos brindaron a seguir adelante en el día a día.

Agradecemos a nuestro director de trabajo de grado Ph.D. Oscar Caicedo y al co-director Ph.D. Carlos Estrada por su apoyo y dedicación en la orientación durante todo el trabajo de grado, exaltando su excelente labor en la enseñanza y dirección.

Agradecemos también a nuestros amigos que hicieron parte de esta etapa de nuestra vida, les deseamos buena salud, éxitos y fortuna en esta nueva etapa de nuestras vidas.

# Resumen

Actualmente, las redes desempeñan un papel crucial en la conectividad y comunicación entre dispositivos, servicios y usuarios. Sin embargo, el aumento exponencial de la demanda de datos y dispositivos conectados ha generado importantes desafíos para las redes tradicionales. Uno de estos problemas es el enrutamiento, ya que este proceso puede ser complicado y estático, lo que conduce a cuellos de botella, ineficiencias y tiempos de respuesta lentos. Por lo tanto, las Redes Definidas por Software surgen como una posible solución para abordar los desafíos del enrutamiento, al separar el plano de control del plano de datos, lo que permite un control más flexible y programable. No obstante, la arquitectura centralizada de las Redes Definidas por Software puede enfrentar problemas de escalabilidad y fiabilidad debido a la dependencia de un solo controlador, el cual se puede ver saturado por el aumento del flujo de información. Si este único controlador falla, toda la red quedaría incomunicada al no poder realizar enrutamiento. Debido a esto surgieron las arquitecturas de plano de control distribuidas.

En el presente trabajo de grado se propone un enfoque para abordar el enrutamiento en una arquitectura distribuida jerárquicamente en Redes Definidas por Software utilizando Aprendizaje por Refuerzo con múltiples agentes. Este enfoque tiene en cuenta la información del estado del enlace para tomar decisiones de enrutamiento basadas en las métricas de la calidad de servicio (*packet loss, delay* y *throughput*). Se empleó la interacción con el entorno, el Aprendizaje por Refuerzo y la visión global de la topología, en conjunto con el control de la red, para calcular e instalar las rutas óptimas para cada par origen-destino en los conmutadores. Los resultados muestran que este enfoque supera tanto a la variación de Dijkstra, basada en el *throughput*, como al algoritmo Q-learning, en términos de *packet loss, delay* y *throughput*.

# Contenido

# Lista de Figuras

# Lista de Tablas

# Lista de Algoritmos

# Acrónimos

| | | |
|---|---|---|
| IP | *Internet Protocol* | Protocolo de Internet |
| OSPF | *Open Shortest Path First* | Abrir el Camino más Corto Primero |
| RIP | *Routing Information Protocol* | Protocolo de Información de Encaminamiento |
| BGP | *Border Gateway Protocol* | Protocolo de Puerta de Enlace de Frontera |
| QoS | *Quality of Service* | Calidad de Servicio |
| SDN | *Software-Defined Networking* | Redes Definidas por Software |
| ML | *Machine Learning* | Aprendizaje Automático |
| SPOF | *Single Point Of Failure* | Punto Único de Fallo |
| DRL | *Deep Reinforcement Learning* | Aprendizaje de Refuerzo Profundo |
| RL | *Reinforcement Learning* | Aprendizaje por Refuerzo |
| NN | *Neural Networks* | Redes Neuronales |
| SARL | *Single-Agent Reinforcement Learning* | Aprendizaje por Refuerzo de un Solo Agente |
| MARL | *Multi-Agent Reinforcement Learning* | Aprendizaje por Refuerzo de Múltiples Agentes |
| DQN | *Deep Q Networks* | Redes Q Profundas |
| ONF | *Open Networking Foundation* | Fundación de Redes Abiertas |
| QU | *Queue Utilization* | Utilización de la Cola |
| AS | *Autonomous Systems* | Sistemas Autónomos |
| TE | *Traffic Engineering* | Ingeniería de Tráfico |
| IoV | *Internet of Vehicles* | Internet de los Vehículos |
| IoT | *Internet of Things* | Internet de las Cosas |
| AO | *Aquila Optimizer* | Optimizador Aquila |
| TCP | *Transmission Control Protocol* | Protocolo de Control de Transmisión |
| LLDP | *Link Layer Discovery Protocol* | Protocolo de Descubrimiento de Capa de Enlace |
| ARP | *Address Resolution Protocol* | Protocolo de Resolución de Direcciones |
| CSV | *CommaSeparated Values* | Valores Separados por Comas |
| JSON | *JavaScript Object Notation* | Notación de Objetos JavaScript |
| UDP | *User Datagram Protocol* | Protocolo de Datagramas de Usuario |
| IQL | *Independent Q-Learning* | Q-Learning independiente |

| | | |
|---|---|---|
| IQRSMR | *Intelligent and Qos-Aware Routing in Hierarchical SDN using Multi-Agent Reinforcement Learning* | Enrutamiento Inteligente y Consciente de Qos en SDN Jerárquico usando Aprendizaje por Refuerzo de Agentes Múltiples |
| HTTP | *Hypertext Transfer Protocol* | Protocolo de Transferencia de Hipertexto |
| SBI | *SouthBound Interface* | Interfaz Hacia el Sur |
| NBI | *NorthBound Interface* | Interfaz Hacia el Norte |

# Capítulo 1

# Introducción

## 1.1 Planteamiento del Problema

El enrutamiento selecciona y define rutas para el tráfico de paquetes IP (*Internet Protocol*) desde un nodo de origen a un nodo de destino [3]. Las redes tradicionales, que integran estrechamente el software y el hardware [4], han utilizado algoritmos de enrutamiento convencionales como OSPF (*Open Shortest Path First*) [5], RIP (*Routing Information Protocol*) [6] y BGP (*Border Gateway Protocol*) [7]. Las técnicas de enrutamiento mencionadas generalmente toman decisiones basadas en el estado del enlace, el número de saltos e información de la ruta limitada, respectivamente, lo que lleva a una adaptación lenta a los cambios dinámicos del tráfico y restringe el cumplimiento de los requisitos de QoS (*Quality of Service*) [8]. Además, el crecimiento continuo de Internet y la diversidad de aplicaciones (centros de llamadas, centros de datos e internet de las cosas) han generado desafíos importantes impactando la eficiencia del enrutamiento.

Las SDN (*Software-Defined Networking*), abordan los problemas de capacidad de gestión en las redes tradicionales separando los planos de control y datos, para facilitar la innovación y la evolución de las redes [9]. El plano de control es programable, lo que posibilita el enrutamiento a través de un mecanismo simple, escalable y que ahorra tiempo para ofrecer una mejor QoS [10]. La arquitectura SDN se puede clasificar en centralizada (caracterizada por el uso de un solo controlador) y distribuida (caracterizada por el uso de varios controladores), esta última se divide en dos dependiendo de la disposición de los controladores: Planas (caracterizada por el uso de controladores distribuidos conectados horizontalmente) y Jerárquicas (caracterizada por el uso de controladores organizados verticalmente).

El cálculo de rutas SDN centralizadas se ha abordado en [11, 12] utilizando técnicas

convencionales. En [11] se calculó la ruta más corta con respecto a dos nodos y en [12] teniendo en cuenta el *packet loss* y el *delay*. Esas investigaciones no explotan la inteligencia que ofrecen las técnicas de ML (*Machine Learning*), las cuales permiten aprender de forma autónoma y así tomar decisiones óptimas de enrutamiento adaptables a las variaciones del tráfico [13]. Los trabajos en [13–16] introdujeron estrategias de enrutamiento que abarcan ML y SDN. Aunque utilizan métricas de QoS (*packet loss, delay* y *throughput*), todavía existen problemas con las arquitecturas centralizadas. Primero, la baja escalabilidad por tener un solo controlador, el cual se puede ver saturado por el crecimiento de la red debido a la cantidad de tráfico que se genera. Segundo, el problema de SPOF (*Single Point Of Failure*) [17], donde si el único controlador falla, toda la red quedaría incomunicada al no poder realizar enrutamiento. Como una posible solución a las dificultades que presentan los sistemas SDN centralizados, se propusieron arquitecturas de plano de control distribuidas: Planas y Jerárquicas [18].

Las SDN Planas se puede considerar como una división horizontal de la red, en la que cada división consta de conmutadores SDN administrados por un solo controlador [9]. El enrutamiento en las SDN Planas se ha abordado en varias investigaciones [19–21] utilizando la técnica tradicional de la ruta más corta, mejorando así el *delay* y la resiliencia de la red. Pero esas investigaciones no usan técnicas de ML, por lo que no pueden aprender los caminos. Aunque estas propuesta pueden llegar a elegir la ruta óptima, no se garantiza la QoS. La investigación en [18] aplicó DRL (*Deep Reinforcement Learning*) para mejorar el enrutamiento utilizando RL (*Reinforcement Learning*) y NN (*Neural Networks*), que pueden predecir la carga del controlador y el *delay* de un extremo a otro. Lamentablemente, la solución basada en DRL no tiene en cuenta la métrica del *throughput*, por lo que no puede encontrar la ruta óptima para cada flujo (que satisfaga los requerimientos de QoS).

De acuerdo con [22] las arquitecturas SDN planas presentan gran sobrecarga en la red debido a la comunicación entre controladores cuando se utilizan mecanismos para cumplir con los requisitos de QoS, afectando la escalabilidad, disponibilidad y confiabilidad, por ende no se puede garantizar un enrutamiento eficiente, ni satisfacer la QoS.

Las SDN Jerárquicas asumen que el plano de control de la red está dividido verticalmente en múltiples capas [9]. La capa inferior consta de controladores locales, que solo tienen la vista del subdominio bajo su control. Lo anterior significa que no existe interconexión entre estos controladores, eliminando así la sobrecarga que esto implica [22]. En los trabajos [10, 23–36] se han utilizado técnicas tradicionales como *shortestpath* y *hop count* para realizar el enrutamiento en la red, teniendo mayor escalabilidad, rendimiento y baja complejidad computacional, debido a que distribuyen la carga de trabajo entre cada capa de controladores que brindan servicios específicos [37]. Sin

embargo, no utilizan ninguna técnica de ML para realizar un enrutamiento autónomo, por tanto no pueden elegir rutas óptimas y presentan limitaciones para cumplir con los requisitos de QoS. A diferencia de los trabajos anteriores que no aprenden caminos, las soluciones [38] y [39], con la ayuda de técnicas de ML, brindan la posibilidad de un enrutamiento inteligente. Sin embargo, tienen a un solo agente (un agente percibe e interpreta el entorno, ejecuta acciones y aprende a través de prueba y error), enfrentando el problema de baja escalabilidad y robustez a medida que las redes se expanden [40].

Los enfoques existentes para realizar enrutamiento inteligente en SDN muestran las siguientes limitaciones. En primer lugar, las soluciones [10, 23–26, 30, 33, 34, 36] no son conscientes de la QoS, es decir que no usan métricas como el *throughput*, *packet loss* y *delay* para garantizar un alto rendimiento en la SDN. En segundo lugar, los trabajos [27, 28, 31, 32] no pueden realizar un enrutamiento eficiente debido a que utilizan algoritmos de enrutamiento tradicionales. En tercer lugar, ninguno de los trabajos anteriores ha estudiado la técnica de ML con múltiples agentes para realizar un enrutamiento inteligente consciente de la QoS en una SDN jerárquica. En este sentido, la pregunta de investigación del presente trabajo de grado es:

**¿Cómo realizar enrutamiento inteligente y consciente de la calidad del servicio en redes jerárquicas definidas por software?**

## 1.2 Objetivos

### 1.2.1 General

Proponer un mecanismo de enrutamiento inteligente y consciente de la calidad del servicio en SDN jerárquicas.

### 1.2.2 Específicos

- Diseñar un mecanismo inteligente con múltiples agentes basado en ML y un enrutamiento consciente de la calidad del servicio en SDN jerárquicas.
- Implementar un prototipo del mecanismo diseñado.
- Evaluar el mecanismo diseñado con respecto al *packet loss, delay* y *throughput*.

## 1.3 Contribuciones

Las contribuciones del presente trabajo de grado son las siguientes:

- Un mecanismo de enrutamiento inteligente y consciente de la calidad de servicio para redes jerárquicas definidas por software.
- Una implementación del mecanismo propuesto utilizando el algoritmo de Q-Learning con múltiples agentes, evaluando su desempeño en cuanto a *throughput*, *delay* y *packet loss*.
- Una evaluación comparativa del mecanismo propuesto con técnicas que utilizan información respecto al estado del enlace, como lo son Dijkstra y Q-Learning.

## 1.4 Estructura

Este documento está organizado de la siguiente manera.

- En este Capítulo introductorio, se establece el planteamiento del problema, los objetivos del presente trabajo de grado, las contribuciones y una descripción de la estructura general de la disertación.
- El Capítulo 2 realiza una revisión exhaustiva de los conceptos fundamentales vinculados a este trabajo de investigación. Explora temáticas clave como el aprendizaje por refuerzo distribuido, el enrutamiento inteligente y consciente de la calidad

de servicio, así como las redes jerárquicas definidas por software. Además, se expone detalladamente el trabajo previo y las investigaciones relacionadas que han abordado el enrutamiento en las redes jerárquicas definidas por software.

- El Capítulo 3 presenta el diseño del prototipo, incluyendo la descripción detallada de los módulos construidos, tanto para el algoritmo de enrutamiento, como para la infraestructura de control.
- El Capítulo 4 aborda la evaluación del prototipo, describiendo la configuración del entorno de prueba y las métricas de rendimiento utilizadas, como la generación de tráfico y la determinación de los parámetros del algoritmo. Presenta también un análisis exhaustivo de los resultados obtenidos.
- El Capítulo 5, expone las conclusiones derivadas del presente trabajo de grado y plantea futuras investigaciones.

# Capítulo 2

# Marco Teórico y Estado del Arte

Este capítulo presenta los conceptos fundamentales que sustentan el contenido de este trabajo de investigación. En primer lugar, examina el concepto de aprendizaje por refuerzo distribuido, un enfoque crucial en el ámbito de la inteligencia artificial. Además, aborda la noción de enrutamiento inteligente y consciente de la calidad del servicio, un elemento esencial para optimizar el rendimiento en entornos dinámicos y demandantes.

La arquitectura de red jerárquica definida por software emerge como otro componente central abordado en este capítulo. Esta estructura innovadora provee una plataforma sólida para la implementación y gestión eficiente de las redes, permitiendo una adaptabilidad y control en la configuración y operación de los sistemas que la componen. Por último, se presenta el trabajo relacionado en el ámbito de estudio, destacando conexiones y comparativas entre investigaciones previas. Además, se identifican las brechas existentes, lo que motiva y justifica la contribución del presente trabajo de grado al conocimiento actual.

## 2.1 Aprendizaje por Refuerzo Distribuido

El aprendizaje por refuerzo (RL, por sus siglas en inglés) es un paradigma de ML que define el entrenamiento de uno o varios agentes para aprender a realizar una tarea a través de interacciones repetidas de prueba y error con un entorno dinámico [41]. La Figura 2.1 muestra las acciones de un agente en el entorno en base a las observaciones (retroalimentación). La retroalimentación es basada en una función de recompensa o costo que evalúa los estados del sistema con el propósito de mejorar la métrica asociada a la recompensa [42] y luego mejorar el próximo comportamiento para lograr el objetivo final [43].

Una forma de clasificar las técnicas de RL es por el número de agentes:

- **Aprendizaje por refuerzo con un solo agente:** SARL (*Single-Agent Reinforcement Learning*) implementa un agente que interactúa con el entorno, como se muestra en la Figura 2.1. El agente actúa automáticamente para que el entorno cambie (acción). Luego recibe el efecto de esa acción (retroalimentación). La retroalimentación conduce a un nuevo estado y una señal de recompensa que indica si logro algún objetivo [1].

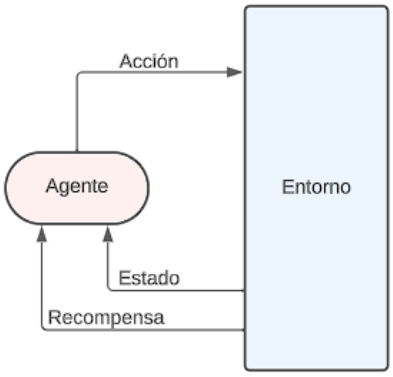



Figura 2.1: Modelo de aprendizaje por refuerzo con un solo agente [1].

Algunos de los algoritmos utilizados por este modelo son:

  - **SARSA** (*state, action, reward, state, action*) es utilizado para aprender una política de toma de decisiones en un entorno particular. Este algoritmo actualiza los valores $Q$, que representan la estimación del valor esperado de una acción en un estado específico. Sigue un enfoque de tipo *on-policy*, es decir, que se adapta a medida que el agente elige sus acciones. SARSA requiere que cada par estado-acción sea visitado, además de una política codiciosa para lograr la convergencia [1].
  - **Q-learning** se destaca como una de las estrategias más populares y ampliamente empleadas en RL, en gran parte debido a su facil implementación. Es un método *off-policy* que aprende los valores $Q$ óptimos en lugar de valores de estado, al mismo tiempo que determina una política óptima.

    Trabajar con valores $Q$ es justificado por la necesidad de tomar decisiones basadas en estos. En cada iteración, el agente conoce dos estados: el estado actual y uno de sus sucesores. Los valores $Q$ ofrecen una estimación de la calidad futura de las acciones, simplificando así la tarea de seleccionar una acción. El agente sigue iterando hasta alcanzar un número predefinido de episodios o hasta que logre converger hacia los valores óptimos [1].

  - **DQN** (*Deep Q-Network*) utiliza dos NN con una arquitectura idéntica para estimar los valores $Q$. Una es denominada NN principal y su función es estimar el valor $Q$ para el par estado-acción actual, así como llevar a cabo el proceso de aprendizaje. Por otro lado, la NN objetivo es empleada para estimar el valor correspondiente al par estado-acción en el siguiente estado. Es importante destacar que la NN objetivo no se actualiza en cada iteración de entrenamiento, sino que cada varios miles de pasos copia los valores de la principal. Este enfoque evita problemas en el proceso de entrenamiento, ya que una actualización constante podría generar dificultades en la convergencia [44].

- **Aprendizaje por refuerzo con múltiples agentes:** En MARL (*Multi-Agent Reinforcement Learning*) varios agentes toman acciones basadas en el mismo entorno, como se observa en la Figura 2.2, pero sin poder predecir las acciones de otros agentes, ni los cambios en el entorno, ya que no siempre tienen una visión completa de este [45]. MARL puede ejecutar tareas más complejas que SARL, ya que varios agentes toman una serie de decisiones para maximizar la recompensa. Todos los agentes cooperan para completar una tarea en el menor tiempo posible [41].

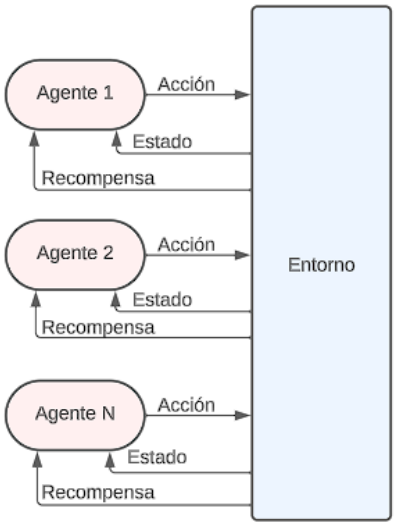



Figura 2.2: Modelo de aprendizaje por refuerzo con múltiples agentes [1].

Algunos de los algoritmos utilizados por este modelo son:

  - **IQL** (*Independent Q-Learning*) es uno de los enfoques más simples y populares para abordar problemas de RL con múltiples agentes. En este enfoque, cada agente opera de manera independiente, sin tener en cuenta la presencia o acciones de los demás en el entorno. Cada uno aprende su propia

función $Q$, que condiciona su comportamiento en función del estado actual y de su propia acción. Este modelo resulta atractivo debido a su capacidad para evitar el problema de escalabilidad en situaciones con múltiples agentes.

A pesar de su simplicidad, IQL enfrenta el desafío de la no estacionariedad del entorno al aplicarse a problemas de DRL, debido a esto es posible que la memoria de repetición de la experiencia ya no refleje la dinámica actual que el agente debe aprender. Como resultado, reutilizar experiencias obsoletas puede llevar a la confusión continua de la red neuronal y hacer que el proceso de entrenamiento sea inestable [46].

- **Minimax-Q** fue desarrollado para juegos estrictamente competitivos. Su principal objetivo es seleccionar una política que maximice la recompensa acumulada esperada, incluso cuando se enfrenta circunstancias desfavorables. Este enfoque considera que el espacio de acciones de otros agentes se comporta como si fueran adversarios en el juego. Cada uno busca maximizar su recompensa, tomando en cuenta la suposición de que los demás actúan de manera hostil y seleccionan sus acciones con el fin de minimizar las ganancias del agente en cuestión [47].
- **WoLF Policy Hill Climber** es un enfoque en donde los agentes ajustan su tasa de aprendizaje y su probabilidad de exploración en función de su desempeño previo. Cuando un agente está obteniendo recompensas altas, tiende a reducir su tasa de aprendizaje y aumentar la probabilidad de explotación, con el propósito de consolidar su estrategia actual. Por otro lado, si un agente no logra obtener recompensas satisfactorias, puede incrementar su tasa de aprendizaje y la probabilidad de exploración para explorar estrategias alternativas.

  Se utiliza en entornos competitivos, donde los agentes aprenden estrategias equilibradas a través de la competencia y la cooperación. Los agentes adaptan sus estrategias con el objetivo de ganar, pero también están dispuestos a aprender si no obtienen los resultados deseados [1].

El aprendizaje por refuerzo distribuido se presenta como una variante del aprendizaje por refuerzo convencional. Su objetivo principal radica en la distribución del proceso de aprendizaje entre múltiples agentes. Esto conlleva a un incremento en la velocidad de adquisición de conocimientos y a una comprensión más completa del entorno en cuestión [48].

En este escenario, cada agente toma decisiones basadas en sus observaciones y participa en interacciones con el entorno. Lo que puede llevar a un aprendizaje más eficiente y a la identificación de estrategias que podrían no ser evidentes para un único

agente.

Algunas de las arquitecturas más antiguas encontradas en este campo son:

- Gorila [49] describe un aprendizaje distribuido basado en muchos agentes que adquieren las experiencias y las comparten con múltiples estudiantes, para mejorar la confiabilidad [50].
- A3C[1] descrita en [51]. La idea principal es ejecutar los episodios en paralelo usando múltiples actores en diferentes hilos. De esta forma las experiencias se recogen mucho más rápido y con un mayor abanico de diferencias, lo que conduce a un mejor conocimiento del entorno [48].
- En [52] se diseña un algoritmo DQN distribuido, llamado Ape-X DQN, utilizando un búfer de reproducción de prioridad distribuido [53] para mejorar la eficiencia del entrenamiento [50].

Una de las principales ventajas del RL distribuido es que puede escalar para manejar entornos grandes y complejos, donde un solo agente puede no ser suficiente para explorar y aprender la política óptima [54]. También se puede utilizar para mejorar la solidez y la eficiencia de los algoritmos de RL, ya que permite que los agentes puedan aprender de la experiencia de los demás.

## 2.2 Enrutamiento Inteligente y Consciente de la Calidad de Servicio

Las técnicas de ML brindan inteligencia a SDN, lo que le permite aprender de forma autónoma. Estas técnicas de ML consideran las métricas del estado de la ruta para producir un enrutamiento inteligente, proactivo y eficiente que se adapta a los cambios dinámicos del tráfico [13]. Este enrutamiento inteligente mide periódicamente el rendimiento de la red y ajusta los flujos de tráfico para proporcionar una conectividad y una distribución de la carga óptima con la mayor precisión posible [55]. Hay dos grupos de algoritmos de enrutamiento:

- **Los algoritmos no adaptables o estáticos**, los cuales no utilizan información de tráfico de red/estado para tomar decisiones de enrutamiento. Además, las tablas de enrutamiento se configuran manualmente y normalmente se implementan como algoritmos centralizados. Es decir, un conmutador calcula todas las rutas y las transmite a los demás [56].

[1] algoritmo diseñado para entrenar agentes en entornos secuenciales utilizando aprendizaje asincrónico y la arquitectura actor-crítico para mejorar la eficiencia de la muestra y la estabilidad del entrenamiento.

