





## **Resumen**

El desarrollo y despliegue de sistemas inteligentes en la carretera (ITS) trae consigo un incremento de la seguridad y una mayor rapidez a la hora de responder a eventos de emergencia. Este tipo de tecnologías han demostrado ser lo suficientemente robustas para utilizarse como apoyo para lidiar con la gestión de tráfico, mejora de la seguridad vial... y actualmente están apareciendo vehículos que además de tener un propio sistema de auto-diagnostico, pueden interactuar con su entorno.

## **Descriptores**

redes vehiculares, agentes vulnerables, ciudades inteligentes.



# Índice general

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1	Antecedentes . . . . .	3
<b>2</b>	<b>Solución</b>	<b>5</b>
2.1	Arquitectura del sistema . . . . .	5
2.2	Nube de Conductores . . . . .	5
2.2.1	Comunicación entre plataformas . . . . .	6
2.2.2	Formato de los mensajes . . . . .	7
2.2.3	Procesos . . . . .	8
2.3	Aplicación de ciclistas . . . . .	9
2.3.1	Modos de funcionamiento . . . . .	9
2.3.2	Comunicación con la nube . . . . .	11
2.3.3	Comunicación del grupo . . . . .	11
2.3.4	Casco BLE . . . . .	12
<b>3</b>	<b>Verificación del sistema</b>	<b>19</b>
<b>A</b>	<b>Apéndice</b>	<b>21</b>



## Índice de figuras

### Capítulo 2

2.1	Arquitectura del sistema . . . . .	6
2.2	<i>Hub</i> del líder . . . . .	10
2.3	Ingreso al grupo del seguidor . . . . .	10
2.4	UI de la salida . . . . .	11
2.5	Controlador de la salida . . . . .	12
2.6	Estructura de la comunicación GCM . . . . .	13
2.7	Comunicación líder-seguidor . . . . .	13





# Índice de tablas

## Capítulo 2

2.1	Formato de mensaje Vehículo a Motor . . . . .	8
2.2	Formato de mensaje Ciclista . . . . .	8
2.3	Formato de mensaje Ciclista . . . . .	8
2.4	Tipo de mensajes en grupo . . . . .	12



# Índice de algoritmos

## Capítulo 2

2.1	Envío de mensajes mediante GCM . . . . .	7
2.2	Envío de mensajes mediante GCM . . . . .	15
2.3	Formato de mensajes . . . . .	15
2.4	Cálculo de la proximidad de los vehículos . . . . .	15
2.5	Envío de peticiones desde la aplicación de ciclistas a la Nube de Ciclistas . . . . .	16
2.6	Servicio de cambio de LED . . . . .	17



## 1. INTRODUCCIÓN

---

Según la Directiva General de la Comisión Europea para el Transporte y la Movilidad, en 2014, algo más de 25.700 accidentes de tráfico fueron informados en la Unión Europea. Aunque el número de accidentes está decreciendo sustancialmente, el informe sobre Transporte ha anunciado un objetivo estratégico para la seguridad en las carreteras europeas para el período de 2011 a 2020: reducir el número de muertes en carretera a la mitad. Si las estadísticas son analizadas, en el período de 2010 a 2012, el número de ciclistas muertos en siniestros ha aumentado un 6 %; siendo el único agente de la carretera cuyos resultados vayan a peor. Esto se explica, al menos parcialmente, por un aumento de la presencia ciclista en la carretera. Se podría decir que el ciclismo es un medio de transporte donde los Usuarios Vulnerables de la Carretera (VRU) tienen un mayor contacto con el tráfico de mayor afluencia y velocidad. Cuando están involucrados en un accidente, son los que sufren las consecuencias más graves derivadas de una colisión con otro agente de la carretera; ya que están completamente expuestos a otros vehículos.

Basados en estos estudios, los accidentes en la que están involucrados VRUs ocurren frecuentemente en vías diseñadas para viandantes y ciclistas; por ejemplo en pasos de peatones y caminos para ciclistas cercanos a infraestructuras comunes de tráfico, las carreteras. Por lo tanto, la pregunta es: ¿Cómo se pueden reducir los accidentes de VRUs, y cómo minimizar la gravedad de un siniestro y sus consecuencias? Se pueden tomar varias soluciones: mejorar el diseño y trazado de las vías de comunicación, mejorar la iluminación, instalar más infraestructuras de protección, promocionar equipamiento de seguridad y enseñar cómo utilizarlo...

Sin embargo, hay otras soluciones viables aparte del re-diseño de las infraestructuras existentes, o soluciones pasivas como el uso del casco de seguridad. Una opción que está ganando fuerza en los países desarrollados es el desarrollo de soluciones para la movilidad en el entorno de ciudades inteligentes. Aunque el término de ciudad inteligente pueda parecer confuso, se podría decir que se considera *inteligente* cuando se ha aplicado tecnologías de la información y comunicación para mejorar la calidad de vida en áreas como la seguridad, gasto energético, reducción de costes, y gobierno y transporte, permitiendo una participación efectiva y activa por parte de los ciudadanos.

En el dominio de las Ciudades Inteligentes, las soluciones para transporte diseñadas tratan de hacer un uso más seguro, sostenible y eficiente de la carretera a través de un mejor entendimiento del estado de tráfico, la posición de los vehículos y usuarios, y el registro de eventos que suceden durante el transporte. Estas soluciones combinan la capacidad y beneficios de los sensores, dispositivos, infraestructura física y arquitecturas de comunicación combinada con sistemas de información en la nube, y la capacidad de analizar grandes volúmenes de datos.

