Diseño y evaluación de un sistema de comunicación V2X mediante simulación con SUMO y OpenC2X

Saúl Ibáñez Cerro

Máster Universitario en Ingeniería de Telecomunicación Telemática

Tutor de TF
Antoni Morell Pérez
Profesor/a responsable de la asignatura
Pere Tuset Peiró

Enero 2026

Ficha del Trabajo Final

Título del trabajo:	Diseño y evaluación de un sistema de comunicación V2X mediante simulación con SUMO y OpenC2X
Nombre del autor/a:	Saúl Ibáñez Cerro
Nombre del Tutor/a de TF:	Antoni Morell Pérez
Nombre del/de la PRA:	Pere Tuset Peiró
Fecha de entrega:	01/2026
Titulación o programa:	Máster Universitario en Ingeniería de Telecomunicación
Área del Trabajo Final:	Telemática
Idioma del trabajo:	Castellano
Palabras clave:	V2X, SUMO, OpenC2X

Resumen del Trabajo

El objetivo de este Trabajo Final de Máster es diseñar, implementar y evaluar un entorno de simulación que permita analizar el comportamiento de las comunicaciones V2X mediante la integración de SUMO → OpenC2X. Para ello, se ha desarrollado un sistema capaz de sincronizar en tiempo real la movilidad generada por SUMO con los módulos de comunicación de OpenC2X, utilizando la API TraCI como interfaz principal.
La primera fase del proyecto consiste en la migración, adaptación y validación de OpenC2X en un entorno moderno (Ubuntu 24.04), resolviendo incompatibilidades en bibliotecas como Boost, gpsd, uci y libubox. Posteriormente, se desarrolla el módulo SumoInterface, encargado de establecer la comunicación SUMO con OpenC2X y de transformar las coordenadas internas de SUMO en datos GPS válidos.
Se valida el sistema con escenarios crecientes (10, 20, 50 y 100 vehículos), analizando métricas como la latencia interna, la tasa de entrega de mensajes CAM y los intervalos temporales de generación. Los resultados muestran que el pipeline SUMO → GPS → CAM → LDM mantiene una latencia estable y baja (1-2 ms) con DCC activado. La desactivación del DCC provoca una degradación significativa, confirmando su importancia como mecanismo de control de congestión.
El sistema ha sido desarrollado sobre una plataforma reproducible, extensible y basada en software libre.

Abstract

The objective of this Master's Thesis is to design, implement and evaluate a simulation environment that allows the analysis of V2X communication behavior through the integration of SUMO → OpenC2X. To achieve this, a system has been developed capable of synchronizing in real time the mobility generated by SUMO with the communication modules of OpenC2X, using the TraCI API as the main interface.

The first phase of the project focuses on the migration, adaptation and validation of OpenC2X in a modern environment (Ubuntu 24.04), addressing compatibility issues in libraries such as Boost, gpsd, uci and libubox. Subsequently, the `SumoInterface` module is developed, responsible for establishing the communication link between SUMO and OpenC2X and for converting SUMO's internal coordinates into valid GPS data.

The system is validated under scenarios of increasing scale (10, 20, 50 and 100 vehicles), analyzing metrics such as internal latency, CAM message delivery ratio and inter-generation intervals. The results show that the SUMO → GPS → CAM → LDM pipeline maintains stable and low latency (1-2 ms) when DCC is enabled. Disabling DCC leads to significant degradation, confirming its importance as a congestion-control mechanism.

The system has been developed on a reproducible, extensible and fully open-source software platform.

Índice

Índice de figuras

Índice de cuadros

1 Introducción

1.1 Contexto y justificación del trabajo

Las tecnologías *V2X (Vehicle-to-Everything)* representan uno de los pilares fundamentales de la movilidad conectada y de los sistemas de transporte cooperativos. Su capacidad para mejorar la seguridad vial, optimizar el tráfico y habilitar nuevos servicios inteligentes ha impulsado un creciente interés académico y profesional en el estudio, evaluación y despliegue de estas tecnologías. En este contexto, surge la necesidad de contar con entornos de simulación que permitan analizar el comportamiento de los sistemas V2X de forma controlada, reproducible y accesible.

Este trabajo se enmarca en dicha necesidad, proponiendo el diseño e implementación de un entorno de simulación que integra SUMO y OpenC2X para analizar la comunicación V2V. La presente sección desarrolla el contexto general del proyecto, su motivación, el ámbito en el que se sitúa, el problema que pretende abordar y su relevancia en el marco del Máster Universitario en Ingeniería de Telecomunicación.

1.1.1 Motivación del proyecto

La evolución hacia las ciudades inteligentes y la movilidad conectada está impulsando el desarrollo de nuevas tecnologías de comunicación vehicular. En este contexto, las redes vehiculares (VANETs) y, en particular, la comunicación *V2X* (Vehicle-to-Everything), permiten el intercambio de información entre vehículos (*V2V*), vehículo a red (*V2N*) y entre vehículos e infraestructuras (V2I), contribuyendo a mejorar la seguridad vial, reducir los accidentes y optimizar el flujo del tráfico urbano.

El presente Trabajo Fin de Máster surge del interés por explorar estas tecnologías desde un enfoque práctico y simulado, que permita evaluar el comportamiento y las prestaciones de la comunicación V2V. La propuesta combina conocimientos de telemática, simulación y comunicaciones, y se alinea con las tendencias actuales del sector y las líneas de investigación en movilidad conectada.

Además, a raíz del reciente apagón eléctrico, he reflexionado sobre cómo podría haberse gestionado mejor el tráfico en las grandes ciudades y en las carreteras, especialmente mediante los semáforos y la coordinación vial. Esta experiencia me impulsó a explorar la comunicación V2X

como una herramienta que, incluso en situaciones críticas, pueda ayudar a mejorar la eficiencia y la seguridad del transporte.

1.1.2 Ámbito del proyecto

El trabajo se desarrolla en el ámbito de la telemática y las comunicaciones vehiculares, combinando redes de comunicación, simulación y sistemas inteligentes de transporte (ITS). En particular, el proyecto se centra en la integración de un entorno de simulación que conecte el tráfico urbano (*SUMO*) con el sistema de comunicaciones *OpenC2X*, permitiendo analizar las interacciones y métricas de rendimiento en escenarios controlados.

El enfoque principal del proyecto es la comunicación *Vehicle-to-Vehicle* (V2V), basada en el intercambio de mensajes CAM entre vehículos simulados. Además, se incorpora una extensión funcional *Vehicle-to-Infrastructure* (V2I), mediante la cual la infraestructura (un semáforo gestionado desde SUMO) puede generar eventos que desencadenan la transmisión de mensajes DENM.

No se van a abordar aspectos como la autenticación, seguridad PKI o simulación de canales radioeléctricos físicos.

1.1.3 Problema que intenta resolver

La validación práctica de tecnologías V2X suele requerir infraestructuras costosas y entornos de prueba complejos. Este proyecto propone un entorno de simulación reproducible y económico, basado en herramientas de código abierto, que permita estudiar la comunicación V2V y evaluar su rendimiento en distintos contextos de movilidad.

El objetivo es disponer de una plataforma flexible que facilite el análisis de parámetros como la latencia, la tasa de entrega de mensajes o la estabilidad de la conexión entre nodos.

1.1.4 Relevancia en el contexto del máster

El proyecto está directamente alineado con las competencias del Máster Universitario en Ingeniería de Telecomunicaciones, al integrar conocimientos de redes, simulación, comunicaciones inalámbricas y análisis de sistemas distribuidos. Además, aporta una aplicación práctica orientada a la investigación y al desarrollo de tecnologías emergentes en el ámbito de las comunicaciones vehiculares.

1.2 Objetivos del trabajo

El objetivo general del proyecto es diseñar y evaluar un entorno de simulación que permita analizar la comunicación *V2V (Vehicle-to-Vehicle)* mediante la integración de *SUMO* y *OpenC2X*, estudiando el comportamiento y las métricas básicas del sistema en escenarios urbanos simulados.

Además, se incorpora una extensión funcional *V2I (Vehicle-to-Infrastructure)*, en la que la infraestructura (un semáforo gestionado desde SUMO) puede generar eventos que desencadenan la creación de mensajes DENM por parte de OpenC2X. Esta funcionalidad complementa el análisis V2V principal, ampliando el alcance del sistema hacia comunicaciones vehículo-infraestructura.

Para lograrlo, se establecen los siguientes objetivos técnicos, clasificados por prioridad:

1.2.1 Objetivo principal

El proyecto presenta un objetivo principal, desglosado en tres líneas de trabajo técnicas:

O1. Integrar *SUMO* y *OpenC2X* para permitir la simulación conjunta de tráfico y comunicaciones *V2V*. Este objetivo abarca la adaptación del entorno, el desarrollo de módulos de comunicación y la conexión entre ambas plataformas mediante la interfaz *TraCI (Traffic Control Interface)*.

O2. Adaptar y configurar *OpenC2X* para su correcta ejecución en los sistemas actuales (Ubuntu 24.04). Incluirá la migración del código, resolución de dependencias, actualización de librerías y validación funcional de los módulos base de *OpenC2X*.

O3. Desarrollar y probar módulos adicionales que amplíen la funcionalidad de *OpenC2X*. Implementación o modificación de módulos orientados a la comunicación y sincronización con SUMO, asegurando la compatibilidad con TraCI.

1.2.2 Objetivos secundarios

O4. Validar el funcionamiento de los módulos de comunicación, incluyendo *CAM*, *GPS* y otros servicios básicos de *OpenC2X*, así como la

recepción y procesamiento de eventos externos que desencadenan mensajes DENM en el flujo V2I.

O5. Diseñar y ejecutar escenarios de simulación controlados. Creación de distintos escenarios vehiculares en SUMO (densidad baja, media y alta) que sirvan para analizar la influencia de la densidad del tráfico en el intercambio de mensajes V2V.

O6. Diseñar y ejecutar escenarios de simulación para la comunicación V2I.Creación de escenarios con distinto número de semáforos controlados desde SUMO, destinados a analizar el comportamiento del sistema ante múltiples eventos de infraestructura.

O7. Medir y analizar métricas básicas de rendimiento. Evaluación de parámetros como latencia, tasa de entrega de mensajes y estabilidad de comunicación entre vehículos, utilizando los datos generados por OpenC2X y SUMO.

1.2.3 Requisitos técnicos

A continuación se definen los requisitos técnicos que debe cumplir el simulador integrado SUMO↔OpenC2X. Estos requisitos representan las capacidades mínimas, medibles y verificables que el sistema final debe proporcionar.

R1. **Comunicación SUMO↔OpenC2X.** El simulador debe permitir comunicación bidireccional mediante la interfaz TraCI. *Criterio de validación:* conexión TCP estable y lectura continua de posición, velocidad y orientación desde SUMO.

R2. **Correspondencia vehículo↔nodo.** Cada vehículo definido en SUMO debe estar asociado a un nodo independiente en OpenC2X. *Criterio de validación:* mapeo 1:1 vehicleID↔nodeID registrado en logs.

R3. **Generación de mensajes CAM.** Los nodos OpenC2X deben generar y transmitir mensajes CAM. *Criterio de validación:* presencia de mensajes CAM en los registros de envío y recepción.

R4. **Actualización de posición desde SUMO.** La posición de cada nodo OpenC2X debe provenir exclusivamente de los datos obtenidos vía TraCI. *Criterio de validación:* coincidencia entre trazas SUMO y OpenC2X, sin desfases superiores a un tick.

R5. **Latencia de comunicaciones.** La latencia extremo a extremo en la entrega de mensajes CAM debe ser inferior a 40ms (valor de referencia para ITS-G5). *Criterio de validación:* cálculo de latencia media en simulaciones de prueba.

R6. **Registro de métricas.** El sistema debe registrar métricas de rendimiento, incluyendo latencia, tasa de entrega y frecuencia de mensajes CAM y DENM. *Criterio de validación:* generación de ficheros de log completos y analizados.

R7. **Procesamiento de eventos V2I.** OpenC2X debe ser capaz de recibir eventos externos provenientes de SUMO (p. ej., semáforo en rojo) y generar un mensaje DENM asociado. *Criterio de validación:* recepción del evento, generación del DENM y su registro en los logs de DENM, DCC y LDM.

R8. **Estabilidad operativa.** El simulador debe ser estable en las ejecuciones. *Criterio de validación:* ejecución de los escenarios sin fallos TraCI ni desconexiones.

1.3 Impacto en sostenibilidad, ética y diversidad

El desarrollo de este Trabajo Final de Máster presenta un impacto positivo en términos de sostenibilidad, ética y diversidad.

En primer lugar, el proyecto se basa íntegramente en herramientas de software libre y accesible(SUMO, OpenC2X, Ubuntu 24.04 y VirtualBox), lo que reduce la dependencia de soluciones propietarias, promueve la transparencia tecnológica y facilita la accesibilidad al conocimiento. Asimismo, el uso de entornos completamente simulados elimina la necesidad de vehículos reales o equipamiento físico, disminuyendo el consumo de recursos materiales y minimizando la huella ambiental del proyecto.

Desde una perspectiva ética, el trabajo contribuye al avance de los sistemas cooperativos ITS orientados a la seguridad vial. La simulación de comunicaciones V2X permite reproducir situaciones de riesgo sin poner en peligro a personas, infraestructuras o vehículos, garantizando así un enfoque responsable en el análisis experimental.

En términos de diversidad e inclusión, las tecnologías V2X tienen el potencial de mejorar la movilidad urbana y la seguridad de colectivos especialmente vulnerables, como personas mayores, peatones, ciclistas o

usuarios con movilidad reducida. Aunque este TFM aborda la integración desde un punto de vista técnico, el desarrollo de sistemas de transporte más eficientes y cooperativos contribuye a un entorno urbano más seguro, accesible e igualitario.

1.4 Enfoque y metodología seguida

El trabajo se ha desarrollado siguiendo un enfoque incremental y basado en validación continua. En una primera fase se aseguró la correcta instalación, compilación y ejecución de OpenC2X en un entorno moderno, resolviendo incompatibilidades con bibliotecas y dependencias. Posteriormente, se integró SUMO mediante la API TraCI, desarrollando el módulo SumoInterface para sincronizar en tiempo real la movilidad simulada con los servicios internos de OpenC2X. Cada avance técnico se verificó mediante pruebas unitarias y funcionales, siguiendo una metodología experimental en la que se comparaban los resultados obtenidos con el comportamiento esperado del sistema. Finalmente, se definieron escenarios de simulación progresivos y se evaluó el rendimiento del sistema utilizando métricas estandarizadas en el ámbito de las comunicaciones V2X.

1.5 Breve resumen de los resultados

Las pruebas realizadas confirman que la integración SUMO↔OpenC2X funciona de manera estable, permitiendo sincronizar en tiempo real la movilidad simulada con la generación de mensajes CAM. El sistema mantiene latencias internas muy bajas (1-2 ms) y una tasa de entrega del 100 % en todos los escenarios evaluados. Además, se observa que el mecanismo DCC tiene un papel clave en la estabilidad del sistema, ya que su desactivación incrementa la latencia y altera el intervalo de generación de mensajes. En conjunto, los resultados validan la plataforma como herramienta adecuada para el análisis de comunicaciones V2V en entornos controlados.

De forma complementaria, se desarrolló una extensión funcional *V2I* que permite generar mensajes DENM a partir de eventos de infraestructura, en particular desde semáforos modelados en SUMO. Las pruebas verificaron el flujo completo SUMO → OpenC2X → DENM → LDM, mostrando una latencia reducida (2-3 ms) y una tasa de entrega cercana al

100 %. Asimismo, se comprobó que la presencia de múltiples semáforos no afecta al rendimiento del sistema.

Los resultados obtenidos validan la plataforma desarrollada como una herramienta adecuada para el análisis de comunicaciones V2V y V2I en entornos controlados.

1.6 Estructura de la memoria

La memoria se organiza en varios capítulos. El **Capítulo 1** introduce el contexto del trabajo, sus objetivos, la metodología seguida y un breve resumen de los resultados. El **Capítulo 2** presenta el estado del arte relativo a las tecnologías V2X y a las herramientas utilizadas en el proyecto. El **Capítulo 3** describe el diseño y la arquitectura del sistema propuesto. El **Capítulo 4** detalla el desarrollo e implementación de los distintos módulos y la integración entre SUMO y OpenC2X. El **Capítulo 5** recoge la evaluación experimental del sistema y el análisis de los resultados obtenidos. El **Capítulo 6** incluye el estudio de viabilidad técnica, económica y ambiental del proyecto. El **Capítulo 7** presenta las conclusiones, las limitaciones del trabajo y posibles líneas de mejora futuras. Finalmente, se incorporan el glosario, la bibliografía y los anexos.

2 Estado del arte

La seguridad vial continúa siendo una prioridad a nivel global. En España, según los datos publicados por la Dirección General de Tráfico (DGT), durante 2024 se produjeron 1040 siniestros mortales, en los que fallecieron 1154 personas y 4634 resultaron heridas graves que requirieron hospitalización [1]. Aunque estas cifras suponen una mejora respecto a años anteriores, reflejan la necesidad de seguir avanzando en sistemas inteligentes de transporte (ITS) que contribuyan a reducir los accidentes y la mortalidad en carretera.

A nivel internacional, los datos son igualmente significativos. En Corea del Sur, durante el año 2022 se registraron 196836 accidentes de tráfico con 2735 fallecimientos, una reducción del 6.2 % respecto al año anterior. Sin embargo, el número de muertes en accidentes que involucraron vehículos de dos ruedas, como motocicletas o patinetes eléctricos, aumentó un 36.8 % [2]. Para reducir estos accidentes y fallecimientos, se están desarrollando CITS (Cooperative Intelligent Transport Systems) y servicios de conducción autónoma que combinan la fusión de sensores y las tecnologías de comunicación de vehículo a todo (V2X).

El artículo añade que los sensores habituales en conducción autónoma (como las cámaras, radares y lidar) presentan limitaciones bajo condiciones meteorológicas adversas, motivo por el cual se pone de manifiesto la relevancia mundial del desarrollo de las tecnologías V2X, orientadas a mejorar la seguridad vial mediante la comunicación cooperativa entre vehículos (V2V) y entre vehículos e infraestructuras (V2I).

En este marco se sitúa el presente trabajo, que estudia y evalúa la comunicación V2V mediante mensajes CAM (Cooperative Awareness Messages) utilizando la plataforma OpenC2X y el simulador de tráfico SUMO, con el objetivo de analizar el comportamiento y las prestaciones de las comunicaciones vehiculares en entornos controlados y reproducibles.

2.1 Tecnologías V2X y requisitos de servicio

2.1.1 Comunicaciones V2X basadas en ITS-G5

El estándar ETSI ITS-G5, constituye la base tecnológica de la comunicación cooperativa vehicular en Europa [3].

OpenC2X se ha consolidado como una plataforma de software abierta

que implementa la pila ETSI ITS-G5 sobre IEEE 802.11p, con un diseño modular y extensible. OpenC2X consta de módulos altamente independientes, que pueden ampliarse y reutilizarse fácilmente. Esta característica permite que cada componente se ejecute de forma independiente, simplificando el desarrollo y las pruebas.

OpenC2X implementa los principales servicios definidos por ETSI, incluyendo los mensajes CAM (Cooperative Awareness Message) y DENM (Decentralized Environmental Notification Message), así como el DCC (Decentralized Congestion Control). Este último gestiona la carga del canal inalámbrico ajustando la frecuencia de transmisión según la densidad de tráfico, empleando diferentes estados operativos (relaxed, active o restricted) y el mecanismo EDCA para priorizar los mensajes críticos.