- **Los algoritmos adaptativos o dinámicos**, toman decisiones de enrutamiento en función de un conjunto de requisitos que permiten elegir la ruta óptima. Esta ruta no es necesariamente la más corta, ya que las restricciones las impone QoS, que define la capacidad de una red para proporcionar los servicios necesarios para un tráfico de red en particular [57]. Esto permite una estandarización del rendimiento de una red, a través de las siguientes métricas [56, 58]:
  - ***Throughput*:** Es una medida de la capacidad de transmisión de datos, que se refiere a la máxima cantidad de bits por segundo que pueden viajar con éxito de un extremo a otro [59, 60]. Está relacionado con el *bandwidth* disponible. Un mayor *bandwidth* de enlace da como resultado un mayor *throughput* [61].
  - ***Delay*:** Es el tiempo necesario para que un paquete de datos se emita de un transmisor hasta el receptor de la comunicación. Es una medida que expresa el tiempo gastado en el subsistema de comunicación [62].
  - ***Packet Loss*:** Indica el número de paquetes que no llegan a su destino durante la transmisión. Es causado por errores en la transmisión de datos, generalmente debido a la congestión de un conmutador en la red, que ocurre cuando la demanda de tráfico excede los recursos disponibles [59]. También puede ser ocasionado por colisiones, rupturas de enlaces y poca potencia de la señal [63].

El objetivo principal de QoS es permitir que el algoritmo de enrutamiento encuentre la ruta con la mayor recompensa para un par origen-destino, utilizando la información sobre los recursos de red disponibles [39]. La recopilación de información sobre el tráfico y el estado de la red en tiempo real, es esencial para las estrategias de enrutamiento QoS. Además, el rendimiento de cualquier algoritmo de enrutamiento está relacionado con la precisión de la información sobre el estado de la red. Es decir, cuanto más precisa sea la información del estado del tráfico, más preciso será el algoritmo de enrutamiento para implementar estrategias de red [11].

## 2.3 Redes Jerárquicas Definidas por Software

La ONF (*Open Networking Foundation*) [64] ha proporcionado la siguiente definición: SDN es una arquitectura de red emergente en la que el control de la red está desacoplado del reenvío y es directamente programable [65]. Hay tres planos en una arquitectura SDN típica como se observa en la figura 2.3, que pueden variar su ubicación física. Primero, el plano de datos, que consiste en un conjunto distribuido de conmutadores a cargo de reenviar paquetes. Segundo, el plano de control, el cual consta de

un controlador de software lógicamente centralizado, que se encarga de manejar las comunicaciones entre las aplicaciones de red y los dispositivos a través de interfaces abiertas (por ejemplo, OpenFlow[2] hacia el plano de red, y API REST[3] hacia el plano de aplicación) Por último, está el plano de aplicación, el cual comprende programas (enrutamiento, cortafuegos, balanceo de carga) diseñados para implementar estrategias y lógica de control de red.

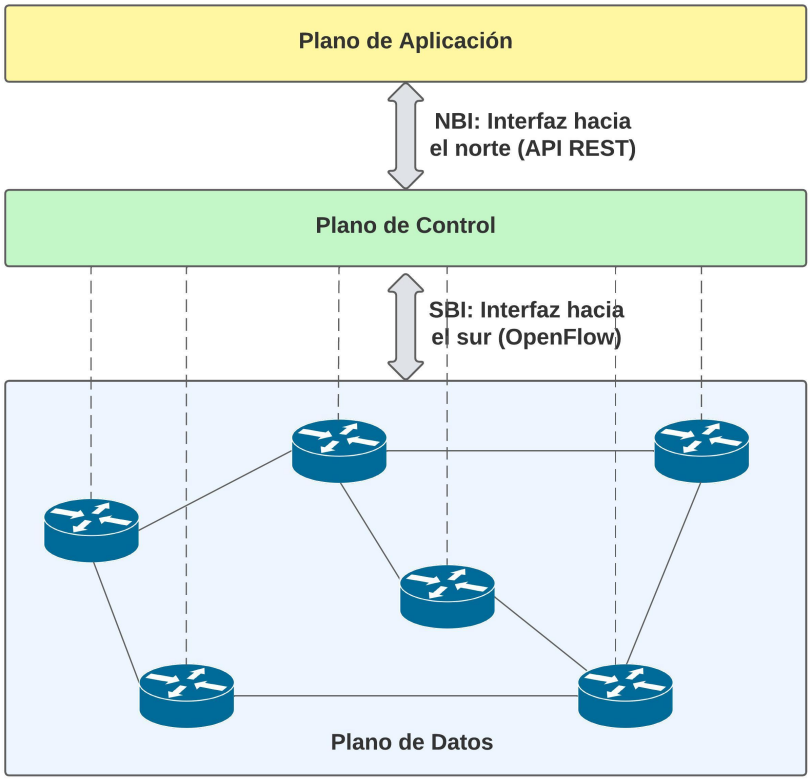



Figura 2.3: Arquitectura de Red Definida por Software.

La arquitectura SDN se puede clasificar según la ubicación del controlador raíz:

- **Redes Definidas por Software Centralizadas**: constan de un solo controlador para toda la red, lo que significa que el controlador raíz tiene conectados todos los conmutadores de red [68]. Este modelo tiene dos problemas críticos. Primero, escalabilidad pobre, porque un solo controlador puede verse abrumado (cuello de botella) cuando las solicitudes desde el plano de datos aumentan. Segundo, baja robustez, ya que el único controlador puede fallar (SPOF), provocando que el plano de datos no pueda reenviar los paquetes de flujos desconocidos [69].

[2]interfaz definida entre el plano de datos y control, usada para el acceso y manipulación de dispositivos de reenvió como conmutadores o enrutadores [66].

[3]interfaz que se comunica mediante peticiones HTTP para realizar funciones como crear, leer, actualizar y eliminar [67].

- **Redes Definidas por Software Distribuidas**: constan de un plano de control distribuido físicamente en varios controladores que comparten la carga de la red. En los últimos años se han propuesto varios diseños (híbrida, jerárquica y plana) [68], como una alternativa a los problemas encontrados en la SDN centralizada (escalabilidad pobre y baja robustez). Este trabajo de grado hace énfasis en diseño jerárquico, el cual se explica a continuación.

Las SDN jerárquicas son una arquitectura en donde el plano de control esta dividido verticalmente en múltiples capas según los servicios requeridos. Idealmente, un enfoque jerárquico puede admitir cualquier número de niveles, pero es fundamental encontrar el numero óptimo para cada escenario [70]. La Figura 2.4 muestra la arquitectura SDN jerárquica, donde los controladores locales manejan los requisitos de las aplicaciones con eventos frecuentes y envían su vista al controlador de nivel superior (controlador raíz), el cual se ocupa de las aplicaciones que requieren una vista de red global [69]. Las SDN jerárquicas, tiene la ventaja de mejorar el *delay* de la comunicación entre el conmutador y controlador, además de reducir significativamente la información de topología [71], dado que un controlador local gestiona cada dominio, y cada controlador ve a las otras redes como conmutadores individuales [72], mejorando la escalabilidad y el rendimiento de la red.

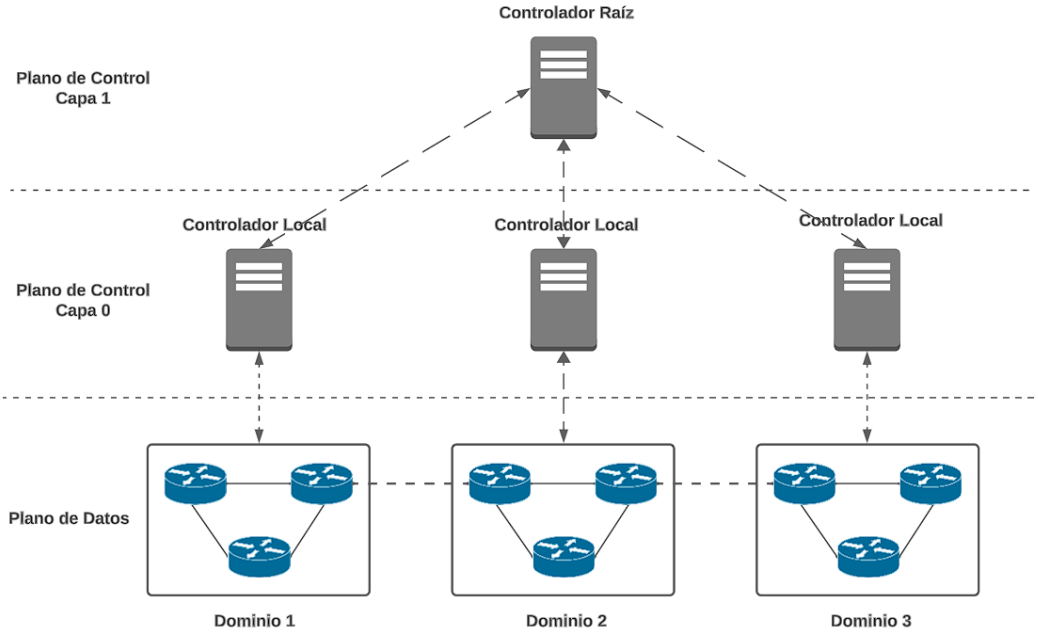



Figura 2.4: Arquitectura de Red Jerárquica Definida por Software [2].

Algunas de las diferentes plataformas de control para SDN jerárquica son:

- Kandoo [73], el cual propone una arquitectura de control jerárquico de dos capas que permite la coexistencia de aplicaciones con diferentes requisitos. Los contro-

ladores de capa inferior administran los conmutadores OpenFlow de un dominio sin tener información de estado de toda la red. Por otro lado, la capa superior consiste en un controlador raíz centralizado lógicamente que mantiene la información de estado de toda la red. Este controlador raíz es responsable de las aplicaciones globales y de instalar las entradas de flujo en los conmutadores, delegando solicitudes a los controladores locales [2].

- B4 [74] de Google es una SD-WAN entre dominios que conecta múltiples centros de datos en todo el mundo. Consiste en un plano de control jerárquico de dos niveles para satisfacer las demandas de *bandwidth* elástico de los centros de datos. Implementa varias aplicaciones de control para lograr redes rentables. La capa inferior se compone de controladores basados en Onix que gestionan cada sitio del centro de datos y utilizan aplicaciones de control local. En la capa superior, una SDN Gateway global recopila y gestiona la información de red de los controladores de nivel inferior y la reenvía a un servidor [2].

- Espresso [75] utiliza una arquitectura de plano de control jerárquico de dos capas. Los controladores locales reaccionan a los eventos de redes locales, como fallas en los puertos, mientras que los controladores de capa superior optimizan el tráfico global para mejorar la eficiencia. Está diseñado para soportar operaciones a gran escala seguras, automatizadas e incrementales. Además, utiliza principios de diseño de desarrollo de software que permiten la evolución de la red e implementar nuevas características innovadoras. Proporciona un sistema estático de fallas en el que si un controlador local falla, el plano de datos funciona según el último estado conocido [2].

- ORION [76] plantea un plano de control jerárquico híbrido que actúa como la arquitectura del plano de control plana (controladores conectados horizontalmente) mientras distribuye jerárquicamente los controladores. Consta de dos capas. La capa inferior incluye controladores locales que son responsables de gestionar solicitudes y abstraer información de red para enviarla a la capa superior. La capa superior contiene los controladores de dominio que mantienen la vista de red global. El problema de extensión de ruta de la arquitectura jerárquica, se reduce mediante el cálculo de la ruta más corta al sumar los saltos entre y dentro de las áreas [77].

## 2.4 Trabajo Relacionado

Esta sección describe trabajos relacionados a este trabajo de grado, enfocándose principalmente en el enrutamiento en SDN jerárquicas.

### 2.4.1 Enrutamiento en Redes Jerárquicas Definidas por Software

**Machine Learning Aided Load Balance Routing Scheme Considering Queue Utilization [38]**. Propuso una arquitectura jerárquica basada en *Software Defined Routing*, la cual consiste en enrutadores locales responsables de detectar el estado de la cola para mantener el equilibrio de carga, y en enrutadores centrales que detectan la QU (*Queue Utilization*) y el patrón de tráfico de la cola. Los esquemas de enrutamiento se basan en el equilibrio de carga y los algoritmos basados en ML para explorar la relación entre el estado del tráfico, la topología de la red y los estados de cola de los enrutadores. Además, no solo consideran el número de saltos para realizar enrutamiento, sino también QU basado en *Deep Neural Networks* [4]. Usando NN, la ruta óptima se predice basado en QU. Es notable que esta propuesta no logre un enrutamiento QoS adecuado debido a que los resultados experimentales muestran que el *delay* medio es más significativo en los esquemas propuestos que en el algoritmo de Bellman-Ford [79].

**A Three-level Routing Hierarchy in improved SDN-MEC-VANET Architecture [23]**. Introdujo una arquitectura jerárquica basada en el algoritmo de Dijkstra[5]. La arquitectura presentada se denomina software mejorado definido por VANET[6] basado en *Mobile Edge Computing*, cuyo objetivo es mejorar el rendimiento del enrutamiento y enriquecer el modo de transmisión de datos para VANET. En cuanto a los niveles de diseño: en el nivel I, los vehículos pueden comunicarse entre sí a través de diferentes protocolos VANET, el nivel II se encarga del enrutamiento, y en el nivel III, el controlador global mantiene la topología de red completa. Finalmente, la simulación de cuatro diferentes arquitecturas VANET toman el índice de entrega de paquetes y el tiempo de ida y vuelta como métricas. Cabe señalar que esta investigación no considera las métricas de QoS (*packet loss, delay* y *throughput*) para el enrutamiento en la red.

**Performance Evaluation and Optimization of Hierarchical Routing in SDN Control Plane [24]**. Expuso una arquitectura compuesta por varios controladores utilizando la teoría de colas para analizar y modelar el procesamiento de solicitudes en cada controlador y derivar una expresión para el tiempo de respuesta promedio. El modelo presentado tiene tres capas de controladores. El plano de control implementa los controladores de red y tiene funciones de enrutamiento y monitoreo. Teóricamente concluyen que el modelo propuesto es mejor en tiempo de respuesta que la arquitectura *Peer-to-Peer*[7], donde los controladores tienen una estructura *equal-to-equal*, y cada controlador tiene toda la información del plano de datos. La solución planteada no aprende las rutas de manera óptima porque no utiliza métricas de QoS (*packet loss,*

[4] modelo complejo de ML compuesto de múltiples capas de neuronas artificiales [78].
[5] algoritmo de recorrido de gráficos utilizado para encontrar el camino más corto entre nodos [80].
[6] red de comunicación inalámbrica de vehículo a vehículo y de vehículo a infraestructura [81].
[7] diseño de red descentralizado donde los nodos se comunican directamente entre sí [82].

*delay* y *throughput*) para realizar el enrutamiento.

**A Scalable Inter-AS QoS Routing Architecture in Software Defined Network (SDN) [25]**. El objetivo principal de la propuesta es mejorar la escalabilidad del plano de control al reducir la cantidad de mensajes que procesa. La arquitectura desarrollada consta de dos niveles. En el primero, están los dominios independientes, y en el segundo un supercontrolador que supervisa los controladores del primer nivel. Los resultados de la simulación muestran que, en comparación con los entornos no jerárquicos, los mensajes en una arquitectura jerárquica disminuyen, pero las conexiones entre los sistemas aumentan. La solución propuesta no utiliza métricas de QoS (*packet loss, delay* y *throughput*) para realizar un enrutamiento eficiente.

**Traffic engineering in hierarchical SDN control plane [26]**. Presentó un algoritmo TE (*Traffic Engineering*) introducido por Google B4 para rediseñar el plano de control de SDN de modo que se pueda aplicar en múltiples dominios. Este trabajo uso una arquitectura jerárquica, en la cual cada dominio es administrado por un controlador local, mientras que un controlador raíz realiza la gestión global. El *delay* y *bandwidth* fueron utilizados como métricas para obtener la mejor ruta entre dominios. Sin embargo, no había métricas de QoS como el *packet loss* y *throughput* para garantizar una red de alto rendimiento. Para las pruebas, la arquitectura jerárquica se comparó con una centralizada (sin garantía de demora en la asignación de *bandwidth*). Los resultados de la simulación en una topología de 5 dominios con 717 conmutadores mostraron que la arquitectura jerárquica podría impulsar el índice de utilización del enlace a más del 85 %, pero su rendimiento es inferior que el de la arquitectura centralizada.

**Routing Strategy for Internet of Vehicles based on Hierarchical SDN and Fog Computing [27]**. La arquitectura propuesta consta de varios niveles jerárquicos: un controlador SDN central, un controlador SDN semicentral, *Fog computing*[8] y vehículos IoV[9] (*Internet of Vehicles*). Todos los controladores utilizados obtienen la mejor ruta posible en términos de *delay*, *bandwidth*, la velocidad y la ubicación del vehículo. La estrategia de enrutamiento propuesta se compara con el modelo IoV-Fog System, que no usa SDN, e IoV-Fog Central (usa SDN). Los resultados de la simulación mostraron que la arquitectura propuesta mejora la pérdida de paquetes, la transmisión exitosa dentro del plazo y el *delay* promedio de extremo a extremo. Lo anterior se debe a que los enrutadores usan tablas de flujo para reenviar paquetes. Además, al ser una arquitectura distribuida, el controlador central no sufre una sobrecarga de flujos. Por lo tanto, es más rápido calcular la ruta óptima, ya que depende de la cantidad de dispositivos y

[8] extiende las capacidades de la computación en la nube hasta el borde de la red, permitiendo el procesamiento de datos y la ejecución de aplicaciones más cerca de las fuentes de datos [83].

[9] red de vehículos e infraestructura conectados que intercambian datos en tiempo real [84].

enlaces de la red. Sin embargo, la solución propuesta no aprende a mejorar el cálculo de rutas porque no utiliza técnicas de ML.

**Routing in Fog-Enabled IoT Platforms: A Survey and an SDN-based Solution [28]**. Este enfoque expuso una solución basada en Dijkstra y una arquitectura jerárquica *Fog computing* basada en SDN con el objetivo principal de mejorar el enrutamiento de datos ubicuos de IoT[10] (*Internet of Things*). La arquitectura tiene dos niveles de controladores, mas el nivel de IoT que está compuesto de diferentes dispositivos conectados. El nivel de niebla incluye servidores que recopilan y procesan datos de IoT de primer nivel y controladores encargados de administrar a los servidores. El nivel de nube implica un servidor que procesa datos y un controlador que garantiza la gestión global de toda la red. Los resultados de la simulación muestran que el rendimiento de la red mejora con respecto a las métricas de QoS, como el *packet loss, delay* y *throughput* al aumentar la cantidad de controladores de niebla en la arquitectura. Sin embargo, el algoritmo de Dijkstra no es apropiado, ya que carece de técnicas de RL y, por lo tanto, no es capaz de aprender las rutas.

**A hierarchical approach for accelerating IoT data management process based on SDN principles [33]**. Propuso un enfoque jerárquico basado en SDN para acelerar la gestión de datos y equilibrar la carga entre dispositivos IoT en múltiples dominios. El algoritmo utilizado es el *Central Scheduler Load Balancing* [86] que consta de tres pasos. Primero, calcula y actualiza la carga en cada dispositivo. Después, toma decisiones en función del valor y la elección del mejor dispositivo para la migración. Por último, determina el exceso de carga que debe transferirse (de un controlador sobrecargado a otro). Las métricas utilizadas son el tiempo medio de respuesta (tiempo total dedicado al proceso desde la llegada de una tarea hasta su finalización) y el tiempo medio de espera (intervalo de tiempo en el que una tarea espera en cola para ser procesada). Sin embargo, la solución propuesta no aprende rutas basadas en métricas de QoS, ni utiliza un algoritmo de enrutamiento eficiente. Finalmente, los resultados demuestran que la solución pudo reducir el tiempo promedio de respuesta y el tiempo promedio de espera, en comparación con un enfoque sin equilibrio de carga.

**LSEA: Software-Defined Networking-Based QoS-Aware Routing Mechanism for Live-Soccer Event Applications in Smart Cities [10]**. Introdujo una arquitectura de red jerárquica y un algoritmo de enrutamiento con restricciones múltiples basado en SDN. Está dividido en dos niveles, el inferior cuenta con un controlador de zona, y el superior, con un controlador maestro, que gestiona a los de zona y reduce las interacciones entre ellos. El algoritmo implementa *Constrained Shortest Path First*[11], que

[10] red de dispositivos que recopilan, intercambian y transmiten datos a través de Internet [85].
[11] algoritmo de enrutamiento utilizado para encontrar la ruta más corta entre nodos considerando res-

verifica que cada conmutador cumpla con los requisitos de *bandwidth*, *delay* y fluctuación tanto para el enrutamiento intra-dominio como para entre-dominios. Este algoritmo reduce considerablemente la sobrecarga de enlaces y asegura la QoS en comparación con el algoritmo de Dijkstra, pero al igual que este, no puede aprender rutas. Las pruebas tanto teóricas como experimentales se realizaron en una red aleatoria generada.

**QoS-Aware Adaptive Routing in Multi-layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach [39]**. La arquitectura consta de tres niveles de controladores: supercontroladores, de dominio y esclavo. Esta arquitectura permite una señalización rápida para resolver la complejidad del diseño y la inconsistencia de la información. Además, el algoritmo está asistido por RL al examinar la política de acción, la función de calidad, los ingresos a largo plazo y el modelo de sistema con función de recompensa. Este último considera la función QoS-aware, que permite al agente encontrar la ruta con mayor recompensa según el tipo de tráfico y aplicación (*packet loss, delay* y *throughput*). Logrando así un enrutamiento de aprovisionamiento de QoS eficiente y adaptativo en comparación con el enfoque de aprendizaje convencional en una red Sprint GIP [88]. Como la solución expuesta implementa solo un agente, puede tener problemas para manejar la complejidad y la cantidad de solicitudes.

**SDN- and fog computing-based switchable routing using path stability estimation for vehicular ad hoc networks [29]**. Introdujo un método para mejorar la transmisión de datos en las comunicaciones de vehículo a vehículo, haciendo uso de SDN y *Fog computing*. El plano de control tiene una estructura jerárquica con un controlador SDN central en su nivel más alto. El nivel inferior esta compuesto por estaciones base y controladores SDN de carretera, este nivel brinda servicios de *Fog computing* y envía información periódicamente al controlador central. El método propuesto de enrutamiento conmutable, proporciona la mejor ruta para la transmisión de datos a través de la infraestructura VANET y la transmisión de Internet. El enrutamiento se basa en el peso calculado por los conmutadores de niebla para cada borde del gráfico. El peso es calculado periódicamente y enviado al controlador, el cual para encontrar la ruta ejecuta el algoritmo de Dijkstra. Luego, los conmutadores calculan el peso de cada ruta. Un peso más bajo indica una ruta óptima con una longitud más corta, menos *delay*, mayor densidad y estabilidad. El método propuesto fue simulado y comparado con AODV [89], GPSR [90], IGR [91] y SFIR [92]. Los resultados de la simulación basada en un entorno urbano real, sugieren un mejor rendimiento en la tasa de entrega de paquetes, el *delay* promedio de extremo a extremo, *packet loss*, la sobrecarga de enrutamiento y la tasa de fallas de enrutamiento, con la limitación de no poder aprender las rutas.

tricciones específicas [87].

**Topology-Preserving Traffic Engineering for Hierarchical Multi-Domain SDN [30]**. Presentó el primer esquema de TE para una SDN multidominio mediante el uso de un plano de control jerárquico, sin que ningún dominio revele su vista de red local. Para esto construyó una red, donde solo los conmutadores de borde estaban expuestos a su controlador de capa superior. También diseñó un protocolo de comunicación para permitir que los controladores en diferentes capas y dominios colaboren en la asignación de *bandwidth*. El algoritmo tiene como objetivo determinar la ruta más corta para cada solicitud de flujo, que puede ser tanto entre dominios como entre hosts. Calcula el QoS para cada ruta considerando el *delay*. Posteriormente, se asigna el *bandwidth* de acuerdo con el nivel de participación, con el fin de maximizar la utilización de la red y garantizar una distribución equitativa. Los experimentos en una red real [93] muestran que la propuesta puede aumentar el uso del enlace por encima del 85 %, en comparación con el método centralizado. El esquema propuesto no puede aprender las rutas.

