Trabajo de Fin de Grado

SISTEMA INTELIGENTE DE DETECCIÓN Y AVISO DE INFRACCIONES EN SEMÁFOROS MEDIANTE SMARTPHONES

"Intelligent System of Detection and Warning of Traffic Light Infringements through Smarpthones"

RUSHIL LAKHANI LAKHANI

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS
ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

La Laguna, 7 de julio de 2015

Dª. Pino Caballero Gil con N.I.F. 45534310-Z profesora Titular de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. Francisco Martín Fernández, con N.I.F. 78629638-K contratado FPI adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A N

Que la presente memoria titulada:

"Sistema Inteligente de Detección y Aviso de Infracciones en Semáforos mediante Smartphones"

ha sido realizada bajo su dirección por D. Rushil Lakhani Lakhani, con N.I.F. 78727722-X.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de julio de 2015

Licencia

Agradecimientos

La realización del presente trabajo de fin de grado no habría sido posible sin la ayuda de mi tutora Pino Caballero Gil, cuyo compromiso y dedicación ha sido total para que el trabajo haya podido realizarse. La otra persona gracias a la cual he podido realizar este proyecto ha sido mi cotutor Francisco Martín Fernández, que ha puesto todo el empeño para que este proyecto pudiera salir adelante, con el que siempre he podido contar ante cualquier duda que se me presentaba a lo largo del trabajo, y que ha estado conmigo en todo momento apoyándome y guiándome para conseguir la elaboración de dicho trabajo. Además la colaboración del resto del grupo CryptULL ha sido de gran ayuda.

Resumen

Las tecnologías de las comunicaciones nos están cambiando la vida en todos los ámbitos. En España tenemos más de 18 millones de teléfonos inteligentes y hay, al menos, 12 millones de personas que usan a diario aplicaciones móviles. Los constantes avances en la industria de la telefonía móvil han servido, entre otros muchos logros, para poner a nuestro alcance multitud de funciones que facilitan nuestro día a día.

Además, en los últimos años se ha incrementado notablemente el interés por las redes ad-hoc que permiten a los usuarios de dispositivos móviles con tecnología inalámbrica (Bluetooth, Wi-Fi) conectarse entre sí sin necesidad de usar infraestructuras centralizadas para intercambiar información entre ellos. En particular, destacamos las redes ad-hoc formadas por vehículos, conocidas como VANETs (Vehicular Ad-hoc NETworks). Con estas redes se quiere dotar a los conductores y peatones de un ecosistema inteligente donde poder desplegar multitud de aplicaciones para prevenir diferentes circunstancias adversas que pueden ocurrir en las carreteras.

En este trabajo se presenta una aplicación que va a hacer de las carreteras un sitio más seguro, previendo posibles accidentes de conductores temerarios. Se propone una nueva aplicación vial aplicable al ecosistema de las VANETs y desarrollada mediante teléfonos inteligentes, que permita detectar cuando un vehículo se salta un semáforo, y avisar inmediatamente a los conductores cercanos de forma segura mediante la protección de anonimato, integridad y confiabilidad.

Dicha aplicación permite notificar en tiempo real a todos los vehículos cercanos para informarles de que hay otro vehículo a su alrededor cometiendo infracciones graves de tráfico con objeto de que dichos usuarios puedan estar prevenidos de otras posibles infracciones que desembocaran en accidentes de tráfico. El objetivo por tanto no es en absoluto la tramitación de multas.

Para ello se hace uso de diversas tecnologías tales como Android, sistema operativo en el que se ha desarrollado la aplicación, y NodeJS, servidor utilizado para poder comunicarse con la aplicación y viceversa. Estas dos tecnologías son las principales bases sobre las que se asienta la aplicación móvil. También se hace uso de algoritmos criptográficos para garantizar el anonimato de la persona que ha denunciado la infracción cometida, con objeto de fomentar el uso de la aplicación.