Los mensajes CAM se generan de forma periódica cuando cambia la posición, la velocidad o el rumbo al menos una vez por segundo. El módulo LDM (Local Dynamic Map) almacena tanto los mensajes enviados como los recibidos, lo que permite monitorear en tiempo real y realizar un análisis posterior.

En cuanto a la información de detección, la plataforma utiliza el servicio gpsd de Linux para obtener datos de posición, velocidad y tiempo, y admite el uso de trazas GPS previamente registradas.

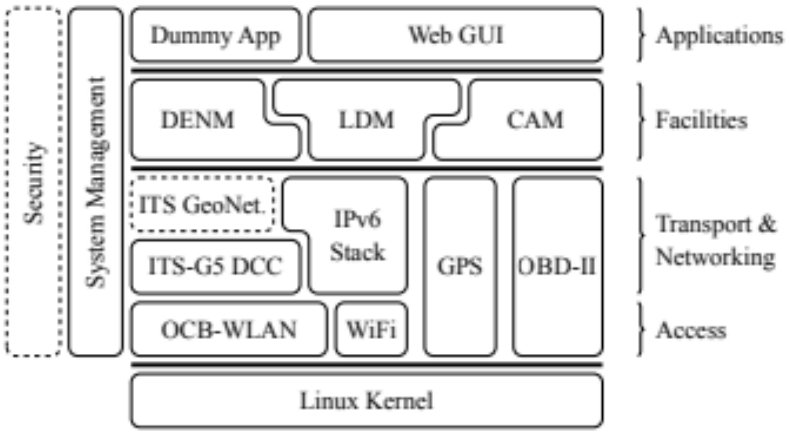

Figura 1: High-level architecture of OpenC2X realizing ETSI ITS-G5 stack. The modules communicate with each other via ZeroMQ. Dashed modules are not yet implemented in OpenC2X.

Las pruebas de realizadas con equipos comerciales demuestran interoperabilidad del entorno, recibiendo y enviando mensajes CAM y DENM con un Cohda MK5.

2.1.2 Comunicaciones V2X basadas en C-V2X

En paralelo a las tecnologías ITS-G5, la industria de las telecomunicaciones ha desarrollado la familia Cellular V2X (C-V2X) [4], que aprovecha la infraestructura móvil existente y la interfaz PC5 (sidelink) para la comunicación directa entre vehículos.

C-V2X combina dos modos de transmisión complementarios: el modo LTE-PC5, que permite la comunicación directa entre vehículos dentro de un área de proximidad sin pasar por la infraestructura, y el modo LTE-Uu, que utiliza la red celular cuando el vehículo se encuentra bajo cobertura.

Una de las ventajas fundamentales de C-V2X frente a ITS-G5 es su esquema de acceso al medio. Mientras que 802.11p utiliza un mecanismo de contienda CSMA/CA, C-V2X emplea un planificador (scheduler) para asignar recursos de radio de forma controlada. Esto permite reducir colisiones e interferencias, además de ofrecer garantías de calidad de servicio (QoS).

2.1.3 Comunicaciones V2X basadas en 5G NR V2X

Con la llegada de 5G NR (New Radio) V2X [5] se introducen funcionalidades avanzadas diseñadas para soportar los casos de uso más exigentes de la conducción conectada y automatizada.

NR V2X está diseñado para soportar requisitos de calidad de servicio (QoS) diversos y estrictos, definidos en términos de prioridad, velocidad de transmisión, latencia, fiabilidad, velocidad de datos y alcance de comunicación. Esta arquitectura define flujos QoS asociados a diferentes tipos de comunicación (unicast, groupcast y broadcast), cada uno con sus propios niveles de prioridad y gestión de recursos.

El modelo QoS de NR V2X se basa en el de 5G, utilizando reglas PC5 QoS y flujos QoS Flows mapeados sobre portadoras de radio laterales (SLRB). Esta estructura permite mantener garantías de servicio incluso fuera de cobertura.

Los requisitos definidos por 3GPP para aplicaciones críticas, como el platooning, son especialmente exigentes: latencias extremo a extremo de 10 a 25ms, fiabilidad de 90—99.99 % y rangos de comunicación de 80—350m.

Asimismo, países como Corea del Sur han impulsado proyectos nacionales de despliegue de 5G NR V2X, destacando que esta tecnología

Use case group	Payload (Bytes)	Tx rate (Message/ Sec)	Max end-to-end latency (ms)	Reliability (%)	Data rate (Mbps)	Required communication range (meters)
Vehicles Platooning	50-6000	2-50	10-25	90-99.99	<= 65	80-350
Advanced Driving	SL: 300-12000 UL: 450	SL: 10-100 UL: 50	10-100	90-99.999	SL: 10-50 UL: 0.25-10 DL: 50	360-700
Extended Sensors	1600	10	3-100	90-99.999	10-1000	50-1000
Remote Driving	16000-41700	33-200	5	99.999	UL: 25 DL: 1	1000+

Note 1: If not specified otherwise, the requirement applies to all link types (SL, DL, and UL).
Note 2: In case of the Remote Driving use case group, [12] does not specify the values for Payload, Tx rate, and Required communication range. For completeness, we include these missing values based on [127].

Figura 2: Rangos de requisitos para los casos de uso de 3GPP.

admite velocidades ultra altas, retrasos ultra bajos y alta fiabilidad, con objetivos de más de 150 Mbps de capacidad, latencias inferiores a 3 ms y fiabilidad superior al 99.99 %.

En conjunto, las soluciones NR V2X suponen una evolución sustancial respecto a ITS-G5, al ofrecer mayor capacidad de gestión de la calidad de servicio (QoS) y mayor flexibilidad en los modos de transmisión. Sin embargo, su despliegue exige hardware especializado, infraestructura de red avanzada y una gestión compleja del espectro, factores que dificultan su adopción en entornos académicos y de simulación.

2.2 Plataformas abiertas sobre ITS-G5

2.2.1 OpenC2X

OpenC2X [6] [3] implementa, además de sus módulos principales y el control de congestión (DCC), el estándar IEEE 802.11p en el kernel de Linux y aplica la priorización de tráfico mediante EDCA para diferenciar los mensajes CAM y DENM.

En sus conclusiones, los autores describen el sistema como “una plataforma experimental y de prototipado de código abierto compatible con el estándar ETSI ITS G5”, señalando que complementa implementaciones anteriores que carecían de funcionalidades clave como la gestión de DCC o LDM.

Asimismo, destacan que su carácter abierto y arquitectura extensible permiten realizar pruebas reales y añadir nuevos módulos, tales como GeoNetworking o mecanismos de seguridad, desarrollados por la comunidad investigadora.

Arquitectura y módulos de OpenC2X. Además de los módulos principales de comunicación vehicular, OpenC2X incluye una arquitectura modular inspirada en el estándar ETSI ITS-G5, donde cada servicio se ejecuta como un proceso independiente comunicado mediante ZeroMQ. Esta organización permite extender y reemplazar fácilmente componentes, así como realizar pruebas experimentales con configuraciones personalizadas.

En la Figura 3 se muestra la arquitectura lógica de OpenC2X, basada en capas funcionales que incluyen servicios de percepción, gestión de posición, generación de mensajes y control de congestión.

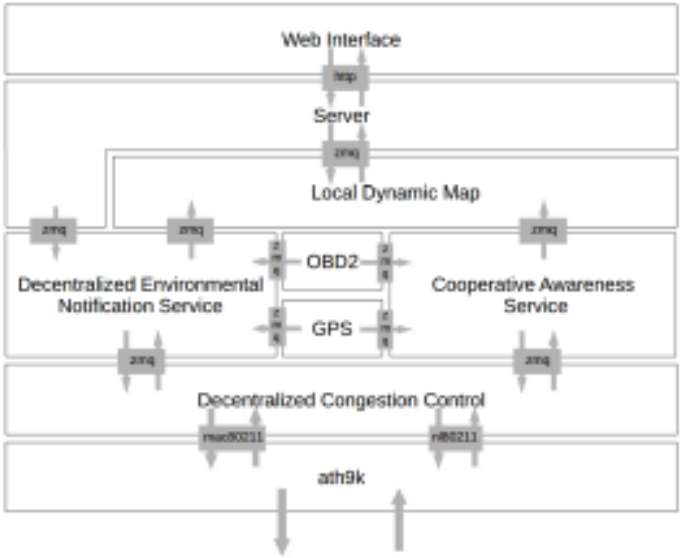

Figura 3: Arquitectura modular de OpenC2X [7].

Los módulos más relevantes para los sistemas V2X implementados son:

- **CAM (Cooperative Awareness Message):** Responsable de generar los mensajes CAM que describen posición, velocidad y estado del vehículo.
- **DENM (Decentralized Environmental Notification Message):** Encargado de gestionar eventos y situaciones de riesgo en el entorno.
- **GPS:** Proporciona la información de posición que utilizan los módulos CAM y DENM para completar sus campos.

- **LDM (Local Dynamic Map):** Base de datos local que almacena objetos, eventos y mensajes recibidos.
- **DCC (Decentralized Congestion Control):** Controla el uso del canal de comunicaciones para evitar saturación y regular la tasa de transmisión.

Trabajos previos, como el presentado en [7], emplean esta arquitectura modular para desarrollar funcionalidades de V2X, demostrando que OpenC2X es una plataforma madura y flexible para investigación en comunicaciones vehiculares.

2.2.2 AutoC2X

La plataforma AutoC2X [8] amplía las capacidades de OpenC2X al integrar la percepción cooperativa entre los vehículos autónomos. Esta solución combina el software de conducción autónoma Autoware con la plataforma OpenC2X, permitiendo que los vehículos intercambien información sensorial sobre objetos detectados en su entorno.

El sistema se diseñó para que un vehículo “host” ejecute Autoware (encargado de la detección y localización mediante sensores LiDAR y cámaras), mientras que un “router” ejecuta OpenC2X, encargado de la comunicación ITS-G5 y del intercambio de mensajes V2X entre vehículos y unidades de carretera (RSU).

El proceso de percepción cooperativa se basa en el envío de la información sensorial de los objetos detectados por Autoware hacia OpenC2X a través de TCP/IP. Esta información se transforma en mensajes proxy-CAM, que permiten representar objetos que no poseen capacidad de transmisión V2X propia.

Cuando un vehículo recibe estos mensajes, el módulo OpenC2X decodifica la información, la almacena en el Local Dynamic Map (LDM) y la reenvía a Autoware para su visualización.

2.3 SUMO (Simulation of Urban Mobility)

SUMO es un paquete de simulación de tráfico continuo, microscópico, altamente portátil y de código abierto, diseñado para gestionar grandes redes. Permite la simulación intermodal, incluyendo peatones, e incluye un amplio conjunto de herramientas para la creación de escenario [9].

Las caracteristicas más destacadas de SUMO son [10]:

- Simulación microscópica: los vehículos, los peatones y el transporte público se modelan de forma explícita.
- Simulación de tráfico multimodal, por ejemplo, vehículos, transporte público y peatones.
- Los horarios de los semáforos pueden importarse o generarse automáticamente con SUMO.
- Sin limitaciones artificiales en cuanto al tamaño de la red y el número de vehículos simulados.
- Formatos de importación compatibles: OpenStreetMap, VISUM, VISSIM, NavTeq.
- SUMO está implementado en C ++ y utiliza una biblioteca portátil exclusiva.

SUMO es ampliamente utilizado por la comunidad V2X para generar trazas realistas de vehículos y evaluar aplicaciones en un bucle en línea con un simulador de red.

2.4 Desempeño y fiabilidad en comunicaciones V2V

La comunicación vehículo a vehículo (V2V) es un componente esencial de las aplicaciones cooperativas, ya que permite el intercambio rápido y fiable de información entre vehículos en movimiento, como el platooning, donde los vehículos deben coordinarse para responder instantáneamente.

El artículo [11], propone una plataforma redundante basada en OpenC2X, capaz de operar simultáneamente en las bandas de 5.9GHz y 700MHz, con el objetivo de mejorar la cobertura y la estabilidad de la conexión.

Para aplicaciones como el platooning, puede ser necesaria una latencia de capa de aplicación de extremo a extremo de aproximadamente 40 ms o menos, lo que se consigue mediante DSRC. Este valor constituye una referencia práctica aceptada para sistemas V2V de baja latencia. Sin embargo, el rendimiento de las comunicaciones DSRC se degrada con la distancia y en entornos urbanos o suburbanos, donde los obstáculos y la

difracción reducen la relación señal/ruido. Por este motivo, el trabajo introduce una banda suplementaria a 700 MHz, más robusta ante obstrucciones, con el objetivo de mantener la conectividad cuando la comunicación a 5.9 GHz se interrumpe.

Se realizaron pruebas con hardware embebido y OpenC2X configurado en modo OCB, los cuales mostraron que el canal a 700MHz mantiene una mayor robustez ante pérdidas de señal. Concretamente, para distancias superiores a 130m:

- La relación señal-ruido (SNR) de 5.9GHz es bastante baja.
- La SNR de 700 MHz es aproximadamente 15 dB más alta.
- La latencia es inferior a 30 ms para ambas señales de comunicación.

Esta mejora confirma que el uso de una segunda banda de 700MHz permite compensar la degradación de la comunicación en entornos con obstáculos o interferencias.

3 Diseño y Arquitectura del sistema

3.1 Vista general

En esta sección se presenta la arquitectura general del sistema, mostrando los principales módulos involucrados y el flujo de información entre ellos.

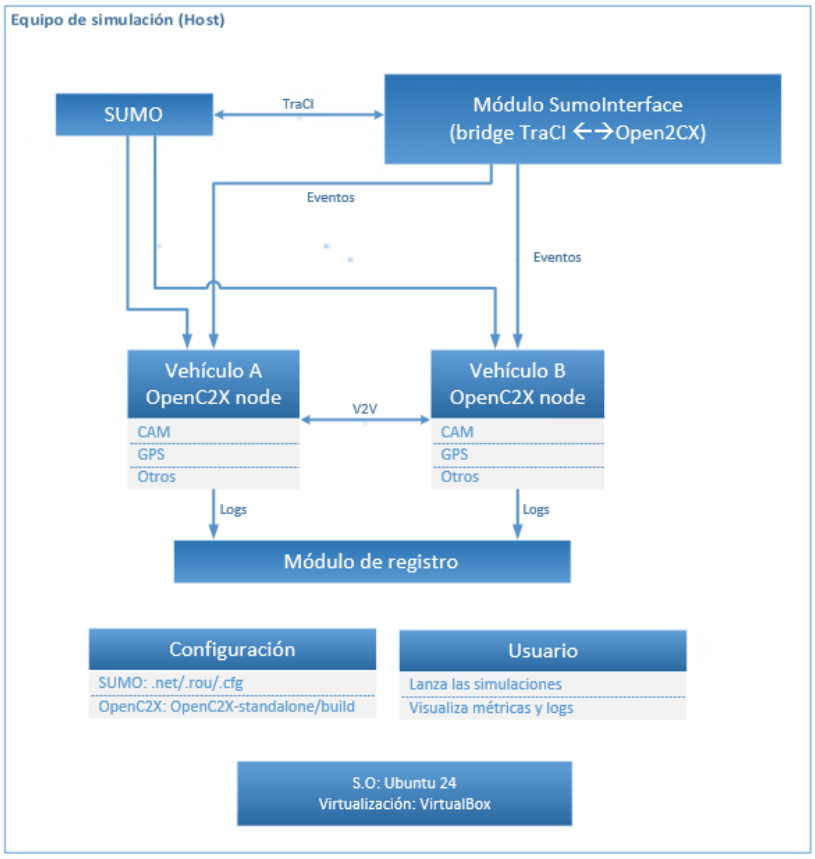

Figura 4: Arquitectura del sistema SUMO ⟷ OpenC2X (escenario V2V)

3.2 Bloques y funciones (descripción breve)

1. *SUMO (escenario de tráfico)*
    - Simula la movilidad de los vehículos (trayectorias, velocidades, interacciones).
    - Ficheros: red (.net.xml), rutas (.rou.xml) y configuración (.sumocfg).
    - Expone los estados de cada vehículo a través de *TraCI*.

2. *Módulo SumoInterface (bridge TraCI ↔ OpenC2X)*
    - El módulo SumoInterface implementa un cliente *TraCI* basado en la librería oficial libtraci (SUMO 1.23.1), estableciendo una conexión TCP con SUMO en el puerto configurado (por defecto, 9999).
    - En cada tick, este cliente consulta las variables de movilidad (posición, velocidad, orientación) y publica los estados actualizados a los nodos *OpenC2X*.
3. *Nodos OpenC2X (Vehículo A, B, . . .)*
    - Procesos independientes que representan "vehículos conectados".
    - *CAM*: generación y recepción de mensajes V2V.
    - *GPS*: actualiza la posición usando los datos recibidos desde SUMO.
    - *DENM*: módulo encargado de generar mensajes de alerta a partir de *triggers* internos o externos.
    - Otros (DCC, LDM)
4. *Módulo de registro*
    - Recoge logs de los nodos (timestamps, IDs, payloads, pérdidas).
    - Genera métricas básicas.
    - Exporta los resultados.

3.3 Flujo de datos

1. *SUMO* avanza el tiempo y actualiza los estados vehiculares.
2. *SumoInterface (TraCI)* lee el estado de cada vehículo y lo distribuye a los nodos *OpenC2X* correspondientes.
3. Cada nodo *OpenC2X*:
    - Actualiza el *GPS*.
    - Genera y recibe mensajes *V2X* mediante *CAM*.

4. Los mensajes *V2V* se intercambian entre nodos mediante *sockets* internos.
5. El módulo *DENM* puede generar alertas a partir de *triggers* externos recibidos (flujo V2I), activados cuando *SumoInterface* detecta un evento en la infraestructura.
6. El módulo de registro revisa los logs y genera las métricas básicas.
7. El usuario revisa las métricas y ajusta los parámetros o escenarios.

3.4 Entorno físico y red

- *Host con virtualización*
  - VirtualBox.
  - Máquina virtual configurada con *Ubuntu 24.04 LTS*, *10 GB de RAM*, *2 núcleos asignados* y *50 GB de almacenamiento en disco*.
- *Red*: comunicación intra-host basada en *sockets*; no se requiere de una interfaz inalámbrica real.
- *Escalabilidad*: añadir más vehículos implica más procesos/nodos *OpenC2X* y más suscripciones *TraCI*.

3.5 Interacción con el Usuario

La ejecución se realiza mediante scricps de arranque, con los parámetros en los ficheros de configuración. Los resultados se analizan revisando los logs generados por *OpenC2X*.

4 Desarrollo del proyecto

Durante el desarrollo del proyecto se han completado diversos hitos técnicos que permiten disponer de una versión funcional del entorno de simulación integrando OpenC2X con SUMO. A continuación se describen los avances más relevantes, agrupados por áreas de trabajo.

4.1 Migración, instalación y validación de OpenC2X

OpenC2X fue originalmente desarrollado y probado en sistemas basados en Ubuntu 16.04, por lo que su instalación directa en un entorno moderno (Ubuntu 24.04 LTS) provocó diversos errores de compilación relacionados con versiones obsoletas de bibliotecas, cambios en dependencias externas y funciones retiradas en Boost y gpsd. Para garantizar la compatibilidad con el sistema actual fue necesario realizar una migración completa del entorno de compilación.

En primer lugar, se descargó el código fuente actualizado desde el repositorio oficial *OpenC2X-standalone* de Florian Klingler [12]. A continuación, se instalaron todas las dependencias del proyecto, incluyendo bibliotecas de red, ASN.1, GPS, SQLite, XML, ZeroMQ y herramientas de compilación (`gcc/clang`, CMake, `protobuf`, `pkg-config`). Algunas de estas dependencias no están disponibles en Ubuntu 24.04 en las versiones esperadas por OpenC2X, lo que obligó a compilar manualmente varios paquetes externos.