**Weighted routing in hierarchical multi-domain SDN controllers [31]**. Expuso una aplicación de enrutamiento bajo una arquitectura jerárquica que consta de dos tipos de controladores, local y global. Este trabajo propone evitar enlaces congestionados mediante el uso de una función de peso de enlace que considera la capacidad del enlace el *bandwidth* consumido y la magnitud del impacto de la utilización del enlace. Se utiliza tanto para el enrutamiento intra-dominios, donde el controlador de dominio calcula la información de peso, como para el enrutamiento entre-dominios, donde el controlador global calcula la ruta ponderada más corta desde el conmutador de origen hasta la puerta de enlace del borde de destino. Midieron el *packet loss, delay, bandwidth* y *throughput*. Los resultados experimentales en cuatro topologías diferentes mostraron que la aplicación de enrutamiento en la arquitectura jerárquica mejoró significativamente con respecto a la arquitectura de un solo controlador. La solución planteada no considera el enrutamiento consciente del QoS.

**A hierarchical control plane for software-defined networks-based industrial control systems [32]**. La integración de los *Industrial Control Systems*[12] y SDN, a pesar de mejorar la resiliencia de la red, también la expone a las amenazas. Para resolver esto, este trabajo presentó un plano de control SDN jerárquico basado en un controlador OptimalFlow que rediseña la red como un problema de optimización *Integer Linear Programming*. OptimalFlow proporciona la ruta más corta y armoniza los requisitos de flujo, incluido QoS, la seguridad de las comunicaciones y la confiabilidad. Para minimizar el impacto de las actualizaciones de la red, proponen dos algoritmos. El primero reduce la cantidad de variables en la optimización, por lo que solo se redistribuyen los

[12]sistemas integrados de hardware y software utilizados en procesos industriales para monitorear, controlar y automatizar operaciones.

flujos que son afectados por una perturbación. La optimización selecciona la ruta más corta mientras elige enlaces con las capacidades más altas. Segundo, *Dependency Graph Construction* crea un gráfico de dependencia para las actualizaciones de la red. Este gráfico evita la congestión de los enlaces, mediante la aplicación de una estrategia de migración de flujo que se fundamenta en la asignación de prioridades. El análisis fue realizado en escenarios experimentales y basados en simulación tanto para un dominio único como para dos dominios SDN. Los resultados muestran la adaptabilidad del esquema en varios escenarios SDN. Sin embargo, la solución presentada no puede aprender las rutas.

**A Novel Routing Protocol for Hierarchical Software Defined Vehicular Adhoc Network [34]**. Propuso una VANET jerárquica híbrida definida por software con múltiples protocolos de interfaz de red y con dos interfaces. La primera interfaz proporciona la función de escalabilidad y heterogeneidad, y la segunda reduce la inferencia y maximiza la utilización de la red. El algoritmo se basa en encontrar el vecino del conmutador de origen contando los saltos. Analizaron el protocolo de enrutamiento propuesto en función de la confiabilidad del enlace, la tasa de entrega de paquetes y el *delay* de extremo a extremo. Para el análisis matemático, primero, calcularon el rendimiento para encontrar el efecto de rango y la disponibilidad de conmutadores. Luego, analizaron la confiabilidad del enlace con diferentes velocidades. Los resultados muestran que el algoritmo planteado tiene un impacto más significativo en todos los parámetros de rendimiento de la red, pero con la limitación de no ser consciente del QoS.

**QoS Support Path Selection for Inter-Domain Flows Using Effective Delay and Directed Acyclic Graph in Multi-Domain SDN [35]**. Propone un método de decisión de ruta efectivo entre dominios que satisface los requisitos de QoS (*delay* y *bandwidth*) utilizando un *Directed Acyclic Graph*[13] en SDN multidominio. La decisión de flujo se basa en la teoría del *bandwidth* efectivo del proceso de martingala[14]. Con *Effective Delay*[15], el método propuesto puede determinar la ruta de extremo a extremo para los flujos entre dominios, garantizando su QoS. La arquitectura tiene dos niveles, en donde los controladores locales intercambian información con cada conmutador utilizando el protocolo de descubrimiento de capa de enlace. Los controladores locales se comunican con el controlador global para intercambiar periódicamente información sobre su dominio y solicitar la selección de ruta entre dominio. Además de mantener la información de todos los controladores locales, el controlador global considera cada dominio como un conmutador virtual y administra un gráfico de red que representa las cone-

[13]estructura de datos con nodos conectados por aristas con un flujo direccional y sin ciclos.

[14]representa un proceso estocástico en el que la expectativa del valor futuro, dada la información presente y pasada, permanece constante a lo largo del tiempo [94].

[15]tiempo de transmisión efectivo para transmitir un flujo de datos.

xiones entre ellos. Para la evaluación implementaron una simulación donde realizaron comparaciones de rendimiento con el algoritmo de Dijkstra y DOLPHIN [95], en tres topologías de red diferentes. Aunque el método propuesto es capaz de satisfacer los requisitos de QoS, no utiliza técnicas de RL.

**Hierarchical SDN Multi-controller Placement Strategy Based on Improved Aquila Optimizer [36]**. Sugiere una estrategia de *Multi-Controller Placement Problem* [96] para una SDN jerárquica basada en AO (*Aquila Optimizer* [36]) mejorado, con los objetivos de minimizar la latencia de la red y lograr el equilibrio de carga del controlador local. Debido a la débil capacidad de exploración global, baja precisión de optimización y lenta velocidad de convergencia de AO, esta estrategia mejora la diversidad de los esquemas de ubicación inicial de los múltiples controladores SDN aplicando el aprendizaje basado en oposición estocástica a la fase de inicialización de AO. En la fase de exploración global, se añade una *Opposition-Based Learning* [97] dinámica para mejorar la diversidad de los planes de colocación y evitar perder mejores esquemas de colocación. Finalmente, utiliza una estrategia de caminata gaussiana en la etapa de desarrollo local para establecer la mejor opción de colocación a partir de una variedad de mejores esquemas. Los datos experimentales sugieren que la estrategia dada es más ventajosa con respecto a los algoritmos aleatorios K-means [98] y *Particle swarm optimization* [99], resolviendo el esquema de ubicación de múltiples controladores de la arquitectura jerárquica con menor latencia y equilibrio de carga, pero con la limitación de no ser consciente al QoS.

### 2.4.2 Brechas

La tabla 2.1 brinda un resumen de los artículos analizados en la Sección 2.4.1. La primera columna proporciona la referencia bibliográfica de cada artículo. La siguiente columna, describe las técnicas implementadas y el algoritmo utilizado, si está disponible. Las métricas empleadas en cada estudio se detallan en la tercera columna. Por último, la cuarta columna describe el resultado, haciendo referencia al tipo de simulación utilizado en cada caso.

Los trabajos en [10, 23–26, 30, 33, 34, 36] utilizan el algoritmo de enrutamiento de ruta más corta, que no puede aprender las rutas o elegir la ruta más adecuada para realizar un enrutamiento eficiente. Además, los trabajos anteriores no utilizan métricas que satisfagan QoS, por lo que la red no podrá proporcionar un alto rendimiento o los servicios necesarios para un tráfico de red particular [100]. Las investigaciones en [27, 28, 31, 32, 35] utilizan métricas de QoS (*delay, packet loss, bandwidth* y *throughput*). Sin embargo, estas investigaciones tienen la misma limitación que los trabajos anteriores (algoritmos de enrutamiento basados en el camino más corto), que no eli-

| Ref | Técnica | Métricas | OutPut |
|---|---|---|---|
| [10] | Constrained Shortest Path First | Bandwidth, delay and jitter | Experimentos numéricos y simulación |
| [23] | Dijkstra | Índice de entrega de paquetes y tiempo de ida y vuelta | Simulación |
| [24] | Asignación de recursos optimizada | Tiempo de respuesta y tasa media de llegada | Experimentos numéricos |
| [25] | Border node pairs | Número de mensajes | Simulación |
| [26] | Google B4 (TE) | Delay y Bandwidth | Implementación de red real |
| [27] | DRSFI | Packet loss, throughput y delay | Simulación |
| [28] | Dijkstra | Delay, throughput y packet loss | Simulación |
| [29] | Dijkstra | Tasa de entrega de paquetes, *delay* promedio, *packet loss*, sobrecarga de enrutamiento y tasa de fallas | Simulación |
| [30] | Google B4 (TE) | Bandwidth y delay | Inplementación de red real |
| [31] | Aplicación de enrutamiento | Throughput, bandwidth, packet loss, y delay | Simulación |
| [32] | Integer Linear Programming | Throughput, packet loss y bandwidth | Simulación |
| [33] | Central Scheduler Load Balancing | Tiempo medio de respuesta y tiempo medio de espera | Simulación |
| [34] | Conteo de saltos | Tasa de entrega de paquetes y *delay* de extremo a extremo | Experimentos numéricos |
| [35] | Directed Acyclic Graph | Bandwidth y delay | Simulación |
| [36] | Multi-Controller Placement Problem basado en AO | Latencia y equilibrio de carga | Simulación |
| [38] | RL (Shallow y deep Artificial Neural Networks) | *Packet loss, delay* y *throughput* | Simulación |
| [39] | RL (Q-learning) | *Packet loss, delay* y *throughput* | Implementación de red real |

Tabla 2.1: Brechas de investigación.

gen la mejor ruta para realizar un enrutamiento eficiente.

Por otro lado, los enfoques [38, 39] usan algoritmos de enrutamiento basados en RL y métricas conscientes de QoS para encontrar rutas óptimas. Sin embargo, estas soluciones se limitan al uso de un único agente RL para realizar el enrutamiento de red inteligente. El presente trabajo de grado propone IQRSMR (**I**ntelligent and **Q**os-Aware **R**outing in Hierarchical **S**DN using **M**ulti-Agent **R**einforcement Learning) considerando las métricas de *packet loss, delay* y *throughput*.

# Capítulo 3

# Diseño de Enrutamiento en una Red Jerárquica Definida por Software

Este capítulo aborda en profundidad la arquitectura de una red jerárquica, que sirve como base fundamental para construir todo el marco de comunicación y transferencia de datos. Además, se analiza detalladamente los diversos elementos que componen la estructura de la red, incluyendo la topología y los diferentes controladores.

Además, esta sección explora el método IQRSMR basado en el enfoque MARL, este último ofrece una mejora considerable en la eficiencia del aprendizaje, al abordar de manera efectiva el desafío de la no estacionariedad. En comparación con enfoques de RL y DRL de un solo agente que no consideran la información proveniente de otras entidades en la red, MARL permite que estas desarrollen una política más estable y robusta. Este enfoque, junto con la aplicación del algoritmo Q-learning, se emplea para determinar la política óptima de selección de acciones mediante la utilización de una función $Q$. La combinación de estos elementos potencia tanto el rendimiento como la eficiencia de la red. Esto es posible debido a que el algoritmo Q-learning es simple de entender e implementar, además es apropiado para aprender funciones de valor en problemas discretos y se basa en un enfoque libre de modelo, eliminando la necesidad de un conocimiento completo del entorno para tomar decisiones.

Este capitulo estudiará minuciosamente el proceso de entrenamiento de IQRSMR, su capacidad para actualizar y almacenar valores $Q$ y cómo la interacción entre los distintos agentes conduce a la búsqueda de las rutas óptimas, considerando las métricas de QoS de los enlaces de red.

## 3.1 Arquitectura Jerárquica

La Figura 3.1 muestra la arquitectura de red jerárquica diseñada para ofrecer enrutamiento inteligente en entornos SDN. Este tipo de arquitectura ofrece una serie de ventajas, como la mejora en la escalabilidad, rendimiento y gestión de redes complejas. Esta sección profundiza en las características fundamentales de esta arquitectura, que incluyen la organización en capas, la distribución de funciones, el papel del controlador raíz SDN en la gestión centralizada y cómo estos componentes trabajan en conjunto para optimizar el tráfico y proporcionar un control integral sobre la red.

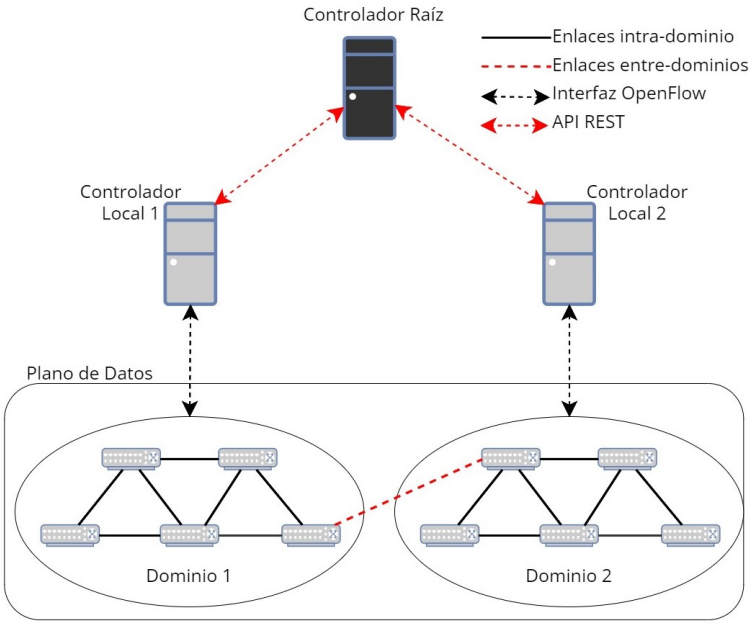



Figura 3.1: Arquitectura SDN jerárquica.

La arquitectura se compone de tres elementos clave para el funcionamiento eficiente de la red, los cuales son:

### 3.1.1 Plano de Datos

Esta compuesto por cada uno de los dominios que serán administrados por su respectivo controlador local. A su vez, estos dominios están compuestos por conmutadores y enlaces que los conectan, enfocados en la recepción y en el reenvío de paquetes a través de flujos.

El tráfico se mueve de acuerdo con la información de ruta instalada en las tablas de flujo de los conmutadores; estos funcionan sin conocimiento del resto de la red y dependen de los planos de control y conocimiento para completar e instalar sus tablas de flujo

[101, 102]. El Plano de Datos proporciona constantemente información de la red al plano de control a través del protocolo Openflow. Esta información incluye la versión del protocolo, el estado y las estadísticas de los puertos, el rol de los conmutadores (esclavo, maestro) y detalles sobre la conexión entre el controlador y cada conmutador.

### 3.1.2 Controlador Local

La Figura 3.2 muestra la arquitectura del controlador local, encargado de llevar a cabo el enrutamiento dentro de su respectivo dominio, debido a que solo dispone de una vista parcial de la red. Este controlador establece comunicación tanto con el plano de datos como con el controlador raíz. En la comunicación hacia el sur (SBI)[1] con el plano de datos, utiliza una interfaz OpenFlow para instalar las reglas de enrutamiento en los conmutadores. Por otro lado, en la comunicación hacia el norte (NBI)[2] con el controlador raíz, utiliza solicitudes HTTP[3] (*Hypertext Transfer Protocol*) para proporcionarle información sobre su dominio, ya que el controlador raíz posee una vista global de la red. El controlador local está compuesto por:

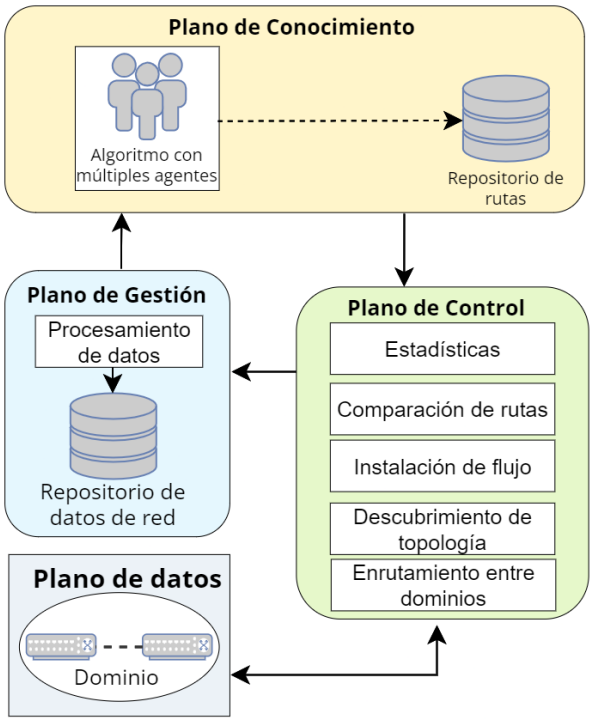



Figura 3.2: Controlador local.

[1]permite que un componente de nivel superior se comunique con los de nivel inferior [103].
[2]permite que un componente de red de nivel inferior se comunique con un componente de nivel superior o más central [103].
[3]protocolo cliente-servidor de la capa de aplicación.

***Plano de Control*** tiene como objetivo proporcionar una vista del respectivo dominio ubicado en el Plano de Datos al recopilar información de los conmutadores. Toma decisiones sobre el enrutamiento y los cambios que deben realizarse en las tablas de los conmutadores en el Plano de Datos [101]. Para lograrlo, el Plano de Control se compone de los siguientes módulos:

- **Descubrimiento de Topología:** Envia mensajes LLDP[4] (*Link Layer Discovery Protocol*) a los conmutadores en el Plano de Datos. A su vez, los dispositivos responden enviando mensajes que contienen información como la identificación, la cantidad de puertos y su estado. Utilizando la información recibida, el módulo infiere la topología de la red al relacionar el puerto de cada conmutador con los puertos de los conmutadores vecinos y los hosts conectados a cada uno de ellos.

- **Estadísticas:** Su función principal es mantener información estadística en tiempo de ejecución, la cual será procesada por el Plano de Gestión. Para ello, envía mensajes de estado de solicitud *multipart-request* con tipo *port-desc* [66] (mensaje que solicita estadísticas de todos los puertos de los conmutadores en la red) a cada conmutador en el Plano de Datos cada $t$ segundos y a su vez, recibe de forma asíncrona los mensajes de respuesta *multipart-reply* con tipo *port-desc* [66] (mensaje que entrega información sobre los puertos de los conmutadores enviados al controlador).

- **Instalación de Flujo:** Se lleva a cabo de manera proactiva, completando las tablas de flujo de los conmutadores con todas las posibles rutas antes del intercambio de tráfico, tanto para el enrutamiento intra-dominio como para el enrutamiento entre-dominios. La interfaz hacia el sur OpenFlow, permite realizar la instalación de las rutas calculadas por IQRSMR en las tablas de flujo [105], a través del mensaje *flow-mod* [66] (mensaje que envía el controlador para realizar modificaciones en las tablas de flujo). Es importante tener en cuenta que una entrada de flujo incorrecta puede provocar el envío de tráfico hacia rutas muy utilizadas, lo que a su vez ocasionaría congestión en la red.

- **Comparación de Rutas:** Tiene la función de almacenar una copia de las rutas actuales de la red para todos los pares origen-destino del dominio correspondiente. Posteriormente, compara las rutas almacenadas con las rutas recién calculadas y enviadas por el módulo de conocimiento. Su objetivo principal es identificar y seleccionar únicamente las rutas que difieren de las rutas previas, evitando así introducir información redundante en la red. Al realizar esta comparación, este módulo permite un enrutamiento más eficiente y evita sobrecargar la red con

[4] mensajes utilizados por el controlador para descubrir y actualizar la topología de red [104].

información que ya está presente en ella y no ha cambiado desde la última actualización. De esta manera, se logra mejorar el rendimiento general del sistema de enrutamiento.

- **Enrutamiento Entre Dominios:** Es el encargado de mantener una comunicación continua con el controlador raíz mediante peticiones HTTP. Constantemente cada controlador local envía peticiones POST con información de su respectivo dominio (nodos, puertos, enlaces, *throughput*, *delay* y *packet loss*). También este módulo es el encargado de obtener las rutas intra-dominio del controlador raíz mediante una petición GET.

***Plano de Gestión*** tiene como objetivo garantizar que la red en su conjunto funcione de manera óptima mediante la comunicación con el Plano de Control [106]. Contiene el módulo de Procesamiento de Datos y el Repositorio de Datos de Red.

- **Procesamiento de Datos:** Tiene como función recuperar y utilizar los datos sin procesar, los cuales han sido previamente recopilados por los módulos de estadísticas y descubrimiento de topología del Plano de Control y son utilizados para llevar a cabo los cálculos del *packet loss, delay* y *throughput* descritos en la Sección 3.3.3. Con base en estas métricas, el Plano de Conocimiento puede tomar decisiones más inteligentes y optimizar la eficiencia de las comunicaciones entre los diferentes conmutadores y dominios.

- **Repositorio de Datos de Red:** Almacena las métricas calculadas por el módulo de procesamiento de datos. Este repositorio contiene un conjunto de entradas que representan los pares de conmutadores interconectados y las métricas correspondientes para todos los enlaces entre conmutadores. Un ejemplo de una entrada en este repositorio sería el siguiente: {”Conmutador_fuente”: 1, ”Conmutador_destino”: 2, ”Throughput”: 500Kbps, ”Delay”: 1.3ms, ”Packet_loss”: 0.5 %}. La información del estado de enlace es una entrada al plano de conocimiento tanto del controlador local como del raíz, ya que este último necesita dicha información para crear la topología de red global.

***Plano de Conocimiento*** aprende el comportamiento de su dominio y utiliza inteligencia a través de RL para tomar decisiones. Esto se logra gracias a los datos recopilados por el Plano de Gestión, los cuales proporcionan información sobre el estado de los enlaces [107]. Una vez son calculadas las rutas óptimas para todos los pares origen-destino del dominio correspondiente, se envían al Plano de Control para su correspondiente instalación. Este plano contiene a IQRSMR y el Repositorio de Rutas.

- **IQRSMR:** Se emplea para completar el Repositorio de Rutas calculando la ruta óptima para cada par origen-destino. Cada IQRSMR utiliza el algoritmo 3.1

para aprender la mejor política de enrutamiento. Durante el proceso, explora todos los estados de acción posibles para que una fuente alcance su destino. Al mismo tiempo, actualiza su política de decisión basada en la función de valor $Q$ estado-acción y obtiene la recompensa correspondiente, la cual indica el impacto de ejecutar una acción $A_t$ (elegir un conmutador vecino como siguiente salto) en el estado $E_t$ (conmutador actual). La recompensa obtenida por cada agente que compone a IQRSMR, depende de una de las métricas de QoS, ya sea *packet loss, delay* o *throughput* (Sección 3.3.4). Este enfoque permite mejorar significativamente la eficiencia del enrutamiento, ya que aprende y adapta su comportamiento para optimizar el flujo de datos en la red.

- **Repositorio de Rutas:** Almacena un conjunto de datos con información sobre las rutas entre todos los pares origen-destino. Cada entrada en este repositorio está representada como una tupla que contiene la secuencia óptima de conmutadores que forman la ruta para un par específico. Por ejemplo, el repositorio de ruta sigue el formato: {"rutas": [[1, 5, 2], [2, 6, 7, 3], [3, 11, 5, 10],...]}, donde en el primer elemento, indica la primera ruta:
  - 1 es el número del conmutador de origen.
  - 2 es el número del conmutador de destino.
  - 5 es el número del conmutador intermedio, por el cual se debe pasar para establecer la comunicación entre el origen y el destino. Pueden existir múltiples conmutadores que interconectan origen y destino.

  Cada una de estas rutas se envía al Plano de Control, para luego pasar por los módulos de comparación de rutas e instalación de flujo descritos anteriormente.

### 3.1.3 Controlador Raíz

La Figura 3.3 muestra la arquitectura del controlador raíz, el cual tiene como función principal el enrutamiento entre-dominios. Este controlador se comunica con los controladores locales para obtener tanto la topología como la información de red de cada uno de los dominios. De esta manera, adquiere una visión global de la red, lo que le permite calcular la ruta de los flujos entre dominios y enviarla a los respectivos controladores locales. Estos, a su vez, se encargan de instalar la ruta en el Plano de Datos. El controlador raíz esta compuesto por:

***Plano de Control*** tiene como objetivo proporcionar al Plano de Conocimiento una visión global del Plano de Datos, al recopilar información sobre el dominio de cada controlador local. Para lograrlo, el Plano de Control se compone del siguiente módulo:

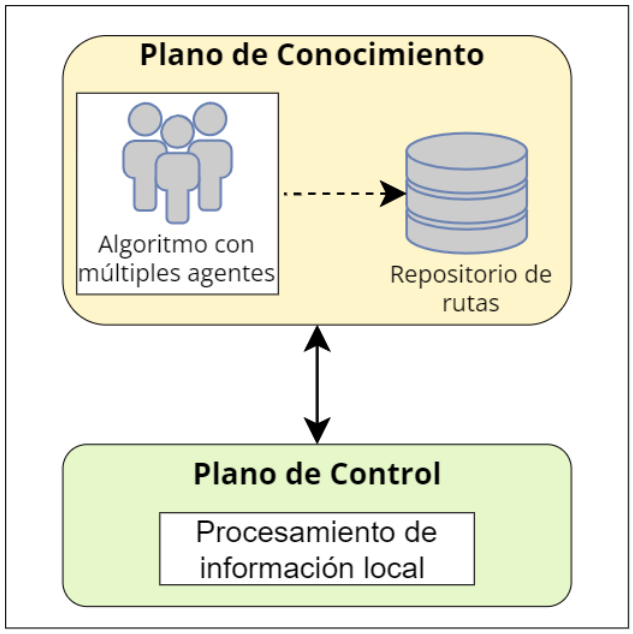



Figura 3.3: Controlador raíz.