En este contexto, los Sistemas Inteligentes de Información (ITS) emergen como una respuesta tecnológica para una mejor monitorización y caracterización del tráfico. Estos sistemas permiten al mismo tiempo mejorar el uso y eficiencia de la carretera, así como la seguridad de los usuarios, particularmente aquellos definidos como vulnerables; ciclistas, peatones o motoristas. Los ITS

## 1. INTRODUCCIÓN

actuales requieren el uso de cámaras de tráfico, paneles informativos, o sensores de inducción que obtengan datos para ser posteriormente mandados y procesados en la central de gestión de tráfico. A diferencia de estas soluciones que requieren el uso de sensores, actualmente surgen sistemas conocidos como Información Vehicular Flotante (FCD), que se encargan de reunir información de los sistemas de posicionamiento global (GPS) obtenidos de terminales móviles y el uso de páginas web colaborativas como Waze, que permite a los conductores obtener y proveer información sobre la carretera sin necesidad de ningún sensor en la carretera. Este tipo de soluciones basadas en FCD tienen a favor la monitorización del estado de los usuarios de manera ubicua, pero su fiabilidad depende del número de vehículos y usuarios informando sobre los eventos y aportando datos.

En el dominio de los ITS, los ITS Cooperativos (C-ITS) son sistemas que permiten la conexión directa entre vehículos (comunicaciones V2V) ó entre vehículos e infraestructuras (comunicaciones V2I) para intercambiar de información con el objetivo de mejorar la seguridad vial y la gestión del tráfico. Estos enlaces son posibles gracias a las unidades de abordo (OBU), dispositivos C-ITS dedicados que habilitan interfaces de comunicación, y dispositivos localizados en infraestructuras llamados RSU.

En un escenario C-ITS, hay generalmente cuatro agentes a considerar: dos entidades móviles (OBUs y peatones), y dos entidades estacionarias (la RSU y el sistema central). Estas entidades son capaces de ejecutar cuatro tipos diferentes de aplicaciones: seguridad activa en la carretera, tráfico eficaz cooperativo, servicios locales cooperativos, y servicios globales en Internet. Sobre cada tipo, hay diferentes definiciones de casos de uso y aplicaciones, donde cada agente puede ser considerado un sensor que genera información. Dependiendo de la aplicación y las restricciones de tiempo, el intercambio de información entre las entidades se puede clasificar como:

Mensajes de alerta: se definen como notificaciones descentralizadas y pueden ser enviados desde cada vehículo o RSU.

Mensajes de latido o *"beacons"*: son usados por los OBU para notificar su posición, velocidad e identidad a las RSU que forman parte del FCD. Además, estos mensajes también son usados para conocer la situación actual del tráfico. Por ello, el Acceso Inalámbrico en Entornos Vehiculares (WAVE) define los Mensajes de Aviso Cooperativo (CAMs), que son transmitidos periódicamente a todos los vehículos en área de alcance.

Mensajes sobre infotainment: notificaciones no relacionadas con la seguridad, sino que son usados para aportar mayor información y confort al conductor; datos turísticos, acceso a Internet, asistencia en navegación, etc.

En el campo de los servicios C-ITS, una gran variedad de aplicaciones y casos de uso se centran en incrementar la seguridad del usuario. Teniendo en cuenta requisitos estratégicos, económicos y de organización, características del sistema así como requisitos legales y de estandarización, el Comité del Instituto Técnico Europeo de Estándares en la Telecomunicación ha definido un conjunto básico de aplicaciones para usar como referencia en ITS para desarrolladores. Entre ellos, los Avisos a Usuarios Vulnerables en Carretera tratan de proveer notificaciones a los vehículos sobre la presencia de usuarios vulnerables, por ejemplo ciclistas, y en caso de existir situaciones de peligro también se avisa a los VRU sobre la presencia de un vehículo cercano.

Siguiendo los requisitos presentados por el ETSI, este proyecto presenta un sistema que emplea a los vehículos y ciclistas como sensores móviles que aportan información sobre su posición, velocidad y rumbo con el objetivo de detectar la proximidad entre estas dos entidades y avisarles en el caso de detectar peligro. Esta solución tiene un sistema centralizado que despliega comunicación inalámbrica vehicular, conectividad móvil y computación en la nube, y gestiona la información obtenida por los usuarios (vehículos y ciclistas). El sistema ha sido desplegado y verificado en un dominio real, y se han realizado pruebas de rendimiento en diferentes escenarios para comprobar el correcto funcionamiento de las comunicaciones en los peores escenarios.

Actualmente, existe un proyecto llamado Detección de agentes Inteligentes Cooperativos para la mejora de la eficiencia en el tráfico (ICSI), con similares aplicaciones que se encuentra bajo desarrollo usando una proximación descentralizada con la finalidad de mejorar el rendimiento y la seguridad.

## **1.1. ANTECEDENTES**

La importancia de las tecnologías C-ITS para la administración pública y la Comisión Europea está reflejada en la directiva 2010/40/EU, donde la UE reconoce la capacidad de los C-ITS para mejorar los sistemas de gestión de tráfico actuales y de dirigir los procesos de implantación y despliegue de estos sistemas en las carreteras europeas. Tras decenas de investigaciones y proyectos de desarrollo tales como Sistemas Cooperativos Vehículo-Infraestructura (CVIS), Sistemas Cooperativos para la seguridad en carretera *vehículos inteligentes en ciudades inteligentes* (SAFESPOT), El transporte flexible y adaptable del mañana (TEAM), el despliegue masivo de sistemas C-ITS se está acercando. Un ejemplo es el Memorandum del Entendimiento (MOU) firmado por la industria automovilística y organizaciones constructoras con el objetivo de comenzar a desplegar soluciones basadas en C-ITS en 2015. Las administraciones públicas también están trabajando en la misma dirección, resaltando el tratado que han pactado Alemania, Austria y Holanda para desplegar sistemas C-ITS en las vías que comunican estos tres países.

Se puede destacar de la misma manera el anuncio realizado en Febrero del 2014 por la Administración Nacional de Seguridad del Tráfico en Autopistas (NHTSA), que pertenece al Departamento de Transporte de los Estados Unidos (USDOT), sus intenciones para dar los pasos necesarios para el despliegue de sistemas V2V cooperativos en los próximos años, exactamente a partir de 2017, para vehículos comerciales.