Resolución de dependencias faltantes

Durante la fase de configuración del proyecto (`cmake ..`) aparecieron fallos relacionados con las bibliotecas `uci` y `libubox`, utilizadas por el módulo de configuración interna del sistema. Dado que no existen paquetes compatibles en Ubuntu 24.04, ambas bibliotecas se descargaron desde los repositorios oficiales de OpenWRT [13] [14] y se compilaron manualmente. Tras su instalación, fue necesario actualizar la ruta de bibliotecas mediante la variable `LD_LIBRARY_PATH`.

Correcciones en el código fuente

Una vez resueltas las dependencias, la compilación volvió a fallar debido a cambios en las APIs de Boost. Concretamente, la función `boost::posix_time::millisec()` no admite valores de tipo `double` en las versiones recientes de la biblioteca, lo que provocaba errores en varios módulos (`ChannelProber.cpp`, `PktStatsCollector.cpp`, `dcc.cpp`). La solución consistió en realizar conversiones explícitas a entero:

```
ChannelProber::ChannelProber(string ifname, double probeInterval, boost::asio::io_service* io, LoggingUtility& logger) : mLogger(logger) {
        mProbeInterval = probeInterval;
        mIfname = ifname;
        mIoService = io;
        mTimer = new boost::asio::deadline_timer(*mIoService, boost::posix_time::millisec((int)(probeInterval * 1000)));
}
```

Figura 5: boost::posix_time::millisec((int)(probeInterval * 1000))

Durante la compilación también aparecieron errores en el módulo GPS derivados de los cambios introducidos en las versiones recientes de `gpsd`. En la implementación original de OpenC2X (orientada a Ubuntu 16.04), el campo `gps_data.fix.time` era un valor de tipo `double`. Sin embargo, en Ubuntu 24.04 este campo ha pasado a representarse mediante una estructura `timespec`, lo que provocó dos errores de compilación:

- No era posible comparar directamente `timespec` con un `double`:

```
error: no match for operator==
    (operand types are timespec and double)
```

- Tampoco era posible asignar un `timespec` a una variable `double`:

```
error: cannot convert ‘timespec’ to ‘double’ in assignment
```

El código original (Figura 6) utilizaba directamente el campo `fix.time` en comparaciones y asignaciones:

```
if (mGpsData.fix.time == mLastTime) {
    continue;
}
mLastTime = mGpsData.fix.time;
```

Figura 6: Fragmento original de `GpsService.cpp` que producía errores al comparar y asignar un `timespec`.

Para resolver el problema fue necesario convertir manualmente la estructura `timespec` a un valor escalar en formato `double`. La solución definió un nuevo valor temporal:

```
double currentTime =
    mGpsData.fix.time.tv_sec +
    mGpsData.fix.time.tv_nsec / 1e9;
```

y se sustituyeron todas las comparaciones y asignaciones originales para emplear este valor escalar en lugar del `timespec`. El resultado final se muestra en la Figura 7, donde se observa cómo el módulo GPS utiliza `currentTime` tanto para detectar actualizaciones de tiempo como para actualizar `mLastTime`.

```
double currentTime = mGpsData.fix.time.tv_sec + mGpsData.fix.time.tv_nsec / 1e9;
if (std::isnan(currentTime) || currentTime == mLastTime) {
    continue;
}
mLastTime = currentTime;
```

Figura 7: Solución implementada: conversión de `timespec` a `double` mediante `currentTime`.

Otro conjunto de errores (Figura 8) estuvo relacionado con la eliminación de campos y constantes en la biblioteca gpsd. La API actual ya no incluye los campos `gps_data.fix.status` ni constantes como `STATUS_NO_FIX`, y la firma de la función `gps_read()` también ha cambiado. Estas diferencias causaban errores adicionales durante la compilación.

```
/home/saul/OpenC2X-standalone/gps/src/GpsService.cpp:99:29: error: too few arguments to function ‘int gps_read(gps_data_t*, char*, int)’
   99 |                 if (gps_read(gpsdata) == -1) {
      |                             ^~~~~~~~
In file included from /home/saul/OpenC2X-standalone/gps/src/GpsService.h:32:
/usr/include/gps.h:2878:12: note: declared here
 2878 | extern int gps_read(struct gps_data_t *, char *message, int message_len);
      |            ^~~~~~~~
/home/saul/OpenC2X-standalone/gps/src/GpsService.cpp:103:46: error: ‘struct gps_data_t’ has no member named ‘status’
  103 |                 if (gpsdata->set && gpsdata->status > STATUS_NO_FIX) {
      |                                              ^~~~~~
/home/saul/OpenC2X-standalone/gps/src/GpsService.cpp:103:55: error: ‘STATUS_NO_FIX’ was not declared in this scope; did you mean ‘STATUS_RTK_FIX’?
  103 |                 if (gpsdata->set && gpsdata->status > STATUS_NO_FIX) {
      |                                                       ^~~~~~~~~~~~
      |                                                       STATUS_RTK_FIX
/home/saul/OpenC2X-standalone/gps/src/GpsService.cpp: In member function ‘gpsPackage::GPS GpsService::gpsDataToBuffer(gps_data_t*)’:
/home/saul/OpenC2X-standalone/gps/src/GpsService.cpp:131:40: warning: self-comparison always evaluates to false [-Wtautological-compare]
  131 |         if(gpsdata->satellites_visible != gpsdata->satellites_visible) { // set satellites visible to -1 in case of NaN
      |            ~~~~~~~~~~~~~~~~~~~~~~~~~~ ^~ ~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/saul/OpenC2X-standalone/gps/src/GpsService.cpp: In member function ‘void GpsService::receiveData()’:
/home/saul/OpenC2X-standalone/gps/src/GpsService.cpp:146:39: error: no match for ‘operator!=’ (operand types are ‘timespec_t’ {aka ‘timespec’} and ‘timespec_t’ {aka ‘timespec’})
  146 |                 if (mGpsData.fix.time != mGpsData.fix.time) {   //skip if time is NaN
      |                     ~~~~~~~~~~~~~~~~~ ^~ ~~~~~~~~~~~~~~~~~
      |                                       |                  |
      |                                       |                  timespec_t {aka timespec}
      |                                       timespec_t {aka timespec}
```

Figura 8: Errores derivados de cambios en las estructuras y constantes de gpsd.

Para mantener la funcionalidad, se actualizaron las comprobaciones internas utilizando los campos actuales de gpsd (`fix.status`, etc.) y se adaptaron las llamadas a `gps_read()`.

Compilación final

Tras estas modificaciones, OpenC2X pudo compilarse correctamente en Ubuntu 24.04 LTS, generando una versión estable de todos los módulos principales (CAM, DCC, LDM y GPS). Se validó la instalación mediante la ejecución individual de los servicios y la verificación manual de los logs generados.

En la Figura 9 se muestra la salida final del proceso de compilación completado.

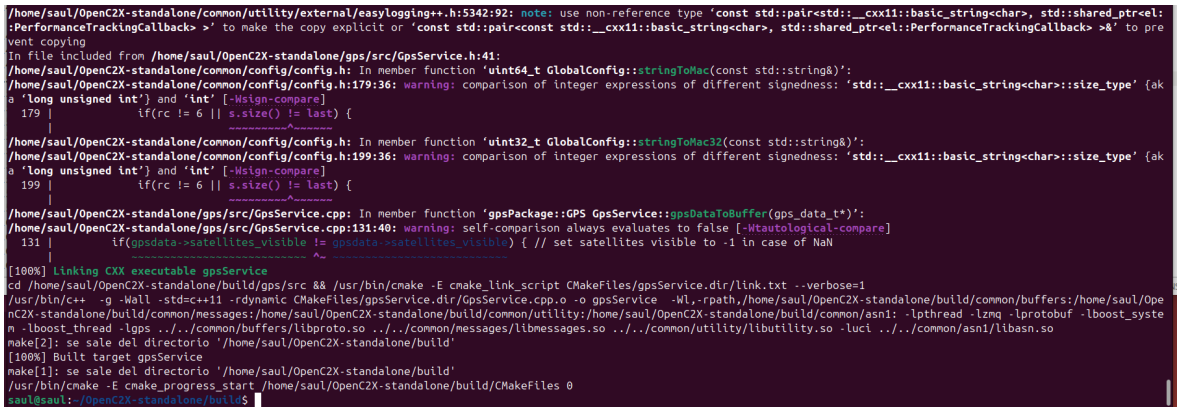

Figura 9: Compilación de OpenC2X en Ubuntu 24.04 tras la migración del entorno

Problemas de configuración y ejecución inicial

A pesar de que la compilación se completó correctamente, la primera ejecución de OpenC2X con el script `run0penC2X.sh` produjo nuevos fallos relacionados con la carga de paquetes de configuración. El ejecutable buscaba los ficheros de configuración exclusivamente en rutas fijas del sistema:

```
/var/run/uci/openc2x
/etc/config/openc2x
```

Esto se confirmó mediante un análisis con `strace`, donde se observaron accesos fallidos (`EN0ENT`) a dichas rutas.

Para resolverlo, se copiaron los archivos de configuración por defecto incluidos en el directorio `config/` del repositorio oficial de OpenC2X [12] a la ruta esperada por el sistema:

```
sudo mkdir -p /etc/config
sudo cp OpenC2X-standalone/config/* /etc/config/
```

Una vez presentes los ficheros, OpenC2X pudo cargar correctamente los módulos de configuración (`openc2x`, `openc2x_cam`,`openc2x_common`, `openc2x_dcc`, `openc2x_denm`, `openc2x_gps`, `openc2x_httpServer`, `openc2x_ldm`, `openc2x_obd2`).

Sin embargo, durante el arranque seguía apareciendo el error:

```
Invalid MAC. Check /etc/config/openc2x_common
```

Tras revisar el código, se identificó que OpenC2X intentaba obtener la dirección MAC leyendo directamente el archivo:

```
/sys/class/net/notDefined/address
```

Esto ocurría porque la variable `mEthernetDevice` del módulo de configuración tomaba por defecto el valor `"notDefined"` al no estar inicializada correctamente. Para verificar el comportamiento, se añadieron trazas de depuración en `config.h`, observándose la ruta errónea y el valor devuelto. La solución consistió en modificar la variable `mEthernetDevice` para que tomara la interfaz de red correcta (en este caso, `enp0s3`).

```
sudo nano /etc/config/openc2x_common
option 'ethernetDevice' 'enp0s3'
```

Tras esta corrección, OpenC2X pudo leer la MAC real del sistema y continuar la ejecución sin errores.

Conclusión de la instalación

Con todos estos ajustes (resolución de dependencias, corrección de incompatibilidades en Boost y gpsd, creación de los módulos de configuración y fijación de la interfaz de red) OpenC2X quedó completamente operativo en Ubuntu 24.04.

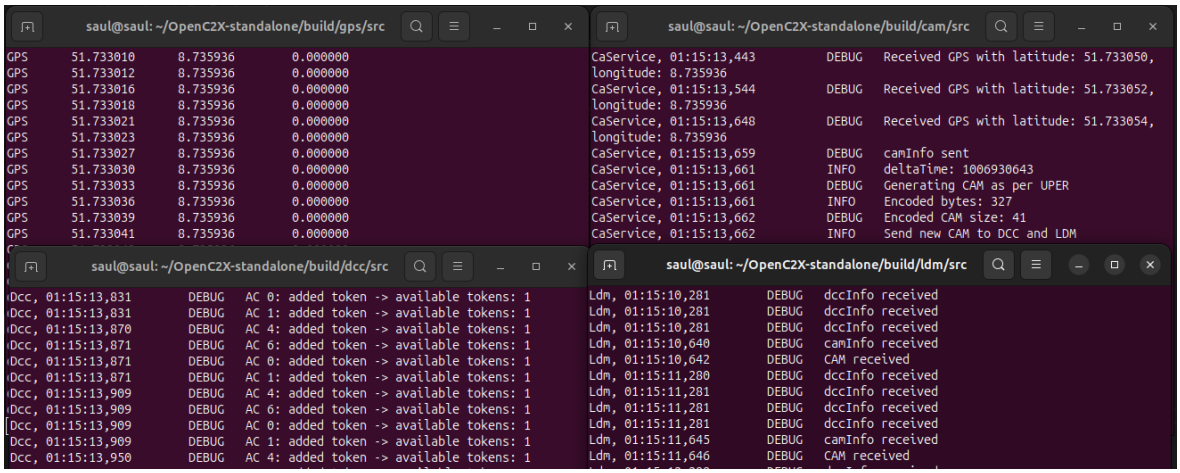

Figura 10: Ejecución de los modulos GPS, CAM, DCC y LDM de OpenC2X en Ubuntu 24.04

4.2 Instalación y validación de SUMO

Se instaló SUMO (Simulation of Urban Mobility) en su versión 1.23.1 mediante los repositorios oficiales, junto con las herramientas auxiliares `sumo-gui`, `netconvert`, `netgenerate` y `netedit`. La instalación no presentó incidencias, y la herramienta quedó operativa tras la comprobación inicial:

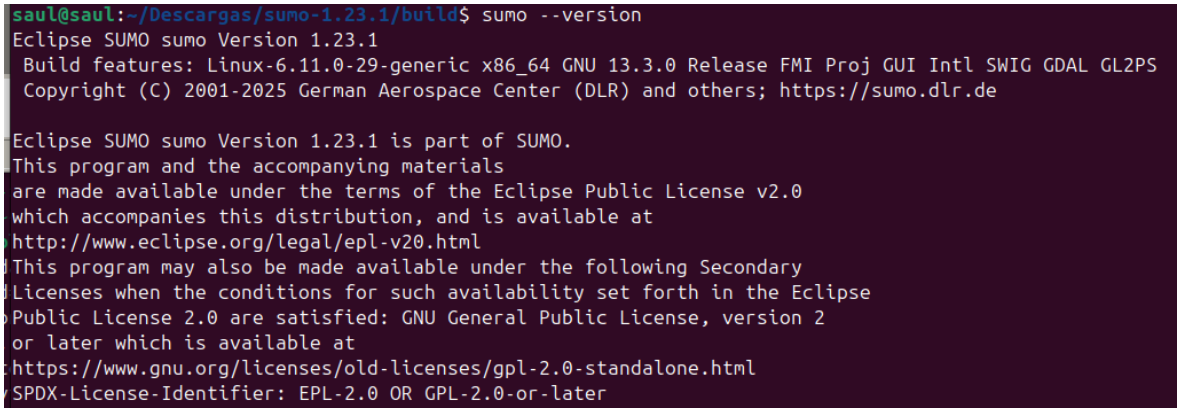

Figura 11: SUMO 1.23.1 instalado correctamente en Ubuntu 24.04.

Creación del escenario de pruebas

Para validar SUMO y preparar la integración posterior con OpenC2X, se creó un escenario sencillo compuesto por una red cuadrada de cuatro nodos conectados entre sí. Inicialmente se utilizó un fichero de red (`net.net.xml`) generado manualmente. Aunque su estructura XML era válida (verificada

mediante `xmllint`), SUMO producía un *segmentation fault* al cargarlo, como se observa en la Figura 12.

Figura 12: Fallo *segmentation fault* al cargar la red generada manualmente.

Para evitar errores derivados del formato interno, se decidió generar la red de forma automática utilizando `netgenerate` [15]:

```
netgenerate --grid --grid.number=2 --output-file=net.net.xml
```

Posteriormente se editó la red con `netedit` para confirmar su geometría, como se muestra en la Figura 13.

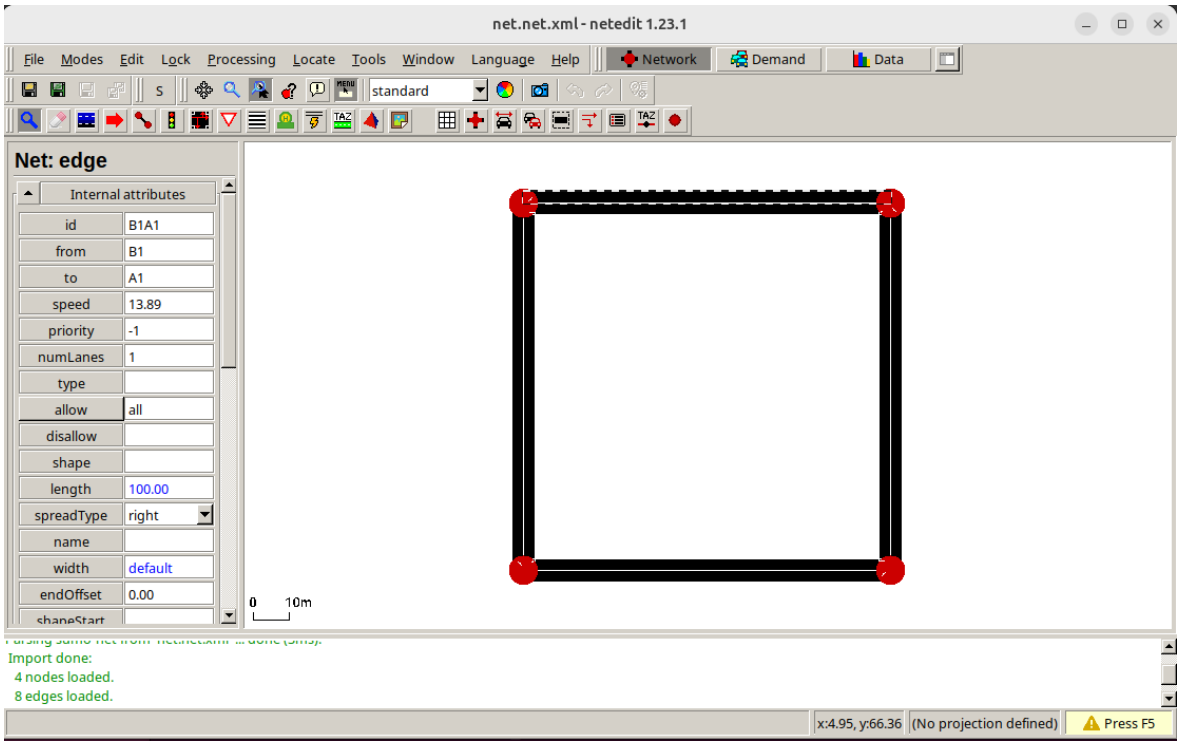

Figura 13: Red cuadrada generada con netgenerate y visualizada en netedit.

