
UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

# Implementación del protocolo GeoNetworking en un software de comunicación V2X

Autor:
**D. José Luis Santamaría Corrales**

Tutor:
**D. Juan Carlos Aguado Manzano**

Valladolid, 21 de julio de 2022

TÍTULO: **Implementación del protocolo GeoNetworking en un software de comunicación V2X**

AUTOR: **D. José Luis Santamaría Corrales**

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

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

**TRIBUNAL**

PRESIDENTE: **D. Ramón Durán Barroso**

VOCAL: **D. Ramón de la Rosa Steinz**

SECRETARIO: **D. Javier Aguiar Pérez**

FECHA: 21 de julio de 2022

CALIFICACIÓN:

## Resumen del TFM

Los Sistemas Inteligentes de Transporte Cooperativos (C-ITS) buscan aumentar la eficiencia del tráfico y mejorar la seguridad vial. Una tecnología clave en los C-ITS es la comunicación V2X. El objetivo de este Trabajo Fin de Máster es la implementación del protocolo GeoNetworking, parte de la arquitectura ETSI C-ITS estandarizada en la Unión Europea, en el software de comunicación V2X OpenC2X desarrollado en el grupo CCS Labs de la Universidad de Paderborn, Alemania.

Para ello se ha realizado un estudio detallado de la evolución de los C-ITS y su estandarización en la Unión Europea, para pasar a analizar con mayor profundidad el protocolo GeoNetworking y su integración con el resto de la arquitectura de comunicación. Por último, se ha llevado a cabo el desarrollo del software que integra GeoNetworking en OpenC2X.

Aunque GeoNetworking es un protocolo complejo, que maneja una gran cantidad de estructuras de datos auxiliares, se ha conseguido el intercambio de los mensajes CAM y DENM cumpliendo el formato indicado por el estándar, a pesar de no ser una implementación completa del protocolo. Sin embargo, el proceso de implementación nos ha permitido una revisión completa de OpenC2X y se ha llevado a cabo la actualización de varios módulos que utilizaban versiones anteriores de los estándares ETSI.

## Palabras clave

V2X, C-ITS, GeoNetworking, OpenC2X, CAM, DENM, ETSI-5G, LTE-V2X

## Abstract

Cooperative Intelligent Transport Systems (C-ITS) aim to increase traffic efficiency and improve road safety. V2X communication is a key technology for C-ITS development. The main purpose of this project is the implementation of the GeoNetworking protocol, part of the ETSI C-ITS architecture standardized in the European Union (EU), in the V2X communication software OpenC2X developed by the CSS Labs group of the University of Paderborn, Germany.

For this purpose, a comprehensive study of the evolution of C-ITS and its standardization in the EU has been carried out, to proceed to a more in-depth analysis of the GeoNetworking protocol and its integration within the complete communication architecture. Finally, the development of the software which integrates GeoNetworking in OpenC2X has been carried out.

Although GeoNetworking is a complex protocol, which handles a lot of auxiliary data structures, the exchange of CAM and DENM messages has been achieved in compliance with the format indicated in the standard, despite not being a complete implementation of the protocol. However, the implementation process has allowed a complete revision of OpenC2X and the update of several modules that used outdated versions of the ETSI standards had been done.

## Keywords

V2X, C-ITS, GeoNetworking, OpenC2X, CAM, DENM, ETSI-5G, LTE-V2X

## Agradecimientos

Me gustaría agradecer en primer lugar a Juan Carlos Aguado Manzano, mi tutor de este TFM, por toda su ayuda y guía durante la realización de este trabajo y muy especialmente por su paciencia y saber adaptarse a mi ritmo, seguramente muy diferente al que está acostumbrado.

Y por supuesto agradecerle a mi familia, especialmente a mis padres y Elisa, por su infinita paciencia y todo lo que han trabajado durante la realización de este TFM para allanarme el camino y que pudiera finalizarlo.

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

# ÍNDICE

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# Glosario de términos

**AC:** Access Class
**ASN:** Abstract Syntax Notation
**BC:** BroadCast
**BTP:** Basic Transport Protocol
**C-ITS:** Cooperative Intelligent Transport Systems
**C-V2X:** Cellular Vehicle to Everything
**CAM:** Cooperative Awareness Message
**CBF:** Contention-Based Forwarding
**DAD:** Duplicate Address Detection
**DE:** Destination
**DENM:** Decentralized Environmental Message
**EPV:** Ego Position Vector
**FIFO:** First In First Out
**GAC:** Geographically-Scoped Anycast
**GBC:** Geographically-Scoped Broadcast
**GF:** Greedy Forwarding
**GN:** GeoNetworking
**GN_ADDR:** GeoNetworking ADDRess
**GN-PDU:** GeoNetworking Protocol Data Unit
**GN-SDU:** GeoNetworking Service Data Unit
**GN6-SDU:** GN6 Service Data Unit
**GN6ASL:** GeoNetworking to IPv6 Adaptation Sub-Layer
**GUC:** Geographically-Scoped Unicast
**HST:** Header Sub-Type
**HT:** Header Type
**ITS:** Intelligent Transport Systems
**LocT:** Location Table
**LocTE:** Location Table Entry
**LPV:** Local Position Vector
**LS:** Location Service
**LT:** LifeTime
**LTE:** Long Term Evolution
**LTE-V2X:** Long Term Evolution Vehicle to Everything
**MAC:** Medium Access Control
**MHL:** Maximum Hop Limit
**MIB:** Management Information Base
**MID:** MAC ID
**MTU:** Maximum Transmit Unit
**NH:** Next Header
**OBD:** On Board Diagnostic
**OBU:** On Board Unit
**OCB:** Out of the Context of a BSS
**OSI:** Open Systems Interconnection
**PAI:** Position Accuracy Indicator
**PCI:** Protocol Control Information
**PDR:** Packet Data Rate
**PDU:** Protocol Data Unit
**PL:** Payload Length
**POS:** POSition
**PV:** Position Vector
**RHL:** Remaining Hop Limit
**RSU:** Road Side Unit
**SAE:** Society of Automobile Engineers
**SCF:** Store Carry & Forward
**SDU:** Service Data Unit
**SE:** SEnder
**SHB:** Single Hop Broadcast
**SN:** Sequence Number
**SO:** SOurce
**SPV:** Short Position Vector
**ST:** Station Type
**T-SDU:** Transport Service Data Unit
**TAI:** Temps Atomique International (International Atomic Time)
**TC:** Traffic Class
**TC ID:** Traffic Class Identifier
**TSB:** Topologically Scoped Broadcast
**TST:** TimeSTamp
**UC:** UniCast
**UTC:** Universal Time Coordinated
**V2I:** Vehicle to Infrastructure
**V2V:** Vehicle to Vehicle
**V2X:** Vehicle-to-Everything
**WGS:** World Geodetic System

# 1

# Introducción

## 1.1 Motivación

Los sistemas de transporte se han enfrentado históricamente a varios retos entre los cuales los más importantes son el aumento de la seguridad, la mejora de la eficiencia y la reducción de los efectos en la conducción de los fenómenos medioambientales adversos.

Aunque la tendencia en el número de muertes en accidentes de tráfico es a la baja, con una disminución del 23% entre los años 2010 y 2019 en el conjunto de la Unión Europea y un 31% en España, aproximadamente 28.000 personas fallecieron en accidentes de tráfico en 2019 en Europa (último año representativo debido a las restricciones de movilidad impuestas en los diferentes países debido a la COVID-19). Además, se calcula que por cada fallecido otras 5 personas sufren heridas de gravedad que afectan de manera importante a su calidad de vida, lo cual representa de manera habitual un coste mayor para la sociedad en concepto de rehabilitación y gastos médicos. [1]

Por otro lado, el transporte por carretera sigue siendo responsable de la mayor parte de las emisiones del conjunto del transporte, en lo que se refiere a gases de efecto invernadero y a contaminantes del aire, representado más del 70% de las emisiones de gas de efecto invernadero originadas por el transporte, el 39% de los óxidos de nitrógeno (NOx) y el 13% de las emisiones de partículas.

No menos importante, a diario los atascos en las carreteras suponen grandes costes a la economía de la UE, estimados en el 1% del PIB. [2]

Es por todo ello por lo que durante los últimos años ha habido un creciente interés en el desarrollo de los Sistemas Inteligentes de Transporte. Los Sistemas Inteligentes de

Transporte (ITS) representan la aplicación de las Tecnologías de la Información y la Comunicación (TIC) al sector del transporte. Estas tecnologías están cada vez más presentes tanto en los vehículos como en las infraestructuras de transporte tales como semáforos, señales de tráfico, paneles informativos o sistemas de peaje.

Los vehículos modernos incorporan un número creciente de sensores, radares y cámaras que captan tanto información propia como del entorno y las utilizan para ofrecer al conductor información útil para mejorar su percepción a la hora de conducir y tomar decisiones. También permiten el desarrollo de aplicaciones activas que ayudan a una conducción más segura y eficiente, conocidas como ADAS (Advanced Driver-Assistance Systems: Sistemas Avanzados de Asistencia al Conductor) entre las cuales se incluyen los reguladores de velocidad adaptativos, la detección de vehículos en los ángulos muertos, la detección de obstáculos y peatones con frenado activo de emergencia, los sistemas de asistencia al aparcamiento, etc.

La posibilidad de compartir toda esta información con otros vehículos o con la infraestructura de transporte da lugar a los Sistemas de Transporte Inteligente Cooperativos (C-ITS), los cuales pueden tener un impacto significativo en el incremento de la seguridad en el transporte. Además, pueden incrementar la eficiencia del sector del transporte (incluyendo la eficiencia energética) y reducir la congestión del tráfico. Resumiendo, conducen a una movilidad más segura, sostenible y más limpia. [3]

Es este intercambio de información el que da a los C-ITS su carácter cooperativo. Dicho intercambio se hace a través de las tecnologías de comunicación conocidas como comunicación V2X (Vehicle to Everything), término que engloba múltiples tecnologías:

- V2V: Comunicación Vehículo a Vehículo directa.
- V2I: Comunicación Vehículo a Infraestructura. Se utiliza para el intercambio de información con las denominadas Road Side Units (RSU), como pueden ser sistemas de peaje, paneles informativos, semáforos, etc.
- V2P: Comunicación Vehículo a Peatón. Se utiliza para el intercambio de información con peatones y otro tipo de usuarios como ciclistas o usuarios de patinetes eléctricos.
- I2I: Comunicación entre RSUs.
- Otras variantes como V2M (Vehículo a Motocicleta) o V2N (Vehículo a red).

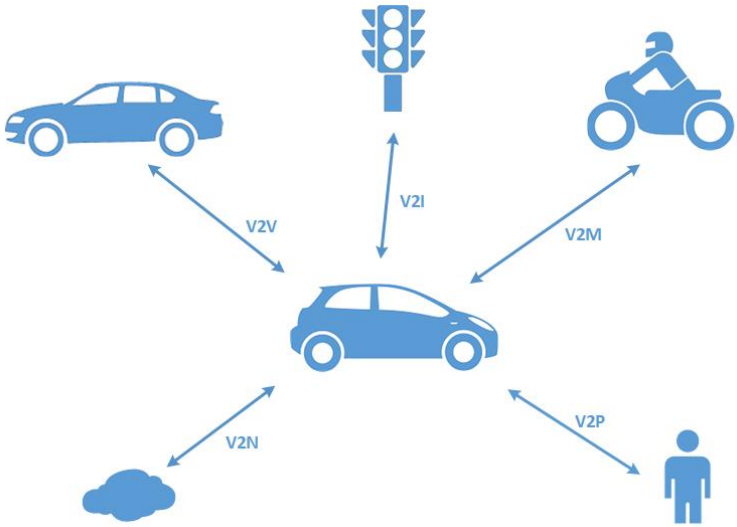


**Figura 1: Ejemplos de comunicación V2X**

Las aplicaciones de los C-ITS son amplias: desde aviso de incidencias como frenada de emergencia de vehículos precedentes, aproximación de vehículos de emergencia, estado del tráfico y posibles retenciones o eventos como accidentes, hasta implementación de sistemas como las caravanas de camiones (platooning) o, en un futuro a medio plazo, el vehículo autónomo.

El gran impacto positivo que los C-ITS pueden ejercer en la seguridad, el incremento de la eficiencia del transporte, incluida la energética, así como la reducción de la congestión del tráfico, puede reportar beneficios a nivel económico al hacer para los inversores más atractivos países y regiones donde los niveles de congestión y contaminación sean menores. Por otro lado, los consumidores se benefician de una mayor seguridad, un menor gasto energético y una información del tráfico más fiable. No menos importante, los C-ITS representan una oportunidad de negocio para la industria del sector del transporte, en especial la automovilística, al presentar un valor añadido a sus productos, incrementando su competitividad [3].

Es por ello que desde las instituciones ha habido un gran interés en promover soluciones estandarizadas que garanticen objetivos de interoperabilidad e interconectividad entre diferentes sistemas y servicios.

En Estados Unidos, el IEEE (Institute of Electrical and Electronics Engineers) y la SAE (Society of Automobile Engineers) definieron un conjunto de estándares denominados DSRC (Dedicated Short Range Communication), término que en ocasiones

se usa de forma generalizada para las diferentes soluciones basadas en WiFi, y en Europa el ETSI y CEN crearon una arquitectura llamada C-ITS.

En ambos casos la tecnología utilizada para la capa de acceso se basaba en el estándar WLAN IEEE 802.11, concretamente la enmienda 802.11p, desarrollada específicamente para la comunicación en entornos vehiculares y que más tarde fue integrada en el estándar en 2012. En Europa, el Comité Técnico de ITS del ETSI estandarizó una versión adaptada a la banda de frecuencias de 5875-5925MHz, la banda asignada en Europa para los ITS, denominada ITS-G5.

Posteriormente el 3GPP (3rd Generation Partnership Project) desarrolló una alternativa basada en redes celulares, denominada LTE-V2X o C-V2X, cuya primera versión se publicó en la release 14 del 3GPP en 2016.

Estas dos tecnologías de comunicación V2X luchan por ser la tecnología estándar preferente para los C-ITS y, en un futuro, el vehículo autónomo. Como veremos posteriormente en el capítulo dedicado al estado del arte, ambas tecnologías presentan ventajas e inconvenientes y actualmente muchos de los trabajos que se están realizando se centran en conseguir la coexistencia de las dos.

Pero la arquitectura de comunicación C-ITS no solo define la capa de acceso, sino que plantea una pila de protocolos completa, que sigue los principios del modelo OSI de protocolos de comunicación en capas extendido para la inclusión de las aplicaciones ITS. [4]

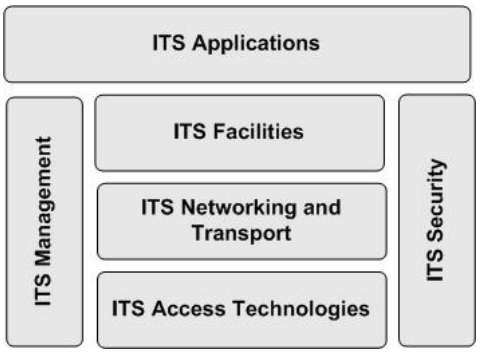


**Figura 2: Pila de protocolos de una estación ITS [5]**

La pila de protocolos define 3 capas horizontales más otras 2 entidades verticales, las capas de Gestión y Seguridad, que dan servicio de manera transversal al resto de capas y aplicaciones ITS en la parte superior:

- **Capa de Tecnologías de Acceso:** cubre las capas física y de enlace de datos del modelo de referencia OSI.
- **Capa de Red y Transporte:** compuesto por protocolos para la entrega de datos entre estaciones ITS y la red. Incluyen el enrutamiento de datos entre la fuente y el destino a través de nodos intermedios. Es posible usar varios protocolos de Red, como IPv6 con extensiones para movilidad (NEMO: Network Mobility), CALM (Communications Access for Land Mobiles) y varios protocolos de transporte como UDP y TCP. Pero también en la arquitectura C-ITS se crearon dos nuevos protocolos de red y transporte: GeoNetworking y BTP (Basic Transport Protocol).
- **Capa de Servicios (*Facilities*):** contiene funcionalidades de las capas de aplicación, presentación (por ejemplo, codificación y decodificación ASN.1, encriptación) y sesión (comunicación *inter-host*) con añadidos para C-ITS. En esta capa se definen protocolos muy importantes para las comunicaciones V2X como el protocolo CAM (Cooperative Awareness Messages), que permite el intercambio periódico de mensajes con información de estado crítica para el soporte de aplicaciones de seguridad y eficiencia, de manera que los vehículos puedan seguir la posición y movimiento de otros vehículos [6] o DEMN (Decentralized Environmental Notification Messages) que son mensajes de información relacionada con un evento que tenga un impacto importante en la seguridad de las carreteras o las condiciones del tráfico. [7]
- **Aplicaciones ITS:** las aplicaciones que implementan los casos de uso para seguridad vial, eficiencia del tráfico, *infotainment* y negocio.
- **Gestión ITS:** responsable de la configuración de una estación ITS, intercambio de información entre las diferentes capas y otras tareas.
- **Seguridad ITS:** proporcionan servicios de seguridad y privacidad, incluyendo mensajes seguros en diferentes capas de la pila de protocolo, gestión de identidades y credenciales de seguridad y aspectos para plataformas seguras como *firewalls*, *gateway* de seguridad, *hardware* a prueba de modificaciones (*tamper-proof*).

La capa de acceso ha sido estudiada anteriormente en los Trabajos Fin de Máster de Javier Fernández Pastrana [8] y la capa de *Facilities* en los Trabajos Fin de Grado de María Pilar Sánchez Martín[9] y Alejandro Lobo González [10]. En dichos trabajos, así como en el Trabajo Fin de Máster de Roberto Herreras Babón [11], se utilizó el software de comunicación OpenC2X [12], desarrollado por el grupo de investigación de sistemas

distribuidos embebidos de la Universidad de Paderborn (Alemania), para implementar una estación ITS tanto de a bordo como una Road Side Unit. Aunque este software tiene muchos de los elementos de la arquitectura de comunicación C-ITS de ETSI no implementaba ninguno de los protocolos de la capa de Red y Transporte, BTP y GeoNetworking. BTP fue posteriormente implementado por Daniel Monje González en su Trabajo Fin de Grado [13], pero GeoNetworking seguía sin estar implementado. Por lo tanto, este será uno de los principales objetivos de este Trabajo Fin de Máster.

## 1.2 Objetivos

El propósito final de este Trabajo Fin de Máster es la implementación del protocolo GeoNetworking, perteneciente a la capa de Red y Transporte de la arquitectura comunicación definida por el conjunto de estándares C-ITS de la Unión Europea, en el software de comunicación OpenC2X.

Dicho software, como ya se ha mencionado, fue utilizado en los trabajos de fin de estudios de varios compañeros en cursos anteriores. Sin embargo, debido a que alguno de esos trabajos, y el propio software, tienen algunos años se definieron una serie de objetivos secundarios:

- Estudiar la situación actual de las tecnologías V2X, y más concretamente las tecnologías de la capa de acceso basadas en WiFi (802.11/ETSI-G5) y redes celulares (LTE y 5G).
- Estudiar el estado de la estandarización de las tecnologías V2X en la Unión Europea y los proyectos actualmente en marcha.
- Estudiar la evolución del software OpenC2X y de sus alternativas en los últimos años.
- Evaluar la utilización de otro software de comunicación V2X alternativo en nuestras estaciones ITS.
- Finalmente, implementar el protocolo GeoNetworking en nuestro software de comunicación y realizar las pruebas necesarias para comprobar su correcto funcionamiento.

## 1.3 Fases y Métodos

La primera fase del proyecto fue dedicada al estudio de la situación de las tecnologías de comunicación V2X, mediante la búsqueda en trabajos de investigación y noticias sobre el tema.

Debido a la diversidad de trabajos y fuentes, con información en ocasiones contradictoria o incompleta, llevó a una segunda fase de estudio de la evolución histórica de la actividad de estandarización, centrándonos en el caso europeo, que abarca varios actores como ETSI, La Comisión Europea u organismos como el Car-2-Car Consortium, que incluye también a fabricantes de automóviles y empresas tecnológicas.

La siguiente fase, ya más centrada en nuestro objetivo final, fue revisar si había alguna nueva alternativa a OpenC2X más adecuada a nuestros usos. En esta fase se realizó la instalación y pruebas de Vanetza.

Las pruebas de Vanetza revelaron la necesidad de actualizar algunos de los módulos de OpenC2X ya implementados, pues se basaban en versiones antiguas de los estándares correspondientes.

La siguiente fase, una vez decidida la implementación de GeoNetworking en OpenC2X en vez de realizar una migración a Vanetza, fue el estudio teórico del protocolo GeoNetworking para comprender su funcionamiento, sus estructuras de datos, y su integración con el resto de protocolos de la arquitectura.

Posteriormente se realizó la programación del servicio que implementa GeoNetworking en OpenC2X, y las modificaciones necesarias en el resto de módulos de OpenC2X existentes para realizar una integración que respetase al máximo posible lo establecidos por los diferentes estándares ETSI que aplican a cada una de las partes implicadas.

Finalmente se realizaron las pruebas necesarias para comprobar el correcto funcionamiento del protocolo implementado, y que éste permitía una correcta comunicación entre los ordenadores utilizados como estaciones ITS.

## 1.4 Medios utilizados en la realización del TFM

Para la realización de este Trabajo Fin de Máster se han utilizado los siguientes medios:

- Dos mini-PCs de pequeño tamaño con tarjetas de red inalámbricas con chipset Atheros ATH9K y sistema operativo Ubuntu 16.04.
- Un PC de sobremesa más potente para realizar el desarrollo del software y compilación con mayor agilidad ejecutando también Ubuntu 16.04 y el IDE Visual Studio Code.
- Dos geo-localizadores GPS Holux GPSlim 236.

## 1.5 Estructura de la Memoria del TFM

En el capítulo "Estado del arte" se realizará un estudio retrospectivo de la evolución de los ITS y C-ITS en los últimos años, con el foco puesto en la Unión Europea, desde la Decisión Armonizada que asignaba el rango para el uso de la banda de 5.9GHz para los C-ITS hasta el momento actual, en el que la lucha entre las tecnologías WiFi basadas en el estándar 802.11 y las tecnologías C-V2X basadas en redes celulares sigue abierta. Se analizarán ambos enfoques, sus ventajas e inconvenientes y los avances y proyectos en marcha. Se realizará también un breve estudio sobre las alternativas de comunicación V2X Open Source disponibles.

En el capítulo "El protocolo GeoNetworking" se realizará un estudio teórico del protocolo GeoNetworking, incluyendo los escenarios de comunicación, las estructuras de datos y el funcionamiento del mismo.

El capítulo "Instalación de Vanetza y comunicación con OpenC2X" se analizarán los pasos que han sido necesarios para la instalación y puesta en funcionamiento de Vanetza en nuestros ordenadores y se comprobará la interoperabilidad entre Vanetza y OpenC2X.

En el capítulo "Implementación del protocolo GeoNetworking en OpenC2X" se analizan los diferentes ficheros y clases que implementan nuestro servicio GeoNetworking y las modificaciones que ha sido necesario llevar a cabo en el código fuente de otros

módulos de OpenC2X para la integración del nuestro servicio. También se muestra el resultado de las pruebas realizadas, que demuestran el correcto funcionamiento de OpenC2X con el servicio GeoNetworking integrado.

En el capítulo “Conclusiones y líneas futuras” se hace análisis de grado de consecución de los objetivos marcados y se marcan algunas de las mejoras futuras que podrían ser realizadas en futuros Trabajos de Fin de Grado/Máster.

# 2

# Estado del arte

A lo largo del siguiente capítulo se revisará la evolución histórica de los C-ITS centrándonos en la Unión Europea, desde los primeros programas de investigación en la materia hasta el estado actual marcado por la aparición y popularización de las diferentes tecnologías inalámbricas.

A continuación, se hará un análisis de las dos familias de tecnologías inalámbricas propuestas para su uso en la comunicación V2X, basadas por un lado en el estándar IEEE 802.11 y por otro en las redes celulares.

Por último, se hará una introducción a OpenC2X y Vanetza, dos softwares Open Source que realizan una implementación de la arquitectura ETSI C-ITS.

## 2.1 Evolución histórica de los C-ITS

### *2.1.1 Primeros pasos*

Los primeros programas de investigación en Europa sobre C-ITS se remontan a los años 80s, con el programa PROMETHEUS (1987-1995), que buscaba reducir los efectos adversos del tráfico por carretera mediante el uso de Tecnologías de la Información. En el proyecto, en el que colaboraron numerosas universidades y fabricantes de automóviles de toda Europa, se definieron tres áreas de investigación principales: Asistencia al conductor, comunicación entre vehículos y comunicación entre vehículo y la carretera (*roadside*). Algunas de las demostraciones de los resultados de estos proyectos que se realizaron durante el encuentro final en 1994 incluían la conducción cooperativa, el regulador de velocidad inteligente, la llamada automática de emergencia o el asistente de mantenimiento en el carril. [14]

Posteriormente, a principios del siglo XXI, el avance de las Tecnologías de la Información y las Comunicación (TICs), con la aparición de tecnologías inalámbricas cada vez más eficientes, el desarrollo de los sistemas GPS y la miniaturización y abaratamiento de los sistemas embebidos, propició la aparición de una ola de nuevas actividades de investigación y desarrollo tanto a nivel académico como de la industria, con proyectos como SAFESPOT, GeoNet, SEVECOM, DRIVE C2X o SCORE@F [6].

### *2.1.2 Desarrollo del conjunto de estándares C-ITS europeo.*

Debido a la importancia de los problemas a resolver y al gran impulso económico que representan los C-ITS, un requisito fundamental para el desarrollo de los sistemas de comunicación V2X son los estándares internacionales, que proporcionan especificaciones para asegurar la interconexión entre (sub-)sistemas y componentes V2X, así como interoperabilidad entre las implementaciones de diferentes fabricantes [15]. En el caso de la Unión Europea se hace también necesario asegurar la interoperabilidad de los sistemas en toda la Unión.

En 2008, el Comité de Comunicaciones Electrónicas (ECC: Electronic Communications Commitee) de la Conferencia Europea de Administraciones de Correo y Telecomunicaciones (CEPT: Conférence Européenne des Administration des Postes et des Télécommunications) publicó la Decisión Armonizada para el uso de la banda de 5.9GHz para los Sistemas Inteligentes de Transporte. En dicha decisión, se establece el uso de la banda de frecuencia de 5875-5905 MHz por parte de los ITS relacionados con seguridad vial, con una posible ampliación de 20MHz hasta los 5925MHz. Esto sirvió de base para la Decisión 2008/671/EC del 5 de agosto de la Comisión Europea en la que designaba el uso de esa banda de frecuencia en todos los Estados Miembros. [16]

Una vez decidido el espectro a utilizar, la Comisión Europea publicó su mandato m453 [3] a finales de 2009, en el cual instaba a las organizaciones europeas de estandarización ETSI (European Telecommunications Standards Institute) y CEN (Comité Européen de Normalisation) a crear un conjunto mínimo de estándares C-ITS para asegurar la interoperabilidad entre las comunicaciones V2V, V2I/I2V y I2I.

Tanto ETSI como CEN aceptaron formalmente dicho mandato para elaborar dicho conjunto mínimo y consistente de estándares, cuya primera versión, denominada Release 1, vio la luz en 2013.

La tecnología de acceso designada para la capa de enlace en la primera versión de dicha Release 1 dentro de la arquitectura C-ITS es la denominada ITS-G5, una tecnología WLAN de corto alcance (DSRC) basada en la tecnología inalámbrica 802.11p, que es una evolución del estándar 802.11a adaptado a las redes vehiculares.

En 2014 la Comisión Europea creó la Plataforma para el despliegue de los sistemas C-ITS en la Unión Europea (C-ITS platform), que ofrece un instrumento operativo para el diálogo, el intercambio de conocimientos técnicos y la cooperación entre la Comisión, las partes interesadas del sector público de los Estados Miembros, las autoridades locales o regionales y las partes interesadas del sector privado, como los fabricantes de coches (reunidos desde 2002 en el Car 2 Car Communication Consortium), los fabricantes de equipos, los operadores de las carreteras y las telecomunicaciones, y los proveedores de servicios.

La primera fase de trabajos de dicha plataforma dio como resultado un informe de expertos [17] en el que, entre otras cuestiones, se identificaron una serie de servicios de implantación prioritaria debido a sus beneficios sociales y a la madurez de la tecnología. Estos servicios fueron denominados “servicios del día 1” (Day-1 services) y se muestran en la Tabla 1.

| |
|---|
| **Notificaciones de ubicación peligrosa:**<br>• avisos de circulación lenta o congestionada y avisos sobre el tráfico;<br>• avisos de obras en la carretera;<br>• condiciones meteorológicas;<br>• luz de frenado de emergencia;<br>• vehículo de emergencia aproximándose;<br>• otros peligros.<br>**Aplicaciones de señalización:**<br>• señalización en el vehículo;<br>• límites de velocidad en el vehículo;<br>• incumplimiento de la señalización / seguridad en los cruces;<br>• solicitud de señalización prioritaria por parte de los vehículos designados;<br>• señal luminosa verde para la velocidad óptima recomendada;<br>• datos compartidos por el vehículo;<br>• amortiguador de movimientos sísmicos (forma parte de la categoría «advertencia de peligro local» del Instituto Europeo de Normas de Telecomunicación, ETSI) |

**Tabla 1: Servicios del día 1**

A estos servicios se le añadieron una serie de servicios que, aunque sus especificaciones o estándares podrían no estar listos aún, se consideraban maduros y con

gran interés por parte del mercado. Estos servicios fueron los denominados “servicios del día 1,5” (1,5 Day services), y se muestran en la Tabla 2.

| |
|---|
| • información sobre estaciones de repostaje y de recarga para los vehículos que utilicen combustibles alternativos;<br>• protección de los usuarios vulnerables de la vía pública;<br>• gestión e información de los aparcamientos en la vía pública;<br>• información sobre los aparcamientos que no se encuentran en la vía pública;<br>• información sobre aparcamientos disuasorios;<br>• navegación conectada y cooperativa para entrar y salir de las ciudades (primer y último kilómetro, aparcamiento, consejos sobre la ruta, semáforos coordinados);<br>• información sobre el tráfico y enrutamiento inteligente. |

**Tabla 2: Servicios del día 1,5**

En cuanto a la tecnología de comunicación a utilizar, la plataforma consideraba esencial asegurar que los mensajes C-ITS se pudieran transmitir independientemente de la tecnología de comunicaciones subyacente (agnosticismo de capa de enlace) cuando fuera posible. Recomendaba además usar la tecnología ETSI ITS-G5 (IEEE802.11p) para comunicaciones de rango corto en la banda de 5.9GHz debido a que era la más madura y cercana a su despliegue, aunque estudiando si fuera posible incrementar la cobertura de los servicios C-ITS mediante la infraestructura de comunicaciones celular.

Por otro lado, el 3GPP (3rd Generation Partnership Project) estandarizó en 2016 en su release 14 su versión de tecnología de comunicación V2X basada en redes celulares, conocida como LTE-V2X o C-V2X (Cellular Vehicule to Everything). Este estándar incluye un modo de comunicación directo, denominado LTE sidelink, que presenta una alternativa a los estándares basados en 802.11p como el ETSI ITS-G5. Varias compañías del sector de la automoción y las telecomunicaciones se unieron para formar la 5GAA (5G Automotive Association), que promueve el uso de la tecnología LTE-V2X.

Esto abrió el debate, vigente a día de hoy, sobre cuál debería ser la tecnología de comunicación de corto alcance estándar preferente para el vehículo conectado y, en un futuro, el vehículo autónomo.

Del lado de LTE-V2X se encuentran marcas como BMW, Daimler, Ford, Huawei, Samsung, Qualcomm e Intel mientras que la tecnología WiFi es defendida por marcas como General Motors, NXP, Volkswagen, Volvo, Renault o Toyota. [18]

El 13 de marzo de 2019 la Comisión Europea publicó su esperada acta delegada, la cual se mostraba a favor de un enfoque de "comunicación híbrida" donde se utilizaría la tecnología ITS-G5 para comunicación de corto alcance entre vehículos debido a que se había desarrollado específicamente con ese fin, había alcanzado la madurez, se había sometido a ensayos y se había implantado [19]. Esta se vería complementada con tecnologías de mayor alcance existentes como 3G o 4G para los servicios V2I.

También se contempla una cláusula de revisión para facilitar la integración de las tecnologías LTE-V2X en un plazo de antes de 3 años desde la entrada en vigor del reglamento.

Sin embargo, en julio de 2019 el Consejo Europeo rechazó dicha acta, basado en la recomendación del comité de transportes del Parlamento Europeo, con 21 de los 28 estados miembro votando en contra de la misma, en una decisión que los defensores de la tecnología DSRC ven influenciada políticamente por el *lobby* de telecomunicaciones ETNO, cuyos miembros han gastado gran cantidad de dinero invirtiendo en tecnología 5G.

Por este motivo la Comisión Europea, recién elegida en 2019, deberá redactar una nueva propuesta al Parlamento y al Consejo Europeo con un enfoque "neutral", que todavía no ha sido presentada.

Mientras tanto, el ETSI ha ido ampliando su conjunto de estándares para integrar LTE-V2X en la arquitectura C-ITS, tanto a nivel de la capa de acceso [20] como en capas superiores, como definiendo la parte dependiente del medio del protocolo GeoNetworking [21], por lo que todo parece apuntar a una coexistencia de ambas tecnologías, lo cual plantea no pocos retos a resolver, ya que no son interoperables a nivel de acceso radioeléctrico y pugnan por el mismo espectro. La ETSI ha publicado en 2021 dos informes técnicos para la definición y evaluación de mecanismos de coexistencia en el mismo canal y en canales adyacentes entre ITS-G5 y LTE-V2X [22] [23], entre los cuales se encuentra la asignación de prioridades de acceso a los diferentes canales para cada tecnología concreta.

### *2.1.3 Despliegue actual de sistemas C-ITS.*

Aunque se esperaba que el despliegue de los sistemas C-ITS que ofrecerán los servicios de día 1 y vehículos que integrarán esta tecnología deberían haber empezado a

generalizarse en 2019, a día de hoy dicha implantación es mínima. La plataforma C-Roads, una iniciativa conjunta de los estados miembro europeos y los operadores de carreteras para ensayar e implementar los servicios C-ITS con vistas a la armonización y la interoperabilidad transfronteriza [24], integra numerosos proyectos en los estados de la Unión Europea, entre los cuales se encuentran:

- SCOOP@F, que ha realizado el despliegue de tecnología V2X utilizando ITS-G5 a lo largo de 2000Km de carretera, repartidos en 5 áreas de Francia, y 3000 vehículos PSA y Renault.
- El pasillo C-ITS Rotterdam-Frankfurt-Viena, que implementa los servicios de advertencia de obras en carretera y utilización de datos del vehículo para la gestión mejorada del tráfico en una ruta que atraviesa tres países (Países Bajos, Alemania y Austria), probando la interoperabilidad transfronteriza de los C-ITS.
- InterCor (Interoperable Corridors: Pasillos Interoperables): Es un proyecto que intenta conectar las iniciativas anteriores, el pasillo británico Londres-Dover e iniciativas C-ITS en Bélgica con el objetivo de ofrecer una red de pasillos que proporcionen continuidad de servicios C-ITS y servir como plataforma de pruebas de desarrollos presentes y futuros en Europa. [25]

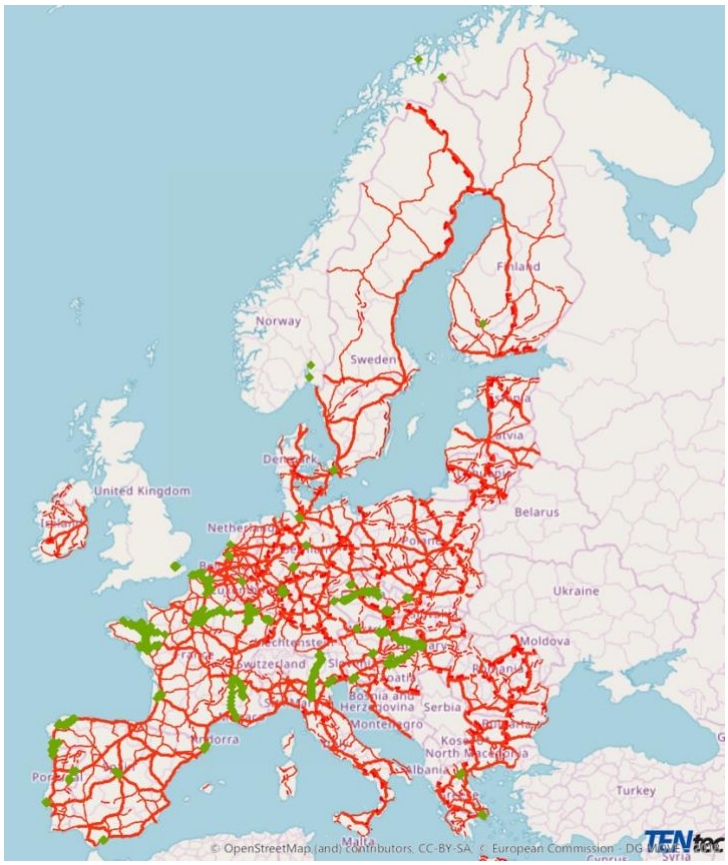


**Figura 3: Mapa de estaciones C-ITS en Europa (En Verde) [26]**

En España, también dentro de la plataforma C-Roads, se están desarrollando 5 pilotos: [27]

- **DGT 3.0**, ubicado a lo largo de toda la red de carreteras de España con una extensión aproximada de 12270 km. Se implementará utilizando tecnologías de comunicación celulares (3G y 4G / LTE).
- **SISCOGA Extended**, comprendido en la extensión que abarca el ya existente *test site* localizado en la ciudad de Vigo y su área metropolitana preparado para probar tecnología de la comunicación ITS-G5. Tendrá una extensión de 150 km.
- **Madrid**, ubicado a lo largo de la vía "Calle 30" en Madrid, con aproximadamente 32 km. Los servicios C-ITS se implementarán utilizando tecnologías de comunicación híbrida.
- **Cantábrico**, desarrollado a lo largo de 75 km en la zona norte de España utilizando tecnologías de comunicación híbrida.
- **Mediterráneo**, implementado a lo largo de 125 km en secciones de carretera seleccionadas de Cataluña y Andalucía utilizando tecnología híbrida.

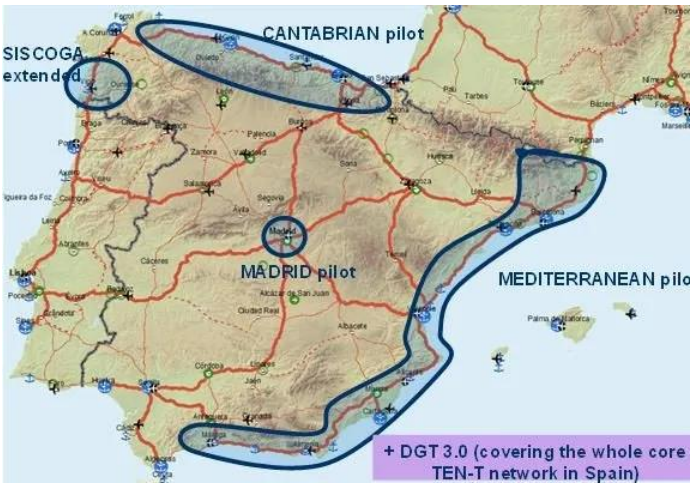


**Figura 4: Los 5 pilotos españoles [27]**

Durante este tiempo también han ido apareciendo nuevas aplicaciones y casos de uso para las tecnologías V2X y los C-ITS. Estas aplicaciones son denominadas del día 2 y posteriores.