Palabras Clave

Android, Sistema inteligente, Hash SHA-512, Smartphone, Seguridad, VANET, NodeJS

Abstract

Communication technologies are changing our lives in many different areas. In Spain, we have more than 18 million smartphones, and at least 12 million people use mobile applications. Constant advances in mobile phone industry have served, among many other accomplishments, for providing us with a multitude of features that make our daily life easier.

Besides, in the last few years there has been an increased interest in ad-hoc networks that allow smartphone users with wireless technology (Wi-Fi, Bluetooth) to connect with each other without using infrastructures to exchange information among them. In particular, we highlight ad-hoc networks formed by vehicles, known as VANETs (Vehicular Ad-hoc NETworks). These networks will provide drivers and pedestrians with an intelligent ecosystem where multiple applications can be deployed to prevent various adverse circumstances that may occur on the roads.

This project focuses on an application that will make the roads a safer place by anticipating possible accidents of dangerous drivers. The proposal is a new application useful in the vial ecosystem of VANETs, developed by smartphones that allow detecting when a vehicle skips a traffic light and sending warning to nearby vehicles, protecting anonymity, integrity and reliability.

All nearby vehicles are notified in real time to inform them that there is another vehicle around committing serious traffic offenses so that those users are immediately warned of possible malignant behaviors that can result in accidents. Therefore, the aim is not the processing of fines.

The application makes use of various technologies such as Android, operating system in which the application has been developed, and NodeJS, server to communicate with the application and vice versa. These two technologies are the bases where the foundations for the mobile application lay. It also uses a cryptographic algorithm to protect the anonymity of the person reporting the offense, in order to promote the use of the application.

Keywords

Android, Intelligent System, Hash SHA-512, Smartphone, Security, VANET, NodeJS

Índice general

Índice de figuras

Índice de tablas

1. Antecedentes

1.1 Motivación

Los smartphones han cambiado totalmente nuestro día a día. Hay móviles que sacan fotos, marcan el ritmo de la música en las fiestas, proyectan videos, y se convierten en el GPS del camino y de la vida. El mundo de las comunicaciones ya no es como antaño. Ha dado un giro de 180 º con la entrada de estos dispositivos. De hecho no se puede concebir la sociedad actual sin smartphones, conectados continuamente a Internet y permitiendo saber lo que ocurre en todo momento y en todo lugar, proporcionando nuevas formas de hacer el trabajo y facilitando muchas tareas. Los usuarios consultan el correo electrónico, leen libros o el periódico e incluso realizan compras con estos dispositivos. La aparición y extensión del uso de los smartphones ha supuesto un cambio rotundo en la forma de hacer las cosas y en la manera en la que las personas están interconectadas y se relacionan entre sí.

El uso de teléfonos inteligentes se ha disparado de forma exponencial en los últimos años. España cuenta con más de 18 millones de teléfonos inteligentes y es ya el país líder europeo en penetración de los smartphones. Cuatro de cada cinco móviles en nuestro país son inteligentes, dato que revela el sorprendente impacto que están teniendo estos dispositivos en nuestra vida cotidiana.

Se denomina smartphones a la familia de teléfonos móviles que disponen de hardware y sistema operativo capaces de realizar tareas y funciones similares a las realizadas por los ordenadores de sobremesa o portátiles, añadiéndole al teléfono funcionalidades extra a la realización y recepción de llamadas y mensajes telefónicos. En particular, el sistema operativo más habitual en este tipo de dispositivos es Android. En la Fig.1 se muestra un smartphone con el logotipo del sistema operativo Android.

Fig. 1. Smartphone Android

1.2 Objetivos

Debido al gran auge que los smartphones están teniendo en nuestra sociedad, cada vez se desarrollan más aplicaciones orientadas al mundo de los smartphones. Como se ha dicho anteriormente, la tecnología y la comunicación nos están cambiando la vida, lo que se extiende a todos los ámbitos, incluidos el del tráfico y la conducción. En España un 17% de las aplicaciones utilizadas en smartphones tiene que ver con servicios de información, y la relacionada con el tráfico vial es una de las más demandadas.