Definición de rutas y vehículo

A continuación, se crearon las rutas del escenario en el fichero `routes.rou.xml`. Para ello se identificaron los IDs de los edges generados automáticamente (visibles en `netedit`) y se definió un único vehículo:

```
<routes>
    <vType id="car" accel="2.6" decel="4.5" length="5"
           maxSpeed="70"/>
    <route id="r0" edges="A0A1 A1B1 B1B0 B0A0"/>
    <vehicle id="veh0" type="car" route="r0" depart="0"/>
</routes>
```

Finalmente, se creó el archivo de configuración principal `config.sumocfg`, que referencia tanto la red como las rutas. Al ejecutar el escenario con `sumo-gui`, la simulación se ejecutó sin errores, validando la instalación:

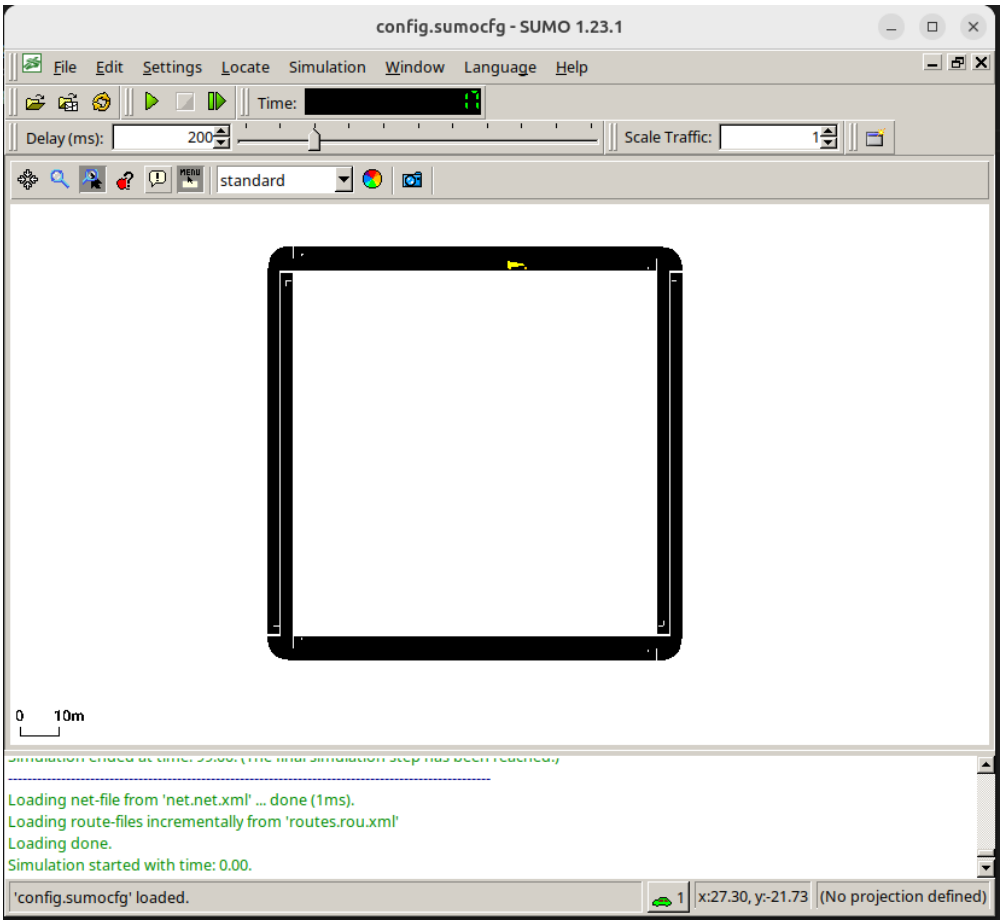

Figura 14: Primer escenario en SUMO funcionando correctamente

Resultado de la validación

La instalación y validación de SUMO ha permitido crear un escenario simple de prueba con un único vehículo, utilizado posteriormente en la primera validación integrada del sistema.

4.3 Desarrollo del módulo SumoInterface

Uno de los avances más significativos del proyecto fue el desarrollo del módulo `SumoInterface`, encargado de establecer la comunicación entre SUMO y OpenC2X mediante la API TraCI [16]. Este módulo opera como un cliente TraCI integrado dentro de OpenC2X y permite obtener en tiempo real la posición, velocidad y orientación de los vehículos simulados en SUMO, sustituyendo completamente al origen GPS real del sistema.

Motivación y ubicación en la arquitectura

Dado que OpenC2X no incorpora por defecto un cliente TraCI, fue necesario implementar uno nuevo dentro del directorio `common/interface/` del proyecto (véase Figura 15). El módulo está formado por los ficheros `SumoInterface.h`, `SumoInterface.cpp` y su correspondiente configuración en CMake.

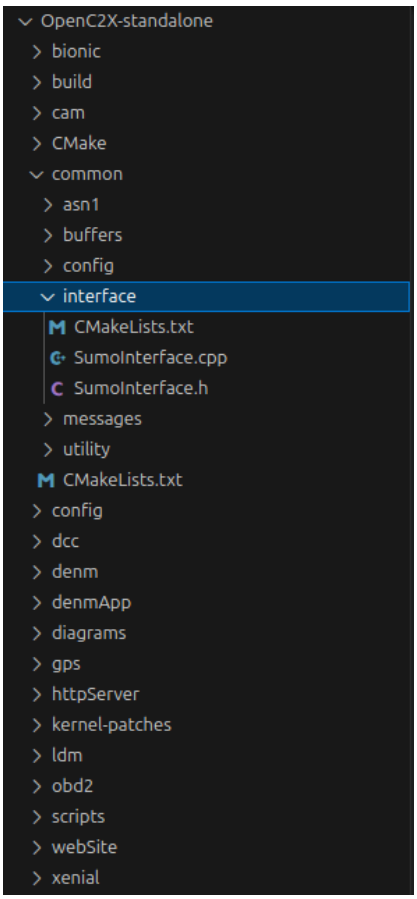

Figura 15: Estructura del directorio interface/ con el módulo SumoInterface.

Integración con TraCIAPI

El primer paso fue integrar en OpenC2X la biblioteca `TraCIAPI` [17] oficial de SUMO (versión 1.23.1). Para ello se añadieron explícitamente los siguientes ficheros y directorios en el `CMakeLists.txt` del módulo:

OpenC2X-standalone > common > interface > CMakeLists.txt

```
set(SUMO_HOME "/home/saul/Descargas/sumo-1.23.1")

add_library(SumoInterface
    SumoInterface.cpp
    SumoInterface.h
    ${SUMO_HOME}/src/utils/traci/TraCIAPI.cpp
    ${SUMO_HOME}/src/foreign/tcpip/socket.cpp
    ${SUMO_HOME}/src/foreign/tcpip/storage.cpp
)

target_include_directories(SumoInterface
    PUBLIC
        ${CMAKE_CURRENT_SOURCE_DIR}
        ${SUMO_HOME}/src
        ${SUMO_HOME}/src/foreign
        ${SUMO_HOME}/src/utils/traci
        ${SUMO_HOME}/src/libsumo
)
```

Figura 16: CMakeLists.txt del directorio interface.

Problemas de comunicación TraCI

Durante las primeras pruebas de comunicación la simulación arrojaba errores del tipo:

```
Error: tcpip::Storage::readIsSafe:
want to read 4 bytes from Storage, but only 3 remaining
```

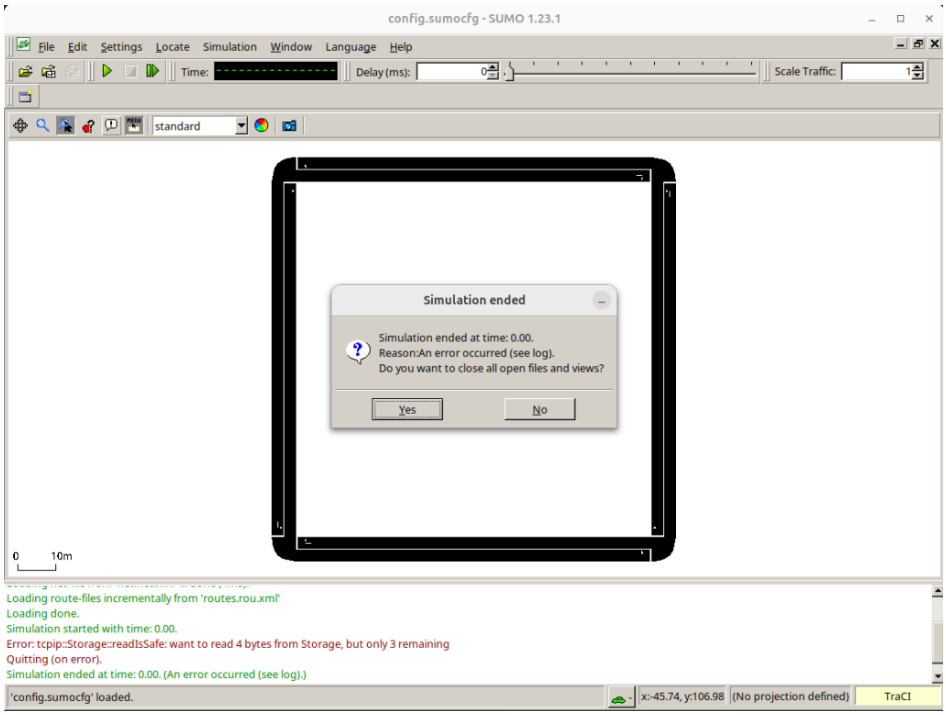

Figura 17: Error: want to read 4 bytes from Storage, but only 3 remaining.

Este error se producía en el primer paso de simulación y sugería un problema de descomposición de mensajes TraCI. Las FAQ [18] de SUMO indicaba que este fallo podía deberse a incompatibilidades entre versiones de SUMO y clientes TraCI. Se probó con versiones antiguas (SUMO 0.32), pero esta alternativa introducía nuevos fallos, y seguía sin resolver el problema original (Figura 18).

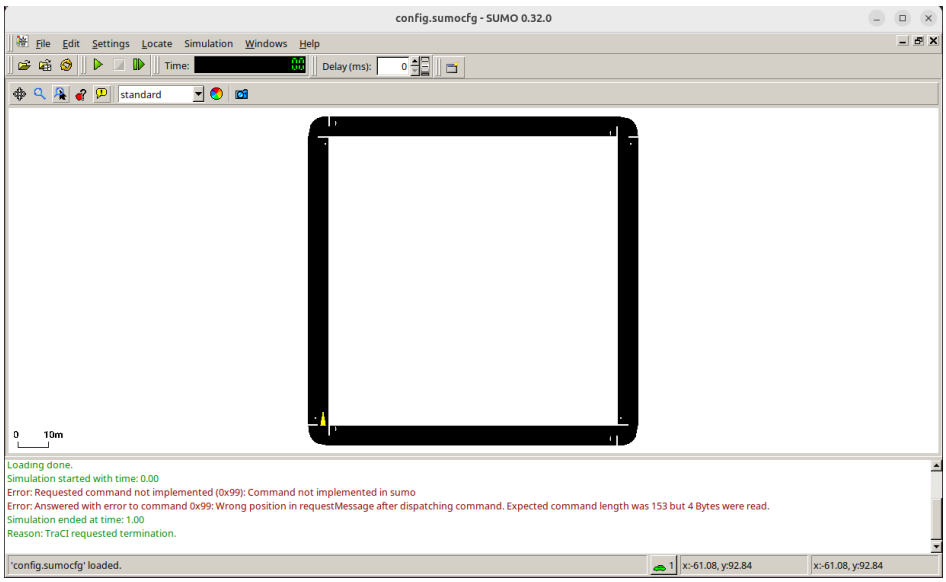

Figura 18: Prueba con SUMO 0.32, Error: Expected command length was 154 bytes but 4 Bytes were read.

El análisis detallado reveló que el error no estaba en SUMO, sino en el cliente TraCI que se había incluido en OpenC2X. La solución definitiva consistió en migrar completamente el módulo de comunicaciones para emplear directamente la biblioteca `libtraci` de SUMO y generar un cliente TraCI estable y compatible con la versión actual.

Conflictos de símbolos y corrección de cabeceras

La integración de `libtraci` provocó inicialmente conflictos de enlazado y redefiniciones de funciones, principalmente en `config.h`, debido a que esta cabecera era incluida múltiples veces en diferentes módulos del sistema. Los errores eran los definidos en la (Figura 19):

Figura 19: Error de redefinición.

Para resolverlo se añadió la directiva `#pragma once` en `config.h`, evitando inclusiones duplicadas, y se marcaron como `inline` las funciones globales que aparecían redefinidas en distintos ficheros. Con ello se eliminaron todos los conflictos de símbolos.

Funcionalidades implementadas

El módulo SumoInterface proporciona las siguientes capacidades:

- Establecimiento de conexión TCP con el servidor SUMO.
- Lectura periódica de variables de movilidad (posición, velocidad, orientación) mediante `TraCIAPI`.
- Conversión de coordenadas SUMO → GPS. El módulo transforma las coordenadas internas x,y de SUMO en latitud/longitud WGS84 antes de enviarlas al módulo GPS, permitiendo una representación geográfica válida en OpenC2X.
- Publicación de esos datos al módulo GPS interno de OpenC2X.
- El módulo se ejecuta en cada tick de simulación.

- Gestión de errores y reconexiones controladas para prevenir fallos de sesión.

Resultado

Con estas operaciones, el módulo SumoInterface ha podido sustituir completamente la posición GPS real por la posición generada por SUMO, de forma transparente para el resto de componentes de OpenC2X, habilitando la integración entre SUMO ⟷ OpenC2X.

4.4 Integración con el módulo GPS de OpenC2X

La integración del módulo `SumoInterface` dentro de la arquitectura de OpenC2X ha requerido sustituir la obtención original de posiciones GPS (realizada mediante `gpsd`) por los datos de movilidad generados por SUMO a través de la API TraCI. con esta modificación se consigue una simulación coherente entre el movimiento del vehículo en SUMO y los mensajes CAM emitidos por OpenC2X.

Adaptación del módulo `GpsService`

Para incorporar la posición simulada, se modificó el servicio `GpsService.cpp` de OpenC2X, añadiendo un nuevo modo de operación (`simulationMode = 2`) en el cual la posición del vehículo ya no se obtiene desde `gpsd`, sino directamente desde el módulo `SumoInterface`.

Los cambios principales realizados fueron:

- Añadir un puntero a `SumoInterface` en `GpsService.h` para permitir la comunicación directa entre ambos módulos.
- Inicializar este componente durante la fase de arranque del servicio cuando `simulationMode` toma el valor 2.
- Llamar periódicamente a la función `simulationStep()` del módulo `SumoInterface` para mantener la sincronización con el avance temporal de SUMO.
- Obtener la posición actual mediante `getVehiclePosition()`, retornando latitud, longitud y altitud ya convertidas al sistema WGS84.

- Construir un mensaje GPS en formato Protobuf (`gpsPackage::GPS`) y enviarlo mediante ZeroMQ al resto de módulos dependientes (CAM, DENM, LDM).

El módulo GPS de OpenC2X emplea mensajes en formato Protobuf, por lo que los datos de posición recibidos desde SUMO deben empaquetarse en un mensaje `gpsPackage::GPS` antes de su envío al resto de servicios del sistema.

Toda esta lógica se implementó en la nueva rutina `simulateFromSumo()`, que ejecuta el ciclo completo:

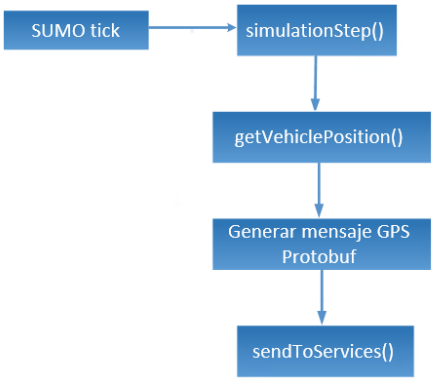

Figura 20: Ciclo de la rutina simulateFromSumo().

Sincronización entre SUMO y OpenC2X

Con esta integración, cada tick de SUMO desencadena inmediatamente la actualización de la posición del vehículo dentro de OpenC2X. Esto garantiza la coherencia espacial y temporal de toda la simulación, permitiendo que los mensajes CAM emitidos por el sistema reflejen fielmente el movimiento del vehículo definido en SUMO.

El flujo de información queda, por tanto, completamente unificado:

SUMO → SumoInterface → GpsService → CAM

Este ciclo fue validado al comprobar que las posiciones enviadas al módulo CAM coincidían exactamente con las trayectorias definidas en los ficheros `.rou.xml` y `.net.xml` empleados en SUMO.

Ajustes adicionales

Durante la integración se realizaron ajustes complementarios:

- Se modificó el archivo `CMakeLists.txt` del servicio GPS para enlazarlo correctamente con el módulo `SumoInterface`.
- Se añadió el parámetro `system.simConFile` en la configuración del sistema, necesario para indicar el archivo `routes.rou.xml` desde el cual se obtiene el identificador del vehículo.
- El temporizador interno (`deadline_timer`) se ajustó a `500 ms` para garantizar la estabilidad durante las primeras pruebas.

Resultado

Tras completar esta integración, OpenC2X pudo ejecutar una simulación end-to-end totalmente funcional. SUMO proporcionaba la movilidad, `SumoInterface` la traducía a un formato compatible, `GpsService` generaba los mensajes GPS adecuados y el módulo CAM transmitía la información con coherencia.

Esta integración validó el funcionamiento del ciclo completo SUMO ⟷ OpenC2X, habilitando las pruebas finales del sistema de comunicación V2V.

4.5 Primera prueba integrada: Hola mundo

Con la integración básica completada, se ejecutó un primer escenario sencillo formado por un único vehículo circulando en la red generada con SUMO. Esta prueba tuvo como objetivo validar el funcionamiento end-to-end del flujo de datos entre la simulación de movilidad (SUMO) y la simulación de comunicaciones (OpenC2X).

Durante la ejecución se verificó lo siguiente:

- SUMO actualizaba la posición del vehículo en cada paso de simulación.
- El módulo `SumoInterface` recibía los datos a través de TraCI sin pérdidas ni retrasos.

- El módulo GPS de OpenC2X actualizaba su estado correctamente con las coordenadas en WGS84.
- El generador de mensajes CAM producía mensajes coherentes con la posición recibida.

Este resultado permitió confirmar que la arquitectura desarrollada funcionaba de extremo a extremo, validando el flujo completo de información entre ambos simuladores.

Comportamiento del módulo CAM

Durante la ejecución también se observaron mensajes del tipo *Ignore heading: not moved more than 0.300000 meters*. Este comportamiento es coherente con la configuración interna del sistema, en el cual se estableció un umbral de desplazamiento mínimo de **0,3 m** para generar un nuevo mensaje CAM. Es decir, sólo se envía un CAM cuando el vehículo ha cambiado su posición al menos esa distancia o ha transcurrido el intervalo mínimo entre transmisiones. Por tanto, este comportamiento confirma que el módulo CAM opera correctamente según los parámetros definidos en nuestra implementación.

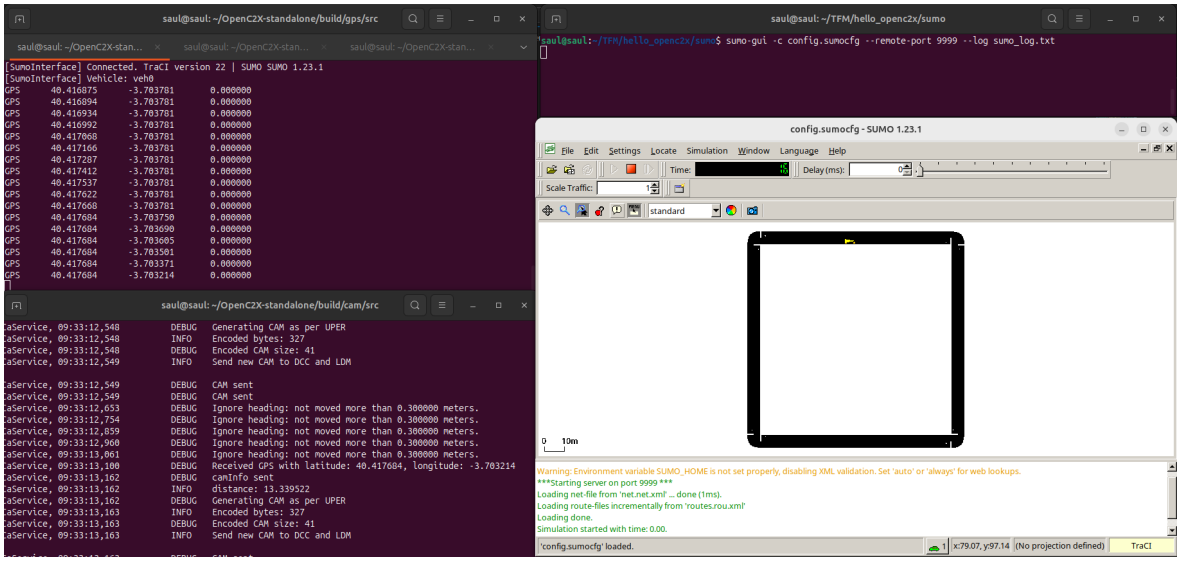

Figura 21: Hola Mundo con SUMO y los módulos GPS y CAM funcionando de forma integrada

El comportamiento observado demuestra que el flujo SUMO → SumoInterface → GpsService → CAM opera de manera coherente, garantizando que los

datos de movilidad generados en SUMO se reflejan correctamente en los mensajes transmitidos por OpenC2X.