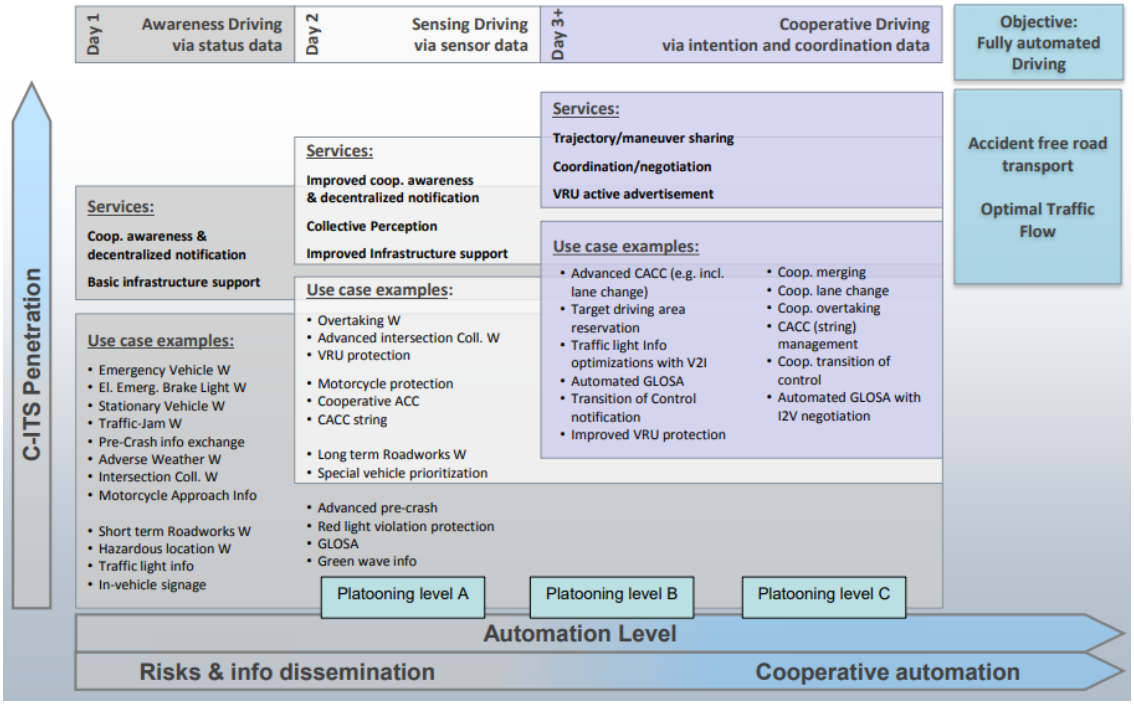


**Figura 5: Hoja de ruta de casos de uso y servicios [28]**

Muchas de esas aplicaciones van dirigidas al desarrollo del vehículo autónomo y requieren de unos requisitos más exigentes que las anteriores en cuanto a rendimiento de tasa de datos, latencia, fiabilidad, etc. La mayoría de las tecnologías actuales no son capaces de cumplir estos requisitos, por lo que se está trabajando en el desarrollo de la evolución de ambas. Así IEEE está trabajando en la enmienda 802.11bd como evolución

a 802.11p y 3GPP en la tecnología 5G NR (New Radio) V2X, que vienen suplir las carencias de sus antecesoras para las nuevas aplicaciones [29].

## 2.2 Tecnologías inalámbricas V2X

### *2.2.1 Tecnologías WiFi basadas en el estándar 802.11*

El IEEE publicó en 2010 en su enmienda 6, Wireless Access in Vehicular Environments (WAVE), el estándar 802.11p, una adaptación del estándar 802.11 para su uso en entornos vehiculares.

802.11p está basado en 802.11a y comparten muchas características. En la capa física se utiliza OFDM (Orthogonal Frecuency Division Multiplexing) con 52 subportadoras (48 de datos y 4 para pilotos), solo que en el caso de 802.11p se utiliza en el modo de operación *half-clocked*, usando canales con anchos de banda de 10MHz, en vez de 20MHz. Las posibles tasas de transferencia son de 3, 4.5, 6, 9, 12, 18, 24 y 27Mbit/s siendo obligatorio el soporte de 3 Mbit/s, 6 Mbit/s y 12 Mbit/s. Se muestra la diferencia de los parámetros principales entre 802.11a y 802.11p en la Tabla 3.

| **Parámetro** | **802.11a** | **802.11p** | **Diferencia** |
|---|---|---|---|
| Bit rate (Mbit/s) | 6, 9, 12, 18, 24, 36, 48, 54 | 3, 4.5, 6, 9, 12, 18, 24, 27 | La mitad |
| Esquema de modulación | BPSK, QPSK, 16-QAM, 64-QAM | BPSK, QPSK, 16-QAM, 64-QAM | Sin cambios |
| Tasa de código | 1/2, 2/3, 3/4 | 1/2, 2/3, 3/4 | Sin cambios |
| Número de subportadoras | 52 | 52 | Sin cambios |
| Duración de símbolo (µs) | 4 | 8 | El doble |
| Intervalo de guarda (µs) | 0.8 | 1.6 | El doble |
| Periodo de FFT (µs) | 3.2 | 6.4 | El doble |
| Preámbulo (µs) | 16 | 32 | El doble |
| Separación de subportadora (MHz) | 0.3125 | 0.15625 | La mitad |

**Tabla 3: Comparativa de los parámetros principales de la capa PHY entre 802.11a y 802.11p**

Una de las características más importantes de 802.11p es el denominado modo OCB (Out of the Context of a BSS). En este modo se desactivan varios procedimientos de gestión como el escaneo de canal, autenticación y asociación, que consumen tiempo y no permiten la comunicación con acceso rápido y baja latencia que requiere un entorno vehicular donde los nodos tienen gran movilidad. De esta forma las estaciones pueden transmitir mensajes de forma directa e inmediata sin los correspondientes retardos debidos al intercambio de información de control.

En la arquitectura C-ITS europea se utiliza una versión de 802.11p llamada ITS-G5, en el cual la capa PHY está adaptada a las bandas de frecuencias designadas para ITS en Europa. La asignación de frecuencias de ITS-G5 se muestra en la Figura 6.

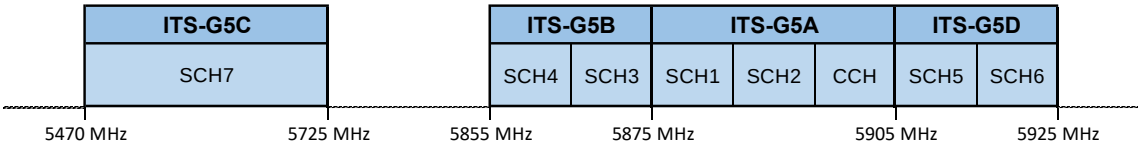


**Figura 6: Distribución de canales de ITS-G5 [30]**

El canal de control CCH y los canales de servicio SCH1 y SCH2 (ITS-G5A) están dedicados a aplicaciones y servicios ITS de seguridad vial, mientras que los canales SCH3 y SCH4 (ITS-G5B) se usan para aplicaciones no relacionadas con la seguridad vial (eficiencia del tráfico, anuncios de servicio, etc.) [31]. La banda ITS-G5D está prevista para aplicaciones futuras. La Comisión Europea amplió en 2020 hasta 5935MHz el espectro dedicado a ITS, reservando los últimos 10MHz para sistemas de ferrocarril urbano, que también tienen prioridad en el canal SCH6, mientras que el SCH5 está priorizado para los sistemas por carretera [32].

En cuanto al canal SCH7 (ITS-G5C) es una banda compartida con BRAN, LAN y WLAN (de 5470 MHz a 5725 MHz) y pueden ser canales de 10MHz o 20MHz. Debido a que comparten el espectro con otros servicios requieren el uso de DFS (Dynamic Frequency Selection) [33], no disponible cuando se trabaja en el modo OCB, y solo son aplicables a comunicaciones V2I con una RSU como maestro DFS. Actualmente no se han concretado casos de uso para esta banda de frecuencias, con lo que no está claro qué aplicaciones podrían usar este canal.

Para el Control de Acceso al Medio (MAC) 802.11p utiliza EDCA (Enhaced Distributed Coordination Access), que está basado en CSMA/CA (Carrier Sense Multiple Access with Collision Advoidance), pero añadiendo atributos de QoS. En CSMA/CA un nodo “escucha” el canal durante un periodo predeterminado de tiempo y si lo percibe como libre puede empezar a transmitir directamente. Si el canal se ocupa durante el periodo de escucha el nodo realiza un procedimiento de *backoff*, en el que tiene que abstenerse de acceder al medio durante un periodo aleatorio de tiempo. EDCA asigna diferentes prioridades de acceso al canal dependiendo de la clase de tráfico.

Cada nodo mantiene 4 colas, denominadas Clases de Acceso (AC: Access Class) con diferentes valores para el tamaño de la Ventana de Contención (CW: Contention Window) y el Espacio Entre Tramas para arbitraje (AIFS: Arbitration InterFrame Space). De esta manera el tráfico con mayor prioridad tiene mayor probabilidad de acceder al canal que el de menor prioridad. [34] Las tramas ITS-G5 tienen la asignación de Clase de Tráfico (TC: Traffic Class) [35] mostrada en la Tabla 4.

| TC | AC | Uso |
|---|---|---|
| 0 | AC_VO | DENM de alta prioridad |
| 1 | AC_VI | DENM |
| 2 | AC_BE | CAM |
| 3 | AC_BK | DENM Multisalto, otro tipo de datos |

**Tabla 4: Clases de tráfico ITS-G5**

Los requisitos que se fijaron para el diseño de 802.11p fueron soportar una velocidad relativa de hasta 200Km/h, tiempos de respuesta cercanos a 100ms y un alcance de comunicación de 1000m. Estos requisitos deberían ser suficientes para soportar las aplicaciones del día 1.

La tecnología ITS-G5 se lleva desarrollando, testando, validando, documentando y perfeccionando durante una década, por lo que se puede considerar una tecnología madura. Se ha utilizado en los proyectos indicados anteriormente como SCOOP@F, Madrid 30 o SISCOGA Extended y Volkswagen la ha incluido en la última generación de su modelo VW Golf [18], el primer modelo de automóvil europeo producido en masa que integra tecnología V2X. Este hecho le ha hecho obtener el galardón Advanced de la Euro NCAP en 2020 [36].

Sin embargo, aunque el desempeño de 802.11/ITS-G5 es adecuado para muchos de los casos de uso del día 1 y día 1,5 se prevé que sea insuficiente para las nuevas aplicaciones que han ido apareciendo (aplicaciones del día 2 y posteriores), por lo que en marzo de 2018 se formó un Grupo de Estudio del IEEE denominado Nueva Generación Vehicular (NGV: Next Generation Vehicular) para trabajar en una nueva enmienda del estándar que recogiera los avances de 802.11 producidos en los últimos años y los aplicara a la comunicación V2X. Se creó así a finales un grupo de trabajo para desarrollar 802.11bd, la evolución de 802.11p con mejoras en la eficiencia espectral, incremento de la fiabilidad y del rango manteniendo compatibilidad hacia atrás con la actual tecnología V2X [37].

Al igual que cuando se diseñó 802.11p se tomó como base 802.11a, se pretende utilizar los avances introducidos en 802.11ac y 802.11ax como base para el desarrollo del nuevo estándar. Entre los requisitos de diseño impuestos están los siguientes [29]:

- Al menos un modo de comunicación que consiga el doble de rendimiento que 802.11p con velocidades relativas de hasta 500Km/h.
- El doble de rango de cobertura que 802.11p en al menos un modo.
- Proporcionar un método de posicionamiento del vehículo.

Para cumplir con estos requisitos se proponen varias mejoras en la capa física como:

- Corrección de errores LDPC (Low Density Parity Check): ofrece incrementos en la eficiencia espectral respecto a BCC (Binary Convolutional Code) usado en 802.11p.
- Modulación 256-QAM: Mejora en rendimiento teórica del 33%.
- MIMO (Multiple-Input/Multiple-Output): el uso de varias antenas de transmisión y recepción permite incrementar la fiabilidad usando diversidad espacial o del rendimiento usando multiplexado espacial [29].
- DCM (Dual Carrier Modulation): Permite rangos de cobertura hasta un 40% mayores mediante la transmisión de los mismos símbolos repetidos en dos subportadoras lo suficientemente separadas, incrementando la ganancia de diversidad.
- Retransmisiones adaptativas: permite mejoras de fiabilidad al enviar varias veces una misma trama. Esta mejora también sería efectiva para los nodos 802.11p que estuvieran en el alcance. La decisión de retransmitir o no, o el número de veces que se retransmitiría dependerían del nivel de congestión.
- MCO (Multi-Channel Operation): Permite operación multicanal concurrente de manera que un canal se use para mensajes de seguridad vial mientras el segundo se use para mensajes no relacionados con la seguridad vial. [37]
- Incremento en el número de subportadoras: 64 portadoras (52 de datos) frente a las 52 (48 de datos) de 802.11p, lo cual permite un incremento de aproximadamente el 8% en la tasa de transferencia.

- Numerología OFDM: permite cambiar el espaciado entre subportadoras, y con ello la duración de símbolo, según el entorno de radio para incrementar la eficiencia de OFDM. Las opciones son 2, 4 u 8 veces *down-clock* con 64, 128 y 256 subportadoras, resultando en una distancia entre subportadoras de 312.5, 156.25 y 78.125 kHz respectivamente. [38]
- *Midambles*: debido las variaciones del canal en un entorno de gran movilidad, la estimación inicial del canal puede volverse incorrecta, por lo que se introducen símbolos conocidos de datos, denominados *midambles*, entre los símbolos de datos OFDM de la trama para actualizar dicha estimación y poder utilizar tasas de datos mayores.
- mmWave: Se está estudiando el uso de las bandas de frecuencia de onda milimétrica de 60GHz, ya utilizadas en estándares como 802.11ad o 802.11ay, para aplicaciones V2X debido a la gran cantidad de ancho de banda disponible, lo cual permitiría el desarrollo de nuevas aplicaciones que requieran gran ancho de banda, pero no requieran de un alcance muy elevado.

| | **IEEE 802.11p** | **IEEE 802.11bd** | **Mejora** |
|---|---|---|---|
| Modulación | BPSK, QPSK, 16-QAM, 64QAM | BPSK, QPSK, 16-QAM, 64-QAM y 256-QAM | 33% mayor rendimiento |
| Corrección de errores | BCC | BCC y LDPC | 2 – 3dB menor sensibilidad -> extensión de rango |
| Ancho de banda del canal | 10MHz ó 20MHz (solo en la banda ITS-G5C) | 10MHz y 20MHz interoperables | Interoperabilidad mejorada |
| Subportadoras de datos | 48 | 48 y 52 | 8% mayor rendimiento |
| MIMO | N/A | MIMO 2x2 | 2x mayor rendimiento |
| Bandas de frecuencia | 5.9GHz | 5.9GHz y 60GHz | Nuevas aplicaciones |
| Retransmisiones adaptativas | N/A | 1 a 3 retransmisiones dependiendo de la congestión del canal | Extensión de rango |
| Localización | N/A | Soportada | Nuevas aplicaciones |
| Modo DCM | N/A | Soportada | 3dB menor sensibilidad -> Extensión de rango |
| Seguimiento del canal | Propietaria | Propietaria y basada en midamble | Menor complejidad del receptor |
| Beneficio total: Extensión del rango | | DCM, retransmisiones y LDPC | Hasta 3 veces mayor rango |
| Beneficio total: rendimiento | | 256 QAM y MIMO | Hasta 3 veces mayor rendimiento |

**Tabla 5: Beneficios de 802.11bd sobre 802.11p [37]**

Uno de los requisitos fundamentales a la hora de diseñar 802.11bd es que todas las inversiones en el desarrollo y despliegue de la actual 802.11p puedan ser aprovechadas, por lo que se han impuesto los siguientes requisitos [39]:

- **Interoperabilidad:** Los dispositivos 802.11p deben de ser capaces de decodificar al menos un modo de transmisión de los dispositivos 802.11bd, y los dispositivos 802.11bd decodificar las transmisiones de 802.11p. Para ello se prevé utilizar un formato de trama que incluya en las tramas 802.11bd el preámbulo de la trama 802.11
- **Co-existencia:** Los dispositivos 802.11p deben ser capaces de detectar las transmisiones de dispositivos 802.11bd, y por tanto abstenerse de transmitir durante las mismas para evitar colisiones (y viceversa).
- **Retro-compatibilidad:** Los dispositivos 802.11bd deben de ser capaces de operar en un modo en el cual puedan interoperar con dispositivos 802.11p.
- **Ecuanimidad:** los dispositivos 802.11p deben de tener las mismas oportunidades de acceder al canal que los dispositivos 802.11bd.

La interoperabilidad y la retro-compatibilidad se prevé conseguir usando el formato de trama mostrado en la Figura 7.

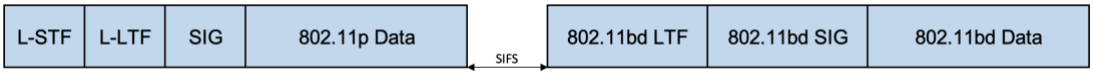


**Figura 7: Formato de trama propuesto para interoperabilidad [29]**

La primera parte de la trama puede ser decodificada tanto por las estaciones 802.11p como por las 802.11bd. Durante la segunda parte las estaciones 802.11p detectarán el canal ocupado y se abstendrán de transmitir. Esta segunda parte, que utilizará las mejoras de 802.11bd como los midambles, LPDC y las mejoras en la modulación, será recibida y decodificada por las estaciones 802.11bd, que además podrán combinar los dos campos de datos para obtener mejor fiabilidad. Esta estructura tiene la ventaja de que no requiere de un cambio en las capas superiores [29].

Otro método propuesto es que cuando la estación 802.11bd detecte una estación 802.11p, lo cual hará con mayor probabilidad antes de ser detectada por la estación 802.11p debido a tener mejor sensibilidad, todos los mensajes se envíen usando el estándar 802.11p. En caso contrario podrá transmitir usando todas las mejoras de 802.11bd.

Cuando una estación 802.11bd envíe tramas en el formato 802.11p deberá ser capaz de notificar que esas tramas vienen de una estación 802.11bd indicándolo a nivel MAC

para evitar que si el resto de las estaciones vecinas también lo son, se sigan enviando tramas en el antiguo formato [40].

La coexistencia, al contrario que la interoperabilidad, solo requiere que las estaciones 802.11p detecten las transmisiones 802.11bd como transmisiones válidas y se abstengan de acceder al canal. Para ello el formato de trama 802.11bd incluirá el preámbulo de 802.11p, como se muestra en la Figura 8.

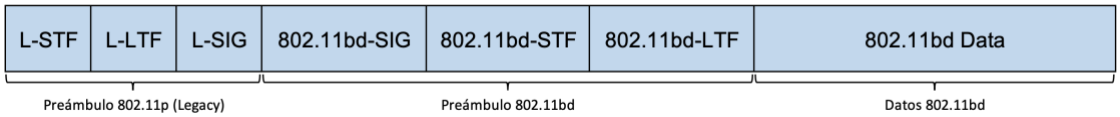


**Figura 8: Formato de trama para coexistencia 802.11p-802.11bd [29]**

A nivel de capa MAC, 802.11bd seguirá utilizando el mismo mecanismo EDCA de acceso al medio para asegurar el acceso en igualdad de condiciones al canal entre los diferentes tipos de estaciones.

Otro de los requisitos impuestos, como se ha mencionado anteriormente, es el posicionamiento respecto a otros usuarios con una precisión de 0.3m y localización del vehículo en lugares sin cobertura de GPS como parkings, con una precisión de 1-2m [39]. Actualmente existe un grupo de trabajo desarrollando una enmienda, la 802.11az, denominada NGP (Next Generation Positioning) que será probablemente el esquema utilizado. 802.11az utiliza FTM (Fine Time Measurement), un protocolo de estimación de distancia (ranging) basado en RTT (Round-Trip Time). 802.11az propone mejoras en FTM como ranging multiusuario, ranging basado en trigger, medidas AoA/AoD (Angle of Arrival/Angle of Departure), etc. [41]

### *2.2.2 Tecnologías basadas en redes celulares*

El 3GPP (3rd Generation Partnership Project) estandarizó en 2016 en su release 14 su versión de tecnología de comunicación V2X basada en redes celulares, conocida como LTE-V2X o C-V2X (Cellular Vehicule to Everything).

El estándar LTE-V2X incluye dos interfaces de radio: El interfaz Uu soporta comunicaciones V2I mientras que el interfaz PC5 soporta comunicaciones V2V utilizando el modo LTE sidelink, o Dispositivo a Dispositivo (D2D). Se definen dos modos de comunicación LTE sidelink: el modo 3 y el modo 4. En el modo 3 es la red celular la que

selecciona y gestiona los recursos de radio utilizados por los vehículos para sus comunicaciones V2V. Sin embargo, en el modo 4 son los propios vehículos los que seleccionan los recursos de radio para sus comunicaciones V2V, lo que permite su funcionamiento fuera de la cobertura de la red celular. Esto hace que sea el modo apropiado para las comunicaciones V2V, ya que las aplicaciones de seguridad no pueden depender de la existencia de cobertura celular. Este modo es una clara alternativa por lo tanto a 802.11p/ETSI-G5 para las comunicaciones V2V. [42]

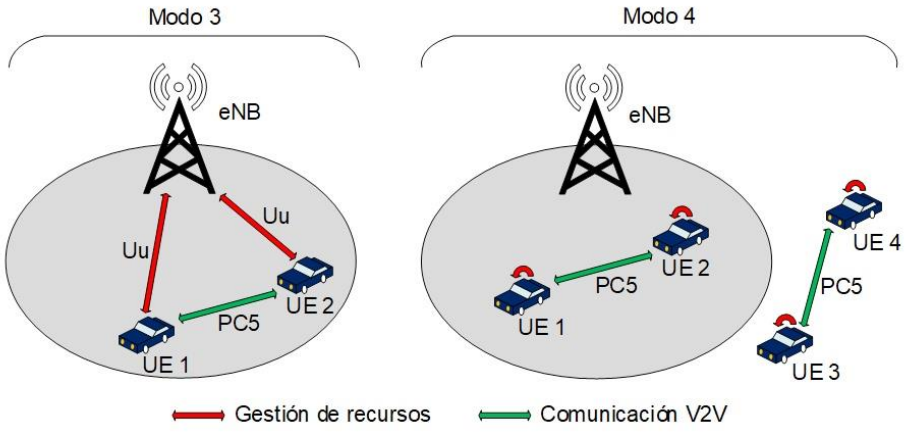


**Figura 9: Modos de comunicación V2V *sidelink* [43]**

La capa física de LTE-V2X utiliza SC-FDMA (Single Carrier-Frecuency Division Multiple Access) y soporta canales de 10MHz y 20MHz. Cada canal está divido en sub-tramas, bloques de recursos RB (Resources Blocks) y subcanales. Las sub-tramas tienen una duración de 1ms, que se corresponde con el TTI (Transmision Time Interval). Los RB son la unidad de frecuencia más pequeña que se puede asignar a un usuario y equivalen a 12 subportadoras de 15KHz (para un ancho total de 180KHz). Los subcanales son grupos de RB en la misma sub-trama. El número de RB por subcanal puede variar, el ETSI ha elegido 10 RBs por subcanal en su estándar [20].

Los datos se transmiten en los TB (Transport Block), empleando el canal PSSCH (Physical Sidelink Shared Channel), y la información de control en los SCI (Sidelink Control Information) a través del canal PSCCH (Physical Sidelink Control Channel). Cuando un nodo envía un TB que contiene un mensaje completo a transmitir (como por ejemplo un mensaje CAM, o un *beacon*), debe de transmitir también su SCI asociado, también referido como SA (Scheduling Assigment). El SCI incluye información como el esquema de modulación y codificación usado para transmitir el TB, los RB que utiliza y el intervalo de reserva de recursos utilizados para la planificación semi-persistente. Esta

información es crítica para que otros nodos pueda recibir y decodificar el TB, por lo que el SCI debe ser correctamente recibido. Un TB y su SCI asociado deben transmitirse siempre en la misma sub-trama. [42]

Se definen dos tipos esquemas de sub-canalización:

- PSCCH+PSSCH adyacentes: el TB y su SCI correspondiente se transmiten en RBs adyacentes, con el SCI ocupando los dos primeros RBs del primer sub-canal utilizado y el TB a continuación, usando varios sub-canales si es necesario. Este esquema es el que ha elegido el ETSI para su uso en V2X en el estándar ETSI EN 303 613. [20]
- PSCCH+PSSCH no adyacentes: Los RBs se dividen en *pools*. Uno de los *pools* está dedicado a transmitir exclusivamente SCIs, que ocuparán 2 RB cada uno, y el otro está reservado para transmitir únicamente TBs y está dividido en sub-canales.

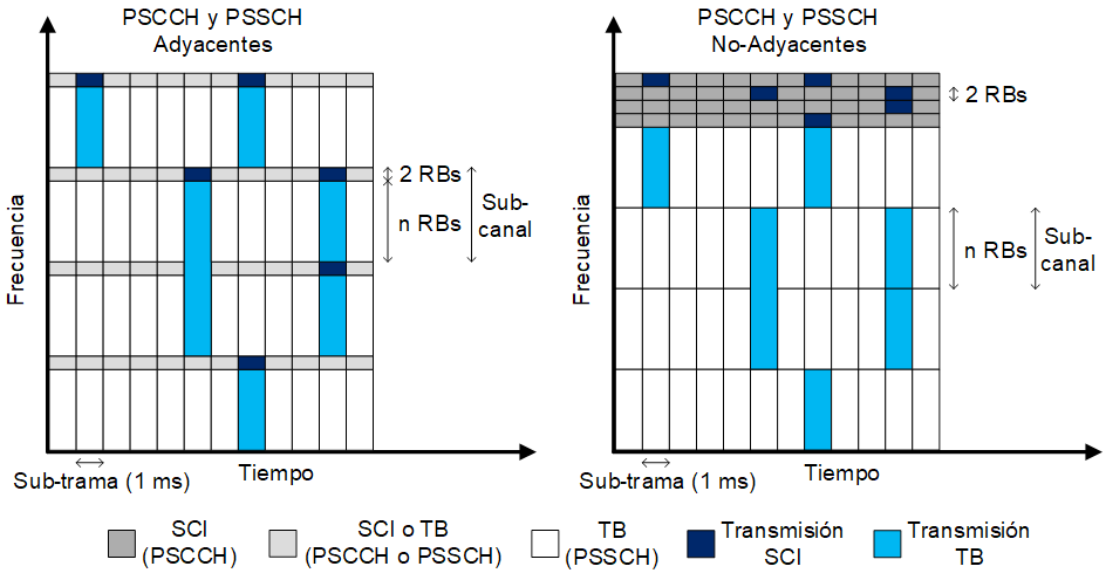


**Figura 10: Esquemas de sub-canalización LTE-V2X [43]**

Los TB se transmiten usando QPSK o 16-QAM, pero los SCI siempre se transmiten usando QPSK. LTE-V2X utiliza turbo coding y prefijo cíclico normal. La sub-portadoras contienen un total de 14 símbolos por sub-trama, de los cuales cuatro están dedicados a DMRS (Demodulation Reference Signals), con el objetivo de combatir el efecto Doppler a altas velocidades. [43]

LTE-V2X define dos modos de comunicación *sidelink*. En el modo 3 la selección de recursos de radio utilizados, los sub-canales a utilizar, es gestionada por la estación base (eNB), por lo que solo está disponible cuando se encuentra bajo cobertura de la red celular. El estándar no define un algoritmo de gestión de recursos, cada operador puede implementar el suyo, que entra dentro de una de las dos siguientes categorías:

- Dynamic Scheduling: Los vehículos solicitan sub-canales para cada transmisión de un paquete. Incrementa la sobrecarga de la señalización en la red celular y retrasa el envío de los paquetes.
- SPS (Semi-Persistent Scheduling): la estación base reserva recursos de forma semi-persistente para las transmisiones periódicas de un vehículo y decide cuanto tiempo se mantiene dicha reserva. Para hacer esta reserva de manera apropiada los vehículos deben de informar a la eNB del tamaño, prioridad y frecuencia de transmisión de los paquetes.

En el modo 4 los vehículos seleccionan sus recursos de red de manera autónoma, independientemente de si se encuentran bajo cobertura de la red celular o no. Cuando los vehículos se encuentran bajo la red celular la estación base decide la configuración del canal V2X e informa a los vehículos de diversos parámetros configurables entre los que se encuentran la frecuencia de portadora del canal V2X, el *pool* de recursos V2X (qué sub-tramas del canal se utilizan para V2X), el esquema de sub-canalización, el número de sub-canales por sub-trama y el número de RBs por sub-trama. Cuando los vehículos no están bajo cobertura celular utilizan un conjunto pre-configurado de parámetros en lugar de los indicados por la estación base. El estándar incluye la opción de dividir el *pool* de recursos V2X en base a áreas geográficas (*Zoning*), de forma que los vehículos en un área geográfica concreta solo puedo utilizar el *pool* de recursos asignados a dicha área. [42]

El mecanismo utilizado por los vehículos para la selección de recursos de radio se denomina Sensing-based Semi-Persistent Scheduling, algoritmo definido en detalle por el 3GPP y que funciona básicamente de la siguiente forma:

En la capa MAC, los recursos (sub-canales) son seleccionados aleatoriamente entre el conjunto recibido por la capa física y se mantiene hasta que un contador denominado Reselection Counter, que se decrementa con cada transmisión y cuyo valor se establece de forma aleatoria entre 5 y 15, llegue a 0. Cuando esto ocurra se mantendrán los mismos

recursos durante otro intervalo aleatorio con una probabilidad P, y volverán a ser seleccionados o reservados en otro caso. [42, 44] P es configurado por cada vehículo y puede tomar cualquier valor entre 0 y 0.8 de manera que valores más altos permiten a los vehículos mantener sus reservas durante más tiempo. Sin embargo, un valor alto de P puede ser contraproducente debido a la movilidad de los vehículos; dos vehículos que reservaron sus recursos cuando estaban fuera de sus respectivos alcances pueden interferirse entre ellos al entrar dentro de dicho alcance y mantener más tiempo dicha interferencia que si el valor de P fuera bajo. [45] También deberá realizarse una nueva reserva cuando el paquete o paquetes a transmitir no encaje en los recursos reservados. [43]

En la capa física, se monitorizan los recursos durante 1 segundo, comparando las medidas de potencia recibida con un umbral dado (dependiente de la prioridad del paquete, establecida por las capas superiores en función de la relevancia y urgencia de la aplicación [42]) y leyendo los mensajes SCI que indican reservas futuras. Entre los recursos que se asuma que no que vayan a utilizar en el próximo intervalo de paquete (o una porción del mismo), el 20% menos interferido son seleccionados y pasado como candidatos a la capa MAC. Si el número de recursos estimados como libres es menos del 20% el umbral se incrementa 3dB y se repite el proceso. [44]

En 2020 ETSI publicó el estándar que integra la capa de acceso de la arquitectura C-ITS usando el interfaz PC5, o *sidelink*, de LTE-V2X [20] y en el cual se definen los parámetros de LTE-V2X que se deben en usar en la Unión Europea.

Como en el caso de 802.11p, los requisitos impuestos para LTE-V2X son suficientes para la mayoría de las aplicaciones de día 1 y día 1.5, pero insuficientes para las de día 2 y posteriores.

El 3GPP publicó en 2020 su Release 16, que incluye el primer estándar V2X basado en la tecnología 5G New Radio (NR). El 5G NR V2X ha sido diseñado para complementar a LTE-V2X. Mientras que LTE-V2X soporta casos de uso de seguridad activa y gestión del tráfico básicos, 5G NR V2X soporta casos de uso más avanzados y mayores niveles de automatización con requisitos más exigentes de ancho de banda, fiabilidad y baja latencia. Algunos de estos casos de uso incluyen las caravanas de vehículos, el intercambio de datos de sensores entre vehículos, RSUs y peatones, la conducción remota y la conducción autónoma. [46]

De manera análoga a LTE-V2X, NR V2X define dos modos de comunicación *sidelink*. En el modo 1 (controlado), equivalente al modo 3 de LTE-V2X, es la estación base (denominada gNodeB) la que organiza los recursos *sidelink* usando la interfaz Uu. Dicho modo solo es posible bajo la cobertura de la red celular. En el modo 2 (autónomo) los vehículos determinan de manera autónoma el conjunto de recursos a utilizar entre aquellos (pre-)configurados por la gNodeB/eNodeB. Este modo puede ser utilizado bajo condiciones de cobertura o fuera de cobertura de forma similar al modo 4 de LTE-V2X. El modo 2 y modo 4 tienen, sin embargo, varias diferencias en el esquema de planificación. [47]

Mientras que el modo 4 de LTE-V2X está diseñado principalmente para el tráfico periódico el modo 2 de NR V2X introduce ajustes enfocados a satisfacer la necesidad de acomodar también tráfico aperiódico, como los DEMNs generados asíncronamente ante la detección de eventos de peligro (como un frenado de emergencia) o incluso CAMs, los cuales no son necesariamente periódicos, ya que su generación depende de la movilidad del vehículo que los transmite. [47]

Las especificaciones de NR V2X definen dos bandas de frecuencias, FR1 para la banda por debajo de los 6GHz y FR2 para la banda por encima de los 6GHz entre 28-52GHz, considerada banda de onda milimétrica (mmWave), que puede cubrir rangos pequeños con alto rendimiento.

La longitud de trama de radio NR V2X es de 10ms, conteniendo 10 subramas en el dominio del tiempo y 12 subportadoras en el dominio de la frecuencia. Los recursos están distribuidos en un espacio bidimensional de tiempo y frecuencia llamado *pool* de recursos, como en el caso de LTE-V2X.

La información de control, al contrario que en LTE-V2X, está dividida en 2 etapas, denominadas stage-I SCI y stage-II SCI. El stage-I SCI contiene la información que es útil para notificar a los vehículos vecinos sobre los recursos reservados y ocupados por la UE transmisora. El stage-II SCI contiene información útil para la decodificación del TB. [48] La transmisión en 2 etapas del SCI simplifica su decodificación; los vehículos que sondeen el canal para conocer la ocupación de recursos solo necesitan decodificar el stage-I SCI. [47]

La capa física de NR V2X utiliza CP-OFDM (Cyclic Prefix Orthogonal Frequency-Division Multiplexing), la cual tiene mayor eficiencia espectral, mayor rendimiento, menor complejidad de implementación y mejor compatibilidad con tecnologías multi-antena. CP-OFDM está bien localizada en el dominio del tiempo, lo cual es importante para aplicaciones donde la latencia es crítica y para implementaciones TDD (Time-Division Duplex). También es más robusta frente a ruido de fase del oscilador y desviaciones de frecuencia Doppler que otras formas de onda multi-portadora, lo cual es crucial para mmWaves y V2X. Los turbo códigos de LTE son reemplazados por códigos LDPC (Low-Density Parity-Check) en NR V2X.

En la Release 15, se ha añadido 64-QAM a las modulaciones QPSK y 16-QAM de LTE-V2X, mientras que NR V2X soporta también 256-QAM con código binario reflejado Gray. Los nuevos formatos de modulación pueden ser utilizados cuando las condiciones del canal son particularmente favorables, proporcionando ganancia en términos de eficiencia espectral y rendimiento a expensas de rango de cobertura. [47]

Como en el caso de 802.11bd, NR V2X hace uso de numerología, que juega un papel clave a la hora de mantener la QoS requerida. Soporta un espaciado entre subportadoras flexible de 15, 30, 60, 120 y 240 KHz en cada subtrama de 1ms. Cada subtrama tiene un número diferente de slot dependiendo de la numerología, y cada slot contiene 14 símbolos OFDM, como puede verse en la Figura 11 [48]. Un espaciado mayor entre subportadoras implica una duración de slot menor, reduciendo la latencia, ideal para aplicaciones de seguridad críticas, y ofreciendo mejor resistencia al efecto Doppler a altas velocidades vehiculares [47].

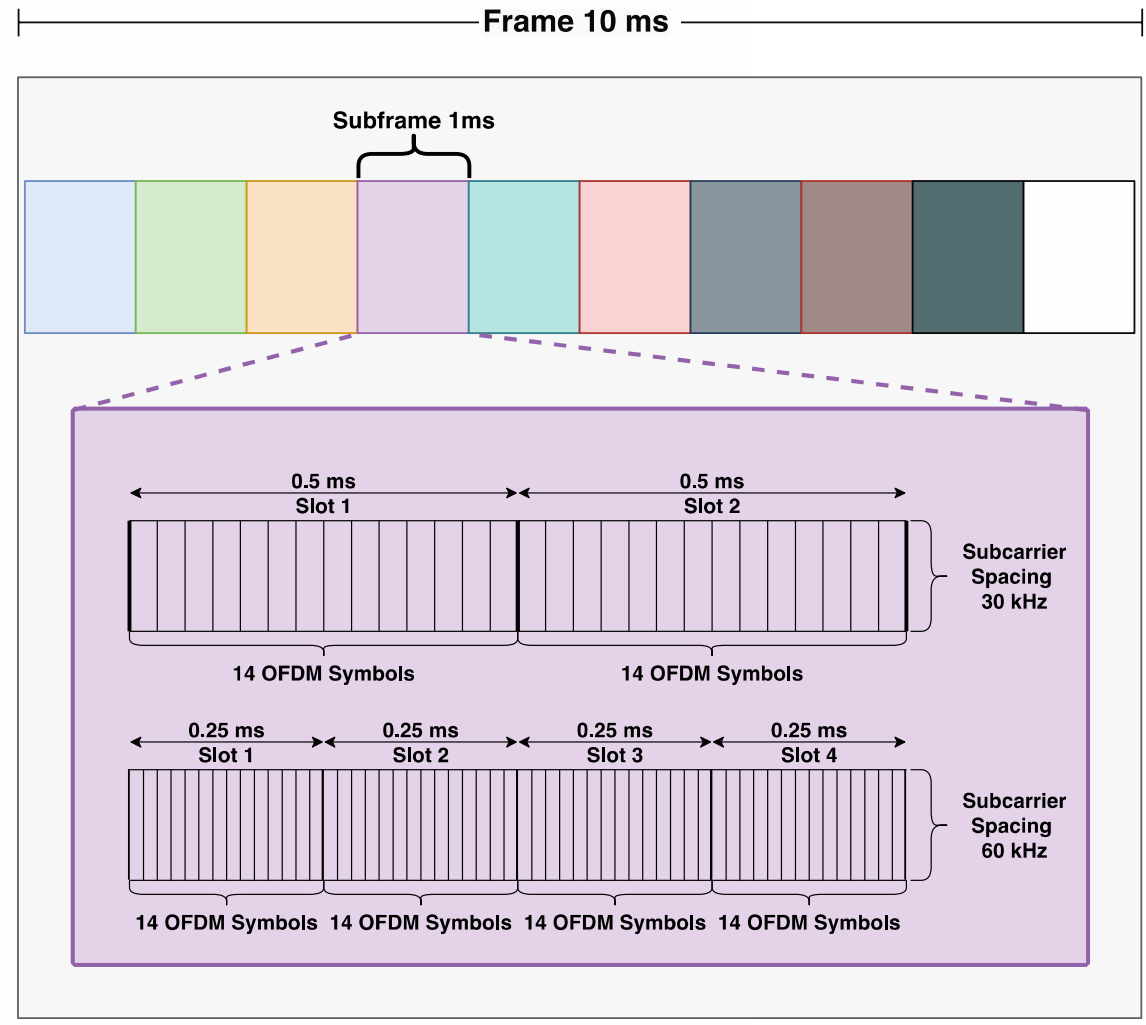


**Figura 11: Ejemplo de numerologías y su correspondiente duración de slot [48]**

NR V2X añade soporte de *unicast*, *broadcast* y *groupcast*. Algunas aplicaciones permiten incluso el uso de varios modos simultáneos, como *platooning*, donde el líder de la caravana se puede comunicar con el resto de miembros de la misma mediante *groupcast* pero enviar mensajes, como CAMs o DEMNs, a otros usuarios no miembros usando *broadcast* como se puede ver en la Figura 12. [29]

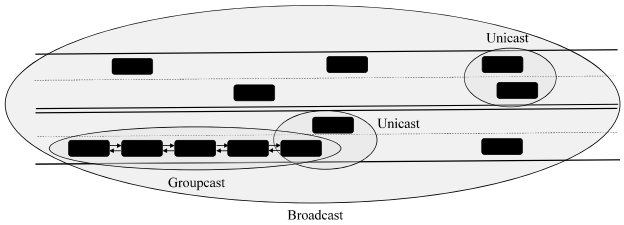


**Figura 12: Tipos de comunicación NR V2X [29]**

Un resumen de las diferencias entre LTE-V2X y NR V2X se puede ver en la Tabla 6:

| Aspecto | LTE-V2X Sidelink | NR V2X Sidelink |
|---|---|---|
| Forma de onda | SC-FDMA | CP-OFDM |
| Codificación del canal | Turbo | LDPC |
| Modulación | QSPK, 16-QAM, 64-QAM | QPSK, 16-QAM, 64-QAM y 256-QAM |
| Numerología | Fija: subtramas de 1ms | Flexible: subtramas de 0.25ms, 0.5ms ó 1 ms |
| Señales de referencia DMRS | 4 símbolos fijos en 1 ms | Varios patrones, entre 12 y 24 subportadoras |
| SCI | 2 PRBs, adyacentes o no adyacentes | SCI en dos etapas dentro del slot de transmisión |
| Feedback | - | ACK y NACK posibles |
| Tipo de transmisión | Broadcast | Unicast, Groupcast y Broadcast |
| Modo controlado | Modo 3 | Modo 1 |
| Modo autónomo | Modo 4 | Modo 2 |
| HARQ | 1 retransmisión ciega | Hasta 32 retransmisiones, ciegas o basadas en feedback |
| Ventana de detección | Opción única: 1s | Dos opciones: 0.1s y 1,1s |
| Porcentaje de CCSRs entregados de la capa PHY a MAC en modo autónomo | 20% | Un mínimo de 20%, 35% ó 50% |
| Re-evaluación y re-selección de recursos | No posible | Posible |
| Pre-emption | No posible | Posible |

**Tabla 6: Comparación entre LTE-V2X y NR V2X (Sidelink) a 5.9GHz [47]**

## 2.3 Software de comunicación V2X

En los Trabajos de Fin de Grado realizados por Pilar Sánchez Martín [9] y Alejandro Lobo González [10] se realizó el estudio e implementación de RSU y OBU. Dicho trabajo fue ampliado por el Trabajo Fin de Máster de Roberto Herreras Babón [11]. En dichos trabajos se buscaron alternativas de software de comunicación V2X de código abierto que implementaran la pila de protocolos de la arquitectura ETSI C-ITS y se encontraron 3 posibles alternativas: OpenC2X [12], Vanetza [49] y GeoNetworking [50].

