ETSIT
Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Emulación de un sistema de prevención de accidentes de tráfico en cruces mediante comunicación vehículo - infraestructura

Autor:

D. José Luis Martín Robles

Tutor:

Dr. D. Juan Carlos Aguado Manzano

VALLADOLID, ENERO 2023

TRABAJO FIN DE Máster

TÍTULO: **Emulación de un sistema de prevención de accidentes de tráfico en cruces mediante comunicación vehículo - infraestructura**

AUTOR: **D. José Luis Martín Robles**

TUTOR: **Dr. D. Juan Carlos Aguado Manzano**

DEPARTAMENTO: **Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE: **Ignacio de Miguel Jiménez**

VOCAL: **Ramón José Durán Barroso**

SECRETARIO: **Alfonso Bahillo Martín**

P. SUPLENTE: **Patricia Fernández Reguero**

S. SUPLENTE: **Javier Aguiar Pérez**

V. SUPLENTE: **Ramón De la Rosa Steinz**

RESUMEN

Los Sistemas Inteligentes de Transporte Cooperativos o C-ITS tienen el objetivo de mejorar la seguridad y eficiencia de la infraestructura vial haciendo uso de las innovaciones en el sector de las TIC. Una tecnología clave para el despliegue de los sistemas C-ITS son las comunicaciones vehiculares o V2X (Vehicle To Everything), dentro de las cuales se enmarcan las comunicaciones Vehículo-Infraestructura o V2I (Vehicle to Infrastructure).

El objetivo de este Trabajo de Fin de Máster es el desarrollo de un modelo de emulación de un sistema de seguridad de prevención de colisiones basado en V2I, motivado por una colaboración del proyecto ARTEMIS del Grupo de Comunicaciones Ópticas con el Grupo Renault España para validar y evaluar sistemas de este tipo.

Con este objetivo en mente, se desarrolla una arquitectura de servicio conforme a las tendencias actuales de las compañías automovilísticas en materia de sistemas V2X. Para ello se implementará el acceso al servicio a través del protocolo celular LTE, objetivo conseguido mediante la emulación de una red 4G a través de tecnología SDR. Además, se implementa estándares de comunicación vehicular a través del *software* OpenC2X, que junto al protocolo IoT MQTT conforman la mayor parte de la pila de protocolos de comunicación empleados para el proyecto.

PALABRAS CLAVE

Comunicación Vehículo-infraestructura (V2I), Sistemas LTE, MQTT, Mensaje de Aviso Cooperativo (CAM), Mensaje de notificación en entorno descentralizado (DENM), Radio definida por Sofware (SDR), latencia de red, servicios de día 1.

ABSTRACT

Cooperative Intelligent Transport Systems or C-ITS aim to improve the safety and efficiency of road infrastructure using ICT sector innovations. A key for C-ITS systems deployment is Vehicular Communications or V2X (Vehicle To Everything), within which Vehicle To Infrastructure communications (V2I) are framed.

The objective of this Master's Thesis is the development of an emulation framework for a V2I-based collision avoidance safety system, motivated by a collaboration of Grupo de Comunicaciones Optica's project ARTEMIS with Grupo Renault España to evaluate and validate systems of this kind.

With this objective in mind, a service architecture is developed in accordance with the current trends of automotive companies in V2X systems. To that end, access to the service will be implemented through the LTE cellular protocol, which was achieved by 4G network emulation through the use of SDR technology. In addition, vehicular communications standards are implemented through the OpenC2X software, which alongside the IoT protocol MQTT, form the bulk of the protocol stack used for the framework.

KEYWORDS

Vehicle to Infrastructure communcations (V2I), Long Term Evolution (LTE), MQTT, Cooperative Awareness Message (CAM), Decentralized Environment Notification Message (DENM), Software Defined Radio (SDR), network delay.

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mi tutor Juan Carlos Aguado Manzano por darme la oportunidad de trabajar en este proyecto, así como los medios materiales para llevarlo a cabo. Este trabajo se habría hecho bastante más pesado de no ser por las largas charlas de geopolítica y los debates sobre los fundamentos de la sociedad que acompañaban los cafés de las once de la mañana en el laboratorio. Muchas gracias, además, por la infinita paciencia y comprensión que has tenido con mis constantes idas y venidas.

Agradezco también a mi familia: a mi padre, por motivarme e instigarme a ser constante, a mi madre por proporcionarme un espacio de tranquilidad y confianza cuando el estrés me sobrepasaba, a mis hermanas por inculcarme la pasión por la ciencia desde que tengo uso de memoria y en especial a Nieves por transmitirme su pasión por el proceso científico y ayudarme con la revisión de los textos aquí presentados y finalmente a mi Abuela, esto va por ti más que por nadie.

No podría olvidarme tampoco de agradecer a mis amigos/familia del Curueño, gracias por ser mi eterno oasis de tranquilidad, mi ancla a la realidad, y sobretodo, mi válvula de escape. Este año tan extraño se me ha hecho mucho mas fácil de sobrellevar gracias a todas y cada una de las historias, a cada cual más sórdida, que nos han acontecido en los lugares recónditos de la Montaña Leonesa.

Finalmente, muchas gracias a todos mis compañeros de Teleco, espero que nos queden muchos más viernes de salir a tomar una cerveza “de tranquis” y que de ahora en adelante llevemos a cabo más de las aventuras ambiciosas que siempre proponemos pero que escasas veces salen. Dentro de mis amigos de Teleco me veo en la obligación de dedicarle un espacio específicamente a mis compañeros de laboratorio, las pocas tardes que coincidíamos todos no fueron precisamente productivas, pero se compensaron con lo memorable de estas.

AGRADECIMIENTOS INSTITUCIONALES

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

Este trabajo ha sido realizado gracias a Vodafone, que autorizó de manera gratuita a la UVa a utilizar algunas de las frecuencias que tiene licenciadas para servicios de telefonía móvil para uso propio en actividades de investigación y docencia.

Este trabajo se ha realizado en colaboración con el Grupo Renault España, que nos ha apoyado con recomendaciones e información relevante para su consecución.

Índice

Índice de Figuras

Índice de tablas

1.Introducción

1.1 Motivación

En el año 2021 se produjeron en las vías interurbanas españolas un total de 921 siniestros mortales, en los cuales se produjeron un total de 1004 fallecimientos además de 3728 heridos. Estas cifras se corresponden con una disminución del 9% de víctimas mortales frente al año 2019. Sin embargo, es conveniente contrastar esta variación con una disminución del 8% de la movilidad (número de desplazamientos largos) en ese mismo intervalo de tiempo, lo cual implica que la reducción de la siniestralidad no es únicamente debida a la reducción del tráfico vial [1]. Esto encaja dentro de la tendencia macro del descenso de la siniestralidad que se ha dado tanto a nivel nacional como a nivel europeo [2].

Aunque la tendencia sea a la baja, cada nuevo deceso es totalmente inaceptable y es de vital importancia trabajar como sociedad por reducir esta cifra mediante todos los medios posibles. En la actualidad, nos encontramos en un momento clave para replantear cómo se aborda esta problemática, ya que nos hallamos en uno de los mayores puntos de inflexión de la historia de la automovilística comercial. Esto se debe al estado de evolución tecnológica actual, que posibilita la implementación y comercialización de nuevas tecnologías tan diversas como son: vehículo eléctrico, tecnologías de vehículo conectado y técnicas de conducción autónoma.

Con todas las herramientas de las que se dispone actualmente, sumadas a su bajo coste y facilidad de producción a gran escala, se puede dotar a cada vehículo tanto de capacidad de percepción del entorno (sensórica avanzada, p.ej: LiDAR, cámaras, sensores de distancia…), como de "inteligencia" para procesar dicha percepción, así como de la capacidad de transmitir y diseminar dicha "inteligencia" (tecnologías *Wireless* de banda ancha). Con todas estas herramientas la cantidad de soluciones posibles a la problemática es básicamente infinita, siendo pues el trabajo del ingeniero encontrar las mejores dentro de este nuevo espacio de posibilidades.

Los sistemas que basan las mejoras implementadas en la conducción en las TIC se denominan Sistemas Inteligentes de Transporte (ITS). El trabajo desarrollado en este trabajo de fin de máster se centra en un subconjunto de estos, los Sistemas de Transporte Inteligentes Cooperativos (C-ITS), es decir, sistemas TIC que distribuyen la información con el objetivo de generar una percepción global o local de la situación de interés tanto para vehículos en movimiento, como entidades reguladoras o elementos activos de la infraestructura vial.

En la actualidad existen ambiciones estandarizadoras de estas tecnologías que buscan establecer conjuntos de protocolos y técnicas que aseguren la interoperabilidad, interconectividad y seguridad de las comunicaciones de los sistemas C-ITS. Todavía no existe un estándar de facto, aunque las propuestas se dividen en dos grandes tendencias marcadas por la tecnología de acceso al medio radioeléctrico. Por un lado, las propuestas basadas en acceso mediante tecnologías móviles (C-V2X) impulsadas por la organización 3GPP, principalmente basadas en LTE (Long Term Evolution) y

5G. Por otro lado, las tecnologías basadas en el acceso definido por el estándar 802.11p, donde se sitúan las propuestas DSRC (Dedicated Short Range Communications), impulsada por organizaciones estadounidenses (IEEE y SAE), y las normas ITS-G5 en Europa definidas por la ETSI y CEN.

La motivación del presente trabajo de fin de máster nace de una colaboración entre el Grupo de Comunicaciones Ópticas y el grupo Renault España, que está trabajando actualmente en la implementación de una aplicación C-ITS para la prevención de accidentes de tráfico en situaciones donde no exista visión directa del peligro inminente. La aplicación que se plantea basa su funcionamiento en comunicaciones vehiculares siguiendo el paradigma V2I (Vehicle To Infraestructure) y acceso mediante tecnologías móviles. La propuesta basa su funcionamiento en la diseminación de información de los vehículos en circulación, a partir de la cual se genera una percepción local de la situación, que permite a un servidor centralizado emitir avisos de potenciales colisiones a los conductores.

En la colaboración establecida entre Grupo Renault España y el Grupo de Investigación Reconocido de Comunicaciones Ópticas (GIR-GCO) de la Universidad de Valladolid se planteó la creación de una maqueta que emule un servicio con características similares a la que Grupo Renault España está implementando. El objetivo será crear una plataforma de pruebas en la que se tenga control sobre todos los componentes funcionales de la misma, abriendo de esta manera, la posibilidad de realizar pruebas extensivas sobre el sistema implementado en un entorno controlado. El testeo de sistemas de este tipo es especialmente relevante, puesto que en un sistema de seguridad crítico como el que se plantea, la fiabilidad y la velocidad de respuesta son características esenciales.

1.2 Objetivos

El objetivo general del trabajo fin de máster es implementar un sistema que permita emular un servicio de día uno de detección de posible colisión en un cruce sin visibilidad utilizando comunicación vehículo infraestructura.

Los objetivos específicos del proyecto se definieron de acuerdo con las especificaciones marcadas por Grupo Renault España, que se inspiran en algunas características de las arquitecturas investigadas en la propia empresa.

Las especificaciones básicas del sistema al inicio del proyecto eran las siguientes: se debía desarrollar mediante tecnología LTE, lo que resultaba posible gracias a que trabajos previos realizados por el GCO están basados en esta tecnología [3] [4]. El sistema debía ser capaz de emular las otras dos partes del sistema, por un lado el vehículo en movimiento y por otro lado el servidor. Estas dos partes se debían comunicar utilizando los mensajes definidos en el estándar ITS-G5, en concreto los protocolos CAM [5] y DENM [6]. El servidor debía funcionar bajo el paradigma publicador-suscriptor e integrando medidas de seguridad.

Por lo tanto, los objetivos específicos del trabajo fueron:

- Desplegar un sistema ITS-G5 en ambos extremos de la red, compatible con soluciones encontradas en la industria.
- Desplegar un servicio LTE basado en opensource y SDR.

- Desplegar un sistema se seguridad similar al que se puede encontrar en algunas soluciones de la industria.
- Implementar las aplicaciones necesarias para emular un caso de uso de comunicación V2I.
- Toma de medidas relevantes del caso de uso para demostrar la viabilidad del servicio.

1.3 Fases del proyecto

Como se ha mencionado en el anterior subapartado durante todo el desarrollo del proyecto se mantuvo una comunicación estrecha con Grupo Renault España, en la que en todo momento se hizo énfasis en que la métrica de rendimiento más crítica en un sistema de este tipo: la latencia extremo a extremo. Por esta razón la primera fase del proyecto consistió en una revisión de literatura científica en lo relativo a la latencia de sistemas con acceso LTE, así como de sistemas publicador-suscritor e IoT en general. En esta fase también se centraron esfuerzos en revisar literatura concerniente a comunicaciones vehiculares, y el estándar ITS-G5.

En la segunda fase se trabajó en la implementación de la red LTE. Inicialmente se estudió varias alternativas viables para este apartado del proyecto, utilizando como apoyo la experiencia adquirida en proyectos previos realizados en el GCO [3] se decidió utilizar el *software* srsRAN.

Para la tercera fase se estudió las posibilidades existentes para implementar los diferentes componentes de un servicio publicador – suscriptor, eligiéndose MQTT. Tras escoger el *software* que se consideró más apropiado para el proyecto se implementó sobre la red LTE.

En la cuarta fase se investigó las implementaciones de los protocolos ITS-G5. De nuevo, el trabajo anterior realizado por el GCO facilitó este procedimiento [7] [8]. Tras escoger una solución se aplicó al sistema junto con todos los componentes introducidos en las anteriores fases.

La quinta fase consistió en estudiar el funcionamiento de virtualizaciones de SO, y desplazar el rol de servidor MQTT a una máquina virtual alojada en la Universidad de Valladolid. En esta fase también se buscan opciones para implementar autenticación TLS en el sistema.

En la sexta fase se trabaja para incluir el último bloque funcional al sistema, autenticación TLS implementada en una pasarela en una plataforma de servicios *cloud*.

En la séptima y última fase, se integró en la arquitectura del servicio una red de acceso GPON con vistas a emular el enlace entre las estaciones base y el núcleo de la red LTE.

1.4 Medios utilizados para la realización del proyecto

Para la realización del proyecto se tuvo a disposición los siguientes equipos electrónicos:

- 6 tarjetas SDR *BladeRF 2.0 micro xA9* de *Nuand* con dos canales RX y TX. MIMO, rango de frecuencias de 47MHz a 6 GHz y 30 MHz de ancho de banda.
- 10 tarjetas SIMs programables sysmoISIM-SJA2.
- PC Ubuntu 22.04, CPU Intel i7-9700 de 6 núcleos, 16 GB RAM, utilizado para implementar la estación base LTE y el *core* de la red.
- 1 PC Ubuntu 16.04, CPU AMD Phenom II x4 955 de 4 núcleos, 4 GB RAM, utilizado como cliente de los servicios para realizar pruebas.
- Vehículo Amazon DeepRacer con Ubuntu 16.04, CPU Intel Atom, 2 GHz de RAM, utilizado como nodo de generación de mensajes ITS-G5.
- 1 mini PC Intel NUC 12 Pro X - NUC12DCMv9, SO Ubuntu 20.04 LTS, CPU Intel core i9 12900, 64 GB de memoria RAM.
- Infraestructura de una red GPON

2.Estado del arte

En la siguiente sección se revisa la evolución de los sistemas C-ITS, en primer lugar, analizando su proceso de estandarización, y pasando después por un breve repaso del estado del despliegue de estos sistemas en la Unión Europea.

Una vez contextualizado el estado actual de los desarrollos en el campo C-ITS en Europa, se revisará el estado actual de la literatura científica de sistemas de prevención de colisiones basados en tecnologías de comunicaciones vehiculares.

Por último, se revisa las tendencias en materia de ciberseguridad en los sistemas de comunicaciones vehiculares.

2.1 Evolución de los estándares C-ITS

La estandarización en comunicaciones es necesaria para diversos propósitos, entre los que se encuentran garantizar la interoperabilidad de sistemas o la conformidad de estos a la legalidad vigente, siendo el objetivo final garantizar una adopción generalizada de las tecnologías y la viabilidad económica de estas. En la actualidad existen dos propuestas estandarizadoras que dominan el mercado, desarrolladas por la UE y el gobierno americano respectivamente [9].

En el año 2014 la Comisión Europea creó la plataforma para el despliegue de sistemas C-ITS en la Unión Europea, cuyo objetivo consiste en identificar barreras técnicas y trazar un plan de acción para la implementación de estos sistemas en el territorio de la UE. La plataforma se concibe como un marco de diálogo entre los actores interesados en el despliegue de los sistemas C-ITS, incluyendo autoridades nacionales y regionales, las partes interesadas del sector privado y la Comisión Europea [10].

La primera fase de desarrollos de la plataforma finalizó en enero 2016, con la presentación de un informe que recogía los esfuerzos realizados en la plataforma hasta el momento [11]. En este informe se traza la cronología de despliegue de aplicaciones C-ITS en la UE, separando los mismos en varias fases denominadas “días”. Los despliegues iniciales se denominan de día 1 y se listan a continuación:

Servicios de día 1:

- Servicios de notificación de ubicación peligrosa:
  - Luz de frenado de emergencia
  - Vehículo de emergencia aproximándose
  - Vehículos lentos o estacionarios
  - Embotellamiento cercano
  - Obras cercanas
  - Condiciones meteorológicas adversas

- Servicios de señalización:
  - Señalización en el vehículo
  - Límites de velocidad en el vehículo
  - Sondeo de datos del vehículo
  - Amortiguación de movimientos sísmicos

- Incumplimiento de señalización / Seguridad en intersecciones
- Señalización para vehículos prioritarios
- Velocidad óptima para luz verde en semáforos

En el documento se indica el paso posterior de implementación de servicios C-ITS, denominados servicios de día 1.5, los cuales son los que se listan a continuación:

Servicios de día 1.5:

- Información sobre repostaje y recarga
- Protección a los usuarios vulnerables de las vías públicas
- Gestión e información de aparcamiento en la vía pública
- Información sobre aparcamiento disuasorios
- Navegación conectada y cooperativa en ciudad
- Información de tráfico y cálculo inteligente de rutas

Como se puede observar el trabajo desarrollado en el presente proyecto se enmarca en los servicios de día 1, en concreto dentro de los denominados servicios de notificación de ubicación peligrosa. El servicio objetivo de este proyecto se puede catalogar como notificación de frenado de emergencia, ya que el planteamiento del servicio de momento no se contempla un frenado autónomo del vehículo ante la detección de una futura colisión.

En el informe de la plataforma C-ITS [11] se recomienda el uso de la pila de protocolos ETSI ITS para implementar su visión de los sistemas C-ITS en la UE. ETSI ITS consiste en una familia de estándares de protocolos para comunicaciones vehiculares publicado en el año 2009 por la organización estandarizadora europea ETSI. En el informe se recomienda además el uso de la tecnología ITS-G5 para el acceso por radio al medio, tecnología que también fue definida en el estándar de la ETSI del año 2009. G5 hace referencia a la banda de 5,9 GHz, utilizada por el protocolo 802.11p, o Wi-Fi vehicular para comunicaciones de corto alcance. En el informe se hace hincapié en fomentar el agnosticismo de capa de enlace, con el objetivo de implementar los protocolos de niveles superiores con independencia de la tecnología de acceso por radio al medio.

De manera paralela, la asociación de estandarización de tecnologías celulares 3GPP, publica en 2016 en la *release* 14 del protocolo LTE, denominada LTE-V2X o C-V2X. Este estándar contempla la estandarización de comunicaciones V2X mediante la tecnología de acceso LTE [12]. En este estándar se detalla el funcionamiento de la tecnología LTE-sidelink, que permite la comunicación directa de dos terminales de usuario, haciendo así posible la comunicación V2V (Vehicle to Vehicle). Al publicarse este estándar, se define una alternativa viable al uso de 802.11p como tecnología de acceso para sistemas C-ITS.

Al publicarse la *release* 14 del estándar LTE se inicia un debate en la industria, aún vigente hoy en día, sobre qué tecnología de acceso es la más adecuada para el despliegue a gran escala de los servicios C-ITS V2X (Vehicle to Everything).

Entre las ventajas de la implementación de C-V2X se encuentra la facilidad para el despliegue de los servicios puesto que ya existe extensiva infraestructura LTE instalada en las proximidades de la infraestructura vial. Además, la tecnología C-V2X

simplificaría el *hardware* V2X de los vehículos, ya que el acceso mediante LTE puede ser utilizado tanto para comunicaciones de largo como de corto alcance, al contrario que 802.11p, limitada a comunicaciones de corta distancia, y que por tanto requeriría de otra tecnología de acceso (generalmente celular), para comunicaciones de larga distancia (ver Figura 1).

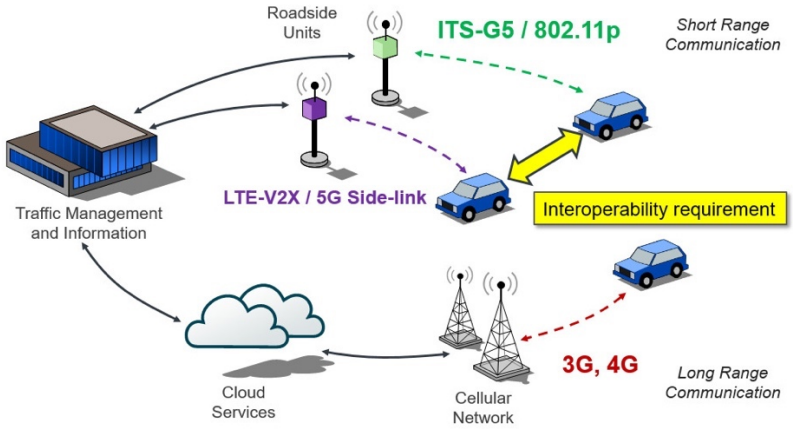


Figura 1.- Diferencias entre paradigmas C-V2X e ITS-G5

De cara al rendimiento de ambas alternativas, en [13] los autores realizan un estudio comparativo modelando el canal, donde reflejan que ninguna de las tecnologías de acceso es inherentemente superior. C-V2X es más apto para comunicaciones de largo alcance, y, presenta menores tasas pérdida de paquetes, aunque a costa de mayor uso de recursos de radio. Para casos de baja congestión (menos de 150 usuarios/km^2), C-V2X arroja un rendimiento superior a ITS-G5, no obstante, para mayores densidades de usuarios el rendimiento de C-V2X decae de manera más rápida que su alternativa. En cuanto a métricas de rendimiento temporal los autores no señalan que ninguna de las alternativas sea superior, ya que el desempeño de estaría fuertemente ligado a las condiciones en las que se produce la comunicación. En [14] se evalúa la tasa de error de bit de ambas tecnologías de acceso para diferentes modelos de canal, encontrándose que C-V2X ofrece mejores resultados para la mayoría de los casos, y, más en concreto una mejora sustancial para vehículos circulando a alta velocidad.

En cuanto a la adopción comercial de cada una de estas tecnologías, no existe consenso actualmente. Los actores empresariales partidarios de las tecnologías C-V2X se reúnen en la asociación 5GAA (5G Automotive Association) [15], que promueve la tecnología LTE-V2X, y que apoya la tecnología 5G en un futuro como estándar para V2X. Contrariamente, las tecnologías basadas en el estándar 802.11p son promovidas por entidades reguladoras americanas como la FCC (Federal Communications Comission) a través del estándar DSRC (Dedicated Short Range Communications), apoyado a su vez por empresas como Toyota, Volkswagen o General Motors. No obstante, ambas tecnologías no son mutuamente excluyentes, y la propia asociación 5GAA planteó los beneficios de la coexistencia de ambas tecnologías en la banda de 5,9 GHz, ya que, aunque ambas tecnologías no sean interoperables, la selección dinámica por parte de los vehículos de una de estas dos tecnologías en función del caso de uso permitiría explotar las ventajas de ambas [16].

2.2 Estado actual de la implementación V2X

En el año 2019 el mercado V2X se valoraba en 2.562 millones de dólares, y se proyecta que para el año 2027 alcance un valor de 11.768 millones de dólares, con una tasa de crecimiento anual compuesto del 28,4%. Europa supone el mayor mercado V2X del mundo, contribuyendo 851,8 millones al valor de mercado en 2019, y esperando mantener un crecimiento acorde al resto del mundo, con una tasa de crecimiento anual proyectada del 24,4% [17].

Debido tanto a las proyecciones económicas, como a la potencial reducción en el número de accidentes de tráfico, la Unión Europea ha mostrado interés en cooperar con el despliegue de los servicios C-ITS a través de la creación de la plataforma C-Roads [18]. La plataforma C-Roads se funda con el objetivo de garantizar la cooperación de los países miembros en el despliegue de servicios C-ITS. El plan de ruta de la plataforma consiste en primer lugar, en desplegar tramos piloto para probar la tecnología en un entorno real, para finalmente desplegar un sistema paneuropeo de servicios C-ITS unificado e interoperable.

A fecha de hoy, existen ya corredores C-ITS instalados en la Unión Europea, estos corredores son principalmente utilizados para que los fabricantes de vehículos realicen pruebas, ya que todavía distan del despliegue a gran escala buscado. Los pilotos desplegados por el momento buscan validar y probar aplicaciones de día 1 y día 1,5, y para ello implementan tanto tecnología de acceso ITS-G5 como C-V2X.

Figura 2.- Pilotos C-ITS C-Roads desplegados en Europa [18]

Actualmente existen 5 proyectos de la plataforma C-Roads desplegados o en desarrollo en el territorio español (ver Figura 3) [19].

- DGT 3.0: El objetivo de este piloto es desplegar servicios C-ITS en toda la red de carreteras española, con una cobertura aproximada de 12.270 km. La plataforma ofrece principalmente servicios de información de tráfico al conductor. El proyecto basa su funcionamiento en tecnología celular y protocolos IoT [20].
- SISCOGA extended: Consiste en una extensión de la infraestructura de pruebas ubicada en el entorno metropolitano de la ciudad de Vigo. Este piloto utilizará tecnología de acceso híbrida (tanto ITS-G5 como celular), para permitir testear servicios de día 1 y día 1.5, entre los que se incluyen servicios de ubicación peligrosa, y servicios de información.
- Madrid: Ubicado en la M-30 madrileña, utilizará tecnologías de comunicación híbridas para testear aplicaciones V2X de seguridad, incluyendo en sus pruebas vehículos autónomos.
- Cantábrico: Se desplegará en varios tramos de la A8 en la cornisa cantábrica. Utilizará tecnologías de comunicación híbridas para implementar servicios de día 1 y 1.5.
- Mediterráneo: Este piloto se desplegará en dos tramos que suman 150 km en total en la autopista AP7. Para este piloto se implementará servicios de información del estado de la carretera con tecnologías de comunicación híbridas.

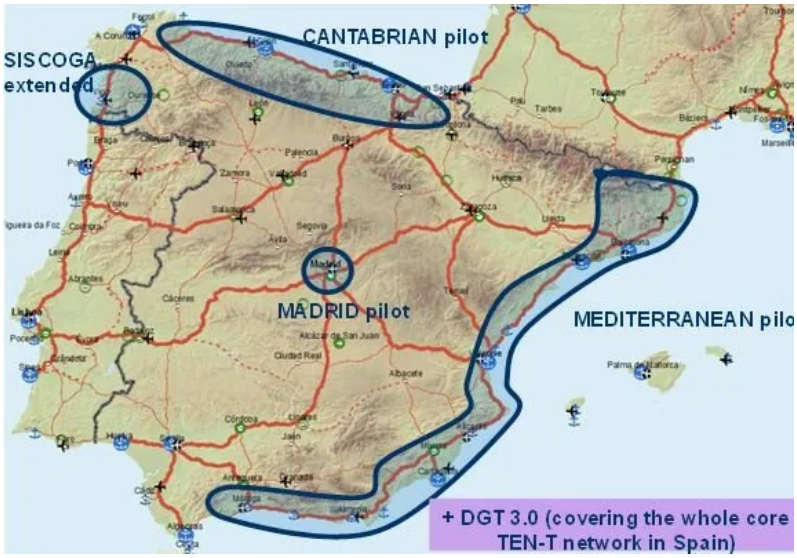


Figura 3.- Pilotos C-Roads desplegados en España [19]

En cuanto a implementaciones comerciales de servicios V2X, las ofertas de vehículos comerciales que implementan servicios C-ITS es todavía muy limitada.

El primer modelo de automóvil de producción europea que implementa tecnologías V2X es el Volkswagen Golf 8, en el mercado desde el año 2019 [21]. La tecnología V2X que implementa este vehículo se encuentra conforme a todos los estándares fijados por la ETSI, y utiliza ITS-G5 como tecnología de acceso. Este vehículo cubre varios servicios de día 1, principalmente aquellos que informan al conductor de cambios en la situación de la carretera. El funcionamiento de estos servicios depende principalmente de comunicación V2V, ya que el despliegue de estaciones C-ITS sigue siendo muy limitado. La evidente limitación de la implementación de estos servicios es por tanto que dependen de la presencia de otros vehículos para resultar efectiva, y hasta el momento el número de vehículos que cumplan con el estándar ETSI en las carreteras es extremadamente bajo, no obstante, es de esperar que el funcionamiento

de estos servicios mejore de manera proporcional a la penetración de la tecnología. Hasta el momento dos modelos más de la marca alemana han aplicado la misma tecnología.

