Universidad Nacional de Asunción
Facultad Politécnica

Ingeniería en Informática

Scalable and Decentralized Urban Traffic Optimization with Verifiable Storage for Smart City Deployments

Kevin Mathias Galeano Saldivar
María José Duarte Kowalewski

Proyecto de Trabajo de Grado presentado en conformidad a los requisitos para obtener el grado de Ingeniería en Informática
San Lorenzo - Paraguay
Febrero - 2026

Universidad Nacional de Asunción
Facultad Politécnica

Ingeniería en Informática

Scalable and Decentralized Urban Traffic Optimization with Verifiable Storage for Smart City Deployments

Kevin Mathias Galeano Saldivar
María José Duarte Kowalewski

San Lorenzo - Paraguay
Febrero - 2026

Universidad Nacional de Asunción
Facultad Politécnica

Ingeniería en Informática

Scalable and Decentralized Urban Traffic Optimization with Verifiable Storage for Smart City Deployments

Kevin Mathias Galeano Saldivar
María José Duarte Kowalewski

Asesor:
PhD Marcos Villagra

Proyecto de Trabajo de Grado presentado en conformidad a los
requisitos para obtener el grado de Ingeniería en Informática
San Lorenzo - Paraguay
Febrero - 2026

Dedico a mi blah blah blah,
a mi blah blah blah.

AGRADECIMIENTOS

Titulo1
Titulo2

Autor: Nombre y apellido

Asesor: Nombre y apellido

RESUMEN

Este es el resumen

Palabras Clave:

Title

Author: full name

Advisor: full name

ABSTRACT

This is the abstract

Keywords:

ÍNDICE

Página

LISTA DE FIGURAS

Página

LISTA DE TABLAS

Página

LISTA DE SÍMBOLOS

CAPÍTULO 1

INTRODUCCIÓN

1.1. Introducción

CAPÍTULO 2
MARCO TEÓRICO

2.1. Ingeniería de Tráfico y Control Semafórico

2.1.1. Semáforos para el Control del Tránsito de Vehículos

Los semáforos constituyen el principal dispositivo de control del derecho de paso en intersecciones urbanas y periurbanas con conflictos significativos de circulación vehicular y peatonal [1]. En el *Manual de Carreteras del Paraguay* (Unidad 3, Volumen 3.3, Sección 3.3.2.7) se establece que los semáforos para el control del tránsito de vehículos deben emplearse para asignar tiempos de paso y de detención a los distintos movimientos que confluyen en la intersección, con el objetivo de reducir conflictos, ordenar las maniobras y aumentar la seguridad vial [1]. Esta función se complementa con el control de flujos peatonales y, cuando corresponde, con fases específicas para giros, carriles exclusivos y prioridades especiales (por ejemplo, transporte público o vehículos de emergencia) [1].

Desde el punto de vista de la ingeniería de tráfico, un semáforo vehicular puede interpretarse como un sistema de control discreto en el tiempo que alterna estados de indicación verde, amarilla y roja para cada grupo de movimientos, de forma tal que en ningún instante se autoricen simultáneamente movimientos incompatibles [1, 2]. El diseño del plan de fases considera la geometría de la intersección, los volúmenes de tránsito por aproximación, la presencia de giros a la izquierda y a la derecha, los flujos peatonales y, en su caso, la coordinación con semáforos próximos en el corredor [1, 3]. Las indicaciones luminosas verde, amarilla y roja poseen significado normado asociado a las obligaciones legales de detenerse o avanzar, y se complementan con flechas direccionales cuando es necesario discriminar maniobras permitidas dentro de una misma aproximación [1].

En la normativa paraguaya se especifican además criterios de ubicación, número de caras y

área de influencia de cada cabezal semafórico, de modo que las indicaciones resulten claramente visibles para todos los usuarios a los que se dirigen [1]. Estos lineamientos aseguran que el dispositivo de control implementado por el sistema desarrollado en el presente trabajo sea compatible con la infraestructura semafórica existente y respetuoso de las disposiciones vigentes en el país.

2.1.2. Semáforos de Tiempos Fijos o Predeterminados

Los semáforos de tiempos fijos o predeterminados operan con un plan de señales cuya secuencia de fases y tiempos se define previamente y se repite cíclicamente, sin interacción directa con el tránsito en tiempo real [1]. El *Manual de Carreteras del Paraguay* indica que estos equipos asignan a cada fase un tiempo de verde y un tiempo intermedio (amarillo y, si corresponde, tiempo de despeje en rojo) calculados a partir de volúmenes de tránsito de diseño, análisis de capacidad y niveles de servicio objetivo [1]. Los tiempos de fase se mantienen constantes durante el período de operación del plan, aunque se contempla el uso de distintos planes fijos según la franja horaria (pico de la mañana, valle, pico de la tarde, periodo nocturno) [1, 2].

La principal ventaja de los semáforos de tiempos fijos reside en su simplicidad de diseño, facilidad de implementación y previsibilidad del patrón de señales, lo que se traduce en menores requerimientos de equipamiento (no se requieren detectores de vehículos) y en un mantenimiento relativamente sencillo [2, 3]. Sin embargo, este tipo de control presenta limitaciones en entornos con variaciones significativas de la demanda, como redes urbanas con fuertes diferencias entre horas pico y valle o con eventos recurrentes que alteran los patrones de flujo, ya que el plan fijo no se adapta a la ocupación real de las aproximaciones y puede generar tiempos de verde desaprovechados o demoras excesivas en ciertas corrientes [4, 5].

Diversos estudios en ingeniería de tráfico han mostrado que, en condiciones de demanda altamente variable, los esquemas de control fijo tienden a producir mayores tiempos de retraso medio, un mayor número de paradas y colas más extensas que los sistemas actuados o adaptativos [4, 5]. Este comportamiento motiva la búsqueda de estrategias de control más flexibles, como las basadas en lógica difusa o algoritmos de optimización metaheurística, que constituyen el foco del trabajo.

2.1.3. Semáforos Accionados por el Tránsito

Los semáforos accionados por el tránsito utilizan información en tiempo real captada por detectores (por ejemplo, lazos inductivos, sensores de radar o cámaras de video) ubicados en las aproximaciones para modificar la duración de las fases dentro de límites mínimos y máximos definidos [1]. El Manual de Carreteras del Paraguay distingue entre control parcialmente actuado (al menos una aproximación bajo control por detectores) y totalmente actuado (todas las aproximaciones controladas por demanda), estableciendo criterios de diseño para la ubicación de detectores, la distancia respecto de la línea de detención y los períodos iniciales mínimos y

extensiones de verde en función de la velocidad que comprende el 85 % del tránsito en el acceso [1].