Ninguna de las tres alternativas implementaba la pila completa de protocolos, pero se decidió en su momento que OpenC2X era la que mejor se adaptaba a los proyectos y pruebas a desarrollar en dichos Trabajos de Fin de Estudio.

OpenC2X implementa los mensajes CAM y DENM, la base de datos LDM, el servicio de posicionamiento mediante GPS y lectura de datos del vehículo mediante OBD-2, así como el control de congestión descentralizado (DCC). Sin embargo, no implementa una de las partes fundamentales de la arquitectura: la capa de red y transporte ITS. Los protocolos principales de esta capa son el BTP y GeoNetworking, que se encargan del

multiplexado de los mensajes de la capa de *facilities* y del enrutado de los paquetes respectivamente.

Actualmente los servicios de la capa *facilites* enviaban sus mensajes al DCC y éste los trataba de manera independiente, rellenando las cabeceras correspondientes a los protocolos BTP y GeoNetworking de manera manual con valores fijos y realizando el procesamiento de los paquetes entrantes de la interfaz de red descartando todos los campos excepto los campos Header type y Header subtype, que indican el tipo de servicio de la capa *facilities* al que corresponde el mensaje, y los enviaban al servicio CA o DEN según correspondiera.

Por ello era necesario, si queríamos tener una estación ITS completa conforme a la arquitectura ETSI, implementar la capa de red y transporte.

Pero antes se decidió, ya que los trabajos de Pilar, Alejandro y Roberto tienen ya cierto tiempo, revisar si había alguna novedad en los softwares analizados. Tanto Geonetworking como OpenC2X no se han actualizado desde entonces, pero Vanetza se sigue manteniendo actualmente y va recibiendo mejoras y actualizaciones. Actualmente Vanetza implementa BTP, GeoNetworking, la capa transversal de seguridad y DCC. Asimismo, ofrece soporte para mensajes basados en ASN.1 como CAM y DENM.

Por ello se decidió instalarlo para ver si era factible, en vez de desarrollar los protocolos faltantes en OpenC2X, montar las estaciones existentes con Vanetza. Sin embargo, al contrario que OpenC2X, Vanetza implementa una serie de librerías para crear nuestra propia aplicación V2X sobre ella, pero no es una aplicación como tal. Aunque incluye una pequeña aplicación, llamada Socktap, ésta es muy básica y solo permite el envío y recepción de CAMs periódicos y de mensajes Beacon, que veremos en más detalle en el próximo capítulo.

Por ello se decidió que, en vez de realizar una nueva aplicación sobre Vanetza, se implementarían los protocolos BTP y GeoNetworking sobre OpenC2X. La implementación de BTP ha sido realizada por Daniel Monje González en su TFG [13]. Dicho trabajo ha sido muy útil para entender como OpenC2X realiza el intercambio de la información entre los diferentes servicios y las limitaciones de la implementación del DCC del mismo. En el presente Trabajo Fin de Máster se realizará la implementación de GeoNetworking.

Sin embargo, también se ha aprovechado la instalación de Vanetza para hacer pruebas de comunicación entre ambos y así comprobar la interoperabilidad entre dichos softwares, lo que nos proporciona una buena visión del cumplimiento de los estándares de ambos.

Todos los detalles sobre la implementación de GeoNetworking y la instalación de Vanezta los veremos en los siguientes capítulos.

# 3

# El protocolo GeoNetworking

El protocolo GeoNetworking es un protocolo de capa de red que proporciona enrutamiento de paquetes en una red *ad hoc* haciendo uso de las posiciones geográficas para el transporte de paquetes. Soporta la comunicación entre estaciones ITS individuales, así como la distribución de paquetes en zonas geográficas.

GeoNetworking puede ejecutarse sobre diferentes tecnologías de acceso inalámbricas de corto acceso como ITS-G5 o LTE-V2X. Para ello se han divido la especificación del estándar entre la parte de funcionalidades independientes del medio [51], que son comunes a todas las tecnologías de acceso y la parte de funcionalidades dependientes del medio, que extienden la funcionalidad del protocolo para una tecnología inalámbrica concreta, como ITS-G5 [35] y LTE-V2X [21].

GeoNetworking apropiado para nodos altamente móviles y cambios frecuentes en la topología de la red. Además, su flexibilidad soporta los requisitos de aplicaciones heterogéneas incluyendo aplicaciones para seguridad vial, eficiencia del tráfico e infotainment. Permite, de manera más específica, la transmisión periódica de mensajes de estado de seguridad a alta tasa, diseminación multisalto rápida de paquetes en regiones geográficas para alertas de emergencia y el transporte de paquetes unicast para aplicaciones de Internet.

## 3.1 Esquemas de reenvío del enrutamiento geográfico

El protocolo GeoNetworking proporciona básicamente dos funciones básicamente acopladas: direccionamiento geográfico y reenvío (forwarding) geográfico. GeoNetworking puede enviar paquetes de datos a un nodo por su posición o a múltiples nodos dentro de un área geográfica. Para reenvío, GeoNetworking asume que cada nodo tiene una vista parcial de la topología de red en sus alrededores y que cada paquete lleva

una dirección geográfica, tal como la posición o área geográfica como destino. Cuando un nodo recibe un paquete de datos compara la geo-dirección en el paquete de datos y la vista del nodo de la topología de red y realiza una decisión autónoma de reenvío. Como resultado los paquetes son reenviados “al vuelo”, sin necesidad de configuración y mantenimiento de tablas de enrutamiento en los nodos.

El método más innovador para la transmisión de datos que habilita el enrutamiento geográfico es dirigir paquetes de datos a áreas geográficas concretas. De esta manera un vehículo puede seleccionar y especificar un área geográfica bien delimitada en la cual deben de ser entregados los paquetes. Los vehículos intermedios tienen la función de enviar la información y únicamente los vehículos que se encuentren dentro del área de destino procesan el mensaje y lo envían a la aplicación correspondiente. De esta manera se consigue que únicamente los vehículos que realmente estén afectados por una situación de peligro o una notificación de tráfico sean notificados, mientras que los vehículos no afectados no serán notificados. [52]

Un ejemplo de este tipo de transmisión de datos sería el caso de un vehículo de emergencias que se acerca a una intersección y no va a respetar el semáforo y/o la prioridad que le corresponde en dicha intersección. De esta manera puede enviar un mensaje y que solo los vehículos en un área definida en torno a dicha intersección lo procesen y avisen al conductor de dicha eventualidad. Con casi total seguridad dicho área geográfica no se cubra totalmente por la cobertura de la señal de radio del vehículo, por lo que habrá vehículos intermedios, en la dirección a dicho área, que reenviarán el paquete.

GeoNetworking define los siguientes esquemas de reenvío:

### *3.1.1 GeoUnicast*

Este esquema define una comunicación punto a punto entre dos estaciones ITS. Para ello la estación de origen determina la posición del destino y después envía el paquete a un nodo en dirección al destino, quien reenvía el paquete a lo largo del camino usando los nodos intermedios necesarios hasta que el paquete alcanza su destino.

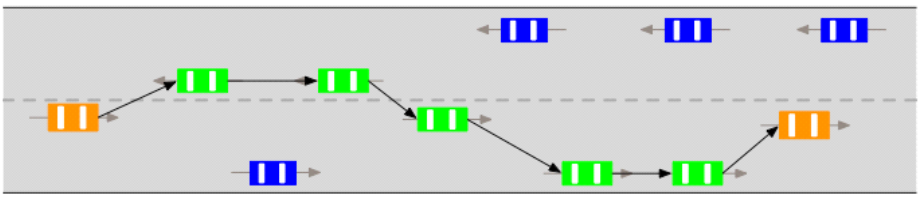

**Figura 13: GeoUnicast [52]**

### 3.1.2 GeoBroadcast

En este esquema un paquete es reenviado salto a salto hasta que alcanza el área de destino determinado por el paquete, y los nodos redifunden (rebroadcast) el paquete si están dentro del área de destino. Este es el esquema que se utiliza para los mensajes DENM. GeoAnycast es similar, pero se diferencia de GeoBroadcast en que un nodo dentro del área de destino no redifundirá ningún paquete recibido.

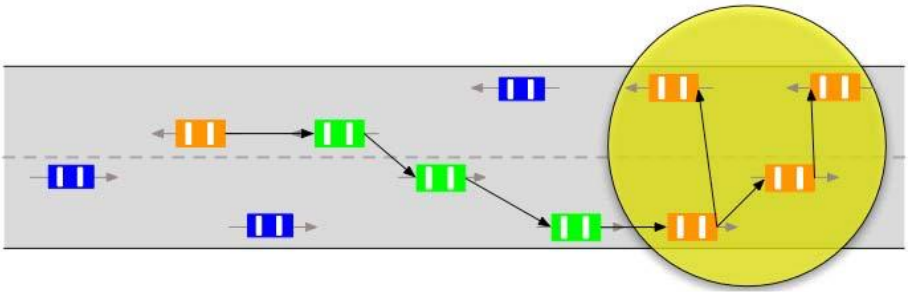

**Figura 14: GeoBroadcast [52]**

### 3.1.3 Topologically-scoped broadcast

En este esquema un paquete es difundido desde un nodo a todos los vecinos que se encuentren a n saltos (n-hop neighbourhood). Se define el caso especial de broadcast de salto único (single-hop broadcast), que se usa para enviar paquetes únicamente a los nodos vecinos a un salto de distancia y que es el utilizado para el envío de mensajes CAM.

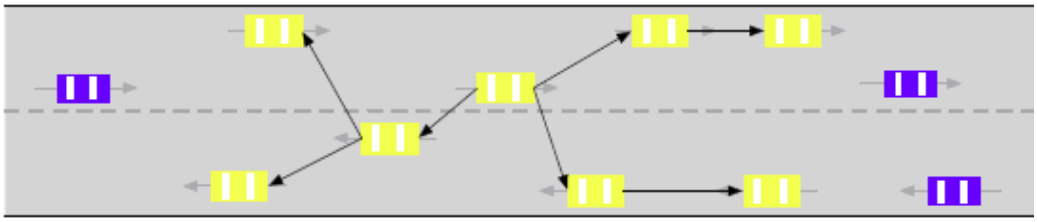

**Figura 15: Topologically-Scooped Broadcast [52]**

## 3.2 Capa de Red y Transporte ITS

Como ya se introdujo en el capítulo 1, el protocolo GeoNetworking se encuentra en la capa de red y transporte de la pila de protocolos ITS, la cual se compone de varios protocolos como se puede ver en la Figura 16:

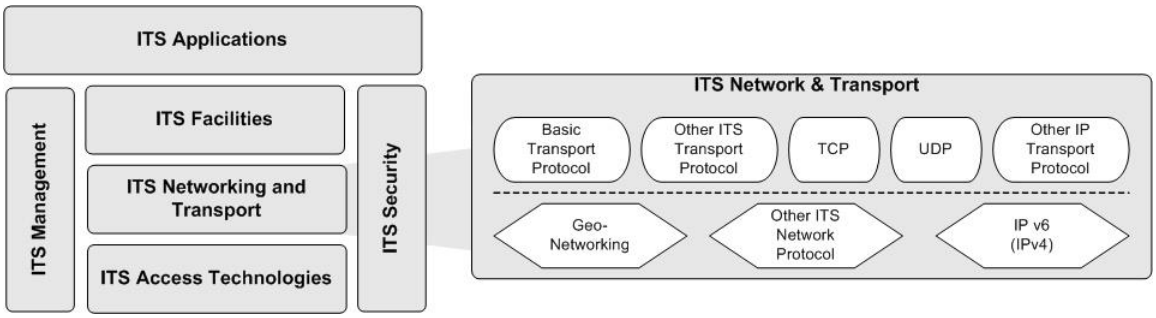


**Figura 16: Capa de Red y Transporte ITS [5]**

Los protocolos que una estación ITS puede utilizar incluyen TCP, UDP e IPv6 pero el ETSI ha creado dos protocolos específicos para su uso en C-ITS: BTP y GeoNetworking. A la entidad que implementa el protocolo GeoNetworking se le denomina GeoAdhoc router.

BTP (Basic Transport Protocol) es un protocolo sencillo que proporciona un servicio de transporte extremo a extremo no orientado a conexión en redes *ad hoc* para sistemas ITS. Su función principal es la multiplexación de los mensajes provenientes de la capa de *facilities*, como los CAM o DEMN, para su envío mediante el protocolo GeoNetworking y la demultiplexación en el destino y entrega a su servicio correspondiente. Para ello utiliza, de forma similar a TCP o UDP, números de puertos bien conocidos, *well-know-ports*, que identifican a los diferentes servicios. También permite a las entidades de la capa de *facilities* acceder a los servicios de GeoNetworking y el envío de información de control a dicho protocolo. [13]

El protocolo GeoNetworking proporciona el enrutamiento de los paquetes y puede ensamblarse con BTP o con IPv6 para formar la pila GeoNetworking como se puede ver en la Figura 17:

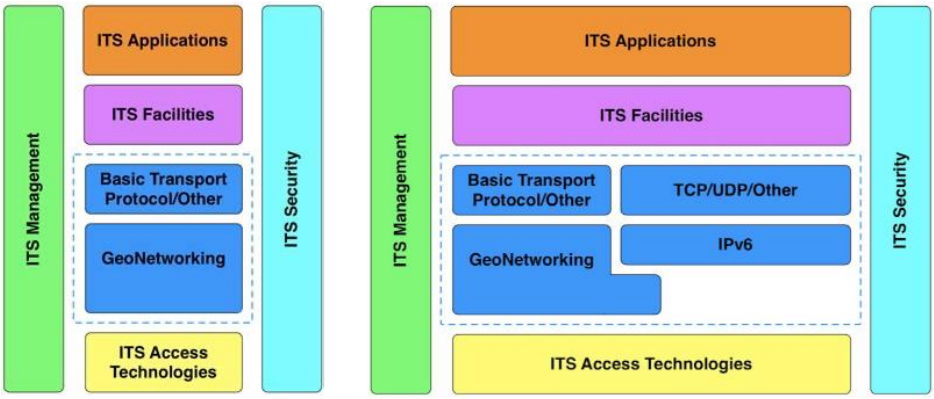


**Figura 17: Pila de protocolos GeoNetworking [5]**

## 3.3 Servicios proporcionados por el protocolo GeoNetworking

Como ya se ha indicado, el protocolo GeoNetworking reside en la capa de red y transporte de la arquitectura ITS y proporciona el transporte de paquetes en la red *ad hoc* ITS, dando servicio a entidades de protocolo superiores (Protocolo de Transporte ITS) como el protocolo BTP, o la Subcapa de Adaptación de GeoNetworking a IPv6 (GN6ASL), especificada en [53] . Dichos servicios se proporcionan vía el GN_SAP usando primitivas de servicios de diferentes tipos que transportan parámetros y la PDU de la entidad de protocolo superior (T/GN6 PDU), como se puede ver en la Figura 18. Una PDU de los protocolos de transporte es considerado como una SDU por GeoNetworking. Dicha SDU se completa con la Información de Control de Protocolo (PCI) y se transmite como una GN PDU usando los servicios de la capa de Acceso ITS.

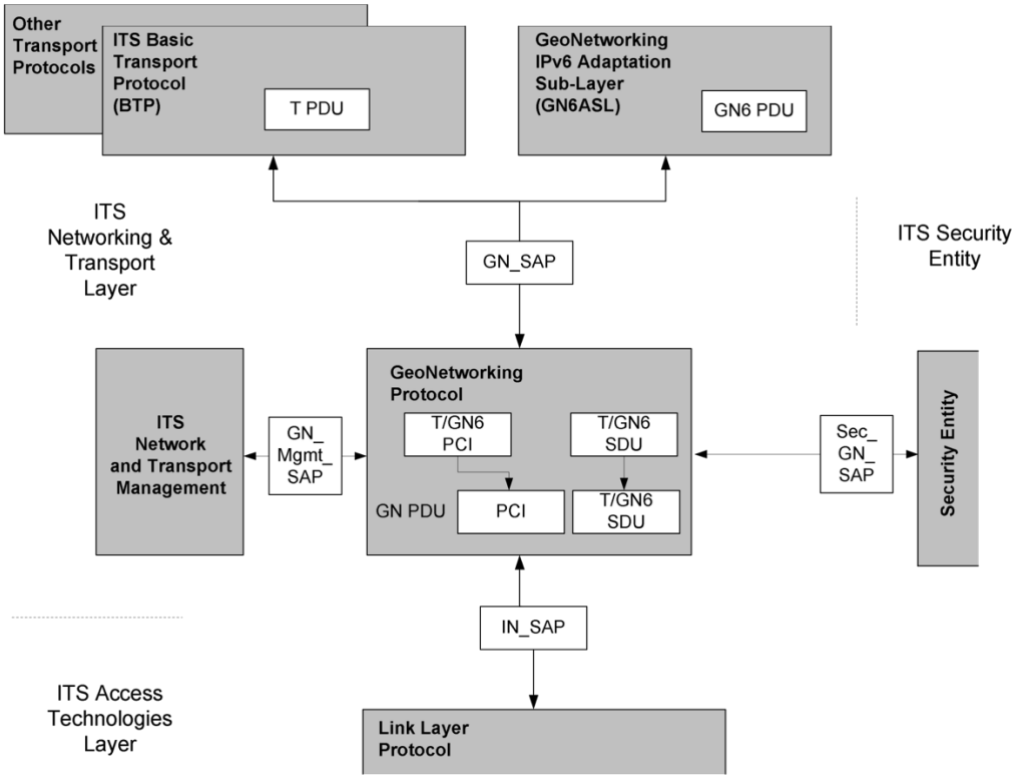


**Figura 18: Puntos de acceso, SDUs y PDUs relevantes del protocolo GeoNetworking [51]**

## 3.4 Seguridad y privacidad

GeoNetworking debe soportar los mecanismos de seguridad definidos por las configuraciones de seguridad indicados en [54], incluyendo protección criptografíca mediante firmas digitales, encriptación, checks de consistencia y checks de plausibilidad.

Para una implementación particular de ITS, todos los GeoAdhoc router deberán implementar las mismas medidas mínimas de seguridad. La aplicación de protección criptográfica se controla mediante la constante del protocolo GN *itsGnSecurity*.

OpenC2X no implementa la entidad transversal de seguridad, por lo que en nuestra implementación del protocolo no se va a implementar ninguno de los mecanismos indicados.

## 3.5 Direccionamiento GeoNetworking

Cada GeoAdhoc router debe de tener al menos una dirección única en su capa de red y transporte, denominada dirección GeoNetworking, la cual debe ser utilizada en la cabecera de los paquetes GeoNetworking e identifica a las entidades que se comunican.

El formato de la dirección GeoNetworking se muestra en la Figura 19, y los campos se detallan en la Figura 20.

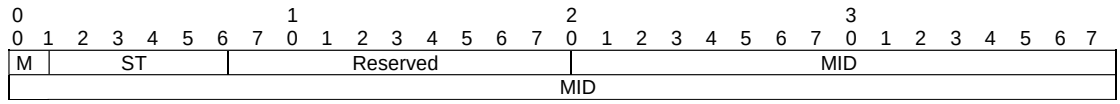


**Figura 19: Formato de la dirección GeoNetworking [51]**

| Field # | Field name | Octet/bit position | | Type | Description |
|---|---|---|---|---|---|
| | | First | Last | | |
| 1 | M | Octet 0<br>Bit 0 | Octet 0<br>Bit 0 | 1 bit unsigned integer | This bit allows distinguishing between manually configured network address (clause 10.2.1.3.3) (update) and the initial GeoNetworking address (clause 10.2.1.3.2). M is set to 1 if the address is manually configured otherwise it equals 0. |
| 2 | ST | Octet 0<br>Bit 1 | Octet 0<br>Bit 5 | 5 bit unsigned integer | ITS-S type<br>To identify the ITS-S type:<br>0 - Unknown<br>1 - Pedestrian<br>2 - Cyclist<br>3 - Moped<br>4 - Motorcycle<br>5 - Passenger Car<br>6 - Bus<br>7 - Light Truck<br>8 - Heavy Truck<br>9 - Trailer<br>10 - Special Vehicle<br>11 - Tram<br>15 - Road Side Unit<br>(see note). |
| 4 | Reserved | Octet 0<br>Bit 6 | Octet 1<br>Bit 7 | 10 bit unsigned integer | Reserved |
| 5 | MID | Octet 2 | Octet 7 | 48 bit address | Represents the LL_ADDR. |
| NOTE: The values of the ITS-S type are aligned with ETSI TS 102 894-2 [11]. | | | | | |

**Figura 20: Campos de la dirección GeoNetworking [51]**

El campo MID (MAC ID) se corresponde a la dirección de la capa de acceso. En el caso de ITS-G5, que es lo que se va a implementar en este TFM, se usa la dirección MAC de 48-bit [35].

El estándar define 3 modos de configuración de la dirección GeoNetworking, definido en la constante de protocolo *itsGnLocalAddrConfMethod*:

- Configuración automática: El GeoAdhoc router asigna en el arranque el campo MID de la dirección GeoNetworking tomándola de la constante de protocolo *itsGnLocalGnAddr*. Este valor, según el estándar, será dependiente de la implementación y no podrá ser cambiado a no ser que la se cambie el valor de *itsGnLocalAddrConfMethod*.

- Configuración gestionada (*managed*): Será la entidad de gestión de la capa de Red y Transporte la encargada de fijar el valor adecuado de la dirección GeoNetworking.

- Configuración anónima: En este caso será la entidad de seguridad la encargada de configurar y mantener el valor de la dirección GeoNetworking.

A pesar de lo que pueda indicar el campo M de la dirección GeoNetworking, este no indica el modo de configuración de la dirección, sino si este es el configurado en el arranque (usando cualquiera de los tres modos indicados) o si ha sido actualizado posteriormente en el caso de la configuración gestionada o anónima.

## 3.6 Vectores de posición

Dado que el protocolo GeoNetworking ofrece funciones de encaminamiento de paquetes mediante el uso de posiciones geográficas, es necesario el intercambio de información sobre las mismas entre los nodos. El protocolo agrupa dicha información en los denominados Vectores de Posición. Se definen dos tipos de vectores de posición.

### *3.6.1 Long Position Vector (LPV)*

Es una estructura de 24 octetos con el formato mostrado en las Figura 21 y Figura 22. Agrupa la posición geográfica (longitud y latitud) de la estación ITS, su velocidad, rumbo (heading) y un indicador de la precisión de la posición (PAI). Almacena también la dirección GeoNetworking de la estación, para identificarla, y un sello temporal.

El timestamp es un número de 32 bits e indica, en milisegundos, cuando se ha obtenido la posición de la estación, codificado como se muestra en la Fórmula 1:

$$TST = TST(TAI) mod 2^{32}$$

**Fórmula 1: Timestamp**

El TST(TAI) indica el número de milisegundos TAI (Tiempo Atómico Internacional) transcurridos desde las 00:00:00.000 UTC del 01 de enero de 2004. Con este esquema de codificación se produce una vuelta a 0 cada 49,71 días. [51]

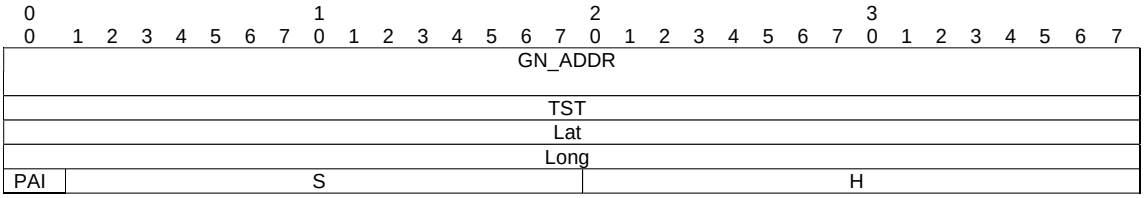


**Figura 21: Long Position Vector [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | GN_ADDR | Octet 0 | Octet 7 | 64 bit address | n/a | Network address for the GeoAdhoc router entity in the ITS-S |
| 2 | TST | Octet 8 | Octet 11 | 32 bit unsigned integer | [ms] | Expresses the time in milliseconds at which the latitude and longitude of the ITS-S were acquired by the GeoAdhoc router. The time is encoded as: $TST = TST(TAI) \bmod 2^{32}$ where TST(TAI) is the number of elapsed TAI milliseconds since 2004-01-01 00:00:00.000 UTC |
| 3 | Lat | Octet 12 | Octet 15 | 32 bit signed integer | [1/10 micro-degree] | WGS 84 [i.6] latitude and longitude of the GeoAdhoc router reference position expressed in 1/10 micro degree |
| 4 | Long | Octet 16 | Octet 19 | 32 bit signed integer | [1/10 micro-degree] | |
| 5 | PAI | Octet 20 Bit 0 | Octet 20 Bit 0 | 1 bit unsigned integer | n/a | Position accuracy indicator of the GeoAdhoc router reference position<br>Set to 1 (i.e. True) if the semiMajorConfidence of the PosConfidenceEllipse as specified in ETSI TS 102 894-2 [11] is smaller than the GN protocol constant `itsGnPaiInterval` / 2<br>Set to 0 (i.e. False) otherwise |
| 6 | S | Octet 20 Bit 1 | Octet 21 | 15 bit signed integer | [1/100 m/s] | Speed of the GeoAdhoc router expressed in signed units of 0,01 metre per second |
| 7 | H | Octet 22 | Octet 23 | 16 bit unsigned integer | [1/10 degrees] | Heading of the GeoAdhoc router, expressed in unsigned units of 0,1 degree from North |

**Figura 22: Campos del LPV [51]**

## *3.6.2 Short Position Vector (SPV)*

Es una estructura de 20 octetos en los que se prescinde de alguno de los campos del LPV, como se puede ver en la Figura 23.

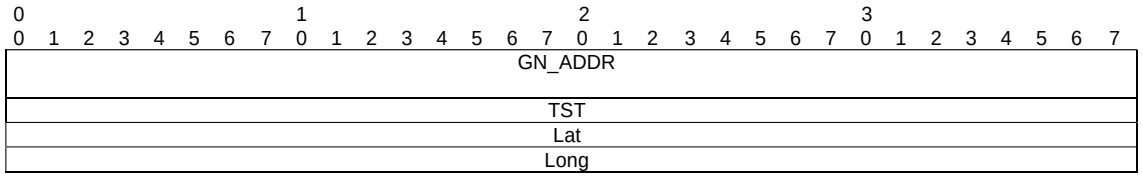


**Figura 23: Short Position Vector [51]**

Los campos son equivalentes a los definidos en la Figura 22.

## 3.7 Estructuras de datos

Para su funcionamiento correcto, el protocolo GeoNetworking necesita mantener varias estructuras de datos, como la Location Table, el Ego Position Vector, Sequence number, el Location Service Packet Buffer y el Forwarding Packet Buffer.

### *3.7.1 Location Table (LocT)*

Es una estructura local de datos que debe almacenar cualquier GeoAdhoc router y que almacena información sobre otras estaciones ITS que ejecutan el protocolo GeoNetworking.

Cada entrada de la tabla, denominada Location Table Entry (LocTE), debe almacenar al menos los siguientes elementos:

- La dirección GeoNetworking de la estación ITS: *GN_ADDR.*
- La dirección de la capa de acceso de la estación ITS: *LL_ADDR*.
- Tipo de estación ITS (vehículo, *roadside unit*, etc.)
- La versión del protocolo GeoNetworking usada por la estación ITS.
- Vector de posición de la estación ITS, compuesto por:
    - Posición geográfica *POS(GN_ADDR)*;
    - Velocidad *S(GN_ADDR)*;
    - Heading *H(GN_ADDR)*;
    - Sello temporal de la posición geográfica *TST(POS, GN_ADDR)*;
    - Indicador de exactitud de la posición *PAI(POS, GN_ADDR)*;
- *Flag LS_PENDING(GN_ADDR)*: *flag* que indica que el Servicio de Ubicación (LS) está en progreso.
- *Flag IS_NEIGHBOURG(GN_ADDR)*: *flag* que indica que el GeoAdhoc router está en un rango de comunicación directo, es decir, es un vecino.

- *DPL(GN_ADDR)*: lista de paquetes duplicada para la *GN_ADDR* de origen.
- Tasa de datos de paquete *PDR(GN_ADDR)* expresada como la Media Móvil Exponencial (EMA).

En el caso de utilizar ITS-G5 el estándar añade los siguientes campos a cada LocTE, denominadas Location Table Entry Extensions for ITS-G5 (LocTEX-G5), para vecinos GeoNetworking en interfaces ITS-G5 [35]:

- Sello temporal (*local to ego station*) de la última actualización de la LocTEX-G5, *TST_G5(GN_ADDR)*.
- Sello temporal en el Vector de Posición de la cabecera del paquete SHB, *TST_SO_PV_G5(GN_ADDR)*.
- Potencia transmitida del paquete que actualizó la entrada LocTEX-G5, *TX_POWER_G5(GN_ADDR)*.
- RSSI del paquete que actualizó la entrada LocTEX-G5, *RSSI_G5(GN_ADDR)*.
- CBR local diseminado recibido de *GN_ADDR*, *CBR_R_0_HOP(GN_ADDR)*.
- CBR de un salto diseminado.

Las entradas de la LocT se añaden con un tiempo de vida *T(LocTE)*, que se toma del valor de la constante de protocolo *itsGnLifetimeLocTE*, y se eliminan cuando dicho tiempo de vida expira.

El *flag LS_PENDING(GN_ADDR)* debe de ser puesto a 0 cuando el *flag* no es renovado dentro del tiempo de vida 3 x *itsGnBeaconServiceRetransmitTimer*.

### *3.7.2 Ego Position Vector (EPV)*

El Vector de Posición Propio es una estructura de datos que almacena información relacionada con la posición del GeoAdhoc router local. Debe contener al menos los siguientes elementos:

- Posición geográfica *POS_EPV*.

- Velocidad *S_EPV*.
- Dirección *H_EPV*.
- Sello temporal de cuando fue generada la posición geográfica, *TST_EPV*.
- Precisión de la posición geográfica *PAI_EPV*.

Al inicio, todos estos elementos toman el valor 0 para indicar un valor desconocido. El EPV se actualiza con una frecuencia determinada por el valor de la constante de protocolo *itsGnMinUpdateFrecuencyEPV* o mayor.

En el caso de una estación ITS estacionaria, el sello temporal del EPV se actualiza con una frecuencia igual o mayor al valor de *itsGnMinUpdateFrecuencyEPV*. Los campos de velocidad y dirección deberán fijarse a 0. Mientras la estación esté estacionaria el valor del campo de posición no debe cambiar.

### 3.7.3 Sequence Number

Cada GeoAdhoc router debe mantener un número de secuencia local que determina el campo Sequence Number (SN) del siguiente paquete GeoNetworking a transmitir.

Debe de ser inicializado a 0 y ser incrementado para cada paquete GeoNetworking P siguiendo la Fórmula 2:

$$SN(P) = (SN(P) + 1) mod\ SN\_MAX$$

**Fórmula 2: Sequence Number**

SN_MAX es el número de secuencia máximo posible. El valor obtenido debe incluirse en el paquete GeoNetworking.

El SN solo se incrementa solo para paquetes GeoNetworking multi-salto, los paquetes de salto único (BEACON, SHB) no lleva un campo SN.

### 3.7.4 Location Service Packet Buffer

El Location Service es un servicio que obtiene el vector de posición para una dirección destino determinada y que es invocada por el origen cuando tiene una SDU del protocolo de transporte que enviar y no tiene dicha información de posición.

Cuando se ejecute el LS el GeoAdhoc router debe encolar un paquete en un LS packet buffer para el destino buscado hasta que se complete el LS. Los paquetes subsecuentes, que se procesan mientras el LS está en proceso, también deben almacenarse.

El tamaño mínimo de este buffer se almacena en la constante de protocolo GN *itsGnLocationServicePacketBufferSize*.

El funcionamiento es el siguiente:

- Los paquetes que lleguen al LS packet buffer para un destino determinado (*GN_ADDR de una estación ITS)* se almacenan en la cola de la fila.
- Cuando llega un paquete nuevo al buffer y excede su capacidad (*buffer overflow*), los paquetes de la cabeza de la fila se eliminan y el paquete nuevo se almacena en la cola (*head drop*).
- Cuando el LS se completa, se envían todos paquetes del buffer usando un esquema FIFO.
- Cuando el tiempo en cola del paquete en el buffer excede el tiempo de vida especificado en el campo LT de la cabecera básica del paquete, este se descarta.
- Cuando un paquete almacenado se envía se reduce el valor del campo LT por el tiempo que ha estado en la cola y, preferiblemente, se actualiza el SO PV.
- Si el LS no se completa, todos los paquetes almacenados deben de ser descartados.

### *3.7.5 Forwarding packet buffer*

Los GeoAdhoc router deben utilizar *buffers* de encaminamiento de paquete (*forwarding packet buffers*) para almacenar temporalmente los paquetes durante el proceso de reencaminamiento. Deberá mantener al menos los siguientes *buffers*:

- *UC forwarding packet buffer* para almacenar los paquetes GUC por GN_ADDR.
- *BC forwarding packet buffer* para almacenar los paquetes TSB, GBC y GAC.

Además, si el encaminamiento basado en contienda (CBF) está activado:

- la constante de protocolo *itsGnNonAreaForwardingAlgorithm* tiene el valor 2 (CBF)

- la constante de protocolo *itsGnAreaForwardingAlgorithm* tiene el valor 2 (CBF) ó 3 (ADVANCED)

- el *router* deberá mantener un CBF packet buffer.

El tamaño mínimo de los buffers UC, BC y CBF viene determinado por las constantes de protocolo *itsGn**XX**ForwardingPacketBufferSize* (XX = Uc, Bc o Cbf).

El funcionamiento es similar para los tres, en el caso de los UC y BC forwarding packet buffer:

- Los paquetes GeoNetworking se almacenan al final de la cola.

- Cuando un paquete GeoNetworking llega y excede la capacidad del *buffer* los paquetes de la cabeza de la fila se eliminan y el paquete nuevo se almacena en la cola (*head drop*).

- Cuando el *buffer* se vacía, los paquetes almacenados se encaminan usando un esquema FIFO.

- Cuando el tiempo en cola del paquete en el *buffer* excede el tiempo de vida especificado en el campo LT de la cabecera básica del paquete, este se descarta.

- Cuando un paquete almacenado se envía se reduce el valor del campo LT por el tiempo que ha estado en la cola y, preferiblemente, se actualiza el SO PV.

- En el caso del buffer CBF los paquetes tienen un temporizador asociado, cuyo valor es fijado por el algoritmo CBF que cuando expira se elimina de la cola y se envía. En este caso el SO PV debe ser actualizado.

## 3.8 Estructura y formatos de paquetes GeoNetworking

La cabecera de GeoNetworking sigue la estructura mostrada en la Figura 24:

| Basic Header | Common Header | Extended Header (optional) |
|---|---|---|

**Figura 24: Cabecera GeoNetworking [51]**

El formato de la Basic Header y la Common Header es igual para los paquetes de todos los tipos de transporte, mientras que Extended Header es diferente.