Actualmente el fabricante de automóviles Audi, también perteneciente al grupo Wolkswagen, está desarrollando servicios C-ITS basados en tecnología C-V2X [22]. El desarrollo de la tecnología se encuentra en un estado muy avanzado, con prototipos funcionales ya mostrados al público que prueban el funcionamiento de sistemas de aviso de colisión [23]. Los vehículos del fabricante alemán también integrarán los servicios ITS-G5 desarrollados para vehículos Volkswagen, ya que pertenecen al mismo conglomerado empresarial.

2.3 Revisión de literatura de sistemas V2X de prevención de colisiones

El crecimiento de la tecnología V2X, como es habitual, viene acompañado de gran abundancia de literatura científica. En este subapartado se explorará el estado actual del campo desde la perspectiva académica, y más específicamente, dado el objetivo del proyecto, en sistemas de prevención de colisiones basados en comunicaciones V2I.

En [24] se revisa el estado del arte de la literatura académica concerniente a los desarrollos de técnicas de testeo de aplicaciones V2X hasta el año 2019. En este artículo se destaca que, en dicho momento, las tecnologías V2X se encontraban todavía en etapa exploratoria. Además, se manifiesta que el rol de las instituciones académicas debe de ser validar las técnicas V2X con objetivo de verificar que estas se atengan a las restricciones impuestas por las entidades reguladoras, con el objetivo de asegurar su funcionamiento correcto de cara a la comercialización de vehículos que implementen estos sistemas.

En [24] además se subraya que los principales desafíos que deben enfrentar las técnicas V2X son dos: garantizar que los sistemas desarrollados sean resistentes a ciberataques, y que el rendimiento de la red sea elevado, especialmente en cuanto a latencia de red. Para hacer frente a estos desafíos se debe testear las tecnologías extensivamente, teniendo en cuenta todos los casos de uso, y vulnerabilidades conocidas. En el artículo además se cataloga los esfuerzos de testeo de tecnologías V2X en 8 categorías diferentes, estando los desarrollos del presente proyecto enmarcados bajo la categoría de *performance testing*, categoría que abarca modelos de testeo cuyo objetivo principal es garantizar niveles de latencia adecuados para diferentes situaciones.

En [25] los autores realizan un estudio del rendimiento de los protocolos 802.11p y LTE-V en una intersección de pruebas en la ciudad de Shanghái. En el estudio se buscaba demostrar la aptitud de los protocolos en un entorno realista de intersección sin línea de visión directa, debido a que no se desarrolla una aplicación para el estudio, los autores aplicaron un modelo estadístico para estimar si la comunicación realizada sería capaz de notificar la inminente colisión a tiempo. Los resultados obtenidos muestran que no se puede garantizar la notificación a tiempo de un potencial accidente de tráfico utilizando tan solo comunicación V2V, por lo que los autores proponen utilizar comunicaciones V2I para este caso de uso.

En [26] los autores describen un modelo estadístico para el canal y el algoritmo de un servicio de prevención de colisiones basada en C-V2X. El modelo descrito tiene en cuenta las posiciones y velocidades de los vehículos, así como un modelo de comunicación que tiene en cuenta no idealidades, tales como pérdida de paquetes por perturbaciones del canal. Este artículo calcula la probabilidad de que se genere un mensaje de aviso de posible colisión, llegando a la conclusión obvia de que dicha probabilidad aumenta con la densidad y la velocidad de los vehículos. Sin embargo, no desarrollan conclusiones de cómo este incremento de avisos puede afectar a los vehículos (en el caso de conducción automática o semiautomática), campo en el que su estudio podría tener interés.

En el artículo [27] los autores describen el desarrollo de un algoritmo de prevención de colisiones mediante comunicaciones V2I y V2V basadas en la tecnología LTE-V. Para evaluar el algoritmo los autores recurren a la simulación de tráfico vehicular en conjunción con la simulación de comunicaciones, haciendo uso para ello del simulador de tráfico SUMO junto con simuladores del *framework* MOSAIC. El funcionamiento del algoritmo se basa en la percepción global de la situación generada a partir de las percepciones locales de cada uno de los vehículos, puesto que los vehículos difunden periódicamente información mediante mensajes conformes al estándar de la ETSI CAM [5]. Los autores de [28] también plantean simular los casos de uso del servicio mediante el uso del simulador SUMO. En este artículo se plantea una arquitectura de servicio basada en Edge-Computing, comunicaciones C-V2X y mensajes de actualización periódicas de la percepción local del vehículo (en este caso generados siguiendo el estándar equivalente a CAM de los EE. UU.). Los autores reportan que el sistema desarrollado notificaría a los usuarios finales a tiempo en el 100% de los casos.

La simulación de tráfico es una herramienta muy versátil para presentar primeros prototipos de nuevos conceptos, no obstante, presenta claras limitaciones respecto a la fidelidad de sus resultados frente situaciones reales. Los autores de [29] desaconsejan especialmente su uso para la predicción de colisiones de vehículos, afirmación a la que llegan basándose, entre otros, en el artículo [30]. En este artículo se muestra que los simuladores de tráfico tienden a infrarrepresentar la incidencia de los accidentes de tráfico, debido a diversas razones, tales como la imposibilidad de simular correctamente el comportamiento humano, o lo poco realista de la simulación de la cinemática de los vehículos.

Otros autores plantean alternativas a la simulación del tráfico, como, por ejemplo, en [31], donde se utiliza varios simuladores de la cabina de un vehículo con el objetivo de que seres humanos reales sean participes de las interacciones del tráfico, y de los casos de uso que se pretenda desarrollar. Los autores sin embargo no implementan los protocolos subyacentes a la aplicación que desarrollan, ya que el principal objetivo del estudio es medir la reacción de seres humanos frente a la notificación de una colisión inminente.

La mayoría de los sistemas propuestos en la literatura delegan mayormente su funcionamiento en comunicaciones V2I, no obstante, otras alternativas se siguen explorando. Tanto [32] como [33] buscan implementar aplicaciones de prevención de colisiones haciendo uso tan solo de comunicaciones V2V, siguiendo la arquitectura VANET (Vehicular Ad-hoc Network). En [32] se plantea un algoritmo de prevención de colisiones siguiendo la arquitectura VANET, para ello los autores de nuevo hacen uso del simulador SUMO, aunque no implementan los protocolos de acceso al medio.

Por otra parte, los autores de [33] recurren a la emulación para validar su sistema, haciendo uso de dos maquetas de vehículos a escala, de fabricación propia, que integran un controlador Arduino Nano. La maqueta desarrollada hace uso de un módulo de comunicación Wi-Fi estándar de Arduino.

El trabajo aquí desarrollado se encuentra en línea con las ambiciones y desarrollos futuros del estado del arte del campo V2X. La propuesta de valor del presente proyecto es principalmente la creación de una maqueta de emulación que implementa todas las capas de un modelo de comunicación C-V2X LTE, así como el caso de uso concreto de la notificación de potenciales accidentes de tráfico en intersección. La emulación permite testear casos de uso en un entorno más cercano a la realidad que la simulación, pero sin requerir de la gran inversión de capital que supondría un testeo extensivo con *hardware* final. En el artículo [33] ya se mostraba una maqueta de emulación funcional para el caso de uso de prevención de colisiones, pero presenta claras limitaciones frente a la propuesta del presente proyecto, ya que, al contrario que este, no implementa protocolos estándar en comunicaciones V2X.

2.4 Ciberseguridad, anonimato y soberanía de datos en V2I

En los últimos años el campo de la ciberseguridad ha sufrido un crecimiento meteórico, estando valorado en 140 mil millones de dólares en el año 2021, crecimiento que no solo no se detendrá en los próximos años, sino que acelerará [34]. La prevalencia de los ciberataques en la infraestructura de redes TCP/IP es cada vez más elevada, y el impacto económico y social de estos es cada vez mayor debido a la creciente presencia de dispositivos conectados en todos los ámbitos de la sociedad. Es por esto por lo que es de extremada importancia el desarrollo de técnicas de ciberseguridad que se ajusten apropiadamente a una tecnología emergente como es V2X. La importancia de la ciberseguridad en comunicaciones V2X es especialmente crítica, ya que una intromisión en el funcionamiento de estos sistemas podría suponer la diferencia entre la vida y la muerte tanto de conductores, como de otros usuarios de las infraestructuras viales.

Los ataques que sufrirán las futuras implementaciones de sistemas V2X no serán diferentes a aquellos sufridos actualmente por las redes TCP/IP. El encriptado del canal es especialmente relevante para aplicaciones V2X, puesto que la inyección de tráfico malicioso podría resultar en notificaciones falsas que podrán desencadenar todo tipo de situaciones perniciosas, desde accidentes de tráfico hasta alterar la circulación. Por otra parte, también es muy relevante la necesidad de autenticar los usuarios de un servicio V2X, siendo imperativo imposibilitar a clientes ilegítimos ser partícipes de estos servicios. El anonimato de los datos de los clientes también deberá ser garantizado, puesto que, de no ser así, podría resultar en, por ejemplo, poder realizar un seguimiento permanente de un vehículo concreto [35].

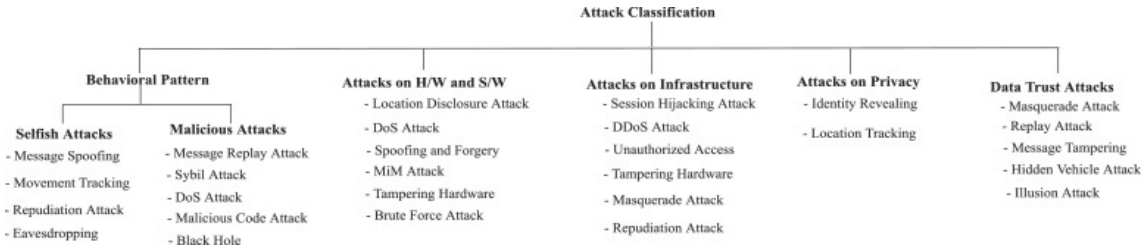


Figura 4.- Tipos de ataques a un servicio V2X [35]

En los últimos años la tendencia más marcada en el ámbito académico en cuanto a la implementación de sistemas de ciberseguridad en V2X es hacia las infraestructuras de clave pública (KPIs) [36]. El concepto de infraestructura de clave pública comprende todos los componentes, tanto *hardware,* como *software*, que actúan para garantizar encriptado de clave pública. Esta tecnología es ampliamente utilizada en otros sectores de las TIC, y, han demostrado con los años de uso gran fiabilidad y efectividad.

Debido a que los mensajes transmitidos por los vehículos contienen información sensible y son transmitidos en modo *broadcast*, estos no deberán contener ninguna información que permita identificar ni al vehículo ni al conductor. Sin embargo, para construir una conciencia colectiva de una situación en la carretera, será necesario diferenciar entre los clientes del servicio, por lo que usualmente se utilizan pseudónimos temporales concedidos por una autoridad central [36].

En este subapartado se ha cubierto hasta el momento la privacidad y anonimato de los datos en lo que concierne a los clientes, sin embargo, ya que las ambiciones de este proyecto surgen de una colaboración con el sector privado, también se deberá tener en cuenta los métodos seguidos por las empresas para proteger la información que manejan, o, dicho de otra forma, proteger su soberanía de datos. La noción de soberanía de datos ha permeado el sector privado de forma creciente en la última década, ya que la relevancia económica del control de datos no ha parado de crecer en estos años junto a la tendencia a desplazar servicios a la nube, esto es, fuera del control de la empresa. Este concepto describe la ambición de organizaciones tanto públicas como privadas de mantener control y poder sobre sus datos, y es usualmente utilizado en contextos de aplicaciones TIC y de ciencia de datos [37].

En el caso de los servicios V2X, las fuentes de datos, los propios vehículos, están bajo control de las empresas productoras de estos, no obstante, los servidores que alojen los servicios habitualmente no pertenecerán a las empresas que manufacturan los vehículos. Esto es lógico, puesto que los servidores deberán prestar servicios a todos los vehículos que circulen por la vía pública independientemente del fabricante, y, por tanto, los servidores podrán ser responsabilidad de entidades públicas o de empresas de terceros.

Existen diversas soluciones para garantizar la soberanía de datos, siendo algunas de ellas soluciones propietarias ofrecidas por compañías de servicios en la nube, aunque se considera especialmente relevante para este proyecto el uso de un servidor pasarela de re-encriptación (también denominado *gateway* o *proxy*), ya que es una solución flexible y sencilla de implementar puesto que no requiere de contratar servicios de una empresa [38].

Se denomina servidor pasarela a un servidor cuya ubicación lógica se encuentra entre los clientes y el servidor, y suple el rol de intermediario en la comunicación. El uso de un servidor pasarela, permite al fabricante modificar los datos generados por los vehículos, ya sea para filtrar información sensible, o añadir nueva información. Además, como servidor intermediario, permite enmascarar la dirección IP origen del mensaje original mensaje con la dirección IP del servidor pasarela. Aplicar re-encriptación simplemente consiste en generar un nuevo túnel de encriptado entre el servidor principal y el servidor pasarela, esto es notable de cara a la soberanía de datos, puesto que otorga al fabricante control sobre las claves criptográficas utilizadas en su flota de vehículos.

3. Desarrollo de la arquitectura del emulador

En este capítulo se detalla el desarrollo de la arquitectura del emulador del servicio de aviso de colisión en cruces sin visibilidad directa. Dado que el emulador parte por un lado de los desarrollos realizados previamente por el grupo GCO en el contexto del proyecto de investigación ARTEMIS, y por otro lado de la colaboración con Grupo Renault España, el capítulo se divide de la siguiente manera.

En una primera parte se explica cómo está construida la maqueta desarrollada por el GCO para el proyecto ARTEMIS. Dicha maqueta permite dar servicio LTE a vehículos por control remoto, permitiendo simular entornos metropolitanos.

En una segunda parte se explica la arquitectura de servicio desarrollada en colaboración con Renault, detallando las claves de dicha arquitectura.

Finalmente se describe cómo la arquitectura se implementó en diversas fases para poder realizar las medidas pertinentes.

3.1 Implementación de una red LTE, maqueta y vehículo

El presente proyecto se contextualiza de forma paralela al proyecto de investigación ARTEMIS desarrollado por el GCO [39], estando disponible para este proyecto todos los recursos utilizados para el mismo. El proyecto ARTEMIS plantea la exploración de tecnologías *edge-computing* en el contexto de comunicaciones V2I en entornos metropolitanos, siendo este Trabajo Fin de Máster un caso de uso concreto dentro del proyecto ARTEMIS, y haciendo posible hacer uso tanto de los recursos del proyecto, como de los conocimientos obtenidos en desarrollos anteriores para ARTEMIS.

Las técnicas V2I desplegadas para el proyecto ARTEMIS se basan en protocolos de comunicación de tecnología de telefonía móvil para el acceso al medio de comunicación entre el vehículo y la infraestructura. Para poder hacer uso de tecnologías de telefonía móvil será necesario implementar los componentes que conforman una red celular, para ello, se hizo uso de diferentes dispositivos y *softwares*, los cuales se explicarán brevemente en esta subsección.

En el proyecto ARTEMIS se trabajó además en la implementación de un *testbed* de escenarios de comunicaciones V2I. El desarrollo de un *testbed* permite la implementación de diversos casos de uso y planteamientos en un entorno más complejo y extrapolable a la realidad que los proporcionados por simuladores. Este *testbed* consiste en una maqueta que permite simular diversas situaciones posibles en carretera, además de vehículos con capacidades de comunicación radio para ocupar el rol de actores dentro de la maqueta.

3.1.1 Red LTE

La red móvil que se implementará esta basada en la tecnología LTE, paradigma bajo el cual las redes móviles consisten en 3 componentes principales [40], mostradas en la Figura 5.

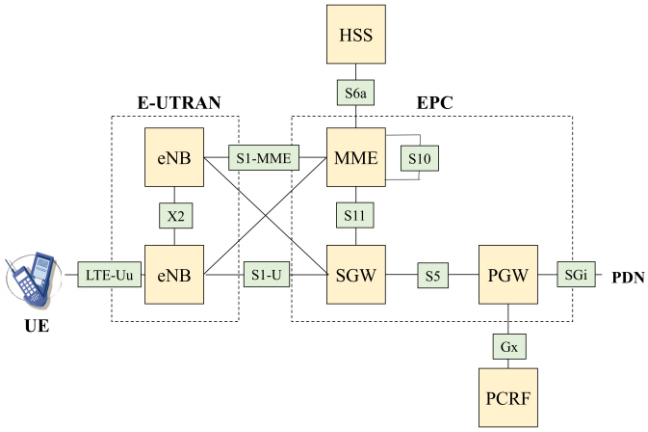


Figura 5.- Arquitectura de una red LTE [41]

- Los primeros y más intuitivos dado el uso habitual de las redes LTE, son los terminales de usuario o UEs, normalmente, teléfonos móviles.

- Otro de los componentes principales de las redes LTE es la red de acceso radio o RAN (Radio Access Network), que en el caso de LTE se denominan E-UTRAN (Evolved UMTS Terrestrial Radio Access Network). La RAN gestiona las comunicaciones radio entre los dispositivos móviles y las redes de conmutación de paquetes IP. El componente principal es el eNodeB o eNB (Evolved Base station), encargado de gestionar los recursos de radio de la célula y las interfaces de control y de usuario entre los dispositivos y el núcleo de la red.

- El núcleo de la red, o EPC (Evolved Packet Core) en el caso concreto de una red LTE. La infraestructura del *core* de la red aporta la interconexión de las estaciones base con las redes troncales IP o PDN (Public Domain Network). Además, cumple la función de alojar diversos servicios divididos en dos planos, plano de control y plano de usuario. Los servicios de plano de control aportan funcionalidades tales como, señalización y control de recursos (servicio MME, Mobile Management Entity), almacenamiento de datos de clientes (servicio HSS) o control de políticas y tarificación de la red (servicio PCRF). Los servicios de plano de cliente implementan interfaces tanto en el lado de las estaciones base (servicio SGW), como de salida a las redes públicas (servicio PGW).

  Para la implementación de los diferentes componentes de la red LTE existen diversos *softwares* que hacen uso de radio definida por *software* (SDR, Software Defined Radio).

  Una plataforma SDR es un sistema de comunicaciones por radio en el cual componentes que tradicionalmente se implementaban mediante *hardware* analógico son en su lugar implementadas mediante *software* ejecutado en PCs o en *hardware* de tipo FPGA. Estos sistemas deben ir equipados con algún *hardware* de adaptación de radiofrecuencia, habitualmente denominado también SDR.

  El esquema básico de un sistema SDR en recepción se muestra en la Figura 6, donde se ve reflejado que se toma de una antena señales en radiofrecuencia para reducirlas a frecuencia intermedia mediante un mezclador, para posteriormente digitalizarlas mediante un conversor A/D y ser reducidas a banda base mediante un mezclador digital. Análogamente, para la transmisión el proceso es el inverso. Las muestras

digitalizadas son transmitidas habitualmente a un PC, donde, con *software* adicional, se emula el comportamiento de un sistema de radio completo.

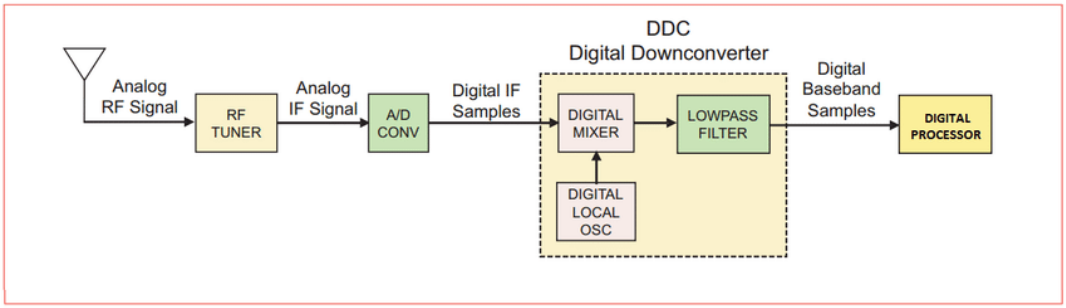


Figura 6.- Diagrama de bloques de un sistema SDR en recepción [42]

Es habitual la implementación de *hardware* SDR con dispositivos FPGA, que permiten modificar la arquitectura del circuito de recepción para poder adaptarlo a diferentes bandas de radiofrecuencia. El beneficio del uso de FPGAs es evidente, puesto que permite implementar sistemas de radiocomunicaciones de un gran rango de frecuencias sin ser necesario disponer de *hardware* específico para cada banda.

Para este proyecto se tenía a disposición diversos dispositivos *hardware* SDR basados en FPGAs, en concreto tarjetas SDR BladeRF 2.0 micro xA9 de la empresa Nuand, y tarjetas USRP NI-B2901. Para el proyecto se utilizó finalmente el *hardware* BladeRF, puesto que a pesar de ser más simple que los dispositivos USRP, las prestaciones que brinda eran suficientes para el proyecto. Este *hardware* permite implementar sistemas de comunicaciones en un rango de 47 MHz a 6 GHz con una tasa de muestreo de 61,44 MS/s y MIMO 2x2 [43]. Estas prestaciones permiten trabajar en las bandas de frecuencia LTE adjudicadas para España, que se encuentran entre 700 MHz y 2,6 GHz.

En cuanto al *software* utilizado para implementar elementos de una red LTE existen diversas alternativas. Para seleccionar una implementación del eNodeB se exploraron comparativas de literatura científica, como [44] donde los autores analizan el rendimiento de los *softwares* srsENB y OAI-RAN, concluyendo que ambas cumplen con los estándares LTE, aunque se destaca que srsENB es una implementación más ligera en cuanto a consumo de recursos. Debido a esta razón, junto con la experiencia proveniente de desarrollos previos en el proyecto ARTEMIS [3], y a la compatibilidad con los dispositivos SDR disponibles, se escogió srsENB como solución.

En cuanto al *core* de la red LTE, de nuevo, existen varias alternativas. De nuevo, en base a experiencias previas del proyecto ARTEMIS, se escogió srsEPC, tanto por concordancia con el *software* de eNodeB, como por ser la implementación comercial más ligera, puesto que su filosofía de funcionamiento radica en tan solo implementar los servicios básicos del *core* de una red LTE.

Para implementar los terminales de usuario o UEs se decidió utilizar módems LTE Huawei E3372. A pesar de ser posible hacer uso de sistemas SDR para este mismo objetivo, los módems LTE ofrecen una funcionalidad equivalente a un menor coste económico y computacional. El desafío que plantea el uso del módem es la necesidad de utilizar una tarjeta SIM física para que estos funcionen correctamente.

Utilizar tarjetas SIM comerciales es totalmente inviable, puesto que esto supondría tener que suplantar estaciones base de operadoras telefónicas para ofrecer servicio a

los módems LTE, además de desconocer las claves que permiten incluir las SIM en las bases de datos de las operadoras. La solución a este problema es el uso de tarjetas SIM programables sysmoISIM- SJA2 de la empresa Sysmocom. Las tarjetas SIM programables permiten ajustar los valores de identificación que estas almacenan, que, en conjunción con la configuración de la base de datos almacenada en el *core* de la red, permite que la conexión del UE no genere conflictos con la infraestructura de las operadoras comerciales españolas.

3.1.2 Maqueta y vehículo

El diseño de la maqueta es una parte clave del *testbed* de vehículo conectado desarrollado para ARTEMIS. Al desarrollar dicha maqueta se buscaba una gran flexibilidad en cuanto a los casos de uso que permitiría emular, es por esto por lo que se planteó un diseño modular, que permite configurar la maqueta de manera flexible para simular diversas configuraciones de vías.

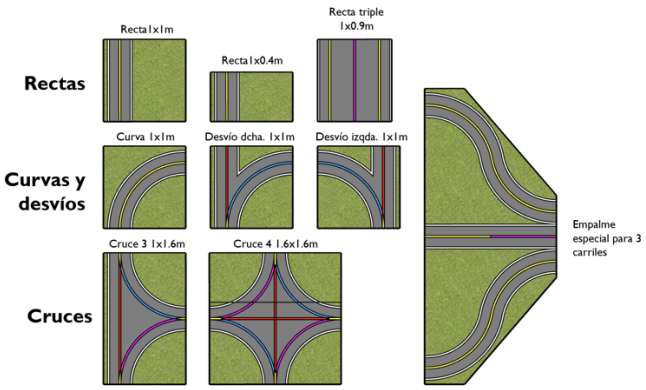


Figura 7.- Módulos disponibles para la maqueta [3]

Los módulos de la maqueta se diseñaron con el objetivo de permitir implementar configuraciones viales complejas, habituales en entornos metropolitanos, tales como vías de varios carriles, incorporaciones, o, especialmente relevante para el presente trabajo, intersecciones. En la Figura 8 se muestra una configuración de la maqueta en intersección.

Figura 8.-Configuración de la maqueta en intersección

Entre los objetivos específicos del diseño de la maqueta se encontraba facilitar el guiado de los vehículos dentro de la maqueta, es por esto por lo que en la versión final de los módulos las líneas de la carretera son exclusivamente de color azul, para simplificar el filtrado por colores. Además, en el trabajo de fin de máster de Ignacio Royuela [3] se desarrolló *software* distribuido para habilitar el guiado autónomo de los vehículos basado en técnicas de procesado de imagen y cálculo de trayectorias.

Los vehículos utilizados para el *testbed*, al igual que la maqueta, se escogieron buscando de nuevo permitir la mayor flexibilidad posible en cuanto a la implementación de casos de uso experimentales, por lo que, por extensión, lo que se busca es que estos sean lo más similar posible a un vehículo autónomo.

Las prestaciones más deseables son, por tanto, la disposición de sensórica y actuadores como los que podrían estar disponibles en un coche autónomo real, así como capacidades de comunicación por radio y un ordenador a bordo programable que permita gestionar la información de los sensores, y, en consecuencia, generar respuestas adecuadas en los actuadores. Los vehículos además deben de estar a escala, puesto que la maqueta fue diseñada alrededor de estos, y unas dimensiones físicas reducidas simplificarían la operación del *testbed.*

El vehículo elegido para el *testbed* fue el Amazon DeepRacer EVO (ver Figura 9), un vehículo diseñado por Amazon con objetivo de ser utilizado para carreras de vehículos autónomos guiados mediante algoritmos basados en inteligencia artificial. Este vehículo incluye un ordenador a bordo que ejecuta Ubuntu 16.04 TLS junto con el *software* ROS (Robot Operating System). Entre las prestaciones *hardware* del ordenador a bordo se incluyen 4 Gb de RAM y un procesador Intel Atom, 32 GB de almacenamiento, conectividad 802.11ac y 3 puertos USB. Los sensores incluidos son los siguientes: un LiDAR 2D 360º de 12 metros de alcance, dos cámaras gran angular de 4MP para visión estéreo, una IMU (Unidad de Medida Inercial) Bosch BMI 160 con acelerómetro y giroscopio. El vehículo incluye una batería de impulso que alimenta el motor eléctrico, la velocidad máxima del vehículo es de 4 m/s e incluye una dirección activada mediante servos que permite un radio de giro mínimo de 70 cm.

Figura 9.- Vehículo Amazon DeepRacer EVO

En cuanto a conectividad de serie el vehículo tan solo puede comunicarse mediante Wi-Fi. No obstante, el vehículo dispone de varios puertos USB, por lo que se podrá implementar conectividad LTE en este mediante los módems Huawei E3372, compatibles con Ubuntu 16.04.

3.2 Planteamiento de la arquitectura de servicio y colaboración con Grupo Renault España

El proyecto actual nace de una colaboración con Renault, con el objetivo de validar arquitecturas de sistemas V2X de día 1.

En Grupo Renault actualmente se está trabajando en el desarrollo de arquitecturas de sistemas V2X de día 1. Este trabajo de fin de máster se plantea como una demostración de la capacidad de la maqueta para probar escenarios realistas V2X de día 1 para su posterior validación. En este caso se ha elegido implementar un sistema V2I de aviso temprano de posible colisión con otro vehículo, siguiendo una arquitectura próxima a lo que se podría encontrar desplegada en una empresa automovilista actual. Para este proyecto se mantuvo un contacto estrecho y continuado a lo largo de todo el desarrollo, a través de reuniones regulares en las que se discutían los resultados obtenidos y se planteaban ideas nuevas a implementar. Debido a esta relación de colaboración, se garantiza que la implementación del proyecto se encuentra en línea con planteamientos actuales aplicados por fabricantes automovilísticos. Los aspectos contemplados para emular una arquitectura de servicio realista se listan a continuación:

- Como cliente del servicio se utilizará los vehículos Amazon DeepRacer Evo equipados con el modem Huawei E3372. Los vehículos generarán mensajes tipo CAM conformes al estándar ETSI ITS.
- Los mensajes deberán hacer uso de alguno de los estándares de mensajería IoT para ser transmitidos a un servidor que aloje el servicio.
- La infraestructura de la red LTE se implementa mediante el *software* de srsRAN en sistemas Ubuntu. El *hardware* SDR para implementar el eNB utilizado es *BladeRF 2.0 micro xA9.*
- Se implementará una red de acceso GPON entre el eNodeB y el *core* de la red, para emular tanto la separación lógica como geográfica de estos componentes de una red LTE.
- Se utilizará una máquina virtual (VM) para implementar una pasarela de autenticación como solución para preservar la soberanía de datos. Se escogió implementar esta parte del servicio en una VM ofrecida por un proveedor de servicios en la nube con objetivo de poder simular una arquitectura más distribuida lógica y geográficamente. De esta manera alejar este componente del servicio de la infraestructura de la UVa, permitirá emular con mayor fidelidad un servicio comercial realista. Para realizar el proceso de autenticación se hará uso del protocolo TLS con certificados cliente.
- Finalmente, el servidor se alojará en una VM en un servidor dentro de la infraestructura de red de la UVa.

