UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

Técnicas de Aprendizaje por Refuerzo para la Delegación de Tareas (*Computation Offloading*) en Aplicaciones de Vehículo Conectado

Autor:

D. Mieszko Jan Ferens Michalek

Tutores:

D. Diego Hortelano Haro
D. Ignacio de Miguel Jiménez

VALLADOLID, ABRIL 2022

TRABAJO FIN DE MÁSTER

TÍTULO:	**Técnicas de Aprendizaje por Refuerzo para la Delegación de Tareas (*Computation Offloading*) en Aplicaciones de Vehículo Conectado**
AUTOR:	**D. Mieszko Jan Ferens Michalek**
TUTORES:	**D. Diego Hortelano Haro**
	D. Ignacio de Miguel Jiménez
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dña. Patricia Fernández del Reguero**
SUPLENTE:	**Dña. Lourdes Pelaz Montes**
VOCAL:	**D. Javier Manuel Aguilar Pérez**
SUPLENTE:	**D. Jaime Gómez Gil**
SECRETARIO:	**D. Ramón de la Rosa Steinz**
SUPLENTE:	**Dña. María Jesús Verdú Pérez**

Resumen

Una de las tecnologías esenciales para ofrecer servicios relacionados con el vehículo conectado es MEC (*Multi-Access Edge Computing*). En ella, parte de los recursos de computación en la nube (*cloud*) se deslocalizan de los grandes centros de datos (*datacenters)* para situarse en una jerarquía que se extiende hasta el borde de la red de acceso radio (*edge*). Esto permite aumentar la capacidad de procesamiento del sistema en su conjunto y reducir la latencia de los servicios.

Este Trabajo Fin de Máster se enmarca en la exploración del uso de MEC en aplicaciones de vehículo conectado en entornos metropolitanos centrándose en mecanismos de delegación de tareas de computación (*computation offloading*). Se trata de decidir si las tareas relacionadas con una cierta aplicación se realizarán de forma local en el equipo de usuario (embarcado en el vehículo) o si se delegará su realización, ya sea parcial o totalmente (y siempre que la aplicación lo permita), a la nube, un servidor MEC o un punto de acceso RSU (*Roadside Unit*). A la hora de delegar las tareas se busca la optimización de alguna métrica, como por ejemplo maximizar la tasa de éxito para procesar las aplicaciones, pero haciendo al mismo tiempo un uso eficiente de los recursos de cómputo y de los enlaces de comunicación. Concretamente, en el TFM se han propuesto e implementado distintos mecanismos de delegación de tareas basados en técnicas de aprendizaje por refuerzo (*reinforcement learning*), y se han comparado entre sí y con diversas heurísticas en un entorno de simulación. Especialmente se analiza el impacto que tiene la incertidumbre provocada por la variación de los tiempos de procesamiento en el rendimiento de distintos algoritmos.

Palabras Clave

Aprendizaje por refuerzo, agente, Deep Q-learning, ChainerRL, OpenAI Gym, delegación de tareas de computación, MEC

Abstract

An essential technology for providing services related to connected vehicles is MEC (Multi-Access Edge Computing). This technology delocalizes a portion of the resources for cloud computation from big datacenters to establish a hierarchy which extends to the edge of the radio access network. This enables the increase in processing capacity of the system as a whole and reduces the latency of its services.

This Master's Thesis explores the use of MEC for applications of connected vehicles in metropolitan environments, focusing on computational offloading mechanisms. The idea is to decide if the tasks related to a specific application are to be processed locally in the user's equipment (embarked in the vehicle) or if their processing is to be delegated, partially or fully (as long as the application allows it), to the cloud, a MEC server or an RSU (Roadside Unit) access point. For the delegation of tasks, we search for the optimization of some metric, like the maximization of the success rate for processing the applications, while making an efficient use of the network's computational resources and communication links. Specifically, in this Master's Thesis different computation offloading mechanisms based on reinforcement learning techniques have been proposed and implemented, and compared with each other and with different heuristics in a simulation environment. Especially, the impact of uncertainty due to the variation of processing times on performance is analyzed.

Keywords

Reinforcement learning, agent, Deep Q-learning, ChainerRL, OpenAI Gym, computation offloading, MEC

Agradecimientos

Deseo realizar un especial agradecimiento a D. Ignacio de Miguel Jiménez y D. Diego Hortelano Haro por tutorizar la realización de este TFM. En concreto agradezco el apoyo proporcionado en la temática de delegación de tareas y redes vehiculares, ya que los conocimientos de D. Diego Hortelano Haro fueron de extrema utilidad para plantear el problema y su solución.

También merece una mención especial el Dpto. de Teoría de la Señal y Comunicaciones e Ingeniería Telemática de la Universidad de Valladolid por permitir la realización de este TFM de forma repentina y en un estricto plazo de tiempo. El apoyo de D. Ramón José Durán Barroso y D. Ignacio de Miguel Jiménez fue fundamental para la exitosa realización del trabajo.

Este trabajo ha sido financiado por la Consejería de Educación de la Junta de Castilla y León, y el Fondo Europeo de Desarrollo Regional (proyecto VA231P20), y el Ministerio de Ciencia e Innovación y la Agencia Estatal de Investigación (proyecto PID2020-112675RB-C42 financiado por MCIN/AEI/10.13039/501100011033).

Índice

1. Introducción 10
   1.1. Motivación 10
   1.2. Objetivos 10
   1.3. Estructura 11
   1.4. Metodología 11
2. Estado del arte 12
   2.1. Aprendizaje por refuerzo (RL) 12
   2.2. Delegación de tareas 13
3. Descripción del entorno 15
   3.1. Concepto 15
   3.2. Elementos del entorno de simulación 16
      3.2.1. Red 16
      3.2.2. Nodos 17
      3.2.3. Aplicaciones 18
   3.3. Funcionamiento del entorno de simulación 19
      3.3.1. Envío de datos 19
      3.3.2. Generación de peticiones 19
      3.3.3. Reservas de recursos de computación en los nodos y procesamiento 20
   3.4. Ruido en el tiempo de procesamiento 22
   3.5. Agente 23
      3.5.1. Observación y acción 23
      3.5.2. Recompensa 24
4. Algoritmos implementados y métricas de rendimiento 26
   4.1. Simulador 26
   4.2. Algoritmos de aprendizaje por refuerzo (RL) 26
      4.2.1. DQN (Deep-Q-Network) 27
      4.2.2. DDQN (Double Deep-Q-Network) 27
      4.2.3. SARSA (State-Action-Reward-State-Action) 27
      4.2.4. PAL (Persistent Advantage Learning) 27
      4.2.5. TRPO (Trust Region Policy Optimization) 27
   4.3. Heurísticas 28
      4.3.1. LP (Local Processing) 28
      4.3.2. CP (Cloud Processing) 28

4.3.3. UDP (Uniform Distribution Processing) 28
4.3.4. MDP (Max Distance Processing) 28
4.4. Métricas 29
4.5. Parámetros de simulación 29
5. Experimentos iniciales (redes con una sola RSU) 31
5.1. Planteamiento 31
5.1.1. Topologías 31
5.1.2. Sets de aplicaciones 33
5.1.3. Escenarios 34
5.2. Resultados 35
5.2.1. Análisis global 35
5.2.2. Consistencia entre políticas de agentes de RL de entrenamientos distintos 40
5.2.3. Entrenamientos prolongados 42
5.2.4. Exploración linealmente decadente 43
5.2.5. Impacto del ruido y de la planificación 44
5.2.6. Conclusiones 46
6. Experimento práctico (red con varias RSU) 47
6.1. Planteamiento 47
6.1.1. Topología y set de aplicaciones 47
6.1.2. Escenarios 48
6.2. Resultados 49
6.2.1. Entrenamiento múltiple versus individual 49
6.2.2. Variación de número de vehículos 51
6.2.3. Variación del coeficiente de variación del ruido 54
6.2.4. Conclusiones 59
7. Conclusiones y líneas futuras 60
7.1. Conclusiones 60
7.2. Líneas futuras 62
Referencias 63

1. Introducción

1.1. Motivación

La popularidad de las aplicaciones IoT (*Internet of Things*) está en constante crecimiento, dando lugar a sistemas con requisitos de cómputo cada vez mayores. Entre las distintas ramas a las que este ámbito se aplica está el de las aplicaciones vehiculares, o vehículos conectados.

Debido a que los dispositivos usados para este tipo de aplicaciones suelen ser dispositivos portátiles o móviles (como los vehículos), sus capacidades de cómputo están limitadas. Bien sea por limitaciones de *hardware* o de otro tipo, como por ejemplo energéticas, los dispositivos finales no pueden seguir el ritmo al que la complejidad de los sistemas aumenta.

Para solucionar este problema en recientes años ha aparecido el concepto de delegación de tareas (*computation offloading*). Mediante el uso de recursos fuera de los dispositivos involucrados, se puede delegar el procesamiento de los datos de las aplicaciones a nodos que no son el origen de los mismos. Esto lleva al siguiente problema, y es que ciertas aplicaciones pueden tener requisitos de latencia muy estrictos, y soluciones como la computación en la nube (*cloud computing*) pueden no ser suficientes para cumplir con esos requisitos debido a la elevada distancia con los centros de cálculo.

Como posible solución está el reciente concepto definido por el Instituto de Estándares de Telecomunicaciones Europeo (ETSI) denominado MEC (*Multi-Access Edge Computing*) [1], que propone la ubicación de servidores o centros de cálculo más cercanos a los dispositivos. Con esto se logra reducir los retardos asociados a las comunicaciones.

Un punto clave estudiado ampliamente en los últimos años es cómo tomar las decisiones sobre qué tareas deben delegarse y cómo hacerlo. Para ello, una de las soluciones más populares es el uso del aprendizaje por refuerzo (*Reinforcement Learning*, RL). Este estudio pretende continuar con la misma idea, ampliando ciertos aspectos.

1.2. Objetivos

En este Trabajo Fin de Máster se propone el uso de algoritmos de aprendizaje por refuerzo profundo (*Deep Reinforcement Learning*, DRL) para la delegación de tareas en una red que da soporte a aplicaciones vehiculares. A continuación, se muestra la lista de objetivos:

- Resolver el problema de la delegación de tareas con algoritmos de aprendizaje por refuerzo profundo y evaluar sus prestaciones mediante simulaciones.
- Mostrar un caso práctico del problema en una red con varios puntos de acceso.
- Comparar el rendimiento de los agentes de aprendizaje por refuerzo entre sí y con algoritmos heurísticos.
- Analizar el impacto de la incertidumbre en el tiempo de procesamiento que requerirá una tarea (esto es, la diferencia entre lo que realmente se tarda en hacer una tarea y lo que se preveía inicialmente que se tardaría) en las prestaciones, evaluando si los mecanismos de aprendizaje por refuerzo son más robustos que las heurísticas en esos escenarios con incertidumbre.

1.3. Estructura

La memoria está estructurada para mostrar inicialmente toda la información necesaria para contextualizar el problema, seguido de las explicaciones relevantes sobre la realización de los experimentos para, por último, mostrar los resultados y conclusiones de estos.

Después de esta primera sección, la introducción (1), se encuentra el estado del arte (2). En dicha sección se comentan tanto los fundamentos del aprendizaje por refuerzo como de la delegación de tareas.

La tercera sección (3) muestra todas las explicaciones necesarias para comprender el funcionamiento del entorno implementado mediante un simulador, desde una descripción conceptual hasta los elementos del mismo. En la misma línea, la cuarta sección (4) expone los aspectos técnicos involucrados en la realización de los experimentos. Esto incluye los algoritmos empleados y las métricas utilizadas para medir sus rendimientos.

La quinta sección (5) muestra unos experimentos iniciales, siendo estos relativamente simples, para demostrar el funcionamiento de nuestro simulador y la validez de los resultados, así como ciertas conclusiones acerca del uso de RL para la delegación de tareas junto con ideas sobre la presencia de ruido de procesamiento. La sexta sección (6) aumenta la complejidad de los experimentos, mostrando una red con varios puntos de acceso y los resultados asociados.

Por último, la séptima sección (7) presenta las conclusiones obtenidas a lo largo del estudio y las líneas futuras más relevantes.

1.4. Metodología

La metodología seguida en la realización del estudio consistió en lo siguiente: en primer lugar, se analizaron otros estudios similares, extrayendo las conclusiones más relevantes y observando puntos no abordados. Seguido de esto, se implementó un simulador que pudiera representar adecuadamente los escenarios necesarios. Por último, se plantearon los escenarios que se deseaban comprobar, se obtuvieron los resultados y se analizaron.

2. Estado del arte

El aprendizaje por refuerzo es uno de los métodos más populares hoy en día para la resolución de problemas de delegación de tareas [2]. Esta sección ofrece una breve explicación de qué es el aprendizaje por refuerzo basada en [3] y [4], así como los conceptos fundamentales de la delegación de tareas.

2.1. Aprendizaje por refuerzo (RL)

Cuando hablamos de aprendizaje por refuerzo (*Reinforcement Learning*, RL) nos referimos a uno o varios agentes interactuando con un entorno. Las interacciones de un agente se basan en la toma de acciones que suelen conllevar resultados en el entorno. Estos resultados se muestran al agente en forma de observaciones y de recompensas inmediatas. El objetivo de un agente es obtener el mejor resultado posible, es decir, maximizar la recompensa acumulada a largo plazo.

A cada observación la seguirá una acción, lo que conllevará una recompensa. Este ciclo se repite dando lugar a la denominada historia, que es la secuencia de observaciones, acciones y recompensas. Mediante esto se puede definir el sistema como una máquina de estados, cuyos estados son una función de la historia. Al tomar una acción se pasa a un nuevo estado. Estos cambios de estado tendrán vinculados recompensas, por lo que, conociendo todos los estados del sistema, la probabilidad de pasar a otro estado en función de la acción tomada, y las recompensas asociadas, el agente puede tomar las acciones óptimas en cada estado.

Dicho esto, normalmente se desconoce al menos parte de la información del entorno. Esto implica que los agentes necesitan descubrir los estados explorando, es decir, un agente no siempre tomará la acción que considere óptima con el objetivo de descubrir nuevos resultados. La explotación del entorno (toma de las acciones óptimas conocidas) y la exploración (toma de acciones no necesariamente óptimas) deben ser balanceadas. Aunque se prioricen aquellas acciones que son consideradas óptimas, inevitablemente se debe seguir algún método de exploración. En sistemas complejos, alcanzar todos los posibles estados no suele ser posible, por lo que es importante que el agente generalice. Esto implica la capacidad de tomar la acción óptima en un estado nunca antes visto.

En términos específicos, la política es una función que indica la acción a tomar en cada estado (política determinista) o, de forma más general, una función que indica la probabilidad de tomar una acción concreta cuando se está en un determinado estado (política estocástica). La acción mapeada a cada estado se actualiza a medida que el agente toma acciones y recibe nuevas observaciones y recompensas. Se deben distinguir dos tipos fundamentales de algoritmos de RL, los *on-policy* y los *off-policy*. La diferencia fundamental es qué política se evalúa o mejora (*target policy*) y qué política se sigue para actuar y obtener los datos (*behaviour policy*).

En el caso de los métodos *on-policy*, ambas *target* y *behaviour policies* son la misma. Esto quiere decir que el agente mejora la misma política que sigue en la toma de sus acciones. La fundamental característica de este método es que durante la evaluación se tiene en cuenta la exploración incluida en la política. Por otra parte, para los métodos *off-policy,* la *target policy* y la *behaviour policy* no son las mismas, ya que la política usada durante la evaluación no es la misma que la que se usa para la toma de acciones.

Con el objetivo de optimizar una política en muchas ocasiones se utiliza el concepto de función de valor. La función de valor indica la recompensa media acumulada que se obtendrá a largo plazo partiendo desde un estado (o desde que se elige una acción concreta) y tomando a partir de ese momento las acciones dictadas por la política. En resumen, es una

métrica de lo bueno que es estar en un estado (función de valor del estado) o seleccionar una acción concreta en un estado (función de valor de la acción). El cálculo de las funciones de valor implica el uso de las denominadas ecuaciones de Bellman, las cuales relacionan la función de valor de un estado (o acción) con las funciones de valor de los estados (o acciones) inmediatamente siguientes.

La denominada recompensa acumulada también se suele llamar retorno. En su cálculo se tiene en cuenta la recompensa inmediata, pero también las recompensas que se irán recibiendo en los instantes posteriores. Para ello el factor de descuento (γ) indica el peso de las recompensas posteriores, y es útil por tanto para darlas mayor o menor importancia. Es importante diferenciar entre tareas continuas y tareas episódicas. En las tareas a resolver mediante aprendizaje por refuerzo puede haber un estado final (tarea episódica) o no (tarea continua). Especialmente para las tareas continuas el factor de descuento debe ser menor que 1 para evitar contar los infinitos estados lejanos (y por tanto que la recompensa acumulada se incremente sin límite).

2.2. Delegación de tareas

El concepto de delegación de tareas (*computation offloading*) consiste en el procesamiento parcial o total de aplicaciones fuera del dispositivo en el que se generan. Es una técnica fundamental para el desarrollo de sistemas y aplicaciones con requerimientos exigentes en dispositivos que típicamente no tienen los recursos mínimos para dar el soporte necesario.

Muy ligado a la delegación de tareas está el reciente concepto definido por el Instituto de Estándares de Telecomunicaciones Europeo (ETSI) denominado MEC (*Multi-Access Edge Computing*) [1]. Este consiste en ofrecer servicios similares a los de servidores de nube (gran capacidad de cómputo), pero estando localizados más cerca de los puntos a los que proveen, reduciendo en gran medida el retardo.

El estudio de [2] resume las últimas tendencias en problemas de delegación de tareas empleando aprendizaje por refuerzo entre otras técnicas. Los resultados muestran la gran popularidad de RL, y sobre todo de DRL (*Deep RL*), para la resolución de dichos problemas. Entre ellos se encuentra la delegación de tareas en el caso concreto de las aplicaciones vehiculares. Varios conceptos importantes pueden definirse para caracterizar el tipo de problema concreto abordado.

En primer lugar, se debe hablar del objetivo a lograr. Si bien existen distintas métricas que pueden mejorarse como: maximización del uso de los recursos de la red, maximización de las aplicaciones procesadas, maximización de la distribución de carga, minimización del consumo de energía, maximización de la seguridad, …; el objetivo más popular es minimizar la latencia de las aplicaciones. Este TFM se centra en gran medida en esta última.

En segundo lugar, el sistema puede definirse como centralizado o distribuido. Si es centralizado, el agente que controla la delegación de tareas es único y típicamente se encuentra en un servidor central en el sistema (por ejemplo, un servidor MEC). Por otro lado, en un caso descentralizado la delegación de tareas está dividida en varios agentes. Sus localizaciones normalmente serán o bien los puntos de acceso a la red, o los dispositivos de usuario que ejecutan las aplicaciones. La interdependencia de los agentes es variable pero no suelen conocer toda la información de la red, sino que cada uno recibe información parcial. En este estudio consideramos un sistema centralizado.

Otra forma de distinguir los planteamientos es fijándose en las aplicaciones. Decisiones como la de ejecutar múltiples aplicaciones en cada dispositivo (en paralelo o secuencialmente) o tener uno o varios dispositivos cambian drásticamente el estudio. A mayores, la posibilidad de dividir las aplicaciones por tareas y si existe dependencia entre

las mismas es otro punto de complejidad añadido. En este TFM consideramos múltiples dispositivos (vehículos) ejecutando múltiples aplicaciones en paralelo. Cada aplicación requiere el procesado de conjuntos de datos (a los que denominaremos paquetes de datos) cada cierto tiempo, y asumimos que el vehículo puede realizar el procesado de cada uno de esos paquetes de datos localmente o bien delegarlos a otros nodos de la red (siempre que se cumplan las restricciones de latencia) que luego devolverán el resultado de ese procesado al vehículo. Además, suponemos que no es necesario procesar todos los paquetes de datos de una aplicación en el mismo nodo, sino que paquetes de datos sucesivos pueden procesarse en nodos de computación distintos de la red.

Respecto a las redes consideradas, sus estructuras juegan un papel importante en diferenciar los problemas. Típicamente se consideran la capa de dispositivos finales (los que ejecutan las aplicaciones) y la capa de la computación en el borde o *edge* (donde se encuentran los recursos de computación cercanos a los dispositivos finales). Una tercera capa es la de la nube, que en muchos casos se añade por su utilidad para aplicaciones menos sensibles a la latencia y de grandes requisitos computacionales. En este caso se consideran las tres capas, con la segunda conteniendo no solo servidores MEC, sino también RSUs (*Roadside Units*). Las RSUs dotan de puntos de acceso a la red, además de actuar como nodos de cómputo.

Por último, consideraciones sobre qué elementos son variantes en el tiempo son muy notables. La complejidad aumenta a mayor número de este tipo de consideraciones. Ejemplos de esto son considerar tiempos de transmisión o propagación, niveles de energía, generación de peticiones o movilidad de los elementos del sistema como variantes en el tiempo. En este estudio solamente consideramos variante la generación de peticiones. Sin embargo, añadimos como variantes en el tiempo los tiempos de procesamiento para observar los efectos que esto produce. Esto es un factor que, hasta donde sabemos, no se ha considerado aún en otros estudios.

3. Descripción del entorno

Esta sección proporciona la información necesaria para comprender el planteamiento del entorno de simulación, desde el concepto básico hasta los elementos funcionales y procesos fundamentales para su funcionamiento. También proporciona una explicación y formulación matemática de la principal novedad del estudio, el ruido de procesamiento, y los detalles relevantes para el uso del entorno con agentes de aprendizaje por refuerzo.

3.1. Concepto

Con el objetivo de evaluar las prestaciones de distintos algoritmos de aprendizaje por refuerzo, creamos un entorno de simulación. Este entorno representa una red junto a vehículos que ejecutan una serie de aplicaciones. Con el fin de mejorar el rendimiento de dichas aplicaciones (concretamente en lo que se refiere a su latencia) la red provee un servicio de cómputo. Desde el punto de vista de los vehículos, el manejo de las peticiones está abstraído y debe ser manejado por el sistema. Para ello, un agente toma el papel de administrador. Podemos ver un esquema conceptual del sistema propuesto en la Figura 1.