4.6 Escenario V2V multivehículo

Una vez validada la integración básica SUMO ⟷ OpenC2X con un único nodo, se procedió a extender el entorno para ejecutar un escenario *V2V* con dos vehículos circulando simultáneamente. El objetivo de esta fase fue verificar el flujo completo de información entre nodos, la generación de mensajes CAM en paralelo y la correcta recepción de dichos mensajes por parte de los módulos internos de OpenC2X (DCC, LDM y CAMService).

Configuración del escenario

Para esta prueba se generó una red sintética en SUMO formada por un trazado cuadrado de 100 m de lado, y se definieron dos rutas independientes recorriendo el mismo circuito en sentido horario. Los vehículos `veh0` y `veh1` se configuraron con parámetros idénticos de aceleración, velocidad y longitud, y se inició la simulación con ambos circulando de forma sincronizada.

El archivo `two_vehicles.rou.xml` establece las rutas y el instante de salida de cada nodo, mientras que en el fichero `two_vehicles.sumocfg` se ha definido la configuración general del escenario, incluyendo la activación del servidor TraCI en el puerto por defecto (9999).

Ejecución del escenario

La simulación se lanzó mediante el script `runv2v.sh`, que arranca SUMO y los módulos de OpenC2X dentro de una sesión de `tmux`. Este script organiza los servicios en paneles independientes para facilitar su monitorización simultánea (Figuras 23 y 24).

En este escenario intervienen los siguientes módulos:

- **GPS Service**: recibe la posición simulada desde *SumoInterface*, construye el mensaje GPS en formato Protobuf y lo envía al resto de módulos. Este servicio garantiza que cada vehículo disponga de una posición coherente con su trayectoria en SUMO.

- **CAM Service (Cooperative Awareness Message)**: genera periódicamente los mensajes CAM a partir de los datos de posición y velocidad proporcionados por el módulo GPS. Estos mensajes representan la información esencial intercambiada entre vehículos en un sistema V2V.
- **DCC (Decentralized Congestion Control)**: regula la frecuencia de emisión de CAM para evitar saturación del canal. En este escenario controla la aceptación o descarte de mensajes según la disponibilidad de tokens.
- **LDM (Local Dynamic Map)**: almacena los CAM recibidos de otros vehículos junto con su información temporal, manteniendo una vista local del entorno dinámico.

Información mostrada en cada panel

Las Figuras 23 y 24 resumen la ejecución completa. Cada panel corresponde a un módulo distinto:

- **Panel izquierdo: GPS**. Se muestran las coordenadas actualizadas de `veh0` y `veh1` en cada tick de SUMO. Las posiciones reportadas (latitud y longitud) coinciden exactamente con los datos obtenidos vía TraCI.
- **Panel superior derecho: Generación de CAM**. Aquí se aprecia la creación de mensajes CAM para ambos vehículos. Además, el sistema aplica reglas de supresión, mostrando líneas como:

  ```
  Ignore heading: not moved more than 0.300000 meters
  ```

  Esto confirma que el módulo CAM evita transmitir cuando el desplazamiento desde el último CAM es inferior al umbral de 0.3 m. Además, puede verificarse que cada CAM incorpora exactamente las coordenadas (latitud y longitud) procesadas previamente por el módulo GPS, lo que demuestra que el flujo SUMO → GPS → CAM funciona correctamente y sin pérdidas de información.
- **Panel central derecho: Actividad del módulo DCC**.
  En esta ventana se muestra el comportamiento del módulo *Decentralized Congestion Control* (DCC), encargado de regular la carga

del canal y evitar congestiones cuando aumentan las transmisiones CAM.

Los registros de las Figuras 23 y 24 permiten observar tres tipos de eventos relevantes:

- **Recepción de CAM**: líneas como `CAM received` indican que el módulo DCC está recibiendo correctamente los mensajes procedentes de ambos vehículos.
- **Control de la cola de prioridad**: mensajes del tipo `AC 0: received and dropped CAM 2, queue full` muestran cómo DCC aplica las reglas eliminando paquetes cuando la cola asociada a la Access Category 0 alcanza su límite, reproduciendo así condiciones reales de congestión.
- **Actualización de tokens**: entradas como `AC 1: added token ->available tokens: 1` reflejan el mecanismo de *token bucket* utilizado por DCC para controlar la tasa de envío permitida por categoría de tráfico.

Estos mensajes confirman que el módulo DCC está procesando y filtrando los CAM recibidos, aplicando el control de congestión definido por ETSI y gestionando adecuadamente las colas internas de transmisión.

- **Panel inferior derecho: Actividad del módulo LDM**.
  En este panel se muestra el funcionamiento del *Local Dynamic Map* (LDM), el módulo encargado de almacenar la información recibida y mantener una base de datos local coherente sobre el entorno.

  Los registros muestran:

  - **Recepción de información de control DCC**: `dccInfo received`, lo que indica que LDM integra también los eventos informativos publicados por DCC.
  - **Recepción de CAM**: entradas como `camInfo received` y `CAM received` demuestran que LDM está almacenando correctamente los mensajes CAM emitidos por ambos vehículos.

  El comportamiento observado en esta ventana verifica que el módulo LDM mantiene un registro coherente de los estados de los vehículos

del entorno, funcionando como capa de agregación y memoria local dentro de OpenC2X.

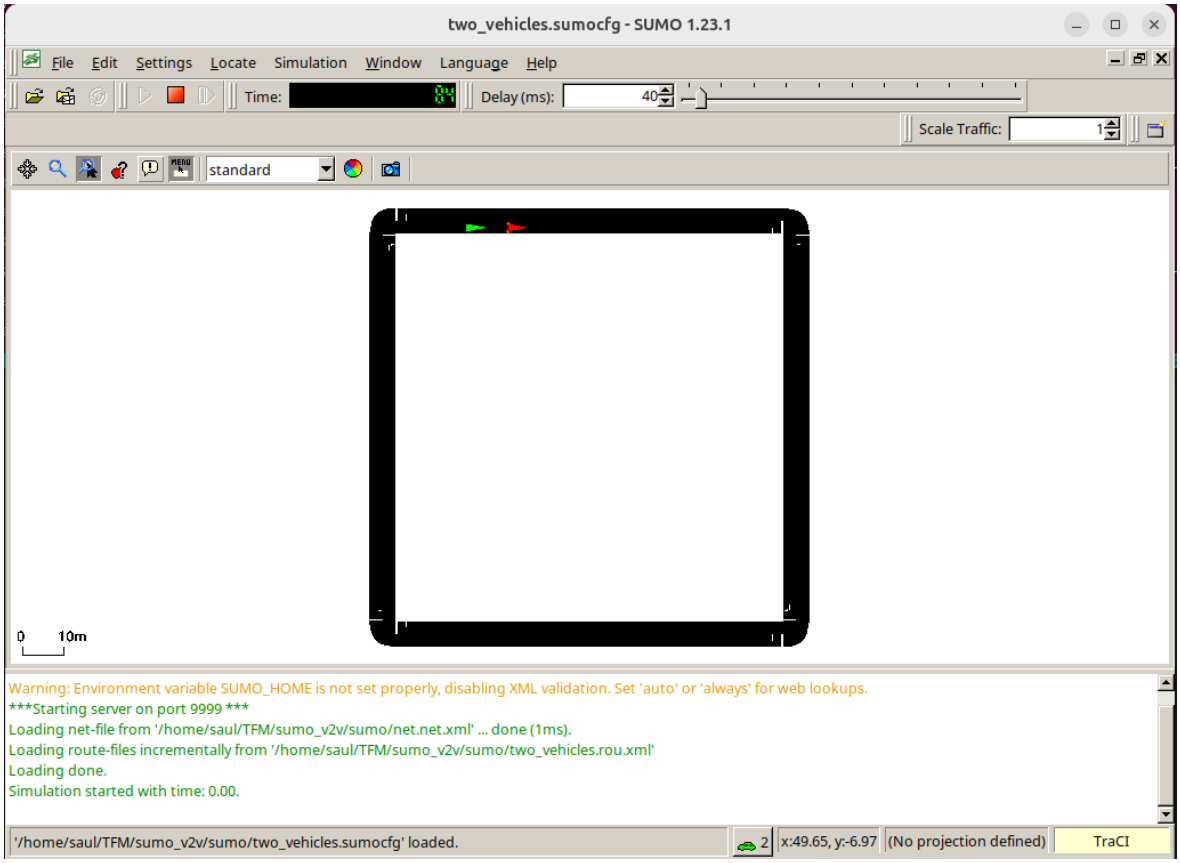

Figura 22: Escenario V2V con dos vehículos ejecutándose en SUMO.

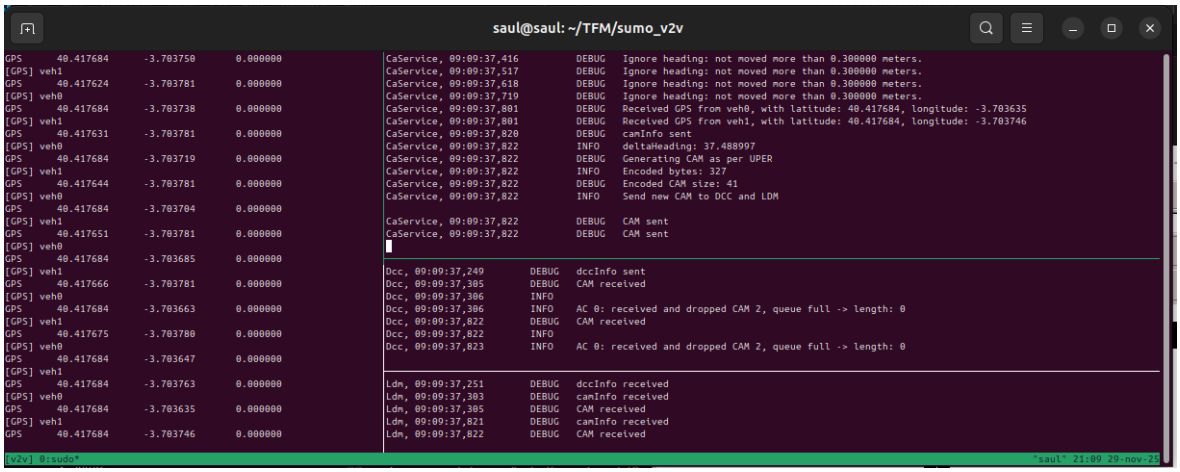

Figura 23: Ejecución conjunta de SUMO y OpenC2X en el escenario V2V multi-vehículo. Se observan los mensajes GPS, CAM, DCC y LDM.

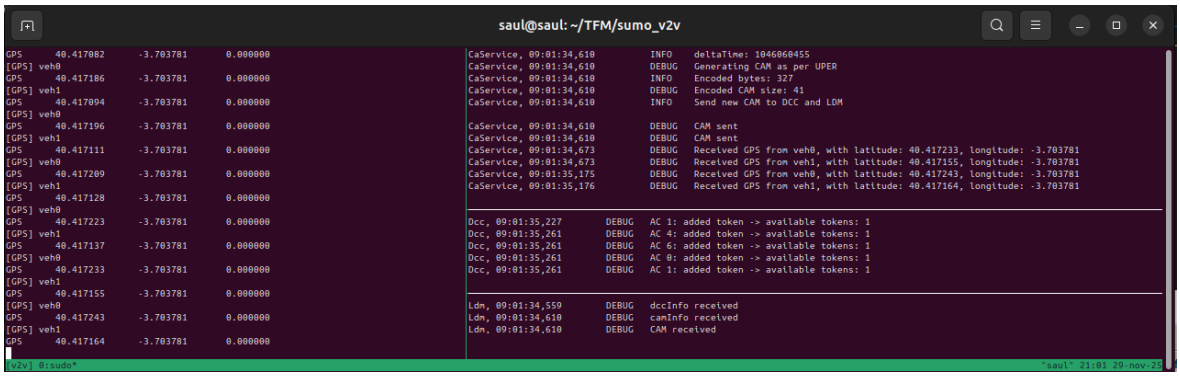

Figura 24: Ejecución conjunta de SUMO y OpenC2X en el escenario V2V multi-vehículo. Se observan los mensajes GPS, CAM, DCC y LDM.

Validación del correcto funcionamiento

A partir de esta prueba puede afirmarse que:

- La posición de `veh0` y `veh1` fluye correctamente desde SUMO hasta el módulo CAM.
- Las coordenadas incluidas en los CAM coinciden exactamente con las coordenadas mostradas en el módulo GPS.
- Los módulos DCC y LDM reciben, procesan y almacenan cada mensaje sin pérdidas.
- La comunicación V2V entre ambos vehículos queda establecida de forma completa y coherente.

Esta prueba confirma la validez del diseño implementado.

4.7 Integración V2I mediante detección de semáforo

La integración inicial entre SUMO↔OpenC2X permitió validar la comunicación V2V mediante mensajes CAM. En esta sección se amplía el sistema para incorporar comunicación *Vehicle-to-Infrastructure* (V2I), permitiendo que el vehículo detecte un semáforo en fase roja y genere un mensaje DENM (*Decentralized Environmental Notification Message*) indicando una situación de peligro.

Esta funcionalidad representa un caso de uso realista en sistemas cooperativos CITS (Cooperative Intelligent Transport Systems), donde la infraestructura proporciona información contextual al vehículo para mejorar la seguridad vial.

Creación del semáforo en SUMO

El primer paso consistió en incorporar un semáforo real a la red construida con `netgenerate`. Para ello, se editó la intersección `A1` mediante `netedit`, lo que permitió definir manual y visualmente el ciclo completo del semáforo [19]. En SUMO, cada semáforo se describe como una secuencia de *fases*, donde cada fase especifica:

- La **duración** en segundos.
- La **cadena de estado**, formada por caracteres que representan cada corriente de tráfico.
- La transición hacia la fase siguiente.

Los caracteres utilizados por SUMO son los siguientes:

- `G`: verde (*go*)
- `g`: verde protegiendo otros movimientos (*protected green*)
- `y`: amarillo
- `r`: rojo

Cada posición dentro de la cadena corresponde a un *link* del cruce (entradas/salidas específicas del nodo). En la Figura 25 se muestra el ciclo diseñado para la intersección, compuesto por seis fases que alternan el derecho de paso entre los ejes horizontal y vertical. El ciclo definido es el siguiente:

1. **Fase 0 (42 s):** `GggrrrGGg`. El eje horizontal recibe luz verde, mientras el vertical permanece en rojo.
2. **Fase 1 (3 s):** `yyyrrrGyy`. Transición de verde a rojo mediante fase amarilla en los movimientos que estaban activos.

3. **Fase 2 (1 s):** `rrrrrrrrr`. Fase de seguridad donde todos los movimientos permanecen en rojo.
4. **Fase 3 (42 s):** `rrrGGgGrr`. El eje vertical obtiene luz verde y el horizontal se mantiene en rojo.
5. **Fase 4 (3 s):** `rrryyyGrr`. Transición a rojo para el eje vertical.
6. **Fase 5 (1 s):** `rrrrrrrrr`. Segunda fase de seguridad antes de reiniciar el ciclo.

Las fases amarillas y las fases de seguridad (*all-red*) son necesarias para garantizar un comportamiento realista del cruce y evitar conflictos entre movimientos opuestos.

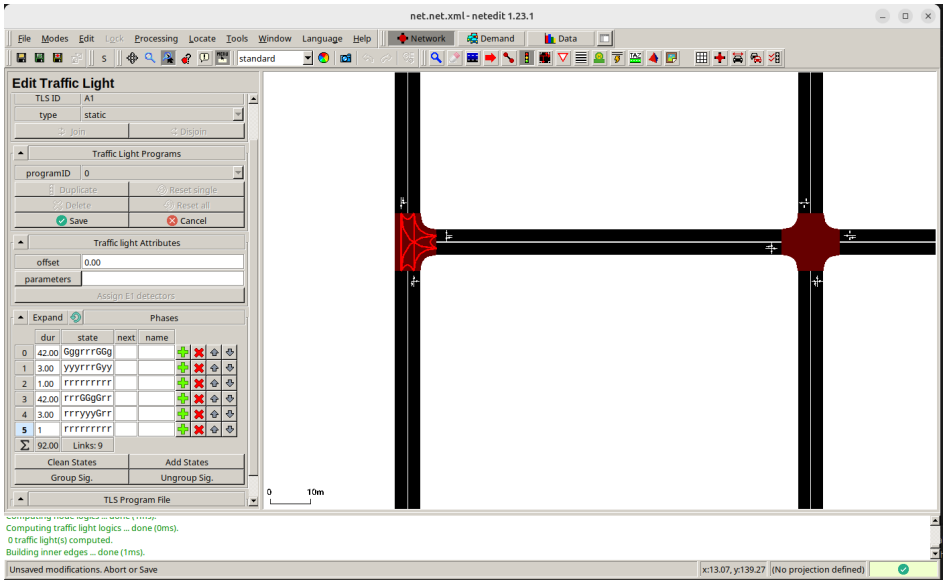

Figura 25: Definición del ciclo del semáforo en `netedit`.

SUMO expone en todo momento el estado del semáforo mediante la API TraCI, lo que permite que OpenC2X consulte en tiempo real qué fase está activa y utilice esta información para activar los eventos V2I.

Lógica de detección de semáforo en OpenC2X

Para habilitar la comunicación V2I se extendió el módulo `SumoInterface` añadiendo una rutina capaz de:

- Consultar el estado del semáforo mediante: `trafficlights.getRedYellowGreenState();`

- Obtener la posición del vehículo principal con `vehicle.getPosition();`
- Calcular la distancia entre el vehículo y el semáforo.
- Detectar si existe al menos un carril en fase roja.
- Activar un aviso interno cuando el vehículo se encuentra a menos de 25 metros.

La función nombrada `isRedLightAhead()` se encarga únicamente de detectar si el vehículo se aproxima a un semáforo en fase roja. Esta función no envía directamente ningún mensaje, sino que actualiza una variable interna (`m_pendingWarning`) que será procesada posteriormente por el módulo `GpsService`, encargado de generar los *triggers* externos del sistema OpenC2X.