### *3.8.1 Basic Header*

La cabera básica está presente en todos los paquetes GeoNetworking y sigue la estructura mostrada en las Figura 25 y Figura 26:

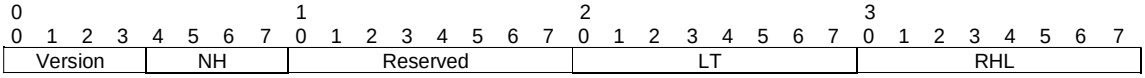


**Figura 25: Basic Header [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | Version | Octet 0 Bit 0 | Octet 0 Bit 3 | 4 bit unsigned integer | n/a | Identifies the version of the GeoNetworking protocol |
| 2 | NH | Octet 0 Bit 4 | Octet 0 Bit 7 | 4 bit unsigned integer | n/a | Identifies the type of header immediately following the GeoNetworking Basic data as specified in table 5 |
| 3 | Reserved | Octet 1 | Octet 1 | 8-bit unsigned integer | n/a | Reserved<br>Set to 0 |
| 4 | LT | Octet 2 | Octet 2 | 8 bit unsigned integer | n/a | Lifetime field. Indicates the maximum tolerable time a packet may be buffered until it reaches its destination<br>Bit 0 to Bit 5: LT sub-field Multiplier<br>Bit 6 to Bit 7: LT sub-field Base<br>Encoded as specified in clause 9.6.4 |
| 5 | RHL | Octet 3 | Octet 3 | 8 bit unsigned integer | [hops] | Decremented by 1 by each GeoAdhoc router that forwards the packet<br>The packet shall not be forwarded if RHL is decremented to zero |

**Figura 26: Campos de la Basic Header [51]**

La versión actual del protocolo GeoNetworking indicada en [51] es la 1. El campo Next Header nos indica si estamos usando los mecanismos de seguridad y, por tanto, a la cabecera le sigue una cabecera de tipo Common Header o un paquete protegido por los mecanismos de seguridad definidos en [55] [54] como encriptación, certificados y firmas digitales. Se codifica como se muestra en la Tabla 7.

| Next Header (NH) | Codificación | Descripción |
|---|---|---|
| ANY | 0 | Sin especificar |
| Common Header | 1 | Cabecera GeoNetworking Common Header |
| Secured Packet | 2 | Paquete GeoNetworking Secured Packet |

**Tabla 7: Valores del campo Next Header (NH) de la Basic Header [51]**

En cuanto al tiempo de vida, se utiliza una codificación no lineal mediante el uso de dos parámetros, una base y un multiplicador, para ofrecer una resolución adecuada a todos los valores. Se codifica según la Fórmula 3 y la base puede tomar los valores mostrados en la Tabla 8.

$$Lifetime = LT_{base} * LT_{multiplicador}$$

**Fórmula 3: Codificación del Tiempo de vida [51]**

| Valor | LTbase |
|---|---|
| 0 | 50ms |
| 1 | 1s |
| 2 | 10s |
| 3 | 100s |

**Tabla 8: Codificación del subcampo $LT_{base}$ [51]**

El tiempo por defecto es determinado por la constante de protocolo *itsGnDefaultPacketLifetime*, que debe de ser menor siempre de 600s como indica la constante de protocolo *itsGnMaxPacketLifetime.*

## *3.8.2 Common Header*

Al igual que la Basic Header, la Common Header está en todos paquetes GeoNetworking y sigue la estructura mostrada en las Figura 27 y Figura 28:

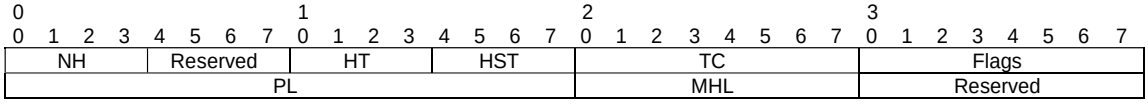


**Figura 27: Common Header [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | NH | Octet 0 Bit 0 | Octet 0 Bit 3 | 4 bit unsigned integer | n/a | Identifies the type of header immediately following the GeoNetworking headers as specified in table 8 |
| 2 | Reserved | Octet 0 Bit 4 | Octet 0 Bit 7 | 4 bit unsigned integer | n/a | Reserved<br>Set to 0 |
| 3 | HT | Octet 1 Bit 0 | Octet 1 Bit 3 | 4 bit unsigned integer | n/a | Identifies the type of the GeoNetworking header as specified in table 9 |
| 4 | HST | Octet 1 Bit 4 | Octet 1 Bit 7 | 4 bit unsigned integer | n/a | Identifies the sub-type of the GeoNetworking header as specified in table 9 |
| 5 | TC | Octet 2 | Octet 2 | 8 bit unsigned integer | n/a | Traffic class that represents Facility-layer requirements on packet transport. Encoding is specified in clause 9.7.5 |
| 6 | Flags | Octet 3 | Octet 3 | Bit field | n/a | Bit 0: Indicates whether the ITS-S is mobile or stationary (GN protocol constant `itsGnIsMobile`)<br>Bit 1 to Bit 7: Reserve, set to 0 |

**Figura 28: Campos de la Common Header [51]**

El campo Next Header tiene en este caso un significado diferente al campo de igual denominación de la Basic Header, indica el tipo de entidad de protocolo superior y puede tener los valores indicados en la Tabla 9:

| Next Header (NH) | Codificación | Descripción |
|---|---|---|
| ANY | 0 | Sin especificar. Se usa en los paquetes Beacon. |
| BTP-A | 1 | Protocolo de transporte BTP-A (interactivo) como se define en [55] |
| BTP-B | 2 | Protocolo de transporte BTP-B (no interactivo) como se define en [55] |
| IPv6 | 3 | Cabecera IPv6 como se define en [53] |

**Tabla 9: Valores del campo Next Header(NH) de la Common Header [51]**

Los campos HT y HST representan el tipo de cabecera GeoNetworking, si se usa el tipo de transporte GBC, GUC, etc. o si es un paquete tipo Beacon o de petición/repuesta del Location Service. En el caso de los servicios con destino un área geográfica (GBC/GAC), el campo HST indica la forma de dicha área definida según [56]. La codificación de los campos HT/HST se indica en la Tabla 10:

| Header type (HT) | Header Subtype (HST) | Tipo | Subtipo | Descripción |
|---|---|---|---|---|
| 0 | 0 | ANY | Unspecified | Sin especificar |
| 1 | 0 | Beacon | Unspecified | Beacon |
| 2 | 0 | GeoUnicast | Unspecified | GeoUnicast |
| 3 | 0 | Geographically-Scoped Anycast | Circular Area | GAC de área circular |
| 3 | 1 | | Rectangular Area | GAC de área Rectangular |
| 3 | 2 | | Ellipsoidal Area | GAC de área elipsoidal |
| 4 | 0 | Geographically-Scoped Broadcast | Circular Area | GBC de área circular |
| 4 | 1 | | Rectangular Area | GBC de área Rectangular |
| 4 | 2 | | Ellipsoidal Area | GBC de área elipsoidal |
| 5 | 0 | Topologically-Scoped Broadcast | Single-hop | TSB |
| 5 | 1 | | Multi-hop | SHB |
| 6 | 0 | Location Service | Request | Petición Location Service |
| 6 | 1 | | Reply | Respuesta Location Service |

**Tabla 10: Tipos y subtipos de cabeceras GeoNetworking [51]**

El campo TC, mostrado en la Figura 29, contiene dos bits de *flags*: SCF (Store-Carry-Forward) indica si el paquete debe almacenarse en el *buffer* si no hay un vecino adecuado. El bit Channel Offload indica si el paquete puede ser asignado a otro canal diferente al asignado en el campo TC ID. El campo TC ID es específico para cada tecnología de acceso como se detalla en la Tabla 11:

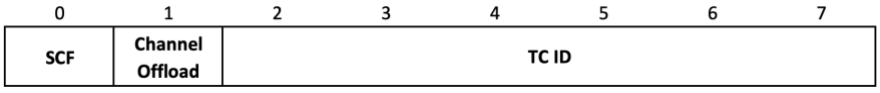


**Figura 29: Campo TC de la Common Header**

| **ITS-G5** [55] | | | **LTE-V2X** [20] | **USO** |
|---|---|---|---|---|
| **TC** | **UP (802.1d)** | **AC (802.11)** | **PPPP** | |
| **0** | 7 | AC_VO | 2 | DENM alta prioridad |
| **1** | 5 | AC_VI | 4 | DENM de prioridad normal |
| **2** | 3 | AC_BE | 5 | CAM |
| **3** | 1 | AC_BK | 7 | DEMN reenviados y otros paquetes de baja prioridad |

**Tabla 11: Clases de tráfico para ITS-G5 y LTE-V2X y su codificación en el campo TC ID**

### *3.8.3 Estructura de paquete completa*

A partir de la cabecera GeoNetworking, una trama/paquete completo presentarían la estructura que se muestra en la Figura 30 para el caso de no usar mecanismos de seguridad, o la Figura 31 para el caso de usar los mecanismos de seguridad (si el valor de la constante de protocolo *itsGnSecurity* es ENABLED):


**Figura 30: Estructura de paquete completa (Sin seguridad) [51]**

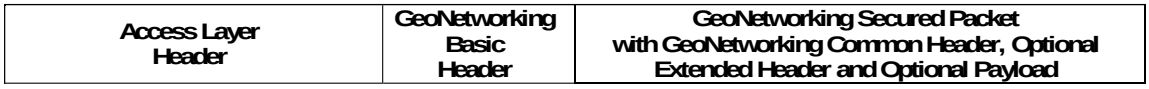


**Figura 31: Estructura de paquete completa (con seguridad) [51]**

La cabecera de la capa de enlace es específica de la tecnología de capa de enlace utilizada (ITS-G5, LTE-V2X, etc.) y el protocolo no se encarga de ella, aunque si indicará (dependiendo del tipo de transporte utilizado) la dirección de capa de enlace con el objeto de identificar el siguiente salto de un paquete GeoNetworking.

La carga (Payload) la componen los datos recibidos de la entidad de capa superior, como BTP, con el objeto de ser enviados. No todos los tipos de paquete tienen este campo.

## 3.9 Tipos de cabecera de paquete GeoNetworking

El protocolo establece varios tipos de cabecera de paquete, dependiendo del tipo de transporte (GeoUnicast, GeoBroadcast, Topologically Scooped, etc.) a utilizar o si son paquetes de asistencia al funcionamiento del protocolo como los Beacon o los correspondientes al Location Service.

Todos estos tipos de cabecera comparten el mismo formato, como ya se ha indicado, de Basic Header y Common Header a las que se añade una Extended Header que es diferente para cada tipo.

### 3.9.1 Cabecera GUC

La cabecera GUC se compone de la Basic Header y Common Header, acompañados del número de secuencia y dos vectores de posición: un vector de posición largo para la información geográfica del origen y un vector de posición corto para la información geográfica del destino.

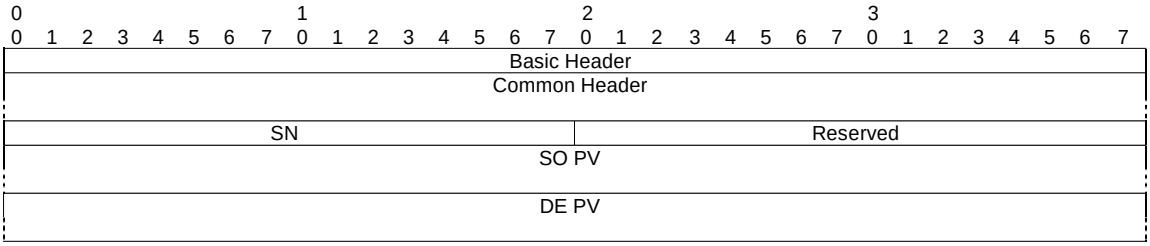


**Figura 32: Cabecera de paquete GUC [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | Basic header | Octet 0 | Octet 3 | Basic Header | n/a | Basic header as specified in clause 9.6<br>Length: 4 octets |
| 2 | Common header | Octet 4 | Octet 11 | Common Header | n/a | Common header as specified in clause 9.7<br>Length: 8 octets |
| 3 | SN | Octet 12 | Octet 13 | 16-bit unsigned integer | n/a | Sequence number field. Indicates the index of the sent GUC packet (clause 8.3) and used to detect duplicate GeoNetworking packets (annex A) |
| 4 | Reserved | Octet 14 | Octet 15 | 16-bit unsigned integer | n/a | Reserved<br>Set to 0 |
| 5 | SO PV | Octet 16 | Octet 39 | Long position vector | n/a | Long position vector containing the reference position of the source as specified in clause 9.5.2 (Long position vector)<br>Length: 24 octets |
| 6 | DE PV | Octet 40 | Octet 59 | Short position vector | n/a | Short position vector containing the position of the destination. It shall consist of the fields as specified in clause 9.5.3 (SPV)<br>Length: 20 octets |

**Figura 33: Campos de la cabecera de paquete GUC [51]**

### 3.9.2 Cabecera TSB

La cabecera TSB se compone de la Basic Header y Common Header, acompañados del número de secuencia y un vector de posición largo para la información geográfica del origen.

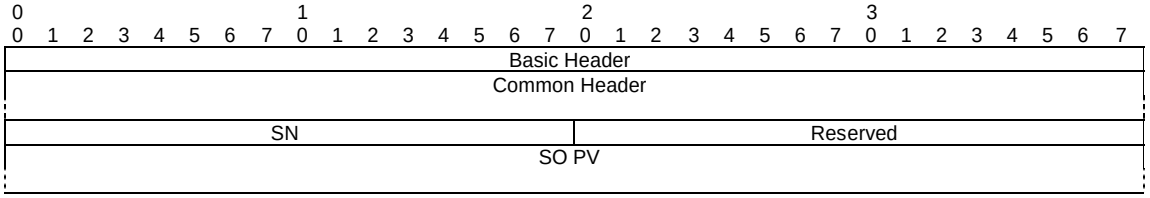


**Figura 34: Cabecera de paquete TSB [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | Basic header | Octet 0 | Octet 3 | Basic Header | n/a | Basic header as specified in clause 9.6<br>Length: 4 octets |
| 2 | Common header | Octet 4 | Octet 11 | Common Header | n/a | Common header as specified in clause 9.7<br>Length: 8 octets |
| 3 | SN | Octet 12 | Octet 13 | 16-bit unsigned integer | n/a | Sequence number field. Indicates the index of the sent TSB packet (clause 8.3) and used to detect duplicate GeoNetworking packets (annex A) |
| 4 | Reserved | Octet 14 | Octet 15 | 16-bit unsigned integer | n/a | Reserved<br>Set to 0 |
| 5 | SO PV | Octet 16 | Octet 39 | Long Position Vector | n/a | Long position vector containing the reference position of the source as specified in clause 9.5.2 (Long position vector)<br>Length: 24 octets |

**Figura 35: Campos de la cabecera de paquete TSB [51]**

## *3.9.3 Cabecera SHB*

La cabecera SHB se compone de la Basic Header y Common Header, acompañados del vector de posición largo con la información geográfica del origen. El último campo contiene información dependiente del medio, en el caso de ITS-G5 información relacionada con el DCC, como se muestra en la Figura 38.

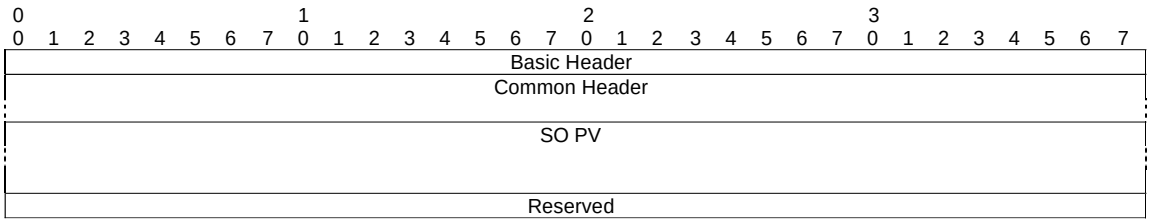


**Figura 36: Cabecera de paquete SHB [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | Basic header | Octet 0 | Octet 3 | Basic Header | n/a | Basic header as specified in clause 9.6<br>Length: 4 octets |
| 2 | Common header | Octet 4 | Octet 11 | Common Header | n/a | Common header as specified in clause 9.7<br>Length: 8 octets |
| 3 | SO PV | Octet 12 | Octet 35 | Long Position Vector | n/a | Long position vector containing the reference position of the source. It shall carry the fields as specified in clause 9.5.2 (Long position vector)<br>Length: 24 octets |
| 4 | Media dependent data | Octet 36 | Octet 39 | 32-bit unsigned integer | n/a | Used for media-dependent operations<br>If not used, it shall be set to 0<br>(see note) |
| NOTE: With ITS-G5 as specified in ETSI TS 102 636-4-2 [i.11], the field is used to transmit DCC-related information. | | | | | | |

**Figura 37: Campos de la cabecera de paquete SHB [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1-3 | See ETSI EN 302 636-4-1 [1], table 13. | | | | | |
| 4 | LocAM (Mode Reserved) | Octet 40 | Octet 43 | 32-bit unsigned integer | n/a | **Octet: 40**<br>Bit 0 to Bit 7: Current CBR_Chp encoded as ffor(CBR_Chp*25)<br>**Octet 41**<br>Bit 0 to Bit 7: Current CBR_Lhp encoded as ffor(CBR_Lhp*25)<br>**Octet 42**<br>Bit 0 to Bit 4: Transmit power of the current packet, E.I.R.P [0 dBm to 31 dBm, unit 1 dBm, values higher than 31 dBm shall be set to 31 dBm]<br>Bit 5 to Bit 7: Reserved for future use<br>**Octet 43**<br>Reserved for MCO |

**Figura 38: Campos específicos para ITS-G5 en la cabecera SHB [35]**

## *3.9.4 Cabecera GBC/GAC*

Los paquetes GeoBroadcast y GeoAnycast tienen como destino áreas geográficas concretas. Por ello la cabecera de dichos paquetes, que es igual para ambos tipos, incluye la información del área geográfica de destino: las coordenadas (latitud y longitud) del centro del área, dos parámetros denominados distancia a y distancia b que dependen del tipo de área descrita y el ángulo que forma la forma geométrica que define el área respecto al norte. El estándar [56] define 3 tipos de áreas geográficas: circular, rectangular y elíptica, como muestra la Figura 39.

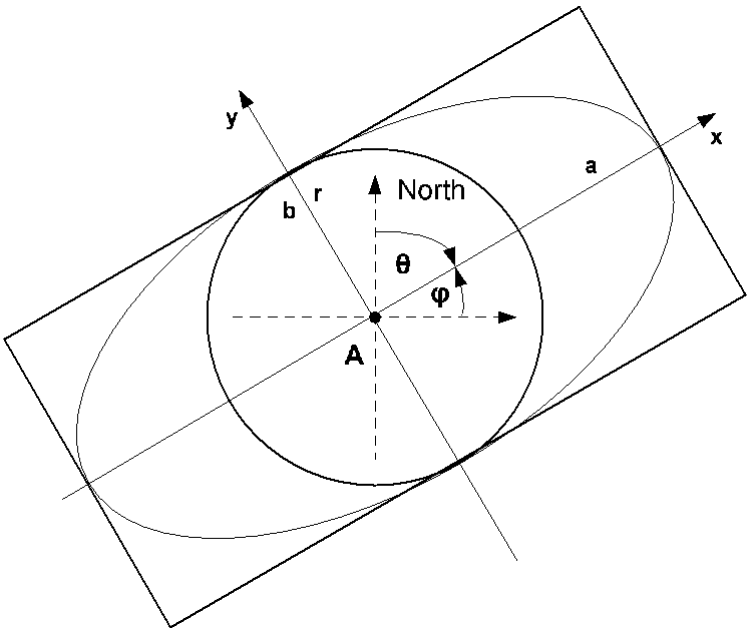


**Figura 39: definición de áreas geográficas [56]**

En el caso de las áreas geométricas circulares el campo *Distance a* representa el radio del círculo y tanto los campos *Distance b* como *Angle* toman el valor 0.

La cabecera incluye además el número de secuencia del paquete, un Long Position Vector con la información de posición del origen y las cabeceras Basic Header y Common Header.

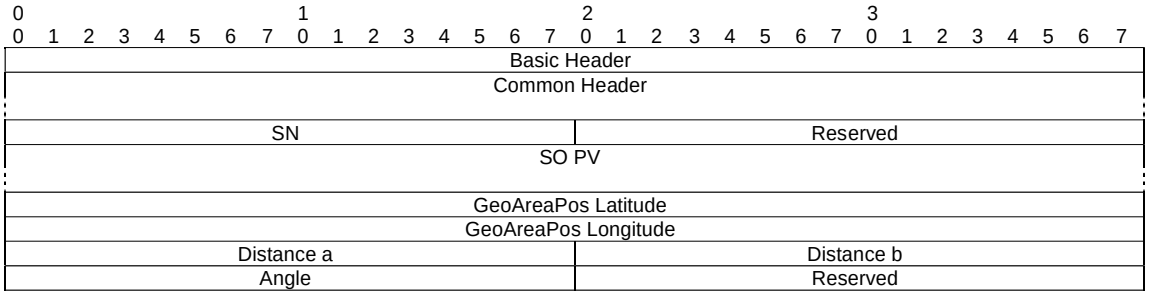


**Figura 40: Cabecera de paquete GAC/GBC [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | Basic header | Octet 0 | Octet 3 | Basic Header | n/a | Basic header as specified in clause 9.6<br>Length: 4 octets |
| 2 | Common header | Octet 4 | Octet 11 | Common Header | n/a | Common header as specified in clause 9.7<br>Length: 8 octets |
| 3 | SN | Octet 12 | Octet 13 | 16-bit unsigned integer | n/a | Sequence number field. Indicates the index of the sent GBC/GAC packet (clause 8.3) and used to detect duplicate GeoNetworking packets (annex A) |
| 4 | Reserved | Octet 14 | Octet 15 | 16-bit unsigned integer | n/a | Reserved<br>Set to 0 |
| 5 | SO PV | Octet 16 | Octet 39 | Long position vector | n/a | LongPositionVector containing the reference position of the source as specified in clause 9.5.2 (LongPositionVector)<br>Length: 24 octets |
| 6 | GeoAreaPos Latitude | Octet 40 | Octet 43 | 32-bit signed integer | [1/10 micro-degree] | WGS 84 [i.6] latitude for the centre position of the geometric shape as defined in ETSI EN 302 931 [8] in 1/10 micro degree |
| 7 | GeoAreaPos Longitude | Octet 44 | Octet 47 | 32-bit signed integer | [1/10 micro-degree] | WGS 84 [i.6] longitude for the centre position of the geometric shape as defined in ETSI EN 302 931 [8] in 1/10 micro degree |
| 8 | Distance a | Octet 48 | Octet 49 | 16-bit unsigned integer | [m] | Distance a of the geometric shape as defined in ETSI EN 302 931 [8] in metres |
| 9 | Distance b | Octet 50 | Octet 51 | 16-bit unsigned integer | [m] | Distance b of the geometric shape as defined in ETSI EN 302 931 [8] in metres |
| 10 | Angle | Octet 52 | Octet 53 | 16-bit unsigned integer | [°] | Angle of the geometric shape as defined in ETSI EN 302 931 [8] in degrees from North |
| 11 | Reserved | Octet 54 | Octet 55 | 16-bit unsigned integer | n/a | Reserved<br>Set to 0 |

**Figura 41: Campos de la cabecera de paquete GAC/GBC [51]**

## *3.9.5 Cabecera Beacon*

Las cabeceras Beacon se envían de forma periódica para anunciar a los nodos vecinos la posición de la estación. Incluyen las cabeceras Basic Header, Common Header y un Vector de Posición largo.

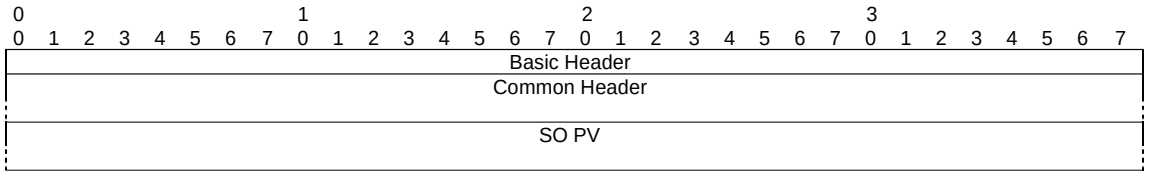


**Figura 42: Cabecera de paquete Beacon [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | Basic header | Octet 0 | Octet 3 | Basic Header | n/a | Basic header as specified in clause 9.6<br>Length: 4 octets |
| 2 | Common header | Octet 4 | Octet 11 | Common Header | n/a | Common header as specified in clause 9.7<br>Length: 8 octets |
| 3 | SO PV | Octet 12 | Octet 35 | Long Position Vector | n/a | Long position vector containing the reference position of the source. It shall carry the fields as specified in clause 9.5.2 (Long position vector)<br>Length: 24 octets |

**Figura 43: Campos de la cabecera de paquete Beacon [51]**

## *3.9.6 Cabeceras LS Request y LS Reply*

El Location Service se utiliza cuando se necesita conocer la posición de otra estación ITS, por ejemplo, cuando se envía un paquete GUC y el GeoAdhoc router desconoce la posición del destino por no tenerla almacenada en su LocT.

La cabecera LS Request está formada por la Basic Header, la Common Header, el número de secuencia, un Long Position Vector con la información de posición del origen y la dirección GeoNetworking del nodo del que se desea conocer la información.

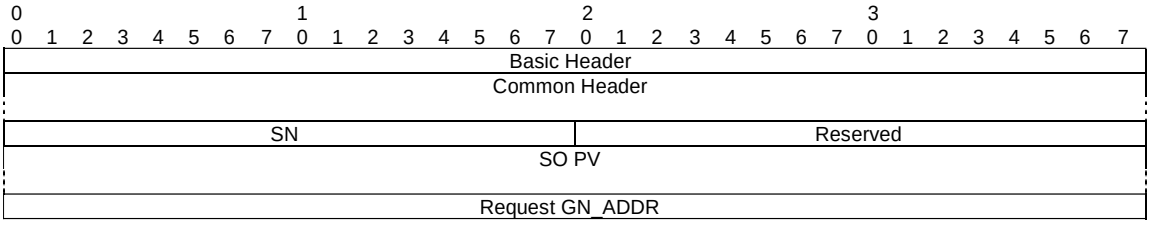


**Figura 44: Cabecera de paquete LS Request [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | Basic header | Octet 0 | Octet 3 | Basic Header | n/a | Basic header as specified in clause 9.6<br>Length: 4 octets |
| 2 | Common header | Octet 4 | Octet 11 | Common Header | n/a | Common header as specified in clause 9.7<br>Length: 8 octets |
| 3 | SN | Octet 12 | Octet 13 | 16-bit unsigned integer | n/a | Sequence number field. Indicates the index of the sent LS Request packet (clause 8.3) and used to detect duplicate GeoNetworking packets (annex A) |
| 4 | Reserved | Octet 14 | Octet 15 | 16-bit unsigned integer | n/a | Reserved<br>Set to 0 |
| 5 | SO PV | Octet 16 | Octet 39 | Long position vector | n/a | Long position vector containing the position of the source as specified in clause 9.5.2 (Long position vector)<br>Length: 24 octets |
| 6 | Request GN_ADDR | Octet 40 | Octet 47 | 64-bit address | n/a | The GN_ADDR address for the GeoAdhoc router entity for which the location is being requested |

**Figura 45: Campos de la cabecera de paquete LS Request [51]**

Cuando el nodo destino recibe la solicitud responde con un paquete LS Reply, formado por las cabeceras Basic y Common Header, el número de secuencia, un Long Position Vector con su información de posición y un Short Position Vector con la información de posición del nodo que originó la petición.

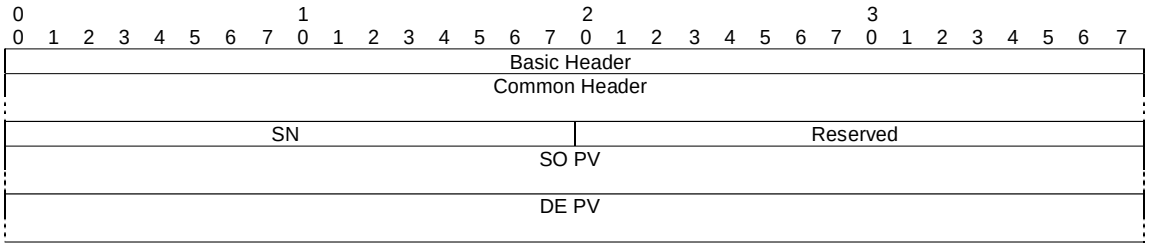


**Figura 46: Cabecera LS Reply [51]**

| Field # | Field name | Octet/bit position | | Type | Unit | Description |
|---|---|---|---|---|---|---|
| | | First | Last | | | |
| 1 | Basic header | Octet 0 | Octet 3 | Basic Header | n/a | Basic header as specified in clause 9.6<br>Length: 4 octets |
| 2 | Common header | Octet 4 | Octet 11 | Common Header | n/a | Common header as specified in clause 9.7<br>Length: 8 octets |
| 3 | SN | Octet 12 | Octet 13 | 16-bit unsigned integer | n/a | Sequence number field. Indicates the index of the sent LS Reply packet (clause 8.3) and used to detect duplicate GeoNetworking packets (annex A) |
| 4 | Reserved | Octet 14 | Octet 15 | 16-bit unsigned integer | n/a | Reserved<br>Set to 0 |
| 5 | SO PV | Octet 16 | Octet 39 | Long position vector | n/a | Long position vector containing the reference position of the source, which represents the Request GN_ADDR in the corresponding LS Request, as specified in clause 9.5.2 (Long position vector)<br>Length: 24 octets |
| 6 | DE PV | Octet 40 | Octet 59 | Short position vector | n/a | Short position vector containing the reference position of the destination. It shall carry the fields as specified in clause 9.5.3 (Short position vector)<br>Length: 20 octets |

**Figura 47: Campos de la cabecera LS Reply [51]**

En este proceso pueden intervenir nodos intermedios, que reenvían la información hasta llegar al destino, como muestra la Figura 48.

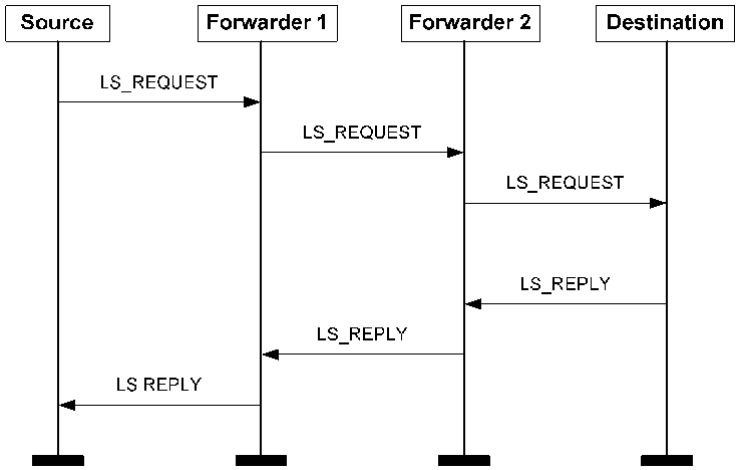


**Figura 48: Secuencia de funcionamiento del Location Service [51]**

## 3.10 Servicios de datos GeoNetworking

El estándar define varias primitivas de servicios de datos, que permiten a las entidades de protocolos de transporte ITS mandar y recibir PDUs a través del GN_SAP.

### *3.10.1 Primitiva GN-Data.request*

Es utilizada por la entidad de protocolo de transporte ITS para solicitar enviar un paquete GeoNetworking. Cuando el protocolo GeoNetworking recibe una primitiva de servicio GN-Data.request realiza las operaciones para crear y enviar el correspondiente paquete GeoNetworking a la capa de acceso. Los parámetros de la primitiva son los indicados en la Figura 49.

```
GN-DATA.request (
            Upper protocol entity,
            Packet transport type,
            Destination address,
            Communication profile,
            Security profile, (optional)
            ITS-AID length, (optional)
            ITS-AID, (optional)
            Security permissions length, (optional)
            Security permissions, (optional)
            Security context information, (optional)
            Security target ID list length, (optional)
            Security target ID list,(optional)
            Maximum packet lifetime, (optional)
            Repetition interval, (optional)
            Maximum repetition time, (optional)
            Maximum hop limit, (optional)
            Traffic class,
            Length,
            Data
            )
```

**Figura 49: Primitiva GN-Data.request [51]**

El estándar no especifica exactamente como deben de ser los parámetros de dicha primitiva, más allá de las unidades de alguno de ellos. Los campos más importantes para nuestra implementación son los siguientes:

- Upper protocolo entity: indica si la petición la realiza un protocolo de transporte específico de ITS, como BTP, o la sub-capa de adaptación GeoNetworking a IPv6 (GN6ASL).

- Packet transport type: indica el tipo de transporte del paquete (GUC, SHB, TSB, GBC o GAC). Los paquetes de tipo Beacon y LS request/reply son paquetes de gestión del funcionamiento del protocolo y son generados por la propia entidad GeoNetworking, por lo que no son solicitados por las capas superiores.

- Destination: especifica la dirección GeoNetworking, u opcionalmente el campo MID de la misma con el resto de campos establecidos a 0, del destino para GeoUnicast o el área geográfica para GeoAnycast/GeoBroadcast.

- Communication profile: indica el tipo de entidad de protocolo de capa de enlace (ITS-G5 o LTE-V2X).

- Maximum lifetime: especifica el máximo tiempo tolerable (en s) que un paquete puede ser almacenado hasta que alcance el destino. Si no se usa se toma el valor de *itsGnDefaultPacketLifetime.*

- Repetition Interval: indica el tiempo entre dos transmisiones consecutivas del mismo paquete (en ms). Es opcional.

- Maximum repetition time: especifica el tiempo (en ms) durante el cual un paquete será repetido si está establecido un intervalo de repetición.

- Maximum Hop limit: número de saltos que puede tener un paquete en la red.

- Traffic class: especifica la clase de tráfico del mensaje.

- Length: especifica la longitud de los datos que se envían en el parámetro *Data*.

- Data: representa la carga del paquete GeoNetworking a enviar, es decir, la SDU de BTP o GN6ASL(T-SDU/GN6-SDU).

### *3.10.2 Primitiva GN-Data.confirm*

Se utiliza para confirmar que el paquete GeoNetworking ha sido procesado con éxito en respuesta a una petición GN-Data.request. No se especifica qué debe hacer cuando la entidad de protocolo superior, como BTP, recibe esta primitiva. Los parámetros de la primitiva son los indicados en la Figura 50.

```
GN-DATA.confirm (
            ResultCode
            )
```

**Figura 50: Primitiva GN-Data.confirm [51]**

El código *ResultCode* indica si la primitiva GN-Data.request ha sido:

1. aceptada;

2. rechazada debido a que el tamaño de la T/GN6-PDU supera al indicado en la constante de protocolo *itsGnMaxSduSize*;

3. rechazada debido a que el tiempo de vida supera el valor máximo indicado en la constante de protocolo *itsGnMaxPacketLifetime*;

4. rechazada debido a que el intervalo de repetición es menor que el valor indicado en la constante de protocolo *itsGnMinPacketRepetitionInterval*;

5. rechazada debido a clase de tráfico no soportada;

6. rechazada debido a que el tamaño del área geográfica excede el valor máximo indicado en la constate de protocolo *itsGnMaxGeoAreaSize*; o

7. rechazada sin motivo específico si la primitiva GN-Data.request no puede ser aceptada por cualquier otra razón.

### 3.10.3 Primitiva GN-Data.indication

Indica a una entidad de protocolo superior que se ha recibido un paquete GeoNetworking. Es generada por el protocolo GeoNetworking para entregar los datos contenidos en un paquete GeoNetworking a la entidad de protocolo superior, que se encargará de procesar dicha información. Los parámetros de la primitiva se muestran en la Figura 51.

```
GN-DATA.indication (
            Upper protocol entity,
            Packet transport type,
            Destination, (optional)
            Source position vector,
            Security report, (optional)
            Certificate id, (optional)
            ITS-AID length, (optional)
            ITS-AID, (optional)
            Security permissions length, (optional)
            Security permissions, (optional)
            Traffic class,
            Remaining packet lifetime, (optional),
            Remaining hop limit, (optional)
            Length,
            Data    -- T/GN6-PDU
            )
```

**Figura** 51**: Primitiva GN-Data.indication [51]**

Los campos más importantes para nuestra implementación son los siguientes:

- Upper protocolo entity: determina la entidad de protocolo que procesa la primitiva de servicio (BTP o GN6).

- Packet transport type: indica el tipo de transporte del paquete (GUC, SHB, TSB, GBC o GAC) del paquete recibido.

- Destination: especifica la dirección GeoNetworking para GeoUnicast o el área geográfica para GeoAnycast/GeoBroadcast con la que se creó el paquete GeoNetworking en origen.

- Source position vector: la posición geográfica del origen del paquete GeoNetworking recibido.

- Traffic class: especifica la clase de tráfico del mensaje con la que se creó el paquete en el origen.
- Remaining packet lifetime: tiempo de vida restante que le queda al paquete.
- Remaining Hop limit: número de saltos restantes del paquete.
- Length: especifica la longitud del parámetro *Data*.
- Data: representa la carga del paquete GeoNetworking recibido, es decir, la SDU de BTP o GN6ASL(T-SDU/GN6-SDU).

## 3.11 Funcionamiento del protocolo

El funcionamiento del protocolo se divide en dos funciones principales: gestión de la red y tratamiento de los paquetes.

En el primer grupo se encuentra la configuración de la dirección GeoNetworking, la actualización del vector de posición local y tiempo, y el Location Service, ya vistos en los puntos 3.5, 3.7.2, y 3.7.4 respectivamente.

Se incluye también la gestión de los *beacons*, o balizas, que como vimos en el punto 3.9.5 son mensajes que se envían periódicamente para anunciar a los nodos cercanos la ubicación y dirección GeoNetworking de la estación ITS. Estos mensajes se envían con una frecuencia definida en la constante de protocolo *itsGnBeaconServiceRetransmitTimer*, que indica cada cuánto tiempo se retransmite un *beacon*. Este temporizador, que tiene un valor por defecto de 3000ms, se resetea si antes de que caduque se envía otro paquete que incluya el vector de posición de origen, como un paquete SHB o GBC. En nuestra implementación no se han incluido los *beacon*, pero el envío continuo de CAM suple su función de anuncio.

En el segundo grupo se distinguen procedimientos diferentes para el tratamiento de los paquetes tanto en el envío, recepción y reencaminamiento de los mismos en función del tipo de transporte: GUC, TSB, SHB, GBC o GAC.

### *3.11.1 Tratamiento de los paquetes*

Cuando el *router* GeoAdhoc recibe una petición mediante la directiva GN-Data.request o un paquete GeoNetworking proveniente de otra estación ITS debe ejecutar varios pasos que dependen del tipo de paquete. Los casos de más interés en nuestro caso son los paquetes SHB y GBC, que son los tipos de transporte utilizados para los mensajes CAM y DENM respectivamente, por lo que serán los que analizaremos en cuestión.

Algunas de los pasos a ejecutar son muy similares en ambos casos, como el procesado de las cabeceras Basic Header y Common Header.

#### 3.11.1.1 Basic Header

Cuando el *router* GeoAdhoc recibe una petición para enviar un paquete, el primer paso a ejecutar es rellenar la cabecera Basic Header (3.8.1) con los valores indicados en la Tabla 12.

A la recepción de un paquete se comprueba si la versión del protocolo está soportada por nuestro *router*, en caso contrario se descarta.

| Campo | Valor |
|---|---|
| Versión | Valor de la constante *itsGnProtocolVersion* |
| NH | Toma el valor 1 (unsecured) o 2 (secured) del campo *Security profile* de la primitiva GN-Data.request, si existe, o de la constante *itsGnSecurity* en caso contrario |
| Reserved | 0 |
| LT | Valor indicado en el campo *Maximum packet lifetime* de la primitiva GN-Data.request, si existe, o el valor de la constante *itsGnDefaultPacketLifetime* en caso contrario |
| RHL | 1 en el caso de los paquetes SHB o BEACON. Para el resto de paquetes el valor indicado en el campo *Maximum Hop limit* de la primitiva GN-Data.request, si existe, el valor de la constante *itsGnDefaultHopLimit* en caso contrario. |

**Tabla 12: Valores de la cabecera Basic Header**

Si el paquete se va a reenviar será necesario decrementar el valor del campo RHL por uno y descontar del campo LT el tiempo que haya estado almacenado en el *buffer*. En nuestra implementación no se ha implementado el reenvío de los paquetes.

#### 3.11.1.2 Common Header

Cuando el *router* GeoAdhoc recibe una petición para enviar un paquete, el siguiente paso es rellenar la cabecera Common Header (3.8.2) con los valores indicados en la Tabla 13.

| Campo | Valor |
|---|---|

| | |
|---|---|
| NH | Valor de la constante *itsGnProtocolVersion* |
| Reserved | 0 |
| HT y HST | Los valores correspondientes al tipo de paquete indicado en la primitiva GN-Data.request codificados según la tabla |
| TC | Traffic Class indicado en el campo *Traffic class* de la primitiva GN-Data.request, si existe, o el valor de la constante *itsGnDefaultTrafficClass* en caso contrario codificados como se indica en la |
| Flags | El bit 0 toma el valor de la constante *itsGnIsMobile*. El resto de bits a 0 |
| PL | El tamaño en octetos de la PDU de la capa de transporte recibida en la primitiva GN-Data.request. 0 si es un paquete BEACON. |
| MHL | 1 en el caso de los paquetes SHB o BEACON. Para el resto de paquetes el valor indicado en el campo *Maximum Hop limit* de la primitiva GN-Data.request, si existe, el valor de la constante *itsGnDefaultHopLimit* en caso contrario. |
| Reserved | 0 |

**Tabla 13: Valores de la cabecera Common Header**

Cuando se recibe un paquete el GeoAdhoc router deberá comprobar si el valor del campo MHL es menor que el del campo RHL de la cabecera Basic Header, en cuyo caso se ha superado el número de saltos permitidos para dicho paquete y debe ser descartado.