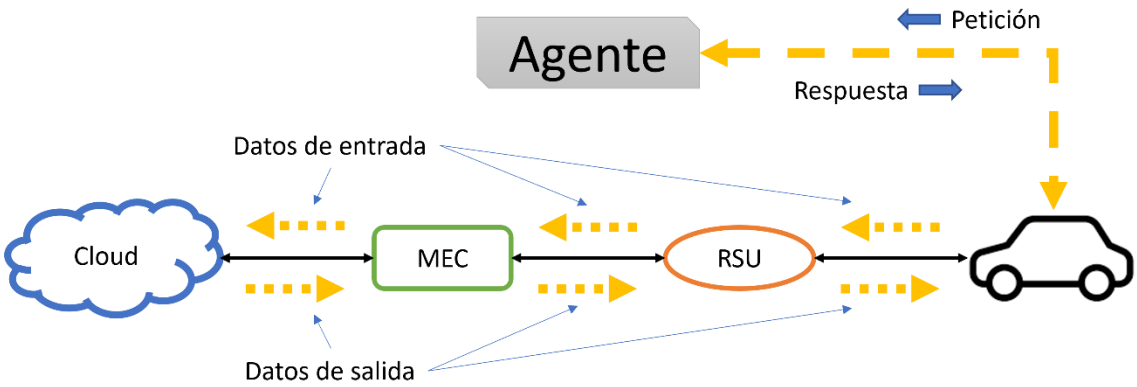

Figura 1. Esquema conceptual del sistema propuesto.

Los vehículos, a medida que generan peticiones correspondientes a sus aplicaciones, informan al agente. Este a su vez los responde sobre dónde deben procesar el conjunto de datos relacionado con cada petición. Con las respuestas del agente, los vehículos procesan localmente los datos o los envían a la red para que sean procesados en alguno de sus nodos.

En las simulaciones se tienen en cuenta los retardos producidos por el manejo y el envío de los datos de las aplicaciones, pero por simplicidad, se supone que las comunicaciones relacionadas con el plano de control de la red son instantáneas. También consideramos que los enlaces en la red tienen una tasa de transmisión y tiempos de propagación fijos. No se tendrá en cuenta la movilidad de los vehículos debido a que consideraremos sobre todo aplicaciones cuyas peticiones se procesan en el orden de las decenas de milisegundos, por lo que es razonable asumir que los vehículos están siempre conectados a la misma RSU en el momento de recibir el resultado de la ejecución.

Estas aproximaciones tienen sentido dado que el interés está en el procesamiento de las aplicaciones, y las variaciones debidas a la red (como la movilidad de los vehículos) no afectan a los resultados en gran medida si se consideran conjuntos de datos pequeños (con tiempos de procesamiento en el orden de milisegundos).

3.2. Elementos del entorno de simulación

El entorno de simulación implementado se compone de tres elementos fundamentales. En esta subsección se describen estos tres elementos junto con los parámetros relevantes: la red, los nodos y las aplicaciones.

3.2.1. Red

En primer lugar, tenemos la red simulada. Consideramos una red formada por cuatro tipos de nodos: nube, MEC, RSU y vehículo. La red debe estar compuesta por un solo nodo nube, uno o varios nodos MEC, uno o varios nodos RSU y uno o varios nodos vehículo. Consideramos una estructura en árbol, donde un tipo de nodo solo puede conectarse con los tipos adyacentes (por ejemplo, un MEC solo puede conectarse al nodo de nube y a las RSUs). Por tanto, la forma genérica de la red es la mostrada en la Figura 2.

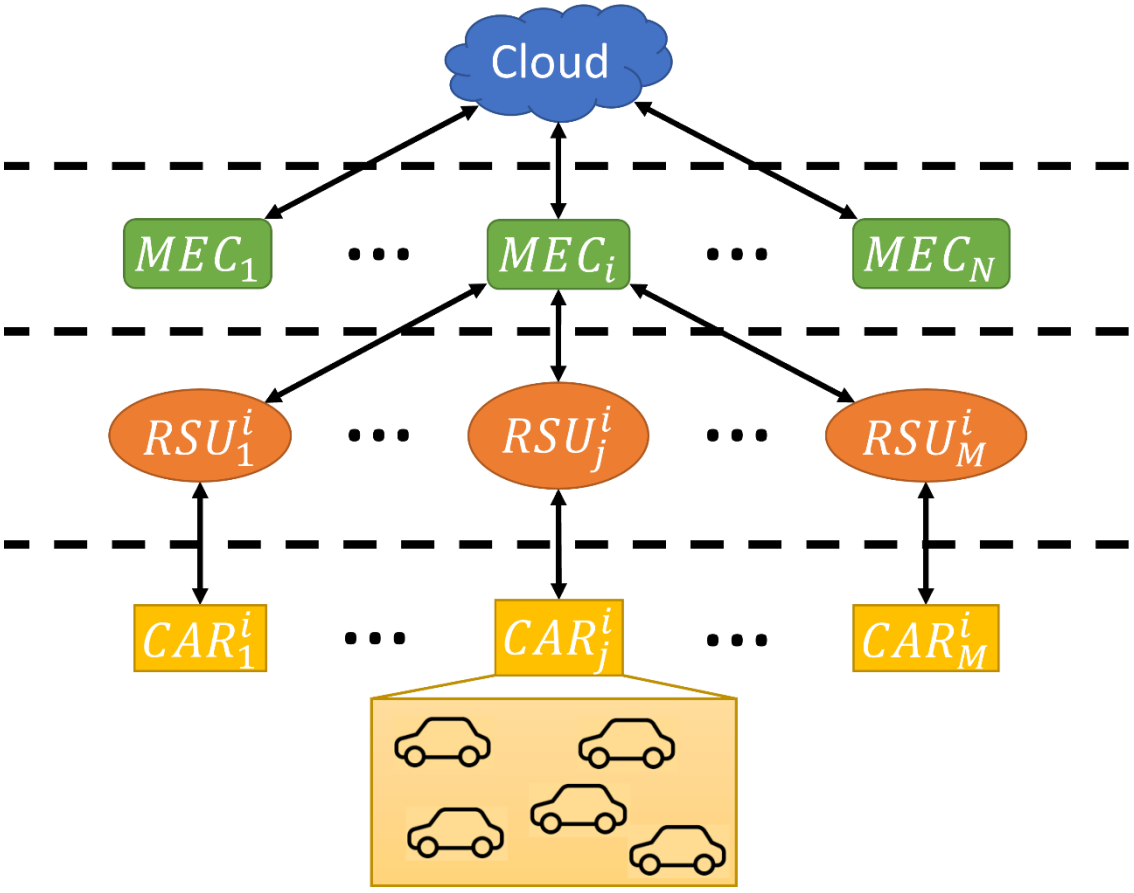

Figura 2. Esquema de la red genérica.

Los nodos vehículo (CAR) son excepcionales en el sentido de que representan un número entero de vehículos (uno o más). Todos los vehículos que estén sujetos a un mismo nodo vehículo utilizan la conexión con el RSU correspondiente. La red define un número total de vehículos y estos se reparten uniformemente por todos los nodos vehículo. En caso de que tras la división haya decimales se aproxima al entero más cercano, por lo que en la práctica solo puede haber un número total de vehículos que sea múltiplo del número de nodos vehículo. No hay movilidad de vehículos en la red.

A cada MEC puede conectarse una o más RSUs, y esto puede cambiar dependiendo del MEC. Por ejemplo, un MEC puede tener conectadas dos RSUs, pero otro al mismo tiempo puede tener tres.

Todos los enlaces entre nodos son bidireccionales y simétricos, con sus tasas de transmisión (r_{TX}) y retardos de propagación (t_{prop}) teniendo valores definidos e invariantes. No se consideran retardos de encolado en lo referido a la transmisión. Por simplificar, todos los enlaces desde un tipo de nodo hasta otro son iguales, y solamente puede haber diferencias entre enlaces que conectan nodos de distintos tipos. Por lo tanto, podemos definirlos según los nodos que interconectan:

$$Cloud - MEC \begin{cases} r_{TX}^{Cloud-MEC}(kbps) \\ t_{prop}^{Cloud-MEC}(ms) \end{cases} \qquad MEC - RSU \begin{cases} r_{TX}^{MEC-RSU}(kbps) \\ t_{prop}^{MEC-RSU}(ms) \end{cases}$$

$$RSU - CAR \begin{cases} r_{TX}^{RSU-Car}(kbps) \\ t_{prop}^{RSU-Car}(ms) \end{cases}$$

3.2.2. Nodos

Todos los nodos de la red se caracterizan por definir unos recursos de procesamiento. Estos recursos son los que usa cada nodo durante las simulaciones para procesar los paquetes de datos de las aplicaciones, con la excepción del ya mencionado nodo vehículo (CAR). Como cada nodo vehículo representa un cierto número de vehículos independientes, los recursos que define están replicados para cada vehículo (cada vehículo tiene recursos de procesamiento independientes de los demás).

Estos recursos consisten en un cierto número de núcleos de CPU (n_{cores}), cada uno de los cuales tiene su propia memoria para una cola de reservas de procesamiento (C_{buffer}). Los nodos también tienen definida la frecuencia de reloj (C_{clock}) para todos sus núcleos. Las características son comunes para los tipos de nodos, y solo se establecen diferencias entre distintos tipos de nodos. Así pues, podemos definirlos por tipos:

$$Cloud \begin{cases} n_{cores}^{Cloud} = \infty \\ C_{buffer}^{Cloud}(ms) = \infty \\ C_{clock}^{Cloud}\left(\frac{ciclos}{s}\right) \end{cases} \qquad MEC \begin{cases} n_{cores}^{MEC} \\ C_{buffer}^{MEC}(ms) \\ C_{clock}^{MEC}\left(\frac{ciclos}{s}\right) \end{cases}$$

$$RSU \begin{cases} n_{cores}^{RSU} \\ C_{buffer}^{RSU}(ms) \\ C_{clock}^{RSU}\left(\frac{ciclos}{s}\right) \end{cases} \qquad CAR\ (cada\ vehículo) \begin{cases} n_{cores}^{Car} \\ C_{buffer}^{Car}(ms) \\ C_{clock}^{Car}\left(\frac{ciclos}{s}\right) \end{cases}$$

El nodo de nube es excepcional debido a que consideramos que posee recursos infinitos. Por tanto, en términos prácticos su número de núcleos y su memoria son infinitos. Sin embargo, no procesa las tareas instantáneamente, porque su velocidad de reloj sí es finita. Consideramos que no se ve afectado por el ruido de procesamiento (ver sección 3.4) al tener recursos infinitos.

Un detalle que salta a la vista es que la memoria está definida en unidades de tiempo en lugar de en bytes. Esto se hace por comodidad, ya que el tamaño de la memoria en milisegundos y en bytes (o bits) está relacionado. Suponemos que el tiempo de procesamiento de un paquete es proporcional a su tamaño, de modo que el tiempo de procesamiento puede calcularse a partir del número de datos que contiene el paquete (en bits), el coste de procesamiento (esto es, el número de ciclos necesarios por cada bit del paquete) y la velocidad de reloj (en ciclos por segundo). El resultado es un tiempo de procesamiento directamente comparable con la cola que representa la memoria.

3.2.3. Aplicaciones

En el entorno de simulación también debe definirse un set de aplicaciones. Cada aplicación del set se caracteriza por requerir cada cierto tiempo el procesamiento de un conjunto de datos (paquete de datos) de un determinado tamaño (A_{in}). Como resultado del procesamiento de cada paquete se generará un paquete de salida de tamaño (A_{out}). Cada aplicación está además caracterizada por el periodo de generación de paquetes (A_T), el coste de procesamiento por bit (A_c), la latencia máxima permitida (A_d) y el peso de ponderación de la aplicación (A_p), el cual permite ajustar la importancia relativa de unas aplicaciones frente a otras. Cuando un vehículo desea procesar un paquete de datos asociado a una aplicación, envía una petición al agente y éste decide en que nodo se debe procesar ese paquete de datos. El vehículo entonces envía un paquete de datos de tamaño A_{in} a dicho nodo y, una vez procesado, el nodo envía la respuesta al vehículo mediante un paquete de salida de tamaño A_{out}. En media, transcurrido un tiempo A_T tras el envío de la petición anterior, el vehículo enviaría una nueva petición al agente para procesar un nuevo paquete de datos.

En definitiva, podemos por tanto definir cada aplicación como:

$$Aplicación\ a \begin{cases} A_c^a\left(\frac{ciclos}{bit}\right) \\ A_{in}^a(kbits) \\ A_{out}^a(kbits) \\ A_d^a(ms) \\ A_T^a(ms) \\ A_p^a \in \mathbb{R} \end{cases}$$

3.3. Funcionamiento del entorno de simulación

Vistos los elementos que conforman el entorno de simulación, es conveniente describir varios de los procesos que se realizan en el mismo. En esta subsección se presentan los cuatro procesos más significativos y que es fundamental comprender para entender el funcionamiento del simulador: el envío de datos, la generación de peticiones, las reservas de recursos de computación y el ruido de procesamiento.

3.3.1. Envío de datos

Para ilustrar el funcionamiento del entorno, se muestra el ejemplo de una petición en la que el agente decide que debe ser procesada en el servidor MEC. Es importante saber que consideramos retardos nulos para todas las comunicaciones salvo los envíos de datos de las aplicaciones. Un esquema del ejemplo propuesto se muestra en la Figura 3.

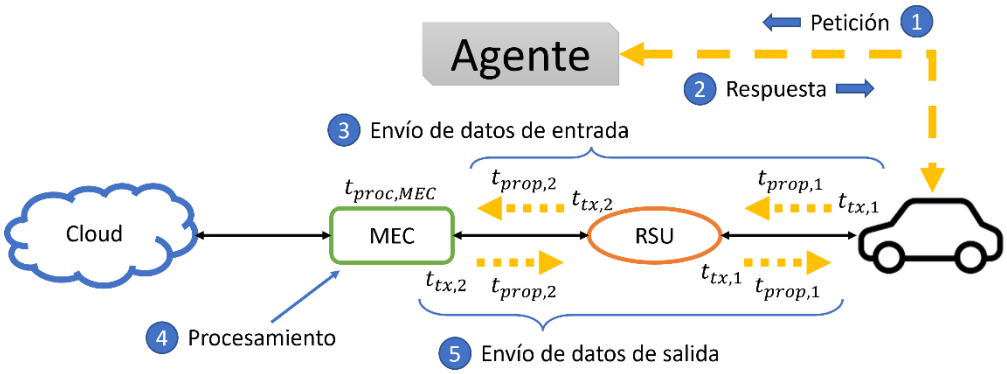

Figura 3. Esquema del ejemplo en el que el paquete de datos de una petición se procesa en el servidor MEC.

En primer lugar, la petición se genera en el vehículo debido a que una aplicación lo solicita. Esta petición se envía al agente, el cual responde sobre dónde deben ser procesados los datos (retardo nulo). Al recibir la respuesta los datos de entrada se envían a través de la red hasta el nodo correspondiente. Una vez procesados, se envían de vuelta hasta el vehículo.

Esta operación supone retardos de transmisión tanto de los datos de entrada como de salida en todos los enlaces, así como los retados de propagación por los enlaces. La ruta tomada por la red siempre es la más corta (o la primera más corta calculada) en número de saltos. También se tiene en cuenta el tiempo de procesamiento que incluye tanto el procesamiento en sí de los datos, como la espera en cola del núcleo de la CPU.

3.3.2. Generación de peticiones

Las peticiones son generadas por una serie de fuentes de tráfico. Cada fuente de tráfico se corresponde con una aplicación de un vehículo individual. Por lo tanto, si tenemos R vehículos por nodo vehículo, con un total de M nodos vehículo en la red y un set de aplicaciones que define L aplicaciones, tendremos un total de $R * M * L$ fuentes de tráfico independientes.

Cada fuente de tráfico genera por su cuenta peticiones iguales y correspondientes con los datos de la aplicación en cuestión. Si bien esto implica que hay un periodo medio entre peticiones de una misma fuente (A_T), en realidad se generan mediante una distribución exponencial de tal media.

3.3.3. Reservas de recursos de computación en los nodos y procesamiento

Si bien el entorno realiza una estimación del tiempo de procesamiento requerido para cada petición, esta estimación no tiene en cuenta el estado de las colas de los núcleos existentes en los nodos. La ocupación de éstas afecta considerablemente a los tiempos de procesamiento reales y por tanto es un factor importante en la simulación.

En el instante en el que se genera la petición y el agente toma una decisión, su respuesta no solo vuelve al vehículo para indicar el nodo de procesamiento, sino que también se informa a dicho nodo de procesamiento. El nodo en cuestión usará la estimación del retardo de envío de los datos y del tiempo de procesamiento para realizar una reserva en la cola de alguno de sus núcleos. Consideramos dos tipos de reservas para nuestros experimentos.

Por un lado, se pueden considerar las reservas con planificación. En este modo de funcionamiento el nodo reservará el primer hueco disponible en cualquiera de sus colas que se ajuste a los datos de la estimación. Podemos ver dos ejemplos ilustrando un servidor MEC de cuatro núcleos realizando esta operación en la Figura 4 y la Figura 5.

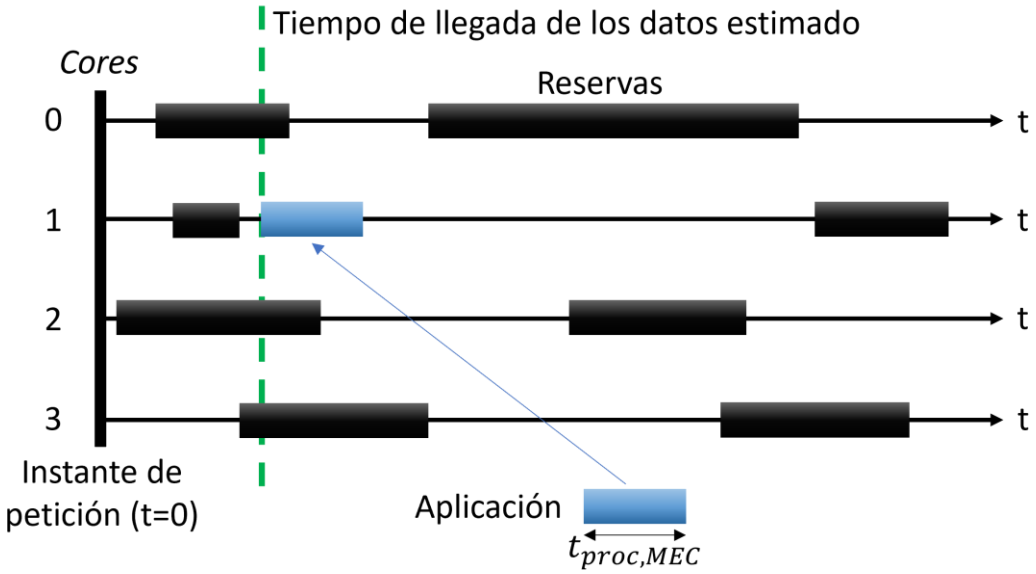

Figura 4. Ejemplo de reserva con planificación en un servidor MEC de cuatro núcleos.

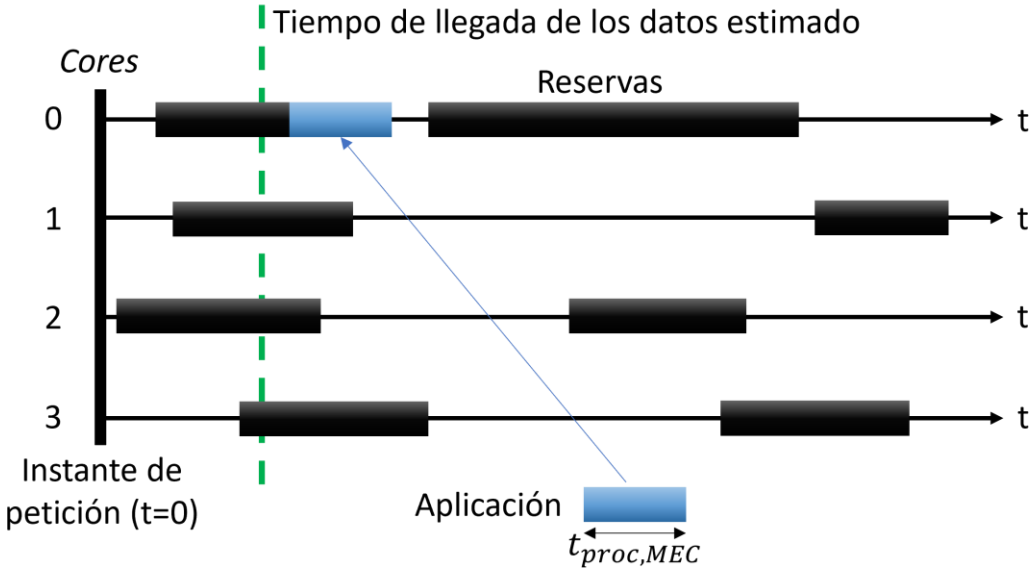

Figura 5. Ejemplo de reserva con planificación en un servidor MEC de cuatro núcleos.

En el ejemplo de la Figura 4 se observa como la reserva se puede realizar en el tiempo en el que se espera que lleguen los datos al nodo. Sin embargo, este no siempre es el caso y puede ocurrir que se deba reservar más tarde. Esto se observa en la Figura 5 y en este caso el tiempo de procesamiento real no se correspondería con el estimado debido a la espera en cola.

El otro modo de funcionamiento es el de reservas sin planificación. Consiste en que los nodos realicen las reservas igual que antes pero solamente pudiendo asignarlas al final de las colas. La Figura 6 muestra el mismo ejemplo anterior, pero sin planificación.