En la Figura 10 se muestra la arquitectura de servicio propuesta para integrar las características deseadas para el servicio. La arquitectura diseñada sigue el usual paradigma cliente/servidor, con las particularidades de que se incluye el acceso de los clientes mediante LTE y GPON, así como la pasarela de autenticación, que actúa como intermediario entre el cliente y el servidor.

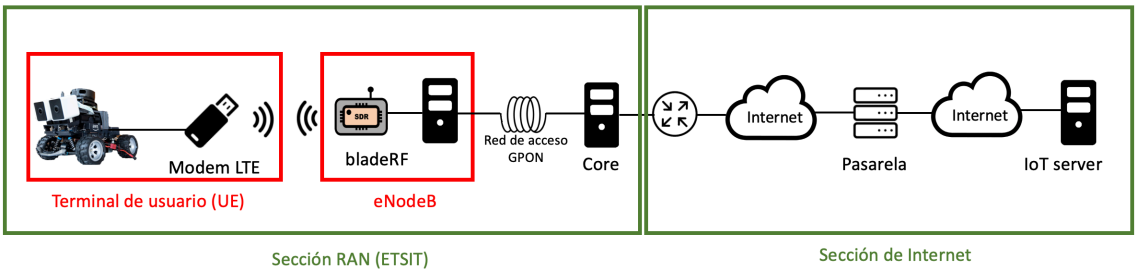


Figura 10.-Arquitectura de servicio planteada.

El desarrollo de esta arquitectura se plantea por fases, comenzando por los componentes lógicos básicos implementados en un entorno local, e ir progresivamente añadiendo y realojando servicios haciendo la arquitectura progresivamente más distribuida.

La filosofía de implementación incremental de la arquitectura tiene además otra ventaja, disponer de varios estados del sistema de diferentes niveles de complejidad configurados y listos para ser levantados. Esto aumenta la versatilidad de la arquitectura de emulación, puesto que abre la posibilidad de realizar pruebas para diferentes niveles de la arquitectura, permitiendo analizar el impacto sobre la arquitectura completa de los diferentes componentes de manera individual.

3.3 Desarrollo del sistema por fases

Como se ha mencionado en la anterior subsección el desarrollo se realizó de manera progresiva y recibiendo constante *feedback* del Grupo Renault durante el proceso. En esta sección se detalla las fases seguidas de manera cronológica hasta la consecución de la implementación de la arquitectura de servicio completa.

3.3.1 Implementación de la red LTE

En esta primera fase se detalla todo el procedimiento seguido para implementar los elementos básicos de la red LTE. El proceso que se detalla a continuación se realizó sobre un PC con un Ubuntu 20.04 con buenas prestaciones *hardware* (CPU Intel i7-9700 de 6 núcleos, 16 GB RAM), puesto que el *software* de srsENB es exigente a nivel computacional.

En primer lugar, habrá que descargar y compilar los drivers de la tarjeta *BladeRF 2.0 micro xA9*. A continuación se muestran los comandos de *Shell* de linux utilizados para ello:

```
$sudo apt install libbladerf2 libbladerf-dev libusb-1.0-0 libusb-1.0-0-dev
$git clone https://github.com/Nuand/bladeRF
$cd bladeRF/host
$mkdir build
$cd build
$cmake ..
$make -j4
$sudo make install
$sudo ldconfig
```

Posteriormente, se instala el *software* srsEPC y srsENB. Esto se aplica con los siguientes comandos *Shell*:

```
$sudo apt-get install -y build-essential cmake libfftw3-dev libmbedtls-dev
libboost-program-options-dev libconfig++-dev libsctp-dev
$git clone https://github.com/srsRAN/srsRAN.git
$cd srsRAN
$mkdir build
$cd build
$cmake ../
$make
$make test
$sudo make install
$sudo srsran_install_configs.sh user
$sudo ldconfig
```

Este es todo el *software* necesario para implementar la red LTE, para utilizarlo correctamente será necesario acceder a los ficheros de configuración `/root/.config/srsran/enb.conf` y `/root/.config/srsran/epc.conf` para los *softwares* srsENB y srsEPC respectivamente.

En el fichero `enb.conf` se permite editar parámetros radio del eNodeB, tales como ancho de banda, frecuencias de operación, potencia de transmisión, etc. Los parámetros radio por defecto no se modificarán, puesto que ofrecen prestaciones suficientes para las características del servicio a desarrollar. En este fichero sí que se editará, sin embargo, los parámetros de configuración MCC (Mobile Country Code) y MNC (Mobile Network Code), parámetros que sirven para que el eNB anuncie tanto el país de operación, como la operadora a la que da servicio. A estos parámetros se les da el valor 901 y 70, valores configurados en las tarjetas SIM programables, y que se asocian a una combinación de MCC y MNC no asignadas a ninguna operadora real.

En el fichero `epc.conf` se configuran las características de los servicios del *core*. En primer lugar, se configurará el servicio MME, encargado de la autenticación de los usuarios que intentan acceder a la red, de nuevo incluyendo los valores MCC y MNC, además de la APN (Access Point Name) de la red. A continuación, se configura el servicio HSS introduciendo la ruta del fichero de base de datos de usuarios, el cual, para funcionar con las tarjetas SIM programables tendrá estos contenidos:

```
sim0,mil,901700000052120,464e9d08edb8b7243a174960a62da208,opc,6c54c1bce5e6
24bcddfaec62ff807299,9000,000000000bfe,9,dynamic
sim1,mil,901700000052121,339e1393952281c83a6d96e2b44179eb,opc,d28bf50e7704
256ae23be188707d2973,9000,0000000004a5,9,dynamic
sim2,mil,901700000052122,1116b35021a7e0784d18f91d7e3bd3c5,opc,81f19fdc9323
293e5e2daf1bd64febdb,9000,00000000039b,9,dynamic
sim3,mil,901700000052123,4055d3c4ea2367fdd41af9bab4cc55c8,opc,c8840a79d631
3d69b9c76b3bcdaa520d,9000,000000000800,9,dynamic
sim4,mil,901700000052124,073dbfd12ac3477e08154d540bc45f25,opc,8229a411867e
487d7613560ee325b511,9000,000000000251,9,dynamic
sim5,mil,901700000052125,97b24ff4b4737112933274b5467a2594,opc,f046807647a4
c1dc606d8712afacc8c0,9000,000000000251,9,dynamic
sim6,mil,901700000052126,a6520d65a41d01fe3a4217ef88886d99,opc,c2fe099cd8ba
d6ac92c607a21a04da73,9000,000000000251,9,dynamic
```

Finalmente se instala en el vehículo uno de los módems LTE para comprobar el correcto funcionamiento de la red. En la Figura 11 se muestra el módem junto con la SIM utilizada para realizar la prueba. La única configuración del módem aplicada será sustituir el valor de la APN al valor utilizado al configurar srsEPC.

Figura 11.- Módem LTE junto con tarjeta SIM programable.

Para configurar correctamente un servicio distribuido, es necesario tener amplio conocimiento del espacio de direcciones IP privadas al que pertenecen diferentes componentes de la red. En la Figura 12 se muestran las subredes existentes en la operación de la red de acceso LTE, con el *core* conectado a la red de la ETSIT (10.0.103.0/24). Por simplicidad para la representación, se muestra los servicios srsENB y srsEPC en máquinas diferentes. A los elementos de infraestructura de la red LTE se les atribuyen direcciones del espacio 172.16.0.0/16. En la Figura 12 se puede observar que en esta subred faltan por etiquetar interfaces, esto se debe a que algunas de estas son invisibles a nivel de red. Finalmente, y marcadamente menos intuitivo, entre el módem LTE y el PC se genera un espacio de direcciones diferente, esto indica que el módem hará una traducción de direcciones tipo NAT para encaminar tráfico hacia la interfaz *ethernet* que este genera en el PC.

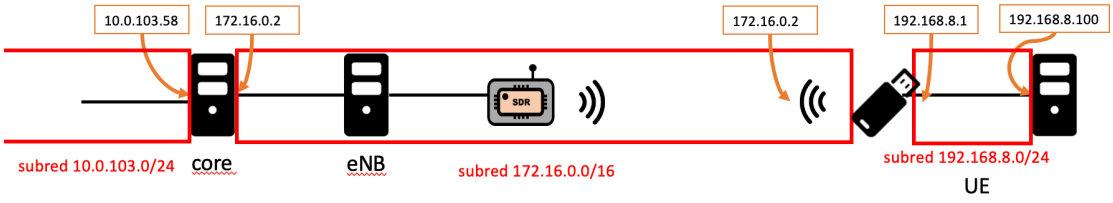


Figura 12.-Direccionamiento IP de la red LTE

Tras aplicar los procedimientos descritos en esta fase, la red LTE opera correctamente, aportando requisitos suficientes para la aplicación a desarrollar. En concreto, para la banda 7 de LTE, se obtiene un máximo de 91,3 Mbps de descarga, 47,4 mbps de subida y una latencia RTT media de 28 ms con la interfaz SGi del *core* (medida mediante el comando `ping` desde un teléfono móvil).

3.3.2 Implementación del protocolo MQTT

Para la correcta transmisión de los datos extremo a extremo será necesario escoger un protocolo de nivel de aplicación. Se exploraron diversas opciones validas para contextos de IoT, de entre las cuales, tras conversaciones con Renault, se decidió optar por MQTT, ya que este es un protocolo diseñado específicamente para aplicaciones IoT, y destaca tanto por su ligereza y eficiencia, como por la abundante documentación existente, ya que es extensivamente utilizado para aplicaciones comerciales.

En primer lugar, en esta fase, se hará una breve revisión de las implementaciones existentes de servidores MQTT con objetivo de seleccionar la más apropiada para el presente proyecto. Existe una gran oferta de implementaciones de servidores MQTT, también denominados brókeres. En internet existe extensa información respecto a las características de estos, no obstante, en la mayoría de los casos la información se centra en diferencias de funcionalidad (ver Figura 13), y no tanto en análisis numérico de métricas de rendimiento [46].

Implemented protocol specific features									
MQTT Brokers	QoS Support	Retain Flag	Persistent Session	Shared Subscriptions	Last Will and Testament	Error Log	Built-in Gateway	MQTT Version	MQTT-SN Support
Mosquitto	0, 1, 2	Yes	Yes	No	Yes	No	No	3.1.1, 5.0	No
Bevywise MQTT Route	0, 1, 2	Yes	Yes	Yes	Yes	Yes, user can view in UI	Yes	3.x, 5.0	Yes
EMQ X	0, 1, 2	Yes	Yes	Yes	Yes	No	No	3.1.1	Yes
HiveMQ CE	0, 1, 2	Yes	Yes	Yes	Yes	No	No	3.x, 5.0	No
HiveMQ	0, 1, 2	Yes	Yes	Yes	Yes	No	No	3.x, 5.0	No
IBM WIoTP Message Gateway	0, 1, 2	Yes	Yes	-	-	Yes	Yes	3.x, 5.0	-
JoramMQ	0, 1, 2	Yes	Yes	Yes	-	Yes	No	3.x	Yes
flespi	0, 1, 2	Yes	Yes	Yes	Yes	Yes	No	3.1,3.1.1,5.0	No
PubSub+	0, 1, 2	Yes	Yes	Yes	Yes	Yes	Yes	3.1.1	No
Thingstream	0, 1, 2	-	-	No	-	-	No	5.0	Yes
VerneMQ	0, 1, 2	Yes	Yes	Yes	Yes	Yes	No	3.x, 5.0	No
RabbitMQ	0, 1	Partial	Yes	No	Yes	Yes	Yes	3.1.1	No
ActiveMQ	0, 1, 2	Yes	Yes	No	Yes	-	No	3.1.1	No
ActiveMQ Artemis	0, 1, 2	Yes	Yes	No	Yes	-	No	3.1.1	No

Figura 13.-Diferencias entre algunas implementaciones de MQTT [46]

Para revisar los estudios comparativos de brókeres MQTT será importante conocer los niveles de servicio MQTT, que impactan en gran medida media a la métrica de latencia. El protocolo MQTT admite 3 niveles de servicio, denominados niveles QoS. Habitualmente el término QoS se aplica para garantía de calidad y de reserva de recursos para determinados servicios dentro de redes de comunicación TCP/IP, no obstante, en el contexto de MQTT los niveles de QoS se refieren al nivel de garantía de recepción de mensaje con el que operará el protocolo [47].

Los tres niveles de QoS definidos son:

- At most once (0): El modo QoS 0 funciona bajo un paradigma *best-effort*, es decir, los mensajes son enviados sin necesidad de confirmación de la recepción. Por tanto, ni el cliente ni el bróker MQTT tendrán conocimiento sobre el estado de una transmisión de información, y, en consecuencia, no se producirán retransmisiones. No obstante, MQTT opera sobre el protocolo TCP, por lo que las transmisiones en este modo tendrán las capacidades de recuperación de errores de dicho protocolo. Este modo de operación tendrá latencia mínima puesto no requiere de confirmaciones, pero, se deberá asumir que ocasionalmente se perderán mensajes.

- At least once (1): En el modo QoS 1 tras la recepción de un mensaje el bróker emitirá un mensaje ACK para confirmar la recepción del mensaje. En este modo, el emisor del mensaje almacenará el mismo hasta que se produzca la confirmación por parte del bróker. Este modo de operación garantiza la entrega de mensajes, sin embargo, en caso de pérdida de un mensaje de confirmación, se producirán duplicados, que no serán reconocidos como tal por el extremo receptor.

  Este nivel garantiza mayor protección ante errores, pero no obstante el periodo de transmisión de un mensaje es mayor, y producirá una mayor ocupación de recursos del canal.

- Exactly once (2) El nivel QoS 2 garantizará que los mensajes son enviados, además de garantizar que no se produzcan duplicados, para ello se basará el proceso de confirmación de un mensaje en el mecanismo *four-part handshake*. Este nivel de servicio se traducirá en las comunicaciones más lentas, puesto que se requerirá de 4 RTTs para finalizar una transmisión, y se ocupará más recursos de la red, no obstante, este nivel de QoS garantiza la transmisión sin errores.

  La elección del nivel de servicio de MQTT implementado dependerá de las características y requisitos de la aplicación implementada. En el caso del *testbed* desarrollado los dos tipos de mensajes generados tienen características muy diferentes.

  En el caso de los mensajes CAM, se trata de mensajes periódicos de importancia generalmente no crítica debido a la alta tasa de actualización (10 Hz), por tanto, para no inundar la red con tráfico innecesario y garantizar una baja latencia, se considera que QoS nivel 0 se adapta a los requisitos de este tipo de tráfico.

  Sin embargo, los mensajes DENM tienen características muy diferentes, se trata de mensajes de importancia crítica, generados en intervalos mayormente impredecibles, por lo que la pérdida de estos mensajes es inadmisible. Por otro lado, también es importante garantizar una baja latencia en la entrega de estos mensajes, puesto que codifican información crítica, y deberán ser notificados a suscriptores de manera inmediata. Por estas razones el modo de transmisión QoS 1 se adapta mejor a las características de este tipo de mensaje.

  Para realizar la comparación entre diferentes implementaciones de bróker MQTT se ha identificado la latencia como la métrica crítica que se busca optimizar dadas las características del proyecto. Además, se considera interesante conocer la variación de esta frente en situaciones de elevada carga por la existencia de un elevado número de publicadores.

  En [48] se realiza un estudio comparativo de los brókeres Mosquitto, Bevywise MQTT, ActiveMQ, HiveMQ, VerneMQ y EMQ X. En el estudio se miden algunas métricas experimentales de los brókeres en una configuración con tres publicadores y tres subscriptores. También se mide la latencia para los diferentes niveles de QoS (ver Figura 14). Finalmente se concluye que para QoS 1 el bróker con el mejor desempeño es Mosquitto, mientras que para QoS 0 el bróker que presenta mejor rendimiento es Bevywise MQTT.

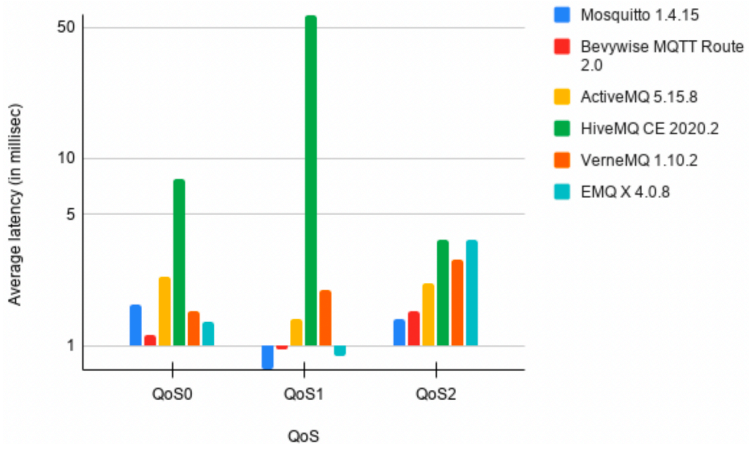


Figura 14.-Latencia de diferentes brókeres para diferentes niveles de servicio

Existen algunos estudios más en lo relativo a comparativas de rendimiento de brókeres MQTT, no obstante, algunos de estos son emitidos por compañías desarrolladoras de los mismos, por lo cual no son considerados fiables [49]. Otros estudios realizan comparativas de otros brókeres diferentes utilizando diferentes métricas y configuraciones, por lo que extraer conclusiones globales de la literatura es complejo.

Finalmente se ha escogido Mosquitto como implementación, tanto por los resultados reflejados en [48] que muestran que se ajusta a los requerimientos del proyecto, como por la calidad de la documentación, que facilita trabajar con este *software*.

Para comprobar el correcto funcionamiento del bróker se ha implementado la arquitectura que se muestra en la Figura 15. Como clientes del servicio se ha utilizado un PC y el Amazon DeepRacer, ambos equipados con módems LTE. El objetivo es que uno de los clientes actúe como publicador, y otro como subscriptor. El bróker Mosquitto se ejecutará en el mismo PC que el *core* y el eNB.

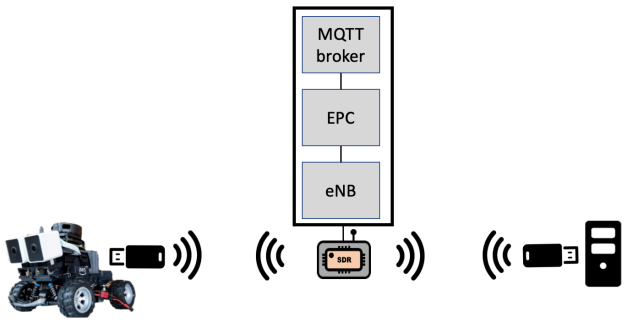


Figura 15.-Arquitectura utilizada para testear el bróker Mosquitto

Mosquitto, como tantos otros *softwares* Linux, se configura a través de un fichero de configuración, aunque de no indicarse uno al ejecutar el programa tomará los valores por defecto. Existe extensa documentación de configuración de Mosquitto en [50]. El fichero de configuración generado para esta fase es muy sencillo, puesto que para testear el servicio será suficiente con los ajustes por defecto en la mayoría de los casos. El fichero se muestra a continuación:

```
bind_address 172.16.0.1
bind_interface srs_spgw_sgi
allow_anonymous true
acl_file /etc/mosquito/aclfile.example
```

En las opciones incluidas se definen la IP en la que escuchará el servicio (el puerto por defecto es 1883), así como la interfaz en la que lo hará. Se incluye la opción `allow_anonymous` que permite a cualquier usuario acceder al bróker sin necesidad de autenticación. Con la línea `acl_file` se define la ruta al fichero ACL, que marca los permisos que tendrán los diferentes tópicos. El fichero ACL utilizado en este caso se configura para que solo se pueda publicar y suscribir en dos tópicos (`casouso/CAM` y `casouso/DENM`). Es importante señalar que estos dos tópicos se refieren a los mensajes del estándar ITS-G5 que deben ser enviados por la aplicación correspondiente, y que será abordada en la siguiente sección. Por ahora, el fichero ACL queda de la siguiente manera:

```
# This affects access control for clients with no username.
topic read casouso/CAM
topic read casouso/DENM
topic write casouso/CAM
topic write casouso/DENM
```

Con esta configuración se lanza el bróker Mosquitto. Para comprobar el funcionamiento del bróker se deberá utilizar *software* cliente MQTT. Para el proyecto se ha utilizado dos clientes MQTT, Paho MQTT, implementado como librería de Python, y los clientes de línea de comandos de Mosquitto, `mosquitto_pub` y `mosquitto_sub`. Al ejecutar Mosquitto con el argumento `-v` (`--verbose`), se mostrará por consola la información en tiempo real de las conexiones de clientes, así como el envío de mensajes debido a publicaciones y suscripciones. A continuación, se muestra la salida por consola del bróker tras interactuar con clientes del tipo `mosquitto_pub` y `mosquitto_sub`.

```
1664877777: New connection from 172.16.0.2:54542 on port 1883.
1664877777: New client connected from 172.16.0.2:54542 as mosqsub/3049-
telecos >
1664877777: No will message specified.
1664877777: Sending CONNACK to mosqsub/3049-telecos (0, 0)
1664877777: Received SUBSCRIBE from mosqsub/3049-telecos
1664877777:     casouso/CAM (QoS 0)
1664877777: mosqsub/3049-telecos 0 casouso/CAM
1664877777: Sending SUBACK to mosqsub/3049-telecos
1664877807: New connection from 172.16.0.2:54544 on port 1883.
1664877807: New client connected from 172.16.0.2:54544 as mosqpub/3150-
telecos >
1664877807: No will message specified.
1664877807: Sending CONNACK to mosqpub/3150-telecos (0, 0)
1664877807: Received PUBLISH from mosqpub/3150-telecos (d0, q0, r0, m0,
'casous>
1664877807: Sending PUBLISH to mosqsub/3049-telecos (d0, q0, r0, m0,
'casouso/C>
1664877807: Received DISCONNECT from mosqpub/3150-telecos
1664877807: Client mosqpub/3150-telecos disconnected.
```

Estos *logs* muestran la publicación desde un cliente (`mosqpub`) en el tópico `casouso/CAM`, y el subsecuente reenvió del mensaje a un subscriptor, en este caso `mosqsub`. Se puede además observar que ambos clientes se conectan al servicio desde la dirección IP 172.16.0.2, dirección asociada a un UE, y perteneciente al espacio de direcciones de la red LTE de srsRAN (ver Figura 12).

3.3.3 Implementación de protocolos ITS-G5

En esta subsección se detallará la fase de desarrollo del servicio en la cual se trabajó para implementar la generación de mensajes CAM y DENM conforme a los estándares ITS-G5, y su subsecuente encapsulado sobre el protocolo MQTT.

En el GCO ya se han desarrollado proyectos centrados en la pila de protocolos ITS-G5, los más recientes [7] y [8]. Ambos trabajos mejoran los protocolos de la pila ITS-G5 ya implementados en aplicación *opensource*, llamada OpenC2X. OpenC2X es una plataforma *software* desarrollada por el grupo de investigación CCS Labs de la Universidad de Paderborn Alemania [51]. Esta plataforma ofrece la implementación de código abierto de una estación ITS, característica muy beneficiosa para el presente proyecto, ya que la gran mayoría de implementaciones ITS-G5, de carácter propietario, dificultan el desarrollo de proyectos de investigación. OpenC2X es compatible con plataformas Linux y soporta la mayoría de los protocolos de la pila ITS-G5, entre los que se incluyen CAM y DENM.

OpenC2X está desarrollado sobre C++, siguiendo un paradigma de diseño modular, donde las diferentes capas del modelo ITS-G5 se encuentran separadas en diferentes programas independientes en constante comunicación entre sí (ver Figura 16).

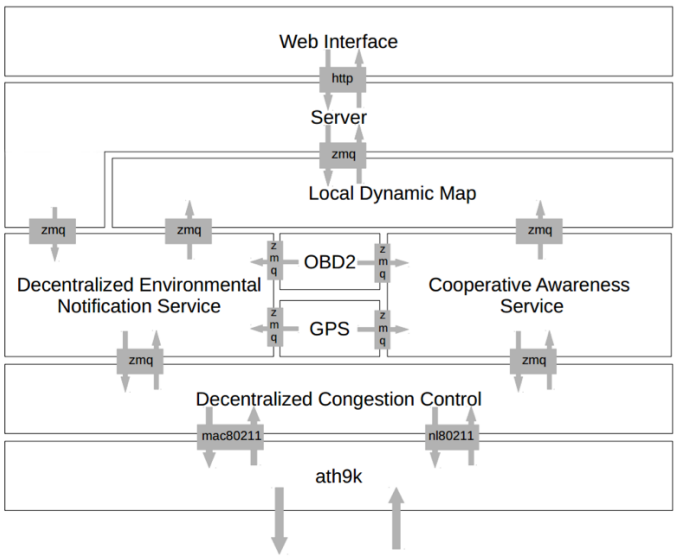


Figura 16.-Diagrama de comunicación de módulos de OpenC2X [51]

El paradigma de diseño modular resulta interesante para el proyecto actual, puesto que permitirá trabajar tan solo con los módulos que resultan relevantes para el desarrollo del proyecto. En concreto serán necesarios lo siguientes módulos:

- **Cooperative Awareness Service**: Módulo encargado de generar mensajes tipo CAM. De ahora en adelante referenciado como módulo CAM.

- **Decentralized Environmental Notification Service**: Módulo encargado de la generación de mensajes tipo DENM. De ahora en adelante referenciado como módulo DENM

- **GPS**: Este módulo transmite muestras GPS a los módulos CAM y DENM para que estos encapsulen esta información en los mensajes en los campos pertinentes. Este módulo puede simular datos, o leerlos de un fichero csv predefinido.

- **OBD2**: Este módulo transmite valores de diferentes sensores del vehículo (velocidad, rpm, consumo, aceleración…) a los módulos CAM y DENM para que estos los incluyan en los campos pertinentes de dichos mensajes. Al igual que el módulo GPS permite simular datos, o también leerlos a través de un puerto USB.

- **HTTPServer**: Este módulo es utilizado por OpenC2X para dar una representación gráfica e intuitiva de los datos mediante un servidor HTTP. La instalación de OpenC2X incluye un fichero HTML que muestra en un navegador la información recibida por el servidor HTTP. Este módulo recibe la información del módulo LDM.

- **LDM**: Base de datos local que almacena los mensajes CAM y DENM.

- **Decentralized Congestion Control (DCC)**: Protocolo encargado de regular el control de flujo de las comunicaciones, con objetivo de preservar la estabilidad y rendimiento del canal.

  El resto de los módulos no son tenidos en cuenta, puesto que al encapsular los mensajes CAM y DENM directamente sobre MQTT se utilizará implícitamente protocolos de capas inferiores del estándar TCP/IP, haciendo innecesaria la implementación de protocolos ITS-G5 de las capas equivalentes.

  Uno de los objetivos del proyecto es implementar las capacidades de generación de mensajes CAM y DENM dentro del Amazon DeepRacer. Para ello se instala el *software* de OpenC2X en el vehículo (recogido en el repositorio [52]), y se comprueba su correcto funcionamiento utilizando un script de *bash* incluido en el repositorio que ejecuta todos los módulos de la plataforma de manera simultánea (ver Figura 17). En cuanto a la arquitectura de servicio utilizada para testear esta configuración se utiliza la configuración de la sección anterior (ver Figura 15), desplegándose por lo tanto un cliente u OBU (on-board unit) en un extremo (el vehículo) y un servidor o RSU (Road Side Unit) en el otro extremo.