## 2. SOLUCIÓN

---

### 2.1. ARQUITECTURA DEL SISTEMA

A grandes rasgos, existe en el medio del sistema una aplicación en la nube denominada *Nube de Conductores* que se encarga de hacer llegar los mensajes procedentes de los ciclistas a los vehículos a motor, y los mensajes enviados por los vehículos a motor a los ciclistas. En la figura 2.1 se puede observar de qué elementos está compuesto el sistema y cómo se comunican entre ellos.

Los vehículos a motor no se comunican directamente con la *Nube de Conductores*, sino que utilizan un dispositivo OBU para mandar mensajes a una unidad desplegada en carretera llamada RSU. Esta última recoge los mensajes que escucha a través de Broadcast y los reenvía a la nube por conectividad 3G.

Por otro lado, los ciclistas envían información a la *Nube de Conductores* a través de 3G o 4G; dependiendo de la disponibilidad. Éstos también pueden agruparse empleando *Wi-Fi 802.11* mediante la creación de un *HUB* de dispositivos móviles, en el cual se envían notificaciones sobre los eventos que aparezcan.

### 2.2. NUBE DE CONDUCTORES

El núcleo del sistema es una aplicación desplegada en la nube, la cual se ha denominado *Nube de Conductores*, donde se concentran en una base de datos la información relativa a ciclistas y vehículos a motor. Un servicio de aplicación web embebido llamado *Jetty* se encarga de recibir y atender los mensajes *HTTP/1.1* que son enviados desde la parte de vehículos a motor y ciclistas. Dos *Handler* independientes se encargan de filtrar los mensajes que no han sido correctamente contruidos, es decir, tienen un formato inválido, e insertar y actualizar los datos de la base de datos.

Para el despliegue de la aplicación se utilizado una máquina virtual *Ubuntu Server 14.04 LTS* que cuenta con 2048 MiB de memoria RAM y 2 núcleos para procesamiento. También se ha reservado un dominio público para que las peticiones puedan ser enviadas al servidor. Gracias a la herramienta *ANT* se puede cambiar fácilmente la plataforma donde se distribuya la aplicación, además esta configurada para poder ser ejecutada directamente con el comando *run*.

## 2. SOLUCIÓN

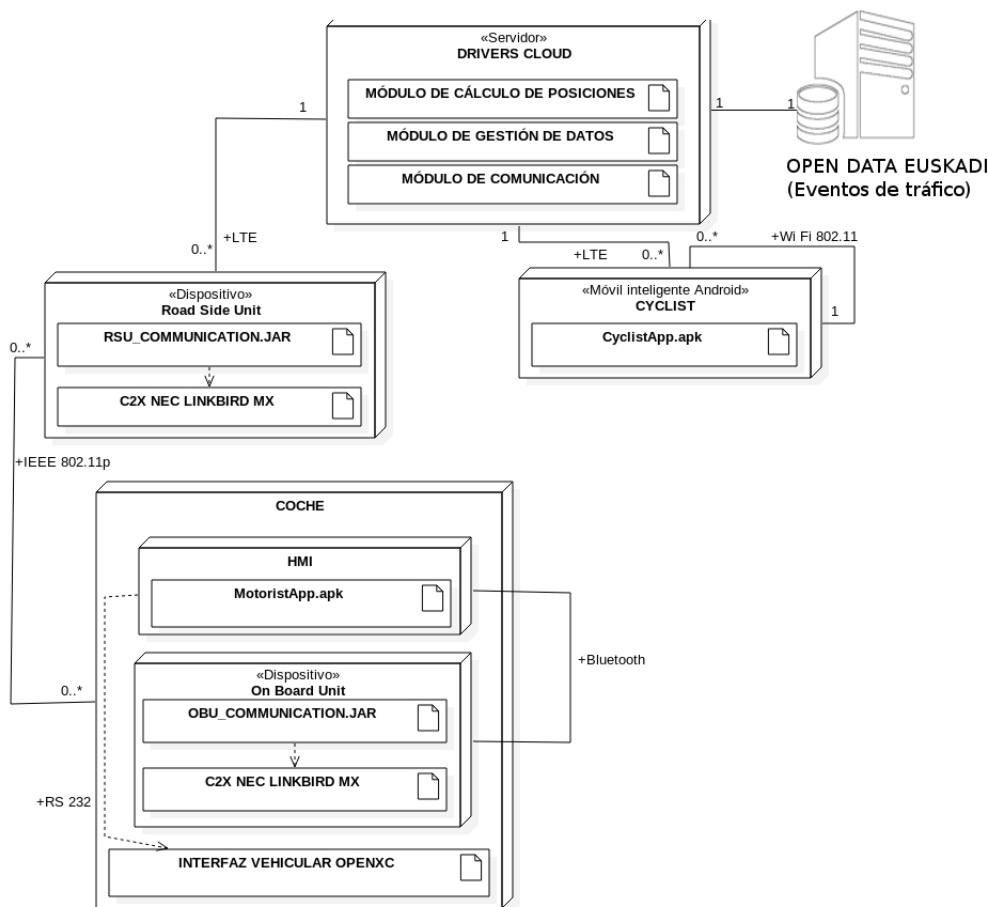


Figura 2.1: Arquitectura del sistema

### 2.2.1. Comunicación entre plataformas

La conexión entre la parte de los vehículos a motor y de los ciclistas hacia la nube se establece a través de tecnología móvil Long Term Evolution (LTE) o Third Generation (3G), dependiendo de la disponibilidad, aunque la manera de comunicarse con la *Nube de Conductores* es diferente.