- **Procesamiento de Información Local:** Proporciona los datos necesarios para mejorar la eficiencia del enrutamiento global en toda la red. Esto mediante la recopilación de los datos de red y de la topología de cada controlar local. Utilizando dicha información, crea un mapa global de la red que permitirá al Plano de Conocimiento realizar el cálculo óptimo de las rutas globales.

***Plano de Conocimiento*** aprende el comportamiento de toda la red y utiliza inteligencia a través de RL para tomar decisiones. Esto se logra gracias a los datos obtenidos por el módulo de procesamiento de información local en el Plano de Control. Este plano contiene a IQRSMR y el Repositorio de Rutas.

- **IQRSMR:** Similar al controlador local, completa el Repositorio de Rutas. En este caso, el algoritmo calcula únicamente las rutas óptimas entre los conmutadores de diferentes dominios.

  Cabe destacar que los agentes que componen IQRSMR no mantienen comunicación entre sí. Esto con el propósito de simplificar tanto el diseño como la implementación del algoritmo. La ausencia de comunicación elimina la necesidad de definir protocolos y reglas específicas sobre cómo deben interactuar entre sí. Además conlleva la ventaja de reducir el costo computacional, ya que no es necesario que los agentes compartan información ni coordinen sus acciones.

- **Repositorio de Rutas:** Almacena las rutas entre dominios para todos los pares

origen-destino. Similar que en el controlador local, la ruta está representada como una tupla que contiene la secuencia óptima de conmutadores.

## 3.2 Enrutamiento

El enrutamiento es un elemento esencial para garantizar el funcionamiento óptimo de cualquier sistema de red, dado que se encarga de dirigir el tráfico de datos de manera eficiente desde su origen hasta el destino. Esta sección describe de manera detallada los procedimientos tanto para el enrutamiento intra-dominio como para entre-dominios. Estos enrutamientos varían según la ubicación del conmutador destino y son fundamentales para asegurar una comunicación efectiva y confiable entre los conmutadores de la red.

### 3.2.1 Enrutamiento Intra-Dominio

La Figura 3.4 muestra el proceso de enrutamiento, gestionado por el controlador local de cada dominio y explicado en detalle a continuación:

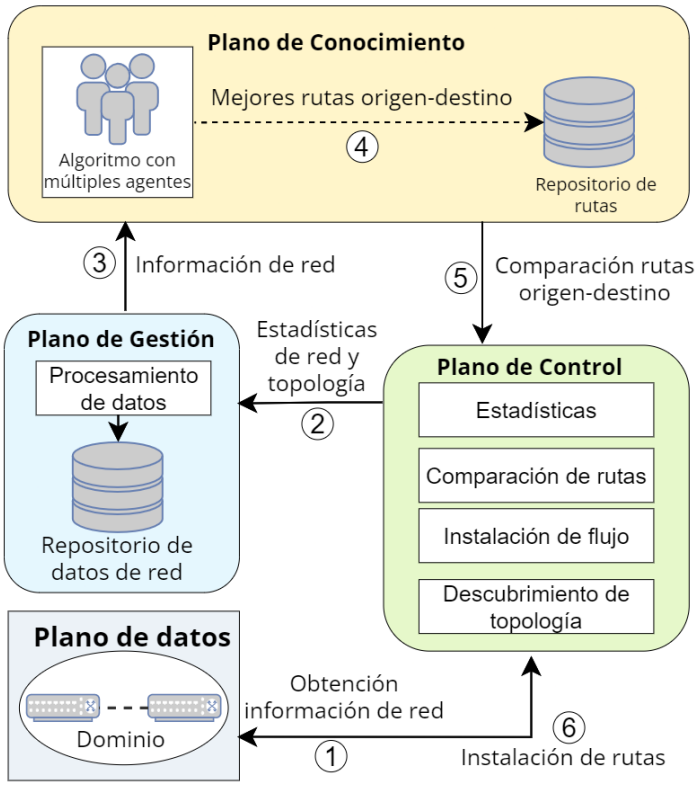



Figura 3.4: Enrutamiento intra-dominio.

1. El Plano de Control envía solicitudes OpenFlow al Plano de Datos mediante los módulos de descubrimiento de topología y estadísticas, solicitando información del respectivo dominio.

2. Esta información sin procesar es posteriormente enviada al Plano de Gestión. Aquí, se realiza el procesamiento de los datos, incluido el cálculo del *packet loss, delay* y *throughput*. Una vez obtenidas todas las métricas de cada uno de los enlaces de la topología, se almacenan en el repositorio de datos de red.

3. Esta información ya procesada es enviada al Plano de Conocimiento, representando una entrada en IQRSMR. Este último se encarga de calcular las rutas óptimas para cada par origen-destino, teniendo en cuenta las métricas de QoS enviadas por el Plano de Gestión.

4. Una vez calculadas las rutas óptimas, estas son almacenadas en el repositorio de rutas, y desde allí se envían al Plano de Control.

5. En el Plano de Control, son comparadas las rutas recién calculadas con las existentes en el Plano de Datos utilizando el módulo de comparación de rutas. Esto evita la introducción de información redundante a la red. Si alguna de las rutas recién calculadas coincide con una ruta existente, no se instalará en el correspondiente conmutador, si no que se mantiene la regla de enrutamiento existente. Por otro lado, si alguna de las rutas calculadas difiere de las rutas existentes, se procederá a su respectiva instalación.

6. El módulo de instalación de rutas es el encargado de enviar las rutas que difieren de las actuales a los respectivos conmutadores en el Plano de Datos, lo que permite una correcta gestión de los recursos de la red.

### 3.2.2 Enrutamiento Entre-Dominios

La Figura 3.5 muestra el proceso de enrutamiento, que involucra tanto a los controladores locales como al controlador global, y se explica en detalle a continuación:

A) Al igual que en el enrutamiento intra-dominio, los controladores locales envían peticiones al Plano de Datos mediante mensajes OpenFlow para obtener información de sus dominios, la cual es procesada en el Plano de Gestión.

B) El Plano de Gestión de cada controlador local envía la información de su dominio al controlador raíz, el cual tiene una vista global de toda la red.

C) La información de cada dominio llega al Plano de Control del controlador raíz para su procesamiento y posteriormente es enviada al Plano de Conocimiento

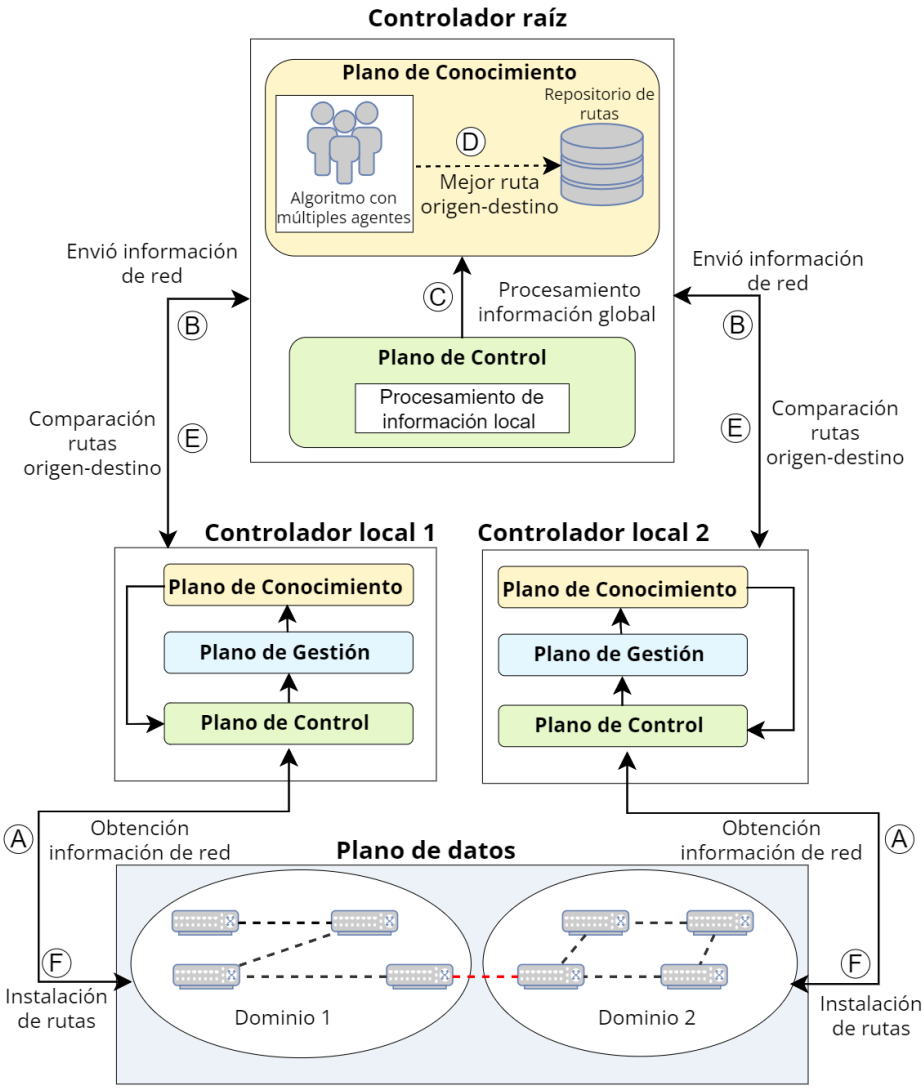



Figura 3.5: Enrutamiento entre-dominios.

del controlador raíz, donde representa una entrada en IQRSMR raíz. Este último utiliza el algoritmo Q-learning para calcular las rutas óptimas para todos los pares origen-destino entre dominios.

D) Una vez que se han calculado las rutas entre dominios, estas son almacenadas en el repositorio de rutas del Plano de Conocimiento.

E) Cada controlador local mediante el modulo de enrutamiento entre dominios, envía peticiones GET para obtener las rutas almacenadas en el repositorio de rutas del controlador raíz. Una vez que los controladores locales tienen las rutas óptimas, el módulo de comparación de rutas verifica si las rutas obtenidas son iguales con las rutas existentes en las tablas de flujo. Al igual que en el enrutamiento intra-

dominio, si la ruta global difiere de la ruta en el Plano de Datos, se procede a su respectiva instalación.

F) El módulo de instalación de rutas las envía a los conmutadores mediante mensajes OpenFlow, permitiendo así el enrutamiento entre-dominios.

## 3.3 Diseño de IQRSMR

El enfoque IQRSMR utiliza la técnica Q-learning para determinar las rutas de los flujos. Esta técnica está diseñada para encontrar una política óptima en entornos con toma de decisiones secuenciales. Propuesto inicialmente por Watkins, este algoritmo se ha convertido en una herramienta fundamental en el campo de ML y la inteligencia artificial.

Al ser una técnica sin modelo, Q-learning no requiere conocimiento previo de las recompensas asociadas a acciones específicas en estados particulares [108]. El objetivo es aproximar la función de valor de acción óptima $Q_t(S_t, A_t)$, conocida como función $Q$, que representa la calidad de tomar una acción específica en un estado determinado. Los valores $Q$ se actualizan iterativamente a medida que IQRSMR interactúa con el entorno, lo que permite la convergencia gradual hacia la política óptima. Durante esta interacción, cada agente recibe recompensas por sus acciones y ajusta sus decisiones futuras para minimizar el costo acumulado a lo largo del tiempo, considerando un número predefinido de episodios independientes [109].

Un episodio de aprendizaje consiste en una secuencia de pasos que conducen a IQRSMR desde un estado inicial hasta un estado objetivo [110]. En cada paso, los agentes seleccionan y ejecutan una acción, pasan de un estado a otro y reciben una recompensa [111]. El valor actualizado de la función $Q$ es calculado en función de la recompensa obtenida al ejecutar la acción $A_t$ en el estado $E_t$, para cada uno de los agentes. A continuación, se detalla sobre los componentes de IQRSMR.

### 3.3.1 Espacio de Estados

El espacio de estados es el conjunto de todos los estados posibles en los que puede estar un sistema [112]. En este contexto, cada estado corresponde a un conmutador en el Plano de Datos, y una transición de un estado a otro equivale a un enlace que conecta dos conmutadores vecinos, como se observa en la Figura 3.6. Por lo tanto, la topología del Espacio de Estado ($E$) corresponde a la topología de los conmutadores en el Plano de Datos, se representa como:

$$|E = \{E_i\}| \equiv N \tag{3.1}$$

Dónde:

$N$: número de conmutadores en el Plano de Datos.

### 3.3.2 Espacio de Acciones

El espacio de acciones es el conjunto de todas las acciones $A$, que se pueden realizar sobre los estados del Espacio de Estados [112]. Para cada estado, $Ei \in E$, el sistema IQRSMR tiene la capacidad de elegir una acción de entre un conjunto de opciones disponibles. Cada una de estas acciones lleva hacia otro estado $Ej \in E$. Se entiende por acción, la transición de un estado a otro estado vecino. Los estados vecinos del estado $Ei$ son los conmutadores vecinos en el Plano de Datos. El número de acciones posibles en un estado $Ej$, es el grado de nodo del estado en cuestión $Gn(Ei)$ y se representa como:

$$|A| \equiv \sum_{Ei \in E} Gn(Ei) \tag{3.2}$$

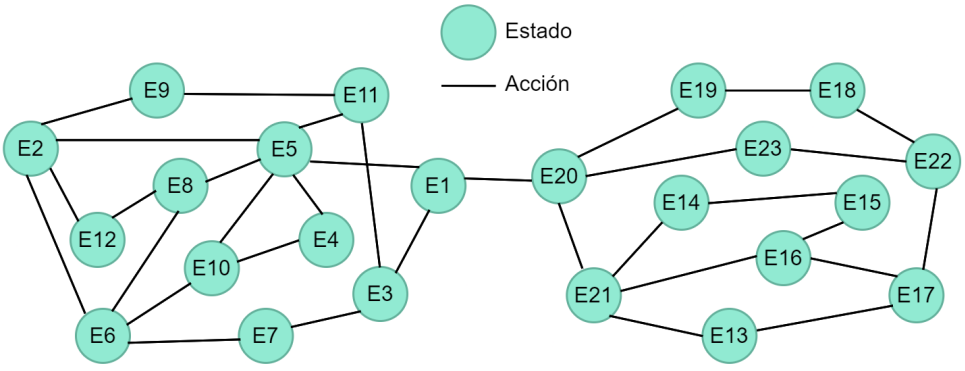



Figura 3.6: Espacio de Estados.

### 3.3.3 Métricas de Calidad de Servicio

Esta sección expone las ecuaciones para determinar las métricas de QoS usadas en la Sección 3.3.4. Para el cálculo de *packet loss* y *throughput* del enlace se realiza la evaluación de la cantidad de paquetes que han transitado a través del puerto del conmutador asociado con el enlace en consideración. En cada uno de estos puertos, el controlador SDN realiza un muestreo periódico para determinar la cantidad de bytes

que se han transmitido y recibido.

La Ecuación 3.3 muestra el *throughput* instantáneo, el cual es calculado al comparar los valores recuperados en dos instantes diferentes. $Th_{t1}$ representa la cantidad de bytes recibidos, esta información es obtenida como respuesta al enviar una solicitud de estado (*multipart-request / multipart-reply*) al plano de control en un instante $t_1$. Luego, tras un intervalo de tiempo $p$ (periodo de muestreo), se emite una nueva solicitud de estado y del mensaje de respuesta se recupera el número de bytes recibidos representado como $Th_{t2}$.

$$Th_{link} = \frac{Th_{t2} - Th_{t1}}{p} \tag{3.3}$$

La Ecuación 3.4 presenta el cálculo de la tasa instantánea de *packet loss*, utilizando los datos de los mensajes de respuesta de las solicitudes de estadísticas. En esta ecuación, $Bt_{t1}$ representa el número de bytes transmitidos en el instante $t_1$, mientras que $Br_{t2}$ refleja el número de bytes recibidos después de un intervalo de tiempo $p$. Esta información se obtiene como respuesta a las solicitudes de estado enviadas al plano de control.

$$Pl_{link} = \frac{Bt_{t1} - Br_{t2}}{Bt_{t1}} \tag{3.4}$$

La Ecuación 3.5 representa el cálculo del *delay* instantáneo en el enlace $(si, sj)$, el cual se basa en el proceso descrito en [104], haciendo uso tanto de los mensajes LLDP como de los mensajes de OpenFlow. En este contexto, $Dlldp_t$ representa el tiempo del paquete LLDP enviado por el controlador SDN $c0$, el cual sigue la ruta $c0 - si - sj - c0$. En donde $si$ y $sj$ son conmutadores conectados por el enlace $(si, sj)$. Además, $D_{c0-si}$ denota el tiempo que demora el mensaje en viajar de $c0$ al conmutador $si$ $(c0 - si)$, y se estima como el tiempo transcurrido entre la transmisión y la recepción de los mensajes OpenFlow *echo-request* y *echo reply*[5]. De manera similar, para la estimación del tiempo que demora el mensaje en transitar de $c0$ a $sj(D_{c0-sj})$, se sigue un procedimiento similar al empleado de $c0$ a $si$. Este método permite el cálculo del *delay* en el enlace $(si, sj)$ mediante una consideración exhaustiva de los tiempos involucrados en la transmisión y recepción de los paquetes, a través de rutas especificas.

$$Dl_{link} = Dlldp_t - (\frac{D_{co-si} + D_{co-sj}}{2}) \tag{3.5}$$

[5]los mensajes echo son usados para conocer el estado de la conexión controlador-conmutador o para medir el delay y el ancho de banda.

### 3.3.4 Función de Recompensa

La función de recompensa se utiliza para determinar la mejor acción posible dentro del Espacio de Acciones. En este contexto, cada agente posee una recompensa asociada a una de las métricas de QoS (*packet loss, delay* y *throughput*) descritas en la Sección 3.3.3, ya que cada uno de ellos utiliza una métrica específica para identificar las opciones óptimas de enrutamiento.

La Ecuación 3.6 muestra la recompensa del agente encargado del *throughput*. En esta configuración, a medida que el *throughput* del enlace aumenta, el costo del enlace disminuye. De esta manera, el agente está incentivado a seleccionar rutas con mayor *throughput* y así priorizar la utilización eficiente de los recursos disponibles, para mejorar el sistema en general.

$$R_{throughput} = \frac{1}{Th_{link}} \tag{3.6}$$

La Ecuación 3.7 muestra la recompensa del agente encargado del *delay*. De esta forma, a medida que el *delay* del enlace aumenta, el costo del enlace también lo hace. Por ende, el agente es motivado a seleccionar rutas que minimicen la recompensa asociada al enlace.

$$R_{delay} = Dl_{link} \tag{3.7}$$

La Ecuación 3.8 muestra la recompensa del agente encargado del *packet loss*. De manera similar a lo que ocurre con el *delay* del enlace, a medida que el *packet loss* del enlace aumenta, el costo del enlace también lo hace. Por tanto, el agente se encarga de priorizar la elección de rutas más estables y con menor probabilidad de *packet loss*.

$$R_{loss} = Pl_{link} \tag{3.8}$$

Para garantizar que cada una de las métricas utilizadas, las cuales tienen unidades diferentes, tengan la misma influencia y estén dentro del mismo rango arbitrario, se utilizo la técnica de normalización Min-Max [113]. Esta normalización realiza una transformación lineal de los datos originales, mapea un valor individual $x_i$ del conjunto a normalizar $X$ a $\hat{x_i}$ en el rango $[a, b]$, siendo $a$ el nuevo mínimo (1) y $b$ el nuevo máximo (10). La normalización Min-Max (Ecuación 3.9) es utilizada para que todas las métricas tengan una escala comparable, evitando que alguna de ellas domine sobre las demás debido a sus unidades de medida. Al mapear los valores originales en un rango en común, se facilita el proceso de toma de decisiones en el enrutamiento.

$$\hat{x_i} = a + \frac{(x_i - min(X)) \cdot (b - a)}{max(X) - min(X)} \tag{3.9}$$

### 3.3.5 Política Óptima

La política óptima ha sido diseñada para minimizar el valor de la recompensa en el proceso de enrutamiento de Q-learning. Mediante este enfoque, IQRSMR aprende a evitar enlaces con un alto *packet loss* y *delay*, y al mismo tiempo prioriza aquellos enlaces con un amplio *throughput*, cuando selecciona una ruta.

Para cada uno de los agentes que conforman a IQRSMR, el valor $Q$ representa una medida de la recompensa general esperada cuando cada agente se encuentra en un estado $E_t$ y realiza una acción $A_t$.

$$Q_{t+1}(E_t, A_t) = Q_t(E_t, A_t) + \alpha \cdot [R_t + \gamma \text{min} Q_t(E_{t+1}, A_t) - Q_t(E_t, A_t)] \tag{3.10}$$

Cada agente utiliza la Ecuación 3.10, para actualizar su respectivo valor de $Q$. En esta ecuación, $Q_{t+1}$ depende de la estimación actual de la función $Q$ para un par estado-acción $Q_t(E_t, A_t)$, y de una estimación futura $Q_t(E_{t+1}, A_t)$. Esta última se basa en el mínimo costo de pasar de un estado $Ei \in E$ a otro $Ej \in E$ mediante una acción $A_t$. La ecuación también es influenciada por el parámetro $\alpha \in [0, 1]$, el cual representa la tasa de aprendizaje, que determina el peso de la información recién obtenida en relación con la información anterior. Un valor de $\alpha = 0$ implica que el agente RL no aprende del último par $(E_t, A_t)$, mientras que un valor de $\alpha = 1$ permite que el agente mantenga la información aprendida considerando la recompensa inmediata $R_t$ para el par $(E_t, A_t)$. $R_t$ representa la recompensa de ejecutar una acción $A_t$ en un estado $E_t$. Esta recompensa depende de cada una de las métricas de QoS. Por otro lado, el parámetro $\gamma \in [0, 1]$ representa el factor de descuento, que determina la importancia de las recompensas futuras. Si $\gamma$ tiene un valor cercano a $0$, el agente solo considerará las recompensas actuales. En cambio, un valor cercano a $1$ hará que el agente se esfuerce por obtener una alta recompensa a largo plazo.

$$Q_t(E_t, A_t) = \frac{1}{N} \sum_{k=1}^{N} Q_k(E_t, A_t) \tag{3.11}$$

Después de que cada agente haya actualizado y almacenado su respectivo valor $Q$, con la Ecuación 3.11 basada de [114], se procede a calcular el promedio de los valores $Q$ obtenidos por todos los agentes, en donde $N$ representa la cantidad de agentes. Una vez obtenido el resultado, se determina la ruta óptima entre los conmutadores de interés.
Este procedimiento con múltiples agentes se diseñó para mitigar posibles conflictos asociados con la determinación de rutas. En la ausencia de este promedio, cada agente generaría una ruta, resultando en tres rutas potenciales, las cuales podrían presentar variaciones y generar incertidumbre respecto a cuál seleccionar. En cambio, mediante

este proceso, se genera únicamente una ruta que considera a cada agente y, por ende, a todas las métricas de QoS. Este enfoque previene conflictos entre los agentes, ya que proporciona una solución que considera de igual manera los objetivos de cada uno.

### 3.3.6 Exploración y Explotación

En Q-learning, se presenta un dilema entre elegir la acción codiciosa para obtener la mayor recompensa según los valores actuales del agente (explotación) y seleccionar una acción diferente con la esperanza de obtener mejores beneficios en el futuro (exploración) [115]. IQRSMR utiliza el método de exploración y explotación $\varepsilon$-greedy, el cual toma un $\varepsilon \in [0, 1]$ como parámetro ajustable que le permite a cada agente explotar y explorar.