Todo ello, junto al interés que en la última década han alcanzado las redes inalámbricas ad-hoc, parece presagiar la implementación de una nueva aplicación útil para el ecosistema de las redes vehiculares o VANETs de manera sencilla, económica y práctica desarrollada mediante teléfonos inteligentes dada la gran capacidad de cómputo de estos dispositivos.

El propósito de este proyecto es dotar de una nueva aplicación vial al ecosistema de las VANETs, desarrollada mediante teléfonos inteligentes. En particular, esta aplicación permitirá detectar cuándo un vehículo se salta un semáforo en rojo con objeto de notificar en tiempo real a todos los vehículos cercanos informando que existe un vehículo cerca que ha cometido una infracción grave de tráfico que puede suponer un potencial peligro, todo ello de forma segura mediante la protección de anonimato, integridad y autenticidad. De esta manera, los usuarios pueden prepararse para prevenir posibles comportamientos peligrosos que puedan desembocar en accidentes de tráfico.

El conjunto de objetivos concretos de este trabajo se pueden desglosar en los siguientes puntos:

- Estudio de las tecnologías requeridas tales como (Android Studio, MongoDB, NodeJS, Express, Android SDK…
- Implementación del cliente (aplicación) en Android
- Implementación del back-end del servidor en Javascript
- Interacción Cliente-Servidor
- Creación de una interfaz amigable a la Aplicación Móvil
- Implementación de la capa de seguridad en las comunicaciones
- Testear el comportamiento de todo el sistema
- Diseño, implementación, prueba y evaluación de la aplicación móvil que ponga en práctica los resultados obtenidos de los objetivos anteriores.

1.3 Fases del Desarrollo

El desarrollo de este trabajo se ha dividido en diferentes fases:

- **Febrero-Marzo:**
    - Documentación y Manejo de las tecnologías a utilizar tales como:
        - Android SDK
        - MongoDB
        - NodeJS
        - Express
        - Google Cloud Messaging
    - Estudio del estado del arte y de las aplicaciones y dispositivos existentes.
    - Descripción de los requisitos del sistema a desarrollar.

- **Abril:**

    - En este período se realizaron las siguientes tareas:
        - Creación de la app móvil nativa para Android
        - Creación del back-end del servidor usando NodeJS, ExpressJS y MongoDB.
        - Comunicación e integración de la parte móvil con la parte de back-end.
        - Creación de una interfaz del usuario en la aplicación móvil siguiendo los paradigmas del diseño de Android.

- **Mayo:**

    - En este período se realizaron las siguientes tareas:
        - Implementar un esquema de comunicaciones seguro entre la app y el back-end
        - Presentación del prototipo y diseño de la aplicación testeado.

1.4 Estructura de la Memoria

El contenido restante de esta memoria está organizado de la siguiente manera:

- Capítulo 1. Se realiza una introducción sobre los smartphones, los objetivos del proyecto, las fases del desarrollo y la estructura de la memoria.

- Capítulo 2. Se realiza una introducción acerca de las VANETS, investigación de proyectos relacionados con el desarrollado y ventajas de la aplicación frente a estos proyectos.

- Capítulo 3. Se realiza una definición preliminar acerca de las tecnologías que se van a utilizar en él proyecto.

- Capítulo 4. Se realiza una definición sobre los sensores integrados en los semáforos, las características de éstos y sus tipos.

- Capítulo 5. Se realiza una definición sobre los semáforos inteligentes, características, tipos, funcionamiento y la comunicación entre el semáforo y la aplicación.

- Capítulo 6. Servidor de la aplicación. Se explicara en profundidad todas las funcionalidades del servidor, la estructura del mismo, la base de datos que integra y por último la comunicación entre el servidor y la aplicación.

- Capítulo 7. Aplicación desarrollada. Se explicara con detalle el funcionamiento de la aplicación, su estructura, la comunicación entre la aplicación y el servidor, la interfaz de la aplicación y por último un caso de uso que muestra el funcionamiento que se sigue tras generar el evento del salto de semáforo.

- Capítulo 8. Seguridad de la aplicación (Anonimato, Firma Digital, …)

- Capítulo 9. Se presenta un presupuesto del coste total del proyecto.