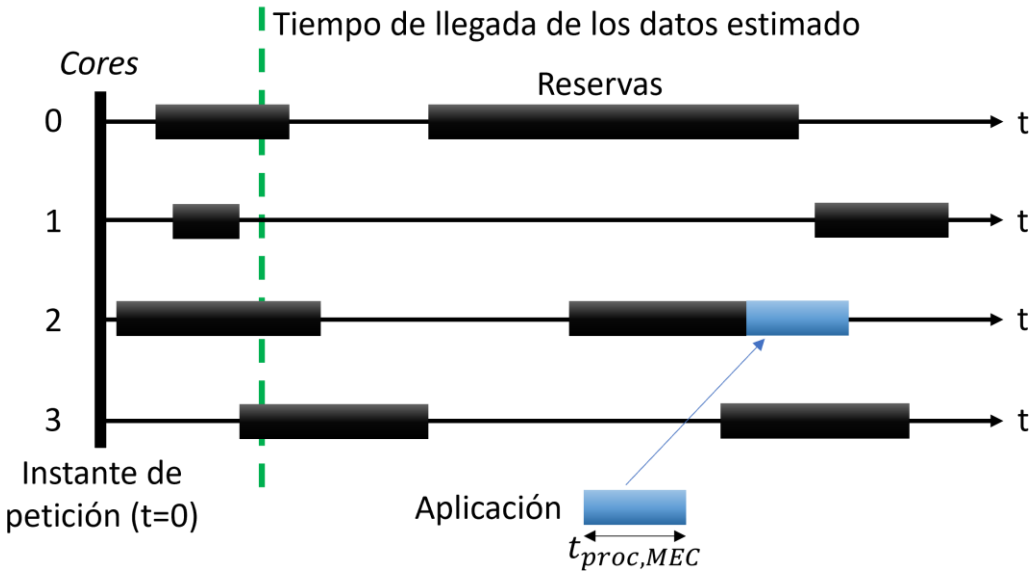

Figura 6. Ejemplo de reserva sin planificación en un servidor MEC de cuatro núcleos.

En ambos modos si al realizar la reserva no existe ningún hueco en las colas que permita insertar el tiempo de procesamiento estimado, se cancela la petición y el paquete de datos no será procesado. Recordemos que las colas están limitadas por la memoria definida en el nodo en milisegundos (C_{buffer}), que establece el límite máximo de la cola.

3.4. Ruido en el tiempo de procesamiento

Si bien consideramos que los enlaces son invariantes y por tanto las estimaciones de los tiempos de ida y vuelta de los datos son exactas, el simulador no asumirá que los tiempos de procesamiento en los núcleos son invariantes (salvo en el nodo de nube), sino que existe una cierta incertidumbre y el procesado de un paquete de datos puede por tanto durar un tiempo diferente del previsto. Para ello se tiene en cuenta un cierto ruido que provoca la variación del tiempo de procesamiento.

Partiendo de la estimación del tiempo de procesamiento ($t_{proc,est}$), se añade un ruido gaussiano definido mediante el coeficiente de variación (CV), que será igual para todas las aplicaciones. A mayores, la gaussiana se trunca para que solamente se generen valores dentro de un rango limitado ($\Delta t_{proc,min}, \Delta t_{proc,max}$) usando unos límites de truncamiento relativos al tiempo de procesamiento estimado (L_{min} y L_{max}). Dependiendo del experimento se pueden definir coeficientes de variación y límites de truncamiento distintos. El tiempo real de procesamiento será, por tanto:

$$t_{proc,real} = t_{proc,est} + \Delta t_{proc}$$

Donde,

$$t_{proc,est} = A_{in} * A_c * C_{clock}$$

$$\Delta t_{proc} = \begin{cases} \Delta t_{proc,min}, & \Delta x < \Delta t_{proc,min} \\ \Delta x, & \Delta t_{proc,min} \geq \Delta x \geq \Delta t_{proc,max} \\ \Delta t_{proc,max}, & \Delta x > \Delta t_{proc,max} \end{cases}$$

siendo,

$$\Delta x = N(\mu = 0, \sigma = CV * t_{proc,est})$$

$$\begin{cases} \Delta t_{proc,min} = t_{proc,est} * (-L_{min}/100) \\ \Delta t_{proc,max} = t_{proc,est} * L_{max}/100 \end{cases}$$

La Figura 7 muestra un escenario de ejemplo en el que se trunca la gaussiana al 25% del tiempo de procesamiento estimado en ambos sentidos. En este ejemplo la desviación del tiempo de procesamiento real es cómo mucho del 25% del tiempo de procesamiento estimado tanto hacia abajo como hacia arriba, y por tanto, L_{min} y L_{max} se fijan (ambos) a un valor de 25.

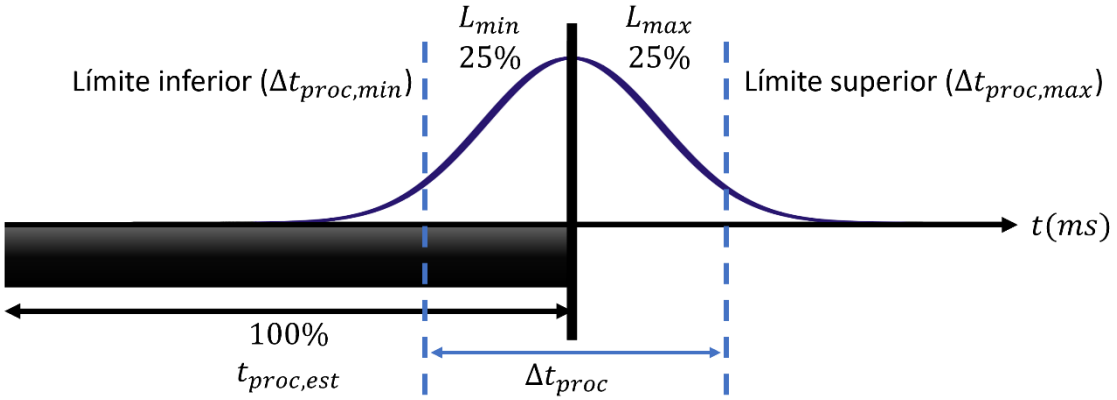

Figura 7. Ruido gaussiano truncado al 25% en ambos sentidos añadido al tiempo de procesamiento de una reserva.

Algo a tener en cuenta es que este ruido puede producir que una reserva empuje a las que la siguen en la cola. Esto solamente ocurre si el tiempo de procesamiento real es mayor que el estimado y si no hay suficiente margen entre las reservas. En caso de empujarse, las

reservas se posponen solamente lo necesario, y cualquier reserva que supere el límite de la cola (C_{buffer}) se cancela. Esto supone que se pueden no procesar peticiones que si lograron reservarse en su momento.

3.5. Agente

Los últimos detalles relevantes del entorno de simulación son los relacionados con la interacción con los agentes. En esta subsección se describen tanto la observación que se genera para los agentes junto con su espacio de acción, como el cálculo de la recompensa.

3.5.1. Observación y acción

Para poder responder a las peticiones, decidiendo dónde procesar un paquete de datos, el agente necesita que estas contengan cierta información sobre la aplicación correspondiente, así como información sobre el estado de la red y del vehículo del que proviene la petición. Como ya hemos dicho, el intercambio de esta información se considera instantáneo en el entorno, y por tanto el agente conocerá el estado de la red y del vehículo en el instante en el que recibe la petición. Dicho instante se corresponde también con el instante en el que la petición se genera.

Concretando más en la información que el agente recibe con cada petición, en primer lugar tenemos la ocupación de cada uno de los núcleos disponibles en la red (en los nodos MEC, RSU y en el vehículo correspondiente). Denotaremos por Q_x a la ocupación del núcleo x y por X al número total de núcleos. Dicho parámetro se representa en tanto por uno con un valor de 1 implicando una cola completamente vacía.

$$Q_x = \frac{tiempo\ libre\ en\ la\ cola\ del\ núcleo\ x}{C^x_{buffer}}$$

En segundo lugar, tenemos información relativa a la aplicación. Esto incluye el coste de procesamiento por bit (A_c), los datos de entrada (A_{in}) y de salida (A_{out}), y la latencia máxima (A_d). Todos estos valores están normalizados ($\hat{A}_c$, $\hat{A}_{in}$, $\hat{A}_{out}$ y $\hat{A}_d$) por el valor máximo de dicho parámetro en todo el set de aplicaciones definido. De esta forma se obtienen valores entre 0 y 1, donde un valor cercano a 1 implica una aplicación menos estricta (por ejemplo, menos datos de entrada o más latencia permitida) mientras que 0 es lo contrario.

A continuación, dado que el número total de nodos que pueden procesar el paquete es K, el agente también recibe una estimación sobre si la latencia máxima permitida es mayor que el retardo que se espera debido al envío de datos y al procesamiento en el nodo k sin tener en cuenta las colas (D_k). Se genera un valor entre 0 y 1 para cada nodo que puede procesar los datos, indicándose con un valor cercano a 1 que el retardo total estimado ($t^k_{T,est}$) es mucho menor que la latencia máxima (A_d), mientras que un valor cercano a 0 implica que no hay mucho margen. Si el cálculo produce un valor inferior a 0 (retardo total estimado mayor que latencia máxima), se trunca a 0. Cualquier valor de 0 implica que en principio la petición no podrá procesarse a tiempo. La expresión en forma de ecuación se muestra a continuación:

$$D_k = \begin{cases} 0, & t^k_{T,est} \geq A_d \\ 1 - \dfrac{t^k_{T,est}}{A_d}, & t^k_{T,est} < A_d \end{cases}$$

Por último, la observación también incluye tantos elementos como nodos vehículo haya en la red (V). Todos estos elementos toman el valor de 0 exceptuando el que sea el origen de la petición, que será 1.

El vector genérico que recibe el agente se muestra a continuación, siendo a la aplicación correspondiente, K el número total de nodos que pueden procesar los datos, X el número total de núcleos que pueden procesar los datos y P el número total de nodos vehículo en la red:

$$Observación: \left[Q_1, \dots, Q_X, \hat{A}_c^a, \hat{A}_{in}^a, \hat{A}_{out}^a, \hat{A}_d^a, D_1, \dots, D_K, V_1, \dots, V_P\right]$$

La respuesta del agente sencillamente indica cuál de los K nodos realizará el procesamiento (el núcleo lo elige el nodo en sí al reservar):

$$Acción \in [1, \dots, K]$$

3.5.2. Recompensa

Para poder entrenar al agente, el entorno también calcula una recompensa cada vez que se recibe una petición y toma una acción. Esta recompensa se envía al agente junto con la observación siguiente (a recibir la siguiente petición). La recompensa tiene en cuenta todos los paquetes de datos que se han ejecutado desde que se tomó la acción anterior hasta la ejecución de la acción actual (es decir, las reservas procesadas o canceladas que hubiera entre esos instantes de tiempo).

Aquellos paquetes de datos que se han procesado con éxito (respetando los requisitos de latencia) contribuyen con una recompensa nula o positiva (tal y como se explicará posteriormente), mientras que aquellos que no pudieron procesarse con éxito (por ejemplo, porque debido a la incertidumbre en los tiempos de procesamiento finalmente no pudieron cumplir con los requisitos de latencia) contribuyen con una recompensa negativa. Así pues, la recompensa que recibe el agente es la suma de todas esas recompensas asociadas a cada reserva procesada o cancelada desde el paso anterior.

Concretamente, para cada reserva, la recompensa asociada a la misma se calcula siguiendo una de las estrategias que se comentan a continuación:

- En primer lugar, tenemos la recompensa por penalización. Se considera que si un paquete de datos no se logra procesar por cualquier motivo la recompensa es de -1000. Si se procesa, pero no a tiempo, la recompensa es -100 menos la diferencia de tiempo (en milisegundos) entre el retardo total (t_T) y la latencia máxima (A_d). Por último, si se procesa a tiempo la recompensa es 0. Es importante no otorgar recompensas positivas por retardos menores para que no compensen las negativas de otras aplicaciones. Podemos ver esto representado en forma de ecuación a continuación:

$$Recompensa\ por\ penalización = \begin{cases} -1000, & no\ procesada \\ -100 - (t_T - A_d), & t_T > A_d \\ 0, & t_T \leq A_d \end{cases}$$

- El otro tipo es la recompensa binaria ponderada. Consiste sencillamente en otorgar una recompensa de $+1$ o -1 en función de si se procesó a tiempo o no el paquete de datos. El valor de $+1$ o -1 se multiplica por el peso de ponderación definido para cada aplicación (A_p) para así otorgar prioridad a ciertas aplicaciones sobre otras. La formulación matemática se muestra a continuación:

$$Recompensa\ binaria\ ponderada = \begin{cases} -A_p, & t_T > A_d \\ +A_p, & t_T \leq A_d \end{cases}$$

Con respecto a la petición que acaba de llegar al agente, si se realiza una reserva de un núcleo para procesar la aplicación más adelante, de momento no contribuye a la recompensa que recibe el agente (ya lo hará más adelante cuando se sepa si pudo ejecutarse

y respetando además los requisitos de latencia o no). Ahora bien, hay otras dos posibilidades que pueden darse y en esas la recompensa ya sí se tiene en cuenta en el instante actual. La primera es si la forma de tratar esa petición es enviar los datos a la nube, puesto que se puede estimar con exactitud si se procesará a tiempo o no al tener recursos infinitos y no estar afectada por el ruido de procesamiento. La segunda es cuando no se logra realizar una reserva (y la recompensa será negativa tal y como acabamos de ver).

4. Algoritmos implementados y métricas de rendimiento

En esta sección se tratan los detalles necesarios para la realización de los experimentos. Esto incluye información sobre las herramientas empleadas para programar y ejecutar el simulador, los algoritmos empleados y sus parámetros de configuración para implementar a los agentes de RL, las heurísticas implementadas, las métricas utilizadas y los parámetros de configuración del entorno.

4.1. Simulador

Para la programación del simulador se ha empleado la librería de OpenAI Gym [5], utilizando Python 3.9. Todos los experimentos fueron realizados en una máquina virtual Debian 11.

4.2. Algoritmos de aprendizaje por refuerzo (RL)

En nuestros experimentos un agente se encarga de las decisiones sobre dónde procesar los distintos paquetes de datos. Estos agentes son instancias de un algoritmo de aprendizaje por refuerzo (RL) implementados mediante la librería de Python, ChainerRL [6]. Concretamente se han utilizado cuatro algoritmos distintos: DDQN, SARSA, PAL y TRPO [4]. Los tres primeros (DDQN, SARSA y PAL) son algoritmos basados en el algoritmo DQN por lo que también mostramos su descripción.

Con el objetivo de realizar una comparación entre los algoritmos, todos se han implementado con la misma red neuronal (una capa oculta con 60 neuronas activadas mediante tangente hiperbólica), tal como se propone en [4]. Solamente se han variado los parámetros de la Tabla 1.

Nombre	Rango de valores	Descripción
Factor de descuento (γ)	[0, 1]	Factor que define cuanta importancia se les otorga a estados posteriores durante el entrenamiento.
Tipo de exploración	Constante / Linealmente decadente	Define si los agentes realizan acciones exploratorias con una probabilidad uniforme o decadente durante el entrenamiento.
Probabilidad de exploración (ε)	[0, 100] %	Probabilidad de que un agente realice una acción aleatoria en lugar de seguir su política actual durante el entrenamiento.
Pasos de decadencia de la probabilidad de exploración	[1, ∞)	Número de pasos de tiempo para pasar desde una probabilidad de exploración inicial (ε_0) hasta una final (ε_f) de forma lineal.

Tabla 1. Parámetros variados de los algoritmos de aprendizaje por refuerzo.

4.2.1. DQN (Deep-Q-Network)

DQN [7] es un algoritmo de aprendizaje por refuerzo profundo cuya idea original proviene del denominado *Q-Learning*. El *Q-learning* se basa en la función de valor de una acción (función Q). Esta función es similar a la función de valor de un estado, pero en lugar de indicar "lo bueno" que es estar en ese estado, indica "lo bueno" que es estar en un estado concreto y tomar una acción determinada (en realidad, indica el retorno esperado cuando se está en un estado concreto, se toma una acción determinada, y se sigue, a partir de ese momento, la política que se esté considerando). Al estar hablando de aprendizaje profundo, esta función de valor de la acción se aproxima mediante una red neuronal para generalizar distintos casos.

4.2.2. DDQN (Double Deep-Q-Network)

El algoritmo DDQN [8] es una variante propuesta para resolver varios problemas detectados con el algoritmo DQN básico. Concretamente, este algoritmo reduce las frecuentes sobreestimaciones de valores de acción.

4.2.3. SARSA (State-Action-Reward-State-Action)

SARSA [3] es un algoritmo similar al DQN con la principal diferencia de que en lugar de aprender la función Q de la política óptima (*off-policy*), aprende una política cercana a la óptima que viene definida por la exploración (*on-policy*). Una ventaja bien conocida de SARSA es que tiene en cuenta malos resultados debidos a acciones exploratorias, mientras que DQN no, lo que permite evitar acciones arriesgadas y será de interés observar cómo afecta esto a los experimentos.

Dicho esto, es importante mencionar que el algoritmo SARSA implementado en ChainerRL es *off-policy*. En concreto, es una variante de *Expected* SARSA [3], un algoritmo definido como *off-policy* debido a que en su caso la *target policy* y la *behaviour policy* no son iguales. El punto clave es que sigue teniendo en cuenta la exploración durante la evaluación de la política, simplemente que la política evaluada no es la misma que la que se usa a la hora de tomar las acciones.

4.2.4. PAL (Persistent Advantage Learning)

En el algoritmo PAL [9], comparado con DQN, aparece un parámetro a mayores denominado peso de las ventajas persistentes (α). Este algoritmo propone una modificación al operador de Bellman que intenta mitigar inconsistencias entre la decisión sobre cuál es la acción óptima en un estado. Esto significa que incrementa la diferencia entre los distintos valores de la función Q en un estado para acciones distintas, diferenciando la óptima de las demás.

4.2.5. TRPO (Trust Region Policy Optimization)

TRPO [10] es un método de gradiente de la política (*policy gradient*). Los algoritmos de gradiente de la política siguen una estrategia diferente de los anteriores. En lugar de aprender las funciones de valor (para luego tomar acciones basándose en las mismas), intentan aprender directamente una política parametrizada para que pueda seleccionar las acciones sin consultar una función de valor. Recordemos que una política es la probabilidad de elegir una acción determinada cuando se está en un determinado estado. Pues bien, estos algoritmos intentan proporcionar los valores de esas probabilidades directamente. Por estos motivos, este algoritmo será el único cuya configuración varíe considerablemente del DQN. La principal razón de esto es que en lugar de tomar una función Q como argumento de entrada que define la red neuronal, se le debe pasar una política a optimizar.

La implementación de TRPO en ChainerRL también requiere una función de valor de estado. Esto se debe a que se hace uso del mecanismo GAE (*Generalized Advantage Estimation*) [11] para reducir la varianza de la estimación del gradiente, si bien aumenta un poco el sesgo.

4.3. Heurísticas

Con el objetivo de comparar el rendimiento de los algoritmos de aprendizaje por refuerzo, también se han implementado cuatro heurísticas: LP (*Local Processing*), CP (*Cloud Processing*), UDP (*Uniform Distribution Processing*) y MDP (*Max Distance Processing*). Todos ellos están programados para imitar a los agentes de RL y funcionar en las mismas condiciones.

4.3.1. LP (Local Processing)

Algoritmo que siempre elige procesar los paquetes de datos localmente en el vehículo que los genera.

4.3.2. CP (Cloud Processing)

Algoritmo que siempre elige procesar los paquetes de datos en el nodo de nube de la red.

4.3.3. UDP (Uniform Distribution Processing)

Algoritmo que elige aleatoriamente un nodo que procese el paquete de datos de entre todos los posibles de la jerarquía (nube, MEC, RSU y vehículo). La idea es que la carga se distribuya por todos los nodos equitativamente. Si hay varios nodos de un mismo tipo, solamente tiene en cuenta los que están conectados directamente por encima del vehículo cuya petición se está atendiendo, por lo que para cada petición considera un solo MEC y un solo RSU (no se elegirán nunca MECs o RSUs paralelos en la jerarquía).

4.3.4. MDP (Max Distance Processing)

Algoritmo que siempre elige el nodo más lejano posible en la jerarquía (nube, MEC, RSU y vehículo en ese orden) que pueda procesar el paquete de datos. El criterio para saber si un nodo puede o no procesar un paquete de datos se basa en comprobar si el retardo estimado de dicha acción es inferior a la latencia máxima. Este cálculo no tiene en cuenta el estado de las colas de los núcleos de cada nodo e ignora el retardo que esto puede suponer a mayores. Si hay varios nodos de un mismo tipo, solamente tiene en cuenta los que están conectados directamente por encima del vehículo cuya petición se está atendiendo, por lo que para cada petición considera un solo MEC y un solo RSU (no se elegirán nunca MECs o RSUs paralelos en la jerarquía). La idea de este algoritmo es priorizar el procesamiento en los nodos con más recursos (más alto en la jerarquía).

4.4. Métricas

En los experimentos se extraen métricas que usamos para medir el rendimiento obtenido. La Tabla 2 describe todas dichas métricas, de las cuales todas salvo las recompensas promedias son medidas durante la fase de testeo. En las simulaciones existen dos fases distinguidas: el entrenamiento y el testeo. El entrenamiento consiste en simular los agentes de RL en el escenario mientras actualizan sus políticas. El testeo es, al igual que el entrenamiento, una simulación del escenario, pero donde los agentes no cambian sus políticas y solamente toman las acciones que estas dictan.