Cada agente utiliza este método para tomar decisiones y seleccionar la siguiente acción en un estado específico. En cada paso, se genera un valor aleatorio $x \in [0, 1]$. Si $x < \varepsilon$, el agente explota, es decir, elige la acción con la mejor recompensa estimada hasta el momento. Por otro lado, si $x \geq \varepsilon$, explora y selecciona una acción al azar, permitiendo descubrir nuevas posibilidades y mejorar su conocimiento del entorno.
Al ajustar el valor de $\varepsilon$, se puede controlar el equilibrio entre la explotación y la exploración en el proceso de aprendizaje de IQRSMR. Por lo tanto, este enfoque permite que cada agente aprenda y tome decisiones efectivas basándose en sus experiencias pasadas. Además, brinda la capacidad de buscar nuevas estrategias con el objetivo de mejorar la recompensa a largo plazo.

## 3.4 Algoritmo de Enrutamiento

El algoritmo de enrutamiento tiene la responsabilidad de determinar la ruta óptima para cada par de conmutadores en un sistema de red específico. El algoritmo 3.1 toma como entrada tanto la topología de la red, como la información del estado de cada enlace. Como resultado de su ejecución, se obtiene un conjunto que contiene las rutas de enrutamiento óptimas para todos los pares de conmutadores existentes dentro de la red.

Para cada par de conmutadores (origen, destino), el algoritmo realiza un proceso de Q-learning siguiendo los pasos definidos en las Líneas 1 a 12. En el marco de IQRSMR, se inicializan las tablas de la función $Q$ $Q(E, A)$ y de la recompensa $R(E, A)$ de cada agente con un valor de cien (Línea 2) para evitar que tomen una acción indeseada, ya

que las acciones posibles están representadas con valores bajos. El proceso de entrenamiento para cada valor $Q$ comienza desde un conmutador aleatorio como estado inicial (Línea 4).

Cada agente se entrena en paralelo para minimizar el valor $Q$ mediante el uso de su respectiva función de recompensa. Los agentes atribuyen valores $Q$ bajos a las rutas formadas ya sea por enlaces con un gran *throughput*, *packet loss* pequeños o *delay* cortos, según corresponda. El algoritmo atraviesa episodios de entrenamiento (Línea 3 a 9). En primer lugar, cada agente selecciona el siguiente conmutador del Espacio de Acción (selecciona $A_t$ para $E_t$), eligiendo un conmutador vecino del conmutador actual utilizando el método de exploración y explotación $\varepsilon$-greedy (Línea 5). A continuación, cada agente utiliza su respectiva información sobre el estado del enlace de la red y su estado $E_t$ para calcular la recompensa asociada con la acción $A_t$, y luego observa el nuevo estado $E_{t+1}$. La recompensa varía dependiendo del agente en cuestión (Línea 6), ya que esta función es diferente para cada uno. Posteriormente, considerando la tasa de aprendizaje, el factor de descuento, las consideraciones iniciales, la recompensa y el nuevo estado, cada agente ajusta el valor de su función $Q$ utilizando la Ecuación 3.10 (Línea 7). Luego, pasa al nuevo estado (Línea 8), finaliza el episodio de entrenamiento y comienza uno nuevo.

Posteriormente, una vez que cada uno de los agentes ha completado el proceso de entrenamiento, las tablas $Q$ resultantes de cada agente son promediadas utilizando la Ecuación 3.11 (Linea 10), generando una única tabla $Q$ consolidada. Esta tabla $Q$ permite determinar la ruta óptima entre los conmutadores origen-destino, en función de los pares estado-acción que alcanzaron los valores $Q$ más bajos (Línea 11).

Finalmente, una vez que el algoritmo IQRSMR encuentra la mejor ruta para todos los pares origen-destino y las almacena en el Repositorio de Rutas (Línea 13), el Plano de Control recupera estas rutas óptimas cada 15 segundos (valor encontrado en la investigación mediante ensayo y error, en donde un tiempo bajo satura la red de mensajes y un tiempo alto entrega rutas ineficientes) para su correspondiente comparación y posterior instalación en la tabla de enrutamiento de los conmutadores, garantizando así una configuración adecuada y eficiente de la red.

**Algorithm 3.1:** Enrutamiento IQRSMR

**Entrada:**
Tasa de aprendizaje: $\alpha$
Factor de descuento: $\gamma$
Parámetro de exploración y explotación: $\varepsilon$
Número de episodios de aprendizaje: $n$
Par de conmutadores ($origen$, $destino$): $Plist$
Estado del enlace de red

**Salida :** Mejor ruta para todos los pares de conmutadores en la red

**foreach** $(origen, destino) \in Plist$ **do**
  Inicializar $Q : Q(E, A) = 100, R : R(E, A) = 100$ Para todos los agentes
  **for** $episodio \leftarrow 1$ **to** $n$ **do**
    Iniciar en el estado $E_t = random$ // $E_t \in E$;
    Seleccionar $A_t$ para $E_t$ utilizando el método de exploración y explotación $\varepsilon$-greedy;
    $R_{t+1} \leftarrow R(E_t, A_t)$ `// El agente obtiene la recompensa del estado de enlace de la red y observa el nuevo estado` $E_{t+1}$;
    $Q_{t+1}(E_t, A_t) = Q_t(E_t, A_t) + \alpha \cdot [R_t + \min Q_t(E_{t+1}, A) - Q_t(E_t, A_t)]$
    `// Actualizar función Q Eq. 3.10`;
    $E_t \leftarrow E_{t+1}$ `// Pasa al siguiente estado`;
  **end**
  $Q_t(E_t, A_t) = \frac{1}{N} \sum_{k=1}^{N} Q_k(E_t, A_t)$
  `// Promedio de las funciones` $Q_k$ `Eq. 3.11`;
  Obtener desde la tabla $Q$ resultante, la ruta entre $origen$ y $destino$ con pares de estado-acción con los valores Q más bajos
**end**
Almacenar el conjunto de rutas para todos los pares de conmutadores en la red en el *Repositorio de rutas*

# Capítulo 4

# Evaluación

Este capítulo, primero, detalla la configuración del entorno de pruebas, abarcando desde el cálculo de las métricas de rendimiento para cada enlace de la red, hasta la generación de tráfico simulado que permitirá evaluar el desempeño de IQRSMR. Segundo, se examinan con detalle los criterios y métodos que han guiado la selección de los parámetros de aprendizaje para el algoritmo de enrutamiento. Finalmente, este capítulo expone los resultados obtenidos a través de la evaluación de IQRSMR. Estos resultados se acompañan de un exhaustivo análisis que brinda perspectivas claras sobre el rendimiento y la eficacia del sistema en su conjunto.

## 4.1 Entorno de Prueba

La Figura 4.1 presenta el entorno de prueba utilizado para la evaluación de IQRSMR. El entorno de prueba consta de una topología dividida en 2 dominios basada de [13]. Uno contiene $12$ conmutadores y $18$ enlaces, mientras que el otro tiene $11$ conmutadores y $14$ enlaces. Estos dominios están interconectados mediante un enlace de alta capacidad entre los conmutadores $S1$ y $S20$, esto para soportar la cantidad de tráfico que implica el enrutamiento entre dominios.

La construcción e implementación de la topología de la Figura 4.1, fue desarrollada en el emulador de redes Mininet 2.3.0 [116] usando un script de Python; esta herramienta fue seleccionada porque permite crear una red virtual realista rápidamente configurable. La capacidad de los enlaces de la topología de la Figura 4.1 se distribuyen: el 50 % de los enlaces con una capacidad de 10 Gbps, el 40 % con 5 Gbps, y el 10 % con 2 Gbps. Sin embargo, es importante tener en cuenta que Mininet está limitado por los recursos de la máquina host. Por ende, en las pruebas realizadas, se escalo la capacidad de los enlaces de 10 Gbps, 5 Gbps, y 2 Gbps a 100Mbps, 50Mbps y 20Mbps, respectivamente.

En la implementación realizada, cada conmutador cuenta con un host conectado encargado de reenviar y recibir tráfico, lo que permitió simular un ambiente realista y evaluar el rendimiento del prototipo de manera efectiva.

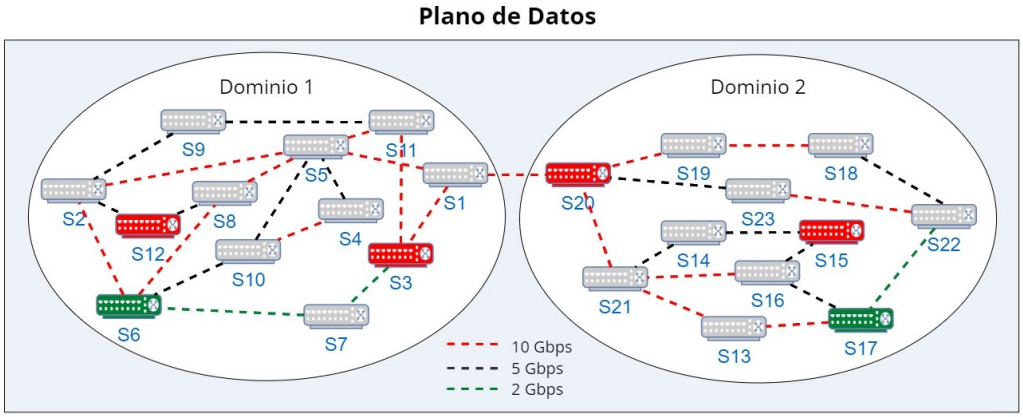



Figura 4.1: Entorno de prueba.

La Figura 4.2 presenta el prototipo de IQRSMR, en el cual se llevó a cabo la implementación del controlador Kandoo basado en Golang[1] para los Planos de Control y Gestión. Este controlador no solo potencia la escalabilidad y eficiencia de las redes, sino que también destaca por su baja complejidad a la hora de realizar su implementación. Los módulos de Estadísticas, Descubrimiento de Topología, Instalación de Flujo, Enrutamiento Entre Dominios, Comparación de Rutas y Procesamiento de Datos fueron desarrollados e implementados utilizando Beehive[2].

Para el desarrollo del plano de conocimiento, se implementó IQRSMR utilizando Python 3.6.9 en conjunto con la biblioteca Numpy 0.22 (*Numerical Python*) [118], la cual es una herramienta de código abierto utilizada para realizar una amplia variedad de operaciones matemáticas. Numpy enriquece Python al agregar poderosas estructuras de datos que aseguran cálculos eficientes y proporcionan una extensa biblioteca de funciones matemáticas de alto nivel.
La información de estado de enlace proporcionada por el Plano de Gestión y las rutas de enrutamiento calculadas por IQRSMR se almacenaron en archivos CSV (*Comma-Separated Values*) y JSON (*JavaScript Object Notation*), respectivamente. El prototipo y todos los scripts de prueba están disponibles en la Sección 5.2.

Los Planos de Control, Gestión y Conocimiento de cada controlador local se ejecutaron en máquinas virtuales Ubuntu 18.04.6 LTS con un procesador AMD Ryzen 5

[1] lenguaje de programación concurrente y de tipado estático, similar a C, desarrollado por google.
[2] plataforma de control distribuida que viene con transacciones integradas, replicación, tolerancia a fallas, instrumentación en tiempo de ejecución y ubicación optimizada [117].

3500U (1 núcleo lógico) y 3GB de RAM. El controlador raíz en una máquina virtual Ubuntu 18.04.6 LTS con un procesador AMD Ryzen 5 3500U (1 núcleo lógico) y 3 GB de RAM. El plano de datos en otra máquina virtual Ubuntu 18.04.6 LTS con un procesador AMD Ryzen 5 3500U (1 núcleo lógico) y 2 GB de RAM. Las máquinas virtuales utilizadas para este prototipo estaban alojadas en un Windows Desktop 10.04 con un AMD Ryzen 5 3500U y 12 GB de RAM.

La comunicación entre los controladores locales y el raíz, fue realizada mediante una API REST, la cual hace uso de mensajes de solicitud HTTP. Por otro lado, entre los controladores locales y el plano de datos, se empleó Openflow1.3, que es el protocolo de facto utilizado como SBI en SDN [119].

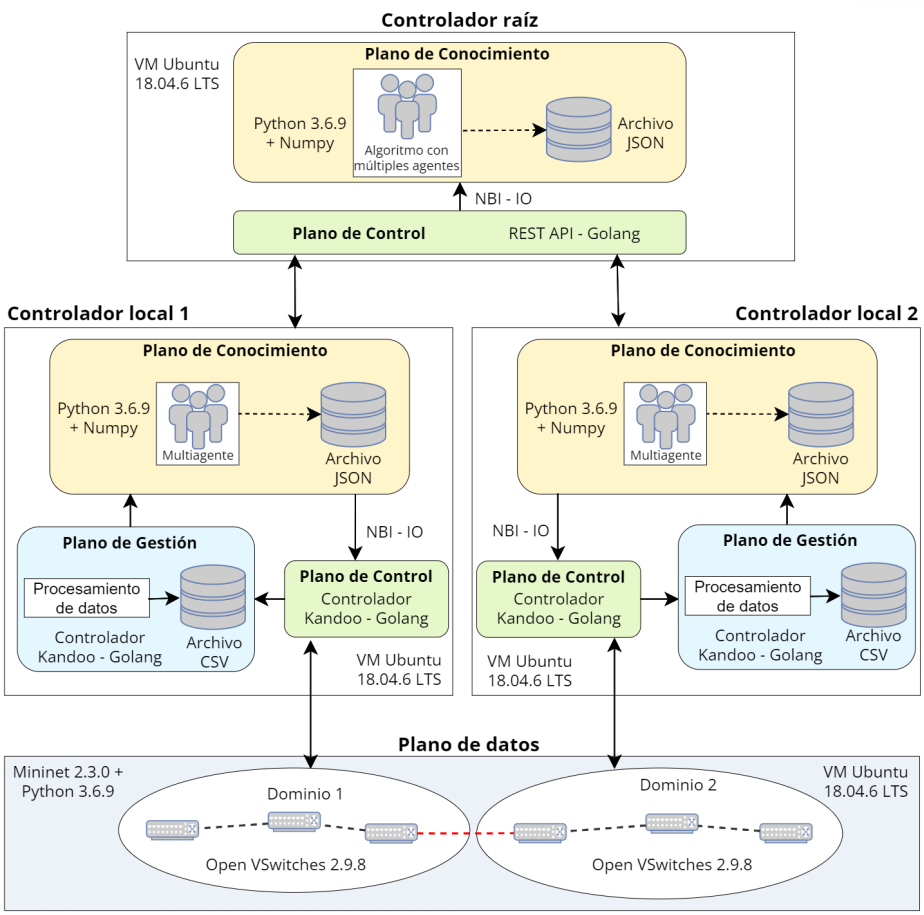



Figura 4.2: Prototipo de IQRSMR.

## 4.2 Métricas y Generación de Tráfico

Las métricas de QoS más utilizadas para la evaluación de las propuestas de enrutamiento, tal como se evidencia en la Tabla 2.1, comprenden el *packet loss, delay* y *throughput*. En adición a estas métricas, se llevó a cabo una comparativa sobre la extensión de las rutas ofrecidas por el algoritmo IQRSMR en contraste con las generadas por el algoritmo de Dijkstra y por un enfoque Q-Learning, utilizando como peso de borde (costo de cada enlace asociado a un valor numérico) el *throughput* y la ecuación de recompensa descrita en [14] respectivamente. Con el propósito de determinar la ruta más corta, fue desarrollada una aplicación que implementa estos algoritmos en cada controlador SDN.

Para la generación de tráfico en la emulación basada en Mininet, se empleó la herramienta iperf3 [120]. Se desarrollaron scripts destinados a la ejecución de clientes y servidores en cada uno de los hosts involucrados. El tráfico fue generado utilizando UDP (*User Datagram Protocol*), ya que iperf3 permite la especificación de una tasa de transmisión objetivo por conexión. El experimento abarcó la ejecución integral de los scripts, los cuales dieron lugar a la generación de tráfico determinado por una matriz de tráfico.

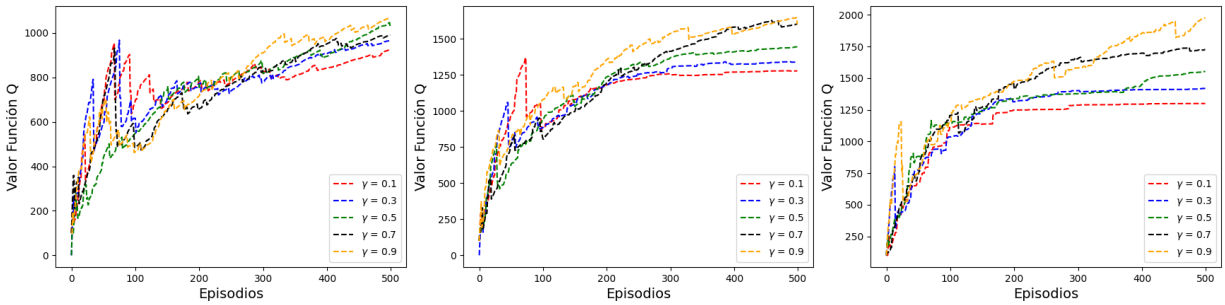
En este sentido, fueron utilizadas dieciséis matrices disponibles públicamente [121] con el fin de generar el tráfico entre los conmutadores emparejados. Los valores contenidos en estas matrices representaron el tráfico intercambiado entre diversos pares de conmutadores a lo largo de distintas horas del día durante un período de cuatro meses en el año 2006. Las matrices utilizadas fueron etiquetadas con las correspondientes franjas horarias del día, y se identificaron las horas pico como aquellas de mayor intensidad de tráfico (específicamente, entre las 7:00 h y las 13:00 h).

## 4.3 Configuración de Parámetros de Aprendizaje

Dentro del ámbito de RL, resulta esencial determinar los valores de la tasa de aprendizaje $\alpha$, el factor de descuento $\gamma$ y el parámetro de exploración-explotación $\varepsilon$. En esta investigación, se empleó el valor de la función $Q$ como indicador de la convergencia del algoritmo IQRSMR para un par de conmutadores origen-destino, lo que sirve como punto de referencia en la comparación de resultados. En este contexto, se llevó a cabo ejecuciones del algoritmo variando los valores de $\alpha$, $\gamma$ y $\varepsilon$ para alcanzar la convergencia, tanto en los controladores locales como en el raíz.

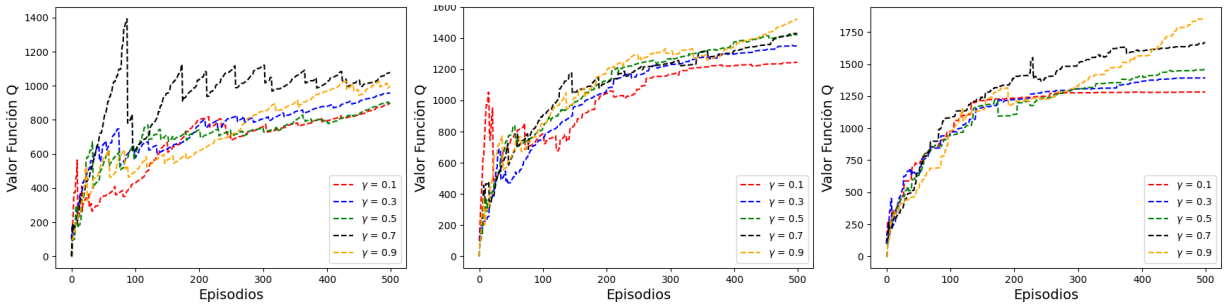
Para las pruebas, fueron seleccionados distintos pares de conmutadores como ob-

jetivos de convergencia en función de su distancia y ubicación en la topología. Por ejemplo, para el controlador local $1$, se estableció como par objetivo los conmutadores $S3$ y $S12$, mientras que para el controlador local $2$, fueron $S15$ y $S20$. Esta elección se basó en la considerable distancia que existe entre estos conmutadores dentro de sus respectivos dominios (como indican los interruptores resaltados en rojo en la Figura 4.1). En el caso del controlador raíz, los conmutadores $S6$ (dominio 1) y $S17$ (dominio 2) fueron seleccionados como par objetivo debido a su marcada separación en la topología (como representan los interruptores resaltados en verde en la Figura 4.1).




(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.3: Controlador local 1 $\varepsilon = 0{,}2$.

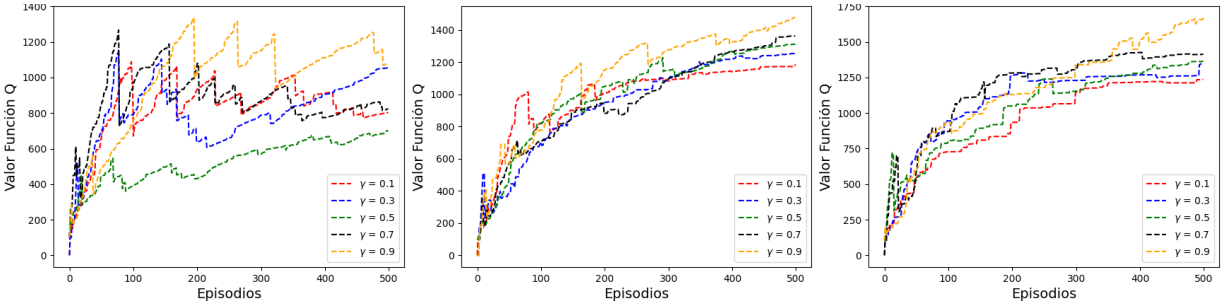



(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.4: Controlador local 1 $\varepsilon = 0{,}4$.

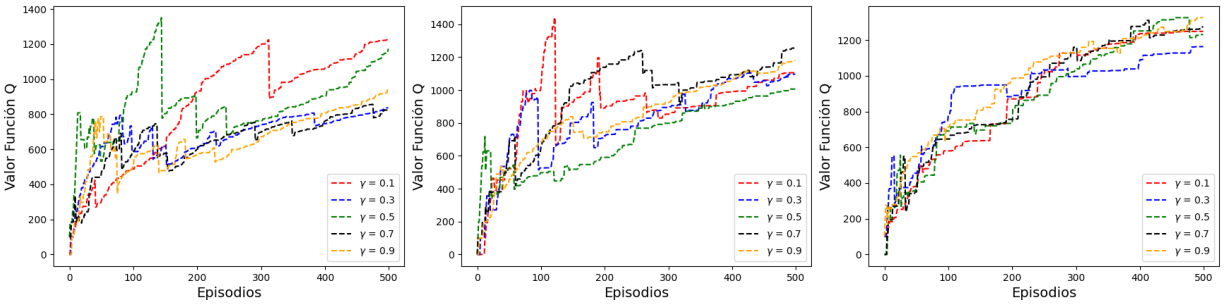
Las pruebas fueron realizadas variando el parámetro de explotación-exploración $\varepsilon$, con valores de $0{,}8$, $0{,}6$, $0{,}4$ y $0{,}2$. Para cada valor de $\varepsilon$, se llevaron a cabo evaluaciones utilizando tres valores diferentes de la tasa de aprendizaje $\alpha$ ($0{,}9$, $0{,}5$ y $0{,}1$). Además, para cada combinación de valores de $\varepsilon$ y $\alpha$, se efectuaron pruebas con cinco valores distintos del factor de descuento $\gamma$ ($0{,}9$, $0{,}7$, $0{,}5$, $0{,}3$ y $0{,}1$). Cada prueba fue realizada con un total de $500$ episodios para cada uno de los controladores locales y $900$ episodios para el controlador raíz.

Las Figuras 4.3 a 4.6 muestran la evolución de la función $Q$ en el controlador $1$ a medida que los episodios avanzan, considerando diversas combinaciones de valores de



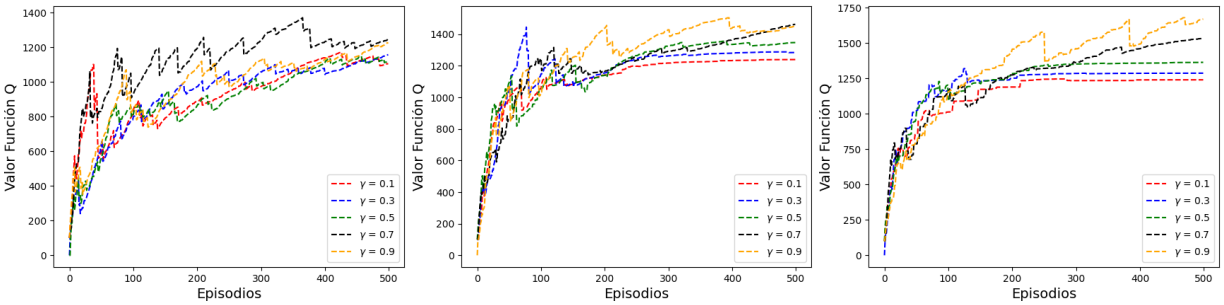

(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.5: Controlador local 1 $\varepsilon = 0{,}6$.