- Capítulo 10. Recoge las conclusiones del trabajo, y posibles mejoras de la aplicación.

2. Estado del Arte

2.1 Introducción

Las redes vehiculares o Vehicular Ad hoc NET Works (VANETs) están consideradas como la tecnología perfecta para proporcionar a los vehículos capacidades de comunicación que se pueden aplicar a la mejora de la seguridad vial.

El acceso a Internet desde las VANETs se puede proporcionar a través de puertas de enlace situadas al borde de la carretera de manera que los vehículos cambian su punto de conexión a Internet al moverse. Esto permite a conductores y pasajeros utilizar servicios comunes de Internet y nuevas aplicaciones que estén especialmente orientadas para ellos, como por ejemplo las aplicaciones para mejorar la eficiencia del tráfico en carreteras y áreas urbanas. Estos servicios servirán como reclamo para los usuarios, lo que permite acelerar la penetración de la tecnología en el mercado.

Muchos automóviles ahora incluyen cámaras u otros sensores que registran el entorno que les rodea y desencadenan un comportamiento inteligente, tales como el frenado automático o de dirección para evitar un obstáculo.

2.2 Audi Travolution

Audi ha presentado un sistema capaz de comunicar semáforos y coches, Audi Travolution [1]. Mediante la conexión con la red local, el vehículo se adelantará y sabrá en todo momento el estado del semáforo al que nos acerquemos, con lo que el propio ordenador determinará si hay bastante tiempo como para pasar con la luz verde o tendremos que pararnos. Gracias a esta información se mejorará la eficiencia del vehículo, con lo que sería posible ahorrar hasta un 15% en combustible y por tanto, también en energía eléctrica.

En la Fig.2 se muestra el cuadro de mandos de un vehículo inteligente diseñado por Audi que permite determinar el estado del semáforo.

Fig. 2. Cuadro Audi Travolution

Una idea muy sencilla pero efectiva, con la que se pueden prevenir pérdidas de energía ineficientes, anticipándose al momento en el que el semáforo cambiará su estado y permitirá reajustar la velocidad del coche para llegar al siguiente semáforo cuando esté en verde. De la misma manera, si el vehículo se encuentra parado, el sistema podrá ir activando el arranque automático en el momento preciso, con el conocido sistema Start-Stop.

2.3 Nvidia DRIVE PX

Nvidia, una marca líder de chips gráficos de ordenador, presentó en la última edición del CES de las Vegas, la feria tecnológica más importante del mundo, una computadora para el vehículo llamado Nvidia drive PX [2], unidad que podría ayudar a los coches a interpretar y también reaccionar ante el mundo que les rodea.

Nvidia Drive parte de la plataforma del Tegra X1[3] y aprovecha esa potencia sin grandes necesidades energéticas ni de disipación de calor para convertir los coches en inteligentes, más allá de las posibilidades que ofrecen los ordenadores actuales. Está dividido en Nvidia Drive PX y Nvidia Drive CX [4].

Nvidia Drive PX es una plataforma de piloto automático que usa dos procesadores Tegra X1 para procesar las imágenes que le llegan de hasta 12 cámaras en resolución 4K a 30 fps, procesando hasta 1.3 Gigapíxeles por segundo. El sistema puede usar la información obtenida de estas imágenes para conducir el coche, como por ejemplo con una función que busca automáticamente espacio para aparcar y ejecuta la maniobra solo; a diferencia de otros asistentes al aparcamiento, Nvidia Drive PX no necesita que le definan un lugar concreto para aparcar sino que lo busca.

La plataforma DRIVE PX está basada en el procesador NVIDIA® Tegra® X1. Permite diseñar sistemas avanzados de ayuda a la conducción (ADAS) que abrirán paso a los vehículos autónomos. Combina tecnología de visión computarizada (CV) del entorno, un completo sistema de aprendizaje profundo y actualizaciones OTA para transformar la manera en que los vehículos ven, piensan y aprenden.