En estos sistemas, el controlador decide en cada ciclo si extiende o termina el verde de una fase atendiendo a la detección de vehículos, lo que permite reducir tiempos de verde desaprovechados en movimientos con baja demanda y asignar capacidad adicional a las aproximaciones más cargadas [3, 4]. La literatura sobre control semafórico actuado y adaptativo destaca que este tipo de sistemas mejora el desempeño frente a planes fijos, especialmente en condiciones de demanda fluctuante, al disminuir el retraso medio por vehículo y el número de paradas, y al reducir la probabilidad de colas críticas en aproximaciones dominantes [4, 5].

El uso de detectores también permite implementar estrategias avanzadas como extensiones de verde para "limpiar" colas residuales, prioridad al transporte público o prioridad a vehículos de emergencia, resultando particularmente relevante en corredores urbanos con alta variabilidad de flujos [4, 6]. Estos conceptos constituyen la base sobre la cual se articula el sistema de control inteligente propuesto en este trabajois, donde un controlador difuso tipo Mamdani y algoritmos de optimización como Particle Swarm Optimization (PSO) se integran para ajustar dinámicamente parámetros de operación semafórica en respuesta a condiciones de tráfico cambiantes.

2.1.4. Parámetros Clásicos de Diseño de Tiempos Semafóricos

En el diseño clásico de control semafórico, la configuración de una intersección se describe mediante un conjunto reducido de parámetros numéricos que determinan su desempeño operativo: la longitud de ciclo, la repartición de verdes por fase (splits), los tiempos de amarillo y todo–rojo, y los tiempos perdidos en cada etapa del ciclo [2, 3]. La longitud de ciclo C representa la duración total de una repetición completa de las fases del semáforo, mientras que los splits determinan qué fracción de ese ciclo se asigna al verde efectivo de cada movimiento o grupo de movimientos. Los tiempos de amarillo y todo–rojo se definen para advertir el fin del derecho de paso y garantizar el despeje seguro de la intersección antes de otorgar el verde a flujos conflictivos, y los tiempos perdidos agrupan intervalos en los que, por razones de seguridad o arranque, la capacidad efectiva de la intersección es menor a la teórica [2].

Estos parámetros influyen directamente en la capacidad y el retraso en cada aproximación. Para una misma demanda, ciclos más largos tienden a aumentar la capacidad por fase pero también pueden incrementar el tiempo medio de espera si los usuarios deben esperar largos intervalos de rojo, mientras que ciclos muy cortos reducen el retraso pero pueden volverse ineficientes si el tiempo perdido ocupa una proporción significativa del ciclo [3, 4]. La capacidad se relaciona con la proporción de verde asignada a cada movimiento y con la saturación de los carriles, en tanto que el retraso medio por vehículo, el número de paradas y la longitud de colas son métricas fundamentales para evaluar la calidad de servicio de un plan de tiempos [2, 7].

Métodos de diseño tradicional, como la formulación de Webster, proporcionan expresiones para estimar un ciclo "óptimo" fijo a partir de los volúmenes de diseño y de los tiempos perdidos,

buscando un compromiso entre capacidad y retraso medio [8, 7]. En la práctica, estos valores se calculan para condiciones de flujo representativas y se implementan como planes de tiempo fijo, eventualmente diferenciados por franja horaria. En contraste, el enfoque desarrollado en este trabajo utiliza un sistema de inferencia difusa tipo Mamdani y un algoritmo de optimización PSO para ajustar dinámicamente la longitud de ciclo efectiva, los splits y otros parámetros de operación en función de mediciones de tráfico en tiempo real, con el objetivo de reducir retrasos y mejorar el desempeño frente a condiciones de demanda variables [4, 5].

2.1.5. Coordinación de Semáforos y Progresión de Pelotones

En redes urbanas, los semáforos rara vez operan de manera completamente aislada; por el contrario, se busca coordinar varios equipos a lo largo de un corredor para facilitar la progresión de pelotones de vehículos y reducir el número de paradas sucesivas [2, 3]. La coordinación se logra ajustando parámetros como la longitud de ciclo común entre intersecciones, los offsets (desplazamientos temporales entre inicios de verde) y, en algunos casos, la secuencia de fases, de modo que un grupo de vehículos que parte con verde en una intersección encuentre verdes sucesivos en las siguientes, configurando la llamada "onda verde" [7, 9]. Estos esquemas buscan compatibilizar la operación de múltiples nodos de control, sacrificando a veces el óptimo local de una intersección en beneficio del desempeño global del corredor.

La coordinación introduce dependencias fuertes entre los parámetros de distintas intersecciones: cambios en la longitud de ciclo o en los splits de una intersección pueden degradar la progresión en tramos adyacentes, generando nuevas demoras o incrementando el número de paradas en el corredor [9, 3]. Por ello, el diseño de planes coordinados suele apoyarse en herramientas de simulación y optimización que consideran simultáneamente varias intersecciones, y en prácticas de monitoreo operativo que permiten ajustar offsets y ciclos en respuesta a cambios de demanda. En este contexto, la idea de un sistema de control y monitoreo distribuido, apoyado en tecnologías como BlockDAG e IPFS para registrar configuraciones y métricas de desempeño, resulta especialmente pertinente: aunque el presente trabajo se centre en la optimización a nivel de una intersección, la misma arquitectura puede escalar a corredores coordinados donde las decisiones de tiempo de verde, ciclo y offset deben gestionarse de forma conjunta y trazable.

2.1.6. Consideraciones Operativas y Métricas de Desempeño

La evaluación de un plan de tiempos semafóricos se basa en un conjunto de métricas operativas que permiten cuantificar su impacto sobre los usuarios de la vía. Entre las más utilizadas se encuentran el retraso medio por vehículo, el número de paradas, la longitud de colas y el nivel de servicio (Level of Service, LOS); en estudios recientes se añaden además indicadores de consumo de combustible y emisiones contaminantes [2, 3, 4]. El retraso medio mide la diferencia entre el tiempo de viaje real y el tiempo teórico sin interferencias, mientras que el número

de paradas y la longitud de colas caracterizan la discontinuidad del flujo y la probabilidad de bloqueo de accesos o de intersecciones adyacentes [2, 7].

El nivel de servicio resume de forma cualitativa la calidad de operación en categorías que van desde “A” (pocas demoras, operación confortable) hasta “F” (congestión severa), definidas a partir de rangos de retraso medio por vehículo y otras variables [3, 7]. Estas métricas permiten comparar diferentes esquemas de control (planes fijos, control actuado, control adaptativo) bajo condiciones de demanda similares y constituyen la base para justificar cambios en la programación de semáforos desde una perspectiva de ingeniería. En el contexto de tu implementación, ese párrafo se puede ajustar así:

En el contexto de este trabajo, el desempeño de los planes semafóricos se evalúa a partir de indicadores derivados de la salida del sistema difuso de congestión (valor numérico de congestión y su categoría lingüística), junto con variables macroscópicas como volumen por minuto, velocidad media y densidad vehicular. El controlador difuso Mamdani proporciona una medida agregada de congestión a partir de estas variables, y el algoritmo PSO ajusta los tiempos de verde para minimizar directamente dicho índice de congestión y mejorar las condiciones de flujo respecto a una asignación base de tiempos calculada mediante fórmulas clásicas de ingeniería de tráfico.