Nombre	Rango de valores	Descripción
Recompensas promedias	(-∞, ∞)	Recompensa promediada usando una ventana de 10000 muestras en cada paso de entrenamiento (solo para el entrenamiento).
Tasa de éxito	[0, 100] %	Proporción de paquetes de datos procesados a tiempo.
Distribución de acción por aplicación	[0, 100] % para cada nodo en la red y el vehículo local	Proporción de paquetes de datos de una aplicación procesados en cada nodo (sin contar paquetes no procesados).
Retardo medio por aplicación	[0, ∞) milisegundos	Tiempo medio de procesamiento de los paquetes de datos de una aplicación (sin contar paquetes no procesados).
Recompensa media	(-∞, ∞)	Recompensa media al final de la simulación.

Tabla 2. Métricas utilizadas en los experimentos.

4.5. Parámetros de simulación

En los experimentos variamos diversos parámetros de la simulación para evaluar distintos escenarios. Dichos parámetros se describen en la Tabla 3.

Como hemos mencionado en el apartado anterior, en las simulaciones existen dos fases distinguidas: el entrenamiento y el testeo. Ambas fases son independientes, es decir, se reinicia el entorno para cada una, y comienzan con unos pasos de inicialización (mil por defecto) durante los cuales no se recopilan muestras para evitar que las métricas tengan en cuenta estados de la red no estacionarios.

Nombre	Rango de valores	Descripción
Topología de la red	Ver sección 3.2.1	Define la topología de la red y los recursos de los enlaces.
Nodos de la red	Ver sección 3.2.2	Define los recursos de los cuatro tipos de nodos de la red (nube, MEC, RSU y vehículo).
Set de aplicaciones	Ver sección 3.2.3	Define los datos de las aplicaciones que se ejecutarán en todos los vehículos.
Número de vehículos	$[1, \infty)$	Número total de vehículos en la red.
Coeficiente de variación (CV)	$[0, \infty)$	Coeficiente de variación de la distribución gaussiana empleada para variar el tiempo de procesamiento aleatoriamente.
Límite superior de la variación (L_{max})	$[0, \infty)$ %	Límite superior del truncamiento (en unidades relativas al tiempo de procesamiento estimado) de la distribución gaussiana empleada para variar el tiempo de procesamiento aleatoriamente.
Límite inferior de la variación (L_{min})	$[0, \infty)$ %	Límite inferior del truncamiento (en unidades relativas al tiempo de procesamiento estimado) de la distribución gaussiana empleada para variar el tiempo de procesamiento aleatoriamente.
Número de pasos de entrenamiento	$[1, \infty)$	Número de pasos de entrenamiento a realizar. Por defecto se usa el valor de 1000000.
Número de pasos de testeo	$[1, \infty)$	Número de pasos de testeo (agentes entrenados) a realizar. Por defecto se usa el valor de 100000.
Número de pasos de inicialización	$[0, \infty)$	Número de pasos de tiempo tanto en el entrenamiento como en el testeo para alcanzar un estado estacionario en la red y no tomar medidas imprecisas. Por defecto se usa el valor de 1000.

Tabla 3. Parámetros utilizados en las simulaciones.

5. Experimentos iniciales (redes con una sola RSU)

5.1. Planteamiento

En esta sección se muestran experimentos en dos redes bajo condiciones distintas. El objetivo es mostrar la viabilidad de los algoritmos de aprendizaje por refuerzo en el problema planteado, así como el rendimiento de las heurísticas que se usan como comparación.

5.1.1. Topologías

Las dos redes consideradas se muestran en la Figura 8 y en la Figura 9.

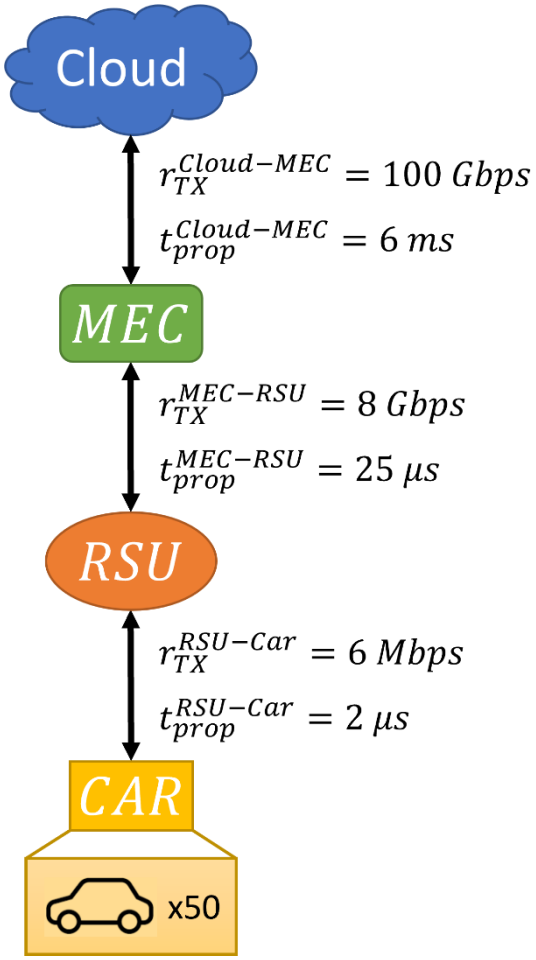

Figura 8. Red con retardos y recursos bajos (RRB).

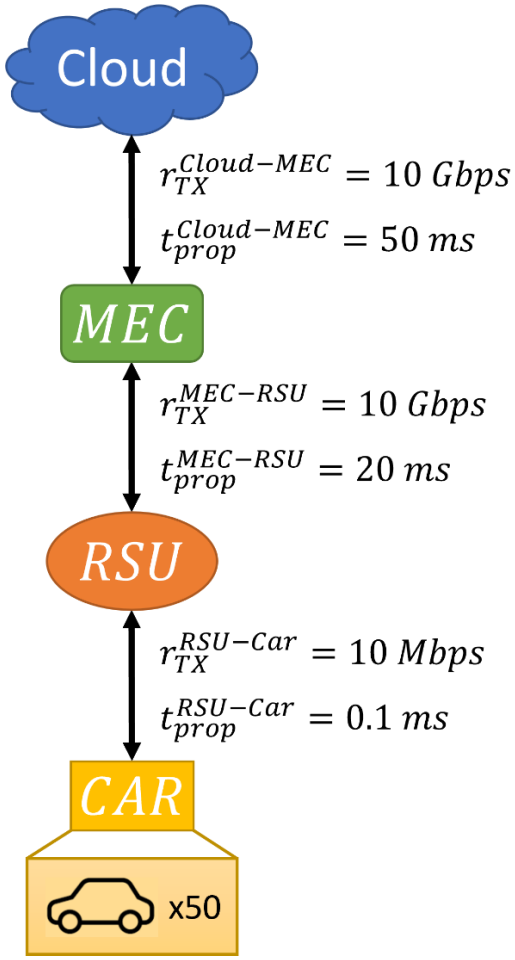

Figura 9. Red con retardos y recursos altos (RRA).

Denominamos a la red de la Figura 8 como red con retardos y recursos bajos (RRB), y la de la Figura 9 como red con retardos y recursos altos (RRA). Para sus definiciones se han usado parámetros tomados de [12], [13], [14], [15], [16] y [17], y, [13], [14], [18] y [19], respectivamente.

La diferencia entre las redes mostradas está en las tasas de transmisión y tiempo de propagación de sus enlaces. Viendo ambas redes queda claro que, para el caso RRB, los enlaces de la red son mucho mejores para enviar datos sin sufrir demasiado por los tiempos de ida y vuelta de los datos. Esto se debe principalmente a que los tiempos de propagación son mucho menores.

Sin embargo, a esto hay que añadir la diferencia entre los recursos de procesamiento de los nodos, mostradas en la Tabla 4. Para los nodos de la red RRB se han tomado parámetros de [20], [21], [22] y [23], mientras que para la red RRA se han tomado de [20] y [21].

		n_{Cores}	C_{Clock}	C_{buffer}
Red con retardos y recursos bajos (RRB)	Cloud	∞	4 GHz	∞
	MEC	8	3 GHz	200 ms
	RSU	4	3 GHz	200 ms
	CAR	1	333 MHz	20 ms
Red con retardos y recursos altos (RRA)	Cloud	∞	3.6 GHz	∞
	MEC	16	3.6 GHz	200 ms
	RSU	8	2.4 GHz	200 ms
	CAR	2	1.2 GHz	200 ms

Tabla 4. Características de los nodos de cómputo de las redes consideradas en los experimentos iniciales.

Se observa como los recursos de la red RRA son mayores de forma global, tal como da a entender su nombre. Muy relacionada con los recursos de cómputo en la red está la carga que producen las aplicaciones. Pues bien, para finalizar la caracterización de los escenarios hay que mirar los sets de aplicaciones que se ejecutan en cada red.

5.1.2. Sets de aplicaciones

La Tabla 5 muestra las características de los sets de aplicaciones usados en ambas redes. Sus definiciones están basadas en las aplicaciones propuestas en [24] para la red RRB, y en [25], [26] y [27] para la RRA. Observando las diferencias vemos que para la red RRB el coste computacional es menor de forma generalizada.

		$A_c\left(\frac{ciclos}{bit}\right)$	$A_{in}(kb)$	$A_{out}(kb)$	$A_d(ms)$	$A_T(ms)$	A_p
Red con retardos y recursos bajos (RRB)	App 1	1	360	100	120	10	1
	App 2	1	3600	100	10	1500	2
	App 3	1	4700	400	100	100	2
	App 4	1	100	100	200	1000	1
	App 5	1	900	64	150	100	1
	App 6	1	500	500	20	100	1
Red con retardos y recursos altos (RRA)	App 1	1	700	200	1	100	1
	App 2	400	10	5	4	100	1
	App 3	1000	900	25	500	100	1
	App 4	3000	100000	25000	200000	100	1
	App 5	200	650	500	200	100	1
	App 6	500	40	1	30	100	1

Tabla 5. Sets de aplicaciones para las redes consideradas en los experimentos iniciales.

5.1.3. Escenarios

Para medir la efectividad de los algoritmos de aprendizaje por refuerzo en situaciones variables, se han considerado múltiples escenarios en ambas redes planteadas. Por ello se han planteado tres casos de ruido de procesamiento distintos (Tabla 6) cruzados con las reservas con y sin planificación en las colas en los nodos (ver Sección 3.3.3). Además, se han realizado pruebas con ambos tipos de recompensa de los agentes, por penalización y binaria ponderada (ver Sección 3.5.2).

	Límite inferior (L_{min})	**Límite superior (L_{max})**	**Coeficiente de variación (CV)**
Sin ruido	-	-	0
Balanceado	50%	50%	1
Truncado	0%	100%	1

Tabla 6. Características del ruido añadido al procesamiento de los paquetes de datos en tres escenarios distintos.

5.2. Resultados

Todas las simulaciones constaban de una fase de entrenamiento de un millón de pasos de tiempo y una fase de testeo de cien mil pasos de tiempo, ambas con mil pasos de tiempo de inicialización a mayores. Ambas fases son independientes y el entorno se reiniciaba para cada una. Todos los agentes realizaban estas fases de forma independiente. En la Tabla 7, Tabla 8, Tabla 9 y Tabla 10 se muestran las tasas de éxito y recompensas medias obtenidas al final del testeo para cada agente en todos los escenarios. Para cada escenario se resaltan los tres algoritmos que obtienen los mejores resultados.

5.2.1. Análisis global

Fijándose primero en el caso de la red RRB (Tabla 7 y Tabla 8) se observa cómo los resultados (en términos de tasa de éxito del procesamiento de los paquetes de las aplicaciones) son aproximadamente iguales entre los algoritmos heurísticos y los de aprendizaje por refuerzo. La ventaja está ligeramente a favor de los algoritmos heurísticos, destacando sobre todo el de máxima distancia (MDP) y el procesamiento en local (LP).

Las tasas de éxito muestran que al procesar todo localmente se obtienen resultados muy buenos. La razón es que cuatro de las seis aplicaciones usadas en la red RRB deben ser procesadas localmente, ya que los retardos para enviar y recibir los datos son demasiado grandes. En cuanto a las aplicaciones que sí pueden delegarse con éxito (la 1 y la 4), es importante ejecutar dicha delegación, ya que supone una carga menor sobre los recursos de los vehículos. Al ser las aplicaciones 2 y 6 muy sensibles a la latencia, si las colas de los vehículos se llenan demasiado, la tasa de éxito disminuye. Aun así, se observa como la diferencia en tasa de éxito entre MDP y LP es pequeña, ya que una de las aplicaciones más afectadas (la 2) genera pocas peticiones comparada con las otras. Con esto se puede concluir que en la red RRB, dado el set de aplicaciones definido la política óptima es prácticamente el procesamiento en local, y esperamos ver políticas similares en los agentes de RL.

Por otra parte, en el caso de la red RRA (Tabla 9 y Tabla 10), los algoritmos de aprendizaje por refuerzo superan a las heurísticas. Este escenario, a diferencia del anterior, requiere una política más compleja donde distintas aplicaciones necesitan ser enviadas a diferentes nodos. Esto ocurre por los considerablemente mayores tiempos de procesamiento, que ahora se equiparan con los retardos de ida y vuelta. Es necesario encontrar el balance entre cantidad de recursos de computación y retardos de transmisión y propagación, ya que las aplicaciones mezclan cantidades de datos y latencias de forma que solo pueden ser procesadas correctamente en ciertos nodos de la red. A esto se añade que, en función de la carga de las colas, ciertos nodos pueden dejar de ser una opción viable. Esto pone de manifiesto que los algoritmos de aprendizaje por refuerzo son capaces de adaptarse mejor que las heurísticas a distintos entornos optimizando las prestaciones.

	Tasas de éxito en la red RRB											
	Recompensa por penalización						Recompensa binaria ponderada					
	Con planificación			Sin planificación			Con planificación			Sin planificación		
	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado
MDP	98.85%	98.69%	97.52%	98.8%	98.73%	97.59%	98.84%	98.65%	97.58%	98.82%	98.7%	97.5%
UDP	82.39%	82.32%	81.86%	59.86%	60.08%	59.23%	82.28%	82.35%	82.09%	61.7%	61.04%	60.45%
CP	76.7%	76.84%	76.8%	76.48%	76.62%	76.72%	76.83%	76.6%	76.57%	76.42%	76.65%	76.53%
LP	98.78%	98.26%	97.1%	98.77%	98.31%	97.17%	98.7%	98.28%	97.22%	98.75%	98.31%	97.16%
DDQN	96.43%	87.21%	88.59%	90.62%	96.65%	97.1%	85.6%	98.64%	82.33%	83.99%	96.46%	96.9%
SARSA	83.58%	96.69%	95.27%	93.3%	71.87%	44.84%	83.79%	83.84%	97.43%	83.58%	85.36%	97.45%
PAL	98.34%	97.1%	94.69%	83.88%	83.78%	95.53%	98.88%	98.35%	97.52%	96.4%	98.25%	95.38%
TRPO	91.34%	88.79%	87.96%	52.43%	51.19%	57.45%	92.33%	95.82%	92.47%	94.27%	94.27%	93.01%

Tabla 7. Tasas de éxito para procesar los paquetes de datos a tiempo en los escenarios planteados en la red con retardos y recursos bajos (RRB) con los tres mejores resultados marcados.

	Recompensas medias en la red RRB											
	Recompensa por penalización						Recompensa binaria ponderada					
	Con planificación			Sin planificación			Con planificación			Sin planificación		
	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado
MDP	-7.25	-10.4	-20.52	-7.46	-10.27	-19.67	1.03	1.03	0.99	1.04	1.03	0.98
UDP	-78.94	-80.33	-83.48	-106.92	-108.08	-111.33	0.6	0.6	0.59	0.19	0.18	0.16
CP	-99.04	-97.68	-97.91	-99.37	-99.03	-98.68	0.46	0.45	0.45	0.45	0.45	0.45
LP	-8.11	-14.69	-24.26	-8.14	-14.18	-23.91	1.03	1.02	0.98	1.03	1.02	0.97
DDQN	-11.61	-32.33	-40.63	-19.92	-12.01	-24.47	0.77	1.03	0.68	0.74	0.99	0.97
SARSA	-35.54	-13.48	-23.11	-16.65	-44.75	-98.77	0.74	0.74	0.98	0.73	0.76	0.98
PAL	-8.79	-13.19	-24.7	-36.01	-29.06	-22.16	1.04	1.02	0.98	0.99	1.02	0.94
TRPO	-20.77	-28.4	-35.22	-69.72	-71.72	-76.14	0.9	0.97	0.88	0.94	0.94	0.89

Tabla 8. Recompensas medias en los escenarios planteados en la red con retardos y recursos bajos (RRB) con los tres mejores resultados marcados.

	Tasas de éxito en la red RRA											
	Recompensa por penalización						Recompensa binaria ponderada					
	Con planificación			Sin planificación			Con planificación			Sin planificación		
	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado
MDP	49.8%	53.1%	46.15%	49.83%	50%	46.09%	49.85%	53.36%	46.3%	49.48%	50.01%	46.1%
UDP	30.97%	29.03%	22.29%	30.52%	24.57%	22.17%	30.4%	28.71%	22.21%	30.7%	25.14%	22.1%
CP	33.66%	33.44%	33.24%	33.19%	33.45%	33.46%	33.57%	33.51%	33.51%	33.14%	33.51%	33.37%
LP	44.18%	39.97%	29.76%	45.22%	39.05%	29.41%	44.96%	39.35%	29.73%	44.62%	39.07%	29.72%
DDQN	50.3%	33.44%	62.85%	33.25%	72.2%	33.09%	77.69%	54.79%	68.19%	56.79%	72.35%	43.48%
SARSA	78.27%	43.07%	58.87%	50.65%	49.7%	48.37%	78.39%	48.4%	52.25%	77.88%	59.84%	57.21%
PAL	58.52%	73.06%	62.67%	49.81%	72.19%	65.21%	86.55%	79.15%	69.51%	77.5%	54.38%	66.34%
TRPO	62.19%	60%	46.07%	62.86%	58.92%	47.39%	79.37%	67.53%	53.45%	70.06%	63.23%	43.09%

Tabla 9. Tasas de éxito para procesar los paquetes de datos a tiempo en los escenarios planteados en la red con retardos y recursos altos (RRA) con los tres mejores resultados marcados.

	Recompensas medias en la red RRA											
	Recompensa por penalización						Recompensa binaria ponderada					
	Con planificación			Sin planificación			Con planificación			Sin planificación		
	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado	Sin ruido	Balanceado	Truncado
MDP	-233.25	-219.96	-279.99	-235.95	-254.47	-283.51	0	0.07	-0.07	-0.01	0	-0.08
UDP	-338.9	-339.54	-379.95	-343.04	-371.1	-383.82	-0.39	-0.43	-0.56	-0.39	-0.5	-0.56
CP	-162.29	-162.92	-163.39	-163.77	-163.12	-162.93	-0.33	-0.33	-0.33	-0.34	-0.33	-0.33
LP	-376.88	-385.55	-421.22	-371.81	-387.97	-422.91	-0.1	-0.21	-0.41	-0.11	-0.22	-0.41
DDQN	-154.94	-214.37	-90.63	-214.59	-54.06	-299.59	0.55	0.1	0.36	0.14	0.45	-0.13
SARSA	-40.15	-162.08	-94.05	-101.77	-152.05	-113.46	0.57	-0.03	0.05	0.56	0.2	0.14
PAL	-58.83	-52.39	-89.69	-235.72	-54.03	-64.01	0.73	0.58	0.39	0.55	0.09	0.33
TRPO	-67.2	-64.54	-151.6	-72.19	-83.12	-145.98	0.59	0.35	0.07	0.4	0.26	-0.14

Tabla 10. Recompensas medias en los escenarios planteados en la red con retardos y recursos altos (RRA) con los tres mejores resultados marcados.

Es interesante observar cómo los agentes de RL son capaces de encontrar en muchos casos políticas con prestaciones similares a las de las heurísticas. Si miramos la distribución de acción de algunos de los casos de la red RRB, podemos confirmar que las mejores políticas que encuentran son muy similares a la que sigue el mejor algoritmo heurístico (MDP). La Figura 10 y la Tabla 11 muestran esto para el escenario de la recompensa binaria ponderada sin planificación y con ruido truncado (un caso en el que todos los agentes lograron tasas de éxito de más del 90%). Para la Figura 10 se debe recordar que como los agentes de RL realizan exploración durante el entrenamiento, y por tanto en ocasiones toman acciones al azar (en lugar de seleccionar la que consideran mejor), esto empeora su recompensa promedia durante dicho entrenamiento.

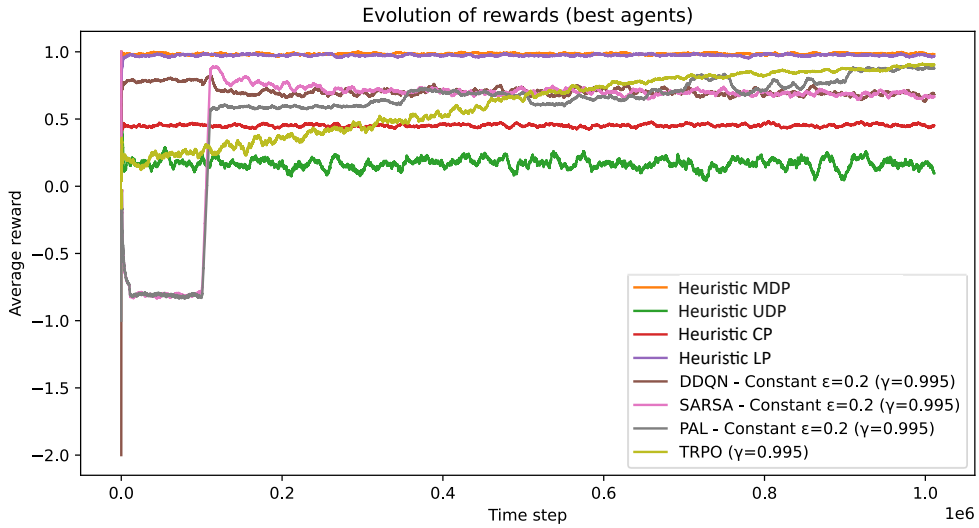

Figura 10. Evolución de recompensas promedias (binarias ponderadas) de los algoritmos durante el entrenamiento en la red con retardos y recursos bajos (RRB) sin planificación y con ruido truncado.