En caso contrario se comprueba el campo HT para ver qué tipo de paquete es y ejecutar los pasos correspondientes a dicho tipo.

#### 3.11.1.3 Paquetes SHB

Cuando la petición recibida es para enviar un paquete SHB, se rellena la cabecera SHB (3.9.3) con los datos del EGO Position Vector y, si la hubiera, información dependiente del medio como la mostrada en la Figura 38 para ITS-G5.

A continuación, si el parámetro SCF del campo Traffic Class está activado, consulta en la LocT si existe algún nodo marcado como vecino. Si no existiera se almacenaría el paquete en el *buffer* de paquetes para ser procesado más tarde.

En nuestro caso no está implementado aún ni la LocT ni el buffer de paquetes, por lo que todos los paquetes creados llevan el campo SCF deshabilitado y se envían inmediatamente mediante broadcast como indica el estándar para ese caso.

A la recepción de un paquete el *router* GeoAdhoc procesará las cabeceras Basic Header y Common Header como se indicó en los puntos anteriores

A continuación, si existe una entrada en la LocT para la dirección GeoNetworking del origen, se actualizarían los datos de ubicación y tasa de envío de paquetes en dicha entrada, marcándolo además como vecino en el *flag IS_NEIGHBOUR*. En caso de que no

existiera se añadiría una entrada nueva para dicho origen. Este paso en nuestra implementación no se ejecuta al no estar implementada la LocT.

A continuación, se crea una primitiva GN-Data.indication con los datos mostrados en la Tabla 14 y se envía a la entidad de protocolo de nivel superior correspondiente.

| Parámetro | Valor |
|---|---|
| Upper protocol entity | BTP-A, BTP-B o IPv6 como indica la Tabla 9 |
| Packet transport type | SHB |
| Source Position Vector | Los valores recibidos en el paquete |
| Security report | Valores relacionados con los mecanismos de seguridad. Son opcionales y no están implementados en nuestro software. |
| Certificate ID | |
| ITS-AID length | |
| ITS-AID | |
| Security permissions length | |
| Security permissions | |
| Traffic class | Valor de TC (Common Header) |
| Remaining packet lifetime | Valor de LT (Basic Header) |
| Remaining hop limit | Valor de RHL (Basic Header) |
| Length | Longitud de la PDU recibida (payload) |
| Data | PDU recibida (payload) |

**Tabla 14: Parámetros de la primitiva GN-Data.indication a enviar a la capa de transporte (SHB)**

#### 3.11.1.4 Paquetes GBC

Si la petición recibida es para enviar un paquete GBC, se rellena la cabecera GBC (3.9.4) con los datos de la Tabla 15.

| Parámetro | Valor |
|---|---|
| SN | Valor actual del *sequence number* según lo visto en 3.7.3 |
| Reserved | 0 |
| Source Position Vector | Los valores del Ego PV |
| GeoAreaPos Latitude | El parámetro *GeoAreaPos latitude* recibido en la primitiva GN-Data.request |
| GeoAreaPos Longitude | El parámetro *GeoAreaPos longitude* recibido en la primitiva GN-Data.request |
| Distance a | El parámetro *distance a* recibido en la primitiva GN-Data.request |
| Distance b | El parámetro *distance b* recibido en la primitiva GN-Data.request |
| Angle | El parámetro angle recibido en la primitiva GN-Data.request |
| Reserved | 0 |

**Tabla 15: Campos de la cabecera extendida GBC**

A continuación, si el parámetro SCF del campo Traffic Class está activado, consulta en la LocT si existe algún nodo marcado como vecino. Si no existiera se almacenaría el paquete en el *buffer* de paquetes para ser procesado más tarde y no se ejecutarían más pasos.

El siguiente paso sería ejecutar el procedimiento de selección de encaminamiento de GeoNetworking, descrito en el Anexo D del estándar ETSI EN 302 636-4-1 [51].

Para ello utiliza la función F(x,y), especificada en el estándar ETSI EN 302 931 [56] para cada una de las formas geométricas soportadas (circular, rectangular y elíptica), la cual devuelve uno de los valores indicados en la Figura 52, para determinar si se encuentra dentro del área geográfica de destino o no.

$$F(x,y) = \begin{cases} = 1 & \text{for } x = 0 \text{ and } y = 0 \text{ (at the centre point)} \\ > 0 & \text{inside the geographic al area} \\ = 0 & \text{at the border of the geographic al area} \\ < 0 & \text{outside the geographic al area} \end{cases}$$

**Figura 52: Determinación de la posición del *router* GeoAdhoc respecto al área de destino**

Si el resultado de esta función indica que el *router* GeoAdhoc se encuentra dentro o en el borde del área, ejecuta uno de los denominados algoritmos *area forwarding*, definidos en anexo F del estándar ETSI EN 302 636-4-1 [51]. De ellos el más sencillo simplemente envía mediante broadcast el paquete. Este es el método utilizado en nuestra implementación del protocolo. La selección del algoritmo a utilizar se realiza mediante la constante de protocolo *itsGnAreaForwardingAlgorithm*.

Si el resultado de esta función indica que el *router* GeoAdhoc se encuentra fuera del área, ejecuta uno de los denominados algoritmos *non-area forwarding*, definidos en anexo E del estándar ETSI EN 302 636-4-1 [51]. Estos son el algoritmo *Greedy Forwarding* (GF), que selecciona al nodo vecino más cercano geográficamente al destino como el destinatario del paquete, y el algoritmo *Contention-based Forwarding* (CBF), que utiliza un mecanismo en el que se envía el paquete por broadcast y todos los vecinos que lo reciben lo almacenan iniciando un temporizador con un *timeout* proporcional a la diferencia entre la distancia del *router* origen y su propia distancia al destino. La selección del algoritmo a utilizar se realiza mediante la constante de protocolo *itsGnNonAreaForwardingAlgorithm.*

Cuando se recibe un paquete GBC, deberán ejecutarse de nuevo estos algoritmos para reenviar el paquete en caso necesario y determinar si el paquete se pasa a la entidad de protocolo superior, en el caso de que se esté dentro o en el borde del área de destino, o no. En caso de encontrarse dentro del área de destino se crea una primitiva GN-Data.indication con los datos mostrados en la Tabla 16 y se envía a la entidad de protocolo de nivel superior correspondiente.

| Parámetro | Valor |
|---|---|
| Upper protocol entity | BTP-A, BTP-B o IPv6 como indica la Tabla 9 |
| Packet transport type | GBC |
| Destination | Área geográfica indicada en el paquete (GeoPos, distance a, distance b, angle) |
| Source Position Vector | Los valores recibidos en el paquete |
| Security report | Valores relacionados con los mecanismos de seguridad. Son opcionales y no están implementados en nuestro software. |
| Certificate ID | |
| ITS-AID length | |
| ITS-AID | |
| Security permissions length | |
| Security permissions | |
| Traffic class | Valor de TC (Common Header) |
| Remaining packet lifetime | Valor de LT (Basic Header) |
| Remaining hop limit | Valor de RHL (Basic Header) |
| Length | Longitud de la PDU recibida (payload) |
| Data | PDU recibida (payload) |

**Tabla 16: Parámetros de la primitiva GN-Data.indication a enviar a la capa de transporte (GBC)**

# 4

# Instalación de Vanetza y comunicación con OpenC2X

Aunque el objetivo principal de este Trabajo Fin de Máster es la implementación de GeoNetworking en el software OpenC2X, con el fin de que nuestros miniPC que hacen las veces de estación ITS ejecuten una versión cada vez más completa de la pila ETSI C-ITS, previamente decidimos realizar una instalación de Vanetza con el fin de valorar la posibilidad de utilizarlo como alternativa a OpenC2X. Estos equipos tienen instalado Ubuntu 16.04 con el Kernel 4.2.8 con las modificaciones necesarias para activar el modo OCB y las frecuencias correspondientes a ITS-G5 en la interfaz Wireless indicadas en el trabajo de Javier Fernández Pastrana [8] y software OpenC2X.

## 4.1 Instalación de Kernel 5.13.8

A mayores, y con el fin de realizar el desarrollo y compilación de las modificaciones necesarias en OpenC2X de manera más cómoda y ágil se utilizó un equipo de escritorio con procesador i5 de 4 generación y 16GB de RAM en el cual se instaló Ubuntu 16.04 y OpenC2X. A la hora de compilar el Kernel se decidió descargar la versión más actual que estaba disponible, la 5.13.8, y comprobar si las modificaciones propuestas por Javier Fernández Pastrana seguían siendo válidas.

Al realizar una revisión de los cambios propuestos se comprobó que algunos de los ficheros ya incluían las modificaciones necesarias, y que únicamente era necesario modificar cinco ficheros con las modificaciones mostradas en las Figura 53, Figura 54, Figura 55, Figura 56 y Figura 57.

1 drivers/net/wireless/ath/ath9k/ani.c

```
@@ -326,6 +326,7 @@ void ath9k_ani_reset(struct ath_hw *ah, bool is_scanning)
326 326
327 327           if (is_scanning ||
328 328               (ah->opmode != NL80211_IFTYPE_STATION &&
329     +              ah->opmode != NL80211_IFTYPE_OCB &&
329 330               ah->opmode != NL80211_IFTYPE_ADHOC)) {
330 331                   /*
331 332                    * If we're scanning or in AP mode, the defaults (ini)
```

**Figura 53: Modificación del fichero drivers/net/wireless/ath/ath9k/ani.c**

20 drivers/net/wireless/ath/ath9k/common-init.c

```
@@ -86,6 +86,26 @@ static const struct ieee80211_channel ath9k_5ghz_chantable[] = {
86  86          CHAN5G(5785, 35), /* Channel 157 */
87  87          CHAN5G(5805, 36), /* Channel 161 */
88  88          CHAN5G(5825, 37), /* Channel 165 */
    89  +       /* Channel definitions for ITS-G5 */
    90  +       CHAN5G(5850, 39), /* Channel 170 */
    91  +       /* ITA-G5B */
    92  +       CHAN5G(5855, 39), /* Channel 171 */
    93  +       CHAN5G(5860, 40), /* Channel 172 */
    94  +       CHAN5G(5865, 41), /* Channel 173 */
    95  +       CHAN5G(5870, 42), /* Channel 174 */
    96  +       /* ITS-G5A */
    97  +       CHAN5G(5875, 43), /* Channel 175 */
    98  +       CHAN5G(5880, 44), /* Channel 176 */
    99  +       CHAN5G(5885, 45), /* Channel 177 */
   100  +       CHAN5G(5890, 46), /* Channel 178 */
   101  +       CHAN5G(5895, 47), /* Channel 179 */
   102  +       CHAN5G(5900, 48), /* Channel 180 */
   103  +       CHAN5G(5905, 49), /* Channel 181 */
   104  +       /* ITS-G5D */
   105  +       CHAN5G(5910, 50), /* Channel 182 */
   106  +       CHAN5G(5915, 51), /* Channel 183 */
   107  +       CHAN5G(5920, 52), /* Channel 184 */
   108  +       CHAN5G(5925, 53), /* Channel 185 */
89  109 };
90  110
91  111 /* Atheros hardware rate code addition for short preamble */
```

**Figura 54: Modificación del fichero drivers/net/Wireless/ath/ath9k/common-init.c**

3 drivers/net/wireless/ath/ath9k/hw.h

```
@@ -73,7 +73,8 @@
73  73
74  74  #define ATH9K_RSSI_BAD                  -128
75  75
76      - #define ATH9K_NUM_CHANNELS      38
        76  + /* Modified from 38 to 54 to enable ITS-G5 channels */
        77  + #define ATH9K_NUM_CHANNELS      54
77  78
78  79  /* Register read/write primitives */
79  80  #define REG_WRITE(_ah, _reg, _val) \
```

**Figura 55: Modificación del fichero drivers/net/wireless/ath/ath9k/hw.h**

3 drivers/net/wireless/ath/ath9k/main.c

```
@@ -1223,7 +1223,8 @@ void ath9k_calculate_summary_state(struct ath_softc *sc,
1223 1223          ath9k_hw_setopmode(ah);
1224 1224
1225 1225          ctx->switch_after_beacon = false;
1226      -       if ((iter_data.nstations + iter_data.nadhocs + iter_data.nmeshes) > 0)
        1226  +       /* MODIFIED TO INCLUDE OCB MODE, MAYBE NOT NEEDED!!! */
        1227  +       if ((iter_data.nstations + iter_data.nadhocs + iter_data.nmeshes + iter_data.nocbs) > 0)
1227 1228                  ah->imask |= ATH9K_INT_TSFOOR;
1228 1229          else {
1229 1230                  ah->imask &= ~ATH9K_INT_TSFOOR;
```

**Figura 56: Modificación del fichero drivers/net/wireless/ath/ath9k/main.c**

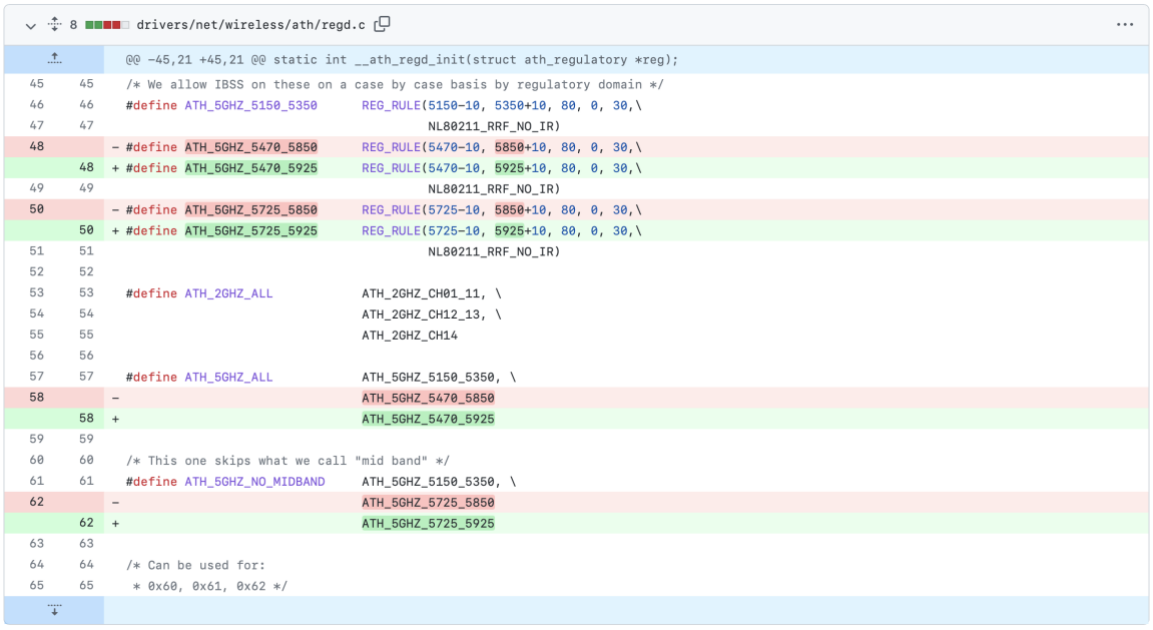


**Figura 57: Modificación del fichero drivers/net/Wireless/ath/regd.c**

## 4.2 Instalación de Vanetza

El primer paso para la instalación de Vanetza es la descarga del software del Github de sus desarrolladores [57] y su compilación.

### *4.2.1 Requisitos de Vanetza*

- Compilador C++ compatible con la C++11: los equipos que estamos utilizando tienen instalado GCC 5.4.0, que tiene soporte para C++11 y C++14, por lo que ha sido necesario actualizarla.

- CMake 3.1 o superior: la versión instalada en nuestros equipos es la 3.5.1, por lo que no ha sido necesario actualizarla

- Libboost 1.5.8 o superior: la versión instalada en nuestros equipos es la 1.5.8, por lo que no ha sido necesario actualizarla.

- GeographicLib 1.3.7 o superior: es una librería C++ que permite la conversión entre coordenadas cartesianas, geocéntricas, geográficas, UPS, MGRS y UTM. No había ninguna versión instalada, por lo que se ha instalado la versión 1.45-2 utilizando el siguiente comando:

```
sudo apt-get install libgeographic14 libgeographic-dev \
geographiclib-tools.
```

- Crypto++ 5.6.1 o superior: es una librería de clases C++ de esquemas criptográficos. No había ninguna versión instalada, por lo que se ha instalado la versión 5.6.1-9 utilizando el siguiente comando:

```
sudo apt-get install libcrypto++9v5 libcrypto++9v5-dbg libcrypto++-dev
```

### 4.2.2 Compilación de Vanetza

Una vez instalados los paquetes faltantes y descargado el software, se ha descomprimido el mismo en las carpetas `/home/gco/vanetza-master` y `/home/gco2/vanetza-master` de cada uno de los miniordenadores disponibles en el laboratorio. A continuación, accedemos al directorio donde hemos descomprimido el software y ejecutamos los siguientes comandos:

```
mkdir build && cd build
cmake ..
make -DBUILD_SOCKTAP=ON
```

La opción `-DBUILD_SOCKTAP=ON` compila Vanetza con la aplicación de ejemplo Socktap.

### 4.2.3 Socktap

Es una aplicación de ejemplo de cómo utilizar la API de Vanetza e implementa tres aplicaciones básicas:

- ca: envía mensajes CAM.
- hello: envía un mensaje simple BTP-B con payload `0xc0ffee`.
- benchmark: cuenta el número de mensajes, de cualquier tipo, recibidos y muestra la tasa de mensajes actual por segundo.

Algunas de las opciones que permite el comando `socktap` son:

- `-i (--interface)`: indica el interfaz de red a utilizar en nuestro equipo.

- `-p (--positioning)`: Socktap permite utilizar posicionamiento mediante gpsd, si hay un GPS conectado al sistema, o especificar una posición estática mediante los parámetros `--latitude` y `--longitude`.

- `--security`: Vanetza tiene, al contrario que OpenC2X, implementados mecanismos de seguridad opcionales.

- `-a (--applications)`: Elige la aplicación, si no se especifica envía CAMs periódicamente. La frecuencia de envío puede especificarse con la opción `--cam-interval` (por defecto 1000ms).

- `--print-tx-cam`: muestra por pantalla el contenido de los CAMs enviados.

- `--print-rx-cam`: muestra por pantalla el contenido de los CAMs recibidos.

La interfaz con la aplicación se realiza únicamente mediante consola, se ejecuta el comando con las opciones necesarias y la aplicación muestra por pantalla mensajes básicos, o como mucho el contenido de los CAM recibidos o enviados si se utiliza la opción correspondiente. No hay interfaz gráfica, o web como en el caso de OpenC2X, que permita ir variando opciones sobre la marcha, ni ficheros de configuración ni guarda logs de ningún tipo, ni de la propia aplicación ni de los servicios proporcionados por Vanetza.

Tampoco tiene todas las funcionalidades de una estación ITS-G5 completa ya que, por ejemplo, no implementa el DCC. Por lo tanto, la funcionalidad de la aplicación es un poco limitada para ser utilizada como sustitución de OpenC2X en nuestras estaciones ITS. Aun así, las librerías de Vanetza implementan de forma muy completa la pila de protocolos ETSI C-ITS y están muy bien desarrolladas, por lo que una vía de estudio futura podría ser la de construir nuestra propia aplicación sobre dichas librerías, lo cual queda fuera del objetivo de este Trabajo Fin de Máster.

## 4.3 Comunicación entre Vanezta y OpenC2X

Ya que habíamos realizado la instalación de Vanetza, se decidió utilizar la aplicación Socktap para comprobar la interoperabilidad de OpenC2X con otros softwares diferentes. Nuestro objetivo era comprobar si nuestros equipos eran capaces de comunicarse ejecutando uno Vanetza y el otro OpenC2X. Para ello ejecutamos socktap mediante el siguiente comando:

```
sudo socktap -i ocb0 -p static --print-rx-cam
```

El comando ejecuta socktap utilizando la aplicación de envío de CAMs, la interfaz ocb0 y posicionamiento estático. Con la última opción muestra los CAMs recibidos por pantalla. El resultado obtenido es el siguiente mensaje de error:

```
Enable application 'ca'...
received packet from 44:6d:57:11:3c:ad (99 bytes)
Router dropped packet because of Decap_Unsuccessful_Strict (6)
received packet from 44:6d:57:11:3c:ad (99 bytes)
Router dropped packet because of Decap_Unsuccessful_Strict (6)
received packet from 44:6d:57:11:3c:ad (99 bytes)
Router dropped packet because of Decap_Unsuccessful_Strict (6)
received packet from 44:6d:57:11:3c:ad (99 bytes)
```

Para intentar averiguar de dónde viene el problema se decide analizar el tráfico con la herramienta Wireshark. En la primera captura, mostrada en la Figura 58, vemos que los paquetes enviados por Vanezta tienen un tamaño de 187 bytes frente a los 99 bytes de los enviados por OpenC2X. Analizando la cabecera vemos que es debido a que Vanetza activa por defecto el envío seguro de paquetes. Otro detalle que se observa es que los CAM de OpenC2X vienen indicados como CAMv1, mientras que los de Vanetza vienen indicados como CAM. En breve veremos por qué.

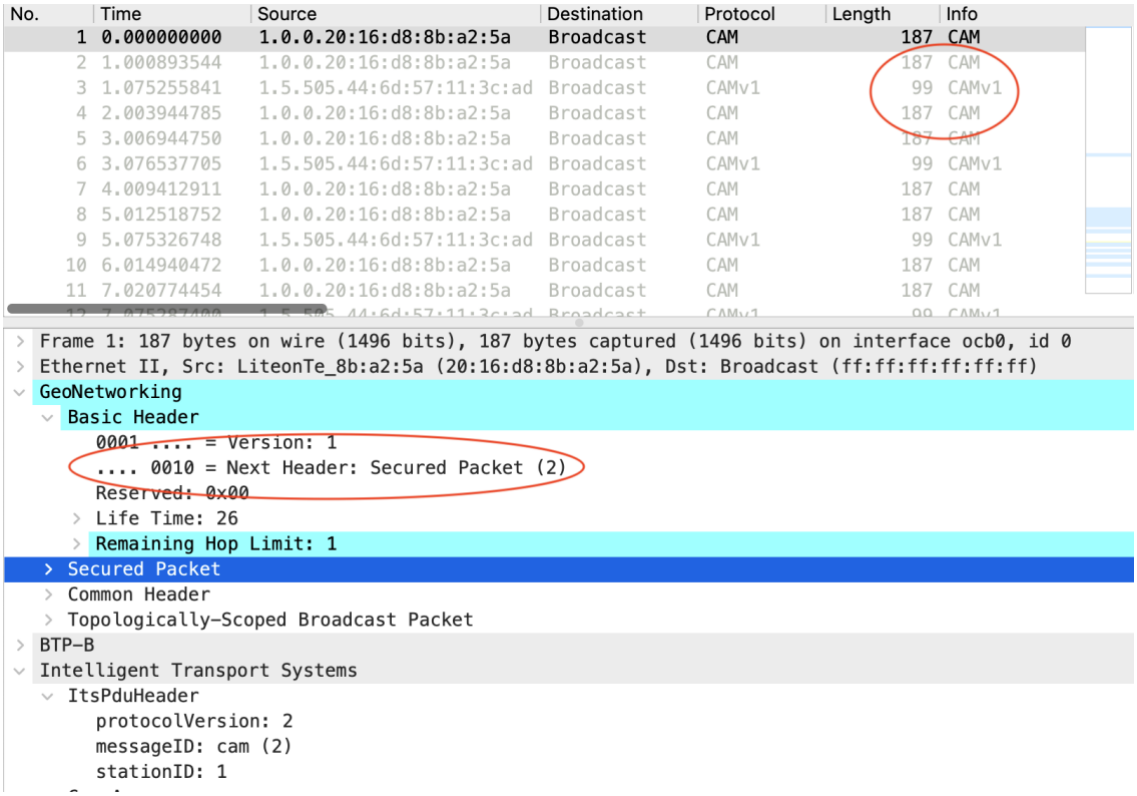


**Figura 58: Captura de Wireshark de paquetes Vanetza seguros**

Para desactivar el envío de paquetes seguros debemos especificárselo a Vanetza cuando ejecutamos socktap añadiendo la opción `--security none` al comando:

```
sudo socktap -i ocb0 -p static --security none --print-rx-cam
```

### 4.3.1 Actualización de versión de protocolos ITS en OpenC2X

Una vez hecho esto, ejecutamos de nuevo socktap y el mensaje de error que obtuvimos fue otro:

```
Enable application 'ca'…
received packet from 44:6d:57:11:3c:ad (99 bytes)
Router dropped packet because of ITS_Protocol_Version (4)
received packet from 44:6d:57:11:3c:ad (99 bytes)
Router dropped packet because of ITS_Protocol_Version (4)
received packet from 44:6d:57:11:3c:ad (99 bytes)
Router dropped packet because of ITS_Protocol_Version (4)
received packet from 44:6d:57:11:3c:ad (99 bytes)
```

Lo primero que se comprobó es la versión del protocolo CAM ya que, dado que OpenC2X tiene varios años, es probable que hubiera cambiado. Tanto OpenC2X como Vanezta forman los mensajes a partir de los ASN.1 incluidos en el estándar ETSI EN 302 637-2.

En el directorio `OpenC2X/common/ASN.1` se encuentran los ASN.1 que utiliza OpenC2X. El ASN.1 utilizado es el correspondiente a la v1.3.2 del estándar, que especifica que el valor que debe tomar Protocol_Version es 1 [58]. La versión actual del estándar es la v1.4.1, que especifica el valor 2 [59]. Para hacer efectivo el cambio se modificó el fichero `OpenC2X/common/ASN.1/its_facilities_pdu_all.asn` que contiene los ASN.1 de los mensajes CAM y DENM, así como el Common Data Dictionary del que dependen ambos protocolos, como se muestra en la Figura 59.

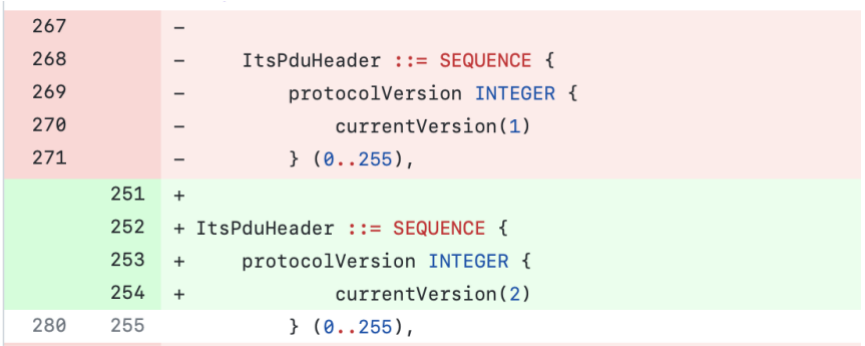


**Figura 59: Modificación del fichero its_facilities_pdu_all.asn para actualizar versión de protocolo CAM y DEMN**

Tras realizar los cambios y recompilar OpenC2X se comprobó que, aunque Wireshark ya detectaba los paquetes de OpenC2X como CAM en vez de CAMv1, Vanetza seguía dando el mismo error. Por ello se revisó el código de Vanezta para averiguar en que condición se daba dicho error. Dicho mensaje se genera en las líneas 405 y 406 del fichero `vanetza/geonet/router.cpp` (Figura 60), cuando se comprueba si la versión del protocolo GeoNetworking del paquete recibido es igual a la configurada en Vanetza:


**Figura 60: Comprobación de la versión del protocolo GeoNetworking en vanetza/geonet/router.cpp**

OpenC2X, al no tener implementado GeoNetworking, rellena las cabeceras de los paquetes con valores fijos en el método fillGeoNetBTPheaderForCam de la clase SendToHardwareViaMAC (fichero `OpenC2X/dcc/src/SendToHardwareViaMAC.cpp`). Se modificó el código para que el valor del campo *Version* de la cabecera Basic Header se correspondiera con el valor indicado en la versión actual del protocolo, que es 1 [51]. Este cambio fue únicamente necesario para la realización de las pruebas de comunicación entre ambos softwares, pues una vez implementado el protocolo esta función de relleno de cabeceras desaparece.

### 4.3.2 Actualización de los ASN.1 de CAM y DEMN

Antes de conseguir una comunicación correcta entre Vanetza y OpenC2X ha sido necesario actualizar los ASN.1 de OpenC2X, ya que tanto Vanetza como OpenC2X eran incapaz de decodificar los mensajes del otro, como muestran las Figura 61 y Figura 62.

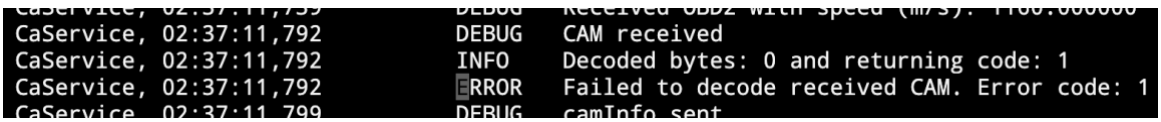


**Figura 61: Mensaje de error CAM de Vanetza recibido por OpenC2X**

```
Enable application 'ca'...
received packet from 44:6d:57:11:3c:ad (99 bytes)
CAM application received a packet with broken content
received packet from 44:6d:57:11:3c:ad (99 bytes)
CAM application received a packet with broken content
received packet from 44:6d:57:11:3c:ad (99 bytes)
```

**Figura 62: Mensaje de error CAM de OpenC2X recibido por Vanetza**

Analizando los paquetes intercambiados en Wireshark se observa, en la Figura 63, que Wireshark muestra una advertencia en los mensajes enviados por OpenC2X que hace sospechar que el mensaje no se está formando correctamente.

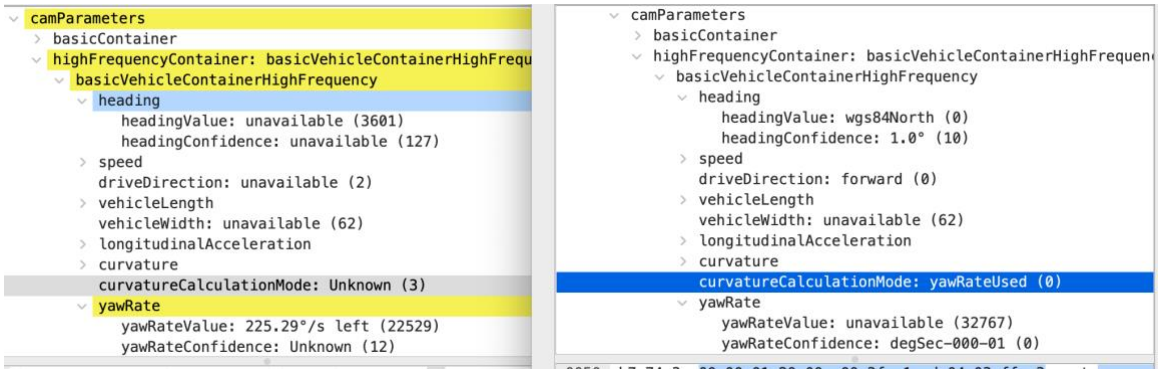


**Figura 63: Captura de CAM de OpenC2X (izquierda) y Vanetza (derecha)**

Los valores de *curvatureCalculationMode* y *yawRate* no parecen tener valores muy coherentes con los que Wireshark decodificaba cuando no habíamos cambiado la versión de protocolo ITS a pesar de no haber cambiado el código de generación del CAM, como se puede ver en la Figura 64.

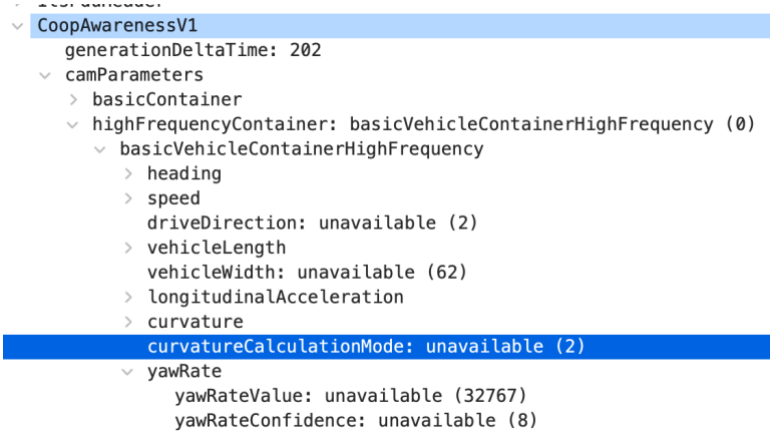


**Figura 64: CAMv1 decodificado por Wireshark**

Revisando de nuevo los módulos ASN.1 de CAM, DEMN y el Common Data Dictionary se comprobó que la codificación del parámetro *curvatureCalculationMode* ha cambiado, como muestra la Figura 65.

```
189     - CurvatureValue ::= INTEGER {straight(0), reciprocalOf1MeterRadiusToRight(-30000), reciprocalOf1MeterRadiusToLeft(30000),
            unavailable(30001)} (-30000..30001)
192     + CurvatureValue ::= INTEGER {straight(0), unavailable(1023)} (-1023..1023)
```

**Figura 65: Cambio de codificación del parámetro curvatureCalculationMode**

Por ello se han cambiado los ASN.1 de OpenC2X a las últimas versiones de CAM (v1.4.1), DENM (v1.3.1) y CDD (v1.3.1) disponibles en el gitlab de ETSI [60]. De esta manera nos aseguramos de que los mensajes CAM y DENM se envían conforme a los estándares actuales. Tras realizar el cambio por fin Vanetza y OpenC2X realizan el intercambio de CAMs correctamente.

```
received packet from 2c:d0:5a:bb:80:f5 (99 bytes)
CAM application received a packet with decodable content
Received CAM contains
  ITS PDU Header:
    Protocol Version: 2
    Message ID: 2
    Station ID: 122111111
  CoopAwarensess:
    Generation Delta Time: 201
    Basic Container:
      Station Type: 5
      Reference Position:
        Longitude: -47047230
        Latitude: 416600730
        Semi Major Orientation: 0
        Semi Major Confidence: 0
        Semi Minor Confidence: 0
        Altitude [Confidence]: 699 [15]
    High Frequency Container [Basic Vehicle]:
      Heading [Confidence]: 3601 [127]
      Speed [Confidence]: 1260 [127]
      Drive Direction: 2
      Longitudinal Acceleration: 161
      Vehicle Length [Confidence Indication]: 1023 [4]
      Vehicle Width: 62
      Curvature [Confidence]: 1023 [7]
      Curvature Calculation Mode: 2
      Yaw Rate [Confidence]: 32767 [8]
    Low Frequency Container: no
```

**Figura 66: Recepción correcta de CAM enviado por OpenC2X en Vanetza**

```
CaService, 00:49:48,451          DEBUG   CAM received
CaService, 00:49:48,451          INFO    Decoded bytes: 41 and returning code: 0
CaService, 00:49:48,451          INFO    Forward incoming CAM 1 to LDM
```

**Figura 67: Recepción correcta de CAM enviado por Vanetza en OpenC2X**

# 5

# Implementación del protocolo GeoNetworking en OpenC2X

Una vez conseguida la comunicación entre ambos softwares V2X, lo que nos ha permitido ir introduciendo ya alguna mejora en OpenC2X, se procedió a la implementación final del protocolo GeoNetworking en OpenC2X.

Se ha intentado respetar al máximo la filosofía de OpenC2X sobre cómo se realiza el intercambio de información entre las diferentes entidades, explicado con detalle en el Trabajo Fin de Grado de Daniel [13].

## 5.1 Integración del servicio GeoNetworking en la arquitectura de comunicación de OpenC2X

Como indica Pilar en su Trabajo Fin de Grado, OpenC2X implementa de manera independiente cada uno de los diferentes protocolos (CAM, DENM, BTP) y servicios como el GPS y OBD-II. Todos ellos tienen su propio directorio dentro de la estructura del proyecto, con el código fuente en el subdirectorio `src` y los ficheros de configuración en el subdirectorio `config`. [9]

Siguiendo este esquema se ha creado un directorio dentro del directorio principal de OpenC2X, llamado `geonetworking`, que contiene los subdirectorios `geonetworking/src`, que contiene los ficheros de cabecera y código fuente de las diferentes clases que forman el servicio GeoNetworking implementado, y `geonetworking/config`, que contiene el fichero de configuración del servicio y los ficheros de configuración de los logs a generar durante la ejecución del mismo.

La incorporación del servicio dentro de la arquitectura de comunicación se ha diseñado añadiendo el servicio entre los existentes de BTP y el DCC, de manera que los

mensajes que antes intercambiaban directamente entre ellos pasan ahora por el nuevo servicio. Además, el servicio de GeoNetworking necesita recibir datos del GPS y de OBD2, para obtener la ubicación y la velocidad de la estación ITS. Por ello ha hecho necesario cambiar la arquitectura de comunicación de OpenC2X, como se muestra en la Figura 68.

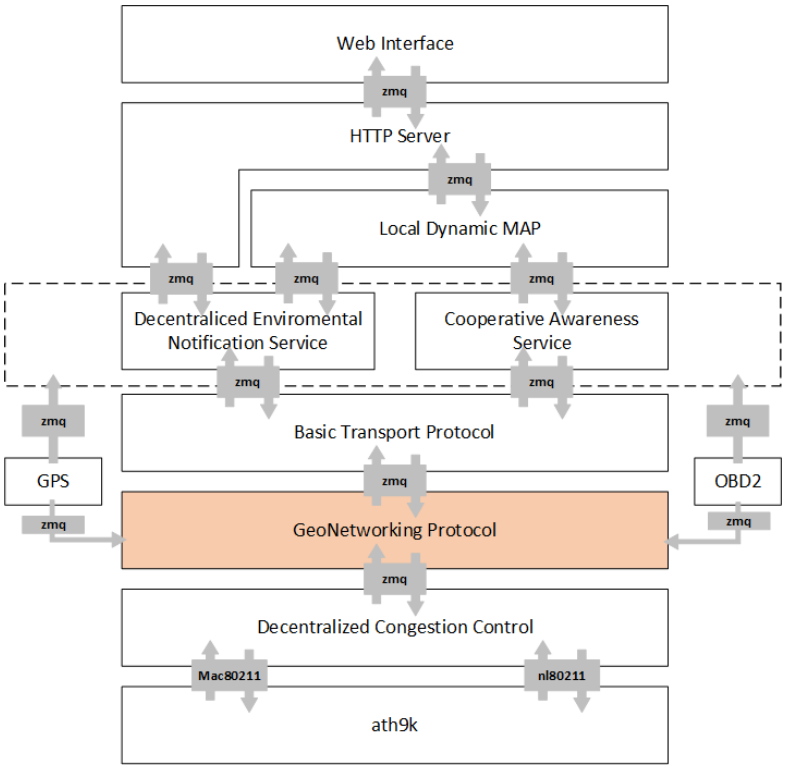


**Figura 68: Nueva arquitectura de comunicación OpenC2X**

También se han añadido dos nuevos puertos de comunicación para acomodar el servicio GeoNetworking, como se muestra en la Figura 69.