En la Fig.3 se muestra la estructura de un vehículo que incorpora las características de la marca NVIDIA con la plataforma DRIVE PX y Drive CX integradas.

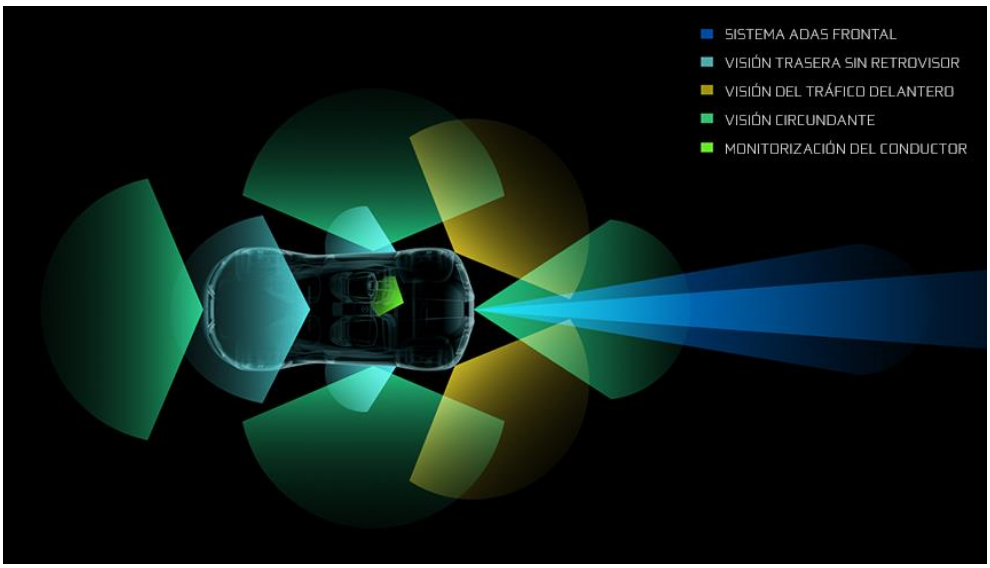


Fig. 3. Funciones NVIDIA DRIVE PX

La tecnología ADAS convencional es capaz de detectar algunos objetos, realizar algunas operaciones de clasificación básicas, avisar al conductor y en algunos casos, detener el vehículo. DRIVE PX lleva estas capacidades a otro nivel e introduce la posibilidad de diferenciar una

ambulancia de un camión de reparto, o un vehículo estacionado de otro que va a incorporarse a una vía de circulación. Ahora, el sistema puede informar al conductor, no solo llamar su atención con una señal de aviso. El vehículo ya no se limita a detectar, sino que entiende lo que ocurre a su alrededor, una capacidad fundamental para pasar a la conducción con piloto automático.

En la Fig.4 se muestra el uso de la tecnología ADAS, que es capaz de detectar como podemos observar todos los objetos a su alrededor. En este caso se distingue varios vehículos y un ciclista.

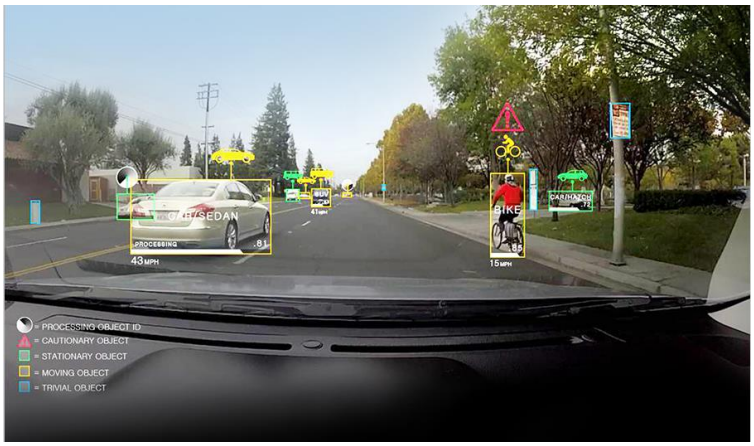


Fig. 4. Funcionalidad Tecnología ADAS

La potencia del sistema gráfico permite a DRIVE PX presentar un coche virtual con modelos altamente detallados y efectos de iluminación de gran realismo que muestran al conductor una imagen muy similar a su vehículo en lugar de una maqueta o un modelo genérico.