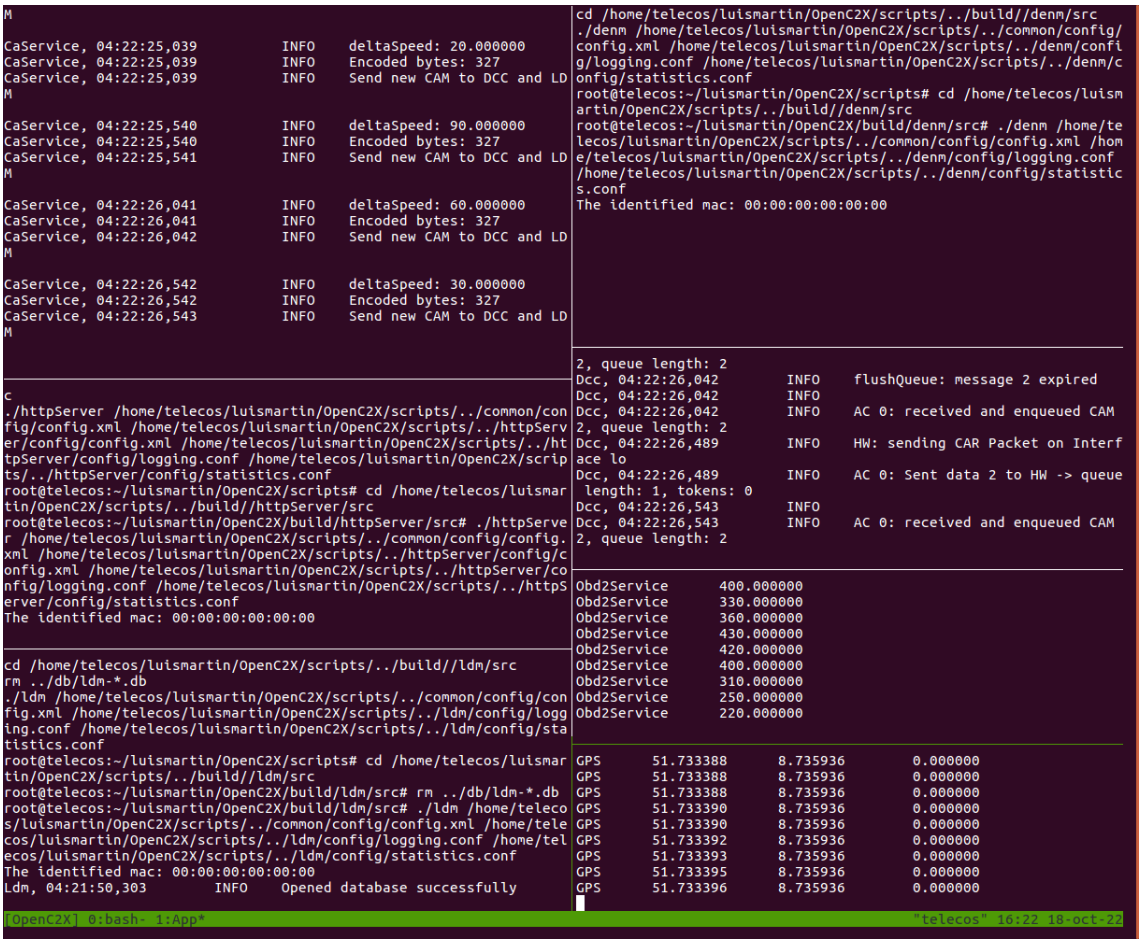


Figura 17.- Software OpenC2X funcionando con todos los módulos en ejecución

Comprobado el correcto funcionamiento de OpenC2X en el equipo, el siguiente paso será el desarrollo de aplicaciones cliente del servicio, cuya funcionalidad principal será la generación y encapsulado en MQTT de mensajes CAM y DENM. Por el momento se implementará dos aplicaciones clientes diferentes para cada uno de los protocolos objetivo. Ambas aplicaciones se implementarán sobre *scripts* de Python.

En primer lugar, se detallará el desarrollo de la aplicación cliente CAM. Ya que la aplicación se implementa sobre Python, el cliente MQTT usado para el encapsulado será Paho MQTT. El primer paso del desarrollo de la aplicación será integrar la generación de mensajes CAM en la ejecución del *script*. Para ello se ejecutará el fichero binario del módulo CAM de OpenC2X mediante una llamada al sistema desde Python con permisos de superusuario (por lo que, por extensión, el *script* de la aplicación debe de ser ejecutado con permisos de superusuario). La ejecución de los módulos se deberá hacer en segundo plano, para que no se detenga en este punto la ejecución del cliente. La sintaxis de la llamada al sistema es la que sigue:

```
os.chdir(openc2x_path+"/build/cam/src")
os.system( "sudo ./cam " + global_config + cam_config
          + logging_conf + statistics_conf + "&")
print("./cam " + global_config + cam_config
          + logging_conf + statistics_conf + "&")
```

Se considera que la generación de datos OBD2 y GPS es de interés para el diseño de escenarios específicos, por lo que se incluye la invocación de dichos módulos mediante llamadas al sistema equivalentes a la llamada al módulo CAM.

El siguiente paso es la captura de los mensajes CAM generados para su posterior encapsulado en MQTT, para ello se explotará las características de la implementación de OpenC2X. Como se puede ver en la Figura 16, los módulos de OpenC2X se comunican entre sí mediante sockets tipo ZMQ, en esta figura se puede ver también que el módulo CAM envía los mensajes generados al módulo LDM. Con esta información, se revisa el código fuente del módulo LDM para ver los detalles de esta comunicación con el objetivo de invocar un *socket* ZMQ de escucha que intercepte estos mensajes "suplantando" al módulo LDM.

Analizando el código del fichero `./ldm/src/ldm.cpp` se observa que los mensajes CAM destinados al módulo LDM se envían al puerto 8888 siguiendo el paradigma ZMQ pub/sub. El paradigma pub/sub de ZMQ es bastante similar al de MQTT en cuanto a su planteamiento, y también utiliza tópicos para organizar la información. Según el código fuente del módulo, este se suscribe a dos tópicos ZMQ, CAM y CAMinfo, siendo el primero donde se publica el mensaje CAM.

El punto elegido para extraer los mensajes CAM no es el más apropiado, puesto que lo correcto sería captar los mensajes enviados por el módulo CA hacia el DCC o bien los enviados por el DCC hacia la capa de enlace, ya que este sería el flujo de mensajes habitual hacia el exterior del vehículo. Sin embargo, esta decisión se debe a que los mensajes entre la capa CA y la capa DCC en OpenC2X no siguen el estándar ITS-G5, debido a que falta por implementar una capa transversal de gestión para pasar información de saturación del canal desde la capa CA a la capa DCC. Esto obliga a modificar los mensajes para incluir dicha información en los mensajes CAM, que dejan de esta manera de ser estándar. Por otra parte, los salientes mensajes de la capa DCC no son tampoco utilizados para evitar que el control de saturación del canal introdujera retardos ficticios en el sistema, dado que dicha capa puede modificar la tasa de salida de mensajes en función del estado del canal [30]. Es evidente que, en el futuro, si se quiere crear un sistema que siga el estándar, se debe mover el punto de recogida de los mensajes a la salida de la capa DCC.

Tras el análisis del código se tiene la información necesaria para definir el *socket* ZMQ de escucha. El *socket* deberá estar suscrito al tópico CAM en la dirección y puerto `127.0.0.1:8888`, un *socket* con estas características se podrá implementar mediante la librería de Python `pyzmq`, de la manera que se muestra en las siguientes líneas de código.

```
import zmq
#Subscripción al socket ZMQ
context = zmq.Context()
sock = context.socket(zmq.SUB)
sock.connect("tcp://localhost:8888")
sock.setsockopt(zmq.SUBSCRIBE, "CAM")
#recepción de mensaje
message = sock.recv()
```

Una vez recibido un mensaje en el *socket* ZMQ se encapsulará en MQTT mediante un cliente Paho MQTT. La arquitectura de servicio MQTT es la misma que se mostraba

en el apartado anterior en la Figura 15, por tanto, la dirección y puerto del bróker siguen siendo `172.16.0.1:1883`. Por defecto los mensajes se publican con el nivel 0 de QoS, por lo que no será necesario indicar explícitamente este valor para la publicación de mensajes CAM. Las líneas de código necesarias para conectarse como publicador a un bróker mediante Paho MQTT son las siguientes:

```
#conexión con el bróker
mqtt_broker="172.16.0.1"
mqtt_port=1883
client=paho.Client("cam_pub")
client.connect(mqtt_broker,mqtt_port)
#publicación de mensaje
client.publish("casouso/CAM",message)
```

Las funciones de subscripción ZMQ y publicación MQTT deberán de estar adecuadamente embedidas dentro de un bucle infinito para asegurar el correcto funcionamiento del cliente del servicio.

Una vez desarrollado el cliente de publicación de mensajes CAM, el siguiente paso es asegurar la integridad de estos en otro extremo de la comunicación. Para ello se desarrolla un cliente suscriptor en Python, implementado sobre otro PC enlazado mediante LTE al bróker (ver Figura 15).

Los mensajes CAM son complejos de interpretar, por lo que asegurar la integridad de los datos resulta farragoso cuando en una traducción a texto plano solo se ve una secuencia de bytes arbitraria. La plataforma OpenC2X brinda un servidor HTTP para visualizar los datos de manera intuitiva, por lo que se plantea utilizarlo para confirmar la correcta recepción de los datos. Por esta razón se decide explotar de nuevo la arquitectura modular de OpenC2X.

De nuevo, se decide implementar este extremo de la comunicación mediante un *script* de Python. De manera análoga al extremo publicador, se utilizan llamadas al sistema para ejecutar en segundo plano, y con permisos de superusuario, los módulos de OpenC2X pertinentes (LDM y httpServer).

Se utiliza el cliente MQTT paho para realizar la conexión al bróker, y la subsecuente subscripción al tópico casouso/CAM. En este caso se utiliza una función implementada en la librería Paho MQTT para forzar el estado de bucle infinito en el programa. Por otra parte, se utiliza un *callback* de la librería para definir el comportamiento del cliente al recibir un nuevo mensaje. El código en cuestión se muestra a continuación:

```
mqtt_broker="172.16.0.1"
mqtt_port=1883
client=paho.Client("cam_sub")
client.on_message = on_message
client.connect(mqtt_broker,mqtt_port,60)
client.loop_forever()
```

En el caso del extremo suscriptor también será necesario suplantar módulos de OpenC2X, en este caso el módulo CAM como fuente de mensajes desde la perspectiva del módulo LDM. Por lo que de nuevo se define un *socket* ZMQ en la dirección y puerto `127.0.0.1:8888`, pero en este caso como publicador.

En este apartado se encontró con que el módulo LDM estaba reconociendo correctamente los mensajes CAM como tal, pero encontraba fallos al decodificar las cabeceras y el cuerpo del mensaje. Para llegar al fondo del problema se analizó el código fuente del módulo CAM en el fichero `./cam/src/caservice.cpp`. En este fichero se encontró que el mensaje se debe enviar como mensaje multiparte ZMQ, un tipo de mensaje en el que varios campos de un mensaje se separan con dos bytes de *buffer* (en este caso tópico y *payload*). Esto explicaría los problemas de decodificación, puesto que anteriormente se estaba mandando el tópico y el *payload* de forma secuencial, y, al faltar los bytes de *buffer*, el módulo LDM interpretaba erróneamente la estructura del mensaje.

El código utilizado para enviar el mensaje multiparte se incluye dentro del *callback* `on_message()` definido anteriormente. La cabecera del mensaje se compone de una cadena de bytes y consiste en los valores ASCII del nombre del tópico ZMQ (CAM).

```
def on_message(client, userdata, msg):
    if(msg.topic=="casouso/CAM"):
        header_bytes=b'\x43\x41\x4d'
        sock.send_multipart([header_bytes, msg.payload])
```

Finalmente, con el *software* de los extremos ya definido, se verifica la correcta recepción de los mensajes CAM en el extremo suscriptor mediante el servidor HTTP de OpenC2X. En la Figura 18 se muestra una captura de un navegador mostrando el fichero HTML vinculado al módulo HTTP de OpenC2X. Se comprueba que la información GPS mostrada en pantalla se corresponde con el fichero .csv de muestras GPS que OpenC2X ofrece como ejemplo. Se verifica que la tasa de recepción de mensajes es de 1 Hz, aunque esta tasa se puede variar mediante el fichero de configuración `/cam/config/config.xml`.

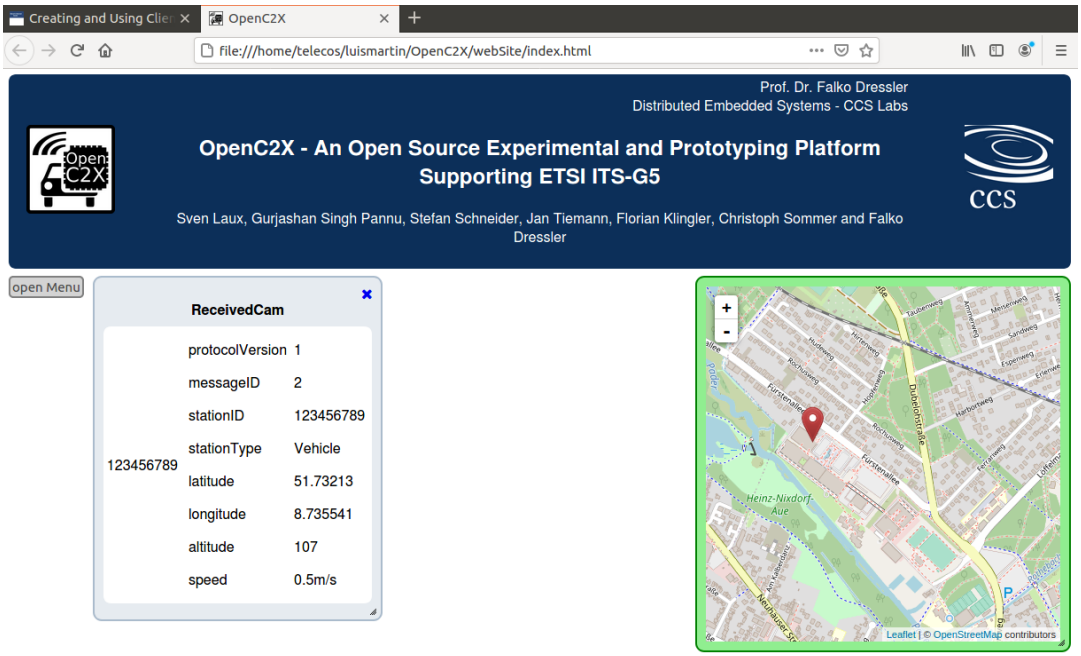


Figura 18.- Servidor web de OpenC2X mostrando la información de los mensajes CAM redibidos mediante MQTT

En cuanto a la implementación de la generación y publicación de mensajes tipo DENM el desarrollo es mayormente redundante respecto a lo ya explicado para los mensajes CAM, aunque existen algunas diferencias que conviene resaltar.

Los mensajes CAM se generan de manera periódica, pero los mensajes DENM requieren de un *trigger*. La solución que plantea OpenC2X es que el usuario indique manualmente el instante de generación de un mensaje DENM mediante la interfaz HTML. Por esta razón, al crear la aplicación generadora de mensajes DENM, además de los módulos DENM, GPS y OBD2, también se invoca el módulo httpServer, utilizado para provocar el *trigger* de los mensajes mediante la GUI de la página web.

De nuevo se aplica el planteamiento de suplantar módulos de OpenC2X para capturar, y reenviar los mensajes DENM. Por tanto, se vuelve a examinar los ficheros de código para averiguar que puertos ZMQ se utilizan. Analizando el fichero `./denmApp/src/denservices.cpp`, se encuentra que, al igual que para el módulo CAM, los módulos se comunican utilizando *sockets* ZMQ mediante el paradigma PUB/SUB, pero en este caso utilizando el puerto 9999.

Teniendo en cuenta estas particularidades, se crean los *scripts* de Python para los dos extremos de la comunicación, teniendo en cuenta además, que en el extremo publicador se utilizara el nivel QoS 1 de MQTT. A continuación, se ejecuta los *scripts* de los dos extremos clientes para comprobar la integridad de los mensajes DENM en recepción. Al igual que para los clientes CAM, se utiliza el fichero HTML vinculado al módulo HTTP para visualizar los mensajes recibidos (ver Figura 19)

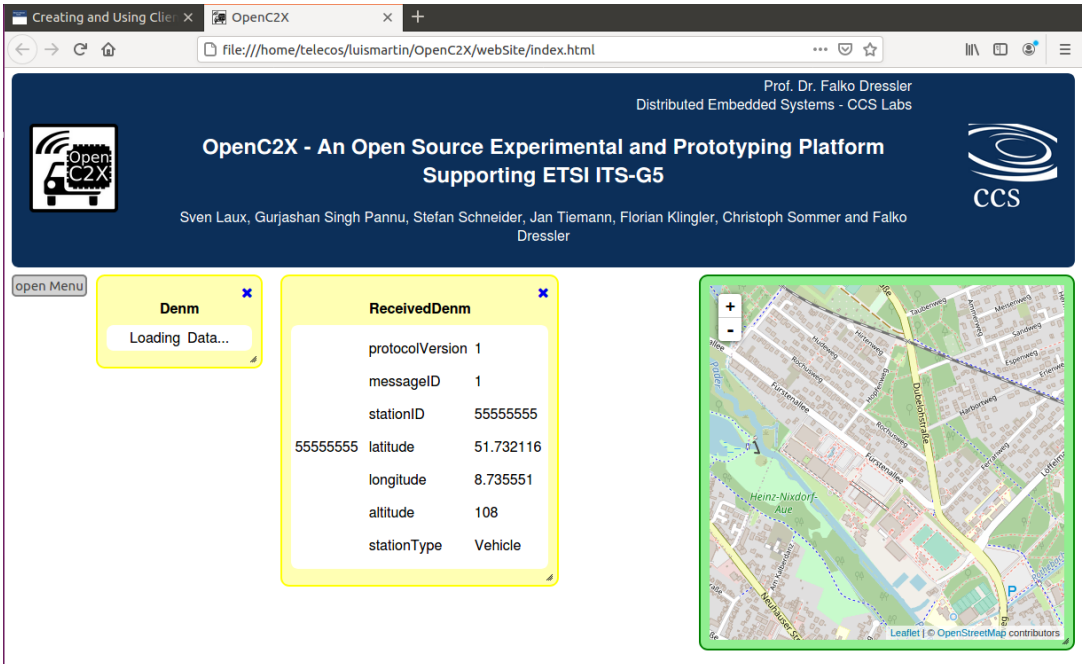


Figura 19.- Servidor web de OpenC2X mostrando la información de los mensajes CAM recibidos mediante MQTT

Finalmente, como última observación, analizando el fichero de código fuente `./denmApp/src/denservices.cpp`, se encuentra que el puerto con el que el módulo DENM se comunica con el módulo httpServer es el puerto 1111, de nuevo, mediante ZMQ PUB/SUB. Analizando mediante el programa Wireshark el tráfico generado en la interfaz *loopback*, y filtrando por el puerto TCP 1111, se podrá observar el mensaje

que provoca el *trigger* del módulo DENM. Este mensaje contiene un mensaje publicado en un tópico denominado TRIGGER, con un *payload* que contiene la cadena "triggered by GUI". Con esta información se abre la posibilidad de provocar el *trigger* de manera totalmente *software*, es decir, sin requerir la interacción de una persona con la GUI HTML, ni depender del módulo httpServer para generar mensajes. No obstante, no se ha llegado a experimentar con esta posibilidad.

3.3.4 Implementación del bróker en una máquina virtual con autenticación TLS

En esta subsección se detalla el desarrollo que tuvo lugar para implementar un bróker distribuido con autenticación y encriptado TLS. Hay que recordar que uno de los objetivos era mantener la seguridad y la soberanía de los datos, por lo tanto, es necesario identificar a los vehículos a la vez que se encriptan las comunicaciones. El bróker con encriptado TLS nos permite asegurar las comunicaciones, así como verificar la identidad de los clientes, mientras que la soberanía de los datos se abordará en un paso posterior cuando se implemente la pasarela.

Con el objetivo de seguir distanciando los componentes del servicio se otorgó al proyecto el acceso a una máquina virtual implementada en un servidor de virtualizaciones ubicado en la infraestructura de red del grupo de laboratorio, aunque a mayor distancia lógica que los nodos ya implementados.

La máquina virtual concedida es una virtualización basada en KVM definida en un servidor que funciona sobre el entorno de virtualziación (VE) Proxmox. Los recursos reservados para la máquina virtual son 80 CPUs virtuales, 128 GiB de RAM y 32 GB de almacenamiento, con Debian 11 como SO virtual. La máquina virtual es accesible desde la red de la escuela, tanto desde el *dashboard* de Proxmox, como desde un terminal mediante SSH.

La máquina virtual (VM) dispone de una única interfaz de red conectada, a la cual tiene asociada una dirección IP local perteneciente a la subred 192.168.168.0/24. Esta subred es una subred privada virtual del servidor Proxmox, y por tanto invisible desde la red de la escuela. Para poder operar un servidor visible desde la red de la escuela será necesario contactar con el administrador del servidor Proxmox para definir reglas NAT de traducción de direcciones IP. Las reglas definidas se muestran en la Tabla I.

Tabla I.-Reglas de puertos definidas para la máquina virtual

Dirección IP entrante	Puerto entrante	Dirección IP saliente	Puerto saliente
10.0.103.69	40007	192.168.168.176	1883
10.0.103.69	40008	192.168.168.176	8883

Una vez definidas las reglas se procede a instalar Mosquitto en esta máquina virtual. En este caso por un conflicto con las fuentes del instalador de paquetes *apt*, resulta imposible instalarlo por ese medio, por lo que se recurre al instalador *snap*. Mediante el programa SFTP se transfiere a la máquina los ficheros de configuración de Mosquitto ya utilizados en apartados anteriores, ya que serán reutilizados pues los cambios serán mínimos, reduciéndose a cambiar la dirección IP asociada al bróker.

El desplazamiento del bróker MQTT a la VM implica un cambio en la arquitectura de servicio que se muestra en la Figura 20. Actualizar el servicio para funcionar con esta arquitectura resulta trivial, consistiendo simplemente en cambiar las direcciones IP y puertos del bróker MQTT en los extremos clientes por las que se muestran en las dos primeras columnas de la primera fila de la Tabla I.

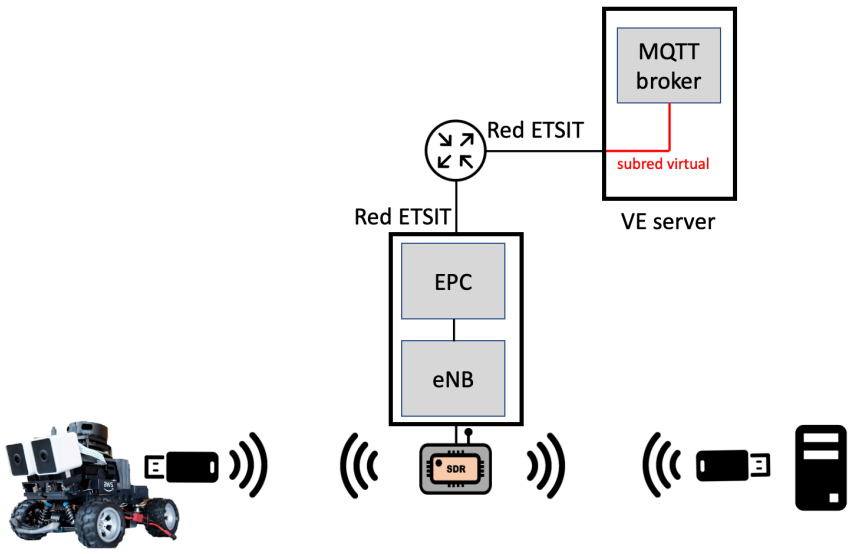


Figura 20.-Arquitectura del servicio en su segunda iteración

En esta fase de desarrollo también se implementó el esquema de autenticación y encriptado mediante TLS. Añadir esta componente al servicio fue relativamente sencillo debido a que Mosquitto incluye soporte nativo del protocolo TLS, por tanto, la dificultad de implementar la securización del servicio radica tan solo en gestionar correctamente los certificados tanto del bróker como de los clientes.

La encriptación TLS puede funcionar tanto con certificados de cliente como sin ellos, aunque en ambos casos deberá estar definido el certificado del servidor. Por simplicidad, se comienza implementando TLS sin certificados clientes.

A continuación, se detalla el proceso de generación de un certificado auto-firmado (*self-signed*) para el bróker. Que el certificado sea *self-signed* implica que el mismo servidor actuará como entidad emisora y certificadora de su propio certificado. Como es natural, habitualmente se desearía que la entidad certificadora fuera un tercero, pero por simplicidad se implementará de esta manera. De cara a esta subsección, es importante resaltar que, a partir de este punto, se hará repetidas menciones tanto a la entidad certificadora (CA) como al bróker, que tienen sentido como entidades lógicas separadas, pero en este caso se encuentran en el mismo nodo físico.

Para comenzar el proceso se accede a la VM mediante SSH, donde se instalará mediante el gestor de paquetes `apt` la librería *software* OpenSSL, que facilita una gran variedad de herramientas criptográficas.

El primer paso será crear el par clave privada/pública para la entidad certificadora (CA), estas claves se crean mediante el algoritmo RSA y podrán ser opcionalmente cifradas mediante una contraseña. La clave privada será utilizada de ahora en adelante por la CA para firmar todos los certificados que esta emita.

El comando de terminal que se muestra a continuación cumple dicha función, en él se indican el tipo de encriptado (`genrsa`) y con el argumento `-des3` se indica que se cifrará el fichero de clave privada con dicho algoritmo de encriptación a partir de una contraseña que se pedirá tras ejecutar la orden. Con el argumento `-out` se indica el fichero de salida y por último el argumento `2048` indica el tamaño de la clave. Se puede observar que existe un fichero único de salida, a pesar de que el comando debe generar un par de claves. Esto se debe a que la clave pública podrá ser obtenida a partir de la clave privada (pero no viceversa por razones evidentes).

```
$openssl genrsa -des3 -out ca.key 2048
```

El siguiente paso será crear el certificado de la entidad certificadora, certificado que será firmado con la clave privada de la CA. Para ello se utiliza la orden que se muestra a continuación; el argumento `req` invoca la herramienta de creación de certificados y ficheros de *request* de OpenSSL, con el argumento `-x509` se indica que se debe generar un certificado autofirmado y no un *request*, y, con el argumento `-days` se indica el número de días de validez del certificado generado.

```
$openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
```

Tras introducir el comando se solicitan una serie de valores que incluir en el certificado, como país de origen, ciudad, nombre de la compañía… El valor de estos campos no afectará al funcionamiento del algoritmo TLS.

A continuación, se genera un par de claves pública/privada para el bróker mediante la orden:

```
$openssl genrsa -out server.key 2048
```

A continuación, se creará un fichero de petición con extensión `.csr`. El fichero de petición es un fichero que emite un servidor donde se incluye la información necesaria para que una CA cree un certificado en su nombre, principalmente la clave pública e información general sobre el servidor.

A continuación, se muestra el comando para crear el fichero de petición, donde con el argumento `-key` se incluye el fichero de par de claves del bróker.

```
$openssl req -new -out server.csr -key server.key
```

Tras introducir el comando se mostrará un formulario como el que se mostró al crear el certificado de la CA. De nuevo, rellenar los campos no es obligatorio, a excepción del campo *common name*, donde se debe incluir el *hostname* al cual identifica el certificado a generar, en este caso, la dirección IP 10.0.103.69.

Habitualmente el fichero de petición se manda a la entidad certificadora para que esta la firme con su clave privada, en este caso, al ser el mismo bróker la CA, el certificado se creará en el mismo servidor.

Por tanto, el siguiente paso es crear el certificado del bróker. Para poder demostrar que el certificado ha sido expedido por la CA este se firmará con su clave privada. Para crear el certificado se utiliza la herramienta `x509` de OpenSSL.

Con el argumento `-req` se indica que la entrada será un fichero de petición, con la opción `-CA` se especifica el certificado de autoridad con el que se firma, con el argumento `-CAkey` se incluye la clave privada de la CA y con el argumento `-days` se especifica el número de días de validez del certificado.

```
$openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial
-out server.crt -days 360
```

Tras finalizar este paso ya se han generado todos los ficheros necesarios para verificar la identidad del bróker. El siguiente paso es configurar el bróker para funcionar sobre TLS.

Primero se copia los ficheros `server.crt`, `server.key` y `ca.crt` a la ubicación `/etc/mosquitto/certs`. A continuación, se añade las siguientes líneas al fichero de configuración `.conf` de Mosquitto.

```
cafile /etc/mosquitto/ca_certificates/ca.crt
keyfile /etc/mosquitto/ca_certificates/server.key
certfile /etc/mosquitto/ca_certificates/server.crt
```

Finalmente, un cliente que inicie el proceso de conexión con el bróker necesitará el certificado de la CA. Esto se debe a que, durante el proceso de establecimiento de una conexión TLS, el bróker enviará su certificado, `server.crt,` para demostrar la veracidad de su identidad. El cliente, por otra parte, necesitará la clave pública de la entidad certificadora para verificar que el certificado del bróker ha sido firmado utilizando la clave privada de la CA. Debido a este propósito, el fichero `ca.crt` se distribuye entre los clientes del servicio mediante el programa SFTP.