		Distribución de acción (%)			
		Nube	**MEC**	**RSU**	**Vehículo**
App 1	**MDP**	100	0	0	0
	DDQN	100	0	0	0
	SARSA	100	0	0	0
	PAL	100	0	0	0
	TRPO	99.87	0.01	0.04	0.08
App 2	**MDP**	0	0	0	100
	DDQN	0	0	2.93	97.07
	SARSA	0	0	2.42	97.58
	PAL	4.09	0	0	95.91
	TRPO	0	0	0	100
App 3	**MDP**	0	0	0	100
	DDQN	0	0	4.81	95.19
	SARSA	1.92	0	2.4	95.67
	PAL	5.24	0.14	0	94.62
	TRPO	0	0.07	0.03	99.9
App 4	**MDP**	100	0	0	0
	DDQN	100	0	0	0
	SARSA	100	0	0	0
	PAL	100	0	0	0
	TRPO	100	0	0	0
App 5	**MDP**	0	0	0	100
	DDQN	0	0	7.1	92.9
	SARSA	0	0	0	100
	PAL	0	17.14	0	82.86
	TRPO	0.21	16.2	15.55	68.04
App 6	**MDP**	0	0	0	100
	DDQN	0	0	1.36	98.64
	SARSA	0	0	0	100
	PAL	0	13.46	0	86.54
	TRPO	0.08	14.66	12.7	72.56

Tabla 11. Comparación entre distribuciones de acción de los algoritmos en la red con retardos y recursos bajos (RRB) sin planificación y con ruido truncado cuando se usa la recompensa binaria ponderada.

Como comentábamos, en la Tabla 11 se observa claramente como las políticas que siguen los agentes RL son muy similares a lo que dicta el algoritmo MDP (la mejor de las heurísticas). En este caso la política óptima es predecible y el MDP la implementa. Es especialmente importante observar que las aplicaciones 1 y 4, las únicas que pueden (y por tanto deben) ser procesadas fuera de los vehículos locales, son también las aplicaciones que los agentes de RL aprenden a delegar. Debido a que los recursos de los vehículos son limitados esto es fundamental para incrementar el éxito, y los algoritmos de aprendizaje por refuerzo demuestran aprenderlo con confianza.

Por otro lado, como ya hemos apuntado, en la red RRA (Tabla 9 y Tabla 10) los resultados de los agentes de RL en términos de tasa de procesamiento exitoso de los paquetes de las

aplicaciones se vuelven considerablemente mejores que los de las heurísticas. Para esta red la política óptima no es tan obvia, y los algoritmos heurísticos no son lo suficientemente avanzados como para adaptarse. En comparación, los algoritmos de RL sí se adaptan muy bien consiguiendo mejorar considerablemente el rendimiento. En ambas redes destaca el algoritmo PAL por su rendimiento superior a los demás.

5.2.2. Consistencia entre políticas de agentes de RL de entrenamientos distintos

Un detalle interesante es que los agentes de RL tienden a encontrar políticas distintas cuando ciertos parámetros cambian. Un claro ejemplo de esto se puede ver en la Tabla 7 y la Tabla 9, ya que los resultados de los algoritmos de RL a veces son dispares. Como es de esperar, las heurísticas generalmente obtienen tasas de éxito menores al pasar de un escenario sin ruido a uno con ruido (fundamentalmente el truncado), pero por el contrario esto no siempre ocurre con RL. Por ejemplo, el algoritmo DDQN consigue una tasa de éxito de más del 97% cuando se utiliza la recompensa por penalización, sin planificación y con ruido truncado en la red RRB (Tabla 7). Este valor es superior al 90% del mismo escenario sin ruido, y sin embargo no se observa la misma tendencia cuando el escenario sí tiene planificación. Se puede comprobar que los algoritmos RL son generalmente consistentes en el entrenamiento si el escenario es exactamente el mismo (Figura 11).

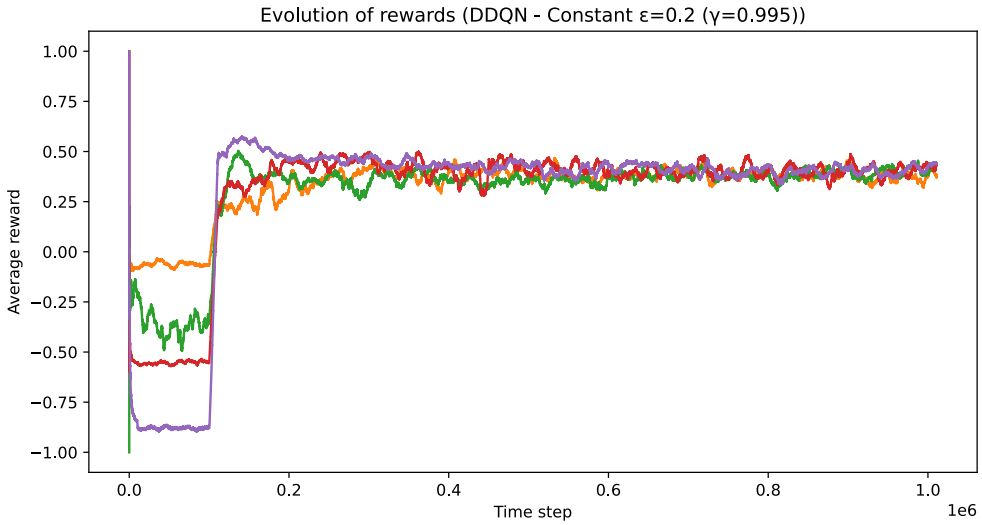

Figura 11. Evolución de recompensas promedias (binarias ponderadas) durante el entrenamiento para cuatro réplicas independientes del algoritmo DDQN en la red con retardos y recursos altos (RRA) con planificación y sin ruido.

La Figura 11 muestra en un escenario concreto como cuatro réplicas iguales del algoritmo DDQN convergen hacía la misma recompensa en el entrenamiento. Sin embargo, eso no implica que sus políticas sean iguales. La Tabla 12 y la Tabla 13 muestran de la fase de testeo las tasas de éxito y recompensas medias, y la distribución de acción de estos algoritmos, respectivamente.

	DDQN 1	DDQN 2	DDQN 3	DDQN 4
Tasa de éxito (%)	82.17	84.8	81.56	67.96
Recompensa media	0.64	0.7	0.63	0.36

Tabla 12. Tasas de éxito y recompensas medias para réplicas del algoritmo DDQN en la red con retardos y recursos altos (RRA) con planificación y sin ruido, con la recompensa binaria ponderada.

		Distribución de acción (%)			
		Nube	**MEC**	**RSU**	**Vehículo**
App 1	**DDQN 1**	0	13.33	45.1	41.57
	DDQN 2	0	30.6	0.02	69.38
	DDQN 3	0	0	18.28	81.72
	DDQN 4	0	92.05	0.12	7.83
App 2	**DDQN 1**	0	14.79	84.98	0.23
	DDQN 2	0	25.37	70.87	3.76
	DDQN 3	0	0	37.32	62.68
	DDQN 4	0	96.42	2.73	0.85
App 3	**DDQN 1**	100	0	0	0
	DDQN 2	100	0	0	0
	DDQN 3	100	0	0	0
	DDQN 4	100	0	0	0
App 4	**DDQN 1**	100	0	0	0
	DDQN 2	100	0	0	0
	DDQN 3	100	0	0	0
	DDQN 4	100	0	0	0
App 5	**DDQN 1**	0	12.47	2.71	84.82
	DDQN 2	0	17.14	0	82.86
	DDQN 3	0	0	16.02	83.98
	DDQN 4	0	42.54	0.04	57.42
App 6	**DDQN 1**	0	27.27	22.18	50.55
	DDQN 2	0	10.23	0	89.77
	DDQN 3	0	0	57.52	42.48
	DDQN 4	0	2.91	43.7	53.39

Tabla 13. Comparación entre distribuciones de acción de las réplicas del algoritmo DDQN en la red con retardos y recursos altos (RRA) con planificación y sin ruido, con la recompensa binaria ponderada.

En la Tabla 12 y la Tabla 13 se observan dos ideas importantes. En primer lugar, distintas políticas obtienen una recompensa promedia aproximadamente igual. En segundo lugar, la misma recompensa promedia en el entrenamiento no siempre se traduce a los mismos resultados en las métricas, ya que la última réplica (DDQN 4) tiene peores resultados que las demás pese a que en la Figura 11 parece alcanzar el mismo valor. Estos resultados parecen sugerir la necesidad de entrenar varios agentes y realizar una etapa de validación para seleccionar finalmente a uno de dichos agentes.

5.2.3. Entrenamientos prolongados

Pese a que no siempre está garantizado que un agente de RL encuentre la mejor política que puede alcanzar, se perciben algunos resultados cercanos al óptimo en la red RRB (recompensa nula para la recompensa por penalización) en la Tabla 8, así como resultados iguales o mejores a las heurísticas en los demás casos. Esto lleva a preguntarse si con entrenamientos más largos los algoritmos de RL siempre encontraran la misma política. Se puede ver en la Figura 12 una simulación donde el entrenamiento se extendió a dos millones de pasos de tiempo.

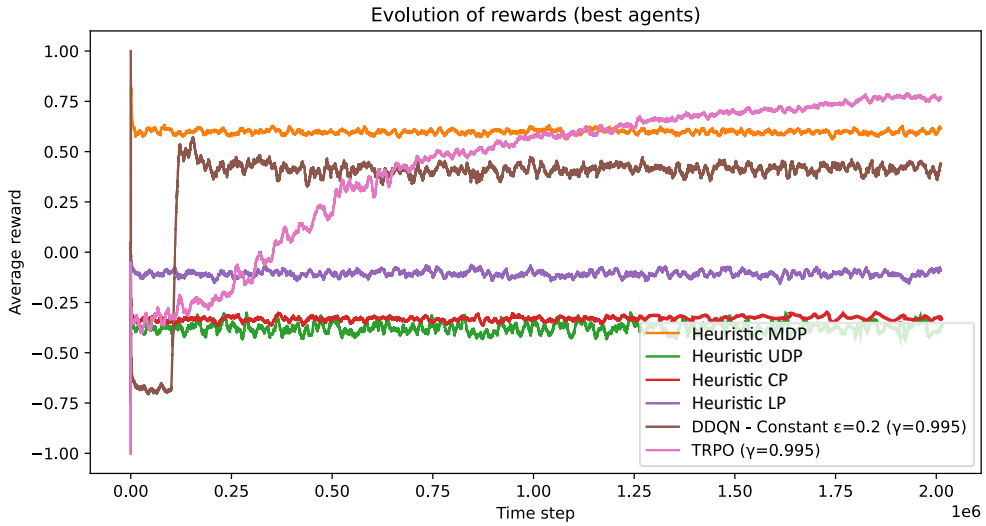

Figura 12. Evolución de las recompensas promediadas (binarias ponderadas) en un entrenamiento largo de dos millones de pasos de tiempo en la red con retardos y recursos altos (RRA) con planificación y sin ruido.

El resultado es que los algoritmos basados en DQN, como DDQN en este caso, no mejoran sus políticas con un entrenamiento más largo. Sin embargo, esto es completamente distinto para el algoritmo TRPO (basado en descenso del gradiente). Sus resultados acaban por superar considerablemente la de cualquier otra alternativa. Para el experimento realizado en la Figura 12, la tasa de éxito del agente TRPO es del 88%. Este experimento concuerda con el caso de la recompensa binaria ponderada con planificación y sin ruido de la Tabla 9, y es comparable con el resultado del agente PAL (86.55%).

5.2.4. Exploración linealmente decadente

Una consideración adicional es que todos los agentes fueron configurados con una exploración aleatoria (ε) constante del 20% en cada paso de tiempo, pero se podría preguntar si una exploración linealmente decadente sería más adecuada. Para ello, la Figura 13 y la Figura 14 muestran los resultados de agentes de RL con ambos tipos de exploración en el mismo escenario.

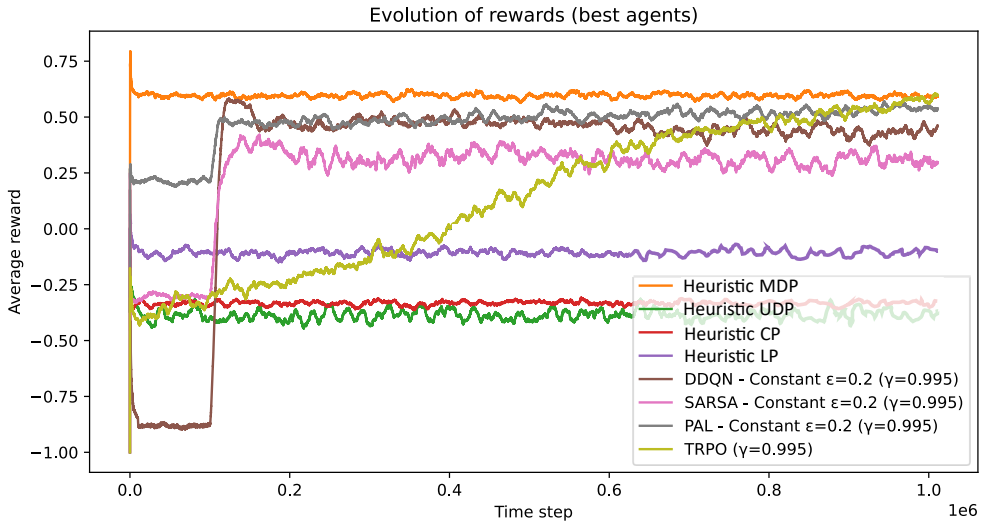

Figura 13. Evolución de las recompensas promediadas (binarias ponderadas) de los agentes con exploración constante en la red con retardos y recursos altos (RRA) con planificación y sin ruido.

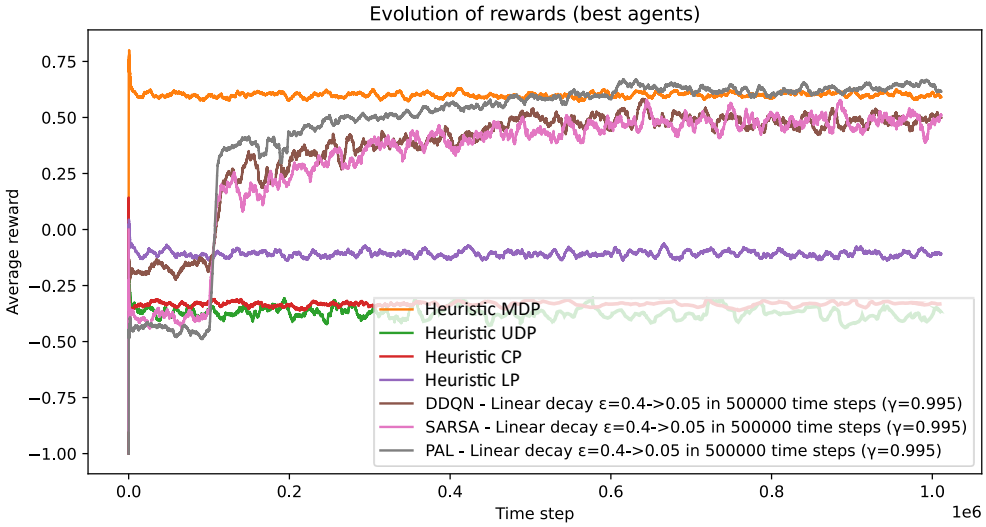

Figura 14. Evolución de las recompensas promediadas (binarias ponderadas) de los agentes con exploración linealmente decadente en la red con retardos y recursos altos (RRA) con planificación y sin ruido.

Se puede observar como en el escenario del experimento (red RRA con planificación y sin ruido), en el cual los agentes de RL obtuvieron los mejores resultados, sus recompensas mejoran ligeramente al cambiar el tipo de exploración. La exploración linealmente decadente ofrece mayor estabilidad en la evolución, pero hay que tener en cuenta que, al reducir la probabilidad de exploración, las recompensas promedias mejoran. Es por ello que, si miramos las tasas de éxito y recompensas medias de estos experimentos, los resultados no mejoran y son aproximadamente iguales a los de la Tabla 9 y la Tabla 10. Con ello concluimos que usar este tipo de exploración no parece aportar una ventaja significativa de cara a los resultados finales.

5.2.5. Impacto del ruido y de la planificación

Para finalizar cabe preguntarse la razón de que la heurística MDP obtenga resultados ligeramente mejores en presencia de ruido balanceado que sin ruido. Para ello podemos fijarnos en los retardos medios por aplicación en las simulaciones. Según la Tabla 14, al pasar de un escenario sin ruido a uno con ruido balanceado las tasas de éxito prácticamente no se degradan en la red RRB. Esto es un claro indicador de que el ruido balanceado no supone una carga a mayores considerable, ya que la probabilidad de que el tiempo de procesamiento de un paquete aumente es igual a la de que disminuya, lo que provoca una compensación. Con ruido truncado (en el que solo se incrementa el tiempo de procesamiento con respecto a lo estimado) sí se observa un leve empeoramiento.

	Retardos medios (en ms) en la red RRB para MDP					
	Con planificación			**Sin planificación**		
	Sin ruido	**Balanceado**	**Truncado**	**Sin ruido**	**Balanceado**	**Truncado**
App 1	88.87	88.87	88.87	88.87	88.87	88.87
App 2	11.31	11.39	14.87	11.11	11.02	14.1
App 3	14.35	13.95	17.14	14.35	14.13	17.01
App 4	45.44	45.44	45.44	45.44	45.44	45.44
App 5	3.8	3.64	4.73	3.72	3.59	4.7
App 6	2.66	2.46	3.19	2.56	2.51	3.26

Tabla 14. Retardos medios por aplicación en la red con retardos y recursos bajos (RRB) para la heurística MDP.

Por otra parte, esto no es igual en la red RRA. La Tabla 15 muestra como los retardos medios de las aplicaciones sí empeoran tanto con ruido balanceado como con truncado. El empeoramiento es mayor con el truncado al solamente aumentar los tiempos de procesamiento, pero hay una excepción que explica los resultados ligeramente mejores del MDP en presencia de ruido balanceado con planificación según su tasa de éxito (Tabla 9).

	Retardos medios (en ms) en la red RRA para MDP					
	Con planificación			**Sin planificación**		
	Sin ruido	**Balanceado**	**Truncado**	**Sin ruido**	**Balanceado**	**Truncado**
App 1	0.58	0.59	0.77	0.58	0.58	0.77
App 2	158	60.79	235.7	157.65	195.35	246.33
App 3	482.88	482.88	482.88	482.88	482.88	482.88
App 4	95998.53	95998.53	95998.53	95998.53	995998.5	95998.53
App 5	248.43	280.92	340.95	248.45	283.2	340.44
App 6	164.22	184.55	244.88	163.98	201.71	256.07

Tabla 15. Retardos medios por aplicación en la red con retardos y recursos altos (RRA) para la heurística MDP.

El valor para la aplicación 2 pasa de unos 158 milisegundos a unos 60 milisegundos. La razón puede atribuirse a que esta aplicación tiene muy pocos bits de datos por paquete (Tabla 5), y la posibilidad de que el ruido produzca un tiempo de procesamiento menor en otra reserva es un cambio muy grande en la cola comparado con el tiempo de procesamiento de los paquetes de esta aplicación. Esto permite de forma aleatoria reservar paquetes de la aplicación 2 mucho antes en la cola de lo que se lograría cuando no hay ruido. Tal cosa no sería posible sin planificación (siempre se reserva al final de la cola y se ignoran los huecos intermedios), y confirmamos como el retardo medio de la aplicación no se reduce en este escenario. Esto proporciona la conclusión de que el ruido puede afectar en gran medida a

ciertas aplicaciones de la red (tanto para bien como mal), mientras que otras no notan un cambio significativo.

Se puede confirmar que esto ocurre viendo los retardos de la aplicación en ambos casos en forma de histograma. La Figura 15 muestra la distribución de los retardos de los paquetes de datos de la aplicación 2 en tres casos con planificación: MDP en la red RRA sin ruido, MDP en la red RRA con ruido balanceado y DDQN en la red RRA con ruido balanceado. La recompensa en todos los casos es por penalización, si bien esto no afecta al MDP.

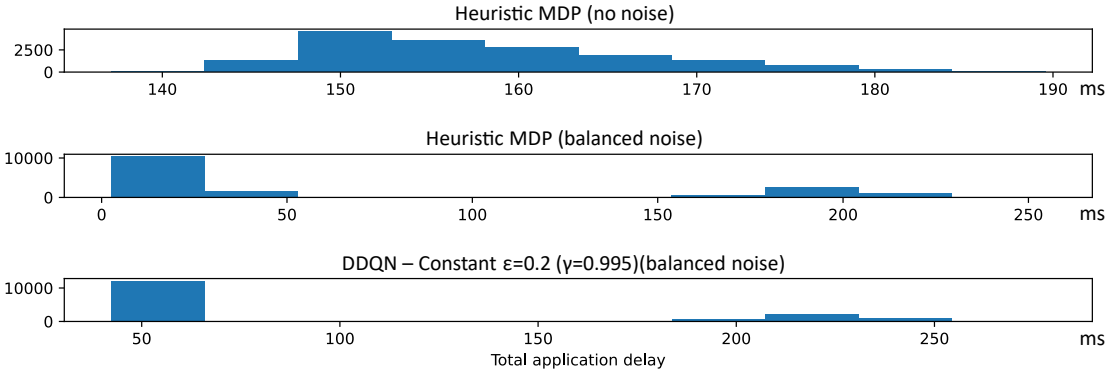

Figura 15. Histograma con retardos totales de los paquetes de datos de la aplicación 2 en la red con retardos y recursos altos (RRA).