En la Fig.5 se muestra el modelo de un coche virtual gracias a la plataforma de piloto automático, NVIDIA Drive PX.

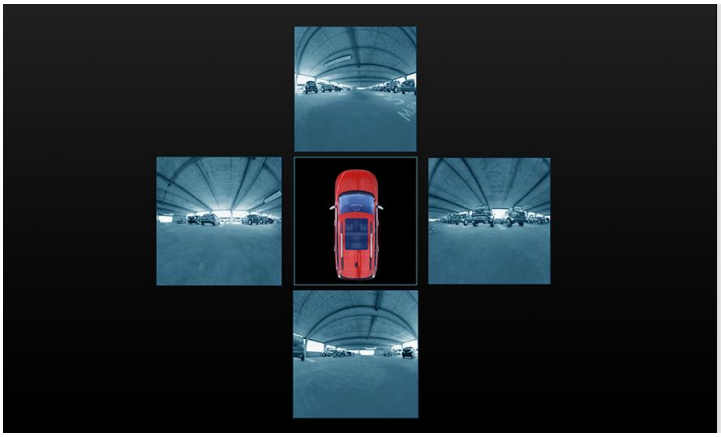


Fig. 5. Modelo de Coche Virtual

DRIVE PX proporciona la inmensa capacidad de proceso necesaria para introducir técnicas de SFM (estructuras obtenidas del movimiento) y de SLAM (localización y mapeo simultáneos) de cuatro cámaras de visión circundante que cubren toda la zona que rodea el vehículo.

En la Fig.6 se muestra los sensores integrados en el vehículo haciendo uso de la plataforma de piloto automático, NVIDIA Drive PX.

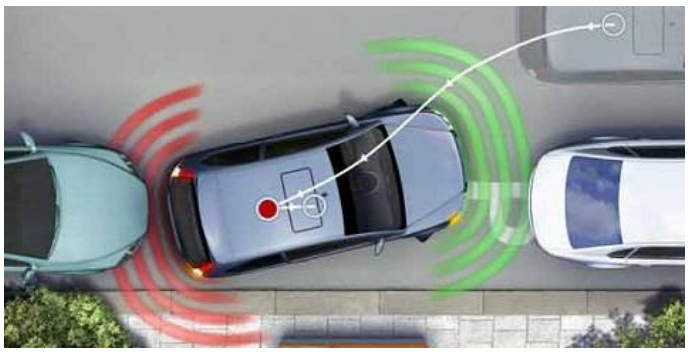


Fig. 6. Sensores NVIDIA DRIVE PX

2.4 Honda

Honda [5] va a liberar una flota de 100 coches con los que validar durante un año, en uso real, la comunicación entre coches y semáforos.

En la Fig.7 se muestra el funcionamiento del sistema de comunicación de un coche Honda basado en balizas infrarrojas al paso del vehículo gracias al cual puede comunicarse con el semáforo.

Fig. 7. Funciones del Prototipo Honda

La clave del sistema es que el coche dotado del sistema de comunicación (basado en balizas infrarrojas al paso del vehículo), es capaz de recibir e interpretar información de la frecuencia de los ciclos de los semáforos de las calles por las que transita.

El sistema se encarga de recomendar una velocidad de circulación en el cuadro de instrumentos para facilitar que el conductor llegue en verde a los siguientes semáforos. En caso de que a la distancia y velocidad de la vía vaya a encontrarse con semáforo rojo, el sistema sugerirá al conductor que levante el pie del acelerador. Se espera que estos sistemas de comunicación puedan reducir tanto el gasto de combustible como la espera en el semáforo, dulcificando además la conducción. De manera colateral, al variar los patrones de conducción se espera también una reducción de accidentes derivada de la aplicación del proyecto.