(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.6: Controlador local 1 $\varepsilon = 0{,}8$.

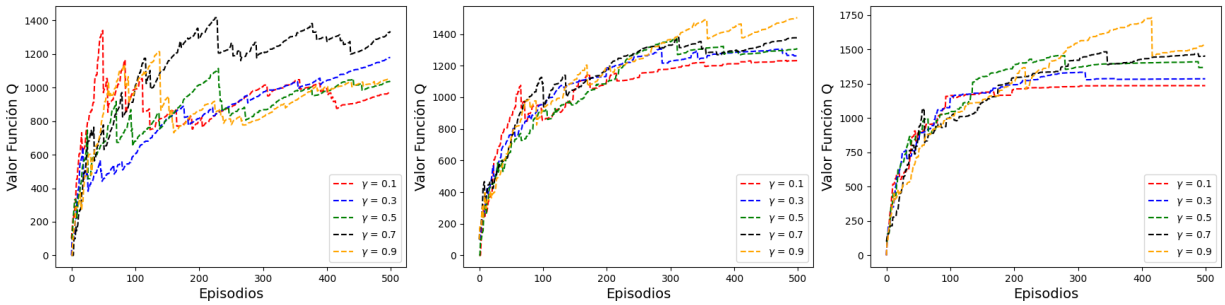



(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.7: Controlador local 2 $\varepsilon = 0{,}2$.

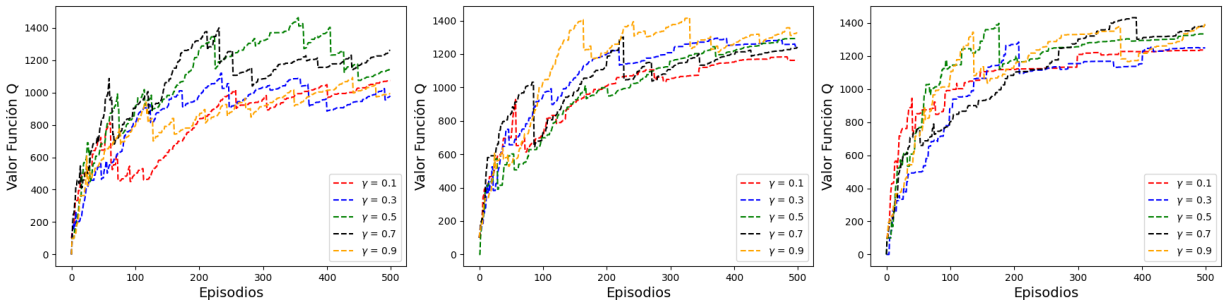
$\alpha$, $\gamma$ y $\varepsilon$. De manera similar, las Figuras 4.7 a 4.10 representan esta evolución en el controlador $2$, y las Figuras 4.11 a 4.14 en el controlador raíz.

Las Figuras 4.3, 4.4, 4.7, 4.8, 4.11 y 4.12 ilustran que cuando el valor de $\varepsilon$ se acerca a $0$, IQRSMR explora acciones aleatorias durante los primeros episodios. Esta dinámica promueve la rápida estabilización de la función $Q$ en un menor número de episodios en la mayoría de los escenarios. En contraste, cuando los valores de $\varepsilon$ se aproximan a $1$ (Figuras 4.5, 4.6, 4.9, 4.10, 4.13 y 4.14), IQRSMR tiende a explotar el conocimiento



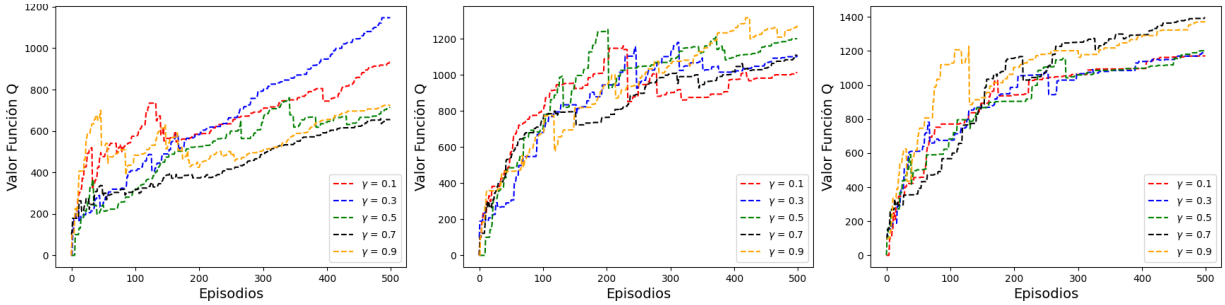

(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.8: Controlador local 2 $\varepsilon = 0{,}4$.




(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.9: Controlador local 2 $\varepsilon = 0,6$.

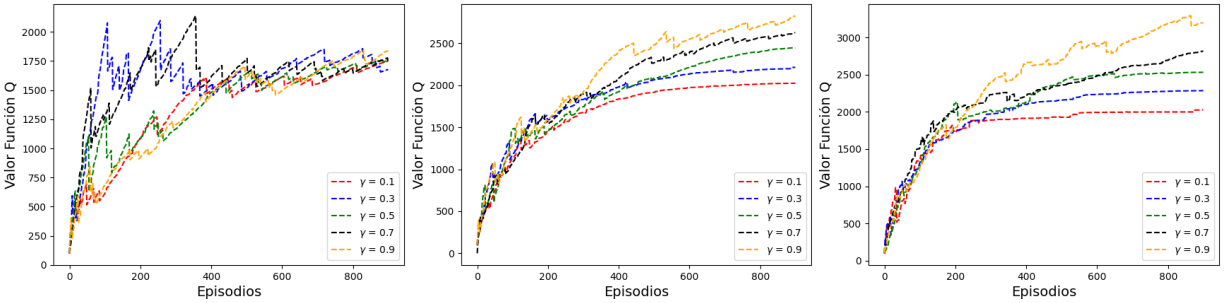



(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.10: Controlador local 2 $\varepsilon = 0,8$.

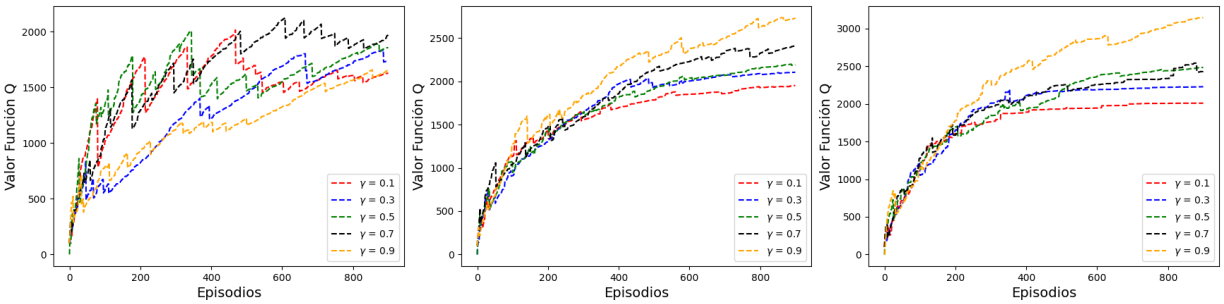
previo en lugar de explorar el espacio de acciones de manera aleatoria. Este comportamiento conduce a que los sistemas permanezcan atrapados en acciones subóptimas a lo largo de varios episodios, obstaculizando la convergencia hacia un valor $Q$ óptimo.

A medida que la tasa de aprendizaje $\alpha$ incrementa, se observa una convergencia más rápida. Para valores bajos de $\alpha$, como $0{,}1$ y $0{,}5$, IQRSMR no retiene el conocimiento previamente adquirido, lo que ralentiza el proceso de convergencia. En contraste, valores cercanos a $1$, como $\alpha = 0{,}9$, permite que se conserve la información de las



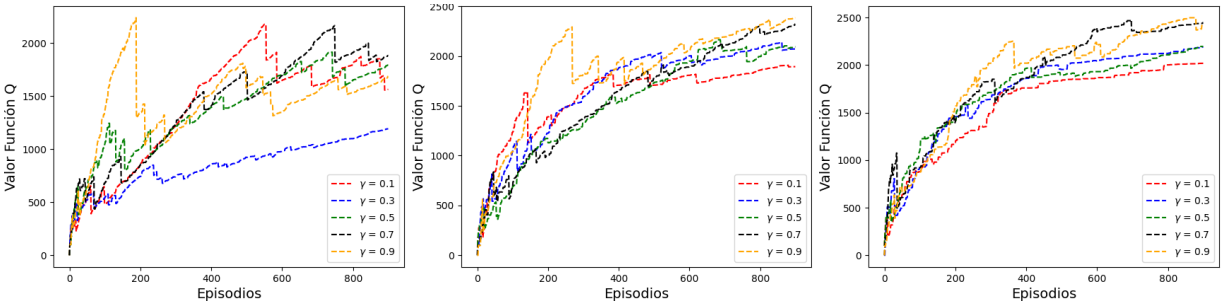

(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.11: Controlador raíz $\varepsilon = 0{,}2$.




(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.12: Controlador raíz $\varepsilon = 0{,}4$.

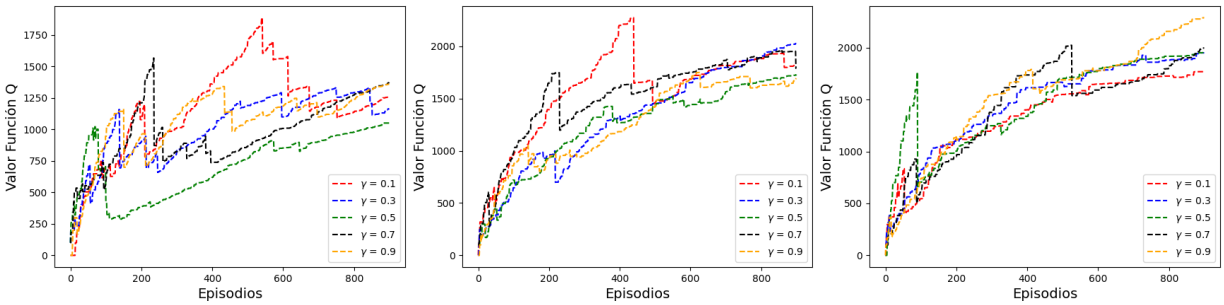



(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.13: Controlador raíz $\varepsilon = 0{,}6$.

recompensas recientes, lo cual resulta en una convergencia de la función $Q$ en menos episodios.

IQRSMR prioriza recompensas menores debido a que la ruta óptima se basa en el mínimo costo en cada enlace. Por consiguiente, valores de $\gamma$ cercanos a $1$, como $0{,}5$, $0{,}7$ y $0{,}9$, aumentan el costo de la función $Q$, haciendo que la convergencia sea más lenta. En contraste, valores próximos a $0$, como $0{,}1$ y $0{,}3$, minimizan el costo de cada enlace haciendo posible una convergencia más expedita.




(a) $\alpha = 0,1$ (b) $\alpha = 0,5$ (c) $\alpha = 0,9$

Figura 4.14: Controlador raíz $\varepsilon = 0{,}8$.

En base a estos resultados, se determinó que los valores más apropiados para los parámetros del enfoque IQRSMR, tanto para los controladores locales como para el controlador raíz, fueron $\alpha = 0{,}9$, $\gamma = 0{,}1$ y $\varepsilon = 0{,}4$. Mediante esta configuración (ver Figuras 4.4c, 4.8c y 4.12c), la función $Q$ logra una rápida convergencia hacia un estado estable.

## 4.4 Análisis de Cambio de Topología

Esta sección conlleva un proceso que se activa cada que se produce un cambio en la topología de la red. Este proceso engloba tres tareas esenciales: detección de cambios en la topología, cálculo de rutas y la comparación e instalación de dichas rutas. El módulo de descubrimiento de topología realiza la detección de estos cambios (causados por la inclusión o la falla de un conmutador o un enlace), mediante la recopilación periódica de información sobre la topología en cada intervalo de Tiempo de Detección de Cambios ($T_{top}$), este valor por defecto el controlador Kandoo lo establece en 1s, el cual podría ser un valor más bajo, pero aumentaría el tráfico entre el conmutador-controlador, provocando congestión y teniendo pérdida de información entre estos.

Para la determinación de rutas, IQRSMR emplea un período denominado Tiempo de Aprendizaje ($T_{learn}$), durante el cual ejecuta el Algoritmo 3.1. Cabe destacar que este período varía dependiendo del número de conmutadores y los parámetros de aprendizaje vistos en la Sección 4.3. Por último, tenemos el Tiempo de Instalación de Flujos ($T_{inst}$), tiempo que tarda el controlador en actualizar las tablas de reenvío en todos los conmutadores del plano de datos con las nuevas rutas calculadas, en caso que difieran de las rutas establecidas en la red. El Tiempo de Instalación de Flujos comprende a los módulos de comparación de rutas y de instalación de flujo.

El Tiempo de Instalación de Flujo ($T_{inst}$) se midió para el controlador raíz desde que

sale una sola regla del controlador raíz al local (el tiempo en el peor de los casos que toma una regla en llegar del controlador raíz al local es de $T_{cont} = 15s$, valor predefinido), entra a los módulos de comparación de rutas e instalación de flujo descritos en la Figura 3.2, hasta que se actualizan las tablas de flujo por medio del mensaje OpenFlow (*flow-mod*) enviado al conmutador. Para el controlador local es el mismo procedimiento anteriormente descrito, ya que usa los mismos módulos, sin embargo no se toma en cuenta $T_{cont}$. Para calcular el tiempo promedio que tarda IQRSMR en manejar un cambio de topología en la red se emplea la siguiente ecuación:

$$T_{total} = T_{top} + T_{learn} + T_{inst} + T_{cont} \tag{4.1}$$

Con la información de la Tabla 4.1, que contiene los tiempos de aprendizaje de IQRSMR en los controladores, y los valores de $T_{top}$ y $T_{inst}$, se calcula por medio de la Ecuación 4.1 los tiempos de manejo de cambio de topología:

| | Controlador Raíz | Controlador Local 1 | Controlador Local 2 |
|---|---|---|---|
| Conmutadores | 23 | 12 | 11 |
| Número de episodios | 700 | 500 | 500 |
| Parametro de exploración-explotación | 0.4 | 0.4 | 0.4 |
| Tasa de aprendizaje | 0.9 | 0.9 | 0.9 |
| Factor de descuento | 0.1 | 0.1 | 0.1 |
| Tiempo de aprendizaje | 47 segundos | 41 segundos | 37 segundos |

Tabla 4.1: Tiempo de aprendizaje de IQRSMR.

- Controlador Raíz:

$$T_{total} = 1s + 47s + 1{,}78s + 15s$$

$$T_{total} = 64{,}78s$$

- Controlador Local 1:

$$T_{total} = 1s + 41s + 1{,}76s$$

$$T_{total} = 43{,}76s$$

- Controlador Local 2:

$$T_{total} = 1s + 37s + 1{,}76s$$

$$T_{total} = 39{,}76s$$

De acuerdo a los tiempos de cambio de topología entregados por IQRSMR, podemos decir que en comparación con protocolos de enrutamiento tradicionales como RIP y OSPF, que toman alrededor de 30s y 10s (por defecto), respectivamente [13], los tiempos en IQRSMR son más elevados. Sin embargo es importante señalar que en los

protocolos tradicionales no se tienen en cuenta métricas de QoS, por ende, el rendimiento de la red sera menor porque se tendrá un mayor numero de rutas saturadas, por esta razón es factible usar IQRSMR, ya que elige las rutas teniendo en cuenta *throughput*, *delay* y *packet loss*. También, se puede expresar con certeza que el tamaño de la topología afecta este tiempo de aprendizaje, ya que el controlador raíz al tener que entregar 264 rutas toma más tiempo que los controladores locales 1 y 2, que solo procesan 132 y 110 rutas, respectivamente. Por tanto, al tener más conmutadores el tiempo de aprendizaje de IQRSMR aumenta.

## 4.5 Análisis de Resultados

En esta sección se presentan los resultados experimentales del mecanismo de enrutamiento inteligente con respecto a las métricas de rendimiento descritas en el la Sección 4.2.

Para evaluar el desempeño de IQRSMR en términos de cumplimiento de QoS, se realizó una comparación con los algoritmos Dijkstra enfocado en el *throughput* ($Dijkstra_{Re}$) y Q-Learning. Se eligió $Dijkstra_{Re}$ porque en [14] se demostró que la métrica mas significativa a tener en cuenta es la del *throughput*. Se obtuvieron resultados de la vista global (controlador raíz) y local (controladores locales 1 y 2) de la topología de red descrita en la Figura 4.1 mediante la información de red que suministra iperf3 cuando se ejecutan las matrices de tráfico descritas en la Sección 4.2.

Para calcular el porcentaje de error de los valores obtenidos en la implementación del prototipo se utilizó la siguiente formula:

$$Porcentaje_{err} = \left| \frac{V_{IQRSMR} - V_{comp}}{V_{comp}} \right| * 100 \tag{4.2}$$

- $V_{IQRSMR}$: Valor promedio de la métrica entregada por IQRSMR.
- $V_{comp}$: Valor promedio de la métrica entregada por $Dijkstra_{Re}$ o Q-Learning.

### 4.5.1 Controlador Raíz

La figura 4.15 muestra el *stretch* (relación entre el número de saltos entregados por los algoritmos analizados y la ruta más corta, un valor bajo indica que el controlador esta tomando decisiones de enrutamiento eficientes) promedio calculado para todas las rutas encontradas por los algoritmos de $Dijkstra_{Re}$, Q-Learning e IQRSMR en el controlador raíz. Para calcular la longitud del *stretch*, se compararon las rutas generadas

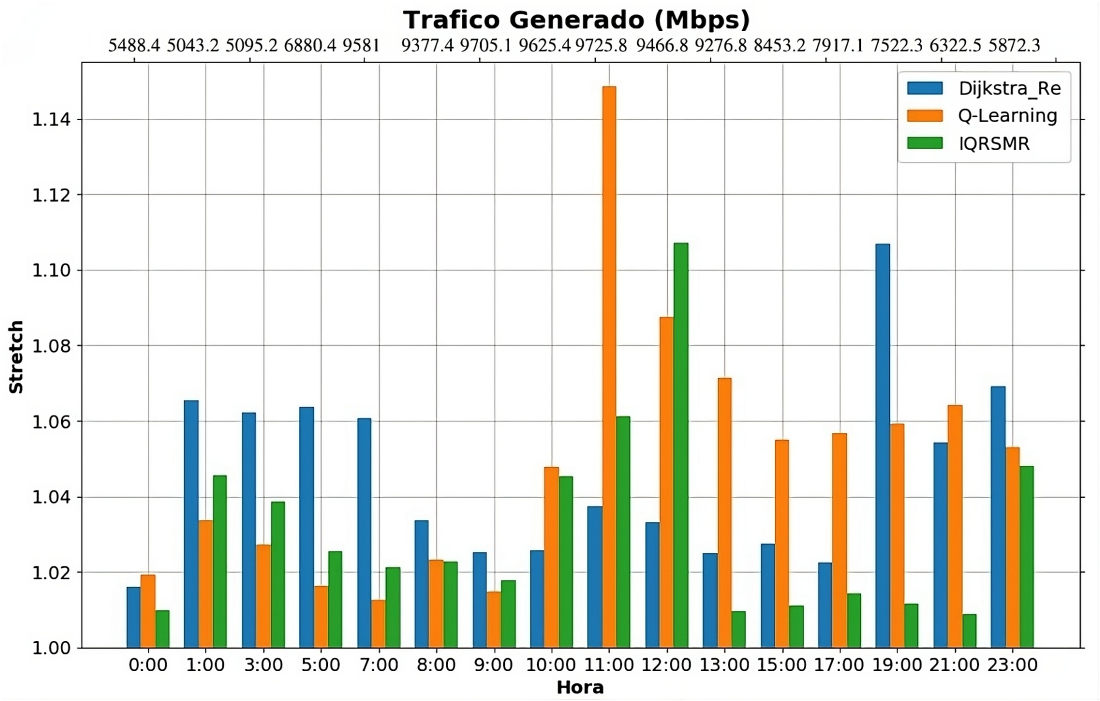



Figura 4.15: Stretch promedio a lo largo del día - Controlador Raíz.

por los diferentes algoritmos con la ruta más corta obtenida mediante el algoritmo de Dijkstra (utilizando pesos de enlace iguales). Los resultados mostraron que las rutas promedio elegidas por IQRSMR tiene valores de *stretch* ligeramente más pequeños que los producidos por $Dijkstra_{Re}$ y Q-Learning en 1,4 % y 1,7 % (máximo, 8,6 % y 7,6 %) respectivamente. También se mostró que IQRSMR en horas de mayor tráfico (10:00, 11:00 y 12:00) indica rutas con valores de *stretch* más altos (1,9 %, 2,3 % y 7,1 %) que los producidos por $Dijkstra_{Re}$. Lo anterior indica que $Dijkstra_{Re}$ es mejor que IQRSMR en términos de *stretch* en horas de alto trafico. Sin embargo, lo anterior es resultado de elegir rutas menos utilizadas, por lo general más largas, debido al alto trafico que se presenta en los enlaces de las rutas cortas, pero mejora en términos de *throughput*, *delay* y *packet loss*, esto demuestra la adaptabilidad que tiene IQRSMR frente a escenarios con una alta congestión de la red.

La figura 4.16 representa el *throughput* promedio de todos los enlaces a lo largo del día en el controlador raíz. Los resultados muestran que el *throughput* del enlace producido por IQRSMR fue en promedio 4 % y 1,9 % (máximo, 7,1 % y 5,3 %) superiores a los producidos por los algoritmos $Dijkstra_{Re}$ y Q-Learning, respectivamente. Lo anterior indica que IQRSMR elige rutas levemente menos congestionadas con respecto a los algoritmos mencionados, debido a que una de las métricas utilizadas por el algoritmo es el *throughput*.

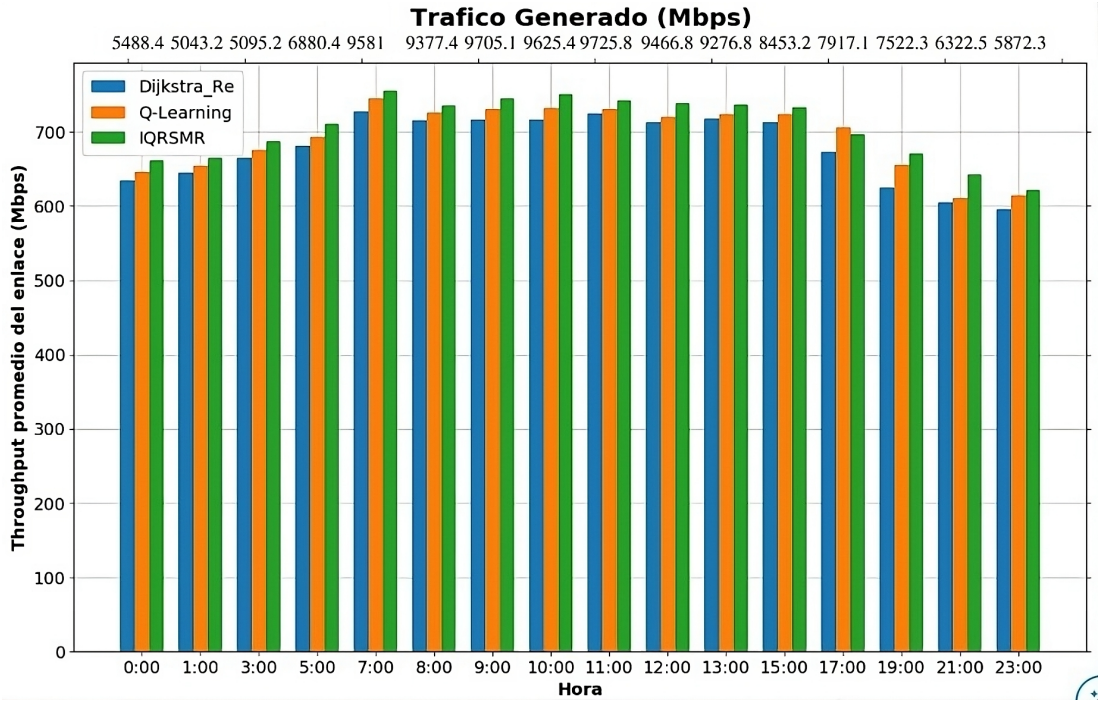



Figura 4.16: Throughput promedio de todos los enlaces a largo del día - Controlador Raíz.