Se observa como la distribución de los retardos de los paquetes de datos de la aplicación 2 para el MDP cambia al aparecer el ruido balanceado. Sin ruido todos estos paquetes se reservan en la cola aproximadamente en el mismo punto (estado estacionario de la red). Esto hace que todos sus retardos estén aproximadamente en los 150 milisegundos (incluyendo tiempos de ida y vuelta). Al haber ruido balanceado, reservas que están procesándose (al principio de la cola) pueden aleatoriamente tardar hasta un 50% menos de tiempo en procesarse. El hueco que esto genera puede ser suficiente para que una aplicación con tiempos de procesamiento estimado pequeños (como la 2) puedan ser reservados al principio de la cola. Esto produce la segunda distribución que observamos en la Figura 15, ya que si bien aún tenemos retardos en el orden de los 150 milisegundos (algo mayores porque el ruido empeora el estado de las colas), muchos de los paquetes se reservan al principio de la cola y obtienen retardos mucho menores. A mayores, se confirma que un agente de RL puede aprovechar estos huecos al igual que la heurística MDP ya que observamos una distribución muy similar (la tercera) con ruido balanceado para el agente DDQN.

5.2.6. Conclusiones

En conclusión, los agentes de RL obtienen resultados similares a los de las mejores heurísticas en escenarios donde la política óptima es obvia (y de ahí que ciertas heurísticas funcionen muy bien). Ahora bien, en otros escenarios, como es el RRA, los algoritmos de RL consiguen tasas de éxito y recompensas medias muy superiores a los de las heurísticas, demostrando por tanto ser una opción viable cuando la política óptima es desconocida o variable. Destaca el algoritmo PAL al obtener resultados excelentes con mayor frecuencia. El algoritmo TRPO también es una buena opción, pero requiere de entrenamientos mucho más largos.

Una observación fundamental es que el ruido de procesamiento es un factor que altera en gran medida el rendimiento de los agentes de RL, y debe ser considerado sobre todo si el ruido solamente puede aumentar los tiempos de procesamiento (ruido truncado). Es interesante identificar que según los resultados de la Tabla 7, Tabla 8, Tabla 9 y Tabla 10 los agentes de RL parecen tener una tendencia a obtener políticas mejores en presencia de ruido. Sin embargo, tal como se veía en la Figura 11, la Tabla 12 y la Tabla 13, los resultados de los algoritmos de RL pueden variar de simulación a simulación. Por ello, no se puede concluir que existe un patrón predecible donde unos escenarios favorecen más a los agentes de RL sin la realización de un estudio posterior al actual. Entrenar varios agentes de RL y realizar una etapa de validación puede ser una estrategia adecuada.

Dicho esto, una observación importante es que si el ruido puede producir tiempos de procesamientos menores, se pueden abrir huecos en las colas que permiten realizar reservas antes de lo normal. Esto solamente funciona al usarse planificación en los nodos de cómputo y es algo que tanto la heurística MDP como los agentes de RL pueden aprovechar en potencia.

Por último, el tipo de recompensa usado obviamente influye en el aprendizaje de los agentes de RL al ser un elemento fundamental. Aun así, es complicado sacar conclusiones sobre cuál de las recompensas usadas es mejor. De todas formas, se puede identificar un patrón en la Tabla 7 donde tanto el algoritmo PAL como el TRPO incrementan sus tasas de éxito en la red RRB en gran medida al cambiar de la recompensa por penalización a la binaria ponderada. Esto indica la importancia del uso de una adecuada recompensa, y que esto es dependiente de otros factores (como la red en este caso).

6. Experimento práctico (red con varias RSU)

6.1. Planteamiento

En esta sección se pretende mostrar el caso de una red más realista en la que ya se consideran varias RSU conectadas a un servidor MEC. El objetivo es analizar la viabilidad del uso del aprendizaje por refuerzo profundo en estos entornos.

6.1.1. Topología y set de aplicaciones

En la anterior sección (5) se plantearon dos topologías de red distintas junto con unos sets de aplicaciones. Viendo los resultados quedaba claro que en la red RRB no había mucho margen de mejora al deberse procesar la mayoría de las aplicaciones localmente. Por otro lado, la red RRA si mostraba un caso en el que era crucial una correcta distribución del procesamiento.

Por ello, en esta sección planteamos una red basada en la RRA, donde la diferencia es que habrá múltiples RSU a las que los vehículos pueden conectarse. La Figura 16 muestra el esquema de la red.

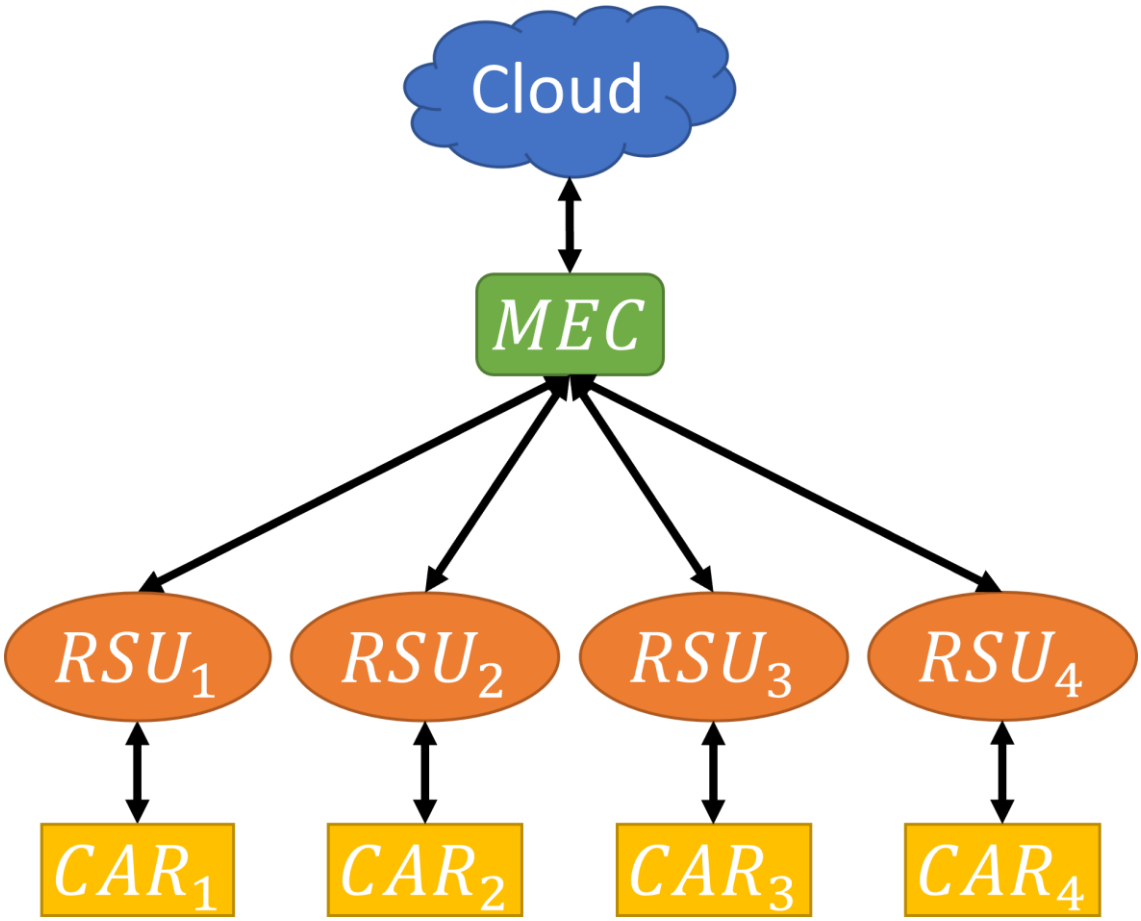

Figura 16. Esquema de la red con varias RSU.

Igual que antes, la red tiene un único nodo de nube con un servidor MEC conectado. La diferencia es que ahora este tiene cuatro puntos de acceso (RSUs). En la red todos los enlaces son iguales a los de la red RRA (Figura 9), pero hay cuatro enlaces entre MEC y RSUs, y RSUs y vehículos, en lugar de uno solo. Los recursos computacionales de los nodos y el set de aplicaciones son idénticos a los ya planteados anteriormente para la red RRA (Tabla 4 y Tabla 5).

6.1.2. Escenarios

Al igual que para el planteamiento de la topología, los escenarios que se simulan se basan en las observaciones obtenidas en la anterior sección (5). Debido a que en esta sección se pretende ofrecer un análisis más profundo de la red planteada, se reduce el número de escenarios simulados.

Si bien es obvio que el ruido de procesamiento en los nodos afecta a los resultados considerablemente, está claro que el ruido truncado proporciona una degradación más notable de los resultados que el balanceado (Tabla 6). Es por ello que en esta sección se consideran escenarios sin ruido y con ruido truncado. Así mismo, se consideran solamente escenarios donde los nodos usan planificación en sus colas (ver sección 3.3.3), debido a que los escenarios sin planificación en general solamente reducen la eficiencia del uso de las colas. En cuanto al tipo de recompensa, por penalización o binaria ponderada (ver sección 3.5.2), como el uso de una u otra produce cambios considerables en las políticas desarrolladas por los agentes, se consideran ambos tipos.

Respecto a otras variaciones que no se analizaron en la sección anterior, aquí se considera qué ocurre cuando el número de vehículos o el coeficiente de variación del ruido en la red son distintos. Por ello, se realizan simulaciones variando dichos parámetros. A mayores, se considera qué ocurre cuando los agentes son entrenados para cada variación de los parámetros y, por el contrario, qué ocurre cuando se entrenan solamente en un valor medio. Los agentes utilizados serán iguales a los anteriores y en las simulaciones las fases de entrenamiento tendrán un millón de pasos de tiempo, mientras que las de testeo cien mil, ambas independientes y con fase de inicialización de mil pasos.

6.2. Resultados

Pasamos ahora a comentar los resultados de experimentar con los escenarios planteados para la nueva red. Se analizan tanto la variación del número de vehículos en la red como la variación del coeficiente de variación del ruido, pero antes se proporciona una discusión sobre la forma adecuada de entrenar los agentes de RL ante estas variaciones.

6.2.1. Entrenamiento múltiple versus individual

En primer lugar, antes de analizar el comportamiento de los distintos algoritmos en situaciones donde varía un parámetro de la simulación, cabe preguntarse si es mejor entrenar a los agentes de RL múltiples veces para distintos valores del parámetro variable o una sola vez.

Obviamente realizar múltiples entrenamientos tiene la ventaja de, en principio, obtener políticas distintas para varias situaciones y por tanto mejorar los resultados. El problema principal de tal planteamiento es que es mucho más costoso computacionalmente, y no muy escalable. La alternativa de realizar un único entrenamiento es mucho más simple, y siempre y cuando el funcionamiento sea satisfactorio para escenarios distintos al del entrenamiento es una opción interesante.

Para comparar una opción con la otra la Figura 17 y la Figura 18 muestran un escenario donde se testearon los agentes en el mismo escenario variable (variación del número de vehículos). En la primera los agentes fueron entrenados independientemente en cada variación del número de vehículos, mientras que en la segunda se entrenaron una única vez con un valor medio (50 vehículos).

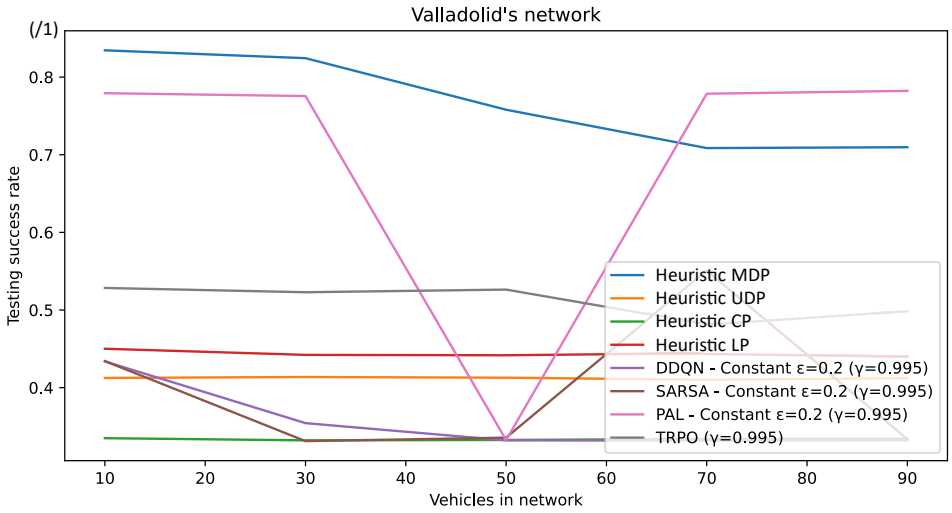

Figura 17. Tasas de éxito para distintos números de vehículos en la red sin ruido y con recompensa por penalización.

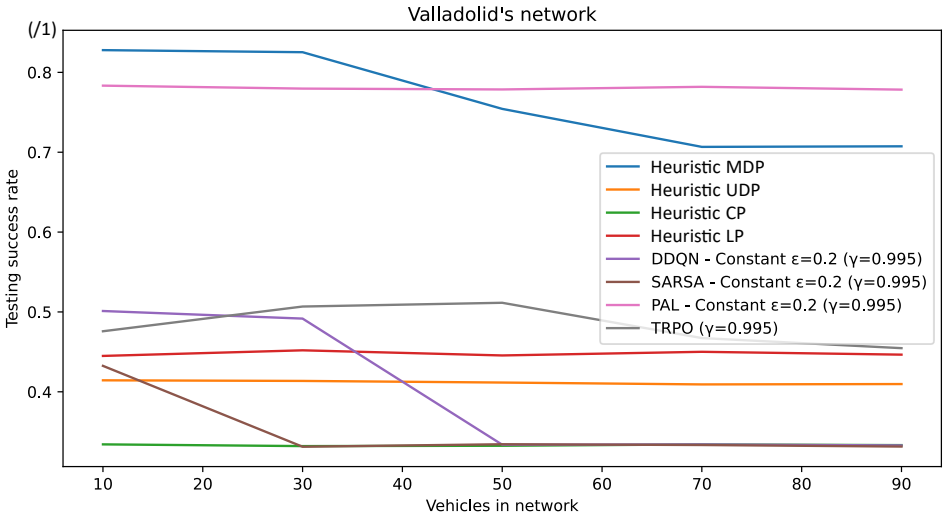

Figura 18. Tasas de éxito para distintos números de vehículos en la red sin ruido y con recompensa por penalización cuando los agentes de RL han sido entrenados solamente con 50 vehículos.

Si bien la Figura 17 y la Figura 18 solamente muestran un escenario, se repitió el análisis para otros escenarios (ruido truncado en la red y recompensa binaria ponderada), así como el caso en el que el parámetro variable era el coeficiente de ruido y no el número de vehículos, y en todos los casos se llega a la misma conclusión.

Si recordamos los experimentos realizados en la sección anterior (5.2) se comprobaba que distintos entrenamientos pueden producir políticas diferentes (Figura 11, Tabla 12 y Tabla 13). Esto se vuelve a observar al realizar múltiples entrenamientos variando un parámetro del entorno en la Figura 17. Se ve muy claramente debido a los picos que se generan en la curva, destacando en este caso la caída del rendimiento del agente PAL.

Por otra parte, un entrenamiento único produce una política consistente. La Figura 18 muestra dos ideas fundamentales. La primera es que una política obtenida entrenando en un escenario concreto puede ser buena para otros, tal como muestra el rendimiento del algoritmo PAL. En segundo lugar, se confirma que el pico de la curva del agente PAL en la Figura 17 se debe a un casual mal entrenamiento, ya que el entrenamiento de la Figura 18 es idéntico a ese mal entrenamiento, pero en este caso obtiene resultados a la par con los demás.

Si bien una solución al entrenamiento múltiple es repetir cada entrenamiento varias veces, esto es muy costoso. Esto también es una buena idea al realizar un único entrenamiento, ya que existe la posibilidad de que también sea malo, pero volvemos al problema del coste de simulación. Un futuro estudio que busque optimizar los algoritmos sería un buen lugar para considerar esta ampliación (es decir, entrenar varias veces y comparar los resultados de dichos entrenamientos en una fase de validación). Debido al coste computacional, para este estudio nos centraremos en el uso de un único entrenamiento.

6.2.2. Variación de número de vehículos

El primer parámetro que consideraremos variante en la red es el número de vehículos. A mayor cantidad, mayor es la carga sobre la red. Es de esperar que esto genere cambios en la política óptima y los agentes deben ser lo suficientemente flexibles. La Figura 19 y la Figura 20 muestran las tasas de éxito de todos los algoritmos con ambos tipos de recompensas para distintas cargas cuando son entrenados para una carga de 50 vehículos y no hay ruido.

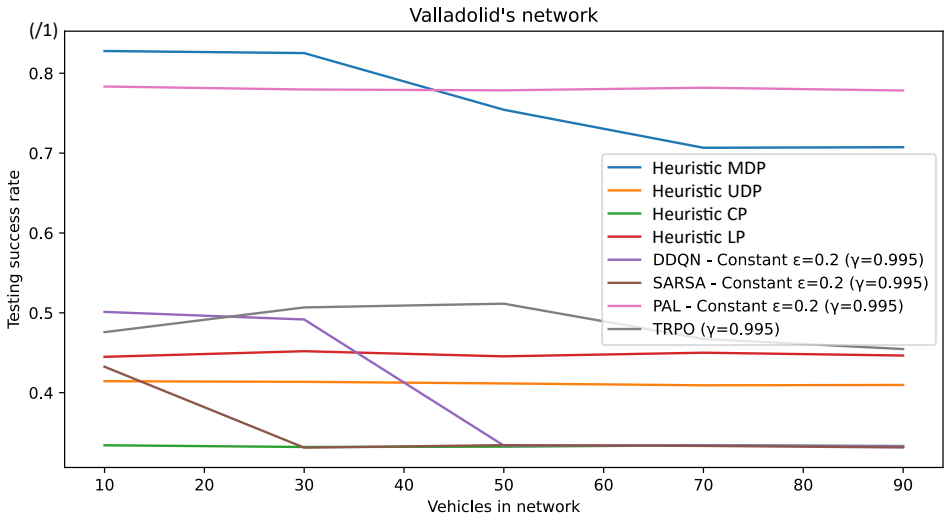

Figura 19. Tasas de éxito para distintos números de vehículos en la red sin ruido y con recompensa por penalización cuando los agentes de RL han sido entrenados solamente con 50 vehículos.

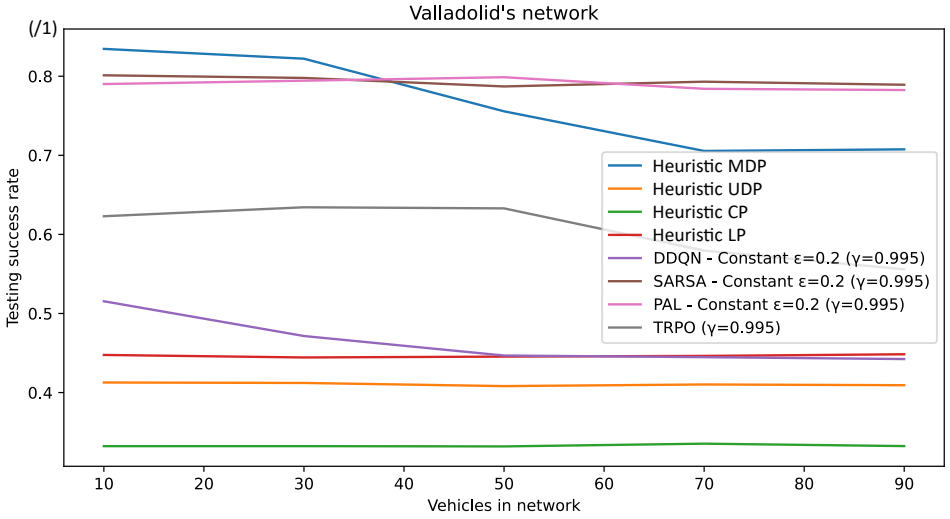

Figura 20. Tasas de éxito para distintos números de vehículos en la red sin ruido y con recompensa binaria ponderada cuando los agentes de RL han sido entrenados solamente con 50 vehículos.

Como se puede observar, el único algoritmo heurístico que funciona adecuadamente es el MDP. Sus resultados son buenos, pero se observa que a medida que la carga aumenta, su rendimiento se reduce. Por otro lado, el único algoritmo de RL que obtiene buenos resultados con la recompensa por penalización es el algoritmo PAL. Esto es consistente con los resultados de la sección anterior y fortalece la idea de que es el adecuado. Cuando la recompensa es binaria ponderada los resultados del aprendizaje por refuerzo mejoran considerablemente. Destaca el algoritmo SARSA al encontrar una política igual de buena que PAL y supera al algoritmo MDP para cargas altas.

En general se comprueba que el algoritmo MDP es utilizable en este caso, pero su rendimiento sufre al aumentar el número de vehículos en la red. Esto no es así para el algoritmo PAL, y el SARSA con la recompensa binaria ponderada. Estos algoritmos muestran un rendimiento consistente para cualquier valor de número de vehículos. Esto es una ventaja al demostrar mucha robustez ante cambios en el entorno. La mayor simplicidad del MDP lo hace más interesante cuando las cargas son bajas, pero cuando la cantidad de vehículos aumenta hasta 70, hay una diferencia del 10% en las tasas de éxito.

Debemos recordar que los algoritmos de RL no buscan aumentar la tasa de éxito en sí, sino aumentar su recompensa a largo plazo. Es por ello que se debe mirar la recompensa media en las mismas simulaciones en la Figura 21 y la Figura 22.