La siguiente imagen muestra la función principal de detección:

```
bool SumoInterface::isRedLightAhead()
{
    if (!m_connected)
        return false;

    auto tlsIds = m_traci.trafficlights.getIDList();
    if (tlsIds.empty())
        return false;

    auto vehs = m_traci.vehicle.getIDList();
    if (vehs.empty())
        return false;

    std::string vehId = vehs[0];
    auto vehPos = m_traci.vehicle.getPosition(vehId);

    const double maxDist = 25.0;

    for (const auto& tlID : tlsIds) {
        std::string state = m_traci.trafficlights.getRedYellowGreenState(tlID);
        bool isRed = (state.find('r') != std::string::npos);

        auto tlPos = m_traci.junction.getPosition(tlID);

        double dx = vehPos.x - tlPos.x;
        double dy = vehPos.y - tlPos.y;
        double dist = std::sqrt(dx*dx + dy*dy);

        std::cout << "[V2I] TL " << tlID << " state=" << state
                  << " dist=" << dist << std::endl;

        if (isRed && dist < maxDist) {
            std::cout << "[V2I] WARNING: Red light detected at " << tlID << "\n";
            m_pendingWarning = true;
            return true;
        }
    }
    return false;
}
```

Figura 26: Función de detección del semáforo.

Durante la simulación, la consola muestra la detección con trazas como la de la Figura 27.

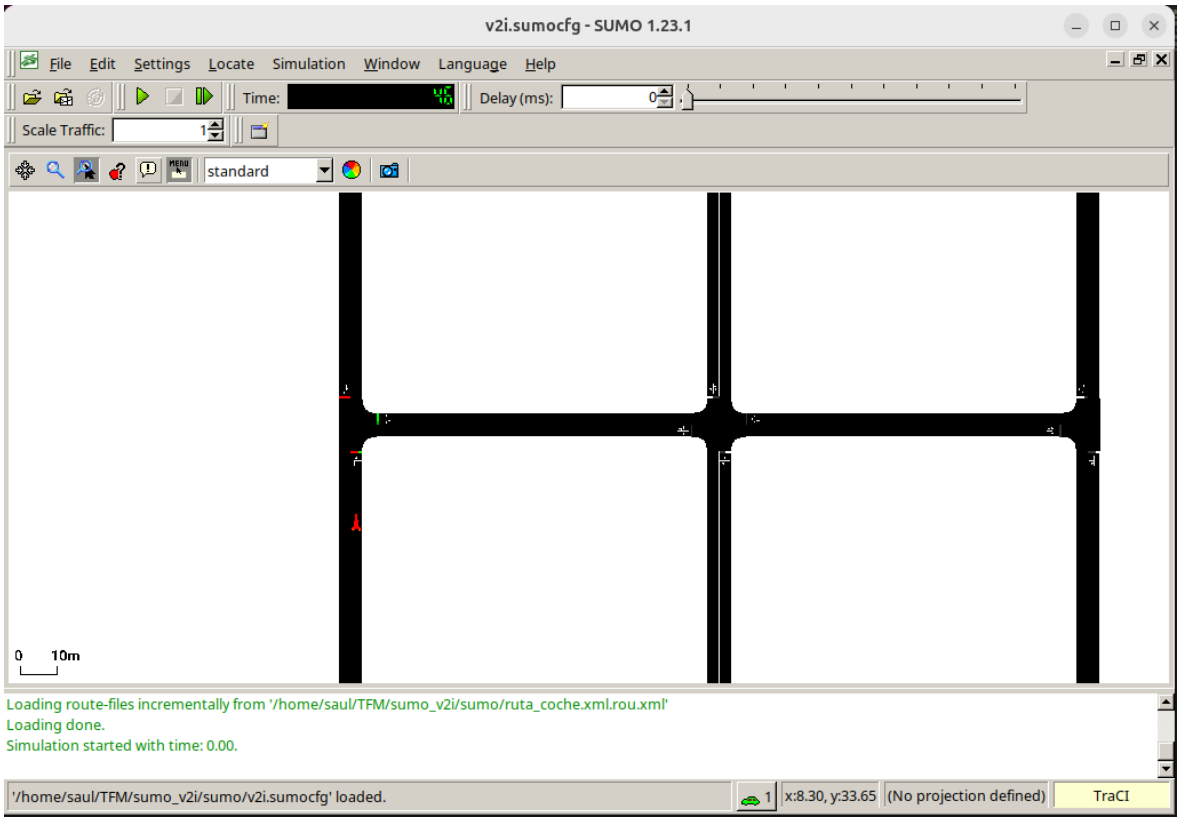

Figura 27: Detección del vehículo aproximándose al semáforo en rojo.

Activación del mensaje DENM desde el módulo GPS

Tras analizar los módulos disponibles en OpenC2X, se determinó que GpsService era el módulo más adecuado para activar los eventos externos, ya que recibe actualizaciones de SUMO en cada *tick*.

Es importante destacar que, en la implementación original de OpenC2X, el módulo `GpsService` no generaba ningún tipo de *trigger* ni activaba eventos externos. Su funcionalidad se limitaba al envío periódico de coordenadas GPS hacia los módulos CAM y DENM.

La integración V2I realizada en este trabajo ha permitido activar eventos DENM a partir de la información procedente de la infraestructura. Para ello, se añadió a `GpsService` la lógica necesaria para:

- Consultar a `SumoInterface` si el vehículo se aproxima a un semáforo en rojo.

- Interpretar esta condición como un evento de peligro.
- Enviar un *trigger* al módulo DENM utilizando el puerto 1111.
- Delegar en DENM la generación del mensaje cooperativo (DENM).

Para permitir que la infraestructura enviase avisos de tipo V2I, fue necesario modificar `GpsService` añadiendo un canal de comunicación adicional hacia el módulo `DENM`.

En concreto, se creó un objeto `CommunicationSender` persistente, inicializado en el constructor de GpsService y asociado al puerto **1111**, que es el puerto utilizado por `DenService` para recibir eventos externos (*triggers*):

```
m_triggerSender = new CommunicationSender("1111", *mLogger);
```

De este modo, cuando `SumoInterface` detecta que un vehículo se aproxima a un semáforo en rojo, marca un aviso pendiente. En el ciclo de ejecución de `GpsService`, dicho aviso se envía realmente a `DENM` mediante el siguiente fragmento:

```
if (m_sumoInterface->hasPendingWarning()) {
    m_triggerSender->send("TRIGGER", "RED_LIGHT");
    m_sumoInterface->clearPendingWarning();
}
```

Figura 28: Envia un trigger cuando hay un aviso pendiente.

El puerto **1111** es el receptor definido por `DenService` para la recepción de mensajes externos etiquetados como `TRIGGER`. Una vez recibido, el módulo `DENM` genera un nuevo mensaje de alarma tipo `DENM`, lo codifica y lo envía a `DCC`, que gestiona el acceso al canal radio. Finalmente, el mensaje queda registrado en el `LDM`, cerrando así el flujo completo de comunicación V2I.

Resultados de la integración V2I

La Figura 29 muestra el funcionamiento completo del sistema:

- Detección del semáforo en rojo por parte del vehículo.

- Activación del evento interno en `SumoInterface`.
- Envío del `TRIGGER RED_LIGHT` desde GPS.
- Recepción del trigger por parte del módulo DENM.
- Generación y envío del mensaje DENM.
- Recepción y procesamiento por parte de DCC y LDM.

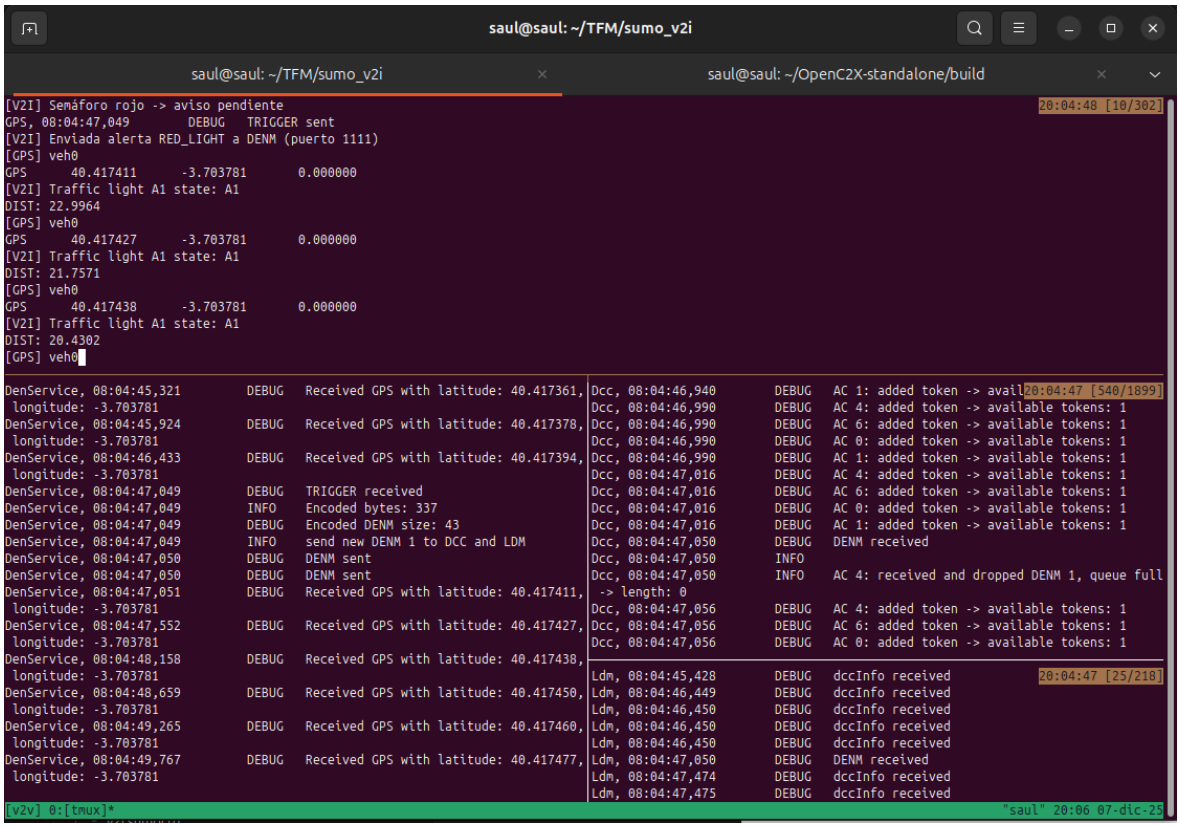

Figura 29: Flujo completo de comunicación V2I: generación y envío de un mensaje DENM.

La integración realizada demuestra que OpenC2X es capaz de procesar eventos de la infraestructura mediante SUMO, generando mensajes DENM de forma automática y coherente con el estándar ETSI ITS-G5.

5 Evaluación del sistema

En este capítulo se presenta la evaluación del entorno integrado SUMO ↔ OpenC2X desarrollado en el proyecto. El objetivo de esta fase es validar el funcionamiento de los módulos de comunicación, comprobar la coherencia temporal entre SUMO y OpenC2X y analizar métricas fundamentales como la latencia, la tasa de entrega de mensajes (PDR) y la estabilidad operativa del sistema en distintos escenarios simulados.

La evaluación se estructura en dos partes. En primer lugar, se analizan los experimentos de comunicación V2V mediante mensajes CAM, verificando la correcta sincronización entre movilidad y generación de mensajes. En segundo lugar, se estudian los experimentos de comunicación V2I, donde un nodo fijo (semáforo) genera eventos DENM desencadenados por el estado del tráfico. Ambos análisis permiten validar que el sistema funciona de forma robusta y reproducible en los distintos escenarios planteados.

5.1 Evaluación del sistema V2V

El objetivo de esta sección es analizar el comportamiento del sistema integrado SUMO↔OpenC2X bajo distintos niveles de carga vehicular. La evaluación se ha centrado en estudiar la estabilidad temporal del sistema, los intervalos reales de generación de CAM y la latencia interna entre el módulo CAM (emisión) y el módulo LDM (recepción).

Es importante destacar que, en este proyecto, OpenC2X se ejecuta en un único nodo software que concentra simultáneamente los servicios GPS, CAM, DCC y LDM. En esta arquitectura centralizada, SUMO proporciona la movilidad de todos los vehículos del escenario, pero el módulo CAM únicamente genera mensajes CAM para el vehículo local asociado al `node_id`. El resto de vehículos no ejecutan una instancia independiente de OpenC2X, sino que actúan como carga computacional adicional en la simulación.

Por tanto, las métricas obtenidas en este capítulo no representan la latencia de transmisión ITS-G5, sino la **latencia de procesamiento interna** entre los módulos CAM y LDM dentro del mismo nodo, así como la estabilidad temporal del pipeline SUMO → GPS → CAM → LDM. Las métricas obtenidas caracterizan el funcionamiento interno del pipeline CAM→LDM, y permiten evaluar su comportamiento temporal en un en-

torno controlado de simulación.

5.1.1 Metodología de evaluación

Para cada escenario (10, 20, 50 y 100 vehículos) se ejecutó el sistema utilizando un script que lanzaba SUMO y OpenC2X de forma sincronizada. La salida del módulo CAM y la del módulo LDM fueron redirigidas a ficheros `log`, permitiendo un análisis posterior reproducible.

El análisis se realizó mediante un script en Python que extrae los eventos `METRIC_CAM_SEND` y `METRIC_CAM_RECV` generados explícitamente en el código fuente. Cada evento incluye un timestamp en milisegundos generado a partir de funciones añadidas en la clase `Utils`. Esto ha permitido calcular:

- **Latencia interna**: diferencia entre envío de CAM y recepción interna en LDM.
- **PDR interno**: porcentaje de CAM emitidos que llegan al módulo LDM.
- **Intervalo CAM**: tiempo real transcurrido entre mensajes CAM consecutivos.

Dado que el escenario de 5 vehículos era demasiado pequeño y no generaba carga significativa, se descartó del análisis comparativo final.

5.1.2 Problemas iniciales en el escenario de 100 vehículos

Durante las primeras pruebas con 100 vehículos, SUMO no logró insertar todos los vehículos definidos en el fichero `routes.rou.xml`, estabilizándose alrededor de 79 vehículos activos. Este comportamiento no se debía a colisiones, sino a que la red original (segmentos de 100 m) no disponía de espacio suficiente para realizar inserciones seguras.

Para resolverlo, se regeneró la red con segmentos de 200 m mediante:

```
netgenerate --grid --grid.number=2 --length=200
--output-file=net.net.xml
```

Con esta ampliación, SUMO pudo insertar los 100 vehículos sin saturación y el escenario pudo ejecutarse correctamente.

5.1.3 Resultados experimentales

En la Tabla 1 se resumen los resultados obtenidos para los escenarios analizados, tanto con DCC activado como desactivado.

Vehículos	DCC	Lat. media (ms)	Lat. min	Lat. máx	Intervalo (ms)	PDR (%)
10	Si	1.50	0	12	861.17	100
10	No	1.25	0	4	910.00	100
20	Si	1.40	0	16	836.51	100
20	No	859.94	101	1113	857.46	100
50	Si	1.76	0	10	678.25	100
50	No	749.88	101	1119	747.39	100
100	Si	1.42	0	37	646.93	100
100	No	629.41	100	1627	637.52	100

Tabla 1: Resumen de métricas V2V por escenario y configuración.

Los resultados muestran una tendencia clara: al aumentar el número de vehículos en SUMO, el intervalo medio entre CAMs disminuye ligeramente debido al incremento en la actividad del sistema, mientras que la latencia interna se mantiene estable y baja (1-2 ms) en todos los escenarios con DCC activado. Esto confirma la robustez del pipeline interno entre los módulos CAM y LDM.

En el caso de las pruebas con DCC desactivado, la latencia aumenta significativamente en los escenarios de 20, 50 y 100 vehículos, alcanzando valores de cientos de milisegundos. Esto sugiere que la ausencia de regulación provoca más carga interna y ralentiza el procesamiento.

5.1.4 Visualización comparativa de resultados

Para complementar los valores resumidos en el Cuadro 1, se incluyen a continuación las tres gráficas principales obtenidas a partir del análisis de los escenarios de 10, 20, 50 y 100 vehículos, tanto con DCC activado como desactivado.

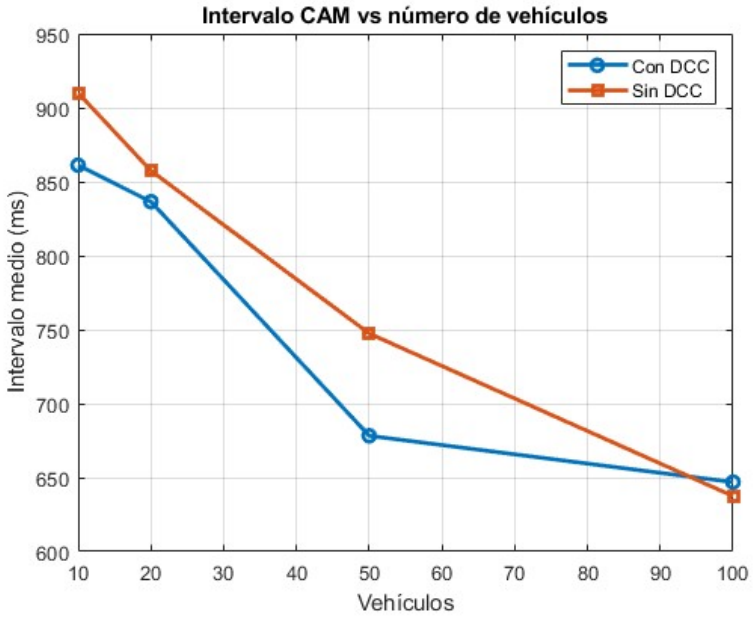

Figura 30: Intervalo medio entre CAMs en función del número de vehículos.

La Figura 30 muestra que el intervalo medio entre CAMs disminuye a medida que aumenta la carga computacional. Tanto con DCC como sin él se observa esta tendencia, aunque los valores son ligeramente menores cuando DCC está activado.

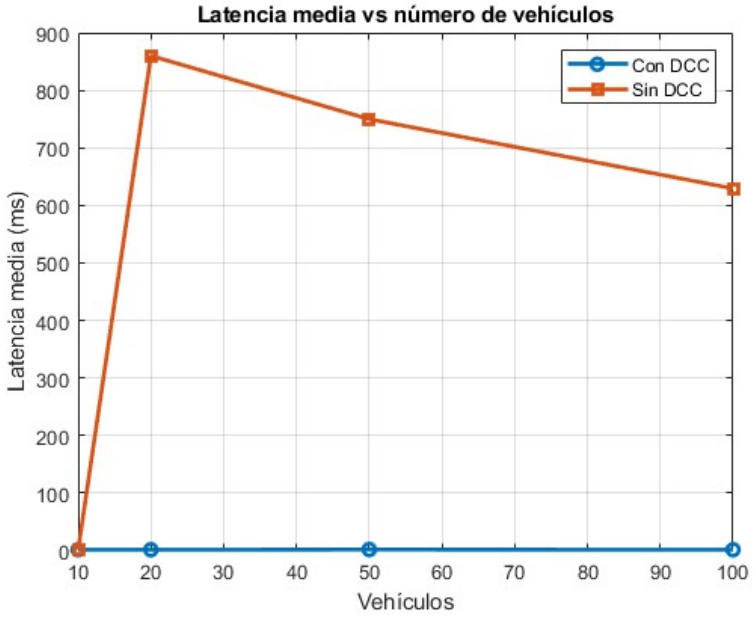

Figura 31: Latencia interna media CAM → LDM en función del número de vehículos.

En la Figura 31 se aprecia una diferencia muy marcada entre los escenarios con y sin DCC. Cuando DCC está activado, la latencia interna permanece estable alrededor de 1–2 ms, incluso para 100 vehículos. Sin embargo, al desactivar DCC, la latencia crece de forma significativa en escenarios con 20, 50 y 100 vehículos, alcanzando valores de varios cientos de milisegundos.