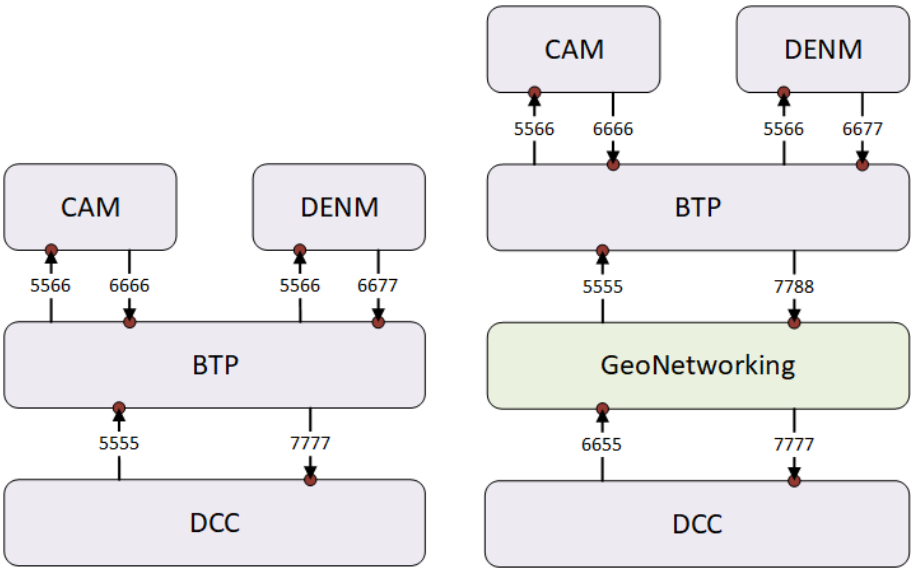


**Figura 69: Número de puerto de los sockets ZMQ antes y después de incluir GeoNetworking**

Ahora el BTP envía los datos al servicio de GeoNetworking al puerto 7788, no utilizado hasta ahora, y los sigue recibiendo por el 5555, pero ahora desde GeoNetworking y no directamente desde el módulo del DCC.

El servicio GeoNetworking, a su vez envía al DCC sus datos al puerto 7777, y los recibe en el puerto 6655 desde dicho servicio.

No se muestran el resto de los módulos por simplificación, pues mantienen el mismo esquema de puertos ya mostrado en el Trabajo fin de Grado de Daniel [13]. La única diferencia es que el servicio GeoNetworking escucha, al igual que el servicio de CAM y DENM, en los puertos 3333 y 2222 también para recibir los datos del GPS y el OBD2 como muestra la Figura 70.

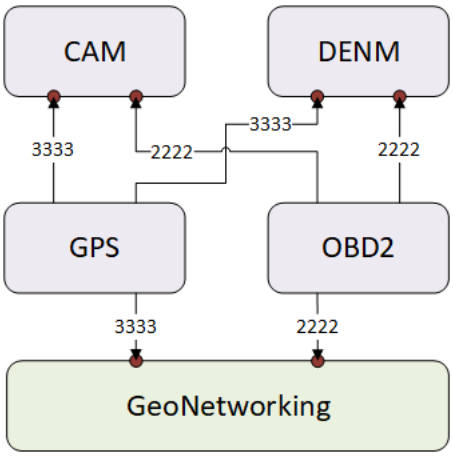


**Figura 70: Puertos de los servicios GPS y OBD2**

## 5.2 Implementación de las primitivas de servicios de datos

Al igual que en la implementación del protocolo BTP realizada por Daniel, se optó por implementar las primitivas de servicios de datos GN-Data.request y GN-Data.indication utilizando la herramienta Protocol Buffers de Google [61]. Esta herramienta permite definir estructuras de datos personalizadas, en ficheros con extensión *.proto*, que se compilan en nuestro caso a clases C++ que permiten su manipulación y serialización en cadenas *string* que intercambiamos entre nuestros servicios dentro de OpenC2X mediante el uso de sockets ZMQ.

Las primitivas GN-Data.request y GN-Data.indication se han implementado en los ficheros `common/buffers/GnRequest.proto` y `common/buffers/GnRequest.proto`, cuyo código se puede observar en la Figura 71.

```
package GeoNetworking;

import "data.proto";
import "area.proto";
import "enums.proto";

message GN_REQUEST {
    required Upper_Protocol upperProtocol = 1;
    required Transport_Type transportType = 2;
    oneof destination {
        int64 geoAddress = 3;
        DestinationArea geoArea = 4;
    }
    required int32 comProfile = 5;
    optional int32 SecurityProfile = 6;
    optional int64 itsAidLength = 7;
    optional int64 itsAid = 8;
    optional int64 secPermissionsLength = 9;
    optional int64 secPermissions = 10;
    optional int64 secContextInfo = 11;
    optional int64 secTargetIdListLength = 12;
    optional int64 secTargetIdList = 13;
    optional int64 maxPackLifetime = 14;
    optional int64 repInterval = 15;
    optional int64 maxRepTime = 16;
    optional int64 maxHopLimit = 17;
    required Traffic_Class trafficClass = 18;
    required int32 length = 19;
    required dataPackage.DATA data = 20;
}
```

```
package GeoNetworking;

import "area.proto";
import "enums.proto";
import "PositionVector.proto";

message GN_INDICATION {
    required Upper_Protocol upperProtocol = 1;
    required Transport_Type transportType = 2;
    oneof destination {
        int64 geoAddress = 3;
        DestinationArea geoArea = 4;
    }
    optional PositionVector sourcePV = 5;
    optional int64 secReport = 6;
    optional int64 certificateID = 7;
    optional int64 itsAidLength = 8;
    optional int64 itsAid = 9;
    optional int64 secPermissionsLength = 10;
    optional int64 secPermissions = 11;
    required Traffic_Class trafficClass = 12;
    optional int64 remainingPackLifetime = 13;
    optional int64 remainingHopLimit = 14;
    required int32 length = 15;
    required bytes pduForTransportProtocol = 16;
}
```

**Figura 71: Ficheros common/buffers/GnRequest.proto y common/buffers/GnIndication.proto**

Aunque no está implementada la capa transversal de seguridad de la pila se optado por incluir, como opcionales, los parámetros de las primitivas relacionadas con ella: *itsAid*, *itsAidLength*, *secPermissions*, *SecPermissionsLength*, *secContextInfo*, *SecTargetIdList* *secTargetIdListLength*, *secReport y certificateID*. De esta manera las primitivas quedan preparadas para una posible futura implementación de la capa transversal de seguridad.

Los campos geoAddress y geoArea describen la dirección GeoNetworking para el caso de los paquetes GUC o el área geográfica de destino en los paquetes GBC/GAC y el estándar los agrupa en un único parámetro llamado *Destination* de las primitivas, por lo que se ha usado el tipo de campo *oneof* de protobuffers que permite agrupar varios campos diferentes (tanto tipo como nombre) de los cuales como mucho se puede utilizar uno en cada mensaje.

Para el campo geoArea se ha definido una estructura de datos, denominada mensaje en el lenguaje protocol buffers, en el fichero `common/buffers/area.proto`. Esta estructura

agrupa todos los datos necesarios para definir el área de destino: forma geométrica, posición y dimensiones de esta. De esta manera se importa en los mensajes que generan las primitivas como un nuevo tipo de datos de manera similar a crear una *struct* o clase en C++ en un fichero de cabecera e incluirlo en un programa. El fichero `common/buffers/area.proto` puede consultarse en la Figura 72.

```
1   package GeoNetworking;
2
3   enum Area_Shape {
4       Circular = 0;
5       Rectangular = 1;
6       Ellipsoidal = 2;
7   }
8
9   message DestinationArea {
10      required Area_Shape shape = 1;
11      required int32 geoAreaLatitude = 2;
12      required int32 geoAreaLongitude = 3;
13      optional uint32 distanceA = 4;
14      optional uint32 distanceB = 5;
15      optional uint32 angle = 6;
16  }
```

**Figura 72: Fichero common/buffers/area.proto**

Los parámetros *Packet transport type* y *Traffic class* se corresponden con los parámetros *GN Packet Transport type* y *GN Traffic class* de las primitivas del protocolo BTP, y son pasadas de forma transparente de la entidad de la capa *facilities* hasta GeoNetworking, por lo que se ha decidido definirlas junto a *Upper Protocol Entity* como *enums* el fichero `common/buffers/enums.proto`, Figura 73, e importar dicho fichero tanto en las primitivas de GeoNetworking como en las de BTP.

```
package GeoNetworking;

// Upper protocol for GN-DATA and BTP Type for BTP-DATA service primitives
enum Upper_Protocol {
    ANY = 0;        // Unspecified
    BTP_A = 1;      // Basic Transport Protocol (interactive packet transport BTP-A)
    BTP_B = 2;      // Basic Transport Protocol (non interactive packet transport BTP-B)
    GN6 = 3;        // Geonetworking to IPv6 adaptation sub-layer
}

// Packet Transport Type parameter of the GN-DATA/BTP-DATA service primitives
enum Transport_Type {
    GeoUnicast = 1;
    SHB = 2;
    TSB = 3;
    GeoBroadcast = 4;
    GeoAnycast = 5;
}

// Traffic class parameter of the GN-DATA/BTP-DATA service primitives
enum Traffic_Class {
    AC_VO = 0;
    AC_VI = 1;
    AC_BE = 2;
    AC_BK = 3;
}
```

**Figura 73: Fichero common/buffers/enums.proto**

También se han definido un mensaje para el *Source position vector* de la primitiva GN-Data.indication, en el fichero `common/buffers/PositionVector.proto`, mostrado en la Figura 74.

```
package GeoNetworking;

message PositionVector {
    optional int64 gnAddr = 1;
    optional uint32 timeStamp = 2;
    required int32 latitude = 3;
    required int32 longitude = 4;
    optional uint32 posAcuIndicator = 5;
    optional int32 speed = 6;
    optional uint32 heading = 7;
}
```

**Figura 74: Fichero common/buffers/PositionVector.proto**

La SDU proveniente de BTP se sigue enviando dentro de la clase DATA ya definida en OpenC2X como se venía haciendo hasta ahora, aunque, como se verá más adelante, no se incluirán todos los parámetros de control destinados al DCC que se incluían hasta ahora, pues ya son parámetros que se pasan en otros campos de la primitiva GN-Data.request. La primitiva GN-Data.indication no usa la clase DATA y envía a BTP únicamente la SDU dentro del campo *pduForTransportProtocol*.

No se ha implementado la primitiva GN-Data.confirm, por lo que no se devuelve ninguna confirmación de si la petición se ha procesado correctamente o no.

## 5.3 Configuración del servicio GeoNetworking

El anexo H del estándar ETSI EN 302 636-4-1 [51] define una serie de parámetros, algunos de los cuales ya hemos visto, que denomina constantes de protocolo. Aunque el estándar también ofrece una representación de dichas constantes como una MIB en ASN.1, para nuestra implementación se optado por seguir el modelo de configuración del resto de módulos de OpenC2X y agrupar las constantes de protocolo en un fichero de configuración, `geonetworking/config/config.xml`, mostrado en la Figura 75.

```
<!-- GeoNetworking protocol configuration constants (MIB) -->
 <geonet>
    <itsGnLocalGnAddr>1</itsGnLocalGnAddr>
    <!-- itsGnLocalAddrConfMethod: AUTO (0), MANAGED (1), ANONYMOUS (2): -->
    <itsGnLocalAddrConfMethod>1</itsGnLocalAddrConfMethod>
    <itsGnProtocolVersion>1</itsGnProtocolVersion>
    <!-- itsGnIsMobile: Stationary (0), Mobile (1) -->
    <itsGnIsMobile>1></itsGnIsMobile>
    <!-- itsGnIfType: Unspecified (0), ITS-G5(1), LTE-V2X (2) -->
    <itsGnIfType>1</itsGnIfType>
    <!-- update frecuency of Ego Position Vector (1/ms) (Standard Default value = 1000)-->
    <itsGnMinUpdateFrequencyEPV>1000</itsGnMinUpdateFrequencyEPV>
    <itsGnPaiInterval>80</itsGnPaiInterval>
    <itsGnMaxSduSize>1398</itsGnMaxSduSize>
    <itsGnMaxGeoNetworkingHeaderSize>88</itsGnMaxGeoNetworkingHeaderSize>
    <itsGnLifetimeLocTE>20</itsGnLifetimeLocTE>
    <!-- GN security: DISABLED (0), ENABLED (1) -->
    <itsGnSecurity>0</itsGnSecurity>
    <!-- security related parameter, SECURITY NOT IMPLEMENTED!-->
    <itsGnSnDecapResultHandling>1</itsGnSnDecapResultHandling>
    <!-- location service parameters, TO-DO (using defaults)-->
    <itsGnLocationServiceMaxRetrans>10</itsGnLocationServiceMaxRetrans>
    <itsGnLocationServiceRetransmitTimer>1000</itsGnLocationServiceRetransmitTimer>
    <itsGnLocationServicePacketBufferSize>1024</itsGnLocationServicePacketBufferSize>
    <!-- beacon parameters-->
    <itsGnBeaconServiceRetransmitTimer>3000</itsGnBeaconServiceRetransmitTimer>
    <itsGnBeaconServiceMaxJitter>750</itsGnBeaconServiceMaxJitter>
    <!-- Maximum hop number -->
    <itsGnDefaultHopLimit>10</itsGnDefaultHopLimit>
    <!-- Duplicated Packet List -->
    <itsGnDPLLength>8</itsGnDPLLength>
    <itsGnMaxPacketLifetime>600</itsGnMaxPacketLifetime>
    <itsGnDefaultPacketLifetime>60</itsGnDefaultPacketLifetime>
    <itsGnMaxPacketDataRate>100</itsGnMaxPacketDataRate>
    <itsGnMaxPacketDataRateEmaBeta>90</itsGnMaxPacketDataRateEmaBeta>
    <!-- Maximum size of the geographical area for a GBC and GAC packet [km^2 ]-->
    <itsGnMaxGeoAreaSize>10</itsGnMaxGeoAreaSize>
    <itsGnMinPacketRepetitionInterval>100</itsGnMinPacketRepetitionInterval>
    <!-- Default forwarding algorithm outside target area: Greedy (1) -->
    <itsGnNonAreaForwardingAlgorithm>1</itsGnNonAreaForwardingAlgorithm>
    <!-- Default forwarding algorithm inside target area: CBF (2) -->
    <itsGnAreaForwardingAlgorithm>2</itsGnAreaForwardingAlgorithm>
    <!-- Minimum buffer time in CBF packet buffer [ms] -->
    <itsGnCbfMinTime>1</itsGnCbfMinTime>
    <!-- Maximum buffer time in CBF packet buffer [ms] -->
    <itsGnCbfMaxTime>100</itsGnCbfMaxTime>
    <!-- theoretical maximum communication range [m] -->
    <itsGnDefaultMaxCommunicationRange>1000</itsGnDefaultMaxCommunicationRange>
    <!-- Default threshold angle for advanced GeoBroadcast algorithm in clause F.4 [degrees]-->
    <itsGnBroadcastCBFDefSectorAngle>30</itsGnBroadcastCBFDefSectorAngle>
    <!-- Packet buffers size [Ko] -->
    <itsGnUcForwardingPacketBufferSize>256</itsGnUcForwardingPacketBufferSize>
    <itsGnBcForwardingPacketBufferSize>1024</itsGnBcForwardingPacketBufferSize>
    <itsGnCbfPacketBufferSize>256</itsGnCbfPacketBufferSize>
    <!-- Forwarding: Default traffic class -->
    <itsGnDefaultTrafficClass>0</itsGnDefaultTrafficClass>
 </geonet>
```

**Figura 75: Fichero de configuración geonetworking/config/config.xml**

La función *loadConfigXML(const std::string& filename)*, en el fichero de cabecera del servicio de GeoNetworking, `geonetworking/src/geonetservice.h`, obtiene del fichero de configuración las constantes de protocolo y las almacena en una estructura para que el servicio las utilice. Aunque en el fichero de configuración se han añadido todas las constantes de protocolo que define el estándar, el servicio actualmente sólo obtiene las que necesita para su funcionamiento, como muestra la Figura 72.

```
45  struct GeoNetworkServiceConfig
46  {
47      int mItsGnLocalGnAddr;
48      int mItsGnLocalAddrConfMethod;
49      int mItsGnProtocolVersion;
50      float mItsGnMinUpdateFrequencyEPV;
51      int mItsGnPaiInterval;
52      int mItsGnMaxGeoNetworkingHeaderSize;
53      bool mItsGnSecurity;
54      int mItsGnMaxPacketLifetime;
55      int mItsGnDefaultPacketLifetime;
56      int mItsGnDefaultHopLimit;
57      int mItsGnDefaultTrafficClass;
58      bool mItsGnIsMobile;
59
60      void loadConfigXML(const std::string &filename)
61      {
62          boost::property_tree::ptree pt;
63          read_xml(filename, pt);
64
65          mItsGnLocalGnAddr = pt.get("geonet.itsGnLocalGnAddr", 1);
66          mItsGnLocalAddrConfMethod = pt.get("geonet.itsGnLocalAddrConfMethod", 1);
67          mItsGnProtocolVersion = pt.get("geonet.itsGnProtocolVersion", 1);
68          mItsGnMinUpdateFrequencyEPV = pt.get("geonet.itsGnMinUpdateFrequencyEPV", 1000);
69          mItsGnPaiInterval = pt.get("geonet.itsGnPaiInterval", 80);
70          mItsGnMaxGeoNetworkingHeaderSize = pt.get("geonet.itsGnMaxGeoNetworkingHeaderSize", 88);
71          mItsGnSecurity = pt.get("geonet.itsGnSecurity", false);
72          mItsGnMaxPacketLifetime = pt.get("geonet.itsGnMaxPacketLifetime", 600);
73          mItsGnDefaultPacketLifetime = pt.get("geonet.itsGnDefaultPacketLifetime", 60);
74          mItsGnDefaultHopLimit = pt.get("geonet.itsGnDefaultHopLimit", 10);
75          mItsGnDefaultTrafficClass = pt.get("geonet.itsGnDefaultTrafficClass", 0);
76          mItsGnIsMobile = pt.get("geonet.itsGnIsMobile", true);
77      }
78  };
```

**Figura 76: Obtención de las constantes de protocolo en geonetworking/src/geonetservice.h**

OpenC2X también tiene un fichero global con parámetros de configuración comunes para todos los módulos, `common/config/config.xml`, en él se ha añadido una variable que identifique el tipo de estación ITS. Está acompañado del fichero de cabecera, `common/config/config.h`, que también contiene una función *loadConfigXML* que lee los parámetros de él y además obtiene la MAC de la interfaz de red, necesaria para crear posteriormente la dirección GeoNetworking

## 5.4 Clase GeoNetService

La clase GeoNetService es la clase principal del servicio de GeoNetworking y está formada por un fichero de cabecera, `geonetservice.h`, y un fichero con la definición de los métodos declarados en el fichero de cabecera y una función *main()* que se encarga de iniciar la ejecución del servicio, llamado `geonetservice.cpp`.

El constructor de la clase, mostrado en la Figura 77, se encarga de crear los sockets ZMQ (*CommunicationReceivers* y *CommunicationSenders*) para comunicarnos con el resto de las entidades de OpenC2X (líneas 61 a 68).

Otra de las funciones principales del constructor es crear la dirección GeoNetworking de la estación. OpenC2X no tiene una entidad de gestión para la capa de red y transporte ITS, por lo que en esta primera implementación se ha optado por generar la dirección en el constructor de la clase (líneas 53 a 55) a partir de la MAC de la interfaz de red y el tipo de estación ITS, obtenidas del fichero de configuración global de OpenC2X.

Por último, se encarga de crear el temporizador para ejecutar las actualizaciones el Ego Position Vector.

```
39  GeoNetService::GeoNetService(GeoNetworkServiceConfig &config, string globalConfig, string loggingConf, string statisticConf)
40  {
41      try
42      {
43          mGlobalConfig.loadConfigXML(globalConfig);
44      }
45      catch (std::exception &e)
46      {
47          cerr << "Error while loading the global config.xml: " << e.what() << endl
48               << flush;
49      }
50
51      mConfig = config;
52
53      // Create GeoNetworking address using interface MAC from global OpenC2X configuration
54      GnAddr.setStationType(mGlobalConfig.mStationType);
55      GnAddr.setMID(mGlobalConfig.mMac);
56      egoPV.setGnAddr(GnAddr);
57
58      mLogger = new LoggingUtility("GeoNetService", mGlobalConfig.mExpNo, loggingConf, statisticConf);
59
60      // Creation of ZMQ sockets for sending/receiving data from other services
61      mReceiverFromBtp = new CommunicationReceiver("GeoNetService", "7788", "", mGlobalConfig.mExpNo, loggingConf, statisticConf);
62      mReceiverFromDcc = new CommunicationReceiver("GeoNetService", "6655", "", mGlobalConfig.mExpNo, loggingConf, statisticConf);
63
64      mReceiverGps = new CommunicationReceiver("CaService", "3333", "GPS", mGlobalConfig.mExpNo, loggingConf, statisticConf);
65      mReceiverObd2 = new CommunicationReceiver("CaService", "2222", "OBD2", mGlobalConfig.mExpNo, loggingConf, statisticConf);
66
67      mSenderToDcc = new CommunicationSender("GeoNetService", "7777", mGlobalConfig.mExpNo, loggingConf, statisticConf);
68      mSenderToBtp = new CommunicationSender("GeoNetService", "5555", mGlobalConfig.mExpNo, loggingConf, statisticConf);
69
70      // Creation of timer for Ego Position Vector updates (T = 1/itsGnMinUpdateFrecuencyEPV [ms])
71      mTimer = new boost::asio::deadline_timer(mIoService, boost::posix_time::millisec(1 / mConfig.mItsGnMinUpdateFrequencyEPV));
72      mTimer->async_wait(boost::bind(&GeoNetService::updatePosition, this, boost::asio::placeholders::error));
73  }
```

**Figura 77: Constructor de la clase GeoNetService**

La función *init()* se encarga de la creación de los hilos que ejecutan las funciones de envío y recepción de mensajes y pone en marcha el temporizador de actualización del Ego Position Vector (línea 104).

```
97   void GeoNetService::init()
98   {
99       mThreadReceiveFromBtp = new boost::thread(&GeoNetService::receiveFromBtpSendToDcc, this);
100      mThreadReceiveFromDcc = new boost::thread(&GeoNetService::receiveFromDcc, this);
101      mThreadGpsDataReceive = new boost::thread(&GeoNetService::receiveGpsData, this);
102      mThreadObd2DataReceive = new boost::thread(&GeoNetService::receiveObd2Data, this);
103
104      mIoService.run();
105  }
```

**Figura 78: Función *init()***

Las funciones *receiveGpsData()* y *receiveObd2Data()* son exactamente las mismas que las homónimas de los servicios CAM y DENM.

La función *main()*, mostrada en la Figura 79, es la función de entrada al programa, carga los datos del fichero de configuración global de OpenC2X, crea la clase y a continuación llama a la función *init()* para iniciar la ejecución del servicio GeoNetworking.

```
745  int main(int argc, const char *argv[])
746  {
747      if (argc != 5)
748      {
749          fprintf(stderr, "missing arguments: %s <globalConfig.xml>\
750                                              <geoNetworkConfig.xml>\
751                                              <logging.conf>\
752                                              <statistics.conf> \n", argv[0]);
753          exit(1);
754      }
755
756      GeoNetworkServiceConfig config;
757      try
758      {
759          config.loadConfigXML(argv[2]);
760      }
761      catch (std::exception &e)
762      {
763          cerr << "Error while loading config.xml: " << e.what() << endl
764               << flush;
765          return EXIT_FAILURE;
766      }
767
768      GeoNetService geonet(config, argv[1], argv[3], argv[4]);
769      geonet.init();
```

**Figura 79: Función *main()***

El resto de métodos de la clase se explican en las siguientes secciones, cuando se explique la implementación del envío y recepción de paquetes usando el servicio GeoNetworking.

## 5.5 Direcciones GeoNetworking y vectores de posición

Las direcciones GeoNetworking y los vectores de posición son dos estructuras de datos imprescindibles para el funcionamiento del protocolo y cada una representa un conjunto de datos de diferentes tipos agrupados sobre las que se realizan operaciones propias de cada uno. Por ello se ha decidido su implementación como clases.

La clase GeoNetAddr se utiliza para representar las direcciones GeoNetworking, ya sea la propia de las estaciones, o la de las estaciones con las que nos comunicamos. Los ficheros que componen el código de la clase son el fichero de cabecera `geonetworking/src/GeoNetAddr.h` y el fichero `geonetworking/src/GeoNetAddr.cpp`, que contiene el código fuente de las definiciones de los métodos de la clase.

Los atributos de la clase son los campos de la dirección GeoNetworking, como se puede observar en la Figura 80.

```
31  class GeoNetAddr
32  {
33  private:
34      uint8_t mIsManually = 0;
35      uint8_t mStationType = 0;
36      uint16_t mReserved = 0;
37      uint8_t mMID[IFHWADDRLEN] = {00,12,34,56,67,78};
```

**Figura 80: Atributos de la clase GeoNetAddr**

Se han definido 3 constructores, mostrados en la Figura 81. El constructor por defecto no asigna ningún valor nuevo a los atributos. El segundo constructor permite especificar el valor de todos los atributos al crear la clase. El último constructor crea una dirección GeoNetworking a partir del MID, en nuestro caso la MAC de la interfaz de red.

```
39  public:
40      // Default constructor, takes default parameters
41      GeoNetAddr() {}
42      // Constructor with all parameters
43      GeoNetAddr(uint8_t isManual, uint8_t stationType, uint8_t MID[]);
44      // Constructor with MID (LL_addr)
45      GeoNetAddr(uint8_t MID[]);
```

**Figura 81: Constructores de la clase GeoNetAddr**

En nuestra implementación actual sólo se utiliza el constructor por defecto y después asignamos el valor de nuestros atributos manualmente, pero se han creado los otros dos pues pueden ser útiles en un futuro si se desarrolla, por ejemplo, una entidad completa de gestión.

Además, se han creado los correspondientes métodos *getter* y *setter* para poder manipular los atributos de la clase una vez creado un objeto. Se ha sobrecargado el método *setter* del campo MID para permitir crearlo a partir de otro array o, si fuera necesario, de un objeto *string*. Eso puede dar flexibilidad a la hora de completar el protocolo en un futuro.

El resto de campos de la dirección GeoNetworking tienen longitud de 1, 5 y 10 bits, ocupando un total de 16 bits siguiendo la estructura ya mostrada en las Figura 19 y Figura 20. Se intentó usar el mecanismo de C++ denominado *bitfields*, para crear variables del número de bits necesarios que se pudieran luego asignar directamente cada uno a su campo en la estructura de la cabecera de los paquetes, pero además de variar el orden de los 2 bytes que forman los tres campos (fácilmente solucionable con la función *htons()* de C++) variaba el orden de los campos dentro de cada byte, por lo que se optó por almacenarlos en campos de mayor tamaño y hacer un método de la clase, *paramsAsInt()* (Figura 82) que devolviera un entero de 16 bits *unsigned* con el orden correcto mediante el uso de desplazamiento de bits.

```
46  uint16_t GeoNetAddr::paramsAsInt () {
47      uint16_t params = 0;
48
49      params = (mIsManually&0x01) << 15;      // IsManually is the MSB
50      params |= (mStationType&0x1F)<<10;      // Station Type is 5 bits long and next to isManually
51      params |= (mReserved&0x03FF);           // mReserved is 10 bits long (the LSBs)
52
53      return params;
54  }
```

**Figura 82: Método paramsAsInt()**

También se han creado dos operadores de comparación (== y !=), cuyo código se muestra en la Figura 83, para poder comparar de manera sencilla dos direcciones GeoNetworking, sin necesidad de comprobar todos los atributos uno a uno cada vez que sea necesario. Un ejemplo sería comprobar si ya tenemos en la tabla LocT los datos de una estación de la que recibimos un paquete.

```
60  bool GeoNetAddr::operator==(const GeoNetAddr& gnAddress) {
61      return gnAddress.mIsManually==mIsManually && gnAddress.mStationType==mStationType &&
62          gnAddress.mReserved==mReserved && std::equal(mMID, mMID+IFHWADDRLEN, gnAddress.mMID);
63  }
64
65  bool GeoNetAddr::operator!=(const GeoNetAddr& gnAddress) {
66      return !(*this==gnAddress);
67  }
```

**Figura 83: Operadores de comparación == y !=**

Por otro lado, se han creado dos clases que representan tanto Long Position Vectors como Short Position Vectors. Su código se encuentra en el fichero de cabecera `geonetworking/src/PositionVectors.h`, con la declaración de los atributos y métodos de la clase, y el fichero `geonetworking/src/PositionVectors.cpp`, que contiene un único método llamado *PAIspeed()*, Figura 84, que devuelve un número de 16 bits con la codificación necesaria de los campos PAI y Speed de manera similar a como lo hacía *paramsAsInt()* en la clase GeoNetAddr.

```
 9   uint16_t LongPositionVector::PAIspeed() {
10
11       uint16_t params = 0;
12       params = ((mSpeed*2) >> 1) & 0x7FFF ; // Speed is SIGNED 15 bits long
13       params |= (mPAI&0x0001) << 15;        // PAI is only a bit long
14       return params;
15   }
```

**Figura 84: Método PAIspeed()**

El resto de métodos son *getter* y *setter* de los diferentes atributos de la clase, que se declaran y definen *inline* en el fichero de cabecera.

Esta clase se usa también para almacenar el vector de posición local, o Ego Position Vector. Para la actualización del EGO Position Vector se utiliza el método *updatePosition()*, Figura 86, que obtiene los valores de latitud, longitud y heading (track) del GPS y la velocidad del OBD2.

Para la generación del *timestamp* se ha creado una función en el fichero `common/utility/Utils.cpp`, Figura 85, la cual utiliza la librería *Posix Time* del paquete de librerías *boost* para crear dos objetos tipo *Ptime* que definen el instante de tiempo actual y el 1 de enero de 2004 fijado por el estándar como referencia inicial y el tiempo pasado entre ellos. Estas funciones trabajan con tiempo UTC y no TAI como indica el estándar, por lo que se añaden los 37 segundos que hay actualmente de diferencia entre ambos para obtener el *timestamp* en TAI.

```
53  uint32_t Utils::timestamp() {
54
55      static const boost::posix_time::ptime epoch(boost::gregorian::date(2004,1,1));
56      boost::posix_time::ptime now(boost::posix_time::microsec_clock::universal_time());
57
58      boost::posix_time::time_duration td = now - epoch;
59      uint64_t diffTAI = td.total_milliseconds() + 37000; // UTC clock is 37 seconds behind TAI clock
60      uint32_t tst = diffTAI%static_cast<uint64_t>(pow(2,32));
61
62
63      return tst;
64  }
```

**Figura 85: Función timestamp()**

El parámetro PAI es un indicador de la exactitud del posicionamiento y se obtiene de los parámetros de error del GPS, *epx y epy*: si el mayor de esos errores es menor que la constante de protocolo *itsGnPaiInterval/2* PAI toma el valor *true*, caso contrario (o si el GPS no proporciona esos valores, como los nuestros), PAI toma el valor *false* (Figura 87).

```
624  void GeoNetService::updatePosition(const boost::system::error_code &ec)
625  {
626
627      mMutexLatestGps.lock();
628      egoPV.setTimestamp(Utils::timestamp());
629      egoPV.setLatitude(mLatestGps.latitude() * 10000000);   // * 10000000
630      egoPV.setLongitude(mLatestGps.longitude() * 10000000); // * 10000000
631
632      if (mLatestGps.has_track())
633      {
634          egoPV.setHeading(mLatestGps.track() * 10); // Heading in 0.1 degree from North
635      }
636      else
637      {
638          egoPV.setHeading(3600); // Heading value unavailable (ETSI TS 102 894-2 V1.3.1)
639      }
640
641      /* Position accuracy indicator of the GeoAdhoc router reference position Set to 1 (i.e. True) if
642       * the semiMajorConfidence of the PosConfidenceEllipse as specified in ETSI TS 102 894-2 [11] is
643       * smaller than the GN protocol constant itsGnPaiInterval/ 2 Set to 0 (i.e. False) otherwise
644       */
645
646      if (mLatestGps.has_epx() && mLatestGps.has_epy())…
659      else {
660          egoPV.setPAI(false);
661      }
662
663      mMutexLatestGps.unlock();
664
665      mMutexLatestObd2.lock();
666      egoPV.setSpeed(mLatestObd2.speed());
667      mMutexLatestObd2.unlock();
668      // Deactivate logging if using default protocol update frecuency or higher. TOO MANY MESSAGES!!!!
669      mLogger->logDebug("Updated Ego Position Vector with timestamp: " + to_string(egoPV.timestamp())
670                        + ", Longitude: " + to_string(egoPV.longitude())
671                        + ", latitude: "+ to_string(egoPV.latitude())
672                        + " and speed: " +to_string(egoPV.speed())) ;
673      scheduleNextUpdate();
674  }
```

**Figura 86: Método updatePosition**

```
646     if (mLatestGps.has_epx() && mLatestGps.has_epy())
647     {
648
649         if ((mLatestGps.epx() >= mLatestGps.epy() ? mLatestGps.epx() : mLatestGps.epy()) < mConfig.mItsGnPaiInterval /2)
650         {
651             egoPV.setPAI(true);
652         }
653         else
654         {
655
656             egoPV.setPAI(false);
657         }
658     }
659     else {
660         egoPV.setPAI(false);
661     }
```

**Figura 87: Parámetro PAI**

## 5.6 Envío de paquetes usando GeoNetworking

La integración del protocolo GeoNetworking en la arquitectura OpenC2X ha implicado modificaciones de mayor o menor importancia en los módulos correspondientes a todas las capas de la arquitectura, tanto los módulos que implementan los servicios CAM y DENM, como el BTP y el DCC.

Los mensajes CAM y DENM, que son los que permite enviar OpenC2X, utilizan los servicios de transporte SHB y GBC respectivamente. Por ese motivo, sólo se ha implementado el envío de esos dos tipos de paquetes en nuestro servicio GeoNetworking.

El envío de paquetes GeoNetworking se implementa en la función *receiveFromBtpSendToDcc()* de la clase GeoNetService (Figura 88). Esta función recibe del módulo BTP, a través del socket ZMQ *mReceiverFromBTP*, una primitiva GN-Data.request que deserializa en un objeto protobuf, *request*, de tipo GN_REQUEST (líneas 118 a 123).

Las cabeceras Basic Header y Common Header son comunes a cada tipo de paquete, y se han creado dos funciones para rellenar dichas cabeceras, *fillBasicHeader( )* y *fillCommonHeader ( ).*

```
106  void GeoNetService::receiveFromBtpSendToDcc()
107  {
108      string envelope;
109      string encodedData; // serialized DATA
110      string pduData;
111      dataPackage::DATA data; // deserialized DATA
112      GeoNetworking::GN_REQUEST request;
113      GeoNetHeaders::BasicHeader basicHeader;
114      GeoNetHeaders::CommonHeader commonHeader;
115      unsigned char GeoHeader[mConfig.mItsGnMaxGeoNetworkingHeaderSize];
116      unsigned int headerSize = 0;
117
118      while (1)
119      {
120          pair<string, string> received = mReceiverFromBtp->receive();
121          envelope = received.first;
122          encodedData = received.second;
123          request.ParseFromString(encodedData);
124          data = request.data();
125          string encodedTPDU = data.content();
126          GeoNetworking::DestinationArea destinationArea;
127
128          // Basic Header
129          basicHeader = fillBasicHeader(request);
130
131          // Common Header
132          uint16_t payloadSize = static_cast<uint16_t>(encodedTPDU.size());
133          mLogger->logInfo("Payload size: " + to_string(payloadSize));
134          commonHeader = fillCommonHeader(request, payloadSize);
```

**Figura 88: Función receiveFromBtpSendToDcc() - Recepción de primitiva GN-Data.request**

### *5.6.1 Función fillBasicHeader ( )*

La función *fillBasicHeader(const GeoNetworking::GN_REQUEST& incomingRequest)* recibe como parámetro por referencia el objeto *request* (Figura 88, línea 129) que representa la primitiva GN-Data.requets deserializada como vimos en el anterior apartado.

A continuación, completa los parámetros de la Basic Header con los parámetros indicados por el estándar. En el caso de los campos Next Header y Version, al ocupar 4 bits cada uno, almacena el valor de versión recibido, lo desplaza cuatro bits y añade el valor de Next Header. Actualmente Next Header siempre valdrá 1, indicando que el siguiente campo es la cabecera Common Header, al no estar implementada la entidad vertical de seguridad de la pila C-ITS.

```
257  GeoNetHeaders::BasicHeader GeoNetService::fillBasicHeader(const GeoNetworking::GN_REQUEST &incomingRequest)
258  {
259      GeoNetHeaders::BasicHeader tmpBasicHeader;
260      uint8_t version;
261      uint8_t basicNextHeader;
262
263      version = mConfig.mItsGnProtocolVersion;
264      tmpBasicHeader.reserved = 0;
265      // We try to use the GN-DATA.request parameter Security profile,
266      // if it doesn't exist we use the ITS-S (MIB) configuration value
267      if (incomingRequest.has_securityprofile())
268      {
269          if (incomingRequest.securityprofile() == 1)
270          {
271              // Security not yet implemented. Should never enter here by the moment.
272              basicNextHeader = 2;
273          }
274          else
275          {
276              basicNextHeader = 1;
277          }
278      }
279      else
280      {
281          basicNextHeader = mConfig.mItsGnSecurity;
282      }
283
284      tmpBasicHeader.versionAndNH = ((version << 4) | (basicNextHeader & 0x0F));
285
286      if (incomingRequest.has_maxpacklifetime())
287      {
288          tmpBasicHeader.lifetime = encodeLifeTime(incomingRequest.maxpacklifetime());
289      }
290      else
291      {
292          tmpBasicHeader.lifetime = encodeLifeTime(mConfig.mItsGnDefaultPacketLifetime);
293      }
294
295      // RHL = 1 if SHB, in other type of transport we use the GN-Data.request parameter
296      // Maximum Hop Limit (if exits) or the MIB value
297      if (incomingRequest.transporttype() == GeoNetworking::SHB)
298      {
299          tmpBasicHeader.remainingHopLimit = 1;
300      }
301      else
302      {
303          if (incomingRequest.has_maxhoplimit())
304          {
305              tmpBasicHeader.remainingHopLimit = incomingRequest.maxhoplimit();
306          }
307          else
308          {
309              tmpBasicHeader.remainingHopLimit = mConfig.mItsGnDefaultHopLimit;
310          }
311      }
312      return tmpBasicHeader;
313  }
```

**Figura 89: Función fillBasicHeader( )**

## *5.6.2 Función fillCommonHeader ( )*

De manera similar a la función vista en el apartado anterior, la función *fillCommonHeader(const GeoNetworking::GN_REQUEST& incomingRequest, uint16_t payload)* recibe como parámetro la primitiva GN-Data.requets deserializada y un entero con la longitud del payload de la petición, necesario para rellenar el correspondiente campo de la cabecera.