CAPÍTULO 3

VISIÓN GENERAL DEL SISTEMA

El sistema propuesto se concibe como un ecosistema modular de microservicios para el monitoreo, análisis y optimización del tráfico urbano en tiempo (casi) real, diseñado para operar sobre dispositivos de bajo costo y en entornos de ciudad inteligente. Cada módulo cumple una responsabilidad bien definida —simulación vehicular, análisis adaptativo de congestión, almacenamiento verificable y orquestación operativa— y se comunica con los demás mediante APIs REST y mensajes JSON, lo que facilita la integración con fuentes de datos heterogéneas y despliegues distribuidos. A diferencia de arquitecturas monolíticas o fuertemente centralizadas, esta organización permite escalar y evolucionar componentes de forma independiente, manteniendo al mismo tiempo una trazabilidad completa de los datos gracias al uso combinado de IPFS y contratos inteligentes sobre BlockDAG para el registro inmutable de resultados.

En términos funcionales, el flujo global de información sigue un ciclo cerrado. El módulo *traffic-sim* utiliza SUMO y el protocolo TraCI para emular redes urbanas y generar observaciones macroscópicas de tráfico en intersecciones semaforizadas, incluyendo métricas como vehículos por minuto, velocidad media, tiempo medio de circulación, densidad y distribución por tipo de vehículo. Estas observaciones, o datos provenientes de sensores reales en un despliegue futuro, se envían al módulo *traffic-control*, que actúa como puerta de entrada al sistema: valida la estructura y coherencia de los mensajes, registra metadatos en una base de datos relacional y coordina las llamadas hacia los servicios de análisis y almacenamiento. A continuación, *traffic-control* delega el análisis al módulo *traffic-sync*, que aplica un sistema de inferencia difusa tipo Mamdani para clasificar el nivel de congestión y un algoritmo PSO para proponer tiempos de verde y rojo óptimos, devolviendo una respuesta estructurada que resume las mejoras esperadas. Finalmente, tanto las observaciones originales como los resultados de optimización se envían al módulo *traffic-storage*, que almacena los artefactos en IPFS y registra los identificadores de contenido en una red BlockDAG mediante contratos inteligentes, asegurando

integridad, verificabilidad y disponibilidad distribuida de la información histórica.

Desde un punto de vista de arquitectura, esta división de responsabilidades proporciona varias ventajas. Por un lado, el acoplamiento entre captura de datos, análisis y almacenamiento es bajo: *traffic-sim* y futuros dispositivos IoT sólo necesitan conocer el endpoint unificado de *traffic-control*, mientras que los detalles de lógica difusa, PSO, IPFS y BlockDAG quedan encapsulados en módulos especializados. Por otro lado, la combinación de simulación reproducible con SUMO, optimización basada en técnicas de inteligencia computacional y persistencia verificable en infraestructuras descentralizadas permite evaluar el sistema bajo distintos escenarios de carga, al mismo tiempo que se generan evidencias auditables sobre su comportamiento, algo poco habitual en soluciones de gestión de tráfico urbano centradas únicamente en el control local o en plataformas cloud cerradas. La Figura 3.1, basada en el archivo `arquitectura.pdf`, resume gráficamente los módulos principales y las interacciones entre ellos, sirviendo como referencia visual del flujo de datos y de la integración con los servicios descentralizados de almacenamiento y verificación.

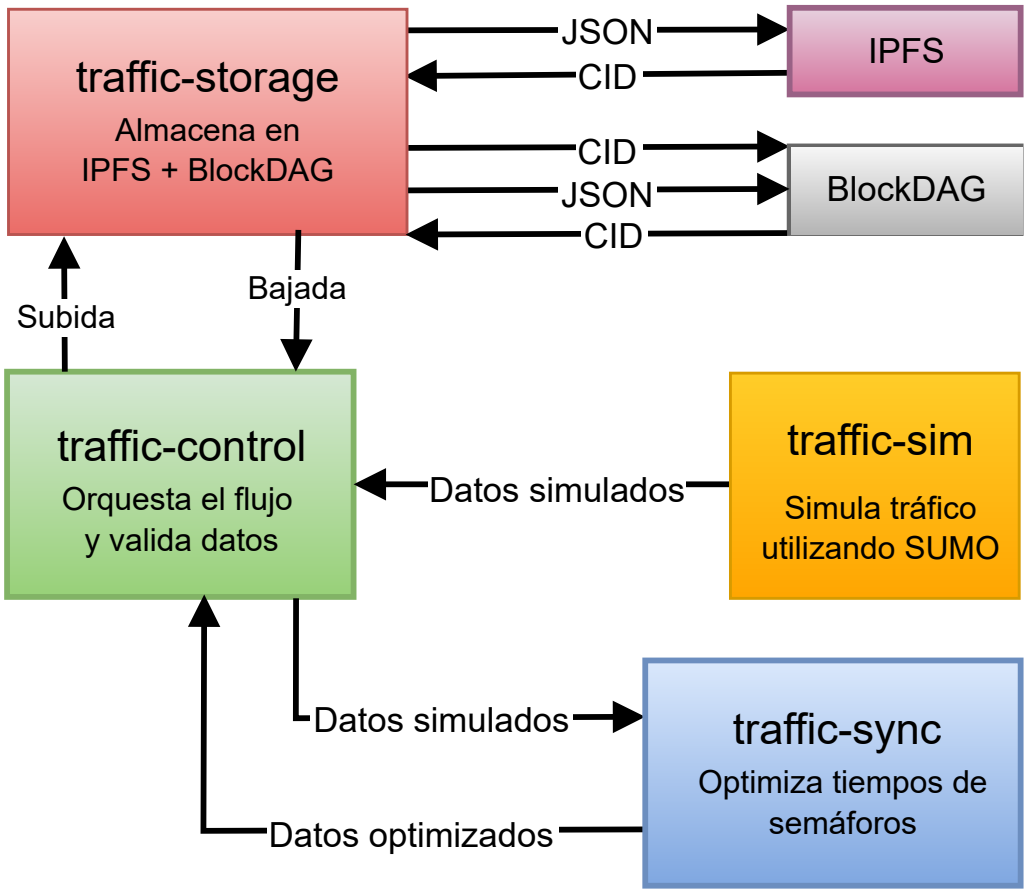

Figura 3.1: Arquitectura general del sistema, mostrando los módulos *traffic-sim*, *traffic-control*, *traffic-sync* y *traffic-storage*, así como su integración con IPFS y BlockDAG.