2.5 IBM

Los ingenieros de IBM [6] han sacado a la luz un semáforo capaz de detener los coches en función del color de sus discos. Lo detiene, apagando el motor directamente. Se denomina **“**sistema y método para el control y funcionamiento del motor de un vehículo en intersecciones con mayor eficiencia de consumo de combustible” se dirige a lograr un modo de gestión ecológica de motores en respuesta a una señal de tráfico.

En la Fig.8 se muestra el prototipo de IBM que consiste en un semáforo capaz de detener los coches en función del estado del mismo.

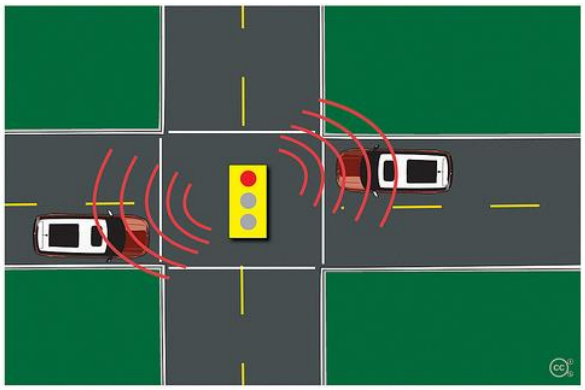


Fig. 8. Prototipo IBM

Se trata de un sistema que puede recibir datos de posición de todos los vehículos que esperan en un semáforo en rojo y enviar una notificación de “stop-motor” a los coches que están aguardando más de un período de tiempo especificado, para que no contaminen tanto al ralentí. También puede calcular cuántos coches esperan y cuándo cada uno debe encender de nuevo el motor para avanzar en función de su posición en el atasco.

2.6 V2V

En Estados Unidos se ha publicado un sistema de comunicación entre automóviles. Se conoce como V2V [7] y permite enviar y recibir información con el objetivo de avisar al conductor en caso de peligro para evitar accidentes.

El proyecto pretende instaurar en los nuevos coches un sistema electrónico de intercomunicación que envía sonidos e imágenes para reportar información o advertencias al conductor.
Dotado de un GPS, un dispositivo Wi-Fi y un determinado programa informático, este sistema recopila datos de todos los elementos electrónicos que se encuentren en un radio de unos 300 metros. Funciona compartiendo información sobre velocidad y localización de otros vehículos para permitir a los pilotos advertir de peligros más allá de su campo de visión, incluyendo atascos o la presencia de un vehículo que se encuentra parado más adelante.

En la Fig.9 se muestra dos vehículos, uno de ellos con el sistema electrónico de intercomunicación que envía información al conductor, en este caso le informa que tiene un stop a unos pocos metros de su posición actual.