El inicio de conexión mediante `mosquitto_pub` y `mosquitto_sub` se realizará con la siguiente orden, donde es importante destacar que se utiliza el puerto 40008 ya que las conexiones TLS de mosquitto se realizan a través del puerto 8883 por defecto (ver Tabla I).

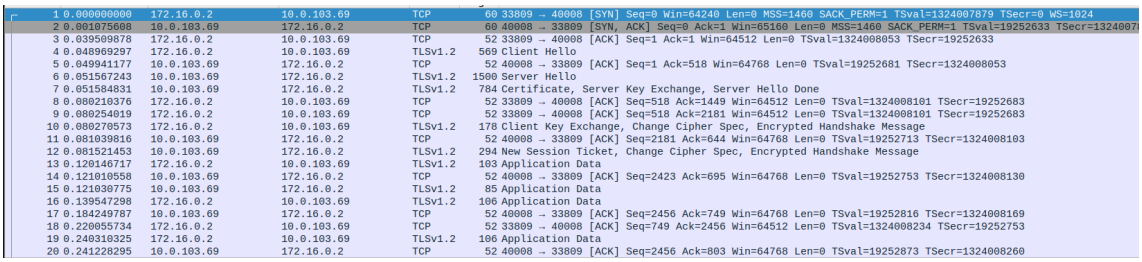
```
$mosquitto_sub -h 10.0.103.69 -p 40008 -t “casouso/CAM” –-cafile ca.crt
```

El inicio de sesión mediante paho MQTT se implementa mediante las siguientes líneas de código Python:

```
client.tls_set("ca.crt")
client.connect(10.0.103.69,40008)
```

Para comprobar que el bróker está funcionando correctamente con el protocolo TLS, se establece una conexión bróker/cliente y, mediante el programa Wireshark, se analiza el tráfico filtrando por puertos. En la Figura 21 se muestra dicha captura de tráfico, donde en los 12 primeros mensajes iniciales se refleja el intercambio que se realiza para el establecimiento de conexión TLS, con el respectivo intercambio de

certificados. A partir de ese punto los mensajes se encuentran cifrados, siendo totalmente ilegibles desde la interfaz de Wireshark.



1	0.000000000	172.16.0.2	10.0.103.69	TCP	60	33809 → 40008 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1324007879 TSecr=0 WS=1024
2	0.001075608	10.0.103.69	172.16.0.2	TCP	60	40008 → 33809 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=19252633 TSecr=1324007
3	0.039509878	172.16.0.2	10.0.103.69	TCP	52	33809 → 40008 [ACK] Seq=1 Ack=1 Win=64512 Len=0 TSval=1324008053 TSecr=19252633
4	0.048969297	172.16.0.2	10.0.103.69	TLSv1.2	569	Client Hello
5	0.049941177	10.0.103.69	172.16.0.2	TCP	52	40008 → 33809 [ACK] Seq=1 Ack=518 Win=64768 Len=0 TSval=19252681 TSecr=1324008053
6	0.051567243	10.0.103.69	172.16.0.2	TLSv1.2	1500	Server Hello
7	0.051584831	10.0.103.69	172.16.0.2	TLSv1.2	784	Certificate, Server Key Exchange, Server Hello Done
8	0.080210376	172.16.0.2	10.0.103.69	TCP	52	33809 → 40008 [ACK] Seq=518 Ack=1449 Win=64512 Len=0 TSval=1324008101 TSecr=19252683
9	0.080254019	172.16.0.2	10.0.103.69	TCP	52	33809 → 40008 [ACK] Seq=518 Ack=2181 Win=64512 Len=0 TSval=1324008101 TSecr=19252683
10	0.080270573	172.16.0.2	10.0.103.69	TLSv1.2	178	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
11	0.081039816	10.0.103.69	172.16.0.2	TCP	52	40008 → 33809 [ACK] Seq=2181 Ack=644 Win=64768 Len=0 TSval=19252713 TSecr=1324008103
12	0.081521453	10.0.103.69	172.16.0.2	TLSv1.2	294	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
13	0.120146717	172.16.0.2	10.0.103.69	TLSv1.2	103	Application Data
14	0.121010558	10.0.103.69	172.16.0.2	TCP	52	40008 → 33809 [ACK] Seq=2423 Ack=695 Win=64768 Len=0 TSval=19252753 TSecr=1324008130
15	0.121030775	10.0.103.69	172.16.0.2	TLSv1.2	85	Application Data
16	0.139547298	172.16.0.2	10.0.103.69	TLSv1.2	106	Application Data
17	0.184249787	10.0.103.69	172.16.0.2	TCP	52	40008 → 33809 [ACK] Seq=2456 Ack=749 Win=64768 Len=0 TSval=19252816 TSecr=1324008169
18	0.220055734	172.16.0.2	10.0.103.69	TCP	52	33809 → 40008 [ACK] Seq=749 Ack=2456 Win=64512 Len=0 TSval=1324008234 TSecr=19252753
19	0.240310325	172.16.0.2	10.0.103.69	TLSv1.2	106	Application Data
20	0.241228295	10.0.103.69	172.16.0.2	TCP	52	40008 → 33809 [ACK] Seq=2456 Ack=803 Win=64768 Len=0 TSval=19252873 TSecr=1324008260

Figura 21.- Captura de tráfico TLS sin certificados cliente

La creación de un certificado de servidor auto-firmado permitirá a los clientes MQTT verificar la identidad del bróker, así como el establecimiento de un canal de comunicaciones seguro en base al intercambio de claves criptográficas incluidas en el propio certificado. Con esta configuración aún existe una vulnerabilidad de seguridad evidente, la ausencia de requerimiento de identificación de los clientes.

Para implementar un esquema de autenticación de clientes se utilizará la autoridad del bróker como entidad certificadora para emitir y firmar certificados de clientes. A partir de dichos certificados, los clientes demostrarán de manera unívoca que su identidad ha sido verificada previamente por el propio bróker.

El proceso que se detalla a continuación deberá ser llevado a cabo en cada cliente del servicio, es decir, no se podrá utilizar un mismo certificado cliente para más de un cliente simultáneo, ya que, desde el punto de vista del bróker el *common name* incluido en el certificado será utilizado como identificador único de cliente.

En primer lugar, se comenzará con la creación de un par de claves pública/privada para el cliente:

```
$openssl genrsa -out client.key 2048
```

A continuación, se crea un fichero de petición de certificado de manera análoga a como se hizo previamente para la creación del certificado del bróker:

```
$openssl req -new -out client.csr -key client.key
```

Al introducir este comando, de nuevo, se muestra un formulario para incluir información en el certificado a generar. En este caso, de nuevo, el único valor relevante es el de *common name*, ya que, como ya se mencionó anteriormente, este será utilizado por el bróker para identificar al cliente.

El siguiente paso será generar el certificado para el cliente, para ello será necesario transferir mediante SFTP el fichero de petición que se acaba de generar a la CA. La CA utilizará el fichero de petición para generar el certificado firmado con su clave privada. Para ello se vuelve a utilizar la herramienta `x509` de OpenSSL de la siguiente manera:

```
$openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial
-out client.crt -days 360
```

Tras generar el certificado se envía al cliente pertinente mediante SFTP.

Todo este proceso se repite dos veces para los dos clientes que actualmente se implementan (ver Figura 20).

Para hacer efectiva la nueva configuración del bróker se vuelve a editar el fichero de configuración de Mosquitto, al que se añaden dos líneas: una para habilitar el uso de certificados cliente, y otra que habilita el uso del *common name* del certificado como identificador del cliente.

```
require_certificate true
use_identity_as_username true
```

Con esta configuración, los clientes necesitarán tres ficheros para realizar la conexión con el bróker. En primer lugar, el certificado de la CA, `ca.crt`, por las razones ya explicadas anteriormente. También será necesario el certificado de cliente `client.crt`, para indicar al servidor cuál es la identidad del cliente, así como su clave privada, `client.key`, para que los clientes puedan demostrar al bróker que la clave pública incluida en el certificado de cliente se corresponde con su clave privada.

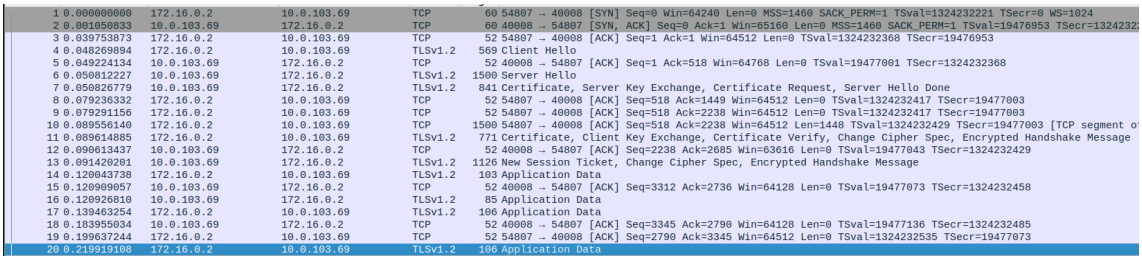
La conexión mediante `mosquitto_pub` y `mosquitto_sub` se realiza con el siguiente comando:

```
$mosquitto_sub -h 10.0.103.69 -p 40008 -t “casouso/CAM” –-cafile ca.crt
--cert client.crt –-key client.key
```

Para Paho MQTT la conexión se realizará mediante las siguientes líneas de código Python:

```
client.tls_set("./ca_certificates/ca.crt","./ca_certificates/client.crt","
./ca_certificates/client.key")
client.connect(mqtt_broker,mqtt_port)
```

Para comprobar el correcto funcionamiento de la configuración se analiza el tráfico mediante *wireshark* para comprobar si está encriptado. En la Figura 22 se muestra una captura del tráfico, donde los 13 primeros paquetes se corresponden con el establecimiento de la conexión TLS. Se pueden ver ciertas diferencias en el establecimiento de la conexión con el establecimiento sin autenticación de cliente (ver Figura 21), como, por ejemplo, en los paquetes 7 y 11, donde se observa cómo se realiza el intercambio del certificado cliente con indicadores “*certificate request*” y “*certificate verify*”. Tras finalizar el intercambio de establecimiento de conexión, en el paquete 14 comienza el intercambio de datos, los cuales se encuentran encriptados correctamente ya que son ilegibles desde la captura de tráfico.



1	0.000000000	172.16.0.2	10.0.103.69	TCP	60	54807 → 40008 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1324232221 TSecr=0 WS=1024
2	0.001050833	10.0.103.69	172.16.0.2	TCP	60	40008 → 54807 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=19476953 TSecr=1324232
3	0.039753873	172.16.0.2	10.0.103.69	TCP	52	54807 → 40008 [ACK] Seq=1 Ack=1 Win=64512 Len=0 TSval=1324232368 TSecr=19476953
4	0.048269894	172.16.0.2	10.0.103.69	TLSv1.2	569	Client Hello
5	0.049224134	10.0.103.69	172.16.0.2	TCP	52	40008 → 54807 [ACK] Seq=1 Ack=518 Win=64768 Len=0 TSval=19477001 TSecr=1324232368
6	0.050812227	10.0.103.69	172.16.0.2	TLSv1.2	1500	Server Hello
7	0.050826779	10.0.103.69	172.16.0.2	TLSv1.2	841	Certificate, Server Key Exchange, Certificate Request, Server Hello Done
8	0.079236332	172.16.0.2	10.0.103.69	TCP	52	54807 → 40008 [ACK] Seq=518 Ack=1449 Win=64512 Len=0 TSval=1324232417 TSecr=19477003
9	0.079291156	172.16.0.2	10.0.103.69	TCP	52	54807 → 40008 [ACK] Seq=518 Ack=2238 Win=64512 Len=0 TSval=1324232417 TSecr=19477003
10	0.089556140	172.16.0.2	10.0.103.69	TCP	1500	54807 → 40008 [ACK] Seq=518 Ack=2238 Win=64512 Len=1448 TSval=1324232429 TSecr=19477003 [TCP segment o
11	0.089614885	172.16.0.2	10.0.103.69	TLSv1.2	771	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
12	0.090613437	10.0.103.69	172.16.0.2	TCP	52	40008 → 54807 [ACK] Seq=2238 Ack=2685 Win=63616 Len=0 TSval=19477043 TSecr=1324232429
13	0.091420201	10.0.103.69	172.16.0.2	TLSv1.2	1126	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
14	0.120043738	172.16.0.2	10.0.103.69	TLSv1.2	103	Application Data
15	0.120909057	10.0.103.69	172.16.0.2	TCP	52	40008 → 54807 [ACK] Seq=3312 Ack=2736 Win=64128 Len=0 TSval=19477073 TSecr=1324232458
16	0.120926810	10.0.103.69	172.16.0.2	TLSv1.2	85	Application Data
17	0.139463254	172.16.0.2	10.0.103.69	TLSv1.2	106	Application Data
18	0.183955034	10.0.103.69	172.16.0.2	TCP	52	40008 → 54807 [ACK] Seq=3345 Ack=2790 Win=64128 Len=0 TSval=19477136 TSecr=1324232485
19	0.199637244	172.16.0.2	10.0.103.69	TCP	52	54807 → 40008 [ACK] Seq=2790 Ack=3345 Win=64512 Len=0 TSval=1324232535 TSecr=19477073
20	0.219919108	172.16.0.2	10.0.103.69	TLSv1.2	106	Application Data

Figura 22.- Captura de tráfico TLS con certificados cliente

3.3.5 Implementación de una pasarela de autenticación

Tras trabajar en la implementación de autenticación de clientes mediante el protocolo TLS en un entorno local, el siguiente paso fue desplazar esta funcionalidad del servicio a un equipo ubicado fuera del entorno físico de la universidad. En este apartado se detalla cómo se implementó una pasarela de autenticación TLS mediante recursos facilitados por un proveedor de servicios *cloud.* Este componente se añade puesto que, pese a no ser estrictamente necesario para el funcionamiento del sistema, será habitual encontrar pasarelas de este tipo en arquitecturas de servicio comerciales equivalentes a la aquí diseñada, siendo el objetivo de esta garantizar la soberanía de datos de los actores privados implicados (ver 2.4).

Existe una gran variedad de proveedores de servicios de *cloud computing*, entre los cuales los más populares son aquellos ofrecidos por grandes multinacionales tecnológicas tales como Amazon, Google, Microsoft o Alibaba. En la actualidad los dos mayores proveedores de servicios *cloud*, Amazon Web Services (AWS) y Microsoft Azure, controlan aproximadamente el 50% del mercado, aunque Microsoft Azure presenta una tasa de crecimiento elevado, mientras que AWS se encuentra en contracción [53]. Por esta razón que se escoge Microsoft Azure como opción para implementar la pasarela de autenticación, ya que, aunque la mayoría de los proveedores ofrecen servicios y prestaciones similares, la tasa de penetración de mercado de Azure es muy elevada, y mantiene perspectivas de crecimiento en el futuro cercano.

Microsoft Azure ofrece servicios de alquiler de Máquinas Virtuales, cuyo coste horario varía en función de las prestaciones del *Hardware* virtualizado. Debido a lo limitado en escala de la implementación por el momento, no será necesario disponer de recursos *hardware* de altas prestaciones dado el bajo número de clientes con los que se está experimentando. Por esta razón el tamaño de VM con el que se ha trabajado, denominado *standard B1,* es el más limitado en cuanto a recursos, pero también el más económico. Entre las prestaciones de este tamaño de VM se encuentran 1 vCPU, 0.5 GiBs de memoria RAM, 4 GiB de almacenamiento SSD y Ubuntu 18.04 LTS como SO.

Para simplificar la configuración del servicio se buscaba que la VM dispusiera de una interfaz de red con una dirección IP pública asociada. Azure ofrece alquiler de direcciones IP públicas, pero tasa este servicio de manera separada al de la VM, por lo que se incluye este servicio en el plan contratado.

Además del alquiler de recursos de red, Azure también ofrece microservicios para implementar en aplicaciones de todo tipo. La arquitectura de microservicios ofrece una gran cantidad de beneficios para facilitar el desarrollo de aplicaciones desde un alto

nivel de abstracción. No obstante, el uso de dichos microservicios requiere de familiaridad y experiencia con el entorno de desarrollo de Azure, y debido tanto a limitaciones de tiempo, como a la sencillez implementación de los objetivos del proyecto en este particular, se consideró más práctico implementar los componentes del servicio a nivel de sistema operativo, como se ha venido haciendo hasta el momento.

Desde la plataforma web de Azure se podrá manipular la VM mediante una GUI, pero se decide trabajar con esta mediante acceso SSH, ya que, debido a la disposición de una dirección IP pública, la máquina se encuentra visible para todo Internet. Gracias a esto, y al programa SFTP, se traslada y adapta la configuración previa del bróker con facilidad. Para configurar el protocolo TLS, se repite todo el procedimiento realizado para el apartado anterior.

Tras replicar los pasos de fases anteriores, se verifica con éxito el correcto funcionamiento de la implementación mediante los clientes MQTT ya mencionados. Una vez comprobado el funcionamiento de las comunicaciones MQTT con la VM de Azure, el siguiente paso será configurar esta como una pasarela entre el cliente y el servidor principal, esto es, tendremos dos brókeres MQTT, uno que hará de pasarela y otro que constituirá el servidor de mensajes CAM y DENM, que también trabajará con un bróker MQTT. Así pues, el problema que se plantea en este momento es cómo los dos brókeres MQTT pueden compartir información de una manera segura (encriptada) y además manteniendo la soberanía de los datos.

Existe una solución no estandarizada para implementar comunicación entre dos brókeres MQTT, MQTT *bridging*. La mayoría de brókeres MQTT comerciales implementan soluciones de este tipo, entre los que se encuentra Mosquitto. El MQTT *bridging* consiste en un bróker MQTT que opera como un cliente de otro bróker, donde, mediante configuración, se podrá escoger qué tópicos son compartidos entre ambos. Este planteamiento implica que, desde el punto de vista del servidor principal, toda la información reenviada por el bróker pasarela provendrá de un solo cliente MQTT (el propio bróker pasarela). Debido a que los protocolos CAM y DENM ya incluyen identificadores propios, esta característica resulta beneficiosa, ya que contribuye a la anonimización de los datos frente al servidor principal. Con el conocimiento de esta opción, se muestra en la Figura 23 la arquitectura de servicio que se implementará en la presente fase de desarrollo.

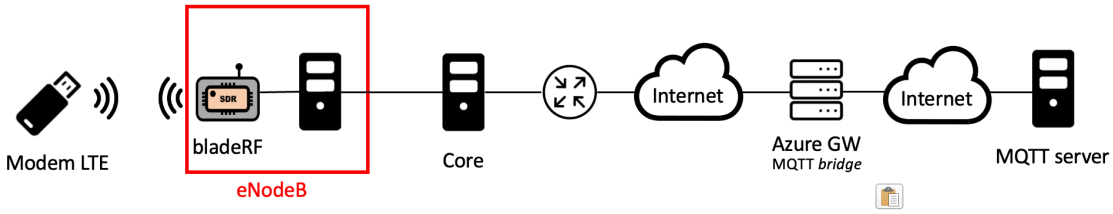


Figura 23.- Arquitectura del servicio en su tercera iteración

Para poder configurar correctamente el bróker *bridge* será necesario que el servidor principal, tenga asociada una dirección IP visible desde el bróker *bridge*. Esto resulta problemático ya que la máquina virtual alojada en el entorno virtual (Virtual Enviornment, o VE) la ETSIT, esta máquina virtual tan solo dispone de una interfaz asociada una dirección IP privada, enmarcada dentro del espacio de direcciones de la

subred de la escuela. Existen varias maneras de circunvalar este problema, la solución que se aplicará será instalar la VPN de la escuela en la máquina virtual de Azure, consiguiendo de esta manera que este equipo disponga de una dirección IP privada de la subred de la escuela.

La VPN de la ETSIT esta implementada mediante el *software* IPsec StrongSwan, en concreto, la VPN implementa el protocolo IKEv2, y soporta autenticación EAP-MSCHAPv2 y con certificado cliente. Para poder instalar la VPN de la escuela en la VM de Azure, el administrador de la red cedió los certificados de la CA y del servidor VPN, así como un certificado cliente específicamente generado para conceder acceso a la máquina.

Para configurar la VM como cliente de la VPN se deberá instalar el cliente StrongSwan mediante el instalador *apt*. Los certificados se transfieren a la máquina virtual mediante el programa SFTP. Para instalar la VPN correctamente el primer paso será editar el fichero de configuración `/etc/ipsec.conf`, donde se añade las siguientes líneas:

```
conn ikev2-rw
    right=vpn.tel.uva.es
    rightid=%vpn.tel.uva.es
    rightsubnet=0.0.0.0/0
    rightauth=pubkey
    leftsourceip=%config
    leftauth=eap
    eap_identity=jmarrob
    auto=start
```

En el apartado `conn` se asocia un alías a la configuración de la VPN. En la nomenclatura utilizada por strongSwan se identifica la configuración del lado del cliente como `left`, mientras que el lado del servidor se identifica como `right`. Por tanto, en la línea `right` y `rightid` se indica el *hostname* del servidor VPN, en la línea `rightsubnet` se debe indicar el valor del espacio de direcciones, en este caso se deja 0.0.0.0, el valor por defecto, finalmente en el valor rightauth se define el tipo de autenticación, en este caso EAP. En cuanto a las opciones añadidas para el extremo `left`, se incluye las opciones `leftauth`, donde se especifica que se utilizará autenticación EAP, y `eap_identity` para indicar la identidad.

Tras editar el fichero el siguiente paso es editar el fichero `/etc/ipsec.secrets`, donde se añade la identidad que el administrador de red otorgo al proyecto, así como la contraseña para desencriptar el fichero:

```
RSA <private_key.file> "passphrase to decrypt key, if any"
jmarrob : EAP "*******"
```

A continuación se copia los ficheros de certificados al directorio de configuración de IPsec, en concreto se copia el certificado de la autoridad al directorio `/etc/ipsec.d/cacerts`, y los certificados de cliente y servidor al directorio `/etc/ipsec.d/certs`.

Finalmente se levanta el servicio IPsec, y se activa la conexión a la VPN mediante los siguientes comandos:

```
$sudo ipsec start
$sudo ipsec up ikev2-rw
```

Finalmente, tras realizar todos estos pasos se utiliza el comando `traceroute` sobre el servidor VE de la ETSIT para comprobar que se ha establecido el túnel VPN correctamente,

```
joseluismartin@Gateway:~$ traceroute 10.0.103.69
traceroute to 10.0.103.69 (10.0.103.69), 64 hops max
  1   157.88.129.84                      26.535ms  26.681ms  26.522ms
  2   10.0.103.69                        26.932ms  26.712ms  27.000ms
```

El paso final de esta fase es configurar correctamente dos brókeres, uno en la VM de Azure, que funciona como *bridge*, y un segundo bróker en la máquina servidora alojada en el VE de la ETSIT.

Para configurar el bróker alojado en Azure como *bridge* se añade las siguientes líneas al fichero de configuración `.conf`.

```
connection azure
address 10.0.103.69:40008
topic # both 0
bridge_cafile /etc/mosquitto/certs/ca.crt
```

Para iniciar la configuración del *bridge* se debe incluir en primer lugar la opción `connection`, que define el comienzo en el fichero de las directivas orientadas a la configuración del bróker cómo *bridge*, y además define el identificador que el bróker *bridge* utiliza como cliente para la conexión con el segundo bróker. La opción `address` define la dirección IP y puerto del segundo bróker, con la opción `topic` se define reglas que conforman el comportamiento por tópico del bróker, en este caso, se indica que todos los tópicos, tanto entrantes como salientes pasen por el *bridge*. Por último, con la opción `bridge_cafile`, se añade el certificado del CA que firma el certificado de servidor del segundo bróker, en este caso se utiliza el mismo que para las fases anteriores, lo cual implica que se debe generar un certificado servidor en el segundo bróker, y transferirlo a la VM de Azure para firmarlo con el certificado CA.

Finalmente, al igual que en el resto de fases, se utilizan los clientes MQTT para verificar el correcto funcionamiento de la arquitectura implementada. Los clientes se deben configurar con la dirección y puerto del bróker *bridge* como dirección servidora, ya que este es el encargado de reenviar la información al segundo bróker. A continuación, se muestra la salida por consola del segundo bróker, donde se ve reflejado el proceso de conexión con el *bridge*, así como una publicación reenviada por este.

```
1665995439: mosquitto version 2.0.15 running
1665995473: New connection from 10.0.103.254:34822 on port 1883.
1665995473: New bridge connected from 10.0.103.254:34822 as Gateway.azure
(p1, c0, k60).
1665995473: Will message specified (1 bytes) (r1, q1).
1665995473: $SYS/broker/connection/Gateway.azure/state
1665995473: Sending CONNACK to Gateway.azure (0, 0)
1665995473: Received PUBLISH from Gateway.azure (d0, q1, r1, m1,
'$SYS/broker/connection/Gateway.azure/state', ... (1 bytes))
1665995473: Sending PUBACK to Gateway.azure (m1, rc0)
1665995473: Received SUBSCRIBE from Gateway.azure
1665995473: # (QoS 0)
1665995473: Gateway.azure 0 #
1665995473: Sending SUBACK to Gateway.azure
1665995475: Received PUBLISH from Gateway.azure (d0, q0, r0, m0,
'casouso/CAM', ... (4 bytes))
```

3.3.6 Implementación de acceso mediante una red GPON

En este apartado se desarrolla la última fase requerida para finalizar la implementación arquitectura de servicio que se planteó al inicio de la sección. Esta última fase se desarrolla el proceso seguido para implementar una red de acceso óptica tipo GPON para la interconexión entre la estación base LTE y el *core* de la red.

Para implementar la infraestructura de una red PON de nuevo se hizo uso de recursos del GCO puestos a disposición del proyecto. El GCO dispone de una red GPON en el laboratorio 2L007, para uso tanto para docencia como para desarrollo de proyectos de investigación. La red GPON del laboratorio 2L007 dispone del siguiente *hardware:*

- 1 OLT (Optical Line Termination).
- Un PC con *software* de gestión de una red GPON, denominado TGMS (Telnet GPON Management System).
- 20 km de fibra óptica monomodo.
- 4 nodos ópticos terminales o ONT de nivel 2. Estos equipos son los que se instala en domicilios para dar acceso a la red GPON, su función consiste en operar como modem para transformar una señal óptica a pulsos eléctricos, habituales conformes al estándar ethernet. El nivel 2 implica que tan solo implementa los dos niveles más bajos de la pila de protocolos TCP/IP.
- 4 ONT de nivel 3. Nivel 3 implica que opera en las 3 capas más bajas del modelo TCP/IP. A diferencia de una ONT de nivel 2, una ONT de nivel 3 implementa el nivel de red, y por tanto, funcionalidades típicamente asociadas a un *router*, como reenvío IP, traducción de direcciones NAT o DNS.
- Infraestructura de red Ethernet interconectada mediante Switches.
- Interconexión con fibra monomodo entre el laboratorio del proyecto, y el laboratorio donde se encuentra la GPON.

La primera tarea a desarrollar para desplegar el enlace GPON entre el eNodeB y el *core*, fue trasladar al laboratorio 2L007 el PC en el que, hasta el momento, se implementaba los elementos de red LTE en el laboratorio del proyecto (2L028). Este PC se conectará a la infraestructura de la red GPON, y será encargado de actuar como *core* de la red LTE, por otra parte, en el laboratorio 2L028 se utilizará el equipo Intel NUC para implementar la funcionalidad del eNodeB.

La arquitectura de la red GPON que se plantea se muestra en la Figura 24, donde se excluyen algunos elementos pasivos de la red como divisores o distribuidores, puesto que no contribuyen al comportamiento lógico de la arquitectura.

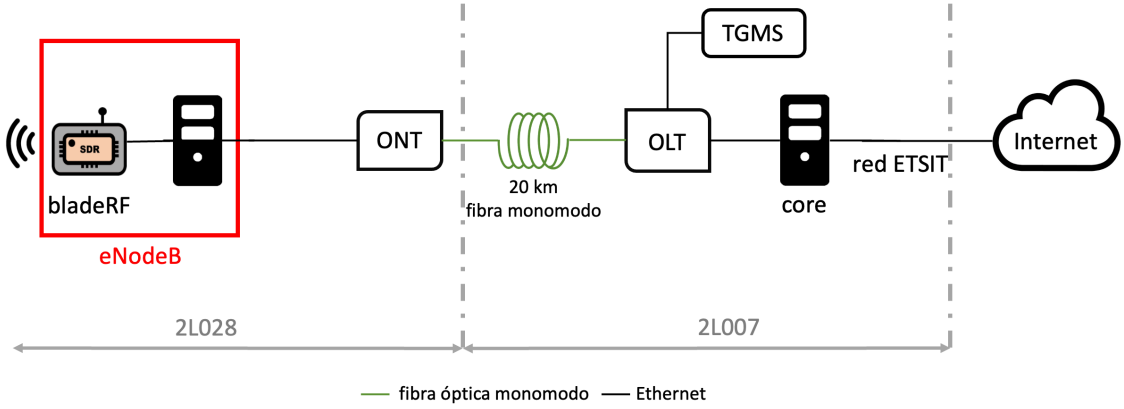


Figura 24.-Arquitectura de la red de acceso GPON

El *software* srsRAN contempla la posibilidad de separar los nodos srsENB y srsEPC en diferentes máquinas, requiriendo tan solo de la definición de IPs de escucha y transmisión en ambos extremos. Por extensión, la configuración de los nodos será considerablemente más sencilla si ambos equipos disponen de interfaces de red asociadas a un espacio de direcciones IP común. Esto será sencillo de implementar sin necesidad de recurrir a aplicar reglas NAT de traducción de direcciones, puesto que, el nodo OLT no implementa protocolos superiores al nivel 2 del estándar TCP/IP y, si se escoge utilizar una ONT de nivel 2, no existirán equipos que implementen nivel de red entre los nodos eNB y EPC.