Con los datos de la petición, y siguiendo lo marcado por el estándar, completa los campos de la cabecera Common Header. Los campos SCF y Channel Offload de momento no se usan y se ha dejado su valor a *false*, pero con una implementación más completa del protocolo podrían ser necesarios. La línea 359, Figura 90, agrupa estos dos campos, junto con el Traffic Class ID de longitud 6 bits, en el campo Traffic Class de 8 bits que se incluye finalmente en la cabecera.

```
313 GeoNetHeaders::CommonHeader GeoNetService::fillCommonHeader(const GeoNetworking::GN_REQUEST &incomingRequest, uint16_t payload)
314 {
315     GeoNetHeaders::CommonHeader tmpCommonHeader;
316     uint8_t commonNextHeader;
317     uint8_t headerType;
318     uint8_t headerSubType;
319     uint8_t reserved = 0;
320     bool SCF;
321     bool channelOffLoad;
322     uint8_t trafficID;
323 
324     switch (incomingRequest.upperprotocol())
325     {
326     case GeoNetworking::ANY:
327         commonNextHeader = 0;
328         break;
329     case GeoNetworking::BTP_A:
330         commonNextHeader = 1;
331         break;
332     case GeoNetworking::BTP_B:
333         commonNextHeader = 2;
334         break;
335     case GeoNetworking::GN6:
336         commonNextHeader = 3;
337         break;
338     }
339 
340     // The 4 bit reserved field after "NH" field
341     reserved = 0;
342 
343     tmpCommonHeader.NHAndReserved = ((commonNextHeader << 4) | (reserved & 0x0F));
344 
345     // SCF and ChannelOffload could maybe be useful for something in the future
346     if (incomingRequest.has_trafficclass())
347     {
348         SCF = false;
349         channelOffLoad = false;
350         trafficID = incomingRequest.trafficclass();
351     }
352     else
353     {
354         SCF = false;
355         channelOffLoad = false;
356         trafficID = mConfig.mItsGnDefaultTrafficClass;
357     }
358 
359     tmpCommonHeader.TrafficClassParamsAndID = (((SCF & 0x01) << 7) | ((channelOffLoad & 0x01) << 6) | (trafficID & 0x3F));
```

**Figura 90: Función fillCommonHeader( ) 1/2**

Los campos HT y HST se rellenan según los valores ya indicados en la Tabla 10. En el caso de los tipos de transporte GBC/GAC el campo HST vendrá indicado en el campo *shape* del objeto *geoarea* de la petición. La Figura 91 muestra cómo se rellenan estos campos.

```
361     // Flags: bit 0 protocol constant itsGnIsMobile - bit 1 to 7 reserved (set to 0)
362     if (mConfig.mItsGnIsMobile)
363     {
364         tmpCommonHeader.flags = 128;
365     }
366     else
367     {
368         tmpCommonHeader.flags = 0;
369     }
370
371     tmpCommonHeader.payload = htons(payload);
372
373     // MHL = 1 if SHB or beacon, in other type of transport we use the GN-Data.request parameter
374     // Maximum Hop Limit (if exits) or the MIB value (or LS Request/Reply)
375     if (incomingRequest.transporttype() == GeoNetworking::SHB)
376     {
377         tmpCommonHeader.maxHop = 1;
378     }
379     else
380     {
381         if (incomingRequest.has_maxhoplimit())
382         {
383             tmpCommonHeader.maxHop = incomingRequest.maxhoplimit();
384         }
385         else
386         {
387             tmpCommonHeader.maxHop = mConfig.mItsGnDefaultHopLimit;
388         }
389     }
390
391     // For GBC/GAC packets HST field must be set to destination area shape
392     if (incomingRequest.has_geoarea()){
393         GeoNetworking::DestinationArea area = incomingRequest.geoarea();
394         headerSubType = static_cast<uint8_t>(area.shape());
395     }
396     else {
397         headerSubType = 0;
398     }
399     switch (incomingRequest.transporttype())
400     {
401     case GeoNetworking::GeoUnicast:
402         headerType = 2;
403         break;
404     case GeoNetworking::GeoAnycast:
405         headerType = 3;
406         break;
407     case GeoNetworking::GeoBroadcast:
408         headerType = 4;
409         break;
410     case GeoNetworking::TSB:
411         headerType = 5;
412         headerSubType = 1;
413         break;
414     case GeoNetworking::SHB:
415         headerType = 5;
416         break;
417     }
418
419     tmpCommonHeader.HTAndHST = ((headerType << 4) | (headerSubType & 0x0F));
420     // The last field of the common header called also reserved.
421     tmpCommonHeader.reserved = 0;
422
423     return tmpCommonHeader;
424 }
```

**Figura 91: Función fillCommonHeader( ) 2/2**

### *5.6.3 Envío de mensajes CAM*

Los mensajes CAM son generados por el correspondiente servicio de OpenC2X, implementado en el fichero `cam/src/caservice.cpp`.

Este servicio envía el mensaje generado al BTP utilizando una primitiva BTP-Data.request la cual incluye un campo *data* que no sólo contiene el mensaje generado, como correspondería al estándar, sino que incluye campos de control para el DCC, debido

a la implementación parcial del DCC que realiza OpenC2X detallada en el Trabajo Fin de Grado de Daniel [13]. Los campos *id* y *type* indicaban al DCC que el mensaje era de tipo CAM, para que pudiera rellenar las cabeceras falsas de GeoNetworking. Con la implementación de GeoNetworking, el DCC no necesita saber esa información, únicamente a cuál de las colas debe asignar ese paquete, por lo que esa información ya no se envía como muestra la Figura 92 (líneas 342 y 343).

```
342     -       data->set_id(messageID_cam);
343     -       data->set_type(dataPackage::DATA_Type_CAM);
344     -       data->set_priority(dataPackage::DATA_Priority_BE);
345     -
346 343         int64_t currTime = Utils::currentTime();
347 344         data->set_createtime(currTime);
348 345         data->set_validuntil(currTime + mConfig.mExpirationTime*1000*1000*1000);
349 346         data->set_content(strCam);
350 347
        348 +       // Generate BTP-Data.request primitive
        349 +       // BTP Header type B as specified by standard EN 302 637-2 V1.4.1 take from config.xml
351 350         if (mConfig.mTypeHeader == false) {
352     -               btpRequest.set_btptype(btp::BTP_REQUEST_btp_Type_B);
        351 +               btpRequest.set_btptype(GeoNetworking::BTP_B);
        352 +               btpRequest.set_destinationportinfo(mConfig.mDestInfo); // Dest port infor only needed in BTP type B
353 353         } else {
354     -               btpRequest.set_btptype(btp::BTP_REQUEST_btp_Type_A);
355     -       }
356     -       btpRequest.set_sourceport(2001);
        354 +               btpRequest.set_btptype(GeoNetworking::BTP_A);
        355 +               btpRequest.set_sourceport(2001); // source port only needed in BTP type A
        356 +       }
        357 +
357 358         btpRequest.set_destinationport(2001);
358     -       btpRequest.set_destinationportinfo(mConfig.mDestInfo);
        359 +       // Fill GeoNetworking primitive parameters
        360 +       btpRequest.set_gntransporttype(GeoNetworking::SHB);
        361 +       // Communication Profile: ITS-G5 or LTE-V2X (from config.xml)
        362 +       btpRequest.set_gncomprofile(mConfig.mComProfile);
        363 +       btpRequest.set_gnsecurityprofile(0); // SECURED (1) OR UNSECURED(0), should only be mandatory if ITS-S configuration (MIB) doesn't
        364 +       btpRequest.set_gntrafficclass(GeoNetworking::AC_BE);
        365 +       btpRequest.set_gnpacklifetime(mConfig.mExpirationTime); // Maximum Packet lifetime 1000ms (mExpirationTime expressed in sec)
```

**Figura 92: Modificaciones realizadas en el fichero cam/src/caservice.cpp**

También se han añadido, como se puede comprobar de nuevo en la Figura 92, todos los campos relacionados con GeoNetworking de la primitiva que no se estaban completando hasta ahora. Entre ellos está el tipo de transporte, SHB, y el Traffic Class, AC_BE, que antes se incluía en el campo *data* (línea 344) y ya no es necesario.

Una vez completada la primitiva, se serializa y envía como hasta ahora al servicio BTP.

El BTP recibe la primitiva y rellena la cabecera BTP y la añade al mensaje CAM como antes pero en vez de encapsular esta PDU creada en un objeto *data* y enviarla al DCC, ahora crea un objeto de tipo GN_Request con los campos de la primitiva GN-Data.request correspondiente y lo envía al servicio GeoNetworking, como muestra la Figura 93.

```
187     // Detects Type-A or Type-B BTP Header and fills second header field
188     if (headerType == GeoNetworking::BTP_A)
189     {
190         sourceOrDestInfo = btpRequest.sourceport();
191         geoNetRequest.set_upperprotocol(GeoNetworking::BTP_A);
192     }
193     else
194     {
195         sourceOrDestInfo = btpRequest.destinationportinfo();
196         geoNetRequest.set_upperprotocol(GeoNetworking::BTP_B);
197     }
198     fillBTPheader(headerType, destinationPort, sourceOrDestInfo);
199
200     // Creates btp PDU structure
201     btpHdrLen = sizeof(struct BTPHeader);
202     btpHdr = reinterpret_cast<uint8_t *>(&mBtpHdr);
203
204     unsigned int packetsize = btpHdrLen + camsize;
205     unsigned char packet[packetsize];
206     unsigned char *payload = packet + btpHdrLen;
207
208     memcpy(&packet, btpHdr, btpHdrLen); // copy BTP header at the memory address where is located the packet
209
210     memcpy(payload, encodedCam.c_str(), camsize); // copy payload to packet
211
212     gnData->set_content(packet,packetsize);
213     gnData->set_createtime(data.createtime());
214     gnData->set_bitrate(data.bitrate());
215     gnData->set_validuntil(data.validuntil());
216
217     // Create GeoNetworking Request (GN.Request primitive)
218     geoNetRequest.set_trafficclass(btpRequest.gntrafficclass());
219     geoNetRequest.set_transporttype(btpRequest.gntransporttype());
220
221     geoNetRequest.set_comprofile(btpRequest.gncomprofile());
222     geoNetRequest.set_securityprofile(btpRequest.gnsecurityprofile());
223     geoNetRequest.set_maxpacklifetime(btpRequest.gnpacklifetime());
224     geoNetRequest.set_allocated_data(gnData);
225     geoNetRequest.set_length(packetsize);
226
227     geoNetRequest.SerializeToString(&encodedGnRequest);
```

**Figura 93: Creación de la primitiva GN-Data.request en el servicio BTP**

La clase GeoNetService, una vez creadas las cabeceras Basic Header y Common Header como hemos visto en las secciones anteriores, comprueba el tipo de servicio de transporte solicitado y, tras comprobar que es SHB, completa el resto de la cabecera GeoNetworking con los campos de la cabecera SHB que vimos en las Figura 36 y Figura 37, como muestra la Figura 94.

Una vez se han rellenado los campos de la cabecera se crea el paquete completo añadiendo los datos recibidos del BTP a la cabecera GeoNetworking (líneas 223 y 225), y se crea la petición al DCC, que consiste en el paquete creado junto al parámetro Traffic Class, se serializa y se envía al DCC

```
136         // Extended Header
137         switch (request.transporttype())
138         {

192         case GeoNetworking::SHB:
193             GeoNetHeaders::SHB SHBHeader;
194             headerSize = sizeof(SHBHeader);
195
196             SHBHeader.basicHeader = basicHeader;
197             SHBHeader.commonHeader = commonHeader;
198
199             // Copy of Position Vector
200             SHBHeader.sopv.addr.params = htons(egoPV.gnAddr().paramsAsInt());
201             egoPV.gnAddr().MID(SHBHeader.sopv.addr.MID);
202             SHBHeader.sopv.timestamp = htonl(egoPV.timestamp());
203             SHBHeader.sopv.latitude = htonl(egoPV.latitude());
204             SHBHeader.sopv.longitude = htonl(egoPV.longitude());
205             SHBHeader.sopv.heading = htons(egoPV.heading());
206             SHBHeader.sopv.PAIandSpeed = htons(egoPV.PAIspeed());
207
208             /* Media-dependended data. With ITS-G5 should be use to transport DCC
209              * information as ETSI EN 302 363-4-2
210              * but not implemented yet, so using the default value (0).
211              */
212             SHBHeader.reserved = 0;
213             memcpy(GeoHeader, &SHBHeader, headerSize);
214
215             break;
216         }
217
218         unsigned int packetsize = headerSize + payloadSize;
219         unsigned char packet[packetsize];
220         unsigned char *payload = packet + headerSize;
221
222         // Fill the GeoNetworking Header
223         memcpy(packet, GeoHeader, headerSize);
224         // Fill the payload (BTP Header + facilities layer data)
225         memcpy(payload, encodedTPDU.c_str(), payloadSize);
226
227         // Fill the data protobuf message for sending packet to DCC
228         switch (request.trafficclass())
229         {
230         case GeoNetworking::AC_VO:
231             data.set_priority(dataPackage::DATA::VO);
232             break;
233         case GeoNetworking::AC_VI:
234             data.set_priority(dataPackage::DATA::VI);
235             break;
236         case GeoNetworking::AC_BK:
237             data.set_priority(dataPackage::DATA::BK);
238             break;
239         case GeoNetworking::AC_BE:
240             data.set_priority(dataPackage::DATA::BE);
241             break;
242         default:
243             data.set_priority(dataPackage::DATA::BE);
244             break;
245         }
246         data.set_content(packet, packetsize);
247
248         // Serialize data protobuf message to string
249         encodedData = data.SerializeAsString();
250
251         // Send message using mSenderToDCC
252         envelope = "GeoNetworking";
253         mLogger->logInfo("Forward incoming PDU from BTP to DCC");
254         mSenderToDcc->send(envelope, encodedData); // Send btp packet to DCC
255     }
256 }
```

**Figura 94: Creación de la cabecera SHB en la clase GeoNetService**

### 5.6.4 Envío de mensajes DENM

La implementación del DEMN de OpenC2X es muy básica. Existe una pequeña aplicación que envía, a petición del usuario a través de la interfaz web del software, un texto de prueba al servicio DENM, que se encarga de generar el DENM correspondiente y enviarlo al BTP mediante una primitiva BTP-Data.request.

Al igual que en caso del servicio CAM, no estaban incluidas en la petición los campos de la directiva correspondientes a parámetros de GeoNetworking, que se han añadido como muestra la Figura 95. Entre ellos está el tipo de servicio, GBC, y Traffic Class, que anteriormente se incluía en el objeto DATA donde se encapsulaba el mensaje DENM (línea 196). También hay que incluir en esta primitiva el área de destino en el cual tendrá validez los DENM enviados (líneas 200 a 209).

```
177       -       data->set_id(messageID_denm);
178       -       data->set_type(dataPackage::DATA_Type_DENM);
179       -       data->set_priority(dataPackage::DATA_Priority_VI);
180 179           
181 180           int64_t currTime = Utils::currentTime();
182 181           data->set_createtime(currTime);
183 182           data->set_validuntil(currTime + 2*1000*1000*1000);      //2s TODO: conform to standard? -> specify using CLI
184 183           data->set_content(strDenm);
185 184           
186       -       btpRequest.set_btptype(btp::BTP_REQUEST_btp_Type_B);
187       -       btpRequest.set_sourceport(2002);
      185 +       btpRequest.set_btptype(GeoNetworking::BTP_B);
      186 +       // Not needed in BTP-B type header
      187 +       // btpRequest.set_sourceport(2002);
188 188           btpRequest.set_destinationport(2002);
189 189           btpRequest.set_destinationportinfo(0);
      190 +       // Fill GeoNetworking primitive parameters
      191 +       btpRequest.set_gntransporttype(GeoNetworking::GeoBroadcast);
      192 +       // Communication Profile: ITS-G5 or LTE-V2X (from config.xml)
      193 +       // btpRequest.set_gnComProfile(mConfig.mComProfile);
      194 +       btpRequest.set_gncomprofile(1); // ITS-G5
      195 +       btpRequest.set_gnsecurityprofile(0); // SECURED (1) OR UNSECURED(0)
      196 +       btpRequest.set_gntrafficclass(GeoNetworking::AC_VI);
      197 +       btpRequest.set_gnpacklifetime(1000); // Maximum Packet lifetime 1000ms
      198 +       
      199 +       // Destination area
      200 +       destination = trigger.geoarea();
      201 +       destinationArea->set_shape(destination.shape());
      202 +       destinationArea->set_distancea(destination.distancea());
      203 +       destinationArea->set_distanceb(destination.distanceb());
      204 +       destinationArea->set_angle(destination.angle());
      205 +       mMutexLatestGps.lock();
      206 +       destinationArea->set_geoarealatitude(mLatestGps.latitude() * 10000000); // in one-tenth of microdegrees
      207 +       destinationArea->set_geoarealongitude(mLatestGps.longitude() * 10000000); // in one-tenth of microdegrees
      208 +       mMutexLatestGps.unlock();
      209 +       btpRequest.set_allocated_geoarea(destinationArea);
      210 +       
190 211           btpRequest.set_length(strDenm.length());
191 212           btpRequest.set_allocated_data(data);
```

**Figura 95: Modificaciones realizadas en el fichero denm/src/denservice.cpp**

Dado que esa área de destino relevante dependerá de la aplicación que utilice el servicio DENM se ha optado por que sea en dicha aplicación donde se defina dicha área de

destino. Aunque en OpenC2X hay una pequeña aplicación que se encuentra en `denmApp/src/` en realidad los DENM se están disparando desde la aplicación web, en la función *triggerDenm()* del fichero `website/script/webAppQuery.js`. Para nuestras pruebas hemos puesto el caso básico de un área circular de 100 metros de radio.

La posición central del área de destino la añade de momento el servicio DENM y no la aplicación. Si se decidiera que fuera la propia aplicación, esta deberá tener acceso a los datos del GPS.

Una vez completada la primitiva BTP-Data.request, esta se envía al BTP que la recibe por un puerto diferente a la correspondiente al servicio CAM, debido a la limitación de ZMQ que no permite a dos servicios enviar mensajes por sockets con el mismo número de puerto explicada con detalle por Daniel en su Trabajo Fin de Grado. [13]

Por ello, la función que rellena la cabecera BTP y la añade al mensaje DEMN, *receiveFromDenmSendToGeoNet()*, es muy similar a la que vimos en la Figura 93: crea un objeto de tipo GN_Request con los campos de la primitiva GN-Data.request correspondiente y lo envía al servicio GeoNetworking. Lo único diferente es que añade los datos del área de destino, como se puede ver en las líneas 311 a 320 de la Figura 96. Sin embargo, con la comprobación de la línea 311, este código podría ser el mismo para los dos tipos de mensaje. Por ello uno de los temas a investigar en futuras versiones de nuestro OpenC2X podría ser el averiguar dónde se produce esta limitación de los sockets ZMQ (o de la implementación de los desarrolladores originales de OpenC2X en sus *communicationSender* y *communicacionReceiver*).

```
306         // Create GeoNetworking Request (GN.Request primitive)
307         geoNetRequest.set_trafficclass(btpRequest.gntrafficclass());
308         geoNetRequest.set_transporttype(btpRequest.gntransporttype());
309
310         // Set the destination area in the GN.Request
311         if (btpRequest.has_geoarea()) {
312             destination = btpRequest.geoarea();
313             gnDestination->set_shape(destination.shape());
314             gnDestination->set_geoarealatitude(destination.geoarealatitude());
315             gnDestination->set_geoarealongitude(destination.geoarealongitude());
316             gnDestination->set_distancea(destination.distancea());
317             gnDestination->set_distanceb(destination.distanceb());
318             gnDestination->set_angle(destination.angle());
319             geoNetRequest.set_allocated_geoarea(gnDestination);
320         }
321
322         geoNetRequest.set_comprofile(btpRequest.gncomprofile());
323         geoNetRequest.set_securityprofile(btpRequest.gnsecurityprofile());
324         geoNetRequest.set_maxpacklifetime(btpRequest.gnpacklifetime());
325         geoNetRequest.set_allocated_data(gnData);
326         geoNetRequest.set_length(packetsize);
327
328         geoNetRequest.SerializeToString(&encodedGnRequest);
```

**Figura 96: Creación de la primitiva GN-Data.request en el servicio BTP para un DENM**

El servicio GeoNetworking recibe la primitiva GN-Data.request que contiene el DENM a enviar de igual manera que lo hizo en el caso de los CAM, y repite el proceso de crear las cabeceras Basic Header y Common Header como hemos visto en las secciones anteriores. Ahora, al comprobar que el mensaje requiere del servicio de transport GBC, creará la cabecera extendida con los campos indicados en las Figura 40 y Figura 41 como muestra la Figura 98.

El servicio de transporte GBC incluye un número de secuencia del paquete, como vimos en el punto 3.7.3, que se ha implementado como una clase, SequenceNumber, que se encuentra en el fichero `geonetworking/src/SequenceNumber.h` e implementa la Fórmula 2 sobrecargando el operador ++ como muestra la Figura 97.

```
24  class SequenceNumber {
25  public:
26      SequenceNumber() {}
27      uint16_t sequenceNumber() { return mSeqNumber; }
28      void operator++() {
29          mSeqNumber = (mSeqNumber+1)%(__UINT16_MAX__);
30      }
31
32  private:
33      uint16_t mSeqNumber = 0;
34  };
```

**Figura 97: Clase SequenceNumber**

```
131         // Common Header
132         uint16_t payloadSize = static_cast<uint16_t>(encodedTPDU.size());
133         mLogger->logInfo("Payload size: " + to_string(payloadSize));
134         commonHeader = fillCommonHeader(request, payloadSize);
135
136         // Extended Header
137         switch (request.transporttype())
138         {
```

```
147         case GeoNetworking::GeoBroadcast:
148             GeoNetHeaders::GBC GBCHeader;
149             headerSize = sizeof(GBCHeader);
150
151             GBCHeader.basicHeader = basicHeader;
152             GBCHeader.commonHeader = commonHeader;
153
154             // Copy of Position Vector
155             GBCHeader.sopv.addr.params = htons(egoPV.gnAddr().paramsAsInt());
156             egoPV.gnAddr().MID(GBCHeader.sopv.addr.MID);
157             GBCHeader.sopv.timestamp = htonl(egoPV.timestamp());
158             GBCHeader.sopv.latitude = htonl(egoPV.latitude());
159             GBCHeader.sopv.longitude = htonl(egoPV.longitude());
160             GBCHeader.sopv.heading = htons(egoPV.heading());
161             GBCHeader.sopv.PAIandSpeed = htons(egoPV.PAIspeed());
162             // Sequence number
163             GBCHeader.sequenceNumber = seqNumber.sequenceNumber();
164             ++seqNumber;
165             GBCHeader.reserved = 0;
166             // Destination area
167             destinationArea = request.geoarea();
168             GBCHeader.latitude = htonl(destinationArea.geoarealatitude());
169             GBCHeader.longitude = htonl(destinationArea.geoarealongitude());
170             GBCHeader.distA = htons(static_cast<uint16_t>(destinationArea.distancea()));
171
172             // Parameters distance B and Angle from the GBC extended header must be set to 0 if area shape is circular
173             if (destinationArea.shape() == GeoNetworking::Circular)
174             {
175                 GBCHeader.distB = 0;
176                 GBCHeader.angle = 0;
177             }
178             else
179             {
180                 GBCHeader.distB = htons(static_cast<uint16_t>(destinationArea.distanceb()));
181                 GBCHeader.angle = htons(static_cast<uint16_t>(destinationArea.angle()));
182             }
183             GBCHeader.resrvd = 0;
184
185             memcpy(GeoHeader, &GBCHeader, headerSize);
186             mLogger->logDebug("GBC packet send");
187             break;
```

**Figura 98: Creación de la cabecera GBC en la clase GeoNetService.**

A partir de aquí, el proceso de creación del paquete GeoNetworking y su envío al DCC es exactamente el mismo que en el caso de los CAM, al ejecutarse la misma función, *receiveFromBtpSendToDcc()*, en ambos casos.

### 5.6.5 Cambios en el DCC

Ahora que el servicio GeoNetworking implementado se encarga de crear el paquete completo GeoNetworking con las cabeceras, este servicio ya no tiene que realizar ningún cambio sobre él, sino sólo encolarlo en la cola indicada por el parámetro Traffic Class y enviarlo cuando le corresponda.

El primer cambio realizado, Figura 99, fue en los sockets ZMQ, dado que el servicio DCC aún mantenía dos *Communication Reciver* que escuchaba del BTP con diferente filtro para discriminar si el mensaje era un CAM o un DENM, algo que ya no necesita saber el DCC para procesar el paquete. También fue necesario modificar el socket que enviaba los datos al BTP para que lo hiciera al nuevo puerto correspondiente al GeoNetworking.

```
55      -   mReceiverFromCa = new CommunicationReceiver(module, "7777", "CAM", mGlobalConfig.mExpNo, loggingConf, statisticConf);
56      -   mReceiverFromDen = new CommunicationReceiver(module, "7777", "DENM", mGlobalConfig.mExpNo, loggingConf, statisticConf);
    55  +   mReceiverFromGeoNetworking = new CommunicationReceiver(module, "7777", "GeoNetworking", mGlobalConfig.mExpNo, loggingConf, statisti
57  56      mSenderToHw = new SendToHardwareViaMAC(module,mGlobalConfig.mEthernetDevice, mGlobalConfig.mExpNo, loggingConf, statisticConf);
58  57      mReceiverFromHw = new ReceiveFromHardwareViaMAC(module, mGlobalConfig.mExpNo, loggingConf, statisticConf);
59      -   mSenderToServices = new CommunicationSender(module, "5555", mGlobalConfig.mExpNo, loggingConf, statisticConf);
    58  +   mSenderToGeoNetworking = new CommunicationSender(module, "6655", mGlobalConfig.mExpNo, loggingConf, statisticConf);
60  59      mSenderToLdm = new CommunicationSender(module, "1234", mGlobalConfig.mExpNo, loggingConf, statisticConf);
```

**Figura 99: Cambios en los Communication Receiver y Communication Sender del DCC**

La clase que implementa el servicio DCC realizaba el envío del paquete CAM o DENM invocado al método *sendWithGeoNet()* del objeto *mSenderToHw*, de la SendToHardwareViaMAC, incluyéndolo como parámetro junto a la prioridad del mismo y si era un CAM o DENM. Éste a su vez invocaba a otro de los métodos de la clase, *fillGeoNetBTPheaderForCAM()* o *fillGeoNetBTPheaderForDENM()* dependiendo del tipo, que rellenaba las cabeceras faltantes, y lo enviaba por fin a la interfaz de red.

Esos métodos han sido eliminados, y ahora se envía el paquete mediante el método *send()* del objeto *mSenderToHw*, como muestra la Figura 100.

```
    @@ -545,7 +450,7 @@ void DCC::sendQueuedPackets(Channels::t_access_category ac) {
545 450
546 451                          string byteMessage;
547 452                          byteMessage = data->content();
548     -                        mSenderToHw->sendWithGeoNet(&byteMessage, ac, data->type());
    453 +                        mSenderToHw->send(&byteMessage, ac);
549 454                          mLogger->logInfo("AC " + to_string(ac) + ": Sent data " + to_string(data->id()) + " to HW -> queue length: " + to_s
550 455                          delete data;
551 456                  }
```

**Figura 100: Cambios en el envío del paquete a la interfaz de red desde el DCC**

Dicho método ya estaba creado en la clase *SendToHardwareViaMAC*, aunque no se usaba, y simplemente manda el paquete creado en el servicio GeoNetworking por la interfaz de red sin más modificación que añadir las cabeceras del protocolo de la capa de acceso (en este caso ethernet).

La recepción de los paquetes por la interfaz de red también se ha simplificado. Hasta ahora la función del DCC que se encargaba de recibir datos de la interfaz de red era *receiveFromHw2()*, que invocaba al método *receiveWithGeoNetHeader()* de la clase ReceiveFromHardwareViaMAC. Este método recibía un paquete de la interfaz de red,

analizaba el byte del paquete correspondiente a los campos HT/HST de la cabecera GeoNetworking para comprobar si era un CAM o un DENM y poder eliminar la cabecera del paquete.

Esta función ha sido eliminada también, y en su lugar se utiliza la función llamada *receiveFromHw()* de la clase DCC, que invoca al método *receive()* de la misma clase utilizada anteriormente, ReceiveFromHardwareViaMAC. Ambos métodos estaban implementados, aunque no se usaban. El método *receiveFromHw()* ha sido modificado levemente como se puede ver en la Figura 101.

```
272  214   void DCC::receiveFromHw() {
273        -         pair<string,string> receivedData;              //MAC Sender, serialized DATA
274        -         string* senderMac = &receivedData.first;
275        -         string* serializedData = &receivedData.second;
276        -         dataPackage::DATA data;
277  215            mLogger->logInfo("start receiving via Hardware");
278  216            while (1) {
279        -                 receivedData = mReceiverFromHw->receive();      //receive serialized DATA
280        -
281        -
282        -                 //check whether the mac of the sender and our own mac are the same and discard the package if we want to ignore those packa
283        -                 if(mConfig.ignoreOwnMessages && senderMac->compare(mSenderToHw->mOwnMac) == 0){
     217   +                 //MAC Sender, serialized DATA
     218   +                 pair<string, string> receivedData = mReceiverFromHw->receive(); //receive serialized DATA
     219   +                 string senderMac = receivedData.first;
     220   +                 string serializedData = receivedData.second;
     221   +
     222   +                 //check whether the mac of the sender and our own mac are the same and discard the package
     223   +                 //if we want to ignore those packages
     224   +                 if(mConfig.ignoreOwnMessages && senderMac.compare(mSenderToHw->mOwnMac) == 0){
284  225                             mLogger->logDebug("received own Message, discarding");
285  226                             continue;
286  227                     }
287  228
288        -                 data.ParseFromString(*serializedData);          //deserialize DATA
289  229                     //processing...
290        -                 mLogger->logInfo("forward message from "+*senderMac +" from HW to services");
291        -                 switch(data.type()) {                                                   //send serialized DATA to corresponding mod
292        -                         case dataPackage::DATA_Type_CAM:              mSenderToServices->send("CAM", *serializedData);        break;
293        -                         case dataPackage::DATA_Type_DENM:             mSenderToServices->send("DENM", *serializedData);       break;
294        -                         default:        break;
295        -                 }
296        -         }
297        - }
```

**Figura 101: Cambios en el método *receiveFromHw()***

## 5.7 Recepción de paquetes usando GeoNetworking

Como hemos visto al final del punto 5.6.5, El DCC recibe el paquete GeoNetworking de la capa de acceso mediante el método *receiveFromHw()*, que lo envía al servicio de GeoNetworking, que lo recibe mediante la función *receiveFromDcc()*.

Lo primero que hace dicha función, Figura 102, es extraer la cabecera Basic Header del paquete recibido y comprobar la versión del protocolo GeoNetworking indicada en la misma, indicando un error si no es la versión 0 o 1, las existentes hasta el momento.

A continuación, comprueba el campo Next Header para verificar si es un paquete en el que se han aplicado los mecanismos de seguridad o no. Como OpenC2X no incorpora ninguno de estos mecanismos, si el paquete recibido si ha sido cifrado, por ejemplo, este se descarta y se indica un error.

```
427  void GeoNetService::receiveFromDcc()
428  {
429      string envelope;
430      string encodedData; // serialized DATA
431      // const char* data;
432      // dataPackage::DATA* data;         //deserialized DATA
433
434      while (1)
435      {
436          pair<string, string> received = mReceiverFromDcc->receive();
437          envelope = received.first;
438          encodedData = received.second;
439          GeoNetHeaders::BasicHeader tmpBasicHeader;
440          GeoNetHeaders::CommonHeader tmpCommonHeader;
441          uint8_t geoNetVersion = 0;
442          uint8_t geoNetNH = 0;
443
444          // Extract the GeoNetworking basic header to check if we can further process the packet
445          memcpy(&tmpBasicHeader, encodedData.c_str(), sizeof(struct GeoNetHeaders::BasicHeader));
446
447          geoNetVersion = tmpBasicHeader.versionAndNH >> 4;
448          geoNetNH = tmpBasicHeader.versionAndNH & 0x0F;
449
450          if ((geoNetVersion != 0) && (geoNetVersion != 1))
451          {
452              mLogger->logError("GeoNetworking protocol version " + to_string(geoNetVersion) + " not supported");
453              continue;
454          }
455
456          if (geoNetNH == 2)
457          {
458              mLogger->logError("Security not yet implemented, discarding PDU");
459              continue;
460          }
461
```

**Figura 102: Procesado de la cabecera Basic Header del paquete recibido**

A continuación, se extrae la cabecera Common Header para comprobar qué tipo de paquete es, como muestra la Figura 103. Como ya se indicó cuando se explicó el proceso de envío, sólo se han implementado los paquetes SHB y GBC, que se corresponden con un valor del campo HT/HST `0x50` para los paquetes SHB y `0x40`, `0x41` y `0x42` (dependiendo de la forma del área de destino) para los paquetes GBC.

Si el campo HT/HST indica otro tipo de paquete, o un valor que no se corresponde con ninguno de los indicados en la Tabla 10, se genera un mensaje de error indicando que el tratamiento de ese tipo de paquete no está aún implementado, o es un valor incorrecto.

```
462     // Extract the GeoNetworking common header to check the type of header and process it
463     memcpy(&tmpCommonHeader, encodedData.c_str() + sizeof(struct GeoNetHeaders::BasicHeader), sizeof(struct GeoNetHeaders::CommonHeader));
464
465     mLogger->logInfo("Type of header: " + to_string(tmpCommonHeader.HTAndHST));
466
467     switch (tmpCommonHeader.HTAndHST)
468     {
469     case 0x10:
470         mLogger->logError("Beacon not implemented yet");
471         break;
472     case 0x20:
473         mLogger->logError("GUC not implemented yet");
474         break;
475     case 0x30:
476     case 0x31:
477     case 0x32:
478         mLogger->logError("GAC not implemented yet");
479         break;
480     case 0x40:
481     case 0x41:
482     case 0x42:
483         processIncomingGBC(encodedData);
484         break;
485     case 0x50:
486         processIncomingSHB(encodedData);
487         break;
488     case 0x51:
489         mLogger->logError("TSB not implemented yet");
490         break;
491     case 0x60:
492     case 0x61:
493         mLogger->logError("Location service not implemented yet");
494         break;
495     default:
496         mLogger->logError("Incorrect header type");
497         break;
498     }
499 }
```

**Figura 103: Procesado de la cabecera Common Header para obtener el tipo de paquete**

### 5.7.1 Procesado de paquetes SHB

Si el paquete recibido es de tipo SHB, la función *receiveFromDcc()* llama a la función *processIncomingSHB(const std::string &receivedPDU*), pasando una referencia al *string* que contiene el paquete entero.

Lo primero que hace la función es extraer la cabecera GeoNetworking completa (línea 512) y analizar los campos para crear una primitiva de tipo GN-Data.indication (líneas 514 a 556) con los parámetros indicados en la Tabla 14, usando un mensaje protobuf de tipo GN_INDICATION, mostrado en la Figura 71.

Por último, incluye en dicha primitiva la PDU de la capa transporte recibida y su longitud (558 a 561), serializa el mensaje (línea 564) y lo envía al BTP (líneas 566 y 567) para su tratamiento como se explica en la sección 5.7.3.

```
502  void GeoNetService::processIncomingSHB(const std::string &receivedPDU)
503  {
504      GeoNetHeaders::SHB receivedShbHeader;
505      GeoNetworking::GN_INDICATION indication;
506      uint8_t headerSize = sizeof(receivedShbHeader);
507      uint8_t nextHeader;
508      uint16_t payloadLen = receivedPDU.size() - headerSize;
509      string encodedGnIndication;
510      char *payload;
511
512      memcpy(&receivedShbHeader, receivedPDU.c_str(), headerSize);
513
514      nextHeader = receivedShbHeader.commonHeader.NHAndReserved >> 4;
515
516      switch (nextHeader)
517      {
518      case 0:
519          mLogger->logInfo("Next Header type \"ANY\", packet discarded");
520          return;
521      case 1:
522          indication.set_upperprotocol(GeoNetworking::BTP_A);
523          break;
524      case 2:
525          indication.set_upperprotocol(GeoNetworking::BTP_B);
526          break;
527      case 3: // IPv6 not allowed
528          mLogger->logInfo("IPv6 not implemented yet, packet discarded");
529          return;
530      default:
531          mLogger->logError("Incorrect Next Header value, packet discarded");
532          return;
533      }
534
535      indication.set_transporttype(GeoNetworking::SHB);
536      indication.set_remaininghoplimit(receivedShbHeader.basicHeader.remainingHopLimit);
537      indication.set_remainingpacklifetime(receivedShbHeader.basicHeader.lifetime);
538
539      switch (receivedShbHeader.commonHeader.TrafficClassParamsAndID & 0x3F)
540      {
541      case 0:
542          indication.set_trafficclass(GeoNetworking::AC_VO);
543          break;
544      case 1:
545          indication.set_trafficclass(GeoNetworking::AC_VI);
546          break;
547      case 2:
548          indication.set_trafficclass(GeoNetworking::AC_BE);
549          break;
550      case 3:
551          indication.set_trafficclass(GeoNetworking::AC_BK);
552          break;
553      default:
554          indication.set_trafficclass(GeoNetworking::AC_BK);
555          break;
556      }
557
558      indication.set_length(payloadLen);
559      payload = new char[payloadLen];
560      memcpy(payload, receivedPDU.c_str() + headerSize, payloadLen);
561      indication.set_pdufortransportprotocol(payload, payloadLen);
562      delete payload;
563
564      indication.SerializeToString(&encodedGnIndication);
565
566      mLogger->logInfo("Forward incoming SHB packet to BTP");
567      mSenderToBtp->send("GeoNetWorking", encodedGnIndication); // Send DCC packet to BTP
568  }
```

**Figura 104: Método *processIncomingSHB()***

### 5.7.2 Procesado de paquetes GBC

De manera similar al caso anterior, si el paquete recibido es de tipo GBC, la función *receiveFromDcc()* llama a la función *processIncomingGBC(const std::string &receivedPDU*), pasando una referencia al *string* que contiene el paquete entero.

Dicha función es prácticamente idéntica a la anterior, ya que por falta de tiempo ha sido imposible implementar alguno de los mecanismos del protocolo propios de este tipo de paquetes, como el reencaminamiento (que no existe en los paquetes SHB), o la comprobación del área de destino.

Al igual que en el caso anterior se completa una primitiva GN-Data.indication con los parámetros indicados en la Tabla 16. La principal diferencia, mostrada en la Figura 105 es que en este caso hay que incluir el área de destino en dicha primitiva e indicar que el tipo de transporte utilizado por el paquete es GBC.

Al igual que en el caso de los paquetes SHB, tras rellenar el objeto GN_INDICATION que representa a la primitiva, este se serializa y se envía al BTP para su tratamiento como se explica en la sección 5.7.3.

```
604     // Fill destination area in GN-Data.indication primitive
605     switch (receivedGBCHeader.commonHeader.HTAndHST & 0x0F)
606     {
607     case 0:
608         destinationArea->set_shape(GeoNetworking::Circular);
609         break;
610     case 1:
611         destinationArea->set_shape(GeoNetworking::Rectangular);
612         break;
613     case 2:
614         destinationArea->set_shape(GeoNetworking::Ellipsoidal);
615     default:
616         break;
617     }
618     destinationArea->set_geoarealatitude(ntohl(receivedGBCHeader.latitude));
619     destinationArea->set_geoarealongitude(ntohl(receivedGBCHeader.longitude));
620     destinationArea->set_distancea(ntohs(receivedGBCHeader.distA));
621     destinationArea->set_distanceb(ntohs(receivedGBCHeader.distB));
622     destinationArea->set_angle(ntohs(receivedGBCHeader.angle));
623     indication.set_allocated_geoarea(destinationArea);
624
625     indication.set_transporttype(GeoNetworking::GeoBroadcast);
```

**Figura 105: Método *processIncomingGBC()***

### 5.7.3 Recepción de la PDU en BTP

Una vez que GeoNetworking ha enviado la primitiva GN-Data.indication, esta es recibida por el BTP. Anteriormente BTP recibía únicamente la PDU, la cual debía procesar según indica el estándar del protocolo BTP [55] y entregar al servicio CAM o DENM según correspondiera. Por ello ahora es necesario extraer la PDU del objeto GN_INDICATION, mediante una pequeña modificación de la función *receiveFromDccSendToServices()* del fichero btp/src/btp.cpp, renombrada a *receiveFromGeoNetSendToServices()* por mantener una coherencia con lo que realmente hace ahora, como se indica en la Figura 106

```
83      - void BTPService::receiveFromDccSendToServices() {
        84  + void BTPService::receiveFromGeoNetSendToServices()
        85  + {
        86  +
        87  +       GeoNetworking::GN_INDICATION receivedGnIndication;
84      88          string envelope;
85      89          string serializedData;
86      90          unsigned int dataLen;
87      -           const char* data;
        91  +       const char *data;
88      92          uint16_t btpDestination;
89      93          uint16_t btpSecondary;
90      94
91      -           while (1) {
92      -                   pair<string, string> received = mReceiverFromDcc->receive();
        95  +       while (1)
        96  +       {
        97  +               pair<string, string> received = mReceiverFromGeoNet->receive();
93      98                  envelope = received.first;
94      -               serializedData = received.second;
95      -               unsigned int btpPDULen = serializedData.size();
        99  +           string serializedGnIndication = received.second;
       100  +
       101  +           receivedGnIndication.ParseFromString(serializedGnIndication);
       102  +           serializedData = receivedGnIndication.pdufortransportprotocol();
```

**Figura 106: Modificación de la función *receiveFromDccSendToServices()***

## 5.8 Compilación de OpenC2X

Para terminar de implementar el servicio de GeoNetworking es necesario recompilar OpenC2X con los nuevos módulos creados. Para ello hay que crear un fichero con las instrucciones para CMake, `geonetworking/src/CMakeLists.txt`, de manera que compile los ficheros de código fuente de las nuevas clases (Figura 107).

```
1  set(geonetservice_SRCS
2     GeoNetAddr.cpp
3     PositionVectors.cpp
4     geonetservice.cpp
5  )
6
7  add_executable (geonetservice ${geonetservice_SRCS})
8  target_link_libraries (geonetservice zmq protobuf boost_system boost_thread asn proto messages utility)
```

**Figura 107: Fichero geonetworking/src/CMakeLists.txt**

También es necesario compilar de nuevo los ficheros Protocol Buffers para que creen las clases que implementan las primitivas GN-Data.request y GN-Data.indication. Para ello existe un script en el directorio `common/buffers` llamado `generate.sh`, en el que hay que incluir los nuevos ficheros a compilar (Figura 108)-

```
7   protoc --proto_path=./ --cpp_out=build/ enums.proto
8
9   protoc --proto_path=./ --cpp_out=build/ area.proto
10
11  protoc --proto_path=./ --cpp_out=build/ PositionVector.proto
12
13  protoc --proto_path=./ --cpp_out=build/ GnIndication.proto
14
15  protoc --proto_path=./ --cpp_out=build/ GnRequest.proto
```

**Figura 108: Fichero common/buffers/generate.sh**

Una vez compilado el Proyecto como indican las instrucciones del fichero `readme`, sólo quedaría añadir las líneas mostradas en el script que lanza OpenC2X, `scripts/runOpenC2X.sh`, y ejecutarlo.

```
tmux send-keys "cd $BUILD_DIR/geonetworking/src" C-m
tmux send-keys "./geonetservice $GLOBAL_CONFIG \
$OPENC2X/geonetworking/$LOCAL_CONFIG_RELATIVE \
$OPENC2X/geonetworking/$LOGGING_CONF \
$OPENC2X/geonetworking/$STATISTICS_CONF" C-m
tmux split-window -v
```

Hay que tener en cuenta que, debido a que el script utiliza tmux para ejecutar y mostrar la salida de todos los módulos de OpenC2X en la misma pantalla, al añadir GeoNetworking no puede abrir todos en la misma pantalla si se ejecuta desde una ventana del cliente gráfico de terminal no maximizada. En ese caso se puede ejecutar el módulo en otra ventana ejecutando el script `scripts/runGeoNetservice.sh`.