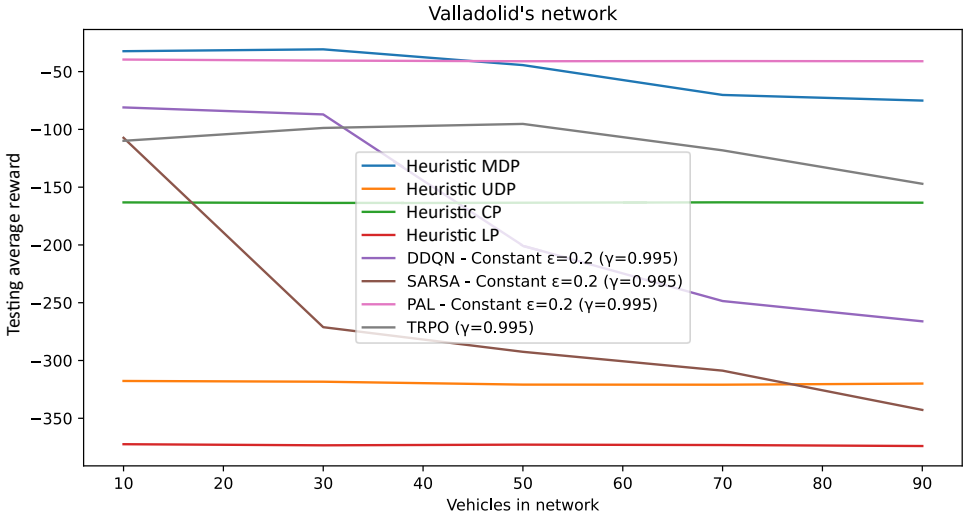

Figura 21. Recompensas medias para distintos números de vehículos en la red sin ruido y con recompensa por penalización cuando los agentes de RL han sido entrenados solamente con 50 vehículos.

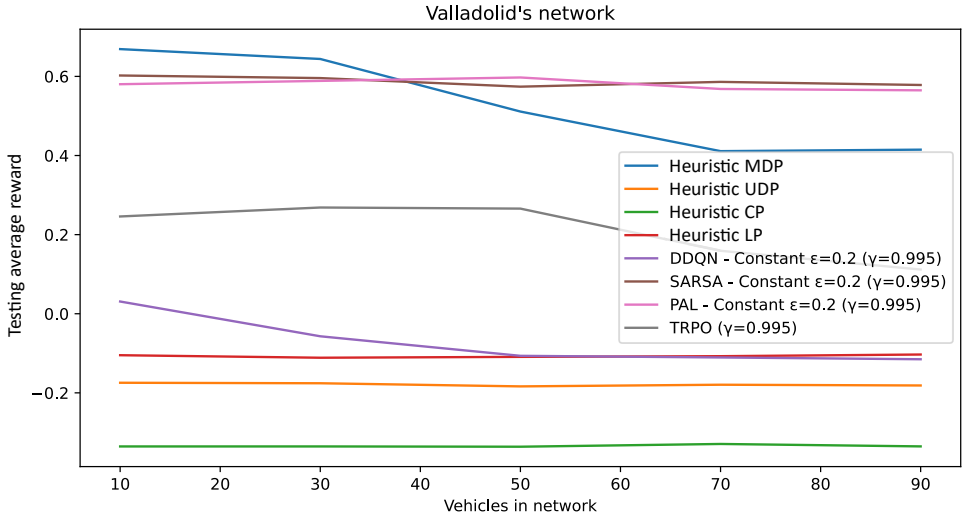

Figura 22. Recompensas medias para distintos números de vehículos en la red sin ruido y con recompensa binaria ponderada cuando los agentes de RL han sido entrenados solamente con 50 vehículos.

Comparando los resultados de las tasas de éxito y las recompensas medias se ve como los algoritmos de RL son mucho más competentes de lo que inicialmente podría parecer cuando la recompensa es por penalización, pero esto no se traduce correctamente a la tasa de éxito. Para la recompensa binaria ponderada las gráficas de la Figura 20 y la Figura 22 son prácticamente iguales, por lo que se puede concluir que aumentar esta recompensa es

prácticamente análogo a aumentar la tasa de éxito. Dicho esto, las conclusiones anteriores no cambian ya que los mejores algoritmos siguen siendo los mismos.

El algoritmo TRPO merece una mención especial ya que, si recordamos en la sección anterior se comentaba que requiere de un entrenamiento más largo para obtener resultados comparables al PAL (Figura 12). Para saber si este caso también es igual podemos observar la evolución de las recompensas promediadas del entrenamiento en la Figura 23 y la Figura 24.

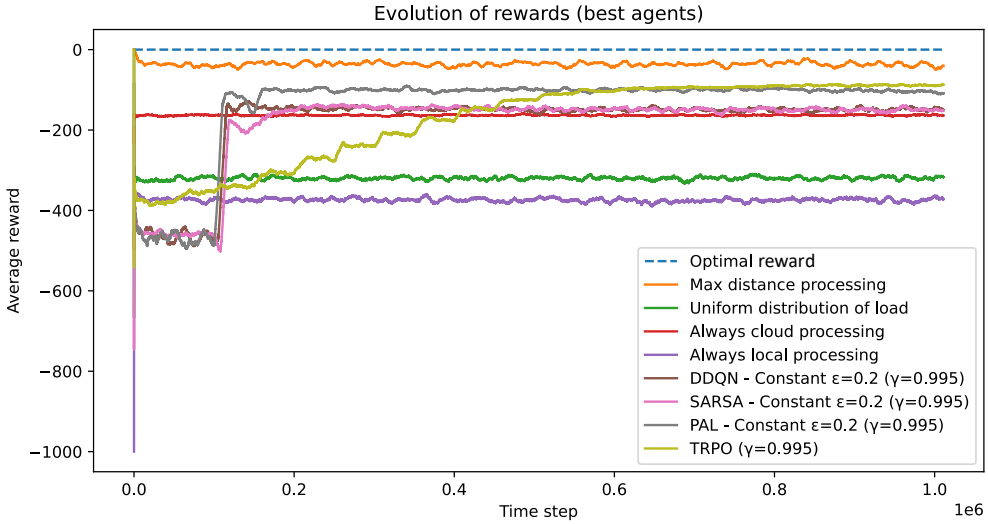

Figura 23. Evolución de las recompensas promediadas por penalización del entrenamiento (50 vehículos).

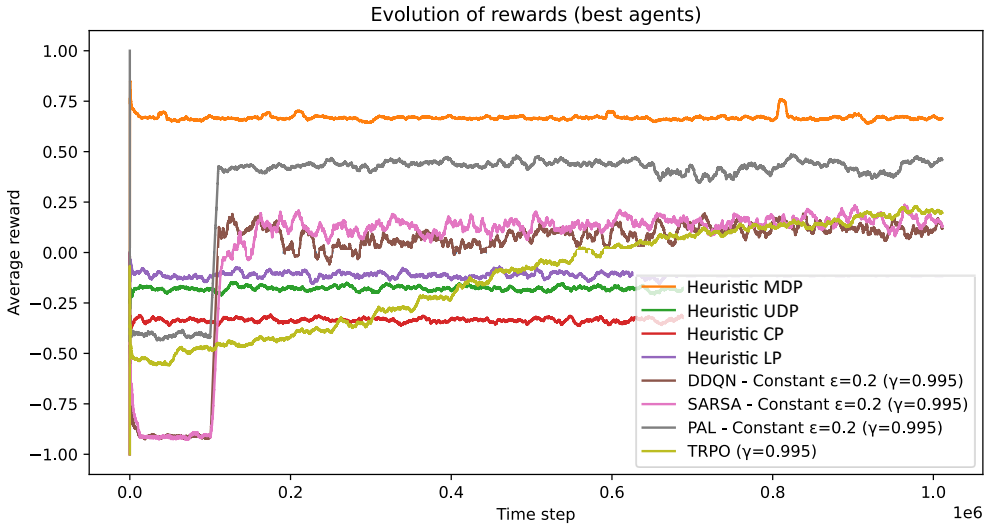

Figura 24. Evolución de las recompensas promediadas binarias ponderadas del entrenamiento (50 vehículos).

Como se puede ver, la evolución de las recompensas del agente TRPO parece tender asintóticamente al valor que ya alcanza al final de las simulaciones, por lo que no se debe esperar mejores resultados con un entrenamiento más largo. Esto se diferencia de lo observado anteriormente (sección 5.2.3) e indica que la posibilidad de mejora del TRPO está ligada al escenario.

Hay que tener en cuenta que en los experimentos que acabamos de comentar el escenario es en ausencia de ruido. En la anterior sección (5.2) ya se observaba que la presencia de ruido puede suponer cambios en las políticas de los agentes de RL. Para comprobar que pueden adaptarse a la aparición del ruido de procesamiento, a continuación, se analizan distintos escenarios de ruido.

6.2.3. Variación del coeficiente de variación del ruido

La variación del ruido truncado se presenta en las simulaciones como la variación del coeficiente de variación (CV) de la distribución gaussiana usada para generar el ruido (ver sección 3.4). Para este caso concreto, es interesante considerar dos entrenamientos, sin ruido y con ruido. La Figura 25 y la Figura 26 muestran las tasas de éxito en este escenario. En ambas simulaciones se considera un número de vehículos fijo de 50 (valor aproximado en el que el MDP igualaba la tasa de éxito con los mejores algoritmos de RL).

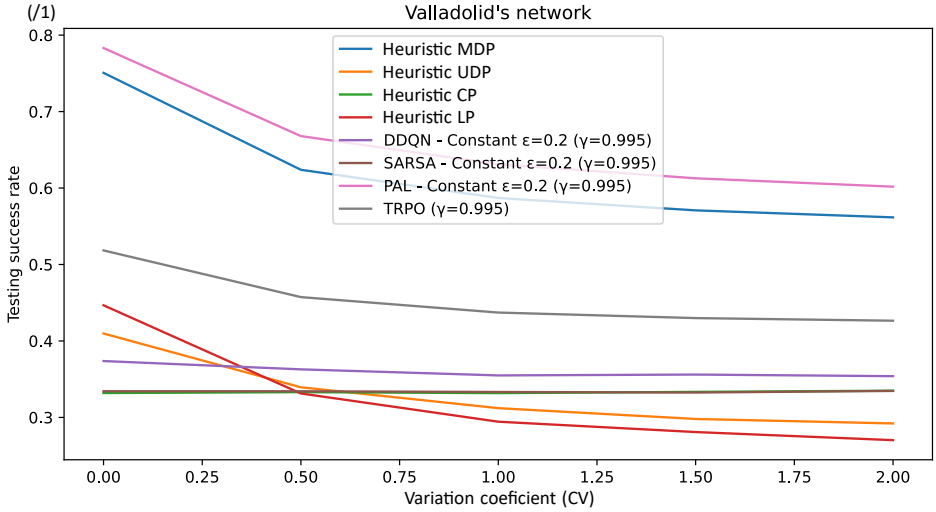

Figura 25. Tasas de éxito para distintos coeficientes de variación del ruido truncado en la red con recompensa por penalización cuando los agentes de RL han sido entrenados sin ruido.

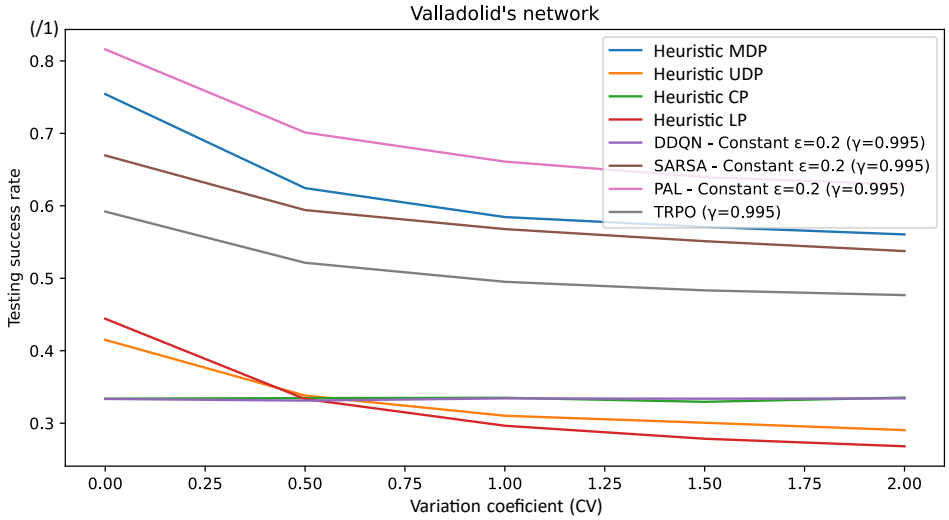

Figura 26. Tasas de éxito para distintos coeficientes de variación del ruido truncado en la red con recompensa binaria ponderada cuando los agentes de RL han sido entrenados sin ruido.

Como se puede ver los resultados de los algoritmos heurísticos en presencia de ruido son ligeramente inferiores al caso sin ruido. A medida que el coeficiente de variación del ruido aumenta, el cambio en los resultados se vuelve menor debido a que el ruido está truncado y dicho límite impide que empeoren demasiado. Sin embargo, fijándose en la tasa de éxito, vuelve a observarse que el único algoritmo heurístico con buenos resultados es el MDP.

Observando a los agentes de RL se vuelven a ver ideas similares a las discutidas cuando la variación era de vehículos. Nuevamente destaca el agente PAL, volviendo a demostrar ser una opción fiable a la hora de obtener buenos resultados. A mayores, a diferencia del caso

anterior ahora el agente PAL supera al MDP indiscutiblemente, con una ventaja en la tasa de éxito de aproximadamente el 5% para cualquier cantidad de ruido. En el caso de la recompensa binaria ponderada volvemos a ver que el agente SARSA consigue tasas de éxito similares al MDP. Sin embargo, en este caso es claramente peor que MDP y PAL.

Fijándose también en la recompensa media en la Figura 27 y la Figura 28, se vuelve a ver que los demás algoritmos de RL no son tan malos como podría parecer en principio cuando la recompensa es por penalización. Aun así, al igual que antes esto no implica que sean opciones mejores ya que el interés está en maximizar métricas como la tasa de éxito.

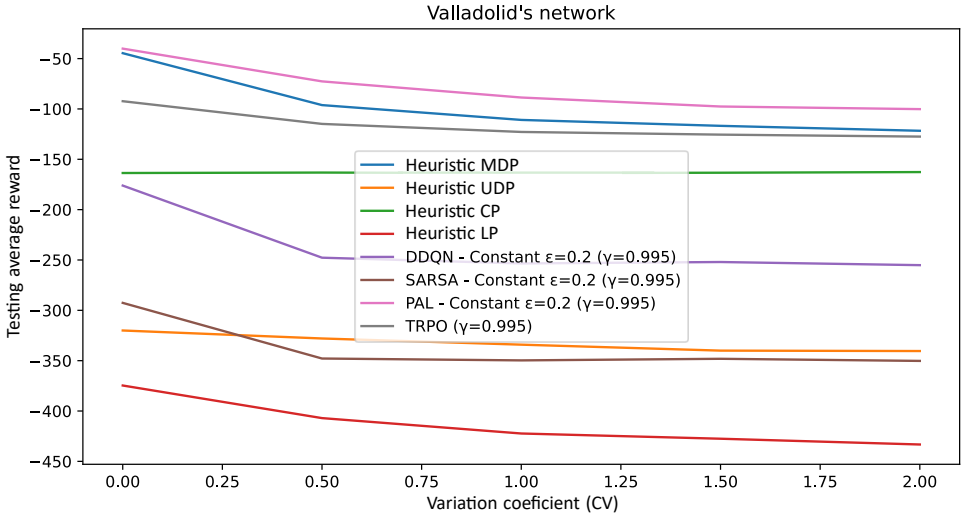

Figura 27. Recompensas medias para distintos coeficientes de variación del ruido truncado en la red con recompensa por penalización cuando los agentes de RL han sido entrenados sin ruido.

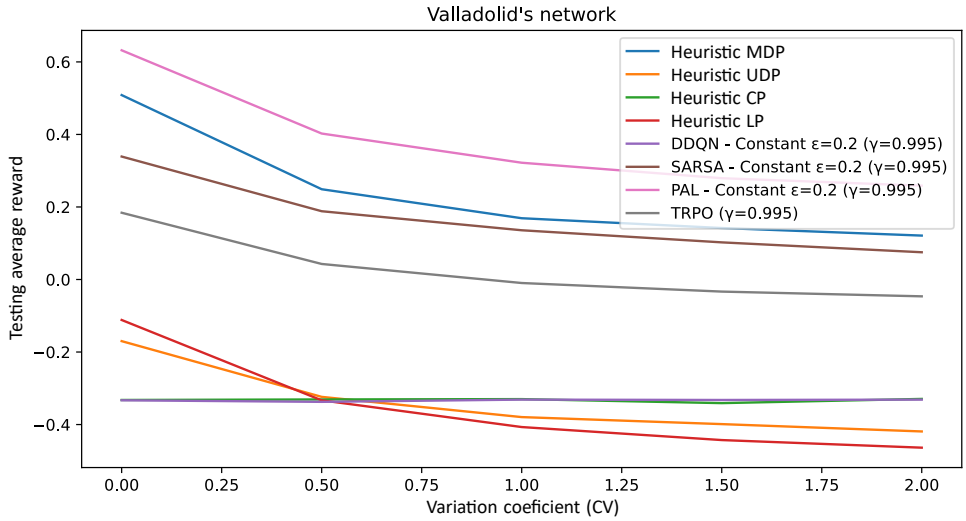

Figura 28. Recompensas medias para distintos coeficientes de variación del ruido truncado en la red con recompensa binaria ponderada cuando los agentes de RL han sido entrenados sin ruido.

Algo destacable es que el algoritmo TRPO demuestra un rendimiento muy similar al MDP cuando el ruido aumenta con recompensa por penalización (Figura 27). Para comprobar si está en el límite de sus posibilidades, la Figura 29 y la Figura 30 muestran la evolución de las recompensas para los entrenamientos anteriores. Al igual que en el caso de la variación de vehículos, no se observan muestras de que el TRPO pueda mejorar más sus resultados.

Figura 29. Evolución de las recompensas promediadas por penalización del entrenamiento sin ruido.

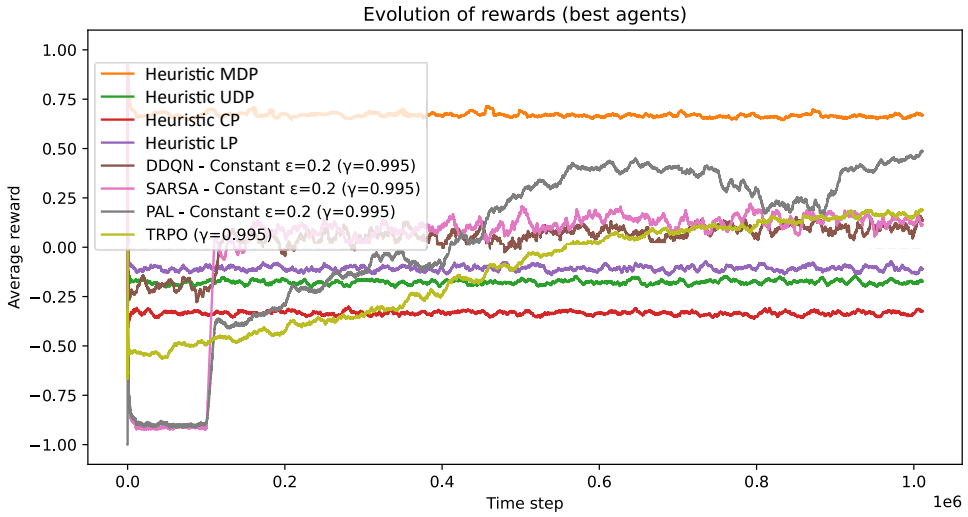

Figura 30. Evolución de las recompensas promediadas binarias ponderadas del entrenamiento sin ruido.

Continuando con el análisis, tal como comentábamos antes, es interesante comprobar si entrenar a los agentes de RL con ruido es mejor. Para ello las mismas pruebas pero con entrenamiento con ruido (de nuevo se consideran 50 vehículos) se muestran en la Figura 31, Figura 32, Figura 33 y Figura 34.

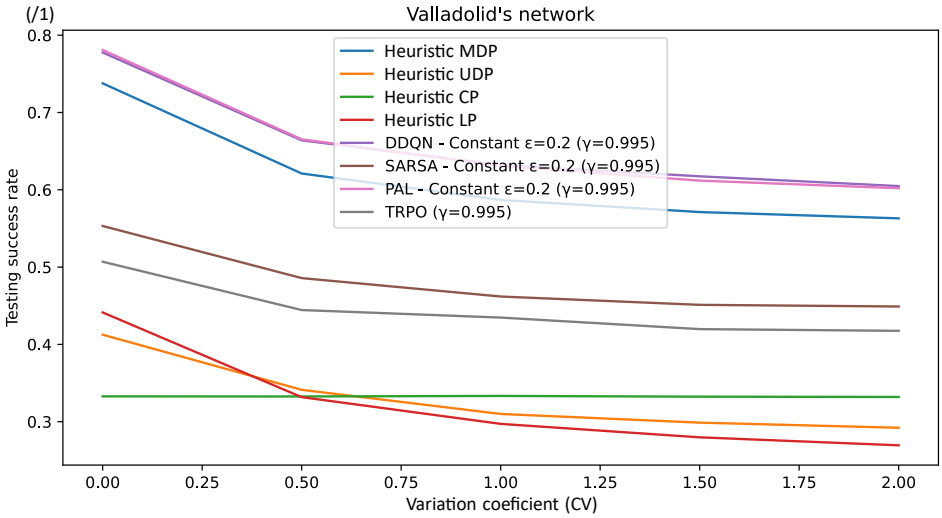

Figura 31. Tasas de éxito para distintos coeficientes de variación del ruido truncado en la red con recompensa por penalización cuando los agentes de RL han sido entrenados con coeficiente de variación de 1.