El primer paso para configurar la red GPON será realizar las conexiones físicas de los diferentes elementos que la conforman. Una vez realizadas estas, será necesario habilitar la ONT de nivel 2 para la OLT a través del nodo de configuración TGMS. El nodo de configuración TMGS se encuentra implementado sobre una máquina virtual en un PC Linux ubicado en el laboratorio 2L007. Para lanzar el nodo de configuración, primero se levanta una máquina virtual mediante el *software* Oracle VirtualBox, esta VM aloja como servicio un servidor HTTP en la dirección IP 172.26.128.1, donde se podrá manipular la configuración del nodo OLT. Accediendo a dicha dirección IP mediante el navegador se muestra un menú donde se muestran todas las ONT conectadas al OLT (ver Figura 25), desde este menú se habilita la ONT utilizada, identificada por el ID 544C52495B02E9CA (ver Figura 26).

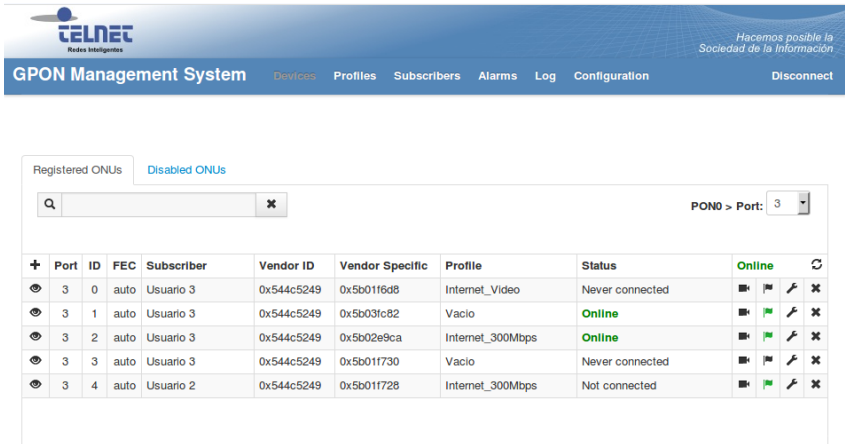


Figura 25.-Listado de terminales ONTs conectadas al nodo OLT

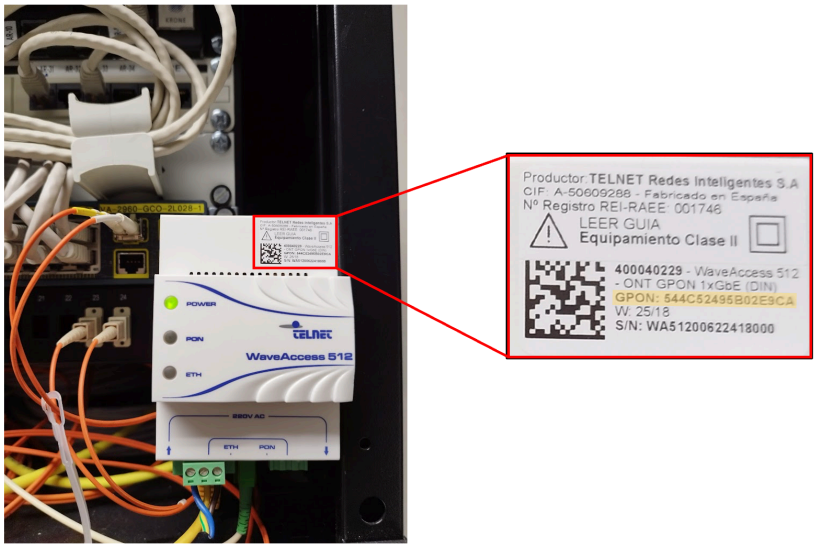


Figura 26.-ONT de nivel 2 en el punto de instalación

Debido a que el *hardware* que compone la infraestructura de la red GPON es de nivel 2 (ver Figura 24), será necesario hacer uso de VLANs para garantizar el correcto encaminamiento del tráfico. En la Figura 26, se muestran las diferentes VLAN IDs que se asociará a las interfaces de los nodos eNB y *core*, se escoge la VLAN 833, ya que está definida por el administrador de la red de la escuela para dar salida a Internet. A pesar de que el nodo *core* implementa capa de red, y, por tanto, las VLANs de cada interfaz serán independientes, se escoge utilizar el mismo ID por simplicidad.

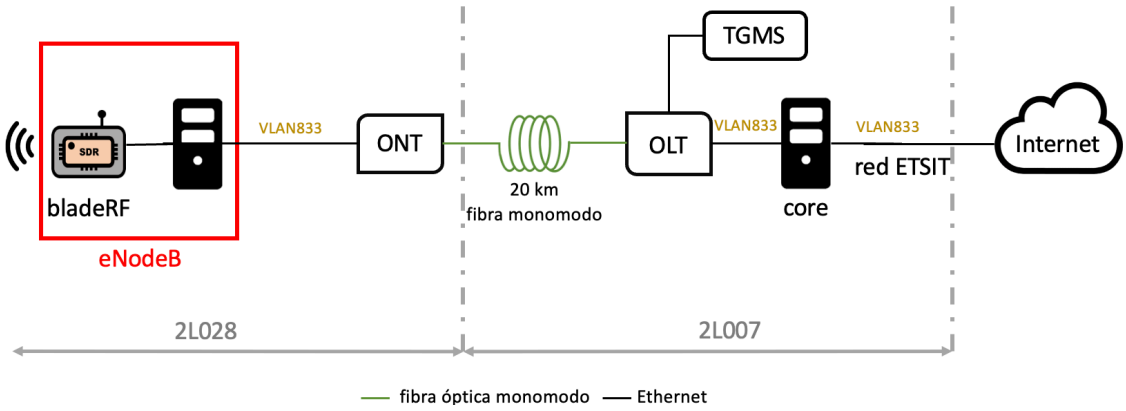


Figura 27.-VLANs configuradas en la red de acceso GPON

El siguiente paso será realizar la configuración de red sobre los nodos *core* y eNB. Comenzando en primer lugar con la configuración del nodo *core*.

En primer lugar, se añade la interfaz pertinente (`eno2`) a la VLAN 833 mediante el comando:

```
$vconfig add eno2 833
```

Tras realizar este paso se tratará la asociación interfaz/VLAN como una nueva interfaz referenciada mediante el nombre `eno2.833`. A esta interfaz se le asociará una dirección IP fija, perteneciente al espacio de direcciones de la red de la escuela:

```
$ip addr add 10.0.103.48/24 dev eno2.833
```

Para levantar la interfaz se utiliza la orden:

```
$ip link set eno2.833 up
```

Finalmente, se configura en la tabla de encaminamiento la interfaz `eno2.833` como ruta por defecto mediante la orden:

```
$ip route add default via 10.0.103.253 dev eno2.833
```

Para comprobar que se ha aplicado la configuración correctamente, se hace uso de la orden `ping` para comprobar con éxito que existe conectividad entre la máquina tanto con diferentes equipos de la red de la universidad, así como con equipos de la red externa.

Para la interfaz del *core* con el OLT los pasos a seguir son equivalentes, siendo las únicas diferencias la interfaz utilizada (`enp3s0`), y la dirección IP fija 192.168.0.1, la primera del espacio de direcciones de la red GPON.

```
$vconfig add enp3s0 833
$ip addr add 192.168.0.1/24 dev enp3s0.833
$ip link set enp3s0.833 up
```

Tras configurar las interfaces del *core*, lo siguiente será configurar el PC como *router*. Primero habilitando el reenvío de paquetes IP modificando el fichero `/proc/sys/net/ipv4/ip_forward` mediante la siguiente orden:

```
$echo 1 > /proc/sys/net/ipv4/ip_forward
```

La regla de encaminamiento que define el reenvío de paquetes de la interfaz `eno2.833` hacia la red externa a través de la interfaz `enp3s0.833` se define utilizando el siguiente comando:

```
$iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eno2.833 -j
```

Con el argumento `-t nat` se indica que la regla se registrará en la tabla NAT, utilizada para aplicar traducción de direcciones IP mediante el protocolo homónimo. Con el argumento `-A POSTROUTING S` se indica que los paquetes se modificarán, después de ser enrutados y justo antes de ser enviados. Con el argumento `-s` se indica sobre qué direcciones IP origen se aplicará la regla. El argumento -o especifica la interfaz por la que se enviarán los paquetes a los que se aplique la regla. Finalmente, el argumento -j MASQUERADE indica que el tráfico de salida estará marcado con una IP origen perteneciente al espacio de direcciones de la interfaz de salida.

Por último, se configurará el servicio DHCP en el nodo *core* para asignar automáticamente una dirección IP a todos los elementos conectados al OLT, tanto ONTs de nivel 2 como de nivel 3. Para ello se empleará el *software* ISC DHCP, instalado mediante el gestor de paquetes *apt*.

Para configurar el servicio DHCP se deberá editar el fichero de configuración `/etc/dhcp/dhcpd.conf`, añadiendo las líneas que se muestra a continuación:

```
subnet 192.168.0.0 netmask 255.255.255.0 {
 range 192.168.0.130 192.168.0.250;
 option domain-name-servers 157.88.129.90;
 option domain-name "RouterTFGLab7-833";
 option routers 192.168.0.1;
 option broadcast-address 192.168.0.255;
 default-lease-time 600;
 max-lease-time 7200;
}
```

En la primera línea se define a que subred se aplicará los parámetros de configuración que se listan entre llaves a continuación. El parámetro `range` configura el rango de direcciones que se podrá ceder a un cliente del servicio DHCP. Mediante los parámetros `option` se añaden diferentes configuraciones, como dirección IP del servidor DNS, nombre de dominio, dirección IP del enrutador y la dirección IP de *broadcast*. Los parámetros `default-lease-time` y `max-lease-time`, definen el tiempo en segundos máximo y por defecto de duración de la concesión de direcciones IP.

También se ha de configurar la interfaz de escucha del servicio DHCP en el fichero `/etc/default/isc-dhcp-server` modificando el fichero para incluir la interfaz deseada.

```
INTERFACESv4="enp3s0.833"
```

Tras modificar los ficheros de configuración del DHCP se invoca el *software* ISC DHCP como servicio, para que se ejecute en segundo plano con permisos de administrador.

Con esto finaliza el proceso de configuración de red del nodo *core*, a continuación, se detalla la configuración de red del nodo eNB, marcadamente más sencilla, pues solo se reduce a habilitar la VLAN 833 sobre la interfaz adecuada, invocar el cliente DHCP para asignar a la interfaz y definir la ruta por defecto del tráfico.

```
vconfig add enp107s0 833
ip link set enp107s0.833 up
dhclient
ip route add default via 192.168.0.1 dev enp107s0.833
```

Se puede hacer diversas comprobaciones de la correcta configuración de la red. Por ejemplo, la consulta de los *logs* del servicio DHCP en el nodo *core* mediante el comando `service status`, de los que se muestra un fragmento a continuación:

```
oct 14 16:59:32 Artemis-Z390-DESIGNARE dhcpd[2806]: DHCPREQUEST for
192.168.0.141 from 1c:69:7a:d7:56:f5 (artemis-NUC12DCMv9) via enp3s0.833
oct 14 16:59:32 Artemis-Z390-DESIGNARE dhcpd[2806]: DHCPACK on
192.168.0.141 to 1c:69:7a:d7:56:f5 (artemis-NUC12DCMv9) via enp3s0.833
```

En los logs del servicio DHCP se muestra la concesión de una dirección IP a través de la interfaz `enp3s0.833`, si se lista las interfaces de red en el nodo eNB se mostrará que la concesión de esa dirección se corresponde con la interfaz de red de dicho nodo.

El correcto encaminamiento del tráfico desde el nodo eNB hacia el exterior de la red GPON se puede comprobar a través de la herramienta `traceroute`, que muestra el tráfico atraviesa la interfaz `enp3s0.833` del nodo *core*, para llegar a la red de la escuela.

```
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  _gateway (192.168.0.1)                0.659 ms  0.671 ms  0.601 ms
 2  10.0.103.254 (10.0.103.254)           0.949 ms  0.881 ms  0.869 ms
 3  157.88.129.250 (157.88.129.250)       1.356 ms  1.685 ms  1.388 ms
 4  157.88.29.211 (157.88.29.211)         1.318 ms  1.305 ms  1.351 ms
 5  157.88.17.51 (157.88.17.51)           1.722 ms  1.660 ms  1.647 ms
 ...
```

Finalizada la configuración de red de ambos nodos, se consigue que las interfaces de red de estos pertenezcan a un espacio de direcciones común, interconectado a través de la red de acceso GPON. Por tanto, el siguiente paso, y el final, es configurar correctamente los *softwares* srsENB y srsEPC para que funcionen de manera distribuida.

La configuración deseada se aplicará a través de los ficheros de configuración `enb.conf` y `epc.conf`, donde se asociará direcciones IP a las diferentes interfaces necesarias para el correcto funcionamiento de los nodos (ver figura Figura 28).

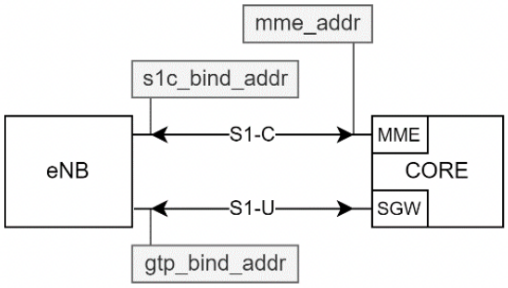


Figura 28.- Interfaces de red configurables en srsENB y srsEPC [3]

Por tanto, en el fichero `enb.conf` se modifica las siguientes líneas incluyendo las direcciones pertinentes:

```
gtp_bind_addr = 192.168.0.141
s1c_bind_addr = 192.168.0.141
```

En el fichero `epc.conf` se edita las siguientes líneas:

```
mme_bind_addr = 192.168.0.1
gtpu_bind_addr   = 192.168.0.1
```

Utilizando esta configuración de los *softwares* srsENB y srsEPC, se ejecuta los *softwares* en sus respectivos nodos con éxito.

4.Resultados

En el capítulo anterior, "Desarrollo de la arquitectura del emulador", tras cada fase desarrollada se fueron mostrando resultados que demostraban la correcta implementación de los objetivos de la fase en cuestión. La sección actual no se centrará en demostrar la funcionalidad de la implementación del servicio, sino en evaluar el rendimiento de la arquitectura de red del servicio, así como de componentes funcionales del servicio.

Se puede medir el rendimiento de una red o servicio con respecto a diversas métricas, pero, dadas las características del servicio emulado, se considera que se debe de centrar los esfuerzos en medir métricas de rendimiento temporal, puesto que resultan críticas para garantizar la seguridad de los usuarios de una potencial aplicación final.

Para garantizar la seguridad de la aplicación se deberá acotar la latencia extremo a extremo máxima aceptable en función a una serie de suposiciones de partida con objetivo de minimizar la distancia de detención del vehículo ante una detección de colisión inminente.

La distancia de detención de un vehículo depende de dos factores, la distancia de reacción y la distancia de frenado. Dada su naturaleza, el tiempo de reacción depende de muchos factores, como la visibilidad, la hora del día, la edad del conductor, etc., por tanto, dar un valor fijo para este tiempo es imposible. Sin embargo, la asociación 5GAA sugiere asumir 1 segundo como tiempo de reacción a la hora de evaluar servicios V2X [54]. En cuanto a la distancia de frenado, de nuevo depende de varios factores, como las condiciones climáticas o la velocidad, para la velocidad, dado el caso de uso de intersección urbana, se asumirá una velocidad máxima de 70 km/h. Para está velocidad la distancia de frenado en condiciones ideales será de 25 metros [55], que junto con los 21,2 metros de distancia recorrida en el tiempo de reacción para esta velocidad, suponen un total de 46,2 metros. El impacto del tiempo de respuesta de un servicio, habitualmente en el rango de las decenas de milisegundos, será bastante reducido en comparación a estos otros factores, no obstante, lo ideal será minimizar este tiempo lo máximo posible. La asociación 5GAA recomienda un tiempo de respuesta máximo de 100 ms para el caso de uso de aviso temprano en intersección [54], tiempo escogido por ser similar al de otros sistemas ADAS.

Además de la latencia extremo a extremo otra métrica de rendimiento temporal relevante para sistemas V2X es el *jitter*. El *jitter* de red es un parámetro de rendimiento habitual en redes TCP/IP que cuantifica la fluctuación de la latencia, por tanto, para el presente caso de uso refleja la fiabilidad de la respuesta temporal del servicio. Por definición, el *jitter* se calcula como la media de la diferencia entre las muestras de valores de latencia RTT, como se muestra en la siguiente ecuación, donde RTT_i son las muestras de latencia y n el tamaño del conjunto de muestras.

$$AvgJitter = \frac{\sum_{i=1}^{n} |RTT_i - RTT_{i-1}|}{n-1}$$

Dada la filosofía de desarrollo por fases seguida para implementar el servicio, al final del desarrollo explicado anteriormente se dispone de mucha flexibilidad en cuanto a la configuración del servicio, puesto que se dispone de todos los ficheros de configuración utilizados previamente. De cara a evaluar métricas de rendimiento de red esto resulta beneficioso, ya que permitirá evaluar el servicio para distintos grados de implementación, y, por tanto, discriminar el impacto de diferentes elementos del servicio de cara a su rendimiento global.

En esta sección se desarrollarán los métodos seguidos para realizar mediciones temporales fiables dentro del servicio, así como los resultados y un análisis. Las métricas analizadas serán tiempo de ida y vuelta (RTT) y *jitter*.

4.1 Medición de latencia RTT

En el servicio a emular, la latencia RTT se corresponderá con la velocidad de respuesta del servicio, puesto que el aviso de una situación de riesgo, mediante mensajes tipo DENM, será generado por el servidor a partir de la información periódica de los mensajes tipo CAM. Para el proyecto actual no se ha reproducido una aplicación centralizada de detección de colisiones a partir de mensajes periódicos, por lo que el retardo de procesamiento debido al algoritmo de detección de situaciones de riesgo de la aplicación será imposible de estimar. Por esta razón el análisis temporal RTT realizado se reducirá a los elementos ya implementados, es decir, arquitectura de servicio y protocolos.

Para medir latencias RTT es común hacer uso de la herramienta `ping`. A través de esta se podrán medir latencias introducidas por la arquitectura del servicio. No obstante, esta herramienta no permite medir retardos de aplicación, como, por ejemplo, los retardos producidos por los protocolos MQTT y TLS.

La solución que se plantea para poder tener en cuenta el impacto de los protocolos en el retardo es hacer uso de la propia aplicación implementada para difundir *timestamps* que serán utilizados como referencia para medir la diferencia con el instante temporal de recepción. Es decir, desde un cliente se publicará en el bróker MQTT los instantes de generación de mensajes, mientras que otro cliente se suscribirá al tópico donde se publiquen estos *timestamps*. El cliente suscrito desencapsulará los mensajes para medir la diferencia de tiempo con el instante de recepción, dando así el retardo total de ida y vuelta teniendo en cuenta todos los elementos del servicio.

Ambos clientes deberán ser ejecutados en el mismo sistema, puesto que se garantiza que el reloj de referencia es el mismo en transmisión y recepción. Para poder implementar este tipo de medición en clientes distribuidos, en primer lugar, se deberá sincronizar los relojes de estos mediante *software* adicional, como, por ejemplo, el protocolo PTP (*Precission Time Protocol*). Las mediciones realizadas para este apartado son llevadas a cabo mediante clientes ejecutados en la misma máquina, puesto que la medición será igualmente válida y realizarla requiere menor complejidad.

Las dos aplicaciones clientes son implementadas en Python con Paho MQTT como *software* cliente MQTT. Para realizar las mediciones se habilitará en el fichero `acl` de Mosquitto un nuevo tópico denominado `aux/time`, donde se publicarán los *timestamps*.

El cliente publicador es bastante sencillo de implementar, consistiendo simplemente en un *script* que provoca un bucle infinito en el cual se encapsula sobre MQTT el tiempo UNIX con una tasa de 10 Hz, para posteriormente publicarlo en el bróker en el tópico `aux/time`.

El *script* de cliente suscriptor, se suscribirá al tópico `aux/time`, del cual tomará las *timestamps* generadas previamente para medir la diferencia entre el tiempo UNIX actual, y el recibido. Tras medir la diferencia temporal, se almacena estos valores en un *array*, a partir del cual se calcularán las latencias máximas, medias y mínimas en tiempo real. El *script* recibirá un total de 10000 mensajes, límite definido en el bucle de recepción. Tras llegar a los 10000 RTTs medidos, se almacena el *array* con los valores de RTT en un fichero con extensión `.mat` (con objetivo de utilizar MATLAB para la visualización de datos), y se finaliza la ejecución. En la Figura 29 se muestra la salida por pantalla del *script*.

```
Latencia del mensaje numero 78: 0.8282661437988281
Latencia media 0.693373191050994
Latencia minima: 0.14662742614746094
Latencia maxima: 0.9458065032958984


Latencia del mensaje numero 79: 0.5536079406738281
Latencia media 0.691604010666473
Latencia minima: 0.14662742614746094
Latencia maxima: 0.9458065032958984


Latencia del mensaje numero 80: 0.4832744598388672
Latencia media 0.6889998912811279
Latencia minima: 0.14662742614746094
Latencia maxima: 0.9458065032958984


Latencia del mensaje numero 81: 0.5991458892822266
Latencia media 0.6878905826144748
Latencia minima: 0.14662742614746094
Latencia maxima: 0.9458065032958984
```

Figura 29.- Salida por pantalla del extremo suscriptor de medición de RTT

Generar 10000 mensajes con una tasa de 10 Hz implica que la medición de latencias se realiza en un periodo de 16 minutos y 40 segundos. Es necesario señalar, que, debido a que los mensajes no se encuentran etiquetados, la medida de latencia asume que los mensajes llegan ordenados al suscriptor, y que no se produce pérdida de mensajes. No obstante, debido al elevado número de mensajes, y a la baja tasa de error de bit, el impacto de estos errores es estadísticamente insignificante.

4.1.1 Medida de latencia RTT en un bróker local

La primera medida de latencia RTT busca cuantificar la influencia del protocolo MQTT, así como de la aplicación Mosquitto en el cómputo global de latencia RTT. Para ello se utilizará los *scripts* cliente ya explicados anteriormente sobre un bróker local. El bróker se lanza con las características básicas detalladas en la subsección 3.3.2, con el matiz de sustituir la interfaz de escucha por la interfaz *loopback* `127.0.0.1` (ver Figura 30).

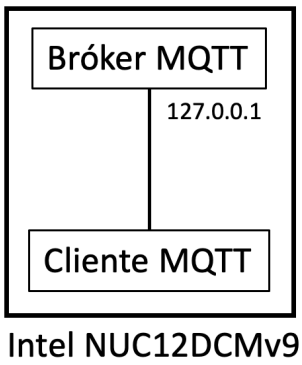


Figura 30.- Configuración de medida con bróker local

Los resultados obtenidos se muestran en la Tabla II y la Figura 31, de estos resultados se puede concluir que la influencia del protocolo MQTT en la latencia del sistema completo será muy limitada,

Tabla II.- Latencias RTT para un bróker MQTT local

Latencia media (ms)	Latencia máxima (ms)	Latencia mínima (ms)	*Jitter* (ms)
0,72	2,19	0,17	0.033

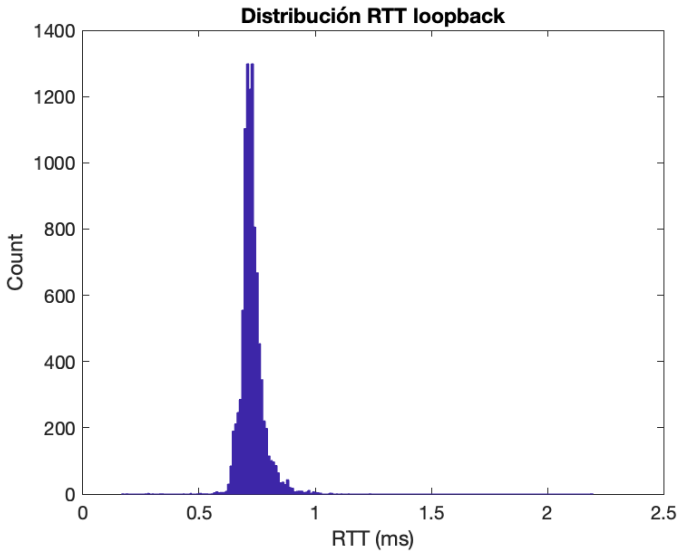


Figura 31.-Distribución de la latencia RTT para un bróker local

4.1.2 Medida de latencia RTT en bróker local que implementa el protocolo TLS

En el subapartado anterior se midió el efecto del protocolo MQTT sobre la latencia, en este apartado se busca medir la influencia del protocolo de encriptación TLS. Para ello se repetirá el planteamiento del subapartado anterior, es decir, utilizar un bróker que escuche en la interfaz de *loopback*, pero en este caso haciendo uso del protocolo TLS con certificados cliente.

Los resultados se muestran en la Tabla III y la Figura 32. De estos resultados se puede concluir que, aunque la influencia del protocolo TLS es perceptible, la diferencia de tan solo 0.27 ms respecto al caso sin encriptado TLS es mínima y, por tanto, un factor prácticamente despreciable en el cómputo de latencia del servicio completo.

Tabla III.- Latencias RTT para un bróker MQTT local con protocolo TLS

Latencia media (ms)	Latencia máxima (ms)	Latencia mínima (ms)	*Jitter* (ms)
0,89	2,83	0,21	0.049

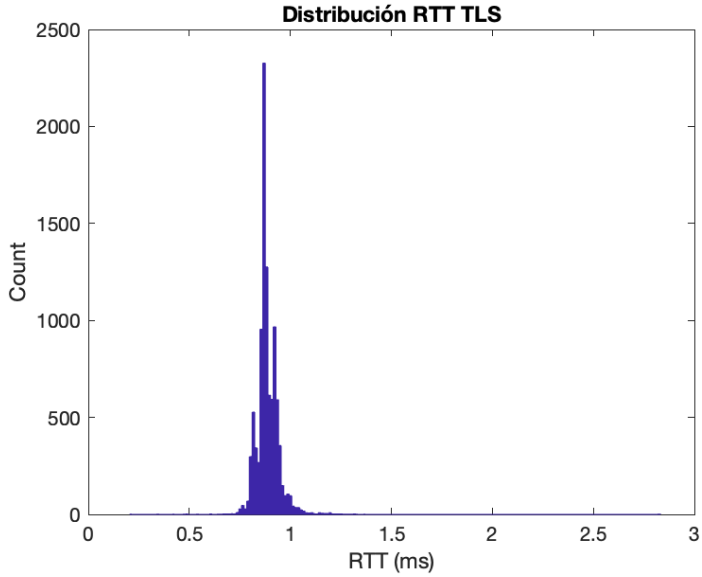


Figura 32.- Distribución de la latencia RTT para un bróker local con protocolo TLS

4.1.3 Medida de latencia RTT en un bróker en LAN

En este apartado se realizará las mediciones de latencia sobre un bróker ubicado en la misma red de área local. Para ello se conecta las interfaces de red ethernet de dos PCs Intel NUC12DCMv9 con un cable de red.

Para que esta configuración sea funcional se ha de realizar una configuración de red mínima, consistente en asociar a la interfaz de red de cada PC una dirección IP fija perteneciente al mismo espacio de direcciones local, y la ruta por defecto por dicha interfaz en el PC que actuará como cliente (de manera análoga a como se vio en la subsección 3.3.6). En la Figura 33 se muestra la configuración utilizada, así como las direcciones IP asignadas a las interfaces de esta.