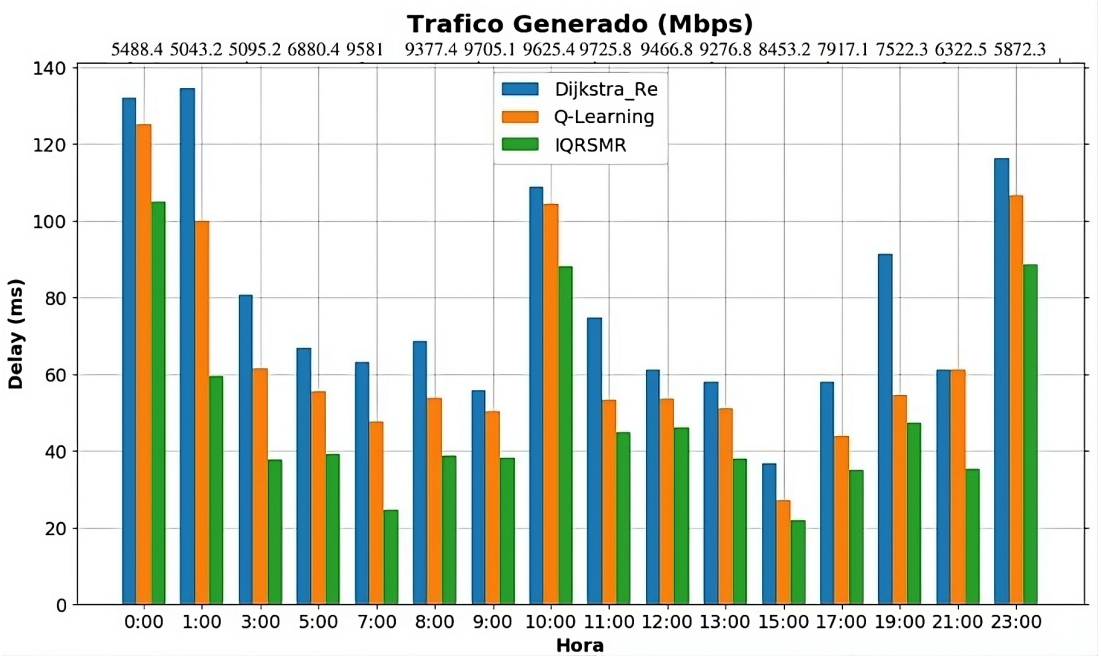



Figura 4.17: Delay promedio de todos los enlaces a largo del día - Controlador Raíz.

La figura 4.17 expone el *delay* promedio de todos los enlaces a lo largo del día para los resultados de los algoritmos $Dijkstra_{Re}$, Q-Learning y IQRSMR en el controlador

raíz. Los valores medios de *delay* de enlace producidos por IQRSMR son en promedio 37,8 % y 24,9 % (máximo, 60,7 % y 47,9 %) más bajos que los obtenidos por $Dijkstra_{Re}$ y Q-Learning, respectivamente. Los valores medios de *delay* conseguidos por IQRSMR son más bajos que los dados por los otros algoritmos, ya que IQRSMR generalmente elige la rutas más cortas y menos congestionadas que los algoritmos con los que se compara.

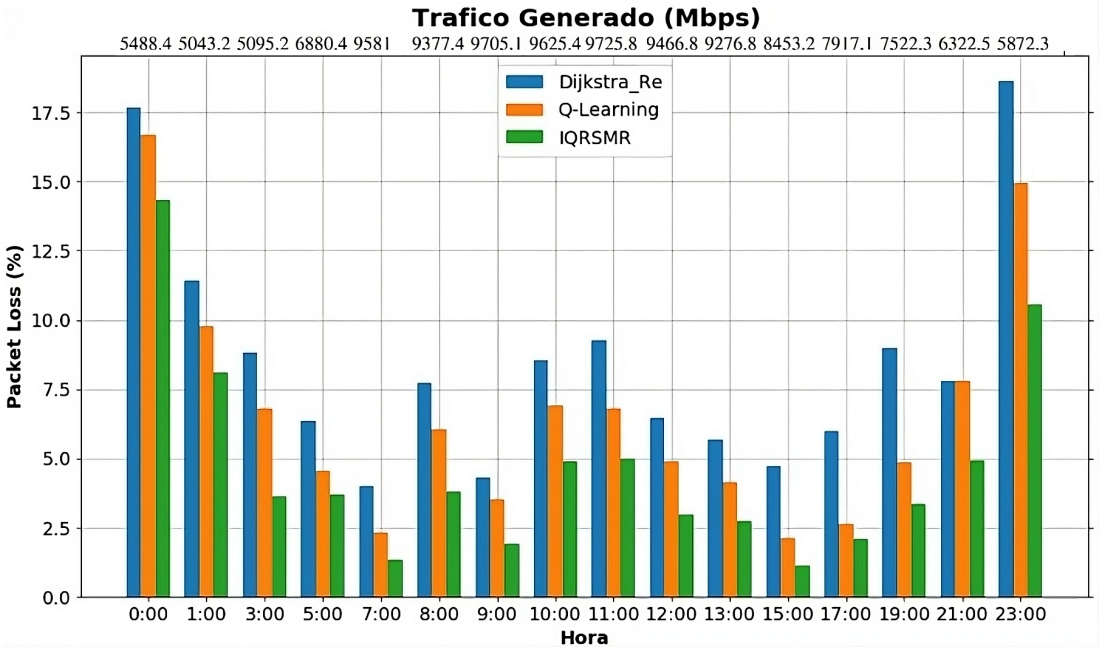



Figura 4.18: Packet Loss promedio de todos los enlaces a largo del día - Controlador Raíz.

La figura 4.18 representa el *packet loss* promedio de todos los enlaces a lo largo del día en el controlador raíz. Los valores medios de *packet loss* producidos por IQRSMR son en promedio 45,4 % y 28,9 % (máximo, 75,7 % y 46,6 %) más bajos que la relación de *packet loss* dada por $Dijkstra_{Re}$ y Q-Learning, respectivamente. Similar al *delay*, esto se debe a que IQRSMR elige rutas menos congestionadas, incluso en situaciones en donde la red tiene un gran flujo de tráfico.

### 4.5.2 Controlador Local 1

El análisis de estos resultados y los de la Sección 4.5.3 son similares a los presentados en la Sección 4.5.1, ya que las condiciones de evaluación son las mismas, solo que en estas se toman en cuenta únicamente los resultados de métricas obtenidas para el dominio que es supervisado por cada controlador. En este caso, se obtienen figuras de

las métricas obtenidas por el dominio 1 de la Figura 4.1.

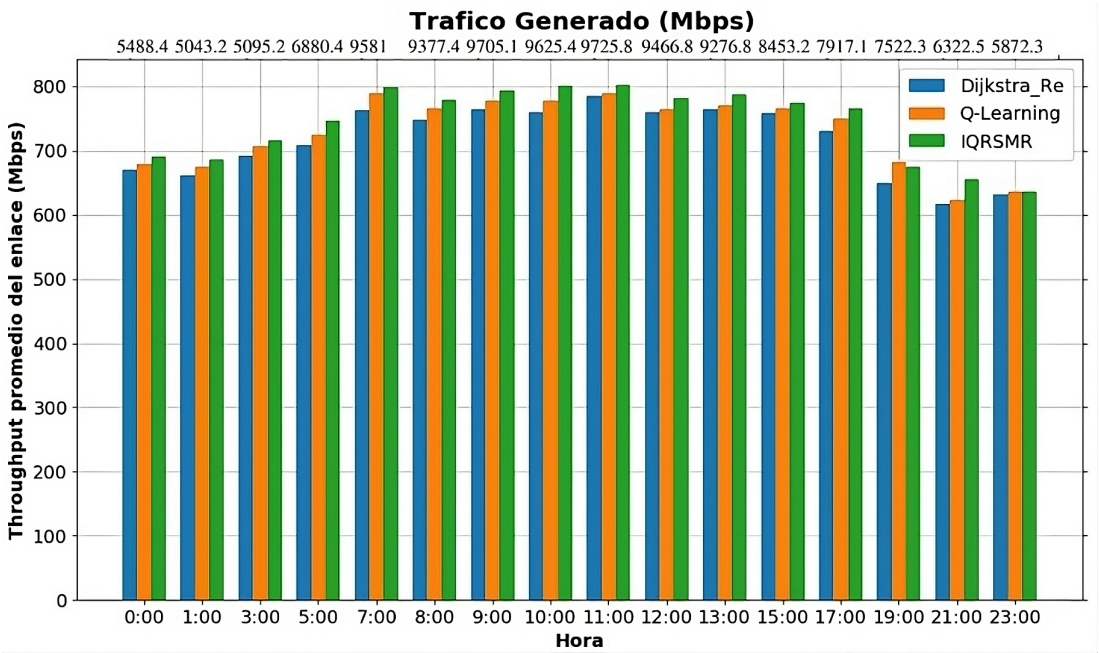



Figura 4.19: Throughput promedio de todos los enlaces a largo del día - Controlador Local 1.

La figura 4.19 muestra el *throughput* promedio de todos los enlace a lo largo del día en el controlador local 1. Los resultados muestran que el *throughput* del enlace producido por IQRSMR fue en promedio 3,7 % y 1,8 % (máximo, 6,1 % y 5,1 %) superiores a los producidos por los algoritmos de $Dijkstra_{Re}$ y Q-Learning, respectivamente. De acuerdo a los porcentajes obtenidos anteriormente, se puede decir que IQRSMR en el controlador local 1, en comparación con IQRSMR en el controlador raíz, presenta un *throughput* ligeramente menor en términos de porcentajes, esto porque el dominio analizado tiene un menor número de enlaces, y por tanto emplea rutas menos utilizadas que $Dijkstra_{Re}$ y Q-Learning.

La figura 4.20 expone el *delay* promedio de todos los enlaces a lo largo del día para los resultados de los algoritmos $Dijkstra_{Re}$, Q-Learning y IQRSMR en el controlador local 1. Los valores medios de *delay* de enlace producidos por IQRSMR son en promedio 38,6 % y 24,4 % (máximo, 80,6 % y 49,7 %) más bajos que los obtenidos por $Dijkstra_{Re}$ y Q-Learning, respectivamente. El análisis indica que el *delay* promedio del enlace obtenido usando IQRSMR en el controlador local 1, en comparación con IQRSMR en el controlador raíz es mas bajo en términos de porcentajes, esto porque elige rutas menos utilizadas, y por ende el nivel de tráfico es menor.

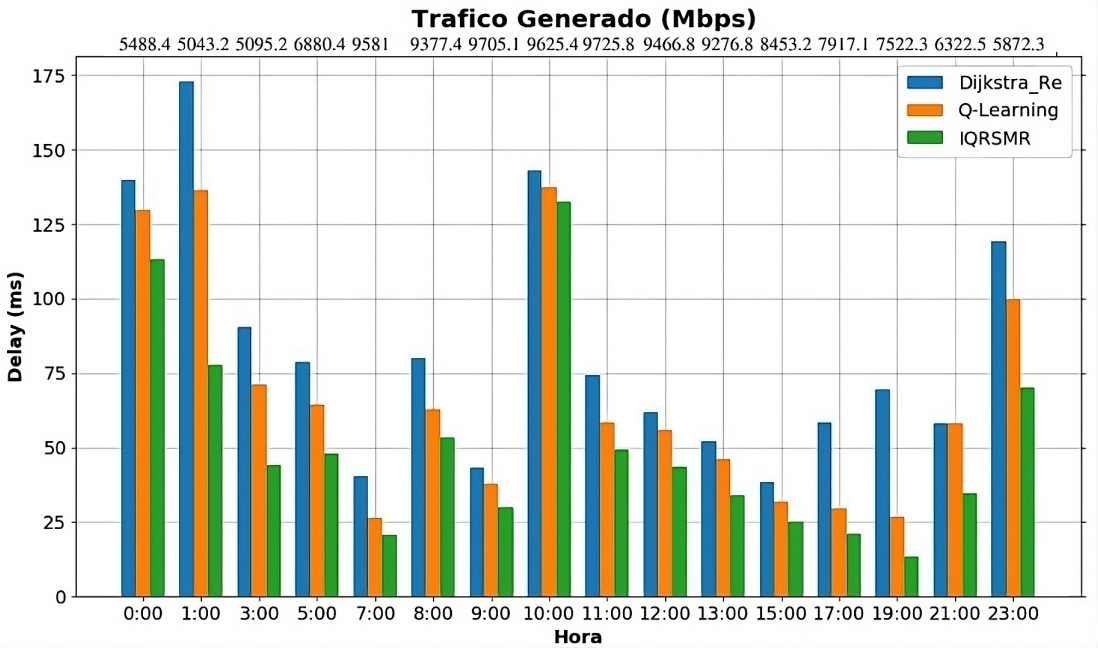



Figura 4.20: Delay promedio de todos los enlaces a largo del día - Controlador Local 1.

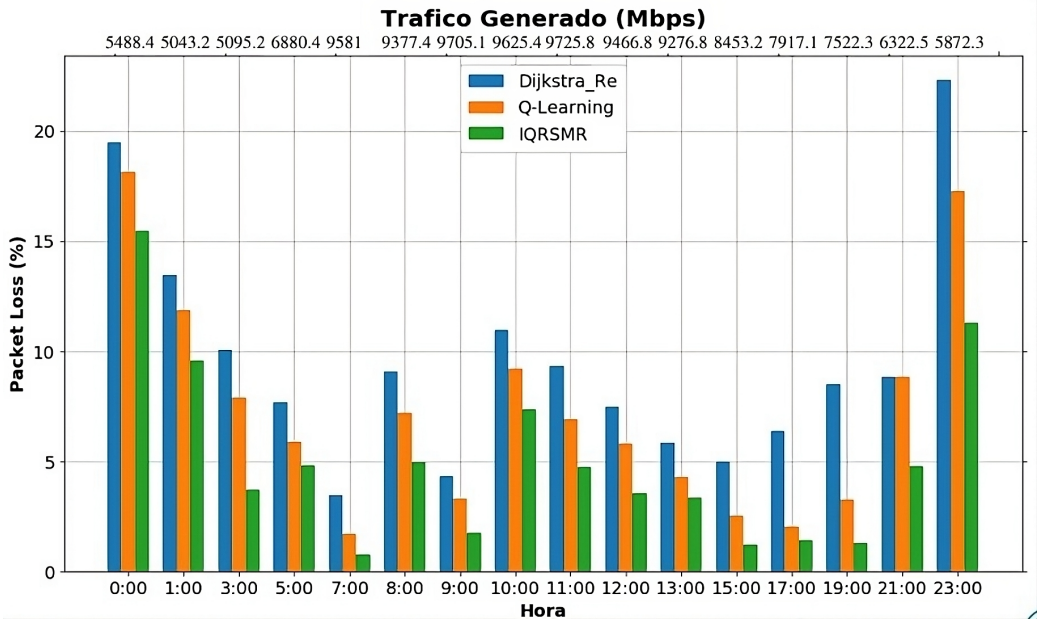



Figura 4.21: Packet Loss promedio de todos los enlaces a largo del día - Controlador Local 1.

La figura 4.21 representa el *packet loss* promedio de todos los enlaces a lo largo del día en el controlador local 1. Los valores medios de *packet loss* producidos por IQRSMR son en promedio 47,3 % y 31 % (máximo, 84,5 % y 59,6 %) más bajos que la relación de *packet loss* dada por $Dijkstra_{Re}$ y Q-Learning, respectivamente. Los resultados obte-

nidos muestran que el *packet loss* promedio del enlace obtenido utilizando IQRSMR en el controlador local 1, en comparación con IQRSMR en el controlador raíz es más bajo en términos de porcentajes, esto sucede porque al elegir rutas menos congestionadas se van a perder menos paquetes.

### 4.5.3 Controlador Local 2

En este caso, se obtienen figuras de las métricas obtenidas solo por el dominio 2 expuesto en la Figura 4.1.

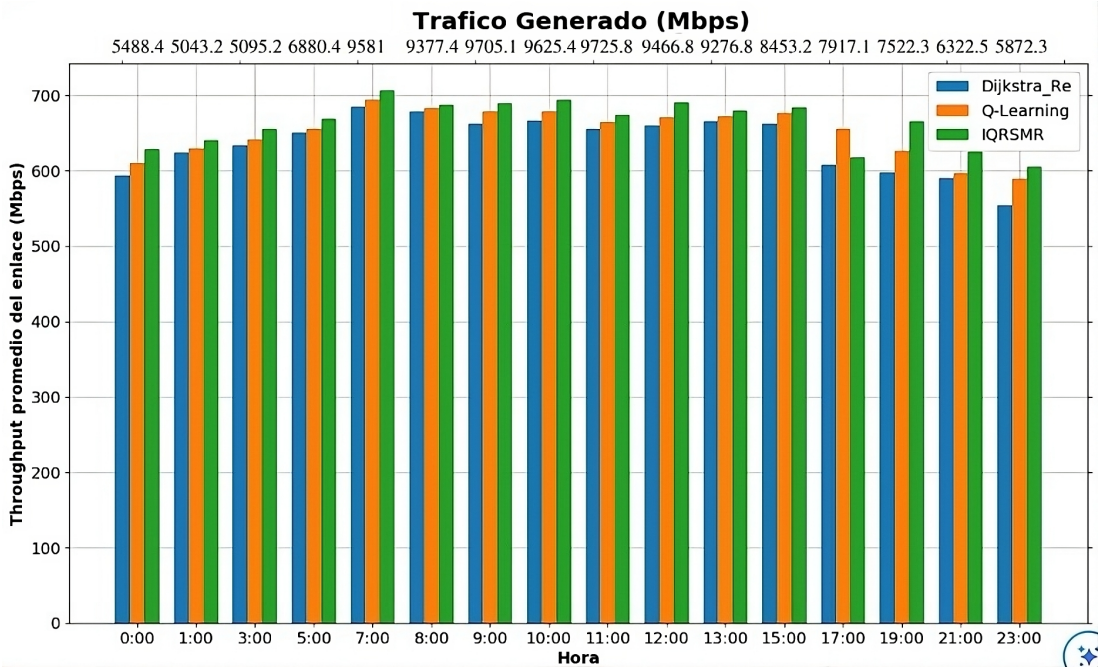



Figura 4.22: Throughput promedio de todos los enlaces a largo del día - Controlador Local 2.

La figura 4.22 muestra el *throughput* promedio de todos los enlaces a lo largo del día en el controlador local 2. Los resultados muestran que el *throughput* del enlace producido por IQRSMR fue en promedio 4,2 % y 1,9 % (máximo, 11,4 % y 6,3 %) superiores a los producidos por los algoritmos de $Dijkstra_{Re}$ y Q-Learning. De acuerdo a los porcentajes obtenidos anteriormente, se puede decir que IQRSMR en el controlador local 2, en comparación con IQRSMR en el controlador raíz, presenta un *throughput* mayor en términos de porcentajes, esto porque el dominio analizado tiene un menor número de enlaces, y por tanto emplea rutas mas cortas o iguales que $Dijkstra_{Re}$ y Q-Learning.
La figura 4.23 expone el *delay* promedio de todos los enlaces a lo largo del día para los resultados de los algoritmos $Dijkstra_{Re}$, Q-Learning y IQRSMR en el controlador

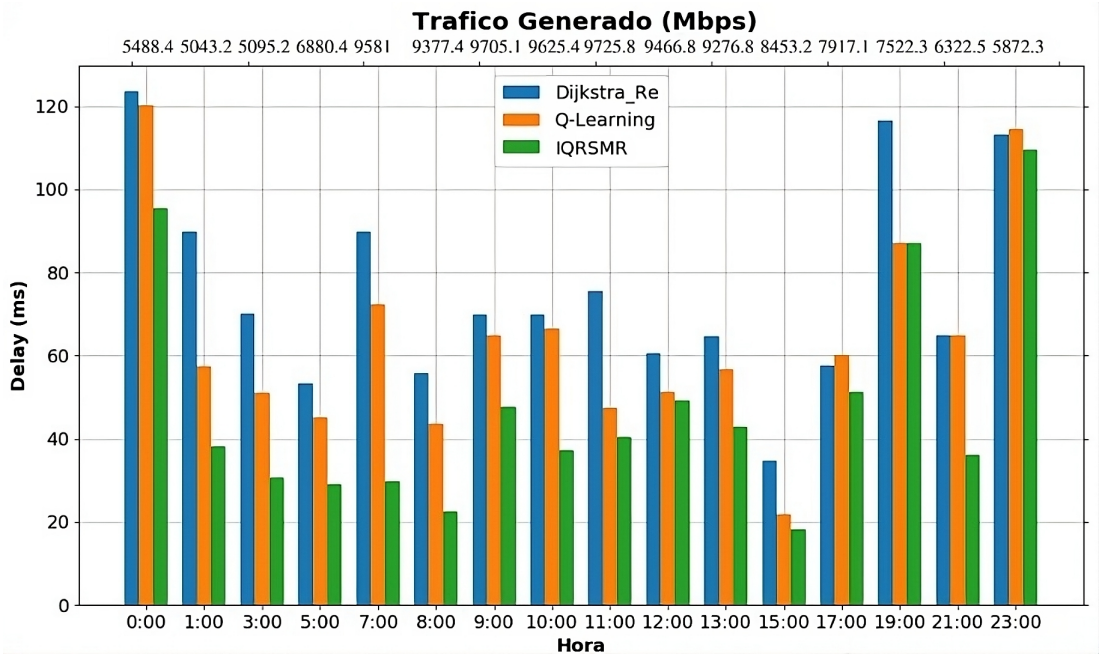



Figura 4.23: Delay promedio de todos los enlaces a largo del día - Controlador Local 2.

local 2. Los valores medios de *delay* de enlace producidos por IQRSMR son en promedio 36,8 % y 25,3 % (máximo, 66,8 % y 58,7 %) más bajos que los obtenidos por $Dijkstra_{Re}$ y Q-Learning, respectivamente. A partir de los resultados obtenidos podemos decir que el *delay* promedio del enlace usando IQRSMR en el controlador local 2, en comparación con IQRSMR en el controlador raíz es mas alto en términos de porcentajes, esto porque elige rutas más cortas al tener un menor número de enlaces, y por ende el nivel de tráfico es mayor, ocasionando un retraso adicional en los paquetes.

La figura 4.24 representa el *packet loss* promedio de todos los enlaces a lo largo del día en el controlador local 2. Los valores medios de *packet loss* producidos por IQRSMR son en promedio 42,6 % y 26,1 % (máximo, 76,1 % y 52,6 %) más bajos que la relación de *packet loss* dada por $Dijkstra_{Re}$ y Q-Learning, respectivamente. Los resultados obtenidos muestran que el *packet loss* promedio del enlace obtenido utilizando IQRSMR en el controlador local 2, en comparación con IQRSMR en el controlador raíz es más alto en términos de porcentajes, esto sucede porque se tienen menos enlaces en el domino 2 y por ende, elige rutas más congestionadas, teniendo una pérdida de paquetes superior que la vista en el controlador 2.