3.1. Módulo de Simulación de Tráfico

El módulo `traffic-sim` implementa un entorno de simulación de tráfico urbano basado en SUMO (Simulation of Urban MObility), utilizado para evaluar el comportamiento del sistema propuesto bajo escenarios controlados y reproducibles. SUMO es un simulador microscópico de tráfico ampliamente adoptado en la comunidad científica, que permite modelar redes viales, rutas de vehículos y programas semafóricos, y ejecutar simulaciones paso a paso mediante una interfaz de control remoto (`traci`) [10]. En este trabajo, `traffic-sim` se integra con el servicio de optimización `traffic-sync` para cerrar el ciclo "simulación–detección–optimización–aplicación" sobre redes generadas a partir de datos reales o escenarios de prueba.

3.1.1. Arquitectura del Módulo

La estructura de `traffic-sim` se organiza en torno a un orquestador de simulación y un conjunto de componentes especializados para detección, control y comunicación externa. El archivo `simulation_orchestrator.py` encapsula la lógica principal: carga la configuración de SUMO (archivos `*.sumocfg`, `edges.edg.xml`, `nodes.nod.xml`, `routes.rou.xml` y `traffic_lights.add.xml`), inicializa la simulación mediante `traci` y gestiona el avance de la simulación en pasos discretos [10]. El script `run_simulation.py` actúa como punto de entrada, recibiendo como argumento un archivo `ZIP` de escenario (generado, por ejemplo, mediante la herramienta auxiliar de construcción de mapas) y ejecutando la simulación en modo *headless* o con interfaz gráfica.

Los detectores de cuellos de botella se implementan en el paquete `detectors`, particularmente en `bottleneck_detector.py`, donde se monitorean métricas como densidad de vehículos, velocidad promedio y longitud de cola por tramo o aproximación. Los umbrales y parámetros de detección (por ejemplo, densidad mínima, velocidad máxima para considerar congestión, intervalo entre detecciones, duración mínima del evento) se centralizan en `config.py`, lo que permite ajustar la sensibilidad del sistema sin modificar la lógica de detección. El controlador de semáforos se encuentra en `controllers/traffic_light_controller.py` y expone métodos para actualizar dinámicamente los tiempos de verde y de ciclo de los semáforos modelados en SUMO, aplicando las optimizaciones recibidas desde el servicio `traffic-sync`.

El módulo `services/traffic_control_client.py` implementa el cliente HTTP que se comunica con la API de `traffic-sync`: cuando se detecta un cuello de botella, este cliente construye un payload con métricas agregadas de tráfico para la intersección crítica y lo envía al endpoint correspondiente (por ejemplo, `/evaluate`). Las utilidades en `services/metrics_calculator.py` y `utils/metrics_validator.py` calculan y validan las métricas macroscópicas a partir de los datos de SUMO (por ejemplo, viajes completados, tiempos de viaje, velocidades y colas), garantizando que la información enviada a `traffic-sync` siga el esquema esperado por el sistema difuso y el optimizador PSO.

La carpeta `visualization` contiene herramientas para análisis posterior de resultados: par-

sers de archivos de salida de SUMO (`tripinfo.xml`, `summary.xml`, `fcd.xml`), funciones para generar gráficos de distribución de tiempos de viaje, evolución temporal de congestión y comparaciones A/B entre diferentes configuraciones de semáforos, así como envoltorios sobre herramientas nativas de SUMO. Estas utilidades permiten cuantificar el impacto de las estrategias de control sobre métricas clave como tiempo de viaje, tiempos de espera y niveles de congestión.

3.1.2. Flujo de Integración

El flujo de integración entre `traffic-sim` y `traffic-sync` sigue una secuencia cíclica que replica el comportamiento de un sistema de control de tráfico en línea:

1. **Ejecución de la simulación**: el orquestador inicia SUMO con el escenario suministrado y avanza la simulación paso a paso, según las rutas y patrones de demanda definidos en los archivos de configuración.

2. **Detección de cuellos de botella**: en intervalos configurables, el detector analiza para cada aproximación variables como densidad, velocidad y longitud de cola, y decide si existe un cuello de botella según los umbrales establecidos en `BOTTLENECK_CONFIG` (`config.py`).

3. **Construcción de métricas de tráfico**: cuando se confirma un cuello de botella en una intersección con semáforo, se calcula un conjunto de métricas agregadas para esa localización (vehículos por minuto, velocidad media en km/h, tiempo medio de circulación, densidad y estadísticos por tipo de vehículo), que se empaquetan en un payload JSON siguiendo el formato consumido por `traffic-sync`.

4. **Comunicación con** `traffic-sync`: el cliente `traffic_control_client.py` envía el payload al servicio `traffic-sync` mediante una petición HTTP (por ejemplo, al endpoint `/evaluate`), y espera la respuesta con los tiempos de verde optimizados y el impacto estimado sobre la congestión.

5. **Aplicación de optimizaciones**: el controlador de semáforos actualiza, a través de `traci`, los tiempos de verde y rojo de los programas semafóricos correspondientes en la simulación SUMO, utilizando los valores recibidos desde `traffic-sync`.

6. **Continuación de la simulación**: la simulación se reanuda con los nuevos parámetros de control y se repite el proceso de detección y optimización mientras existan vehículos en red o hasta alcanzar el tiempo límite definido.

De esta manera, `traffic-sim` se comporta como un "laboratorio virtual" donde se puede evaluar en bucle el desempeño del controlador difuso Mamdani y del optimizador PSO integrados en `traffic-sync`, utilizando escenarios sintéticos o construidos a partir de mapas reales. La salida de herramientas auxiliares de construcción de escenarios (por ejemplo, generadores de redes y rutas compatibles con SUMO) sirve como entrada directa para `traffic-sim`, lo que facilita la creación sistemática de casos de prueba. Los resultados de las simulaciones, incluyendo

métricas temporales y distribuciones de tiempos de viaje, se analizan posteriormente mediante las utilidades de visualización para cuantificar de forma objetiva el impacto de las políticas de control propuestas.

3.2. Módulo de Almacenamiento de Datos

El módulo `traffic-storage` actúa como capa de persistencia verificable del sistema, recibiendo métricas de tráfico y resultados de optimización, almacenando los datos en un repositorio distribuido y registrando huellas inmutables en infraestructuras descentralizadas. En este contexto, blockchain, BlockDAG e IPFS se utilizan de forma complementaria: blockchain y BlockDAG como ledgers inmutables para trazabilidad, e IPFS como almacén de archivos orientado a contenido [11, 12, 13].