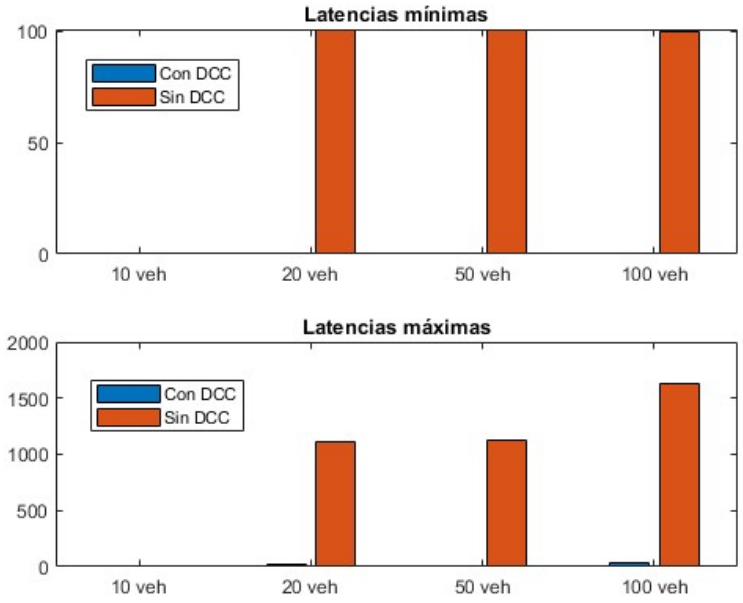

Figura 32: Latencias mínimas y máximas internas registradas en cada escenario.

Finalmente, la Figura 32 muestra que, cuando el DCC está activado, tanto la latencia mínima como la máxima se mantienen en valores muy bajos (en el orden de unos pocos milisegundos) para todos los escenarios. En cambio, al desactivar el DCC, las latencias mínimas se sitúan en torno a 100 ms y las latencias máximas llegan a superar los 1600 ms en los escenarios de mayor carga. Este comportamiento refuerza el papel del DCC como un mecanismo esencial para evitar la saturación interna del sistema y mantener un tiempo de procesamiento acotado.

5.1.5 Escenario multivehículo con 100 nodos

La Figura 33 muestra la ejecución del escenario con 100 vehículos en la red ampliada. En la parte superior se visualizan las trazas generadas por los módulos principales de OpenC2X: el módulo CAM produce

los eventos METRIC_CAM_SEND, que registran cada mensaje CAM emitido por el vehículo local, mientras que el módulo LDM muestra los eventos METRIC_CAM_RECV, correspondientes a la recepción interna de dichos mensajes. En la parte inferior se observa la simulación de SUMO con 100 vehículos, cuya movilidad sirve como entrada para las actualizaciones GPS procesadas por OpenC2X.

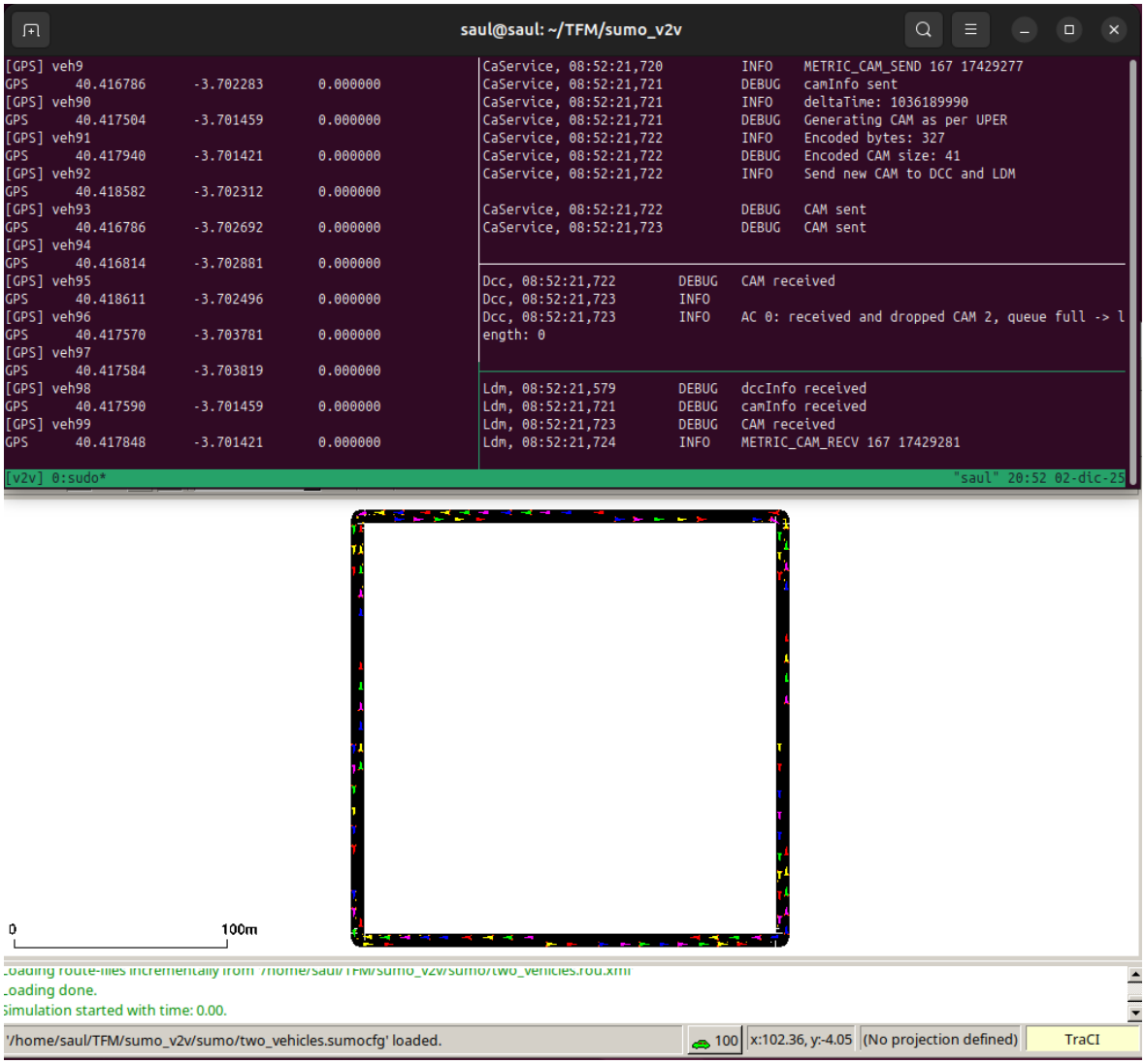

Figura 33: Ejecución del escenario de 100 vehículos en SUMO↔OpenC2X.

Es importante señalar que, aunque se reciben los datos GPS de todos los vehículos del escenario, sólo se genera un CAM por tick para el vehículo local, según el parámetro node_id. El resto actúa como carga para la simulación pero no como nodos V2V independientes.

5.1.6 Discusión de resultados

Los resultados obtenidos permiten extraer varias conclusiones relevantes:

- El pipeline SUMO → GPS → CAM → LDM funciona correctamente y se mantiene estable en todos los escenarios.
- La latencia interna se mantiene en valores muy bajos con DCC activado, incluso en escenarios de 100 vehículos.
- El impacto del número de vehículos afecta principalmente al intervalo entre CAMs, especialmente en escenarios de alta densidad.
- La ausencia de DCC incrementa significativamente la latencia interna en escenarios medios y altos.

5.2 Evaluación del sistema V2I

La segunda parte de la evaluación se centra en analizar el funcionamiento del sistema ante eventos *V2I* (Vehicle-to-Infrastructure), en los que un elemento fijo de la infraestructura (en este caso un semáforo) genera un mensaje DENM cuando detecta que un vehículo se aproxima mientras la fase se encuentra en rojo. Este mecanismo se desarrolló específicamente para este proyecto integrando la API *TraCI* con el módulo *GpsService*, permitiendo que la infraestructura active eventos en tiempo real.

5.2.1 Metodología

Para evaluar la comunicación *V2I* se realizaron pruebas basadas en el número de semáforos activos en el escenario. Cada semáforo genera uno o varios *triggers* cuando el vehículo se aproxima en fase roja, y cada uno de ellos debe transformarse en un mensaje DENM procesado por el módulo *LDM*.

Con el fin de medir la latencia extremo a extremo del flujo:

$$\text{Trigger (GPS)} \rightarrow \text{DENM} \rightarrow \text{LDM},$$

El análisis se realizó mediante un script en Python que extrae los eventos `METRIC_TRIGGER_SEND` y `METRIC_DENM_RECV` generados explícitamente en el código fuente de `GpsService` y `ldm`. Cada evento incluye un ti-

mestamp en milisegundos generado a partir de funciones añadidas en la clase `Utils`.

A partir de estos registros se calcularon la latencia, la tasa de entrega (PDR), la presencia de duplicados y la correspondencia exacta entre triggers y DENM.

5.2.2 Escenario de pruebas

El escenario consiste en una red urbana cuadrada generada con `netgenerate`, sobre la que se insertan varios semáforos configurados con ciclos fijos. El vehículo circula siguiendo una ruta predefinida en SUMO que le lleva a pasar por todos los semáforos. El número de elementos de la infraestructura se varió entre uno y cinco, generando distintos niveles de carga en los módulos *GpsService*, *DENM* y *LDM*.

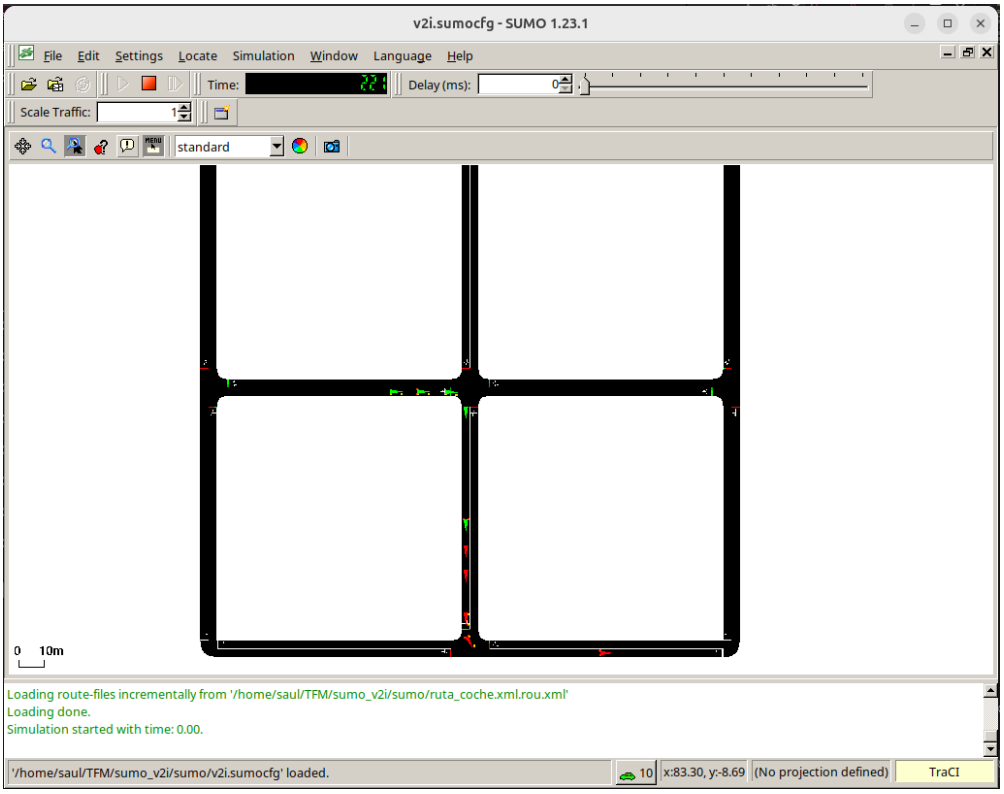

Figura 34: Escenario SUMO utilizado con 4 semáforos activos.

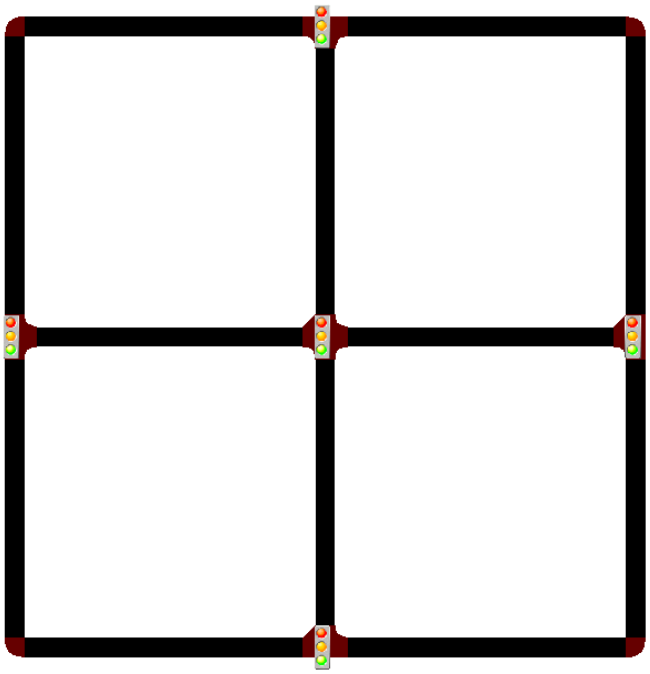

Figura 35: Distribución de los semáforos.

La Figura 35 muestra la disposición de los semáforos utilizados en el escenario V2I. Dependiendo del experimento, se activaron entre 1 y 5 de estos semáforos, lo que permitió evaluar cómo el sistema reaccionaba ante diferentes niveles de carga en la infraestructura. Cada semáforo genera un evento DENM independiente cuando el vehículo se aproxima durante una fase roja, lo que incrementa el número total de disparos a medida que se amplía el número de intersecciones activas.

Durante las pruebas se permitió que cada semáforo generase múltiples activaciones mientras el vehículo se mantenía dentro del umbral de distancia y con fase roja, para evaluar el comportamiento del sistema ante ráfagas de eventos.

5.2.3 Consideración sobre el número de vehículos

Además del estudio basado en el número de semáforos, se realizaron pruebas aumentando la movilidad en SUMO (10, 20, 50 y 100 vehículos). El objetivo era comprobar si la carga vehicular afectaba a la latencia o al PDR. No obstante, en este escenario el flujo *Trigger* → *DENM* → *LDM* sólo depende del vehículo que recibe la información del semáforo. Los vehículos adicionales no participan en la comunicación V2I, ya que el evento lo

genera únicamente el nodo fijo asociado al semáforo. Por este motivo, el número de vehículos no introduce carga adicional en el sistema ni afecta al flujo de procesamiento del evento. Los resultados confirmaron que la latencia y el PDR permanecen constantes independientemente del número de vehículos simulados.

5.2.4 Justificación del envío de múltiples triggers

Aunque sería posible emitir un único evento por semáforo, se decidió mantener la generación de todos los triggers detectados. Esto permite:

- Evaluar el sistema en condiciones de mayor carga.
- Verificar emparejamientos uno a uno entre trigger y DENM.
- Detectar duplicados, pérdidas o latencias anómalas.
- Comprobar la estabilidad del módulo LDM ante ráfagas de mensajes.

5.2.5 Resultados

La Tabla 2 resume los resultados obtenidos para 1, 2, 3, 4 y 5 semáforos con 10 vehículos circulando en el escenario.

Semáforos	Triggers	DENM	PDR (%)	Lat. media (ms)	Lat. max (ms)
1	32	32	100	2.41	5
2	62	62	100	2.81	9
3	90	90	100	2.91	9
4	144	143	99.31	2.68	9
5	192	192	100	2.76	15

Tabla 2: Resultados de latencia y PDR en el escenario V2I.

5.2.6 Análisis

Los resultados muestran una latencia muy baja y estable (2-3 ms) incluso cuando el número de semáforos aumenta significativamente. El PDR

permanece en el 100 % en todos los casos salvo para 4 semáforos, donde se detectó un único emparejamiento no válido debido a una latencia negativa puntual en el log.

No se detectaron duplicados, pérdidas reales ni inconsistencias en los identificadores. Esto confirma que el flujo *Trigger* → *DENM* → *LDM* es robusto, reproducible y estable bajo distintos niveles de carga de infraestructura.

Finalmente, los resultados muestran que el módulo *LDM* es capaz de gestionar ráfagas de eventos sin degradación temporal, y que el sistema completo presenta un comportamiento determinista y predecible en escenarios V2I.

6 Estudio de viabilidad

6.1 Alcance y suposiciones

El proyecto tiene como objetivo integrar **SUMO** y **OpenC2X** para simular la comunicación **V2V mediante mensajes CAM**, registrando métricas de latencia, tasa de entrega de paquetes y estabilidad del canal bajo distintos escenarios de tráfico.

Adicionalmente, se incorpora una extensión funcional de comunicación **V2I**, en la que la infraestructura (semáforos definidos en SUMO) puede generar eventos que desencadenan la transmisión de mensajes DENM por parte de OpenC2X.

Para garantizar un entorno controlado y reproducible, se establecen las siguientes suposiciones y condiciones de partida:

- El sistema se ejecutará en un único *host* (máquina virtual) con Ubuntu 24.04.
- Se empleará la pila ITS-G5 simulada, sin hardware radio real.
- Los escenarios se limitarán a densidad baja, media y alta con redes SUMO simples.
- No se considerará la capa de seguridad (PKI) ni mecanismos de autenticación.

Con estas condiciones, el proyecto es técnicamente viable usando únicamente software libre y recursos de hardware estándar. El principal reto identificado fue la integración con TraCI y la migración del código de OpenC2X a versiones modernas de Linux, ambos superados durante el desarrollo.

6.2 Herramientas utilizadas

Para el desarrollo e integración del sistema se han empleado exclusivamente herramientas de software libre y componentes accesibles, lo que facilita la reproducibilidad del proyecto. Las principales herramientas utilizadas son:

- **Sistema operativo:** Ubuntu 24.04 (máquina virtual).

- **Simulador de movilidad:** SUMO (`sumo`, `sumo-gui`).
- **API tiempo real:** TraCI, mediante un cliente C++ desarrollando el módulo `SumoInterface`.
- **Plataforma V2X:** OpenC2X (módulos CAM, DCC, LDM y GPS).
- **Mensajería interna:** ZeroMQ.
- **Lenguaje de programación:** `C++`.
- **Herramientas de compilación:** `gcc`, `clang`, `CMake`, `pkg-config`.
- **Dependencias:** `gpsd`, `libubox`, `uci`.
- **Scripts de automatización:** Bash (para la ejecución de módulos).
- **Análisis y validación de resultados:** Logs de OpenC2X y scritps en Python.

6.3 Plan técnico por fases

El desarrollo del sistema se ha llevado a cabo siguiendo una estrategia incremental, validando cada componente de forma independiente antes de integrarlo en el entorno final SUMO↔OpenC2X. El plan técnico se centró en los siguientes objetivos principales:

1. **Migración de OpenC2X a Ubuntu 24.04:** OpenC2X fue diseñado originalmente para Ubuntu 16.04, por lo que fue necesario adaptar su código, actualizar dependencias y resolver incompatibilidades con versiones modernas de *gcc*, *libzmq*, *gpsd*, *libubox* y *uci*.
2. **Integración TraCI ↔ OpenC2X:** Conexión entre `GpsService.cpp` y el nuevo módulo `SumoInterface`, lectura por *tick* de posición, velocidad y orientación.
3. **Escenarios SUMO:** Creación de redes sintéticas variando la densidad vehicular.
4. **Ejecución y métricas:** Obtención de latencia, tasa de entrega de mensajes y tasa CAM efectiva bajo distintos escenarios.
5. **Análisis y validación:** Evaluación de los resultados, comparación con referencias teóricas y documentación de las pruebas.