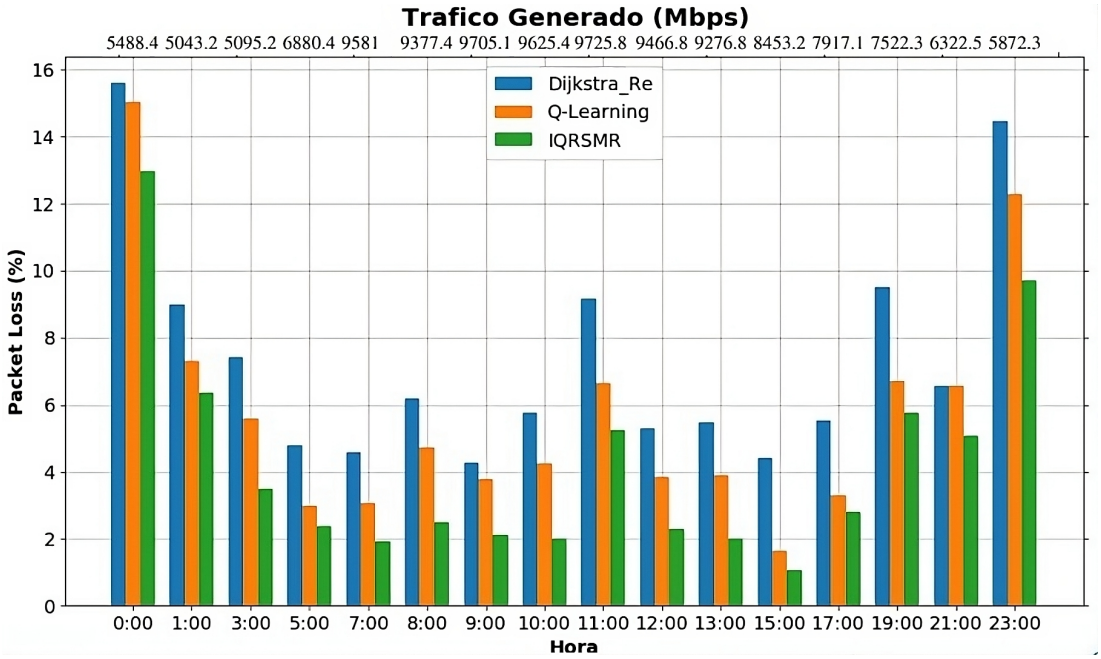



Figura 4.24: Packet Loss promedio de todos los enlaces a largo del día - Controlador Local 2.

# Capítulo 5

# Conclusiones y Trabajo Futuro

Este capítulo presenta la respuesta a la pregunta de investigación: **¿Cómo realizar enrutamiento inteligente y consciente de la calidad del servicio en redes jerárquicas definidas por software?** y sus respectivos trabajos futuros.

## 5.1 Conclusiones

Un mecanismo inteligente con múltiples agentes basado en aprendizaje automático y un enrutamiento consciente de la calidad de servicio en una red definida por software jerárquica fue diseñado para responder a la pregunta anteriormente mencionada. Se evaluó con una topología de 23 conmutadores y 33 enlaces con respecto al *packet loss, delay* y *throughput*. Los resultados mostraron lo siguiente:

- El mecanismo con el algoritmo basado en aprendizaje por refuerzo con múltiples agentes propuesto en el trabajo de grado superó los resultados obtenidos en los algoritmos Dijkstra (basado en *throughput*) y Q-Learning con respecto a las métricas de *packet loss, delay* y *throughput* en escenarios de bajo y alto tráfico en la red.

- si la dejamos???Aunque la cantidad de episodios de convergencia puede llegar a variar según el tamaño de la red. La configuración de los parámetros de aprendizaje demuestra que el enfoque IQRSMR es indiferente en cuanto a la topología, ya que los valores obtenidos de $\gamma$, $\varepsilon$ y $\alpha$ se mantienen consistentes en todos los controladores. Esto respalda la existencia de valores óptimos en el algoritmo, que determinan la toma de decisiones correctas y garantizan una convergencia exitosa.

- El algoritmo de enrutamiento basado en IQRSMR entrega rutas para cada par de conmutadores origen-destino de la red, priorizando las rutas con un mayor

*throughput* y menor *packet loss* y *delay*. La mejor opción de enrutamiento se basa en la recompensa de estas tres métricas del estado de la ruta.

Por tanto, se logró una solución eficiente de enrutamiento inteligente y consciente de la calidad de servicio en una red SDN jerárquica, mejorardo las métricas de *packet loss, delay* y *throughput* con respecto a soluciones clásicas como Dijkstra y Q-Learning, en condiciones de constante tráfico en la red.

## 5.2 Trabajo Futuro

- Implementar un mecanismo de enrutamiento inteligente y consciente de la calidad de servicio en redes definidas por software usando algoritmos de aprendizaje por refuerzo profundo.
- Evaluar el impacto de otras métricas diferentes del *packet loss, delay* y *throughput* en el algoritmo de aprendizaje por refuerzo con múltiples agentes. Por ejemplo, el *jitter* para conocer la congestión de la red y el *bandwidth* para conocer la capacidad de la red.

# Bibliografía


[1] G. Neto, "From single-agent to multi-agent reinforcement learning: Foundational concepts and methods," *Learning theory course*, vol. 2, 2005.

[2] S. Ahmad and A. Mir, "Scalability, consistency, reliability and security in sdn controllers: A survey of diverse sdn controllers," *Journal of Network and Systems Management*, vol. 29, 01 2021.

[3] J. F. Kurose and K. W. Ross, "Computer networking: A top-down approach edition," *Addision Wesley*, 2007.

[4] R. Amin, E. Rojas, A. Aqdus, S. Ramzan, D. Casillas-Perez, and J. M. Arco, "A survey on machine learning techniques for routing optimization in sdn," *IEEE Access*, vol. 9, pp. 104582–104611, 2021.

[5] R. A. Guerin, A. Orda, and D. Williams, "Qos routing mechanisms and ospf extensions," in *GLOBECOM 97. IEEE Global Telecommunications Conference. Conference Record*, vol. 3, pp. 1903–1908, IEEE, 1997.

[6] A. A. Ajani, B. J. Ojuolape, A. A. Ahmed, T. Aduragba, and M. Balogun, "Comparative performance evaluation of open shortest path first, ospf and routing information protocol, rip in network link failure and recovery cases," in *2017 IEEE 3rd International Conference on Electro-Technology for National Development (NIGERCON)*, pp. 280–288, IEEE, 2017.

[7] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "Bgp routing stability of popular destinations," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 197–202, 2002.

[8] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "A tensor based deep learning technique for intelligent packet routing," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2017.

[9] F. Bannour, S. Souihi, and A. Mellouk, "Distributed sdn control: Survey, taxonomy, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 333–354, 2018.

[10] R. Liu, S. Li, and H. Wang, "Hierarchical multi-constraint routing algorithm based on software defined networking," in *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pp. 1–5, 2019.

[11] Y. Zhang and G. Zhao, "Lsea: Software-defined networking-based qos-aware routing mechanism for live-soccer event applications in smart cities," *Wireless Communications and Mobile Computing*, vol. 2020, pp. 1–8, 11 2020.

[12] W.-E. Liang and C.-A. Shen, "A high performance media server and qos routing for svc streaming based on software-defined networking," in *2017 International Conference on Computing, Networking and Communications (ICNC)*, pp. 556–560, 2017.

[13] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "Drsir: A deep reinforcement learning approach for routing in software-defined networking," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021.

[14] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, 2021.

[15] E. H. Bouzidi, A. Outtagarts, R. Langar, and R. Boutaba, "Deep q-network and traffic prediction based routing optimization in software defined networks," *Journal of Network and Computer Applications*, vol. 192, p. 103181, 2021.

[16] T. A. Q. Pham, Y. Hadjadj-Aoul, and A. Outtagarts, "Deep reinforcement learning based qos-aware routing in knowledge-defined networking," in *Quality, Reliability, Security and Robustness in Heterogeneous Systems* (T. Q. Duong, N.-S. Vo, and V. C. Phan, eds.), (Cham), pp. 14–26, Springer International Publishing, 2019.

[17] L. Müller, R. Oliveira, M. Caggiani Luizelli, L. Gaspary, and M. Barcellos, "Survivor: an enhanced controller placement strategy for improving sdn survivability," 12 2014.

[18] A. Ashraf, Z. Iqbal, M. Khan, U. Tariq, S. Kadry, and S.-o. Par, "Scalable offloading using machine learning methods for distributed multi-controller architecture of sdn networks," *The Journal of Supercomputing*, pp. 1–12, 01 2022.

[19] L. Zhao, J. Hua, Y. Liu, W. Qu, S. Zhang, and S. Zhong, “Distributed traffic engineering for multi-domain software defined networks,” pp. 492–502, 07 2019.

[20] Y. Liu, Z. Laiping, J. Hua, W. Qu, S. Zhang, and S. Zhong, “Distributed traffic engineering for multi-domain sdn without trust,” *IEEE Transactions on Cloud Computing*, pp. 1–1, 2021.

[21] H. K. Ravuri, M. T. Vega, J. van der Hooft, T. Wauters, B. Da, and F. De Turck, “On routing scalability in flat sdn architectures,” in *2020 11th International Conference on Network of the Future (NoF)*, pp. 23–27, 2020.

[22] H. Ravuri, M. Torres Vega, J. van der Hooft, T. Wauters, and F. De Turck, “A scalable hierarchically distributed architecture for next-generation applications,” *Journal of Network and Systems Management*, vol. 30, 01 2022.

[23] X. Ji, W. Xu, C. Zhang, and B. Liu, “A three-level routing hierarchy in improved sdn-mec-vanet architecture,” in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–7, 2020.

[24] J. Hu, C. Lin, and P. Zhang, “Performance evaluation and optimization of hierarchical routing in sdn control plane,” *Chinese Journal of Electronics*, vol. 27, no. 2, pp. 342–350, 2018.

[25] M. Karakus and A. Durresi, “A scalable inter-as qos routing architecture in software defined network (sdn),” in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pp. 148–154, 2015.

[26] L. Zhao, J. Hua, X. Ge, and S. Zhong, “Traffic engineering in hierarchical sdn control plane,” in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pp. 189–194, 2015.

[27] A. Kadhim, S. A. Hosseini Seno, and R. Shihab, “Routing strategy for internet of vehicles based on hierarchical sdn and fog computing,” *Journal of University of Babylon for Pure and Applied Sciences*, vol. 26, pp. 309–319, 12 2018.

[28] F. Y. Okay and S. Ozdemir, “Routing in fog-enabled iot platforms: A survey and an sdn-based solution,” *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4871–4889, 2018.

[29] N. Noorani and S. A. Hosseini Seno, “Sdn- and fog computing-based switchable routing using path stability estimation for vehicular ad hoc networks,” *Peer-to-Peer Networking and Applications*, vol. 13, 05 2020.

[30] J. Hua, L. Zhao, S. Zhang, Y. Liu, X. Ge, and S. Zhong, "Topology-preserving traffic engineering for hierarchical multi-domain sdn," *Computer Networks*, vol. 140, pp. 62–77, 2018.

[31] J.-J. Huang, Y.-Y. Chen, C. Chen, and Y. H. Chu, "Weighted routing in hierarchical multi-domain sdn controllers," in *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 356–359, 2015.

[32] B. Genge and P. Haller, "A hierarchical control plane for software-defined networks-based industrial control systems," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 73–81, 2016.

[33] Z. Eghbali and M. Zolfy Lighvan, "A hierarchical approach for accelerating iot data management process based on sdn principles," *Journal of Network and Computer Applications*, vol. 181, p. 103027, 02 2021.

[34] M. Kumar and R. S. Raw, "A novel routing protocol for hierarchical software defined vehicular adhoc network," in *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 771–775, 2022.

[35] G.-m. Lee, C.-w. Lee, and B.-h. Roh, "Qos support path selection for inter-domain flows using effective delay and directed acyclic graph in multi-domain sdn," *Electronics*, vol. 11, no. 14, 2022.

[36] X. Chai and H. Xu, "Hierarchical sdn multi-controller placement strategy based on improved aquila optimizer," in *2022 IEEE 5th International Conference on Electronics and Communication Engineering (ICECE)*, pp. 119–124, 2022.

[37] A. Abuarqoub, "A review of the control plane scalability approaches in software defined networking," *Future Internet*, vol. 12, p. 49, 03 2020.

[38] H. Yao, X. Yuan, P. Zhang, J. Wang, C. Jiang, and M. Guizani, "Machine learning aided load balance routing scheme considering queue utilization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7987–7999, 2019.

[39] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *2016 IEEE International Conference on Services Computing (SCC)*, pp. 25–33, 2016.

[40] N. Geng, T. Lan, V. Aggarwal, Y. Yang, and M. Xu, "A multi-agent reinforcement learning perspective on distributed traffic engineering," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pp. 1–11, 2020.

[41] D. K. Dake, J. D. Gadze, G. S. Klogo, and H. Nunoo-Mensah, "Multi-agent reinforcement learning framework in sdn-iot for transient load detection and prevention," *Technologies*, vol. 9, no. 3, p. 44, 2021.

[42] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *Acm computing surveys (csur)*, vol. 53, no. 2, pp. 1–33, 2020.

[43] T. Fu, Y. Peng, P. Liu, H. Lao, and S. Wan, "Distributed reinforcement learning-based memory allocation for edge-plcs in industrial iot," *Journal of Cloud Computing*, vol. 11, 10 2022.

[44] T. Kovac Martínez, "Reinforcement learning aplicado a videojuegos," 2022.

[45] A. Servin and D. Kudenko, *Multi-agent Reinforcement Learning for Intrusion Detection*, pp. 211–223. 02 2008.

[46] D. F. Aguirre Moreno *et al.*, "Modelo algorítmico para alta disponibilidad en transporte de volúmenes crecientes de tráfico variable en redes ópticas,"

[47] N. Akchurina, *Multi-agent reinforcement learning algorithms*. PhD thesis, Paderborn, Univ., Diss., 2010, 2010.

[48] M. Salcedo Bosch, "Data integration strategies for distributed reinforcement learning in robotics," Master's thesis, Universitat Politècnica de Catalunya, 2020.

[49] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, *et al.*, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[50] Z. Huang, *Distributed reinforcement learning for autonomous driving*. PhD thesis, Carnegie Mellon University, 2022.

[51] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[52] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.

[53] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[54] M. W. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, N. Momchev, D. Sinopalnikov, P. Stańczyk, S. Ramos, A. Raichuk, D. Vincent, *et al.*, "Acme: A research framework for distributed reinforcement learning," *arXiv preprint arXiv:2006.00979*, 2020.

[55] Danny McPherson, Russ White, and Sangli Srihari, "Intelligent Routing - Practical BGP." ISBN: 9780321127006.

[56] B. Dai, Y. Cao, Z. Wu, Z. Dai, R. Yao, and Y. Xu, "Routing optimization meets machine intelligence: A perspective for the future network," *Neurocomputing*, vol. 459, pp. 44–58, 2021.

[57] M. Karakus and A. Durresi, "Quality of service (qos) in software defined networking (sdn): A survey," *Journal of Network and Computer Applications*, vol. 80, pp. 200–218, 2017.

[58] L. Becerra, J. L. Bañol, and J. Padilla, "Un estudio sobre algoritmos basados en restricciones: objetivos ingeniería de tráfico y calidad de servicio," *Entre Ciencia e Ingeniería*, vol. 11, no. 21, pp. 103–111, 2017.

[59] A. Perez, "7 - quality of service principles," in *Implementing IP and Ethernet on the 4G Mobile Network* (A. Perez, ed.), pp. 117–135, Elsevier, 2017.

[60] A. Campbell, G. Coulson, and D. Hutchison, "A quality of service architecture," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 2, pp. 6–27, 1994.

[61] T. K. Randhawa, *Network Impact Modeling and Analysis: A QoS Perspective*. PhD thesis, The University of Western Ontario (Canada), 2020.

[62] J. P. Arango, L. A. Portilla, and J. C. Cuéllar, "Procedimiento para implementar qos en la capa de acceso en redes de próxima generación enfocado en el servicio de voz," *Sistemas & Telemática*, vol. 11, no. 25, pp. 85–104, 2013.

[63] A. Mohamad-Mezher, *Contributions to provide a QoS-aware self-configured framework for video-streaming services over ad hoc networks*. PhD thesis, 04 2016.

[64] "Software-Defined Networking (SDN) Definition," *Open Networking Foundation. https://opennetworking.org/sdn-definition/*.

[65] W. Xia, Y. Wen, C. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *Communications Surveys & Tutorials, IEEE*, vol. 17, pp. 27–51, 01 2015.

[66] Open Networking Foundation, “Openflow switch specification version 1.3.0.” https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf, 2014.

[67] IBM, “¿Qué es una API REST?.” https://www.ibm.com/mx-es/topics/rest-apis. Accessed: 2023-8-8.

[68] F. Bannour, S. Souihi, and A. Mellouk, “Distributed sdn control: Survey, taxonomy and challenges,” *IEEE Communications Surveys & Tutorials*, vol. PP, pp. 1–1, 12 2017.

[69] E. Amiri, E. Alizadeh, and K. Raeisi, “An efficient hierarchical distributed sdn controller model,” in *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, pp. 553–557, 2019.

[70] K. A. Rasol and J. Domingo-Pascual, “Multi-level hierarchical controller placement in software defined networking,” in *Selected Papers from the 12th International Networking Conference: INC 2020 12*, pp. 131–145, Springer, 2021.

[71] O. Blial, M. Ben Mamoun, and B. Redouane, “An overview on sdn architectures with multiple controllers,” *Journal of Computer Networks and Communications*, vol. 2016, pp. 1–8, 01 2016.

[72] M. F. Körner, *Software defined networking based data-center services*. Technische Universitaet Berlin (Germany), 2015.

[73] S. Hassas Yeganeh and Y. Ganjali, “Kandoo: a framework for efficient and scalable offloading of control applications,” in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 19–24, 2012.

[74] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, “B4: Experience with a globally-deployed software defined wan,” *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[75] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, *et al.*, “Taking the edge off with espresso: Scale, reliability and programmability for global internet peering,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 432–445, 2017.

[76] Y. Fu, J. Bi, K. Gao, Z. Chen, J. Wu, and B. Hao, “Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks,” in *2014 IEEE 22nd International Conference on Network Protocols*, pp. 569–576, 2014.

[77] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey, "Toward adaptive and scalable openflow-sdn flow control: A survey," *IEEE Access*, vol. 7, pp. 107346–107379, 2019.

[78] K. Saravanan and A. Z. Kouzani, "Advancements in on-device deep neural networks," *Information*, vol. 14, no. 8, p. 470, 2023.

[79] P. Humblet, "Another adaptive distributed shortest path algorithm," *IEEE Transactions on Communications*, vol. 39, no. 6, pp. 995–1003, 1991.

[80] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pp. 287–290, 2022.

[81] S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti, and H. Zedan, "A comprehensive survey on vehicular ad hoc network," *Journal of network and computer applications*, vol. 37, pp. 380–392, 2014.

[82] R. Steinmetz and K. Wehrle, *Peer-to-peer systems and applications*, vol. 3485. Springer, 2005.

[83] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 2012.

[84] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pp. 241–246, 2014.

[85] O. Hersent, D. Boswarthick, and O. Elloumi, *The internet of things: Key applications and protocols*. John Wiley & Sons, 2011.

[86] M. Becker, A. Schmidt, M. Orehek, and T. Nolte, "Saving energy by means of dynamic load management in embedded multicore systems," in *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, pp. 11–20, IEEE, 2014.

[87] M. KB, "Constrained Shortest Path First," Internet-Draft draft-manayya-constrained-shortest-path-first-02, Internet Engineering Task Force, Feb. 2010. Work in Progress.

[88] Sprint, Overland Park, KS, "Sprint IP network performance." https://www.sprint.net/tools/ip-network-performance, 2011.

[89] C. Perkins and E. Royer, “Ad-hoc on-demand distance vector routing,” in *Proceedings WMCSA’99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, 1999.

[90] B. Karp and H.-T. Kung, “Gpsr: Greedy perimeter stateless routing for wireless networks,” in *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 243–254, 2000.

[91] T. Lu, S. Chang, and W. Li, “Fog computing enabling geographic routing for urban area vehicular network,” *Peer-to-Peer Networking and Applications*, vol. 11, pp. 749–755, 2018.

[92] N. Noorani and S. A. H. Seno, “Routing in vanets based on intersection using sdn and fog computing,” in *2018 8th International Conference on Computer and Knowledge Engineering (ICCKE)*, pp. 339–344, IEEE, 2018.

[93] J. Hua, L. Zhao, S. Zhang, Y. Liu, X. Ge, and S. Zhong, “Topology-preserving traffic engineering for hierarchical multi-domain sdn,” *Computer Networks*, vol. 140, pp. 62–77, 2018.

[94] V. Capasso and V. Capasso, *Introduction to Continuous-Time Stochastic Processes*. Springer, 2021.

[95] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, M. Shahzad, and S. P. Mohanty, “Dolphin: Dynamically optimized and load balanced path for inter-domain sdn communication,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 331–346, 2020.

[96] B. Heller, R. Sherwood, and N. McKeown, “The controller placement problem,” *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 473–478, 2012.

[97] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, “Opposition-based differential evolution,” *IEEE Transactions on Evolutionary computation*, vol. 12, no. 1, pp. 64–79, 2008.

[98] M. Ahmed, R. Seraj, and S. M. S. Islam, “The k-means algorithm: A comprehensive survey and performance evaluation,” *Electronics*, vol. 9, no. 8, p. 1295, 2020.

[99] D. Wang, D. Tan, and L. Liu, “Particle swarm optimization algorithm: an overview,” *Soft computing*, vol. 22, pp. 387–408, 2018.

[100] M. Karakus and A. Durresi, “Quality of service (qos) in software defined networking (sdn): A survey,” *Journal of Network and Computer Applications*, vol. 80, pp. 200–218, 2017.

[101] C. J. Quimbayo Rodríguez, *Propuesta metodológica para la selección de controladores de redes SDN a nivel empresarial*. PhD thesis, Universidad Santo Tomás, 2020.

[102] K. S. Sahoo, S. K. Mishra, S. Sahoo, and B. Sahoo, "Software defined network: the next generation internet technology," *International Journal of Wireless and Microwave Technologies(IJWMT)*, 2017.

[103] G. Wright, "Northbound interface/southbound interface." https://www.techtarget.com/whatis/definition/northbound-interface-southbound-interface, July 2022. Accessed: 2023-7-18.

[104] Y. Li, Z.-P. Cai, and H. Xu, "Llmp: exploiting lldp for latency measurement in software-defined data center networks," *Journal of Computer Science and Technology*, vol. 33, pp. 277–285, 2018.

[105] S. Achleitner, N. Bartolini, T. He, T. La Porta, and D. Z. Tootaghaj, "Fast network configuration in software defined networking," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1249–1263, 2018.

[106] E. Haleplidis, K. Pentikousis, S. Denazis, J. Salim, D. Meyer, and O. Koufopavlou, "Software-defined networking (sdn): Layers and architecture terminology," *IRTF*, 01 2015.

[107] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.

[108] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," *IEEE*, vol. 7, pp. 55916–55950, 2019.

[109] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[110] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16*, pp. 317–328, Springer, 2005.

[111] J. Mackeprang, D. B. R. Dasari, and J. Wrachtrup, "A reinforcement learning approach for quantum state engineering," *Quantum Machine Intelligence*, vol. 2, pp. 1–14, 2020.

[112] A. Zai and B. Brown, *Deep reinforcement learning in action*. Manning Publications, 2020.

[113] L. Al Shalabi and Z. Shaaban, "Normalization as a preprocessing engine for data mining and the approach of preference matrix," in *2006 International conference on dependability of computer systems*, pp. 207–214, IEEE, 2006.

[114] T. Zhang, Q. Ye, J. Bian, G. Xie, and T.-Y. Liu, "Mfvfd: A multi-agent q-learning approach to cooperative and non-cooperative tasks," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21* (Z.-H. Zhou, ed.), pp. 500–506, International Joint Conferences on Artificial Intelligence Organization, 8 2021.

[115] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, IEEE, 2016.

[116] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colombian conference on communications and computing (COLCOM)*, pp. 1–6, Ieee, 2014.

[117] S. H. Yeganeh and Y. Ganjali, "Beehive: Simple distributed programming in software-defined networks," in *Proceedings of the Symposium on SDN Research*, pp. 1–12, 2016.

[118] "NumPy: the absolute basics for beginners — NumPy v1.25 manual." https://numpy.org/doc/stable/user/absolute_beginners.html. Accessed: 2023-8-12.

[119] Y.-C. Wang and S.-Y. 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*, vol. 15, no. 4, pp. 1422–1434, 2018.

[120] S. Srivastava, S. Anmulwar, A. Sapkal, T. Batra, A. K. Gupta, and V. Kumar, "Comparative study of various traffic generator tools," in *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, pp. 1–6, IEEE, 2014.

[121] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.