### Mensajes a ciclistas

Se ha desarrollado una aplicación *Android* desde la cual se manda a la nube actualizaciones sobre la posición del usuario o el grupo que el usuario haya creado. Éste recibe notificaciones sobre las posiciones de los vehículos próximos, y otros diferentes eventos que pueden darse en la carretera; por ejemplo, un accidente de tráfico. Se ha contemplado la posibilidad de salidas en grupo de ciclistas, para ello se ha habilitado una modalidad específica mediante la cual se crea un grupo que se comunica entre sus miembros a través de una red privada *Wi-Fi 802.11*. Los diferentes miembros se mantienen actualizados sobre los diferentes eventos a través de un nodo denominado líder, el cual es el enlace a la nube tanto para reportar la posición del grupo de ciclistas como para recibir mensajes de la nube y retransmitir éstos al resto de miembros.

Para comunicarse con los dispositivos Android se emplea la plataforma Google Cloud Messaging

(GCM), la cual se encarga de gestionar que los mensajes lleguen al destino aunque el destino este temporalmente inaccesible mediante notificaciones *Push*. El mensaje debe respetar el formato que la API de GCM indica y puede observarse en el algoritmo 2.1, donde *ID\_ANDROID* es el identificador del dispositivo Android al que se le va a enviar el mensaje, y *DATOS* un objeto *JSON* con la información se que desea enviar.

```
1 { "registration_ids": [ "ID_ANDROID" ], data: { /*DATOS*/ } }
```

Algoritmo 2.1: Envío de mensajes mediante GCM

## Mensajes a vehículos a motor

Poseen en el vehículo un dispositivo *OBU* que permite comunicarse con la infraestructura en la carretera a través de una red *IEEE 802.11p*. A través de las *RSU* dispuestas en la carretera, las cuales actúan de intermediario, se envían y reciben los mensajes de la nube. Por tanto puede decirse, que las *RSU* actúan de *gateway* de comunicaciones entre los vehículos y las aplicaciones desplegadas en la *Nube de Conductores*. La *OBU* al recibir el mensaje lo muestra en un Interfaz Humano-Máquina (*HMI*) que posee el vehículo. La información del vehículo puede ser recogida a través de un interfaz *OpenXC* ó/y la *OBU*, dependiendo de la tecnología que se emplee para ello.

Los mensajes enviados a los vehículos a motor siguen el formato mostrado en la sección 2.2.2. La *Nube de Conductores* envía mensajes HTTP/1.1 a través del método POST un mensaje con contenido JSON a la *RSU*. Ésta se encarga de comunicarlo al vehículo a través de la red *IEEE 802.11p*.

### 2.2.2. Formato de los mensajes

Para poder realizar la conexión desde diferentes plataformas y entornos de desarrollo, se ha optado por buscar el diseño más abierto y flexible posible. Los datos son almacenados y transmitidos en formato plano con la codificación de caracteres *UTF-8*, para que puedan ser manipulados desde cualquier plataforma. Estos mensajes están contruidos en formato JavaScript Object Notation (*JSON*) para facilitar su análisis. A continuación, se muestra un ejemplo de la forma que tienen los mensajes recibidos de vehículos a motor:

En las siguientes secciones se explica en detalle el formato de los mensajes que son enviados y recibidos a través de la *Nube de Conductores*.

## Mensaje de posición de vehículo a motor

Indican la información geográfica de un vehículo. Los mensajes entrantes en la *Nube de Conductores* tienen que tener todos los campos indicados, mientras que los mensajes salientes se usarán los campos que sean necesarios.

## 2. SOLUCIÓN

Tabla 2.1: Formato de mensaje Vehículo a Motor

Tipo	Uso	Descripción
type	String	Identificador del tipo de mensaje. Su valor es <i>motorist_position</i> .
id	String	Identificador del vehículo. Se emplea el ID del router Linkbird-MX
timestamp	Integer	Marca de fecha y hora a la que se envía el mensaje.
latitude	Double	Latitud en la que se encuentra el vehículo.
longitude	Double	Longitud en la que se encuentra el vehículo.
altitude	Integer	Altitud en la que se encuentra el vehículo.
heading	Float	Dirección que mantiene el vehículo respecto al Norte magnético.
speed	Float	Velocidad a la que circula el vehículo.

### Mensaje de posición de ciclista

Indican la información geográfica de uno o más ciclistas. Los mensajes entrantes en la *Nube de Conductores* tienen que tener todos los campos indicados, mientras que los mensajes salientes se usarán los campos que sean necesarios

Tabla 2.2: Formato de mensaje Ciclista

Tipo	Uso	Descripción
type	String	Identificador del tipo de mensaje. Su valor es <i>cyclist_position</i> .
id	String	Identificador del vehículo. Se emplea el identificador de Android.
timestamp	Integer	Marca de fecha y hora a la que se envía el mensaje.
latitude	Double	Latitud en la que se encuentra el vehículo.
longitude	Double	Longitud en la que se encuentra el vehículo.
altitude	Integer	Altitud en la que se encuentra el vehículo.
heading	Float	Dirección que mantiene el vehículo respecto al Norte magnético.
speed	Float	Velocidad a la que circula el vehículo.
components	Integer	Número de ciclistas sobre los que se informa. Permite la creación de grupos de ciclistas.

### Mensaje de alerta

Cuando la *Nube de Conductores* detecta que un ciclista y un vehículo a motor tienen una gran probabilidad de encontrarse, se envía este tipo de mensaje para comunicar la distancia entre los vehículos y su posición relativa.

Tabla 2.3: Formato de mensaje Ciclista

Tipo	Uso	Descripción
type	String	Identificador del tipo de mensaje. Su valor es <i>alert</i> .
distance	String	Distancia a la que se encuentra un vehículo.
relative_angle	Integer	Ángulo relativo al que se encuentra el vehículo.