## 5.9 Pruebas de comunicación

Para probar el funcionamiento de GeoNetworking se ejecutó OpenC2X en dos de los equipos utilizados como estaciones ITS y se capturó con Wireshark el tráfico entre ambos para verificar que ambos enviaban y recibían correctamente los paquetes de la otra estación y que estos estaban formados cumpliendo lo indicado en el estándar ETSI EN 646-4-1 [51].

La Figura 109 muestra la ventana de OpenC2X ejecutándose con el nuevo servicio ejecutándose en la parte superior derecha.

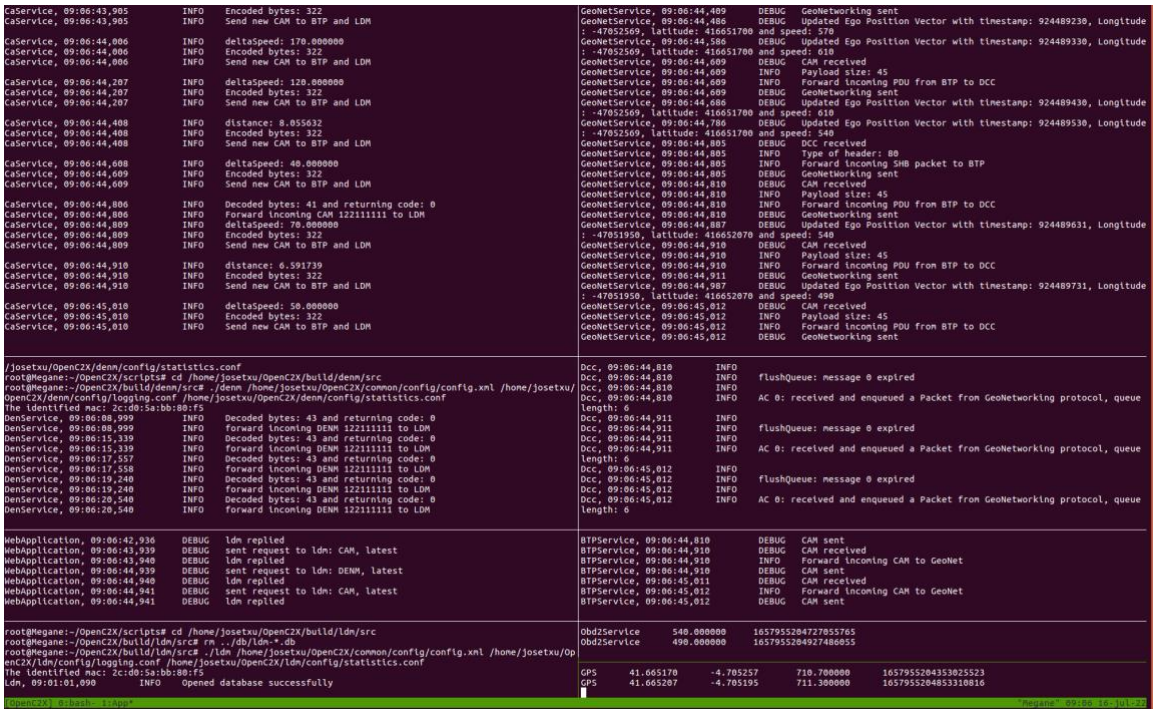

**Figura 109: OpenC2X ejecutándose con el nuevo módulo del servicio GeoNetworking**

Durante la ejecución del software el servicio GeoNetworking muestra, y almacena en sus ficheros de log, mensajes de aviso indicando cuando llega un mensaje un mensaje del BTP, cuando se envía al DCC y el proceso contrario: cuando llega un mensaje del DCC, información sobre él, y cuando se envía dicho mensaje al BTP.

También muestra mensajes sobre las actualizaciones del EGO Position Vector, indicando la posición y velocidad adquiridas de los módulos del GPS y OBD2. Según marca el estándar, las actualizaciones de dicho vector se producen 1000 veces cada segundo por defecto. Para nuestras pruebas, con el fin de que los logs y la salida por pantalla no se saturasen de mensaje de actualización del EGO PV y dificultasen la lectura del resto de la información, se redujo la frecuencia de actualización a 10 veces por segundo. Cuando se utilice en casos reales, si se necesita que el EGO PV se actualice más frecuentemente (y el GPS y el OBD son capaces de proporcionar los datos a dicha velocidad) es conveniente deshabilitar el envío del mensaje para evitar que los ficheros de log y la salida por pantalla se saturen de mensajes de aviso de actualización.

Tanto el servicio BTP como el DCC han visto retocados ligeramente sus mensajes también, como se aprecia en la Figura 110, para que sean coherentes con lo que realmente están haciendo.

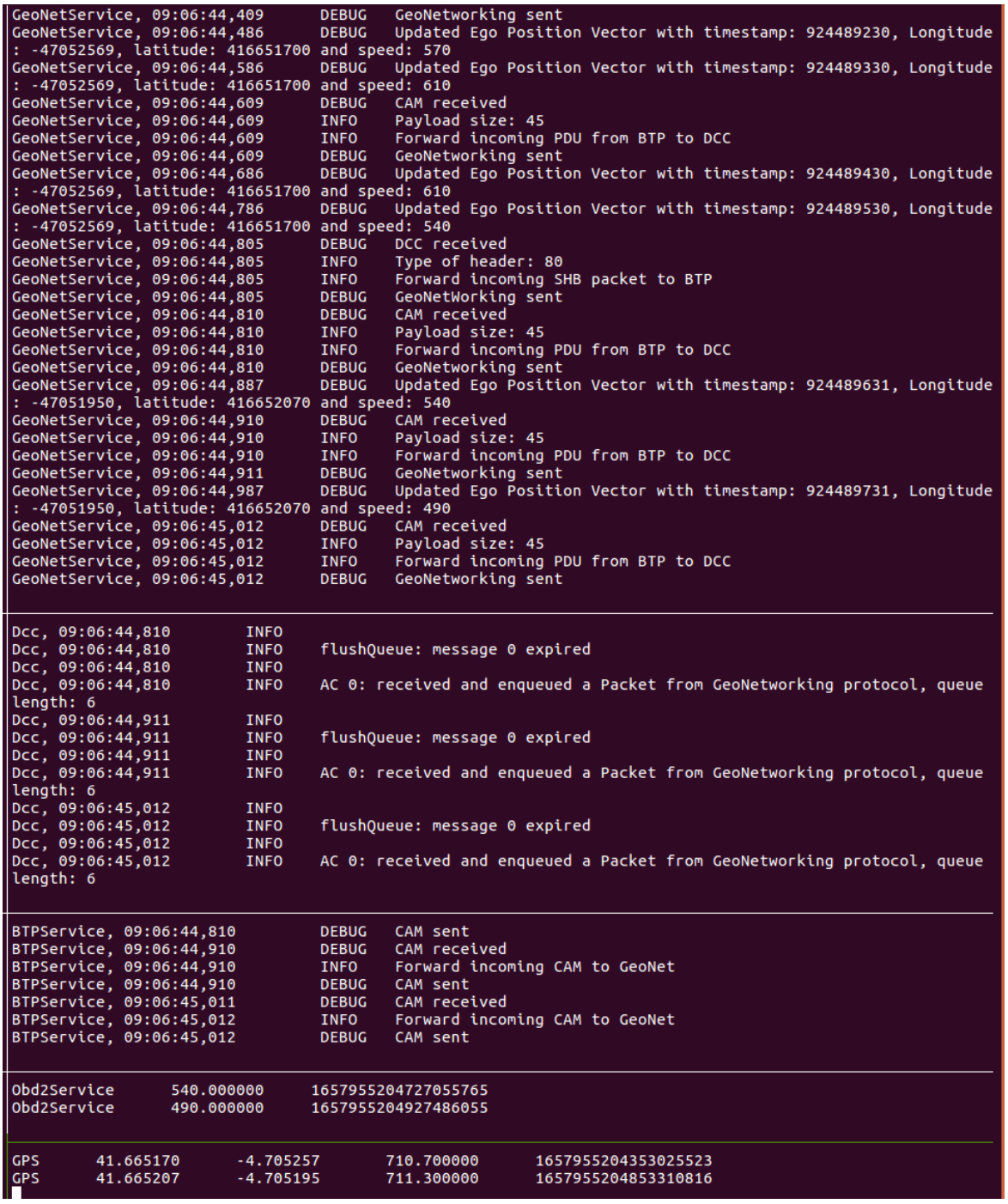


**Figura 110: Detalle de los mensajes de GeoNetworking, DCC y BTP**

La Figura 111 muestra la interfaz web de OpenC2X, con la información de los mensajes CAM y DENM enviados y recibidos por nuestra estación. En nuestro caso la estación está recibiendo mensajes CAM de dos estaciones, la 1 y 111111111, la primera ejecutando Vanetza y la segunda OpenC2X, por lo que la interoperabilidad con Vanetza sigue manteniéndose después de implementar GeoNetworking, como era de esperar.

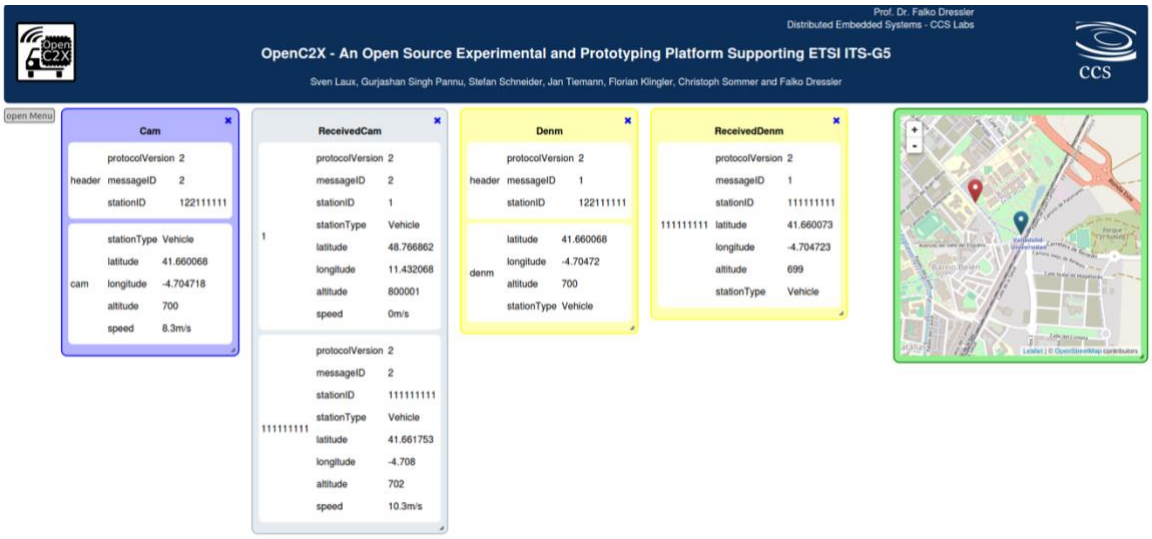


**Figura 111: Interfaz web de OpenC2X**

La Figura 112 muestra una captura de Wireshark de un paquete enviado por una de nuestras estaciones, conteniendo un mensaje CAM, donde se puede ver en detalle el contenido de las diferentes cabeceras de GeoNetworking.

Como se puede ver en la cabecera Basic Header se indica el lifetime codificado como indica el estándar (3.8.1), con la división en base y multiplicador.

En la cabecera Common Header se observa el tipo de transporte utilizado, SHB, la codificación del Traffic Class, y el número de saltos máximo, que en este caso es 1.

También se ve en detalle la dirección GeoNetworking de la estación, formada a partir de la MAC (campo MID) y el tipo de estación, como parte del Long Position Vector de origen con los datos de posicionamiento y su correspondiente timestamp.

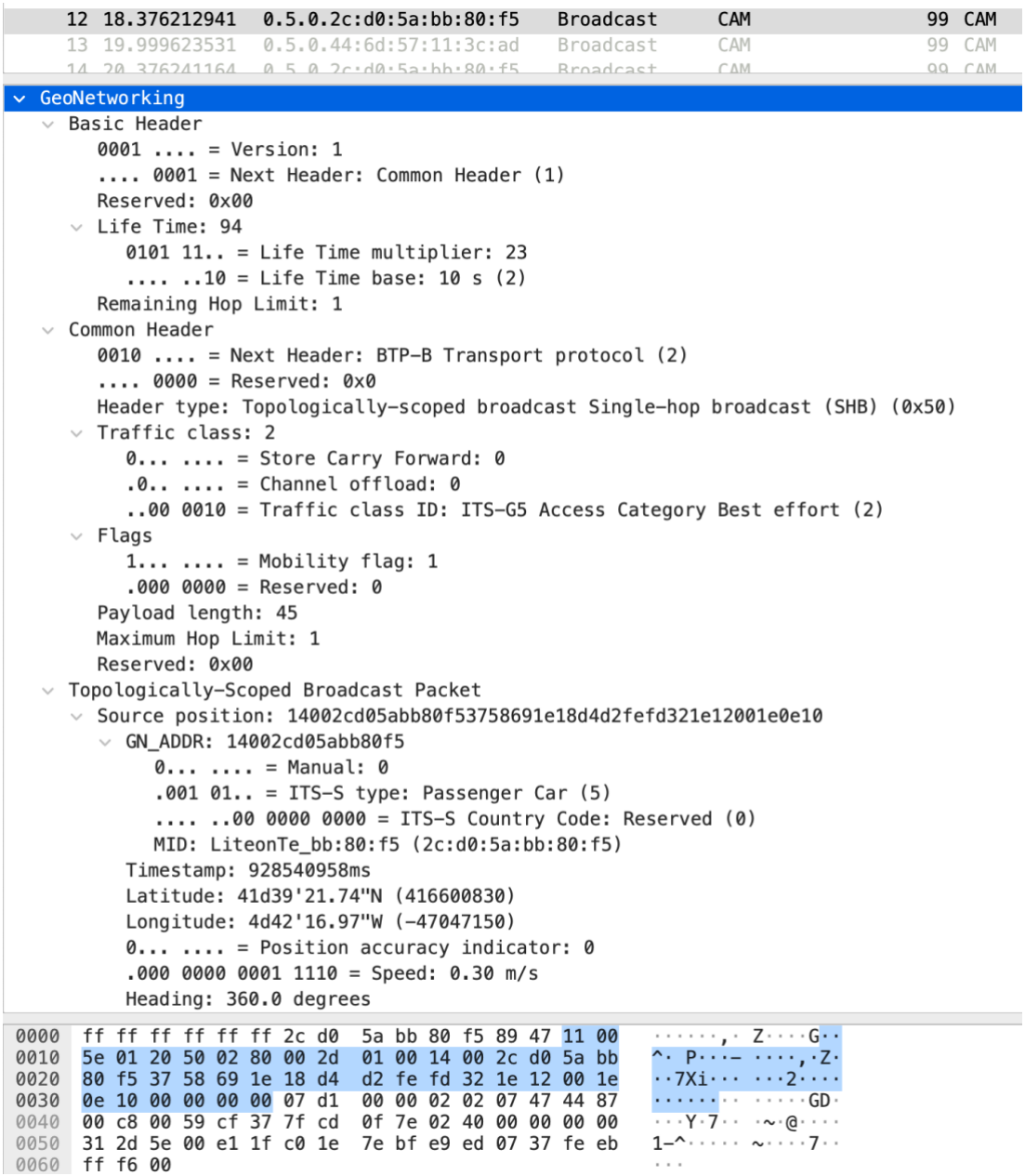


**Figura 112: Captura de Wireshark de un paquete GeoNetworking (CAM)**

De manera similar, la Figura 113 muestra la captura de un paquete que contiene un DENM. Las diferencias con el anterior más reseñables son el tipo de cabecera en la Common Header, que ahora indica un tipo de cabecera GBC con área de destino circular, el Traffic Class ID, que ahora indica una clase de acceso AC_VI, y el número de saltos, que en este caso se corresponde con la constante de protocolo *ItsGnDefaultHopLimit*.

Por último, no presentes en el paquete SHB, están el *sequence number* y, por supuesto, el área de destino de paquete enviado.

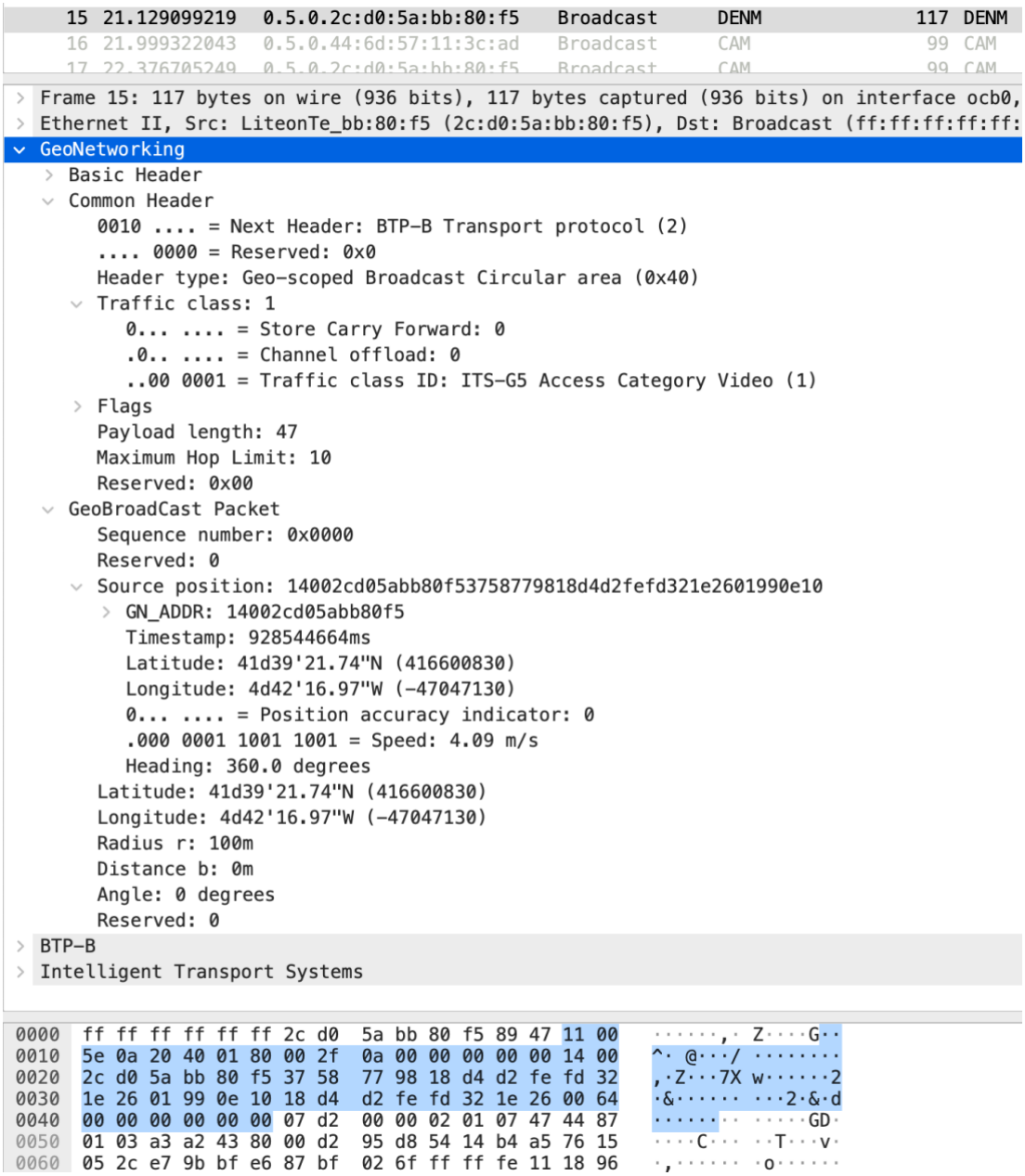


**Figura 113: Captura de Wireshark de un paquete GeoNetworking (DENM)**

La última prueba fue cambiar el área de destino, que se había definido como un área circular de 100 m de radio, por un área elipsoidal. La Figura 114 muestra la captura de un paquete conteniendo un DENM con la nueva área de destino, donde se puede observar que el tipo de cabecera ha cambiado a GBC con área elipsoidal y los nuevos valores de tamaño y orientación del área (*Distance a*, *Distance b* y *angle*).

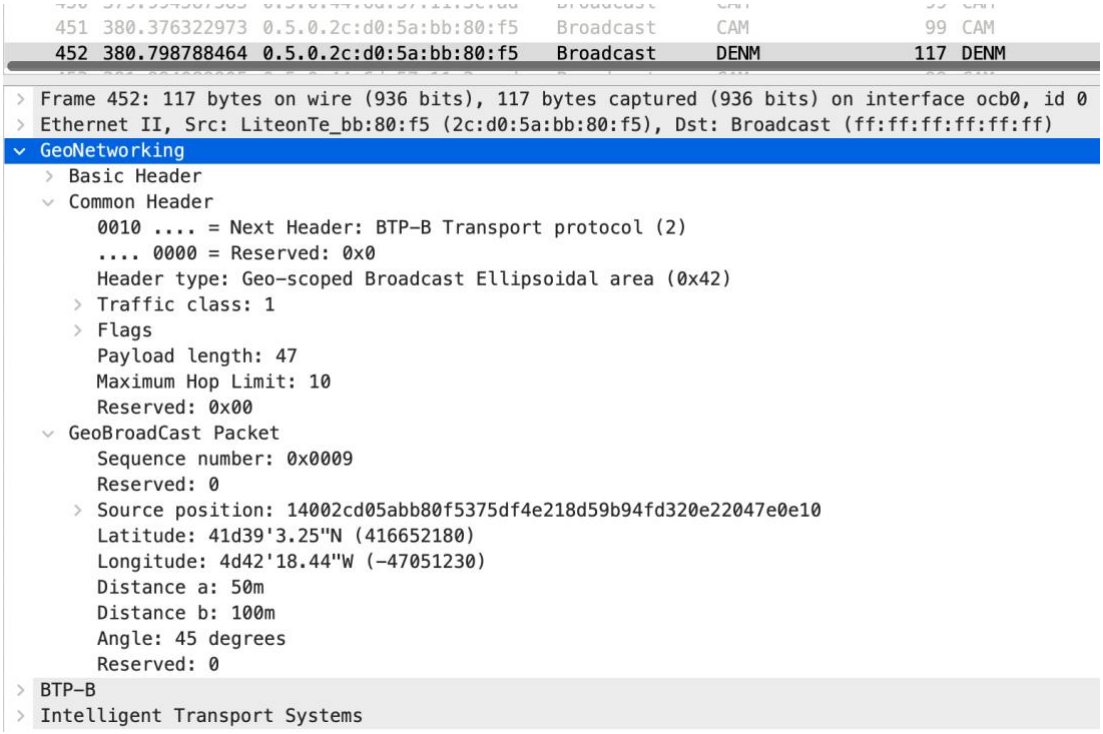


**Figura 114: Captura de Wireshark de un DENM con área de destino elipsoidal**

# 6

# Conclusiones y líneas futuras

El objetivo principal de este Trabajo Fin de Máster era la implementación del protocolo GeoNetworking en el software de comunicación OpenC2X. Junto a este objetivo se definieron una serie de objetivos secundarios.

El primero de ellos, el estudio de la situación actual de las tecnologías de acceso V2X, 802.11 y LTE-V2X/5G y el estado de la estandarización de los C-ITS en la Unión Europea nos ha permitido tener una visión amplia de estas tecnologías, su evolución histórica y los actores más importantes.

El estudio de las alternativas a OpenC2X y la instalación y pruebas de Vanetza nos ha permitido identificar la falta de actualización de alguno de los módulos de OpenC2X, como CAM y DENM, y proceder a los cambios necesarios para que dichos mensajes se generen e intercambien con otras estaciones cumpliendo con las últimas versiones de los estándares correspondientes, mejorando la interoperabilidad con otros softwares.

Por último, la implementación del protocolo en OpenC2X nos permite una mejora importante del mismo, al acercarlo a la implementación completa de la arquitectura de comunicación C-ITS de ETSI. Sin embargo, como hemos comprobado en el capítulo dedicado al estudio del protocolo GeoNetworking, es un protocolo complejo, que maneja una gran cantidad de estructuras de datos auxiliares, como la LocT o los diferentes *buffers* y, aunque tanto los mensajes CAM y DENM se intercambian cumpliendo el formato indicado por el estándar, todavía quedan elementos del protocolo por implementar.

Por ello una de las líneas futuras de mejora del software OpenC2X más inmediata sería la implementación de los elementos no implementados hasta ahora, como el procedimiento de selección del algoritmo de encaminamiento o los propios algoritmos de encaminamiento como el *Greedy Forwarding* o los algoritmos basados en contienda.

Por otro lado, debido a que el desarrollo inicial de OpenC2X data de 2016-2017, la instalación está hecha sobre una base de Ubuntu 16.04, totalmente obsoleto, por lo que sería conveniente comprobar si es posible realizar la instalación sobre versiones más modernas y realizar los cambios necesarios en caso de que no lo fuera. Alguna de las versiones de las librerías utilizadas, como *Boost, ZMQ* o *gpsd*, son bastante antiguas también, como el propio compilador. Hubo muchos problemas para la obtención de datos del GPS, que podrían ser debidas a la versión de gpsd utilizada, que los desarrolladores de Vanetza desaconsejan. Una actualización del compilador, que acepte C++ 20, y de las librerías *Boost* podrían facilitar el desarrollo de alguno de los elementos faltantes de OpenC2X.

Otra línea futura muy importante es realizar la implementación completa del DCC, para que incorpore la capa cruzada que permita el funcionamiento conforme al estándar y que el intercambio de información entre capa lleve exclusivamente la información que corresponde. También la implementación de las entidades de seguridad y de gestión que permitan completar la arquitectura de comunicación C-ITS.

Finalmente, otra línea futura no menos importante sería el estudio de las modificaciones a desarrollar para que OpenC2X permitiera el uso de tecnologías celulares, ya sea LTE-V2X o NR-V2X (5G), las cuales muy probablemente, debido al alto grado de cumplimiento del estándar del resto de capas superiores tras las modificaciones realizadas y la implementación de GeoNetworking, estarán sobre todo centradas a nivel del DCC.

# Bibliografía

[1] «Estadísticas de 2019 sobre seguridad vial: ¿qué esconden las cifras?,» [En línea]. Available: https://ec.europa.eu/commission/presscorner/detail/es/qanda_20_1004. [Último acceso: 27 febrero 2020].

[2] Comisión Europea, «Estrategia europea sobre los sistemas de transporte inteligentes cooperativos, un hito hacia la movilidad cooperativa, conectada y automatizada,» 2016.

[3] Comisión Europea, M/453 EN: STANDARDISATION MANDATE ADDRESSED TO CEN, CENELEC AND ETSI IN THE FIELD OF INFORMATION AND COMMUNICATION TECHNOLOGIES TO SUPPORT THE INTEROPERABILITY OF CO-OPERATIVE SYSTEMS FOR INTELLIGENT TRANSPORT IN THE EUROPEAN COMMUNITY, 2009.

[4] ETSI, *EN 302 665 V1.1.1: Intelligent Transport Systems (ITS); Communications Architecture,* 2010.

[5] ETSI, *EN 302 636-3 V1.2.1: Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 3: Network Architecture,* 2014.

[6] A. Festag, «Cooperative intelligent transport systems standards in europe,» *IEEE Communications Magazine,* vol. 52, nº 12, pp. 166-172, 2014.

[7] ETSI, *TS 103 301: Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service,* 2019.

[8] J. Fernández Pastrana, *802.11p standard and V2X applications on comercial Wi-Fi cards,* Universidad de Valladolid, 2017.

[9] M. P. Sánchez Martín, *Implementación de una unidad de a bordo de comunicación entre vehículos según el estándar ETSI ITS G5,* Universidad de Valladolid, 2017.

[10] A. Lobo González, *Implementación de una Road Side Unit de comunicación infraestructura vehículos según el estándar ETSI ITS G5,* Universidad de Valladolid, 2017.

[11] R. Herreras Babón, *Integración, testado y propuestas de mejora de una unidad de comunicación V2V,* Universidad de Valladolid, 2019.

[12] S. Laux, G. Singh Pannu, S. Schneider, J. Tiemann, F. Klingler, C. Sommer y F. Dressler, «OpenC2X,» CCS Labs, [En línea]. Available: https://www.ccs-labs.org/software/openc2x/. [Último acceso: 24 06 2022].

[13] D. Monje González, *Implementación del Protocolo de Transporte Básico en un Software de Comunicación V2X*, Universidad de Valladolid, 2021.

[14] «Eureka Prometheus Project - Wikipedia,» 14 febrero 2021. [En línea]. Available: https://en.wikipedia.org/wiki/Eureka_Prometheus_Project. [Último acceso: 11 marzo 2021].

[15] A. Festag, «Standards for vehicular communication—from IEEE 802.11p to 5G,» *e & i Elektrotechnik und Informationstechnik,* vol. 132, nº 7, pp. 409-416, 2015.

[16] Comisión Europea, *Commission Decision 2008/671/EC of 5 August on the harmonised use of radio spectrum in the 5875-5905 MHz frequency band for safety related application of Intelligent Transport Systems (ITS).*, 2008.

[17] C-ITS Platform, *Final report,* 2016.

[18] J. Yoshida, «The DSRC vs 5G Debate Continues,» EET Asia, 29 10 2019. [En línea]. Available: https://www.eetasia.com/the-dsrc-vs-5g-debate-continues/. [Último acceso: 6 enero 2021].

[19] Comisión Europea, *Commision Delegated Regulation of 13.3.2019 supplementing Directive 2010/40/EU of the European Parliament and of the Council with regard to the deployment and operational use of cooperative intelligent transport systems.*

[20] ETSI, *EN 303 613 V1.1.1: Intelligent Transport Systems (ITS); LTE-V2X Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band,* 2020.

[21] ETSI, *TS 102 636-4-3 V1.1.1: Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 3: Media-dependent,* 2020.

[22] ETSI, TR 103 667 V1.1.1: Intelligent Transport Systems (ITS); Study on Spectrum Sharing between ITS-G5 and LTE-V2X technologies in the 5 855 MHz - 5 925 MHz band, 2021.

[23] ETSI, TR 103 766 V1.1.1: Intelligent Transport Systems (ITS); Pre-standardization study on co-channel co-existence between IEEE- and 3GPP- based ITS technologies in the 5 855 MHz - 5 925 MHz frequency band, 2021.

[24] C-Roads, «About: C-Roads,» [En línea]. Available: https://www.c-roads.eu/platform/about/about.html. [Último acceso: 25 marzo 2021].

[25] InterCor, «About InterCor,» [En línea]. Available: https://intercor-project.eu/homepage/about-intercor/. [Último acceso: 13 abril 2021].

[26] Comisión Europea, [En línea]. Available: https://ec.europa.eu/transport/infrastructure/tentec/tentec-portal/map/maps.html. [Último acceso: 13 abril 2021].

[27] C-Roads Spain, «C-Roads Spain,» [En línea]. Available: https://www.c-roads.es/c-roads-spain. [Último acceso: 13 abril 2021].

[28] Car 2 Car Communication Consortium, *Guidance for day 2 and beyond roadmap,* 2019.

[29] G. Naik, B. Choudhury y J.-M. Park, «IEEE 802.11bd & 5G NR V2X: Evolution of Radio Access Technologies for V2X Communications,» *IEEE Access,* vol. 7, pp. 70169-70184, 2019.

[30] ETSI, *EN 302 663 V1.2.1: Intelligent Transport Systems (ITS); Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band.*

[31] ETSI , *TS 102 724 V1.1.1: Intelligent Transport Systems (ITS); Harmonized Channel Specifications for Intelligent Transport Systems operating in the 5 GHz frequency band,* 2012.

[32] Comisión Europea, *Commission Decision 2020/1426 of 7 October 2020 on the harmonised use of radio spectrum in the 5 875-5 935 MHz frequency band for safety-related applications of intelligent transport systems (ITS) and repealing Decision 2008/671/EC,* 2020.

[33] ETSI, *ES 202 663 V1.1.0: Intelligent Transport Systems (ITS); European profile standard for the physical and medium access control layer of Intelligent Transport Systems operating in the 5 GHz frequency band,* 2010.

[34] ETSI, *EN 302 663 V1.3.1: Intelligent Transport Systems (ITS); ITS-G5 Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band,* 2020.

[35] ETSI, *TS 102 636-4-2 V1.1.1: Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 2: Media-dependent functionalities for,* 2013.

[36] Green Car Congress, «Volkswagen receives Advanced Award from Euro NCAP for traffic hazard alert Car2X function; based on Wi-Fi p - Green Car Congress,» 2020

marzo 20. [En línea]. Available: https://www.greencarcongress.com/2020/03/20200320-golf.html. [Último acceso: 2021 abril 13].

[37] B. Erdem, «IEEE 802.11bd – A seamless evolutionary access layer for ITS-G5 / DSRC,» *CAR 2 CAR Journal,* nº 23, pp. 21-27, 2019.

[38] Á. Knapp, A. Wippelhauser, D. Magyar y G. Gódor, «An Overview of Current and Future Vehicular Communication Technologies,» *Periodica Polytechnica Transportation Engineering,* vol. 48, nº 4, p. 341–348, 2020.

[39] B. Sun, *IEEE 802.11-18/1323r2: NGV SG Use Cases (Next Generation V2X Study Group),* 2018.

[40] B. Sadeghi, *IEEE 802.11-19/0497r7: 802.11bd Specification Framework Document,* 2020.

[41] S.-W. Ko, H. Chae, K. Han, S. Lee, D.-W. Seo y K. Huang, «V2X-Based Vehicular Positioning: Opportunities, Challenges, and Future Directions,» *IEEE Wireless Communications,* 2021.

[42] R. Molina-Masegosa y J. Gozalvez, «LTE-V for Sidelink 5G V2X Vehicular Communications: A New 5G Technology for Short-Range Vehicle-to-Everything Communications,» *IEEE Vehicular Technology Magazine,* vol. 12, nº 4, pp. 30-39, 2017.

[43] D. Sempere Garcia, «Redes 5G V2X Multi-modo y Escalables,» *Revista Doctorado UMH,* vol. 4, nº 2, p. 2, 2018.

[44] A. Bazzi, G. Cecchini, M. Menarini, B. M. Masini y A. Zanella, «Survey and Perspectives of Vehicular Wi-Fi versus Sidelink Cellular-V2X in the 5G Era,» *Future Internet,* vol. 11, nº 6, p. 122, 2019.

[45] R. Molina-Masegosa, J. Gozalvez y M. Sepulcre, «Configuration of the C-V2X Mode 4 Sidelink PC5 Interface for Vehicular Communication,» de *14th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, 2018.

[46] M. H. García Castañeda, A. Molina-Galán, M. Boban, J. Gozalvez, B. Coll-Perales, T. Sahin y A. Kousaridas, «A Tutorial on 5G NR V2X Communications,» *IEEE Communications Surveys & Tutorials*, pp. 1-1, 2021.

[47] A. Bazzi, A. O. Berthet, C. Campolo, B. M. Masini, A. Molinaro y A. Zanella, «On the Design of Sidelink for Cellular V2X: A Literature Review and Outlook for Future,» *IEEE Access*, vol. 9, pp. 97953-97980, 2021.

[48] M. M. Saad, M. T. R. Khan, S. H. A. Shah y D. Kim, «Advancements in Vehicular Communication Technologies: C-V2X and NR-V2X Comparison,» *IEEE Communications Magazine*, vol. 59, nº 8, pp. 107-113, 2021.

[49] R. Riebl, «Vanetza - Your open-source ETSI C-ITS protocol stack,» Technische Hochschule Ingolstadt, [En línea]. Available: https://www.vanetza.org. [Último acceso: 24 junio 2022].

[50] A. Voronov, «Geonetworking,» [En línea]. Available: https://github.com/alexvoronov/geonetworking. [Último acceso: 24 junio 2022].

[51] ETSI, *EN 302 636-4-1 V1.4.1: Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 1: Media-Independent Functionality*, 2020.

[52] ETSI, *EN 302 636-1 V1.2.1: Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 1: Requirements*, 2014.

[53] ETSI, *EN 302 636-6-1 V1.2.1: Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 6: Internet Integration; Sub-part 1: Transmission of IPv6 Packets over GeoNetworking Protocols*, 2014.

[54] ETSI, *TS 103 097 V2.1.1: Intelligent Transport Systems (ITS); Security; Security header and certificate formats.*, 2021.

[55] ETSI, *EN 302 636-5-1 V2.2.1: Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 5: Transport Protocols; Sub-part 1: Basic Transport Protocol*, 2019.

[56] ETSI, *EN 302 931 V1.1.1: Intelligent Transport Systems (ITS); Vehicular Communications; Geographical Area Definition*, 2011.

[57] R. Rielb. [En línea]. Available: https://github.com/riebl/vanetza. [Último acceso: 21 06 2021].

[58] ETSI, *EN 302 637-2 V1.3.2: Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service*, 2014.

[59] ETSI, *EN 302 637-2 V1.4.1: Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service*, 2019.

[60] ETSI, «ITS - Intelligent Transport Systems - GitLab,» [En línea]. Available: https://forge.etsi.org/rep/ITS. [Último acceso: 21 mayo 2022].

[61] Google, «Protocol Buffers | Google Developers,» [En línea]. Available: https://developers.google.com/protocol-buffers. [Último acceso: 3 junio 2022].

[62] ETSI, *TS 122 185 V14.3.0: LTE; Service requirements for V2X services (3GPP TS 22.185 version 14.3.0 Release 14)*, 2017.