3.2.1. Blockchain

Blockchain puede describirse como un libro mayor distribuido donde las transacciones se agrupan en bloques enlazados criptográficamente en una cadena lineal, de modo que cada bloque referencia al anterior mediante un hash, proporcionando inmutabilidad y resistencia a manipulaciones retrospectivas [11, 14]. Los nodos de la red ejecutan un mecanismo de consenso (como Proof-of-Work o Proof-of-Stake) para acordar qué bloques se consideran válidos, garantizando que todos mantengan una vista consistente del historial de transacciones [15, 16].

En este trabajo, blockchain se utiliza para registrar metadatos de almacenamiento de tráfico de forma inmutable, actuando como capa de "prueba de existencia" para datos que residen físicamente fuera de la cadena. El contrato inteligente `TrafficStorage` (ubicado en el directorio de contratos del proyecto) expone operaciones de alto nivel para registrar eventos de almacenamiento: cada llamada almacena, al menos, un identificador de semáforo, una marca de tiempo y un identificador de contenido (CID de IPFS), junto con el tipo de dato al que hace referencia. Este contrato se despliega y prueba utilizando un entorno de desarrollo estándar (por ejemplo, Hardhat), con scripts de despliegue y pruebas automatizadas que verifican su funcionamiento básico.

El módulo `traffic-storage` ofrece una API HTTP que recibe cargas de datos en un formato unificado (versión 2.0), generadas por otros componentes del sistema (como los módulos de PSO y lógica difusa). Cuando llega una nueva medición o reporte, la API valida el payload, delega la persistencia de contenido a IPFS y, una vez obtenido el CID, invoca al contrato `TrafficStorage` para registrar ese CID junto con los metadatos relevantes en la blockchain. De este modo, los algoritmos de optimización y los módulos de monitoreo solo interactúan con una interfaz REST, mientras que los detalles de interacción on-chain quedan encapsulados en `traffic-storage`.

3.2.2. BlockDAG

BlockDAG (Block Directed Acyclic Graph) generaliza la estructura lineal de blockchain permitiendo que cada bloque referencie a múltiples bloques padres anteriores, formando un grafo dirigido acíclico en lugar de una cadena única [17, 13]. Esta arquitectura permite que bloques creados en paralelo se integren en el consenso sin ser descartados como huérfanos, mejorando el aprovechamiento del ancho de banda de la red y posibilitando mayores tasas de transacciones y menores latencias de confirmación que las típicamente alcanzables en cadenas lineales [17].

En el contexto de este sistema, BlockDAG se considera como infraestructura de registro distribuido complementaria a la blockchain tradicional, pensada para escenarios de mayor escala donde múltiples intersecciones reportan datos con alta frecuencia. El módulo `traffic-storage` incorpora una capa de abstracción que permite, según la configuración, registrar eventos de almacenamiento tanto en un contrato EVM como en una red BlockDAG compatible, utilizando un cliente dedicado que construye y envía mensajes de registro a un nodo remoto. Estos mensajes contienen, de forma análoga a la variante blockchain, el identificador de semáforo, la marca de tiempo y el hash de contenido (CID de IPFS u otro identificador criptográfico).

La lógica de alto nivel en `traffic-storage` se mantiene independiente de la tecnología subyacente: la API recibe métricas y resultados de optimización, delega la persistencia de archivos a IPFS y, posteriormente, invoca una interfaz de "registro de evidencia" que puede mapearse a blockchain, BlockDAG o a un esquema dual. Esta separación permite que, a medida que se exploren o adopten redes DAG de mayor throughput para despliegues reales en entornos de ciudad inteligente, el resto del sistema (clasificación de congestión, PSO, paneles de monitoreo) continúe interactuando con un mismo punto de entrada consistente.

3.2.3. IPFS

El InterPlanetary File System (IPFS) es un sistema de archivos distribuido peer-to-peer que utiliza direccionamiento por contenido: cada objeto se identifica por un Content Identifier (CID) que codifica un hash del contenido más metadatos mínimos sobre cómo interpretarlo [12]. El mismo archivo siempre produce el mismo CID, independientemente del nodo que lo procese, lo que permite verificar integridad de manera determinista y realizar deduplicación automática a escala de red [18].

En este trabajo, IPFS se emplea como almacén de datos off-chain para los artefactos generados y consumidos por el sistema: mediciones agregadas de tráfico por clúster, indicadores de congestión difusa, configuraciones optimizadas de tiempos semafóricos y reportes experimentales. El módulo `traffic-storage` incluye un cliente específico para interactuar con un nodo Kubo local o con un servicio compatible, ofreciendo operaciones sencillas para subir y recuperar objetos (por ejemplo, `upload_json` y `download_by_cid`). El flujo básico consiste en que la API recibe un payload de datos, lo serializa (por ejemplo, como JSON), lo envía al nodo IPFS y

obtiene de vuelta un CID.

Ese CID se utiliza a continuación como enlace entre IPFS y las capas de consenso: se registra en blockchain o en BlockDAG mediante las interfaces descritas anteriormente, de modo que cualquier módulo que necesite auditar o reanalizar un conjunto de datos solo requiera conocer el CID y el identificador de semáforo asociado. Desde la perspectiva del resto del sistema, IPFS y las tecnologías de registro inmutable quedan encapsuladas detrás de `traffic-storage`: los módulos de simulación, clasificación difusa y PSO interactúan únicamente con la API REST, mientras que las garantías de verificabilidad, integridad y persistencia se logran combinando direccionamiento por contenido en IPFS con registros inmutables en blockchain y/o BlockDAG.

3.3. Módulo de Optimización de Tiempos Semafóricos

El módulo `traffic-sync` implementa el servicio de optimización de tráfico del sistema, exponiendo una API que recibe mediciones macroscópicas de tráfico por semáforo y devuelve configuraciones de tiempos optimizadas. Para ello combina dos componentes principales: un sistema de inferencia difusa tipo Mamdani, que evalúa el nivel de congestión a partir de variables de flujo, velocidad y densidad; y un algoritmo de optimización Particle Swarm Optimization (PSO), que ajusta dinámicamente los tiempos de verde con el objetivo de reducir la congestión estimada [19, 20, 21].

3.3.1. Lógica Difusa Mamdani

La lógica difusa proporciona un marco formal para representar razonamiento humano impreciso mediante variables lingüísticas y conjuntos difusos con grados de pertenencia entre 0 y 1 [20, 22]. En un sistema de inferencia Mamdani, las reglas tienen la forma "SI condición ENTONCES acción", donde tanto antecedentes como consecuentes se modelan como conjuntos difusos, y la salida final se obtiene tras un proceso de agregación y defuzzificación [19, 23]. Este enfoque resulta especialmente adecuado para describir estados de tráfico como "flujo libre", "denso" o "congestionado" a partir de variables macroscópicas continuas [24, 25].