#### 2.2.3. Procesos

A través del API de *Jetty* la aplicación crea un servidor con dos manejadores de mensajes, uno para ciclistas y otro para vehículos a motor. A través de ellos la *Nube de Conductores* recibe datos

de ciclistas y vehículos a motor, almacenándolos en una base de datos interna sin necesidad de utilizar un DBMS, ya que no hace falta que los datos sean persistentes más tiempo de lo que los vehículos estén emitiendo su posición. Cada manejador posee un ThreadPooL con el que crea un gestor para cada mensaje recibido, este está limitado a un número de hilos para evitar que la aplicación se colapse.

Un registro se considera antiguo cuando no ha sido refrescado en un período de un minuto. Para evitar que emplee información obsoleta, se ejecuta una rutina que tan solo mantiene en memoria los registros que periódicamente están siendo actualizados; esto se realiza gracias al campo de *timestamp*.

Paralelamente, otro algoritmo compara las posiciones de los vehículos. Cuando se detecta que los vehículos a motor y los ciclistas están próximos - en un rango menor a 200 metros - se manda a ambos vehículos una alerta avisándoles de su proximidad [Algoritmo 2.4].

## 2.3. APLICACIÓN DE CICLISTAS

Con el objetivo de incrementar la seguridad de los ciclistas en las carreteras, se ha desarrollado una aplicación móvil. Esta aplicación permite propagar información sobre el tránsito de vehículos en la carretera y de esta forma, el ciclista puede colocarse en una mejor posición a la hora de ser adelantado por otro vehículo, ó puede saber qué se va a encontrar en una zona de visibilidad reducida antes de aproximarse.

Se ha elegido la plataforma Android debido al predominio de este sistema en el mercado actual, de esta forma se puede maximizar la recepción. Para este desarrollo se ha usado la API 23 de Android con retro-compatibilidad hasta la API 15.

Esta solución requiere de la *Nube de Conductores* para funcionar, ya que la información de los ciclistas es enviada a la misma y, de la misma forma, se puede recibir información sobre otros vehículos en la carretera.

### 2.3.1. Modos de funcionamiento

Existen dos modalidades de funcionamiento diferentes de la aplicación, ambas muestran la misma información al ciclista aunque su modo de proceder variará:

1. Modo individual: el usuario manda mensajes con su posición a través de *HTTP/1.1* a *Driver's Cloud*. Los mensajes provenientes de la nube son mandados al dispositivo mediante el servicio *GCM* de *Google*.
2. Modo grupal: uno de los terminales de los integrantes del pelotón actuará como HUB, y se encargará de gestionar todos los mensajes que lleguen desde la nube; se denomina *líder* del grupo. Este líder retransmitirá los mensajes a los demás miembros del grupo; denominados *seguidores*. Los mensajes que llegan al líder utilizan el mismo método que el modo de funcionamiento individual, pero al reenviar los mensajes que envían datagramas *UDP* dentro del *Hub*. El establecimiento de la comunicación se realiza de la siguiente forma:
  - a) El dispositivo que actúa como líder crea el *Hub* automáticamente al entrar en la opción *líder* de la aplicación.

## 2. SOLUCIÓN

- b) Los seguidores entran en el modo *seguidor* de la aplicación, y seleccionan el grupo al que desean ingresar. El dispositivo enviará una petición al líder.
- c) El líder al recibir una petición, la acepta o rechaza. Dependiendo si su dispositivo está sincronizando dispositivos o no.
- d) Si el líder ha aceptado la petición el seguidor queda a la espera hasta que el líder dé comienzo a la salida.
- e) En cuanto comience la salida el líder mandará mensajes a través de *broadcast* cada vez que reciba notificaciones de la nube.

En las figuras 2.2 y 2.3 se muestra la interfaz gráfica con la que se encuentra el usuario. Nótese en la interfaz del seguidor que puede buscar un grupo de dos maneras: (1) buscando el grupo de manera manual a través de una lista, ó (2) dejando que la aplicación auto-detecte una red y trate de unirse a ella.

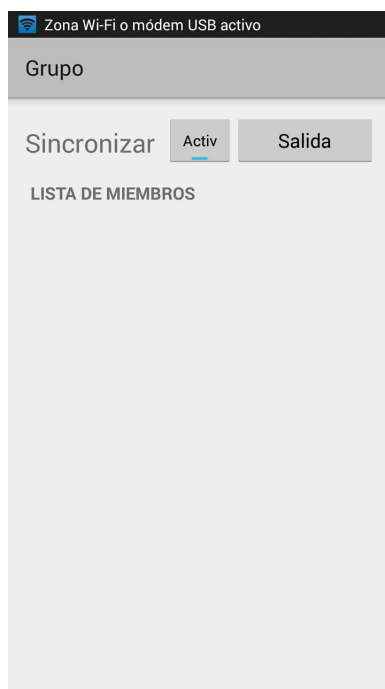


Figura 2.2: *Hub* del líder

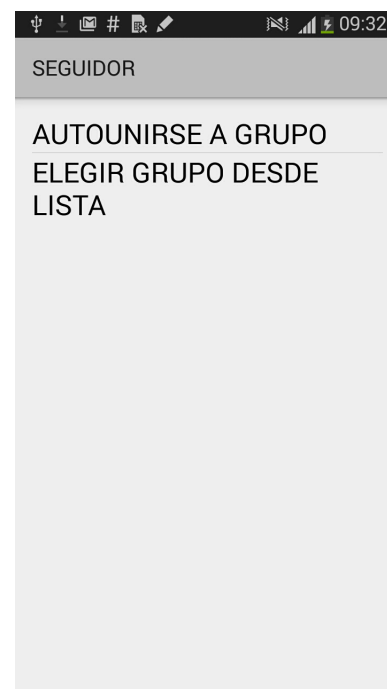


Figura 2.3: Ingreso al grupo del seguidor

Para mantener un registro de la ruta que se está realizando, un controlador mantiene toda la información sobre la salida que se está realizando. En la figura 2.5 se observa la estructura de este controlador, el funcionamiento es como siguiente:

**AJourney y AGroupJourney** interfaz gráfica que se muestra al usuario (figura 2.4).

**UIUpdateListener** escuchador de los eventos que se generan en cuanto un mensaje es recibido. Actualizará la interfaz gráfica para mostrar la información al usuario.