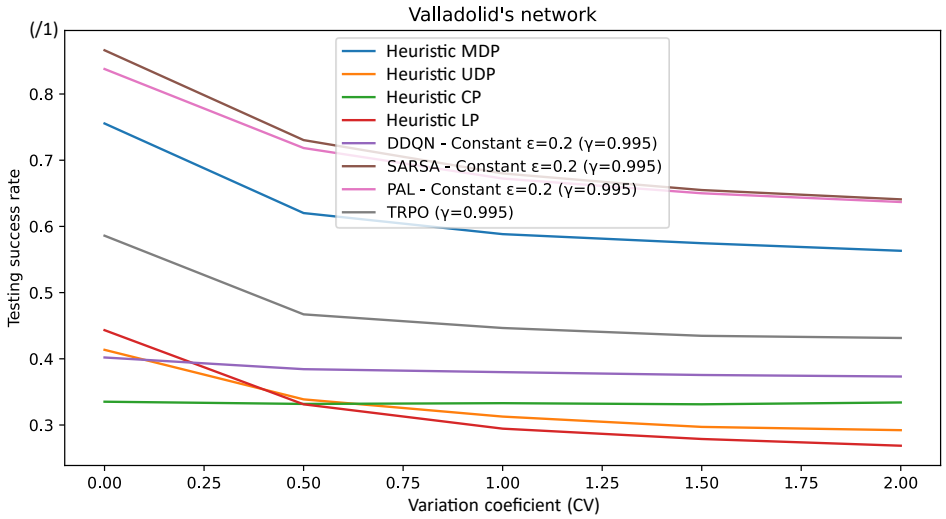

Figura 32. Tasas de éxito para distintos coeficientes de variación del ruido truncado en la red con recompensa binaria ponderada cuando los agentes de RL han sido entrenados con coeficiente de variación de 1.

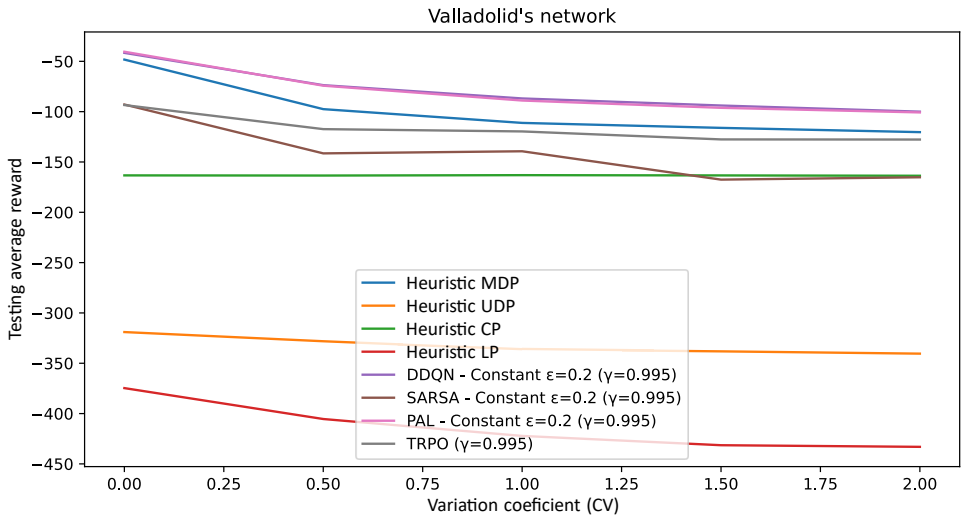

Figura 33. Recompensas medias para distintos coeficientes de variación del ruido truncado en la red con recompensa por penalización cuando los agentes de RL han sido entrenados con coeficiente de variación de 1.

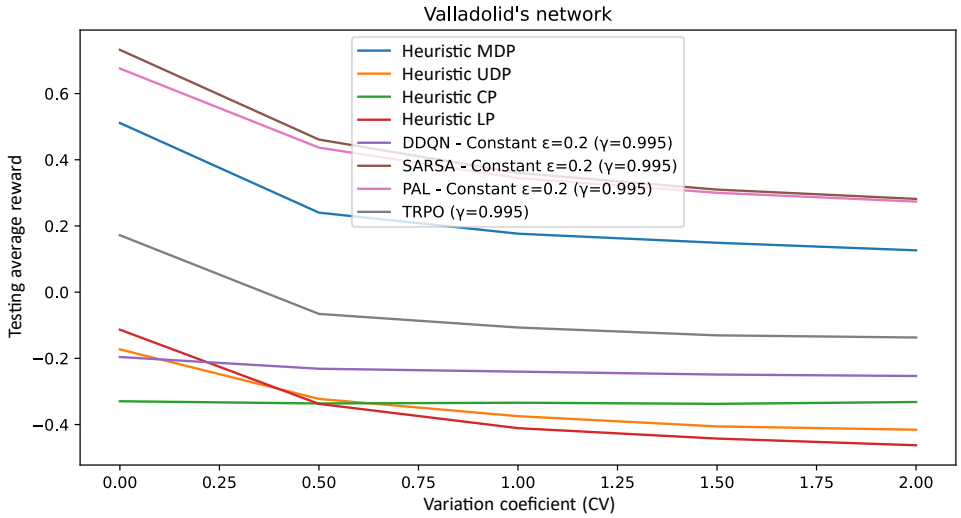

Figura 34. Recompensas medias para distintos coeficientes de variación del ruido truncado en la red con recompensa binaria ponderada cuando los agentes de RL han sido entrenados con coeficiente de variación de 1.

Los resultados son de forma generalizada similares al caso del entrenamiento sin ruido, aunque hay diferencias importantes que deben ser mencionadas. Para empezar, se observa que todos los algoritmos de RL como mínimo no empeoran con respecto al caso anterior. Vuelve a verse los excelentes resultados del algoritmo PAL una vez más en este caso. Ahora bien, es interesante observar que cuando se utiliza la recompensa por penalización, DDQN alcanza aproximadamente la misma tasa de éxito y recompensa media que PAL. Cuando se utiliza la recompensa binaria ponderada, es SARSA el que obtiene un rendimiento similar al de PAL.

No es la primera vez que se observa esto para el algoritmo SARSA (Figura 20 y Figura 22), por lo que una justificación es que este funciona bien para la recompensa binaria ponderada. Por otra parte, el resultado del DDQN es más sorprendente ya que no se han visto otros casos para esta red en los que iguale al PAL. Sin embargo, sí hay ejemplos de que este y otros algoritmos pueden alcanzar al PAL en la sección anterior (Tabla 7 y Tabla 9) por lo que es justificable.

Esto se relaciona con la conclusión obtenida anteriormente sobre la obtención de distintas políticas al realizar varios entrenamientos. Sencillamente a lo largo de esta memoria se ha observado claramente cómo el algoritmo PAL es el que más frecuentemente encuentra una excelente política. Esto significa que de repetir los entrenamientos de los demás agentes de RL estos posiblemente en ciertos casos podrían encontrar políticas similares. Aun así, no hay razón para recomendar ninguno por encima del PAL, pues es el más consistente obteniendo resultados superiores a las heurísticas (tasa de éxito de hasta el 90% en la Figura 32, y muy frecuentemente del 80% en todas los demás).

6.2.4. Conclusiones

En esta sección se ha experimentado variando tanto el número de vehículos como el coeficiente de variación del ruido truncado en una red con varias RSU. Inicialmente se consideró si era preferible realizar múltiples entrenamientos para adaptar a los agentes de RL a las variaciones del entorno. Sin embargo, se ha demostrado cómo un único entrenamiento también puede ser una opción viable, ya que las políticas obtenidas siguen siendo efectivas ante cambios y suponen una carga computacional sustancialmente menor. De todas formas, una línea futura será evaluar la mejora al considerar varios entrenamientos.

Los experimentos al variar tanto el número de vehículos como la cantidad de ruido en la red muestran que esto empeora los resultados de las heurísticas, de las cuales solamente el algoritmo MDP es satisfactorio de cara a resolver el problema de delegación de tareas. En contraste con esto, los algoritmos de aprendizaje por refuerzo han mostrado ser capaces de resolver el problema con la ventaja de que sus prestaciones sufren menor degradación al incrementarse el número de vehículos en la red. Además, en presencia de ruido son superiores al MDP en entre 5 a 10% en la tasa de éxito.

Estos resultados muestran la superioridad del aprendizaje por refuerzo ante soluciones heurísticas para la delegación de tareas, diferencia que se agrava a medida que el problema se vuelve más complejo y la política óptima es desconocida. Concretamente cabe destacar el algoritmo PAL, que muy frecuentemente es capaz de obtener una excelente política. Otros algoritmos como DDQN o SARSA también son una opción, pero requieren más entrenamientos y por tanto son menos recomendables. También existe la posibilidad de usar algoritmos basados en descenso del gradiente como el TRPO, pero no existen garantías de que sus resultados sean mejores, aunque se aumente la longitud del entrenamiento.

Es necesario mencionar que, si el problema no es complejo, el uso de la heurística MDP es preferible por su sencillez. Sin embargo, para un sistema realista esta heurística será incapaz de adaptarse a inevitables cambios en el entorno, y altas cargas o mucho ruido degradan su rendimiento, siendo en esos casos la solución más recomendable el uso del algoritmo PAL.

7. Conclusiones y líneas futuras

7.1. Conclusiones

En este TFM se ha tratado el uso del aprendizaje por refuerzo (*Reinforcement Learning*, RL) [3] para la resolución del problema de delegación de tareas en redes vehiculares. Concretamente se ha abordado el problema mediante aprendizaje por refuerzo profundo (*Deep* RL) usando redes neuronales para implementar una serie de algoritmos basados en DQN (*Deep-Q-Network*) [7], DDQN [8], SARSA [3] y PAL [9], y otro basado en descenso de gradiente (*Gradient Descent*, DG) denominado TRPO [10].

Un punto clave en el problema planteado son las reservas dinámicas en los nodos de cómputo (sección 3.3.3). La principal novedad que aporta esta memoria es la idea de que los tiempos de procesamiento en los nodos de cómputo no necesariamente son estáticos, y pueden cambiar con el tiempo. Modelamos esto con una distribución gaussiana truncada cuya varianza es proporcional al tiempo estimado de procesamiento de los paquetes de datos (sección 3.4).

Las observaciones al respecto son que el ruido de procesamiento puede no degradar el funcionamiento de la red siempre y cuando afecte tanto de forma positiva como negativa a los tiempos (ruido balanceado). Esto ocurre en la red RRB (Tabla 14) pero no en la RRA (Tabla 15), donde los retardos medios de muchas aplicaciones se incrementan ante la presencia del ruido balanceado. Un detalle interesante es que la repentina reducción del tiempo de procesamiento de un paquete de datos puede abrir un hueco en una cola que permite la reserva de otro paquete antes de lo normal. Esto solamente puede aprovecharse si se usa planificación en los nodos de cómputo y es un factor importante a la hora de reducir latencias para ciertas aplicaciones. De haber ruido que solamente empeora las prestaciones (ruido truncado), este sí degrada el funcionamiento de la red en cualquier caso estudiado, observándose caídas en el rendimiento de distintos algoritmos (Tabla 7 y Tabla 9).

En general se puede afirmar que la variación impredecible de los tiempos de cómputo no es un asunto trivial y debe ser tenido en cuenta al diseñar un controlador igual que otros factores variantes en el tiempo.

En cuanto al uso del aprendizaje por refuerzo, en el análisis propuesto se demostró como es una solución viable al problema. Se observó cómo es posible obtener políticas que superan a las de alternativas más convencionales (heurísticas), las cuales son incapaces de adaptar sus políticas a la complejidad de la toma de decisiones.

En un caso donde la política óptima es predecible como el de la red RRB, los algoritmos de RL obtenían políticas muy similares al MDP (Tabla 7 y Tabla 8). Si bien en dicho escenario el uso de algoritmos como el propuesto MDP parece preferible debido a su mayor sencillez, es importante observar que sus prestaciones se degradan rápidamente al pasar a un caso más complejo. Esto se notaba en la red RRA donde la política óptima no era predecible a priori y el rendimiento del MDP resultaba ser inferior al de algoritmos de RL (Tabla 9 y Tabla 10).

Hay que mencionar que a la hora de diseñar heurísticas no se puede usar más que información aproximada de la red (sección 3.5.1). Esto complica la tarea de diseño exponencialmente al acercarse más a un sistema real, y entra en duda la posibilidad de tener en cuenta todos los factores relevantes. Por otro lado, los algoritmos de RL funcionan muy bien con información limitada del entorno, mostrando gran flexibilidad entre escenarios tal como demuestran los buenos resultados en todos los experimentos planteados.

A mayores, al realizar experimentos variando el número de vehículos y el coeficiente de variación del ruido en una red con varias RSU basada en la RRA se observaba como el RL obtiene políticas con robustez suficiente como para afrontar estos cambios (Figura 19, Figura 20, Figura 25, Figura 26, Figura 31 y Figura 32).

Por todo ello, es recomendable el uso del RL para el problema de la delegación de tareas, al ser las heurísticas una alternativa con poca escalabilidad y haber margen de mejora a la hora de implementar los agentes de RL (optimizar la red neuronal y configuración).

Un detalle importante visto en los experimentos es que los algoritmos de RL no siempre obtienen una buena política. Una observación fundamental era que entrenar a agentes idénticos en exactamente el mismo escenario podía producir resultados distintos (Figura 11, Tabla 12 y Tabla 13). La solución no parecía ser los entrenamientos más largos, sino entrenar a los agentes de RL varias veces y tomar la mejor política encontrada. No era necesario encontrar múltiples políticas para hacer frente a la variación del entorno, pero aun así tener que realizar varios entrenamientos es computacionalmente una desventaja. Para mitigar esto es importante notar que no todos los algoritmos de RL son igual de fiables en ese sentido. En este estudio destaca el algoritmo PAL, que obtiene con la mayor frecuencia políticas excelentes y es por tanto la opción recomendada.

7.2. Líneas futuras

En el planteamiento propuesto y las simulaciones realizadas se consideran ciertas simplificaciones. Son algo a tener en cuenta para futuros trabajos, por lo que esta sección se usa para describirlas.

Una primera simplificación ha sido no considerar los retardos asociados al plano de control de la red, esto es, no se han considerado los tiempos necesarios para enviar las peticiones al agente y recibir sus respuestas una vez procesadas. En la práctica, estos retardos existen y pueden tener un impacto muy significativo en las prestaciones.

Otra simplificación ha sido considerar que no hay movilidad en la red, aunque estamos tratando con una red vehicular. Es obvio que en la realidad existirá tal movilidad y es un punto importante a considerar. En este caso se omitió ya que los tiempos de procesamiento de los paquetes de datos están en el orden de los milisegundos, por lo que se puede considerar que los vehículos siempre se encontraran conectados al mismo punto de acceso.

Otra aproximación importante ha sido considerar los enlaces de la red como invariantes en el tiempo. Nuevamente, al tratarse de tiempos de procesamiento cortos, se espera que las variaciones no sean demasiado significativas. Sin embargo, este punto y el anterior son fundamentales a la hora de considerar aplicaciones con paquetes de datos muy grandes.

La ampliación de los experimentos para considerar los puntos anteriores, así como pruebas en redes modeladas a partir de redes reales (y escaladas a áreas más grandes) completarían este estudio en gran medida. Además, se debe mencionar que la búsqueda de redes neuronales mejor optimizadas para el problema, así como otras configuraciones y algoritmos de RL pueden llevar a mejores resultados. Por lo tanto, podemos componer la siguiente lista de las líneas futuras del trabajo:

- Considerar los retardos en el plano de control de la red.
- Considerar movilidad en los vehículos.
- Considerar variabilidad en los enlaces.
- Considerar aplicaciones de gran tamaño.
- Modelar redes a partir de redes reales.
- Optimizar la red neuronal empleada para implementar los agentes de RL.
- Optimizar la configuración de los agentes de RL, incluyendo el uso de varios entrenamientos seleccionando el más adecuado en una etapa de validación.
- Probar otros algoritmos de RL.

Referencias

[1] Instituto de Estándares de Telecomunicaciones Europeo (ETSI), «Multi-access Edge Computing (MEC),» [En línea]. Disponible en: https://www.etsi.org/technologies/multi-access-edge-computing. [Último acceso: 19 Marzo 2022].

[2] A. Shakarami, M. Ghobaei-Arani y A. Shahidinejad, «A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective,» *Computer Networks,* vol. 182, 2020.

[3] R. S. Sutton y A. Barto, Reinforcement Learning: An Introduction (Second Edition), MA: MIT Press, 2018.

[4] M. Ferens, «Enrutamiento y establecimiento dinámico de conexiones en redes de transporte mediante aprendizaje por refuerzo,» Trabajo Fin de Grado en Ingeniería de Telecomunicación, E.T.S.I. de Telecomunicación, Universidad de Valladolid, 5 Octubre 2020. [En línea]. Disponible en: https://uvadoc.uva.es/handle/10324/42740?show=full. [Último acceso: 31 Marzo 2022].

[5] OpenAI, «Gym,» [En línea]. Disponible en: https://github.com/openai/gym. [Último acceso: 19 Marzo 2022].

[6] Y. Fujita, T. Kataoka, P. Nagarajan y T. Ishikawa, «ChainerRL,» [En línea]. Disponible en: https://github.com/chainer/chainerrl. [Último acceso: 19 Marzo 2022].

[7] V. Mnih, K. Kavukcuoglu y D. Silver, «Human-level control through deep reinforcement learning,» *Nature,* vol. 518, pp. 529-533, 2015.

[8] H. v. Hasselt, A. Guez y D. Silver, «Deep Reinforcement Learning with Double Q-learning,» *AAAI,* vol. 30, nº 1, 2016.

[9] M. G. Bellemare, G. Ostrovski, A. Guez, P. S. Thomas y R. Munos, «Increasing the Action Gap: New Operators for Reinforcement Learning,» *AAAI,* vol. 30, nº 1, 2016.

[10] J. Schulman, S. Levine, P. Moritz, M. I. Jordan y P. Abbeel, «Trust Region Policy Optimization,» de *Proceedings of the 31st International Conference on Machine Learning (ICML)*, Lille, 2015.

[11] J. Schulman, P. Moritz, S. Levine, M. I. Jordan y P. Abbeel, «High-Dimensional Continuous Control Using Generalized Advantage Estimation,» 2018. [En línea]. Disponible en: https://arxiv.org/abs/1506.02438.

[12] H. Hao, C. Xu, L. Zhong y G.-M. Muntean, «A Multi-update Deep Reinforcement Learning Algorithm for Edge Computing Service Offloading,» *Proceedings of the 28th ACM International Conference on Multimedia,* pp. 3256-3264, 2020.

[13] M. H. Eiselt y F. Azendorf, «Accurate Measurement of Propagation Delay in a Multi-Span Optical Link,» *International Topical Meeting on Microwave Photonics (MWP),* pp. 1-3, 2019.

[14] WikiLeaks, «Map of Amazon's Data Centers,» [En línea]. Disponible en: https://wikileaks.org/amazon-atlas/map/. [Último acceso: 30 Marzo 2022].

[15] B. Bilgin y G. V.C., «Performance Comparison of IEEE 802.11p and IEEE 802.11b for Vehicle-to-Vehicle Communications in Highway, Rural, and Urban Areas,» *International Journal of Vehicular Technology,* vol. 2013, Enero 2013.

[16] Rohde-Schwarz, «Intelligent Transportation Systems Using IEEE 802.11p,» 14 Febrero 2019. [En línea]. Disponible en: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_application/application_notes/1ma152/1MA152_5e_ITS_using_802_11p.pdf. [Último acceso: 30 Marzo 2022].

[17] Y. Wang, X. Duan, D. Tian, G. Lu y H. Yu, «Throughput and Delay Limits of 802.11p and its Influence on Highway Capacity,» *Procedia - Social and Behavioral Sciences,* vol. 96, pp. 2096-2104, 2013.

[18] NGMN (Next Generation Mobile Networks) Alliance, «5G WHITE PAPER,» 17 Febrero 2015. [En línea]. Disponible en: https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf. [Último acceso: 30 Marzo 2022].

[19] E. Coronado, G. Cebrian-Marquez y R. Riggio, «Enabling Computation Offloading for Autonomous and Assisted Driving in 5G Networks,» *IEEE Global Communications Conference (GLOBECOM),* nº 1-6, 2019.

[20] Amazon, «Amazon EC2,» [En línea]. Disponible en: https://aws.amazon.com/es/ec2/. [Último acceso: 30 Marzo 2022].

[21] Intel, «Procesadores Intel Xeon E,» [En línea]. Disponible en: https://www.intel.es/content/www/es/es/products/details/processors/xeon/e.html. [Último acceso: 30 Marzo 2022].

[22] D. Yuen, «The Future Begins with The Road Side Unit,» 8 Octubre 2020. [En línea]. Disponible en: https://medium.com/predict/edge-computing-is-so-much-more-fun-ac2a8a23e696. [Último acceso: 30 Marzo 2022].

[23] AMD, «Automotive Grade Zynq-7000 SoCs,» [En línea]. Disponible en: https://www.xilinx.com/products/silicon-devices/soc/xa-zynq-7000.html#productTable. [Último acceso: 30 Marzo 2022].

[24] S. Liang, H. Wan, T. Qin, J. Li y W. Chen, «Multi-user Computation Offloading for Mobile Edge Computing: A Deep Reinforcement Learning and Game Theory Approach,» *IEEE 20th International Conference on Communication Technology (ICCT),* pp. 1534-1539, 2020.

[25] H. Liu, H. Zhao, L. Geng y W. Feng, «A Policy Gradient Based Offloading Scheme with Dependency Guarantees for Vehicular Networks,» *IEEE Globecom Workshops (GC Wkshps),* pp. 1-6, 2020.

[26] H. Liu, H. Zhao, L. Geng, Y. Wang y W. Feng, «A Distributed Dependency-Aware Offloading Scheme for Vehicular Edge Computing Based on Policy Gradient,» *8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom),* pp. 176-181, 2021.

[27] 5G Automotive Association (5GAA), «C-V2X Use Cases Methodology, Examples and Service Level Requirements Whitepaper,» [En línea]. Disponible en: https://www.everythingrf.com/whitepapers/details/3617-C-V2X-Use-Cases-Methodology-Examples-and-Service-Level-Requirements. [Último acceso: 30 Marzo 2022].