En `traffic-sync`, el sistema de inferencia difusa se implementa en el paquete `modules.fuzzy`, que está compuesto principalmente por los módulos `system.py` y `evaluation.py`. El módulo `system.py` define las variables lingüísticas de entrada (por ejemplo, vehículos por minuto, velocidad media, tiempo medio de circulación, densidad) y la variable de salida asociada al nivel de congestión, junto con sus funciones de membresía triangulares y trapezoidales calibradas en base a datos de tráfico y criterios de ingeniería [23, 24]. La base de reglas Mamdani captura relaciones cualitativas del tipo "SI flujo es alto Y velocidad es baja ENTONCES congestión es severa", que reflejan tanto conocimiento experto como patrones observados en los datos [25].

El módulo `evaluation.py` expone funciones de alto nivel, como `run_test_cases`, que reciben vectores de métricas de tráfico en formato dict y devuelven, para cada sensor o semáforo, un valor numérico de congestión y su categoría lingüística asociada. Estas funciones se utilizan

en el pipeline principal definido en `api/server.py`: el endpoint `/evaluate` recibe un lote de sensores, construye las entradas para el sistema difuso, ejecuta la evaluación Mamdani sobre cada uno y genera un conjunto de salidas que se utilizarán como insumo para la etapa de optimización PSO. De esta forma, la lógica difusa actúa como módulo de evaluación de desempeño que transforma mediciones de tráfico en un índice de congestión interpretable, que el optimizador buscará reducir.

3.3.2. Optimización de Enjambre de Partículas (PSO)

Optimización de Enjambre de Partículas (PSO) es un algoritmo de optimización metaheurística inspirado en el comportamiento colectivo de bandadas de aves y bancos de peces, en el que un conjunto de partículas explora el espacio de búsqueda guiado por su mejor experiencia individual y la mejor experiencia global del grupo [21, 26]. Cada partícula representa una solución candidata (en este caso, una posible asignación de tiempo de verde para un grupo de semáforos) y actualiza su posición combinando tres componentes: inercia, atracción hacia su mejor posición personal y atracción hacia la mejor posición global conocida, ponderadas por parámetros que controlan el equilibrio entre exploración y explotación [27, 28].

En el módulo `traffic-sync`, la lógica de PSO se implementa en el paquete `modules.pso`, que incluye `fitness.py` y `optimization.py`. El archivo `optimization.py` define la función `pso`, que recibe un `DataFrame` de `pandas` con información agregada de tráfico por grupo (cluster) y aplica PSO para determinar el tiempo de verde óptimo por grupo. Cada partícula codifica un tiempo de verde propuesto para el ciclo semafórico, inicializado alrededor de un valor base calculado mediante fórmulas clásicas de ingeniería de tráfico (función `calculate_green_time` en `fitness.py`), y restringido por límites mínimos y máximos coherentes con la normativa [8, 2]. El algoritmo utiliza configuraciones típicas de PSO (número de partículas, peso de inercia, componentes cognitiva y social, límites de velocidad) y añade mecanismos prácticos como parada temprana y reinicios parciales para evitar estancamiento en óptimos locales pobres.

La función de aptitud (`fitness_function`, definida en `fitness.py`) evalúa cada propuesta de tiempo de verde combinando la lógica difusa y las métricas de tráfico: dado un tiempo de verde candidato, recalcula métricas derivadas (como volumen efectivo, velocidad y densidad ajustadas) y vuelve a evaluar el nivel de congestión mediante el sistema Mamdani, obteniendo un valor numérico que se utiliza como medida a minimizar [4, 5]. De este modo, PSO busca explícitamente reducir el índice de congestión proporcionado por el módulo difuso, en lugar de optimizar una función analítica cerrada.

En el pipeline de `traffic-sync`, orquestado por la función `run_pipeline` en `api/server.py`, el flujo es el siguiente: (i) el servicio recibe un lote de sensores en la API `/evaluate`; (ii) las mediciones se pasan al módulo difuso para obtener un nivel de congestión por sensor; (iii) se agrupan sensores con características similares (agrupamiento jerárquico), produciendo un conjunto de grupos de tráfico; (iv) se construye un `DataFrame` por grupo con métricas agregadas (por ejemplo, vehículos por minuto, velocidad media, densidad media, categoría de congestión);

y (v) se aplica la función `pso` sobre estos grupos para obtener tiempos de verde optimizados por grupo. La respuesta de la API se construye como un lote de optimizaciones, donde cada entrada incluye el tiempo de verde recomendado, el tiempo de rojo derivado, el nivel de congestión original y optimizado, y las categorías lingüísticas correspondientes, preparados para su consumo posterior por el módulo de control de semáforos y por el módulo `traffic-storage`.

En conjunto, la integración entre lógica difusa Mamdani y PSO en `traffic-sync` permite que la optimización de tiempos semafóricos se base en una medida de congestión que captura adecuadamente la percepción y el comportamiento del tráfico urbano, mientras que el metaheurístico proporciona un mecanismo flexible para ajustar los parámetros de control en escenarios donde la función objetivo no es derivable y debe evaluarse a través de simulación o reglas de inferencia [23, 29].

3.4. Módulo de Control y Orquestación

El módulo `traffic-control` actúa como orquestador central del sistema distribuido de gestión de tráfico: recibe observaciones vehiculares desde simulaciones o despliegues reales, valida y registra los datos, coordina el envío de información hacia los servicios de optimización (`traffic-sync`) y almacenamiento distribuido (`traffic-storage`), y expone una API REST unificada para clientes externos. Su objetivo es desacoplar los productores de datos (por ejemplo, `traffic-sim`) de los módulos especializados de optimización y persistencia, garantizando un flujo coherente y trazable de información a través del sistema.

3.4.1. Arquitectura del Módulo

La estructura de `traffic-control` se organiza en capas bien definidas: configuración, API, modelos de datos, servicios de dominio y utilidades de soporte. El archivo `api/server.py` implementa el servicio web principal (basado en FastAPI), incluyendo el endpoint `/process`, que procesa observaciones vehiculares completas, y endpoints auxiliares como `/healthcheck` y operaciones sobre metadatos. El módulo `config/settings.py` centraliza parámetros de configuración, como las URLs de los servicios `traffic-storage` y `traffic-sync`, la cadena de conexión a la base de datos y los niveles de logging; `config/logging_config.py` define la configuración de logs para toda la aplicación.

La capa de modelos se implementa en el paquete `models`, que incluye esquemas Pydantic para validar la estructura de peticiones y respuestas (`schemas/data_schema.py`, `schemas/optimization_s` `schemas/download_schema.py`) y modelos de respuesta estandarizados en `response_models.py`. El módulo `validator.py` encapsula la validación semántica de los datos: verifica versión del payload, formato de marcas de tiempo, tipo de mensaje (`data` u `optimization`), límites de tamaño de lote (1–10 sensores) e integridad de campos como `traffic_light_id`, métricas y estadísticas vehiculares. Cualquier inconsistencia o violación de estos contratos se transforma en una respuesta de error coherente mediante el manejador centralizado de errores

utils/error_handler.py.