**GPSController** encargada de activar el *GPS* y subscribirse a las actualizaciones de posición. Se ha configurado para refrescar la posición cada dos segundos, cuando esto sucede se envía una notificación a la nube con los datos recogidos.

**JourneyController** gestor de la salida. Controla los datos relacionados con la salida: tiempo, distancia recorrida, ruta y calorías quemadas. Permite ser pausada y reanudada.

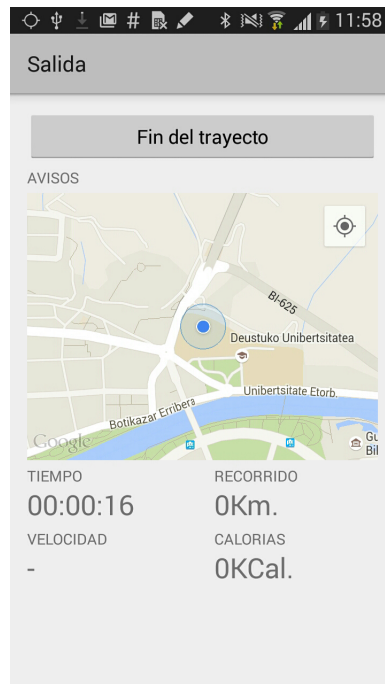


Figura 2.4: UI de la salida

### 2.3.2. Comunicación con la nube

Cuando la posición del ciclista es actualizada, se formatean los datos en un objeto JSON y se envían a la nube por medio de un mensaje *HTTP/1.1 POST*. El dominio del servidor es fijo, por lo que siempre se tendrá localizado la dirección de destino [Algoritmo 2.5]. Cuando el mensaje es recibido por la nube, la aplicación comprobará si el ciclista tiene algún peligro cerca. Si se detecta un vehículo cercano, la aplicación desplegada en la nube contestará con un mensaje de alerta con la información del vehículo detectado y la distancia que les separa.

Para la recepción de mensajes, la aplicación tiene que pedir un token de registro único del servidor *GCM (Google Cloud Messaging)*<sup>1</sup>. Para ello el dispositivo tiene que tener instalado los servicios *Google Play*. Una vez obtiene el token, cuando se envíe un mensaje al servidor se incluirá este identificador dentro del contenido para que la aplicación en el servidor pueda saber a qué dispositivo debe responder. Tras haber realizado la autenticación con los servicios de Google, un escuchador espera a nuevas notificaciones y los procesa una vez han llegado [2.6].

### 2.3.3. Comunicación del grupo

Entre los seguidores y el líder se enviarán notificaciones sobre el estado actual de cada nodo (Tabla ??) a través de del modo *Hub* que tienen los dispositivos; la explicación completa puede encontrarse en la subsección

Los mensajes son enviados a través del protocolo de transporte *UDP* para que la recepción de mensajes sea lo más rápido posible. Al ser un canal poco fiable se ha implementado una capa

<sup>1</sup>Servicio de mensajería ofrecido por *Google* para enviar y recibir mensajes desde diferentes plataformas.

## 2. SOLUCIÓN

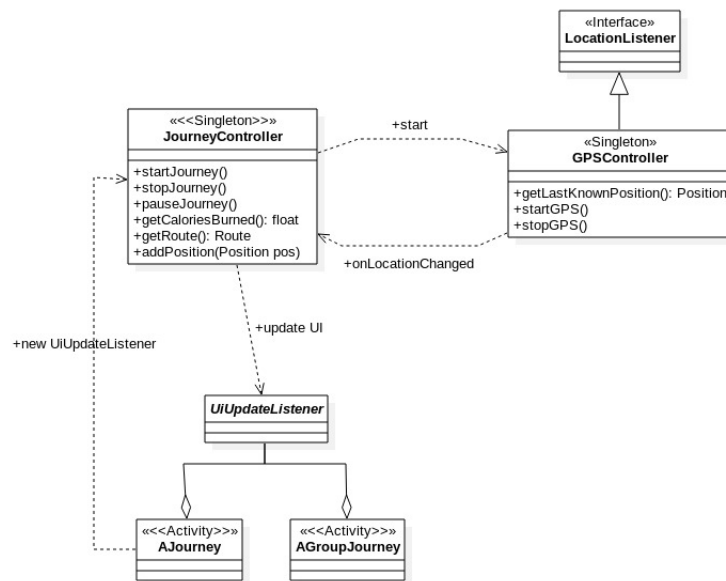


Figura 2.5: Controlador de la salida

Tabla 2.4: Tipo de mensajes en grupo

MENSAJE	Descripción	Campos extra
REGISTER	Petición de ingreso de un seguidor al <i>Hub</i> .	<i>nombre</i>
ACCEPT	Respuesta de aceptación de ingreso de un seguidor al <i>Hub</i> .	-
KICK	El administrador echa del <i>Hub</i> a un seguidor.	-
START	Notificación de comienzo de la salida.	-
STOP	Fin de una salida.	-
PAUSE	Notificación de pausa de la salida.	-
RESUME	Notificación de reanudado de la salida.	-
ALERT	Alerta por vehículo cercano.	Tabla 2.3
MOTORIST_POSITION	Posición de un vehículo.	Tabla 2.1

para garantizar la recepción del mensaje: cuando un mensaje es enviado, es guardado en un HashTable utilizando como clave la IP del destinatario y un número identificativo del mensaje<sup>2</sup>. El receptor mandará un ACK al emisor cuando un mensaje le llegue, y este último eliminará del HashMap el registro previamente almacenado. En caso de que un mensaje no llegue, un *timeout* provocará que el emisor vuelva a enviar el mismo mensaje al receptor; el *timeout* se incrementará al doble cada re-envío. En caso de que un mensaje no sea recibido al quinto intento, se dejará de intentarlo (figura 2.7).