La Tabla 3 resume los principales riesgos técnicos detectados durante el proyecto, junto con su impacto, probabilidad y las acciones de mitigación aplicadas.

Riesgo	**Impacto**	**Prob.**	**Mitigación / Plan B**
Desfase temporal SUMO ↔ OpenC2X	Medio	Media	Usar `step-length` fijo, sincronizar la llamada a TraCI por tick y asegurar que la actualización del GPS se realice siempre con el mismo orden de ejecución. Validar desfases con escenarios multivehículo.
Inconsistencia entre la frecuencia de actualización de SUMO y la frecuencia de envío CAM	Medio	Media	Ajustar intervalos CAM; forzar actualización del GPS antes de cada CAM; registrar el orden de ejecución para la depuración y su posterior análisis.
Sobrecarga con muchos vehículos	Alto	Media	Escalar gradualmente; medir el uso de CPU y ajustar la frecuencia CAM.
Latencias elevadas sin DCC en escenarios densos	Alto	Media	Comparar los resultados con y sin DCC; mantener intervalos CAM constantes; analizar la degradación y ajustar los parámetros de transmisión.
Problemas de compilación y migración	Alto	Media	Fijar versiones del compilador y dependencias; automatizar la compilación mediante CMake; validar la instalación completa en Ubuntu 24.04.
Dependencia del rendimiento de la máquina virtual	Medio	Media	Asignar recursos adecuados; repetir pruebas para garantizar reproducibilidad.

Tabla 3: Riesgos técnicos identificados y medidas de mitigación.

Criterios de éxito

Para considerar completada la integración SUMO ⟷ OpenC2X y validar el correcto funcionamiento del sistema, se establecieron los siguientes criterios de éxito:

- **Sincronización estable** entre SUMO y OpenC2X, sin desfases superiores a un *tick* y manteniendo la coherencia temporal del pipeline SUMO → TraCI → GPS → CAM → LDM.
- **Envío y recepción continua de mensajes CAM** durante al menos cinco minutos de simulación sin interrupciones ni errores críticos.
- **Obtención de métricas reproducibles**, incluyendo latencia interna y tasa de entrega (PDR), en los tres escenarios definidos (baja, media y alta densidad vehicular).

El cumplimiento de estos criterios garantiza que el sistema integrado es estable, funcional y adecuado para la evaluación experimental desarrollada en el capítulo 5.

6.4 Viabilidad económica

El proyecto se desarrolla íntegramente con herramientas de software libre (OpenC2X, SUMO, Ubuntu 24.04, gpsd), por lo que no existen costes asociados a licencias. Asimismo, el hardware empleado consiste en un equipo físico estándar y una máquina virtual, por lo que no se requiere inversión adicional en infraestructura.

El principal coste del proyecto corresponde al tiempo de dedicación necesario para la adaptación del código de OpenC2X a un entorno moderno, la resolución de incompatibilidades, la integración con TraCI mediante el módulo `SumoInterface`, el diseño de los escenarios de movilidad y la realización de pruebas. Dada la complejidad técnica de estas tareas, se ha estimado un coste horario acorde al perfil de un ingeniero sénior.

La estimación económica se desglosa de la siguiente manera:

Concepto	Coste estimado (€)
Tiempo de desarrollo técnico (160 h × 30 €/h)	4800
Tiempo de documentación y análisis (100 h × 20 €/h)	2000
Consumo eléctrico aproximado	20
Hardware y licencias adicionales	0
Coste total estimado	**6820**

Tabla 4: Coste estimado del proyecto

El proyecto resulta económicamente viable, ya que el coste total se asocia exclusivamente al tiempo invertido en su desarrollo. El uso de herramientas abiertas reduce significativamente el presupuesto, manteniendo un entorno de trabajo potente y flexible sin necesidad de adquirir software o equipamiento especializado.

Retorno de la inversión (ROI)

Aunque el proyecto no está orientado a la explotación comercial directa, sí presenta un retorno económico claro desde el punto de vista de investigación, desarrollo e innovación (I+D). En particular:

- Reducir costes frente a pruebas con hardware real o infraestructura ITS-G5.
- Evaluar múltiples escenarios de movilidad sin riesgo para personas o vehículos.
- Experimentar con diferentes configuraciones V2X antes de invertir en equipamiento físico.
- Facilitar futuros proyectos de simulación, investigación o docencia.

Así, el proyecto presenta una relación coste-beneficio favorable, especialmente para universidades, centros de investigación y empresas que deseen explorar tecnologías V2X minimizando riesgos e inversión inicial.

6.5 Impacto medioambiental

El impacto medioambiental del proyecto es reducido, ya que todo el desarrollo se realiza en un entorno de simulación software y no requiere el uso de vehículos reales, antenas ITS-G5 ni equipos radioeléctricos. Esto evita el consumo de combustible y la generación de emisiones asociadas a pruebas físicas.

La ejecución del proyecto en una máquina virtual supone un consumo eléctrico limitado, comparable al uso habitual de un equipo informático. Asimismo, el uso de herramientas de software libre contribuye a la sostenibilidad tecnológica, ya que elimina la necesidad de adquirir hardware especializado.

Además, la posibilidad de repetir experimentos de forma automatizada y sin recursos adicionales permite disminuir el impacto energético del ciclo de desarrollo. Los escenarios de movilidad pueden evaluarse tantas veces como sea necesario sin incrementos significativos de consumo, algo que no sería posible en un entorno físico.

A nivel indirecto, este trabajo puede contribuir positivamente al medio ambiente, ya que la investigación en sistemas V2X permite mejorar la eficiencia del tráfico, reducir congestiones y disminuir emisiones derivadas del transporte urbano.

6.6 Planes de contingencia

Para garantizar la continuidad del proyecto ante posibles incidencias técnicas, se definieron distintos planes de contingencia:

- **Fallo del cliente TraCI o pérdida de sincronización SUMO ↔ OpenC2X**: Utilizar el modo *replay* con trazas previamente exportadas desde SUMO, lo que permite reproducir exactamente el escenario sin depender de la ejecución en tiempo real.
- **Tasa CAM efectiva baja**: Reducir temporalmente la frecuencia de emisión de mensajes o el tamaño del escenario.
- **Inestabilidad en la ejecución en tiempo real**: Ejecutar la simulación en modo no interactivo, registrando el comportamiento en los *logs* para su análisis posterior.

- **Errores de compilación o dependencias inconsistentes**: Revertir al entorno validado documentado (versión de Ubuntu 24.04, dependencias exactas y configuración CMake).
- **Problemas de rendimiento en la máquina virtual**: Aumentar recursos asignados (CPU y RAM).

7 Conclusiones y trabajos futuros

7.1 Conclusiones

El objetivo principal del Trabajo Final de Máster se ha alcanzado con éxito, logrando integrar SUMO y OpenC2X en un entorno de simulación reproducible que permite analizar el flujo completo SUMO → TraCI → GPS → CAM → LDM. Los resultados experimentales permiten extraer las siguientes conclusiones:

- **Integración estable SUMO↔OpenC2X.** La migración de OpenC2X a Ubuntu 24.04 y el desarrollo del módulo *SumoInterface* han permitido sustituir la fuente GPS real por la posición simulada, garantizando coherencia temporal y espacial.
- **Pipeline robusto.** El flujo SUMO → GPS → CAM → LDM. se mantiene estable en todos los escenarios evaluados.
- **Latencias internas muy bajas.** Con DCC activado, la latencia CAM → LDM se mantiene en 1-2 ms incluso con 100 vehículos.
- **Importancia del DCC.** Al desactivar DCC, la latencia interna aumenta de forma significativa, confirmando su papel como mecanismo esencial de regulación.
- **Extensibilidad hacia comunicaciones V2I.** Además del flujo V2V, se ha incorporado una extensión funcional que habilita la generación de mensajes DENM a partir de eventos procedentes de la infraestructura (p. ej., detección de un semáforo en rojo en SUMO). Las pruebas realizadas (incluyendo escenarios con múltiples semáforos) confirmaron el correcto funcionamiento del flujo SUMO → OpenC2X → DENM, obteniéndose latencias muy bajas (2-3 ms) y un PDR prácticamente del 100 %, demostrando que la arquitectura es compatible con la incorporación de funcionalidades V2I sin cambios estructurales en OpenC2X.
- **Reproducibilidad y bajo coste.** El uso de software libre y hardware estándar ha reducido el coste económico y además, facilita la replicación del entorno.

En conjunto, el sistema desarrollado demuestra que es posible integrar de forma eficiente un simulador de tráfico y una pila V2X en un mismo entorno, habilitando análisis controlados y repetibles.

7.2 Limitaciones

Aunque los objetivos se han cumplido, existen limitaciones inherentes al enfoque utilizado:

- El sistema emplea un único nodo OpenC2X; el resto de vehículos no ejecutan módulos V2X reales.
- La latencia evaluada corresponde únicamente al procesamiento interno CAM→LDM.
- Los escenarios SUMO son sintéticos y de complejidad limitada.
- No se han considerado mecanismos de seguridad.

7.3 Trabajos futuros

A partir del trabajo realizado, se identifican varias líneas de mejora:

- Ejecución multinodo de OpenC2X para evaluar métricas V2V reales.
- Extender el mecanismo desarrollado para semáforos a otros elementos de infraestructura (señales dinámicas, detección de obras, atascos, etc.).
- Integrar el sistema con frameworks como Veins o MOSAIC para incorporar simulaciones de radio realistas mediante OMNeT++ o ns-3.
- Creación de escenarios SUMO más complejos o basados en mapas reales.
- Estudiar mecanismos de detección de intrusiones en el entorno V2X, incorporando módulos de IDS (Intrusion Detection Systems) que permitan identificar comportamientos anómalos o mensajes no legítimos en la red [20].
- Estudio del comportamiento del DCC en escenarios dinámicos.

- Desarrollo de una interfaz gráfica de monitorización del sistema.
- Automatización completa de la generación de escenarios, ejecución y análisis.

8 Glosario

API (Interfaz de Programación de Aplicaciones): Conjunto de reglas y protocolos que permite a diferentes programas software comunicarse entre si; en este proyecto, principalmente la interfaz TraCI de SUMO.

CAM (Cooperative Awareness Message): Mensaje V2X periódico que transmite información básica de estado del vehículo, como posición, velocidad y orientación.

DCC (Decentralized Congestion Control): Mecanismo encargado de regular el acceso al canal ITS-G5 evitando congestión.

DENM (Decentralized Environmental Notification Message): Mensaje V2X generado ante un evento concreto (como peligro o aviso de infraestructura).

GPS Service (GpsService): Servicio de OpenC2X encargado de proporcionar a los nodos la posición y el tiempo de referencia. En esta memoria se modifica para recibir posiciones desde SUMO y para generar los eventos V2I.

ITS (Intelligent Transportation Systems): Conjunto de tecnologías aplicadas al transporte para mejorar la eficiencia y la seguridad mediante comunicaciones vehículo-vehículo y vehículo-infraestructura.

ITS-G5: Tecnología de comunicaciones basada en IEEE 802.11p para sistemas ITS. Es el estándar asumido por OpenC2X en la simulación.

LDM (Local Dynamic Map): Base de datos local donde cada nodo OpenC2X almacena la información recibida de otros vehículos o de la infraestructura.

OpenC2X: Framework de comunicaciones V2X de código abierto empleado para simular servicios ITS-G5. En este TFM se adapta a Ubuntu 24.04 y se amplía para integrar SUMO mediante TraCI.

PDR (Packet Delivery Ratio): Porcentaje de mensajes recibidos respecto a los enviados.

SUMO (Simulation of Urban MObility): Simulador de movilidad vehicular utilizado para modelar el tráfico.

TraCI (Traffic Control Interface): Interfaz de control remoto que permite a SUMO comunicarse con aplicaciones externas mediante TCP.

V2I (Vehicle-to-Infrastructure): Comunicación entre vehículo e infraestructura.

V2V (Vehicle-to-Vehicle): Comunicación cooperativa entre vehículos.

V2X (Vehicle-to-Everything): Término general que engloba V2V, V2I y otras modalidades de comunicación vehicular.

Referencias

[1] DGT. *Siniestros de tráfico en España 2024*. [en línea]. URL: https://www.dgt.es/comunicacion/notas-de-prensa/20250110-2024-record-historico-desplazamientos-en-carretera-mientras-se-estabiliza-el-numero-de-fallecidos-en-siniestros-de-trafico/.

[2] Kitaeg Lim; Seonghyun Jang; Byoungman An; Sanghun Yoon. *V2X Communication Technology Trends in South Korea*. [descargado]. URL: https://ieeexplore.ieee.org/document/10457172.

[3] Sven Laux; Gurjashan Singh Pannu; Stefan Schneider; Jan Tiemann; Florian Klingler; Christoph Sommer. *Demo: OpenC2X — An open source experimental and prototyping platform supporting ETSI ITS-G5*. [descargado]. URL: https://ieeexplore.ieee.org/document/7835955.

[4] Haibo Zhou; Wenchao Xu; Jiacheng Chen; Wei Wang. *Evolutionary V2X Technologies Toward the Internet of Vehicles: Challenges and Opportunities*. [descargado]. URL: https://ieeexplore.ieee.org/document/8967260.

[5] M. Garcia et al. IEEE Communications Surveys y Tutoriales. *A Tutorial on 5G NR V2X Communications*. [descargado]. URL: https://arxiv.org/ftp/arxiv/papers/2102/2102.04538.pdf.

[6] Florian Klingler. *OpenC2X*. [en línea]. URL: http://www.ccs-labs.org/software/openc2x.

[7] José Luis Martín Robles. «Emulación de un sistema de prevención de accidentes de tráfico en cruces mediante comunicación vehículo - infraestructura». [descargado]. Tesis de maestría. Universidad de Valladolid, 2023. URL: https://uvadoc.uva.es/bitstream/handle/10324/62683/TFM-G1791.pdf.

[8] Manabu Tsukada; Takaharu Oi; Akihide Ito; Mai Hirata; Hiroshi Esaki. *AutoC2X: Open-source software to realize V2X cooperative perception among autonomous vehicles*. [descargado]. URL: https://ieeexplore.ieee.org/document/9348525.

[9] Centro Aeroespacial Alemán (DLR) y otros. *SUMO*. [en línea]. URL: https://sumo.dlr.de/docs/index.html.

[10] Sara Haddouch; Hanaâ Hachimi; Nabil Hmina. *Modeling the flow of road traffic with the SUMO simulator*. [descargado]. URL: https://ieeexplore.ieee.org/document/8370580.

[11] Adrian Abunei; Ciprian R. Comşa; Constantin F. Caruntu; Ion Bogdan. *Redundancy Based V2V Communication Platform for Vehicle Platooning*. [descargado]. URL: https://ieeexplore.ieee.org/document/8801781.

[12] Florian Klingler. *OpenC2X-standalone*. [en línea]. URL: https://github.com/florianklingler/OpenC2X-standalone.

[13] Felix Fietkau; Álvaro Fernández y otros. *OpenWrt UCI (Unified Configuration Interface)*. [en línea]. URL: https://git.openwrt.org/project/uci.git.

[14] Felix Fietkau; Álvaro Fernández y otros. *libubox (C utility functions for OpenWrt)*. [en línea]. URL: https://git.openwrt.org/project/libubox.git.

[15] Centro Aeroespacial Alemán (DLR) y otros. *netgenerate*. [en línea]. URL: https://sumo.dlr.de/docs/netgenerate.html.

[16] Centro Aeroespacial Alemán (DLR) y otros. *TraCI*. [en línea]. URL: https://sumo.dlr.de/docs/TraCI.html.

[17] Centro Aeroespacial Alemán (DLR) y otros. *TraCIAPI*. [en línea]. URL: https://sumo.dlr.de/docs/TraCI/C%2B%2BTraCIAPI.html.

[18] Centro Aeroespacial Alemán (DLR) y otros. *SUMO*. [en línea]. URL: https://sumo.dlr.de/docs/FAQ.html.

[19] Juan Marcelo Pardo Soliz. *Tutorial SUMO 02: Giros y semáforos*. [en línea]. URL: https://www.youtube.com/watch?v=CpCzI_u0MSc.

[20] Shimaa A. Abdel Hakeem; HyungWon Kim. *Advancing Intrusion Detection in V2X Networks: A Comprehensive Survey on Machine Learning, Federated Learning, and Edge AI for V2X Security*. [descargado]. URL: https://ieeexplore.ieee.org/document/11012728.

Anexos

A. Código fuente y materiales del proyecto

Todo el código desarrollado para este Trabajo de Fin de Máster se encuentra disponible en el repositorio público de GitHub:

`https://github.com/saulibanez/openc2x-sumo-integration`

El repositorio está organizado en varios directorios:

- **OpenC2X-standalone/** Versión adaptada a Ubuntu 24.04 con soporte TraCI integrado y modificaciones en:
  - **common/interface/**: Contiene el módulo *SumoInterface*, desarrollado para integrar SUMO mediante TraCI.
  - **gps/**: Incluye las modificaciones necesarias para recibir posiciones desde SUMO y generar eventos *trigger* que activan mensajes DENM.
  - **resto de módulos en `common/`**: Conjunto de componentes donde se aplicaron diversas correcciones y adaptaciones para permitir la compilación y ejecución de OpenC2X en Ubuntu 24.04 (actualización de librerías, CMake, rutas e *includes*).
  - **cam/, denm/ y ldm/**: La lógica funcional de estos módulos no se modifica. Únicamente se añadieron (y se dejaron comentadas) trazas de instrumentación destinadas a medir métricas del sistema utilizadas durante las pruebas V2V y V2I.
- **SUMO/** Contiene los escenarios empleados en la evaluación:
  - `hello_openc2x/`: Validación básica SUMO ↔ TraCI.
  - `sumo_v2v/`: Escenarios de evaluación V2V.
  - `sumo_v2i/`: Escenarios de evaluación V2I.
  - Dentro de cada escenario:
    - **Scripts de simulación** (`runv2v.sh`, `runv2i.sh`) que lanzan SUMO y cada módulo de OpenC2X en sesiones separadas.
    - **Scripts auxiliares**, como `generar_vehiculos.sh`, utilizados para generar automáticamente un número configurable de vehículos.

    - **Carpeta `log/`** con:
        - Ficheros de trazas de cada simulación.
        - El script de análisis Python empleado para calcular la latencia y el PDR (Packet Delivery Ratio).

- **Memoria/** Contendrá:
    - El código LaTeX completo del TFM.
    - Las figuras utilizadas.
    - El PDF final del documento.

B. Reproducción del entorno

Para ejecutar el sistema es suficiente con clonar el repositorio y seguir los pasos indicados en el archivo `README.md`, que detalla:

- Requisitos de compilación.
- Construcción de OpenC2X.
- Ejecución de SUMO con TraCI.
- Lanzamiento de escenarios V2V y V2I.
- Análisis de resultados con los scripts proporcionados.

C. Scripts proporcionados

El repositorio incluye varios scripts destacados:

- `generar_vehiculos.sh`: Genera automáticamente vehículos en SUMO según el número indicado por el usuario.
- `runv2v.sh`: Ejecuta SUMO y lanza los módulos GPS, CAM, DCC y LDM para las pruebas de comunicación V2V.
- `runv2i.sh`: Ejecuta SUMO con semáforos y lanza GPS, DENM, DCC y LDM para las pruebas V2I.
- `analyze_v2v.py`: Script de análisis utilizado para calcular latencia, intervalo entre CAM y PDR en los escenarios V2V.
- `analyze_v2i.py`: Script de análisis para calcular latencia, PDR y emparejamiento de eventos DENM en los escenarios V2I.