La persistencia de metadatos se gestiona en el paquete database, con db.py como punto de entrada a la conexión SQLAlchemy y metadata_model.py definiendo el esquema de almacenamiento: cada registro guarda, como mínimo, el identificador del semáforo (tls_id), la marca de tiempo, el tipo de dato (por ejemplo, observación de tráfico u optimización) y referencias a los identificadores de contenido almacenados en traffic-storage. La capa de servicios (services/database_service.py) proporciona operaciones de alto nivel para crear y consultar registros, que se exponen a través de endpoints de metadatos (por ejemplo, /metadata/traffic-light/{id}, /metadata/recent, /metadata/stats).

3.4.2. Flujo de Procesamiento

El endpoint POST /process es el punto de entrada principal para observaciones de tráfico, tanto provenientes de traffic-sim como de otros clientes. El flujo de procesamiento sigue una secuencia de pasos bien definida:

1. **Recepción y validación**: la API recibe un payload en formato unificado (versión 2.0), que puede contener uno o varios sensores asociados a un mismo traffic_light_id. El módulo models/validator.py verifica que el mensaje cumpla con la estructura esperada (campos obligatorios en metrics y vehicle_stats, rango de tamaño de lote, formatos de fecha, tipo de mensaje), generando errores explícitos en caso de inconsistencias.

2. **Preprocesamiento de datos**: el servicio services/data_processor.py puede aplicar transformaciones ligeras sobre los datos (normalización de campos, ajuste de formatos, enriquecimiento con información temporal) antes de enviarlos a los módulos externos. Esta capa asegura que, aunque distintos productores tengan pequeñas variaciones, el sistema trabaje con un formato interno coherente.

3. **Persistencia de metadatos**: mediante services/database_service.py, se registra en la base de datos un resumen del evento (identificador de semáforo, marca de tiempo, tipo, estado del procesamiento), lo que permite consultas posteriores sobre qué datos fueron recibidos, cuándo y con qué resultado. Esta base sirve como índice local complementario a los registros en blockchain/BlockDAG gestionados por traffic-storage.

4. **Interacción con** traffic-storage: el proxy de almacenamiento services/storage_proxy.py construye una petición hacia la API de traffic-storage, subiendo el payload o referencias derivadas y recibiendo de vuelta un identificador (por ejemplo, un CID de IPFS u otro ID interno). Esta información se asocia al registro de metadatos para mantener la trazabilidad entre las observaciones y su representación en el almacén distribuido.

5. **Interacción con** traffic-sync: el proxy de sincronización services/sync_proxy.py envía los datos de tráfico a traffic-sync mediante el endpoint /evaluate, en formato compatible con el sistema difuso y el optimizador PSO. Para lotes de sensores, la

función `send_batch_for_optimization` reempaqueta el lote en el formato esperado por `traffic-sync` (incluyendo el campo `traffic_light_id` de referencia) y recibe un conjunto de optimizaciones, una por grupo detectado.

6. **Respuesta al cliente**: finalmente, `services/process_service.py` compone una respuesta estandarizada utilizando los modelos definidos en `models/response_models.py`, indicando el estado del procesamiento (por ejemplo, `success` o `error`), un mensaje descriptivo y, cuando corresponda, datos derivados como tiempos de verde optimizados e impacto estimado sobre la congestión. Esta respuesta se devuelve al cliente original (por ejemplo, `traffic-sim`), que puede aplicar las decisiones de control correspondientes.

Durante todo el flujo, el módulo `utils/error_handler.py` proporciona mecanismos centralizados para capturar y clasificar errores en distintas categorías (validación, almacenamiento, sincronización, base de datos, errores genéricos), generando respuestas HTTP adecuadas y registros de log con contexto detallado para facilitar la depuración.

3.4.3. Interacción con los Módulos

Desde el punto de vista del sistema global, `traffic-control` se posiciona como el módulo de orquestación que articula la interacción entre simulación, optimización y almacenamiento distribuido:

- **Relación con** `traffic-sim`: el módulo de simulación basado en SUMO detecta cuellos de botella y, cuando identifica una situación crítica en una intersección, construye un payload con métricas agregadas de tráfico y lo envía a `traffic-control` mediante el endpoint `/process`. De este modo, `traffic-sim` no necesita conocer detalles sobre `traffic-sync` ni `traffic-storage`; sólo interactúa con un servicio central que gestiona validación, almacenamiento y consulta de optimizaciones.

- **Relación con** `traffic-sync`: `traffic-control` actúa como cliente de `traffic-sync` a través del proxy `SyncProxy` en `services/sync_proxy.py`. Para cada observación (o lote) recibida, reenvía los datos relevantes a `traffic-sync`, que ejecuta el pipeline de lógica difusa y PSO para producir tiempos de verde optimizados y categorías de congestión. La respuesta de `traffic-sync` se integra en la respuesta de `traffic-control`, permitiendo que los consumidores (por ejemplo, `traffic-sim`) apliquen las configuraciones recomendadas.

- **Relación con** `traffic-storage`: mediante `services/storage_proxy.py`, `traffic-control` sube y recupera datos hacia/desde el módulo `traffic-storage`, que a su vez utiliza IPFS, blockchain y BlockDAG como backend de almacenamiento y registro inmutable. `traffic-control` registra en su base de datos los identificadores de contenido (por ejemplo, CIDs) devueltos por `traffic-storage`, de modo que las consultas de metadatos

pueden resolverse localmente y, cuando es necesario, seguir la cadena de referencias hasta los datos almacenados en el sistema distribuido.

Esta arquitectura modular permite que cada componente se especialice en su responsabilidad principal: `traffic-sim` en la generación de escenarios y detección de cuellos de botella; `traffic-sync` en la evaluación de congestión y optimización de tiempos semafóricos mediante lógica difusa y PSO; `traffic-storage` en la persistencia verificable de datos; y `traffic-control` en la validación, coordinación y exposición de un punto de entrada unificado al sistema. En conjunto, estos módulos forman una plataforma escalable y descentralizada para la gestión inteligente de tráfico urbano.

CAPÍTULO 4

capitulo

CAPÍTULO 5

CONCLUSIÓN

REFERENCIAS

[1] Ministerio de Obras Públicas y Comunicaciones. *Manual de Carreteras del Paraguay. Unidad 3: Diseño de Carreteras. Volumen 3.3: Señalización y Obras Complementarias*. Sección 3.3.2.7: Semáforos. Asunción, Paraguay: MOPC, 2019.