### 2.3.4. Casco BLE

El ciclista no puede estar pendiente de los avisos de su Smartphone continuamente, ya que esto puede poner en riesgo su seguridad. Para que el ciclista pueda mantener la vista en la vía y tenga la posibilidad de saber si hay algún vehículo que pueda ponerle en riesgo, se ha integrado una

<sup>2</sup>El HashTable solo permite una clave, por lo que se ha creado una clase que calcula un Hash en base a las dos claves con que deseamos identificar el datagrama.



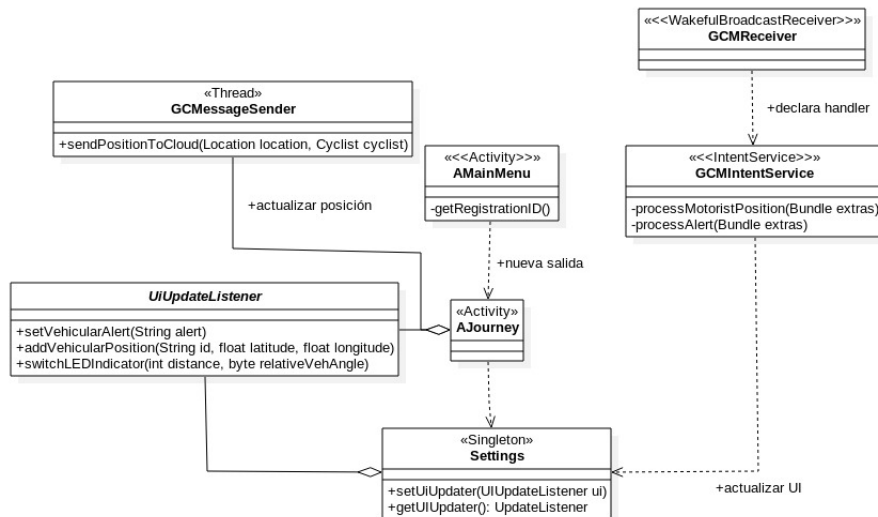


Figura 2.6: Estructura de la comunicación GCM

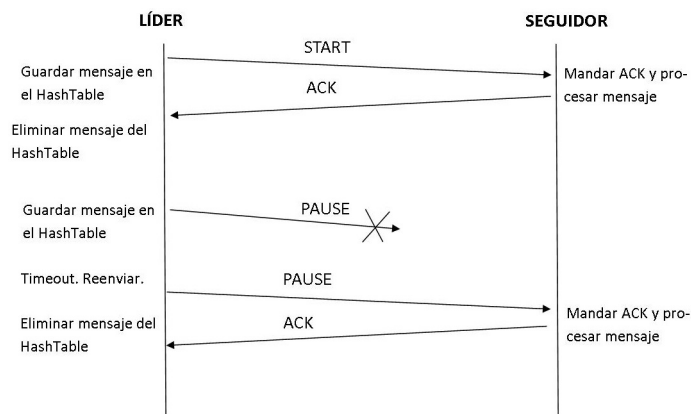


Figura 2.7: Comunicación Líder-seguidor

*mota Texas CC2540* en el casco del ciclista conectado a varios LEDs que según un código de colores le informan al ciclista sobre eventos que sean peligrosos. Si por ejemplo hay un vehículo acercándose por el lado izquierdo, un LED amarillo o rojo se encenderá, dependiendo si la distancia es menor de 50 ó 20 metros respectivamente.

Este dispositivo utiliza el estándar BLE para comunicarse con la aplicación móvil mediante cortos mensajes de 8 bytes, los cuales contienen un código hexadecimal que representa la combinación de LEDs que deben encenderse.

**Programación de la mota** La mota contiene un pequeño programa escrito en lenguaje C que se encuentra flasheado en su ROM (Read Only Memory). Este programa configura el microcontrolador para actuar como servidor (denominado *Central*), a la espera de ser emparejado y recibir mensajes. Hasta que se sincroniza con un dispositivo, cada 50 mili segundos propaga una señal para que los dispositivos puedan emparejarse. Cuando un dispositivo se conecta, el micro controlador espera a recibir un pequeño mensaje con el código de la señal que le especificará qué LEDs debe encender. Cuando llega el mensaje, si el código recibido es el correcto enciende el LED correspondiente. Para el desarrollo de este programa se ha

## 2. SOLUCIÓN

trabajado sobre una plantilla, que provee Texas Instrument junto con el dispositivo CC2540, al que se ha añadido el servicio necesario para encender el LED al recibir una señal. En el algoritmo 2.6 se explica cómo se ha implementado un nuevo servicio para gestionar los mensajes entrantes.

**Programación de la app** La aplicación de ciclistas actúa como cliente, por lo que el usuario debe primero emparejarse con el casco para que pueda comenzar a comunicarse con la moto. El proceso de conexión y envío de mensajes consiste:

1. Buscar los servicios Bluetooth disponibles.
2. Conectar al dispositivo en cuestión.
3. Descubrir los servicios que ofrece el dispositivo. Esto devolverá varios UUIDs con los servicios que tiene disponibles la moto. Una vez se sabe cuál es el servicio que controla la recepción de mensajes, hay que obtener una referencia.
4. Descubrir las características que contiene el servicio. Empleando la referencia del servicio, se pueden obtener uno o varios UUID que representan variables en las que se puede escribir un valor. Aquí es donde se depositará el código de combinación de LEDs que se desea encender.
5. Escribir sobre la característica que gestiona los LEDs.

```

1  DataOutputStream out;
2  HttpURLConnection httpRequest;
3  String inputLine;
4  StringBuffer response;
5  BufferedReader in;
6  final String KEY = AIzaSyAu2LXHx7_rP00UinzizQg5r5mgln4Q-Y;
7
8  try {
9      // abrir conexión con el gestor GCM
10     URL url = new URL("https://android.googleapis.com/gcm/send");
11
12     httpRequest = (HttpURLConnection) url.openConnection();
13
14     // enviar datos mediante POST
15     httpRequest.setRequestMethod("POST");
16
17     // establecer el encabezado
18     httpRequest.setRequestProperty("Content-Type", "application/json");
19     httpRequest.setRequestProperty("Authorization", "key=" + KEY);
20     httpRequest.setDoOutput(true);
21
22     // preparar información
23     out = new DataOutputStream(httpRequest.getOutputStream());
24
25     // enviar información
26     out.write(message.getBytes());
27     out.flush();
28     out.close();
29     logger.debug("Sending 'POST' request to URL : " + url);
30
31     // obtener respuesta
32     in = new BufferedReader(new InputStreamReader(httpRequest.getInputStream()));
33     response = new StringBuffer();
34
35     while ((inputLine = in.readLine()) != null) response.append(inputLine);
36     in.close();
37     logger.debug("Response code : " + response.toString());
38
39 } catch (IOException e) {
40     logger.error(e.getMessage());
41 }

```

Algoritmo 2.2: Envío de mensajes mediante GCM

1

Algoritmo 2.3: Formato de mensajes

```

1  for (Motorist m : lMotorist) {
2      for (Cyclist c : lCyclist) {
3          if (isCollisionDanger(m, c)) {
4              sendWarningToMotorist(c);
5              sendCyclistPositionToMotorist(c);
6          }
7      }
8  }

```

Algoritmo 2.4: Cálculo de la proximidad de los vehículos

## 2. SOLUCIÓN

```
1 HttpClient httpClient;
2 HttpPost httpPost;
3 String data;
4
5 data="{\"id\": \"" + cyclist.getIdentifier() + "\", " +
6     "\"type\": \"" + "cyclist_position\", " +
7     "\"latitude\": \"" + cyclist.getPosition().getLatitud() + "\", " +
8     "\"longitude\": \"" + cyclist.getPosition().getLongitud() + "\", " +
9     "\"altitude\": \"" + cyclist.getPosition().getAltura() + "\", " +
10    "\"heading\": \"" + cyclist.getPosition().getRumbo() + "\", " +
11    "\"speed\": \"" + cyclist.getSpeed() + "\", " +
12    "\"components\": \"" + cyclist.getPersonas() + "\", " +
13    "\"timestamp\": \"" + new Timestamp(new Date().getTime()) + "\"}";
14 httpClient = new DefaultHttpClient();
15 httpPost = new HttpPost("http://cloud.mobility.deustotech.eu/cyclist");
16 httpPost.setEntity(new StringEntity(data));
17 httpClient.execute(httpPost);
```

Algoritmo 2.5: Envío de peticiones desde la aplicación de ciclistas a la Nube de Ciclistas

```

1 // Declarar el UUID del perfil ATT (Attribute Protocol)
2 #define PROFILE_VEHICULAR_POS 5
3
4 // Declarar el UUID del servicio
5 #define SIMPLEPROFILE_SERV_UUID 0xFFFF0
6
7 //Declarar el UUID de la característica, darle un tamaño y asignar
8 //permisos de escritura
9 #define PROFILE_VEHICULARPOS_UUID 0xFFFF6
10 CONST uint8 simpleVehicularPositionProfileUUID[ATT_BT_UUID_SIZE] = {
11     ILO_UINT16(PROFILE_VEHICULARPOS_UUID), HI_UINT16(PROFILE_VEHICULARPOS_UUID)
12 };
13 static uint8 vehicularPositionProps = GATT_PROP_WRITE;
14 static uint8 vehicularPosition = 0;
15
16 // gestionar que se ha hecho una escritura en la característica
17 static bStatus_t simpleProfile_writeAttrCB(uint16 connHandle, gattAttribute_t *PAttr,
18     uint8 *pValue, uint8 len, uint16 offset) {
19     [...]
20     notifyApp = PROFILE_VEHICULARPOS;
21     [...]
22 }
23
24 // gestionar la petición
25 static void simpleProfileChangeCB(uint8 paramID) {
26     [...]
27     case PROFILE_VEHICULARPOS:
28         SimpleProfile_GetParameter(PROFILE_VEHICULARPOS, &newValue);
29         cambiarLED(paramID);
30     [...]
31 }
32
33 // escritura de la característica
34 bStatus_t SimpleProfile_SetParameter(uint8 param, uint8 len, void* value) {
35     [...]
36     case PROFILE_VEHICULARPOS:
37         if (len == SIMPLEPROFILE_CHAR5_LEN) {
38             vehicularPosition = *((uint8 *) value);
39         } else {
40             ret = bleInvalidRange;
41         }
42         break;
43     [...]
44 }

```

Algoritmo 2.6: Servicio de cambio de LED

## 2. SOLUCIÓN

```
1 public void conectarBLE(Device dispositivo) {
2     // El primer argumento indica que la propia clase gestionará los eventos,
3     // el segundo argumento que se autoconectará al dispositivo, y el tercer
4     // argumento a qué dispositivo va a conectarse.
5     bluetoothGatt = device.connectGatt(this, false, dispositivo);
6 }
7
8 public void mandarMensajeBLE(byte msg) {
9     // obtener el servicio que contiene la característica que se va a modificar
10    servicio = bluetoothGatt.getService(UUID.N_SERVICIO);
11
12    // obtener la característica (local)
13    caracteristica.getCharacteristic(msg);
14
15    // modificar el valor de la característica (local)
16    caracteristica.setValor(msg);
17
18    // aplicar cambios en el dispositivo remoto
19    bluetoothGatt.writeCharacteristic(caracteristica);
20
21
22 }
```

Algoritmo 2.7: Envío de mensajes LED desde la aplicación de ciclistas

### **3. VERIFICACIÓN DEL SISTEMA**

---





## **A. APÉNDICE**

---