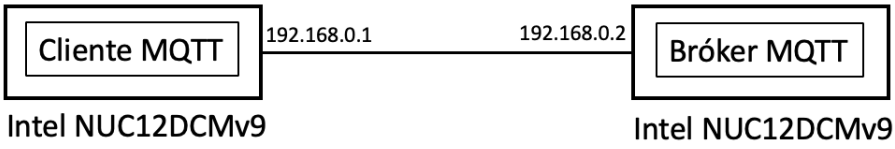


Figura 33.- Configuración de medida para un bróker en LAN

El objetivo de este apartado será cuantificar el impacto tanto de un bróker remoto, como del medio físico ethernet en la latencia del sistema total. Esta prueba permitirá contrastar estos resultados con los resultados de acceso al medio mediante LTE (en la siguiente subsección). Es importante resaltar que en este caso no se ha utilizado el protocolo TLS para realizar las mediciones.

Los resultados se muestran en la Tabla IV y en la Figura 34. Como se puede ver el valor de latencia media ha aumentado, aunque no de manera especialmente reseñable. Sí que se considera relevante el aumento del *jitter*, siendo además el caso de que no exista otro tráfico por el medio físico además del tráfico MQTT. Se considera interesante que en la Figura 34 se observa la aparición de dos picos en la distribución de valores de RTT, pero, se desconoce el origen de estos.

Tabla IV.- Latencias RTT para un bróker MQTT en LAN

Latencia media (ms)	Latencia máxima (ms)	Latencia mínima (ms)	*Jitter* (ms)
1,36	1,87	0,7	0.1284

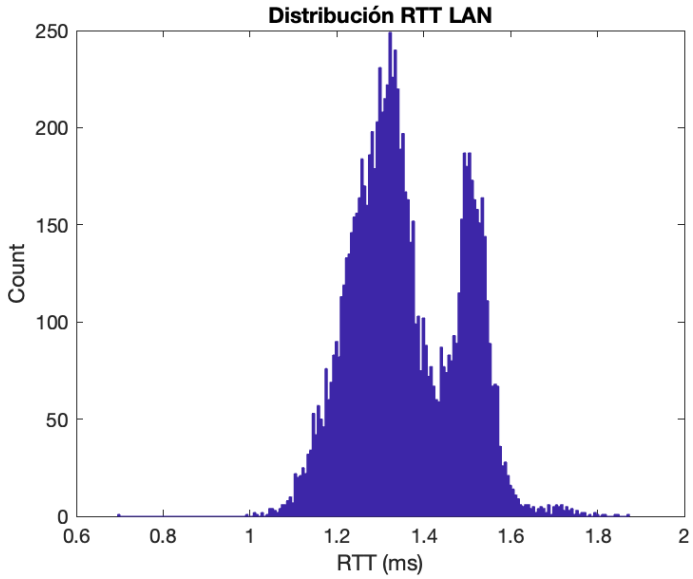


Figura 34.- Distribución de la latencia RTT para un bróker en LAN

En la Figura 35 se muestra en una gráfica de cajas los resultados obtenidos hasta el momento.

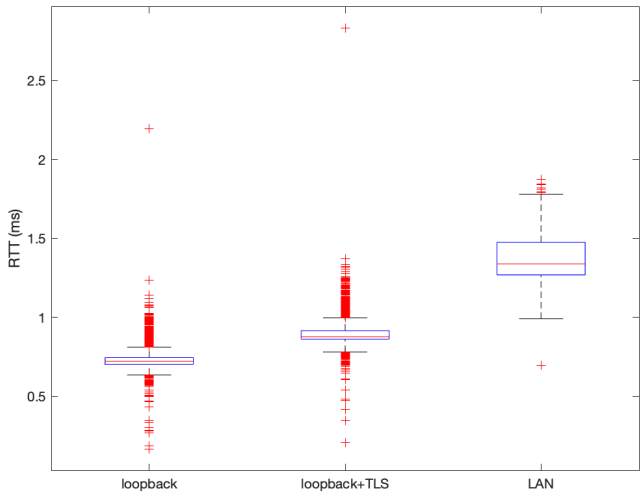


Figura 35.- Gráfica de cajas de los primeros resultados de medición de latencia RTT

4.1.4 Medida de latencia RTT para un bróker accedido mediante LTE

En este subapartado se busca cuantificar el impacto del acceso LTE hasta un bróker MQTT. Para ello se implementa la arquitectura mostrada en la Figura 36Figura 15, pero implementando en este caso los clientes publicador y suscriptor en el mismo nodo físico.

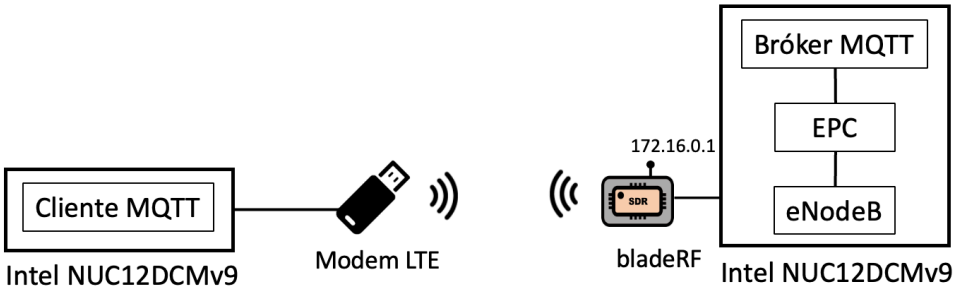


Figura 36.- Configuración de medida para un bróker accedido mediante LTE

Los resultados se muestran en la Tabla I y la Figura 37. Las diferencias con las distribuciones de latencias previas son acusadas, siendo la media y el *jitter* considerablemente más elevadas que para el resto de situaciones. En este caso, además, la distribución es uniforme de 20 a 40 ms, frente a las distribuciones de tipo gaussiano con una media clara y una distribución decreciente de probabilidad alrededor.

Tabla V.- Latencias RTT para un bróker MQTT con acceso LTE

Latencia media (ms)	Latencia máxima (ms)	Latencia mínima (ms)	*Jitter* (ms)
30,95	12,92	51,07	1,3342

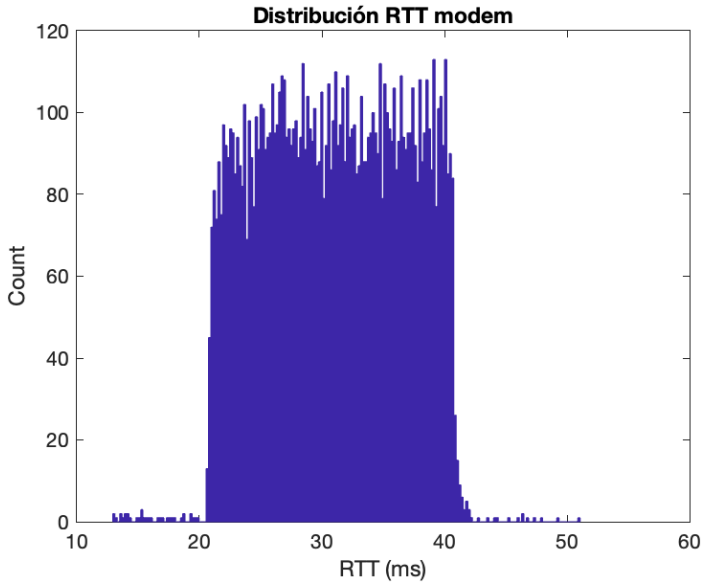


Figura 37.- Distribución de la latencia RTT para un bróker con acceso LTE

Podemos comparar el impacto de introducir otra tecnologías de acceso inalámbrico cómo Wi-Fi, en lugar de LTE. El Wi-Fi introduce un retardo mucho menor, dado que casi no tiene que realizar ningún procesamiento con los mensajes. El hecho de que este procesamiento sea menor afecta también a la forma de la distribución que sigue pareciéndose a una guassiana. (ver Figura 38).

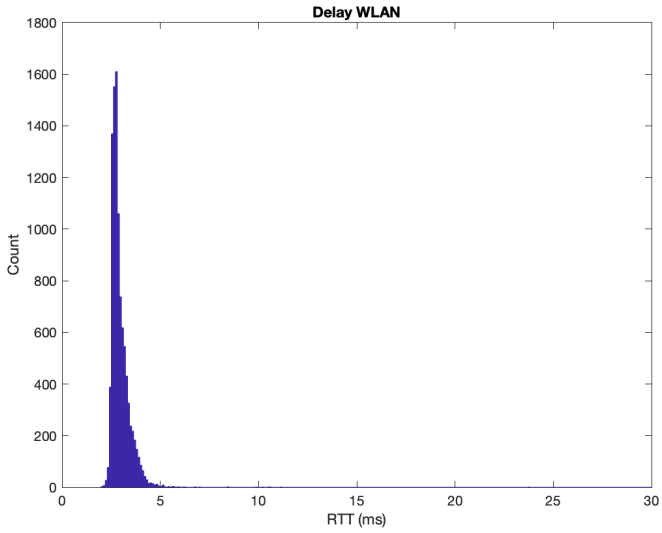


Figura 38.- Distribución de la latencia RTT para un bróker con acceso Wi-Fi

La distribución uniforme de los retardos es muy rara en comunicaciones. Lo normal es tener una distribución gaussiana dado que se espera que todos los mensajes sean tratados por igual, con pequeñas variaciones alrededor de la normal. Dado que no sale dicha distribución, se representó temporalmente los retardos de cada mensaje enviado a ver si mostraba algún patrón reconocible. De esta manera, si se muestra la serie temporal de muestras de RTT para acceso LTE se puede ver un claro patrón de señal diente de sierra (ver Figura 39), esto es, parece que los mensajes tienen un comportamiento más bien determinístico, empezando con un retardo de unos 40 ms para ir disminuyendo hasta un mínimo de 22 ms en 3 segundos, es decir, a lo largo de 30 mensajes enviados. En la Figura 40 se muestra un análisis frecuencial de la serie de

muestras de latencia RTT, donde se puede observar la frecuencia fundamental de la señal, de 0,28 Hz (periodo de 3,6 segundos), junto con los armónicos característicos de una señal de diente de sierra [56].

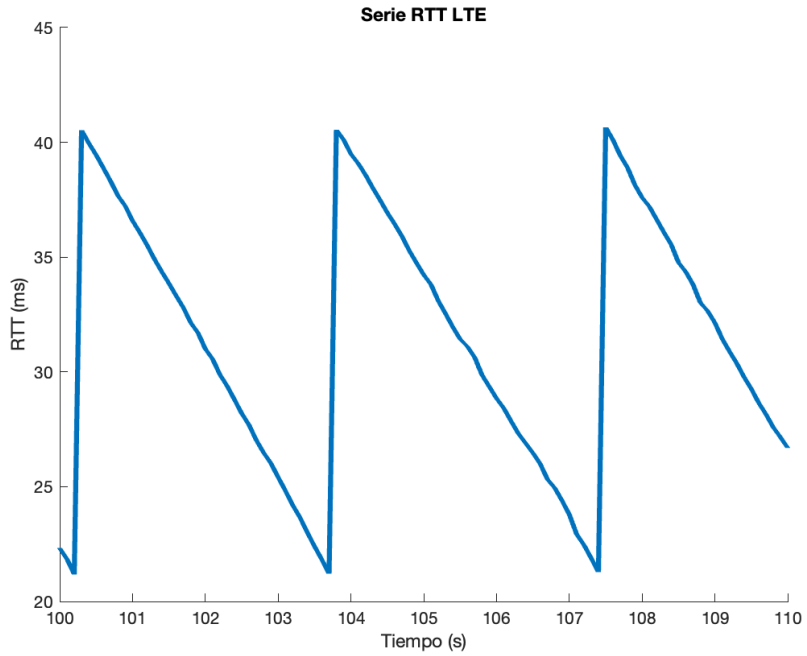


Figura 39.-Detalle de la serie temporal de muestras de RTT para LTE

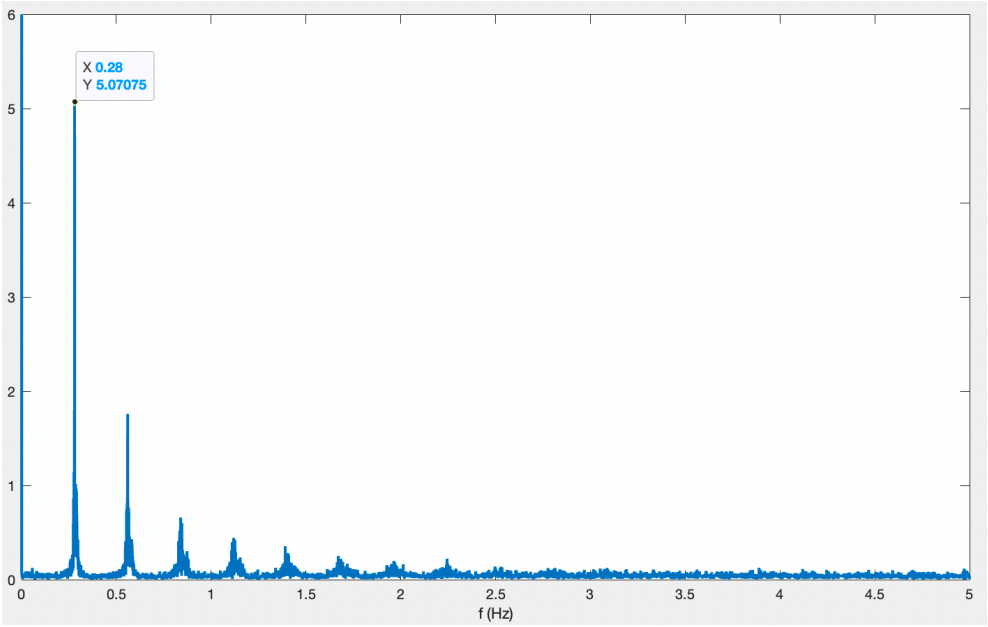


Figura 40.- Análisis frecuencial de las muestras de latencia RTT para acceso LTE

La variación de este patrón en las muestras es muy acusada, de 20 ms entre los máximos y los mínimos, lo que supone más de la mitad de la latencia media para esta configuración. Además, el análisis frecuencial demuestra que este patrón de comportamiento es extremadamente consistente y definido. A partir de este punto todas las configuraciones con las que se harán medidas incluyen el acceso LTE, y por tanto arrastrarán el patrón de latencia al resto de medidas. Por ello se considera

importante realizar más pruebas para esta configuración, con objetivo de descartar posibles fuentes de este comportamiento, para así poder mitigarlo.

En primer lugar, se plantea que el *hardware* SDR utilizado pueda ser la fuente del patrón observado en la latencia. Para ello se repite la prueba con la misma configuración, pero utilizando el *hardware* USRP NI-B2901 en lugar del *BladeRF 2.0 micro xA9*. Para ello será necesario instalar los *drivers* del dispositivo mediante los comandos:

```
$sudo add-apt-repository ppa:ettusresearch/uhd
$sudo apt-get update
$sudo apt-get install -y libuhd-dev libuhd3.15.0 uhd-host
$sudo uhd_images_downloader
```

Tras esto, será necesario recompilar el *software* srsRAN para que este funcione con los nuevos *drivers* instalados. El resto del procedimiento es el habitual.

Tras repetir las medidas con esta nueva configuración se obtienen los resultados mostrados, que como se puede ver en la Figura 41, presentan el mismo patrón de señal de diente de sierra con la misma frecuencia. Se considera reseñable apuntar que el *hardware* USRP muestra picos en la latencia como los que se ven Figura 41 consistentes a lo largo de todo el periodo de medición.

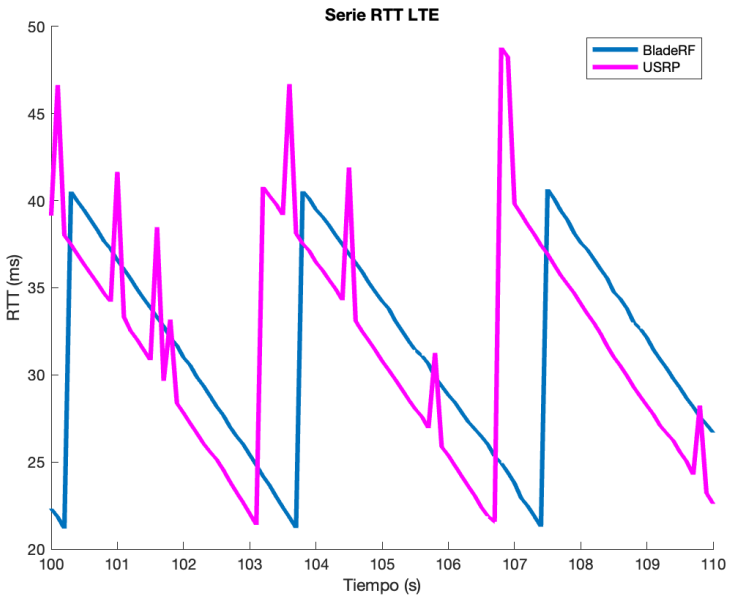


Figura 41.- Serie RTT comparativa entre diferentes hardware SDR

Descartado el *hardware* SDR como fuente del patrón de latencia se plantea que el sistema operativo y la asignación dinámica de recursos al *software* srsRAN pueda producir el patrón. Para descartar esta posibilidad se ejecutará el *software* con máxima prioridad para el sistema operativo. Para ello será necesario instalar la versión del *kernel* de Linux de baja latencia, la cual permite modificar la prioridad de los procesos del SO. Para ello se instala esta versión del *kernel* con la orden:

```
$sudo apt-get install -y linux-lowlatency
```

Posteriormente se reinicia el PC, y desde el gestor de arranque GRUB se selecciona Ubuntu con *kernel* de baja latencia. Posteriormente se lanza los procesos de srsRAN, y se modifica la prioridad de estos con los comandos:

```
$sudo chrt -r -p 99 $(pidof srsenb)
$sudo chrt -r -p 99 $(pidof srsenb)
```

Los resultados de las mediciones en estas condiciones se muestran en la Figura 42, que como se puede ver vuelve a presentar el mismo patrón de señal de diente de sierra, por lo que se descarta la gestión de recursos del SO como fuente del patrón.

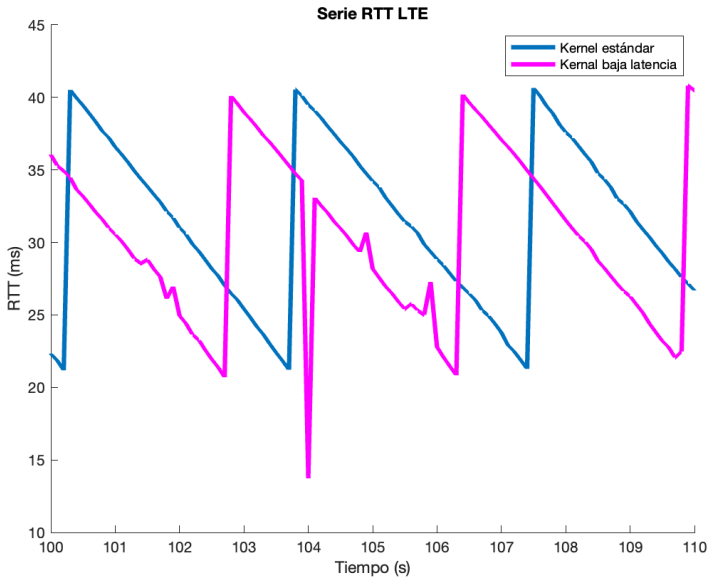


Figura 42.- Serie RTT comparativa entre procesos de srsRAN con diferente prioridad

Finalmente se busca descartar que el módem Huawei E3372 sea la fuente del patrón de latencia. Para ello se utilizará la tarjeta SIM programable en un dispositivo móvil Android, con objetivo de que este actúe como UE en la red LTE, y que este aloje los clientes MQTT para realizar las mediciones de latencia.

Para ejecutar los clientes se instala en el dispositivo móvil la aplicación de emulación de terminal *Termux*, que pone a disposición del usuario un terminal UNIX además de un *software* de instalación de paquetes sencillo. Mediante esta aplicación se instala Python en el dispositivo, además del cliente Paho-MQTT. No todos los paquetes de Python utilizados en los *scripts* de medición son compatibles con la arquitectura ARM de la CPU del dispositivo móvil, por lo que se hace algunas modificaciones a estos, manteniendo el principio de funcionamiento desarrollado anteriormente.

En la Figura 43 se muestra la serie temporal de resultados de las mediciones RTT realizadas con el dispositivo móvil. Resulta inmediato ver las diferencias entre este *hardware* UE y el módem, si bien, se repite el patrón de señal de diente de sierra, la frecuencia de la variación de la latencia RTT es varias veces superior en el caso del dispositivo Android. No obstante, aunque la frecuencia del patrón varíe, el *jitter* sigue siendo extremadamente elevado como se puede ver en la superposición de los histogramas de las mediciones realizadas con el módem y el dispositivo Android (ver Figura 44).

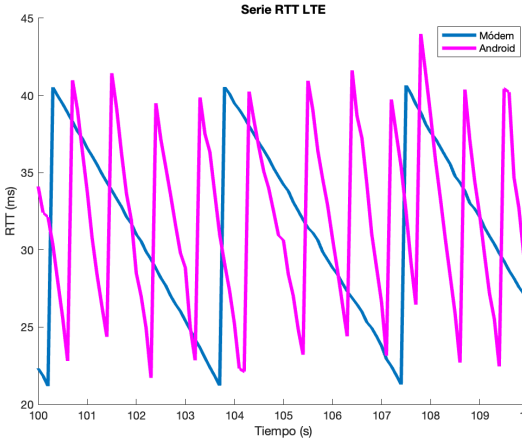


Figura 43.-Serie RTT comparativa entre el uso de Modem y dispositivo Android

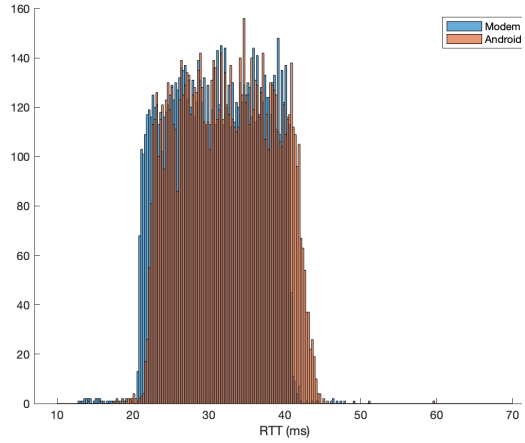


Figura 44.- Distribución de la latencia RTT para el uso de Modem y dispositivo Android

De todos los experimentos realizados para esta subsección se puede extraer varias conclusiones. La primera es que el patrón de señal de diente de sierra dado en la latencia no surge ni por el *hardware* SDR escogido, ni por la asignación de recursos del SO Linux. Por otra parte, diferentes dispositivos de acceso a la RAN también presentan una latencia con características similares, aunque no totalmente equivalentes puesto que la frecuencia del patrón sí varía para los diferentes dispositivos. Estas evidencias parecen apuntar al *software* o a los protocolos implicados como causantes de las grandes variaciones en la latencia del acceso por RAN.

Observando las muestras de la serie de muestras con más detenimiento, se puede observar, que, para el caso del módem, existe un gran número de muestras en el flanco de bajada de la señal, y ninguna en el flanco de subida. Esto indica que el cambio que provoca este comportamiento es de efecto inmediato, lo cual sería consistente con la liberación de un buffer o ventana de aplicación. Esta observación, junto al elevado periodo del patrón, podría apuntar a que la fuente de este radique en el *software* srsRANy no al propio protocolo LTE.

Otra opción posible para explicar el comportamiento periódico de la latencia, es que sea efecto de la liberación de la ventana de transmisión del protocolo TCP, ventana que es determinada de manera dinámica en función de la memoria disponible del dispositivo que establece la conexión. Esta podría ser una explicación para la diferencia de periodos para diferentes dispositivos mostrada en la Figura 43, donde el dispositivo Android refleja un periodo menor, comportamiento que podría estar relacionado con la menor disponibilidad de memoria frente a un PC de sobremesa. Esta hipótesis se podría descartar repitiendo las pruebas utilizando UDP como protocolo de transporte, no obstante, por limitaciones de tiempo no llegaron a efectuar.

Debido a que ha resultado imposible mitigar el problema de la elevada variabilidad de la latencia, es evidente asumir que a partir de este punto las mediciones de latencia RTT, para arquitecturas que implementan el acceso LTE, arrastrarán las consecuencias de este de cara a la latencia. A partir de este punto el resto de las mediciones se realizarán con el módem Huawei.

4.1.4 Medida de latencia RTT para un bróker alojado en la nube de Microsoft Azure

En este subapartado se busca cuantificar el impacto del encaminamiento a la máquina virtual de Azure en el cómputo global del retardo del servicio. Para ello se realiza las mediciones de latencia RTT sobre la arquitectura que se muestra en la Figura 45.

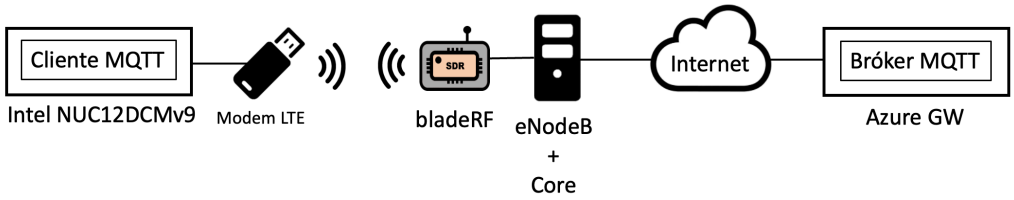


Figura 45.- Configuración de medida para el bróker alojado en Azure

En la Tabla VI y la Figura 46 se muestran los resultados de las mediciones de latencia. En la Figura 47 se muestra la serie temporal de muestras RTT, donde se ve reflejado el patrón de diente de sierra introducido por la componente RAN de la arquitectura. Como se puede ver tanto la latencia como el *jitter* aumentan de manera notable, esto es razonable puesto que, al introducir un elemento ubicado en Internet, el número de elementos lógicos del servicio aumenta de manera impredecible.

Tabla VI.- Latencias RTT para un bróker MQTT alojado en Azure

Latencia media (ms)	Latencia máxima (ms)	Latencia mínima (ms)	*Jitter* (ms)
60,27	546,74	40,87	4,1110

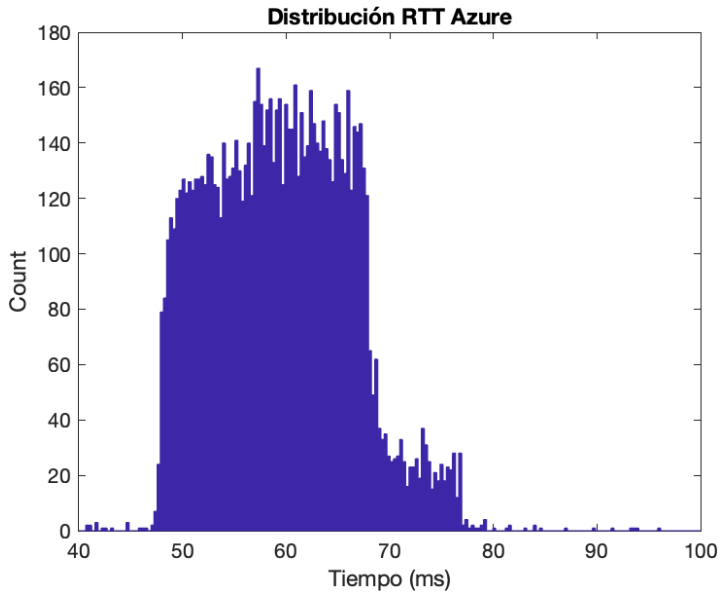


Figura 46.- Distribución de la latencia RTT para un bróker MQTT alojado en Azure

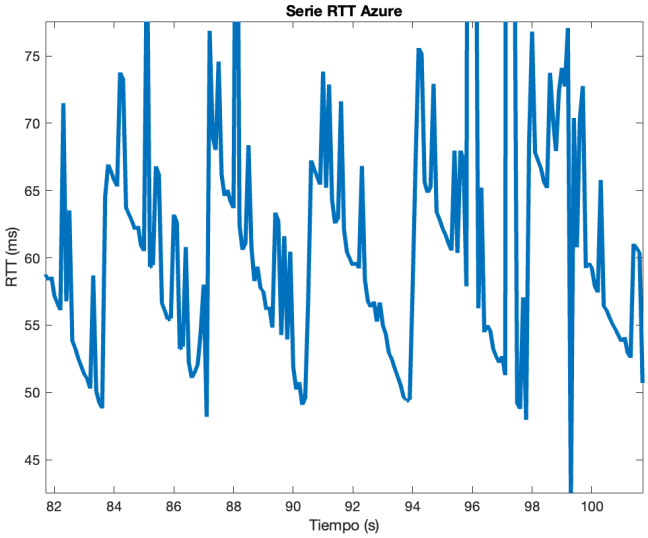


Figura 47.- Detalle de la serie temporal de muestras de RTT para un bróker MQTT alojado en Azure

4.1.5 Medida de latencia RTT para la arquitectura completa de servicio

Finalmente, en este subapartado se desarrollará tanto las técnicas para realizar mediciones de latencia RTT en el sistema completo, como los resultados obtenidos al aplicar dichas técnicas. Para la presente subsección se realizará las mediciones sobre la arquitectura de servicio completa con y sin el acceso mediante una red GPON implementada (ver Figura 48).