[2] Nicholas J. Garber y Lester A. Hoel. *Traffic and Highway Engineering*. 6th. Boston: Cengage Learning, 2019.

[3] Roger P. Roess, Elena S. Prassas y William R. McShane. *Traffic Engineering*. 5th. Upper Saddle River: Pearson, 2019.

[4] Markos Papageorgiou, Christina Diakaki, Vaya Dinopoulou, Apostolos Kotsialos y Yibing Wang. «Review of Road Traffic Control Strategies». En: *Proceedings of the IEEE* 91.12 (2003), págs. 2043-2067. DOI: 10.1109/JPROC.2003.819610.

[5] Aleksandar Stevanovic, Jelena Stevanovic, Kai Zhang y Stuart Batterman. «Optimizing Traffic Control to Reduce Fuel Consumption and Vehicular Emissions: Integrated Approach with VISSIM, CMEM, and VISGAOST». En: *Transportation Research Record* 2128 (2010). Incluye comparación de desempeño entre planes fijos y controladores adaptativos, págs. 105-113. DOI: 10.3141/2128-12.

[6] Nathan H. Gartner, Farhad J. Pooran y Carleton M. Andrews. *Optimized Policies for Adaptive Control: A Case Study for Arterial Traffic*. Capítulos sobre control actuado y coordinado. Boston: Springer, 2001.

[7] Federal Highway Administration. *Traffic Signal Timing Manual*. https://ops.fhwa.dot.gov/publications/fhwahop08024/. FHWA-HOP-08-024, consultado: 2026-01-XX. 2008.

[8] F. V. Webster. «Traffic Signal Settings». En: *Road Research Technical Paper* 39 (1958). Clásico método de diseño de tiempos semafóricos.

[9] S. Erdogan, Bashar Al-Omari y Hesham Rakha. «Signal Timing Optimization for Coordinated Arterials: A Review of Progression-Based Methods». En: *Journal of Transportation Engineering* 139.4 (2013), págs. 358-370. DOI: 10.1061/(ASCE)TE.1943-5436.0000506.

[10] DLR Institute of Transportation Systems. *SUMO User Documentation*. Consultado: 2025-12-XX. Eclipse SUMO. 2025. URL: https://sumo.dlr.de/docs/.

[11] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Consultado: 2025-01-XX. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[12] Juan Benet. *IPFS - Content Addressed, Versioned, P2P File System*. Whitepaper fundacional de IPFS. 2014. arXiv: 1407.3561 [cs.DC]. URL: https://arxiv.org/abs/1407.3561.

[13] Yonatan Sompolinsky y Aviv Zohar. «PHANTOM and GHOSTDAG: A Scalable Generalization of Nakamoto Consensus». En: *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies (AFT '21)*. New York, NY, USA: ACM, 2021, págs. 57-70. DOI: 10.1145/3479722.3480990.

[14] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller y Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016. ISBN: 978-0-691-17169-2.

[15] Vitalik Buterin. «Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform». En: (2014). Whitepaper, Consultado: 2025-01-XX. URL: https://ethereum.org/en/whitepaper/.

[16] Juan Garay, Aggelos Kiayias y Nikos Leonardos. «The Bitcoin Backbone Protocol: Analysis and Applications». En: *Proceedings of the 19th International Conference on Principles of Distributed Systems (OPODIS 2015)*. Consultado: 2025-01-XX. 2015, págs. 1-15. URL: https://opodis2015.irisa.fr/files/2016/01/Garay_OPODIS_2015.pdf.

[17] Qiang Wang, Jiajing Yu, Shuo Chen y Yong Xiang. «SoK: DAG-based Blockchain Systems». En: *ACM Computing Surveys* 55.12 (2023), Article 256. DOI: 10.1145/3576899.

[18] IPFS Project. *IPFS Architecture*. Especificación oficial de arquitectura IPFS, define formalmente nodeID := multihash(publicKey) y esquema de identidad basado en PKI. 2024. URL: https://github.com/ipfs/specs/blob/main/ARCHITECTURE.md.

[19] Ebrahim H. Mamdani y Sedrak Assilian. «An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller». En: *International Journal of Man-Machine Studies* 7.1 (1975), págs. 1-13. DOI: 10.1016/S0020-7373(75)80002-2.

[20] Lotfi A. Zadeh. «Fuzzy Sets». En: *Information and Control* 8.3 (1965), págs. 338-353. DOI: 10.1016/S0019-9958(65)90241-X.

[21] James Kennedy y Russell Eberhart. «Particle Swarm Optimization». En: *Proceedings of the IEEE International Conference on Neural Networks*. Vol. 4. IEEE, 1995, págs. 1942-1948. DOI: 10.1109/ICNN.1995.488968.

[22] Lotfi A. Zadeh. «Similarity Relations and Fuzzy Orderings». En: *Information Sciences* 3.2 (1971), págs. 177-200. DOI: 10.1016/S0020-0255(71)80005-1.

[23] Timothy J. Ross. *Fuzzy Logic with Engineering Applications*. 3rd. Chichester, UK: John Wiley & Sons, 2010.

[24] Gökhan Erdinç. «Application of a Mamdani-Based Fuzzy Traffic State Identifier». Tesis sobre identificación de estado de tráfico usando lógica difusa Mamdani. Thesis. Università "La Sapienza"di Roma, 2023. URL: https://iris.uniroma1.it/retrieve/d51daa72-738b-40a8-b0f8-2f25473ebf1b/Erdin%C3%A7_Application-Mamdani-based_2023.pdf.

[25] Rizka Zuliani Musriroh. «Application of Mamdani Method on Fuzzy Logic to Decision of Traffic Light System». En: *Proceedings of International Conference on Informatics and Computational Sciences*. SciTePress, 2020, pág. 85200. DOI: 10.5220/0008520000000000.

[26] Russell C. Eberhart y James Kennedy. «A New Optimizer Using Particle Swarm Theory». En: *Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS '95)*. IEEE, 1995, págs. 39-43. DOI: 10.1109/MHS.1995.494215.

[27] Maurice Clerc y James Kennedy. «The Particle Swarm – Explosion, Stability, and Convergence in a Multidimensional Complex Space». En: *IEEE Transactions on Evolutionary Computation* 6.1 (2002), págs. 58-73. DOI: 10.1109/4235.985692.

[28] Yuhui Shi y Russell Eberhart. «Parameter Selection in Particle Swarm Optimization». En: (1998). Proposed inertia weight adaptation, págs. 591-600.

[29] Riccardo Poli, James Kennedy y Tim Blackwell. «Particle Swarm Optimization: An Overview». En: *Swarm Intelligence* 1.1 (2007), págs. 33-57. DOI: 10.1007/s11721-007-0002-0.

APÉNDICE

A.1. A