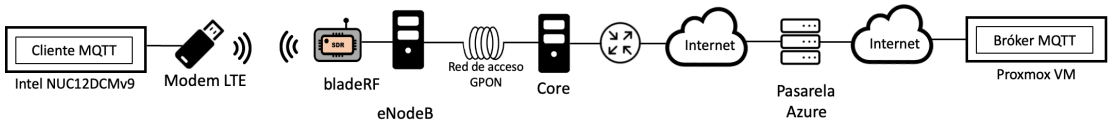


Figura 48.- Configuración de medida para la arquitectura de servicio completa

Para realizar las mediciones de latencia RTT sobre la arquitectura de servicio completo se deberá modificar el planteamiento aplicado en los *scripts* utilizados hasta este punto. Esto se debe a la presencia del *bridge* MQTT, y a que la medida que se busca realizar requiere que la ruta de ida y vuelta del tráfico del servicio sea el mismo. Con los *scripts* que se ha venido usando hasta el momento se podrá publicar sobre el bróker *bridge*, que a su vez reenvía el mensaje al bróker principal, al cual se podrá conectar el extremo suscriptor. El problema radica en que, si se sigue este planteamiento, el camino de vuelta del tráfico no será a través del bróker *bridge*, sino a través de un camino considerablemente más corto pues el bróker principal está alojado en la infraestructura de la ETSIT.

Para garantizar que el tráfico inyectado para medir la latencia sigue el camino deseado se habilita un nuevo tópico en el bróker *bridge*, denominado `aux/time_echo`. Este nuevo tópico será utilizado para que el bróker principal reenvíe los *timestamps* al *bridge* en calidad de publicador de este. Para implementar este funcionamiento se crea un sencillo *script* en el bróker principal cuyo objetivo es suscribirse al tópico `aux/time` local (el contenido del cual proviene del *bridge*), y publicar los mensajes recibidos en el tópico `aux/time_echo` del bróker bridge. En la Figura 49 se muestra el flujo descrito.

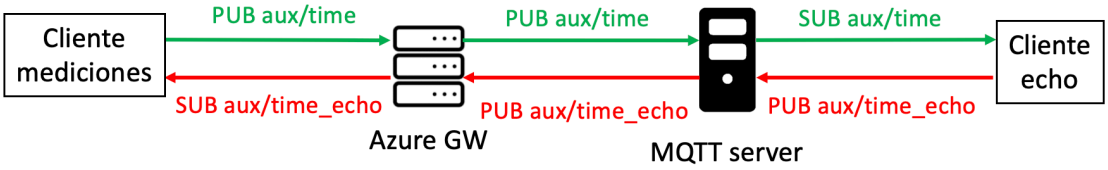


Figura 49.- Flujo de tráfico MQTT para realizar mediciones de latencia RTT sobre el sistema completo

Para configurar adecuadamente el comportamiento del *bridge* se añaden las siguientes líneas al fichero de configuración de Mosquitto:

```
topic aux/time out
topic aux/time_echo in
```

En primer lugar, se realizan las mediciones con la arquitectura completa sin la red de acceso GPON. Para ello se lanza todos los servicios pertinentes, además del *script* cliente ubicado en el servidor MQTT principal. En el *script* del extremo publicador se reduce la tasa de medición a 200 ms, puesto que la tasa utilizada hasta este momento será comparable a la latencia RTT, y, en consecuencia, las medidas serían erróneas.

En la Figura 50 y la Tabla VII se muestran los resultados para las medidas sin acceso GPON, por otra parte, en la Figura 51 y la Tabla VIII se muestran los resultados implementando dicha tecnología. Es inmediato ver que la latencia aumenta de forma notable. Esto es esperable puesto que el tráfico deberá ser enrutado un total de 4 veces entre la infraestructura de red de la escuela e internet. Estos resultados se encuentran alineados con los resultados previos, puesto que frente a la subsección anterior el retardo medio ha aumentado en 30 ms, frente al aumento de 30 ms de diferencia que se encontraba en esta frente al anterior supuesto. En ambos casos el retardo medio añadido es el mismo puesto que en ambos casos la fuente de este es el retardo de encaminamiento del tráfico de ida y vuelta entre la VM de Azure y la red de la escuela.

En cuanto al retardo producido por el acceso GPON a la red se observa que es muy pequeño frente al retardo global, tan solo de 1,2 ms de media. Esto es razonable, puesto que no se producirán reenvíos en este segmento de la arquitectura, ya que todos los dispositivos que la conforman son de nivel 2. A pesar de disponer de 20 km de longitud de fibra, el impacto del retardo de propagación será muy reducido, de tan solo 0,1 ms.

En la Figura 52 se muestra un detalle de la serie temporal de muestras, donde se observa de nuevo el patrón de diente de sierra, notablemente más afectado de ruido que para casos anteriores, debido a la mayor cantidad de reenvíos requeridos para que el paquete complete el flujo esperado.

Tabla VII.- Latencias RTT para un bróker MQTT sobre la arquitectura completa sin acceso GPON

Latencia media (ms)	Latencia máxima (ms)	Latencia mínima (ms)	*Jitter* (ms)
90,34	438,22	66,74	6,8210

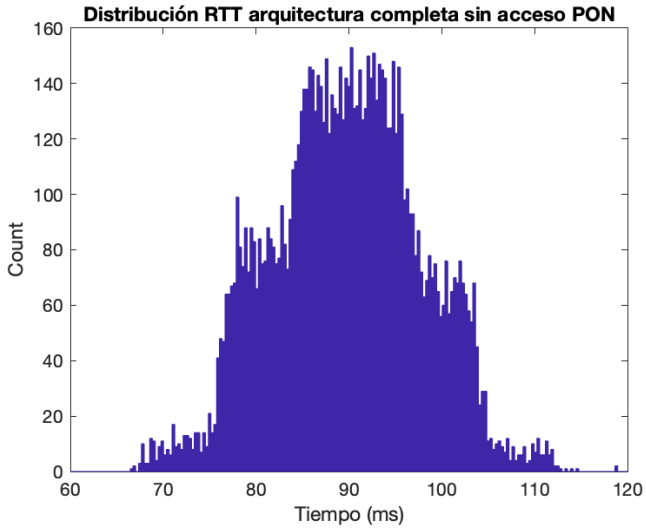


Figura 50.- Distribución de la latencia RTT sobre la arquitectura completa sin acceso GPON

Tabla VIII.- Distribución de la latencia RTT sobre la arquitectura completa con acceso GPON

Latencia media (ms)	Latencia máxima (ms)	Latencia mínima (ms)	*Jitter* (ms)
91,41	423,44	68,95	6,5263

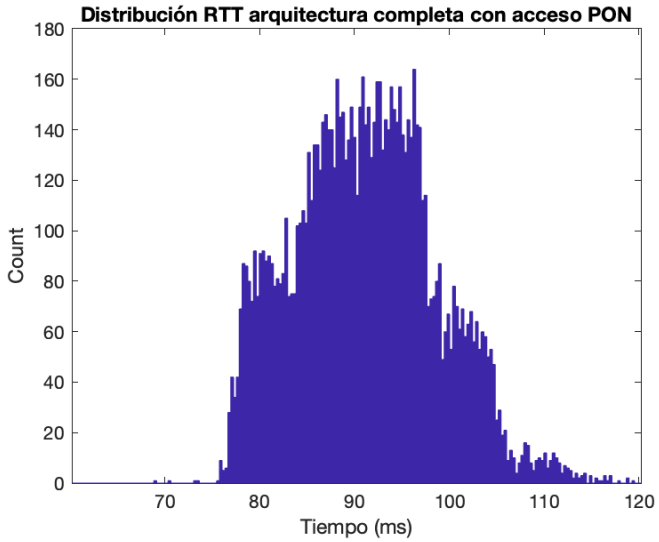


Figura 51.-Distribución de la latencia RTT sobre la arquitectura completa con acceso GPON

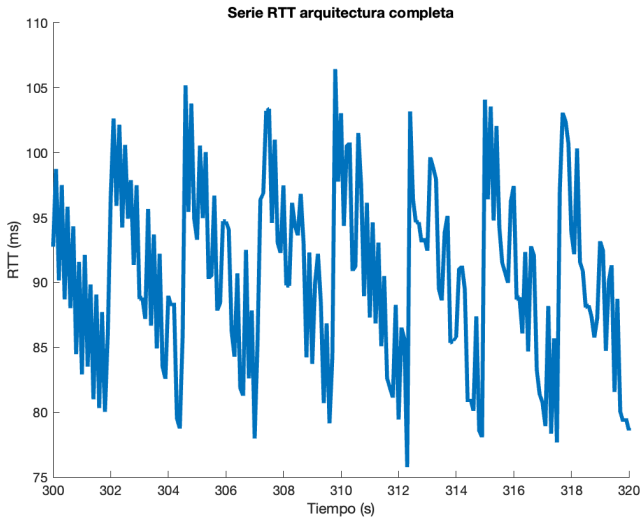


Figura 52.- Detalle de la serie temporal de muestras de RTT para la arquitectura completa

Por último, en la Figura 54, se representan en un gráfico de cajas los resultados obtenidos en las últimas dos secciones.

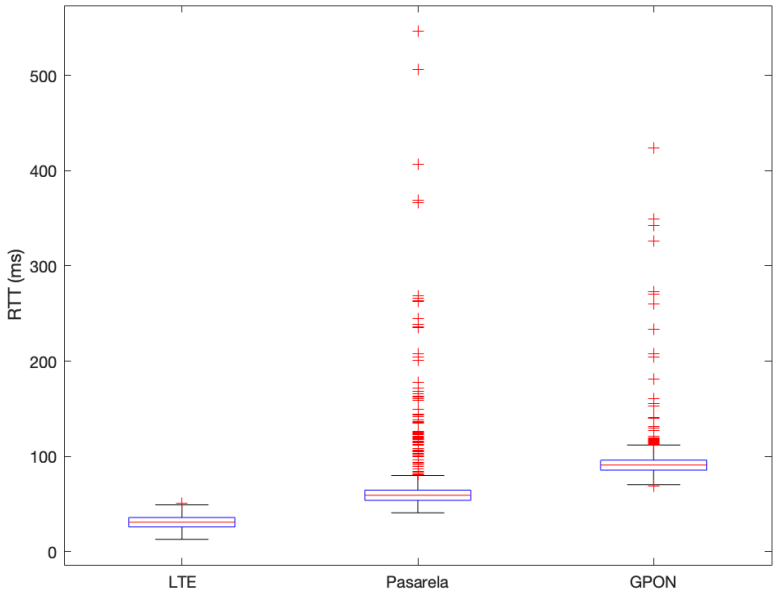


Figura 53.- Gráfica de cajas de algunos de los resultados de medición de latencia RTT

4.1.6 Resumen de resultados

Para cerrar la sección de resultados se ha recopilado los resultados en la Tabla IX donde se muestran la contribución de cada componente al *jitter* y a la latencia promedio globales de la arquitectura de servicio. La contribución se midió como la diferencia entre los valores medios obtenidos para cada una de las subsecciones, con la referencia pertinente. En la Figura 54 se representa de manera gráfica sobre el diagrama de arquitectura de servicio los retardos más relevantes.

Tabla IX.- Contribución al cómputo de latencia global de cada elemento del servicio

Componente	Contribución a la latencia media (ms)	Contribución al jitter (ms)
Encapsulado/desencapsulado MQTT	0,72	0,033
Encriptado TLS	0,16	0,016
Encapsulado/desencapsulado ethernet (1 salto)	0,63	0,095
Acceso LTE	30,22	1,3
1 RTT Cliente - Azure GW	29,32	2,77
1 RTT GW – Servidor principal	30,08	2,71
Acceso PON	1,07	-0,3

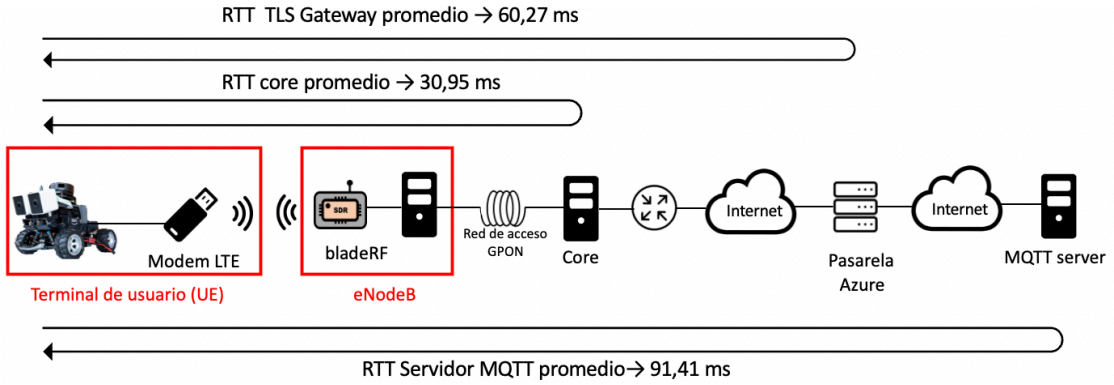


Figura 54.- Retardos promedio a diferentes componentes del servicio

5. Conclusiones y líneas futuras

El objetivo principal de este Trabajo de Fin de Máster era la implementación de una arquitectura de servicio V2I realista, que permitiera realizar pruebas extrapolables a una posible implementación comercial. Los resultados obtenidos demuestran que la maqueta desarrollada es adecuada para realizar pruebas de utilidad para fabricantes de automóviles como es el caso de grupo Renault España, con el que se ha contado para realizar las pruebas de este TFM.

Además de este objetivo principal se considera relevante la consecución de otros objetivos secundarios. El primero fue establecer una relación de colaboración con el grupo Renault España, que no solo se limitó al trabajo aquí descrito, sino que además incluyó la elaboración de revisiones de estándares, revisiones bibliográficas y la descripción de casos de uso.

En este proyecto también se consiguió realizar mediciones de la latencia extremo a extremo que tendría una potencial aplicación implementada en la arquitectura diseñada. El análisis de latencia realizado también tiene en cuenta la influencia de diferentes componentes de la arquitectura, permitiendo discernir cuáles tienen mayor influencia en el rendimiento del sistema, y en consecuencia determinar en qué aspectos del servicio se deberían centrar esfuerzos en caso de futura necesidad de optimización.

La arquitectura de servicio implementada al completo mostró una latencia RTT media de 91,41 milisegundos cuando se incluían todos los elementos previstos. En este tiempo el vehículo tan solo recorrería 1,77 metros hasta obtener la respuesta del servidor, suponiendo una velocidad de 70 km/h. Teniendo en cuenta el tiempo de respuesta del servicio y la distancia de detención desde la emisión del mensaje que provoca la detección de una situación de riesgo, la distancia total de detención será de 48,1 metros, distancia no desdeñable, pero que viene mayormente dada por factores no controlables, por lo que se considera el resultado de latencia como aceptable, ya que además se encuentra acotado dentro de los márgenes dados por la asociación 5GAA para servicios para este caso de uso concreto.

Un resultado reseñable obtenido del análisis de latencia es la elevada variabilidad introducida por un elemento desconocido, aunque como principales sospechosos se encuentran el software LTE y el protocolo TCP. Esta variación es de alrededor de 20 ms y se produce con un patrón predecible. Una línea de trabajo futura evidente es localizar la fuente exacta de esta variabilidad y eliminarla, dado que reduciría los tiempos máximos de retardo en 20 ms. La media no se vería tan afectada, reduciéndose en solo 9,3 ms.

Una clara limitación de este Trabajo de Fin de Máster es no haber llegado a implementar la aplicación de detección de colisiones. Esta aplicación podría ser testeada haciendo uso de la maqueta de cruce del proyecto ARTEMIS [39] en conjunto con el trabajo desarrollado para la localización *in-door* de los vehículos para otro Trabajo de Fin de Máster desarrollado en paralelo a este [57]. Los datos de posicionamiento de los vehículos dentro de la maqueta generados en tiempo real podrán ser encapsulados en un mensaje CAM y hacer uso de la infraestructura de red aquí desarrollada para emular un servicio V2X con un elevado grado de fidelidad, siendo este otro claro potencial desarrollo futuro en la línea de este proyecto.

La tendencia futura de las tecnologías C-V2X será hacia la implementación mediante acceso 5G, por tanto, migrar el servicio hacia esta tecnología es una clara opción para seguir construyendo sobre lo aquí desarrollado. La migración del servicio a esta tecnología además debería ser relativamente sencilla, puesto que tanto srsRAN [58], como otros desarrolladores de servicios de emulación de tecnología celular, disponen de *software* de emulación de redes 5G compatible con los dispositivos SDR utilizados en este proyecto.

Finalmente, otra posible línea futura sería modificar la arquitectura para adaptarla al paradigma de *edge-computing*, ya que permitiría disminuir dramáticamente la latencia del servicio.

Bibliografía

[1] L. moncloa, «Los accidentes de tráfico se cobraron la vida de 1.004 personas el pasado año,» 7 1 2022. [En línea]. Available: https://www.lamoncloa.gob.es/serviciosdeprensa/notasprensa/interior/Paginas/2022/070122-balance-siniestralidad-2021.aspx.

[2] C. Europea, «Estadísticas de 2019 sobre seguridad vial: ¿qué esconden las cifras?,» 11 6 2020. [En línea]. Available: https://ec.europa.eu/commission/presscorner/detail/es/qanda_20_1004.

[3] I. Royuela, «Trabajo de Fin de Máster: Diseño e implementación de un testbed de Edge computing para el soporte de vehículos conectados,» 2022.

[4] I. N. Cuadrado, «Trabajo de fin de grado: Arquitecturas LTE-5G mediante SDR y OpenAirInterface,» 2022.

[5] ETSI, «Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service,» 2019.

[6] ETSI, «ITS), Intelligent Transport Systems; Communications, Vehicular; Applications, Basic Set of; Service, Part 3: Specifications of Decentralized Environmental Notification Basic,» 9 2014. [En línea]. Available: https://www.etsi.org/deliver/etsi_en/302600_302699/30263703/01.02.01_30/en_30263703v010201v.pdf.

[7] J. L. S. Corrales, Trabajo de Fin de Máster: Implementación del protocolo GeoNetworking en un software de comunicación V2X.

[8] D. M. González, Trabajo de Fin de Grado: Implementación del Protocolo de Transporte Básico en un Software de Comunicación V2X.

[9] A. Festag, «Standards for vehicular communication—from IEEE 802.11p to 5G,» *Elektrotech. Inftech,* vol. 132, p. 409–416, 2015.

[10] EUR-Lex, «COMMUNICATION FROM THE COMMISSION TO THE EUROPEAN PARLIAMENT, THE COUNCIL, THE EUROPEAN ECONOMIC AND SOCIAL COMMITTEE AND THE COMMITTEE OF THE REGIONS A European strategy on Cooperative Intelligent Transport Systems, a milestone towards cooperative, conne,» 2016. [En línea]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52016DC0766.

[11] S. o. t. D. o. C.-I. i. E. F. Report, «European Comission: Mobility and transport,» 2016. [En línea]. Available: https://transport.ec.europa.eu/system/files/2016-10/2016-c-its-deployment-study-final-report.pdf.

[12] 3GPP, «Specification 36.785,» 2016. [En línea]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3063.

[13] V. B. S. S. a. E. P. V. Mannoni, «A Comparison of the V2X Communication Systems: ITS-G5 and C-V2X,» de *2019 IEEE 89th Vehicular Technology Conference*, 2019.

[14] D. W. A. W. a. H. D. S. R. Sattiraju, «Link Level Performance Comparison of C-V2X and ITS-G5 for Vehicular Channel Models,» de *IEEE 91st Vehicular Technology Conference* , 2020.

[15] 5GAA, «5G Automotive Association,» [En línea]. Available: https://5gaa.org.

[16] PRNewswire, «La coexistencia pacífica de las tecnologías C-V2X e ITS-G5 ofrece los mayores beneficios netos para Europa,» EuropaPress, 2017. [En línea]. Available: https://www.europapress.es/comunicados/internacional-00907/noticia-comunicado-coexistencia-pacifica-tecnologias-v2x-its-g5-ofrece-mayores-beneficios-netos-europa-20171213163044.html.

[17] S. M. Omkar B, «Automotive V2X Market by Communication (Vehicle-to-vehicle (V2V), Vehicle-to-infrastructure (V2l), Vehicle-to-pedestrian (V2P), Vehicle-to-grid (V2G), Vehicle-to-cloud (V2C), and Vehicle-to-device (V2D)), Connectivity (Dedicated Short-range Communication,» Allied Market Research, 2020. [En línea]. Available: https://www.alliedmarketresearch.com/automotive-v2x-market-A07120.

[18] C-Roads, «Platform: C-Roads,» [En línea]. Available: https://www.c-roads.eu/platform.html.

[19] C.-R. España, «C-Roads España,» [En línea]. Available: https://www.c-roads.es/piloto-dgt-3-0.

[20] DGT, «Qué es DGT 3.0,» [En línea]. Available: https://www.dgt.es/muevete-con-seguridad/tecnologia-e-innovacion-en-carretera/dgt-3.0/.

[21] Volkswagen, «Car2X in the new Golf: A “technological milestone”,» [En línea]. Available: https://www.volkswagen-newsroom.com/en/stories/car2x-in-the-new-golf-a-technological-milestone-5919.

[22] Audi, «Car2X & C-V2X – connected vehicle pilot projects from the US,» [En línea]. Available: https://www.audi.com/en/innovation/autonomous-driving/car-to-x.html.

[23] Youtube, «Audi Shows Us Their Cellular Vehicle-To-Everything (C-V2X) Bicyclist Safety Tech,» [En línea]. Available: https://www.youtube.com/watch?v=9k_Z2ZUvSfo.

[24] Y. S. Y. G. R. Y. Jian Wang, «A survey of vehicle to everything (V2X) testing,» *Sensors,* vol. 19, nº 2, p. 334, 2019.

[25] C. L. Y. Z. a. D. Y. M. Shi, «DSRC and LTE-V communication performance evaluation and improvement based on typical V2X application at intersection,» de *2017 Chinese Automation Congress (CAC)*, Pekín.

[26] S. Z. C. P. G. T. Z. Z. J. Z. Q. Wu, «Performance Analysis of Cooperative Intersection Collision Avoidance with C-V2X Communications,» de *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, 2020.

[27] A. K. P. A. S. a. K. A. U. M. A. Kumar, «Intersection Collision Warning Using Wireless System Module via Cellular-Vehicle to Everything Technology,» de *International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 2022.

[28] G. A. C. C. C. C. F. M. S. S. Marco Malinverno, «Edge-based collision avoidance for vehicles and vulnerable users: An architecture based on MEC,» *IEEE Vehicular Technology Magazine,* vol. 15, nº 1, pp. 27-35, 2020.

[29] V. P. G. G. G. A. V. Vittorio Astarita, «A Review of the use of traffic simulation for the evaluation of traffic safety levels: can we use simulation to predict crashes?,» *Transportation Research Procedia,,* vol. 52, pp. 244-251, 2021.

[30] B. Alonso, V. Astarita, L. Dell'Olio, V. Giofrè, G. Guido, M. Marino, W. Sommario y Vitale, «Validation of Simulated Safety Indicators with Traffic Crash Data,» *Sustainability,* vol. 12, 2020.

[31] Y. W. X. Y. X. L. K. D. Q. X. Yan Huang, «Using a V2V- and V2I-based collision warning system to improve vehicle interaction at unsignalized intersections,» *Journal of Safety Research,* vol. 83, pp. 282-293, 2022.

[32] T. M. W. a. M. Z. O. Than Than Yu, «Rules Based Intersection Collision Avoidance System for V2X Safety,» *International Journal of Advances in Scientific Research and Engineering,* vol. 5, nº 10, 2019.

[33] R. y. M. Belwal, «Hardware Implementation of VANET Communication based Collision Warning System,» de *Proceedings of the Fourth International Conference on Communication and Electronics Systems (ICCES)*, 2019.

[34] «Cyber Security Market Size, Share & COVID-19 Impact Analysis, By Component (Solution and Services), By Deployment Type (Cloud and On-Premise), By Enterprise Size (Small & Medium Enterprise and Large Enterprise), By Industry (BFSI, IT and Telecommunication,» *Market Research Report,* 2021.

[35] M. C. Amrita Ghosal, «Security issues and challenges in V2X: A Survey,» *Computer Networks,* vol. 169, 2019.

[36] I. K. Thanassis Giannetsos, «Securing V2X Communications for the Future - Can PKI Systems offer the answer?,» de *ARES '19: Proceedings of the 14th International Conference on Availability, Reliability and Security*, Nueva York, 2019.

[37] M. B. P. D. Patrik Hummel, «Data sovereignty: A review,» *Big Data & Society,* vol. 8, nº 1, 2021.

[38] A. C. F. F. M. C. Y. Y. K.-K. R. C. Christian Esposito, «On Data Sovereignty in Cloud-Based Computation Offloading for Smart Cities Applications,» *IEEE Internet Of Things Journal,* vol. 6, nº 3, 2019.

[39] G. d. C. Ópticas, «Proyecto ARTEMIS,» [En línea]. Available: https://gco.uva.es/proyecto-artemis/.

[40] 3GPP, «LTE Overview,» [En línea]. Available: https://www.3gpp.org/technologies/keywords-acronyms/98-lte.

[41] A. P. Frédéric Launay, «LTE Advanced Pro,» O'Reilly, [En línea]. Available: https://www.oreilly.com/library/view/lte-advanced-pro/9781786304308/c01.xhtml.

[42] C. M. Jose Núñez, «Software Defined Radio (SDR) on Radiocommunications Teaching,» de *Conference: 10th Annual International Technology, Education and Development Conference* , Valencia, 2016.

[43] Nuand, «bladeRF 2.0 micro xA9,» [En línea]. Available: https://www.nuand.com/product/bladerf-xa9/?gclid=CjwKCAjwvsqZBhAlEiwAqAHElT-CNZt6hoak9aRwdNwA4GmVHcRZ_DbeD-gPgLM3ny4m2gKvpfP7ThoCIbcQAvD_BwE.

[44] X. W. H. L. R. X. a. K. Z. Z. Geng, «Performance analysis and comparison of GPP-based SDR systems,» de *International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications (MAPE)*, 2017.

[45] A. W. Services, «Future: Edge MQTT Broker for V2X,» [En línea]. Available: https://docs.aws.amazon.com/whitepapers/latest/designing-next-generation-vehicle-communication-aws-iot/future-edge-mqtt-broker-for-v2x.html.

[46] B. Mishra y A. Kertesz, «The Use of MQTT in M2M and IoT Systems: A Survey,» *IEEE Access,* 2020.

[47] T. H. Team, «Quality of Service (QoS) 0,1, & 2 MQTT Essentials: Part 6,» [En línea]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/.

[48] B. Mishra, B. Mishra y A. Kertesz, «Stress-Testing MQTT Brokers: A Comparative Analysis of Performance Measurements,» *Energies,* 2021.

[49] ScalAgent, «Benchmark of MQTT servers,» Enero 2015. [En línea]. Available: http://www.scalagent.com/IMG/pdf/Benchmark_MQTT_servers-v1-1.pdf.

[50] Mosquitto, «mosquitto.conf man page,» [En línea]. Available: https://mosquitto.org/man/mosquitto-conf-5.html.

[51] S. a. P. G. S. a. S. S. a. T. J. a. K. F. a. S. C. a. D. F. Laux, «Demo: OpenC2X — An open source experimental and prototyping platform supporting ETSI ITS-G5,» de *IEEE Vehicular Networking Conference (VNC)*, 2016.

[52] C. labs, «OpenC2X standalone,» [En línea]. Available: https://github.com/florianklingler/OpenC2X-standalone.

[53] S. R. Group, «Cloud Market Growth Rate Nudges Up as Amazon and Microsoft Solidify Leadership,» [En línea]. Available: https://www.srgresearch.com/articles/cloud-market-growth-rate-nudges-amazon-and-microsoft-solidify-leadership.

[54] 5. association, «C-V2X Use Cases and Service Level Requirements Volume I,» 2020. [En línea]. Available: https://5gaa.org/wp-content/uploads/2020/12/5GAA_T-200111_TR_C-V2X_Use_Cases_and_Service_Level_Requirements_Vol_I-V3.pdf.

[55] RAC, «Stopping distances made simple,» [En línea]. Available: https://www.rac.co.uk/drive/advice/learning-to-drive/stopping-distances/.

[56] A. &. H. J. Camacho, «A sawtooth waveform inspired pitch estimator for speech and music.,» *The Journal of the Acoustical Society of America,* 2008.

[57] I. V. Vázquez, «Trabajo de Fin de Máster: Integración de sistemas de posicionamiento indoor para un testbed de Edge Computing para el soporte de vehículos conectados,» 2022.

[58] srsRAN, «5G NSA End-to-End,» [En línea]. Available: https://docs.srsran.com/en/latest/app_notes/source/5g_nsa_zmq/source/index.html.