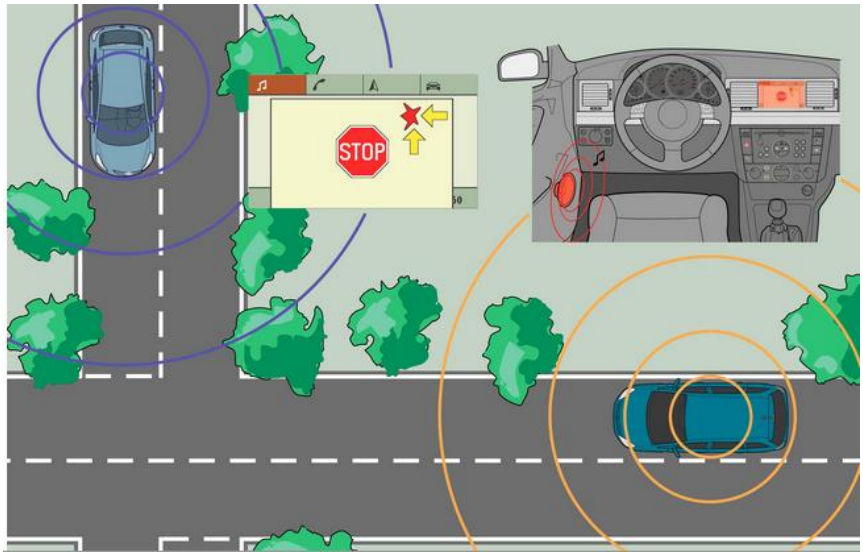


Fig. 9. Funciones V2V del Prototipo Honda

2.7 Ventajas de la Aplicación

La aplicación desarrollada en este proyecto permite hacer de las carreteras un lugar más seguro reduciendo el número de accidentes en la carretera. También destaca que el envío del evento por parte del usuario es de forma totalmente anónima y confidencial sin miedo a que se revelen sus datos, debido a la seguridad implementada. La facilidad de uso permite que el usuario pueda utilizarla sin preocupación por la complejidad de la misma.

3. Preliminares

3.1 VANET

Una VANET (Vehicular Ad-hoc Network) [8] es un tipo particular de red móvil con la particularidad de estar compuesta por conjuntos de vehículos que se comunican entre sí mediante equipos de transmisión radio (lo que es conocido como comunicación inter-vehicular, y, en determinadas ocasiones, también con elementos que forman parte de la infraestructura de las vías de circulación.

En la Fig.10 se muestra un ejemplo de VANET, donde se encuentran todos los vehículos interconectados unos con otros e intercambiando información entre ellos y con el entorno que les rodea.

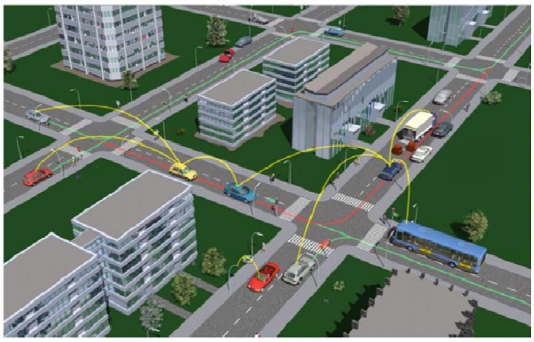


Fig. 10. Ejemplo de VANET

En los últimos años, las redes vehiculares o Vehicular Ad hoc NET Works (VANETs) han recibido la atención de la comunidad investigadora e industrial ya que su aplicación al ámbito de la seguridad vial puede ayudar a reducir el número de víctimas en accidentes de tráfico. Las VANETs también despiertan interés por otro tipo de aplicaciones como aquellas orientadas a la eficiencia del tráfico o las aplicaciones de entretenimiento y servicios de información.

Las principales aplicaciones viales de las VANETS tienen como finalidad disminuir los accidentes de tráfico y por tanto, la pérdida de vidas humanas. Un importante porcentaje de los accidentes que tienen lugar en todo el mundo está asociado a las intersecciones de vías y colisiones frontales y laterales de vehículos.

Las principales aplicaciones de las VANETs en seguridad vial pueden clasificarse en tres categorías:

- incidentes dinámicos (infracción grave detectada, circulación en sentido contrario, vehículos de emergencia, etc.)
- incidentes estáticos (límite de velocidad, obras en carretera, etc.)
- gestión de incidentes (accidente, atropello, etc.)

Algunos de los principales casos de uso existentes son los siguientes:

Advertencias durante adelantamientos: Se trata de reducir el riesgo de colisión lateral, debido a los denominados puntos ciegos, cuando se realiza un cambio de carril.

En la Fig.11 se muestra la intención del vehículo de cambiarse de carril.

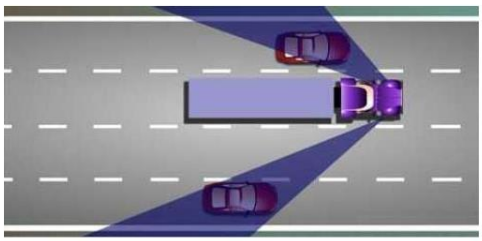


Fig. 11. Asistencia para el Cambio de Carril

Advertencia de posible colisión trasera. El conductor es informado de un riesgo de colisión trasera con el vehículo que se encuentra delante, posiblemente causado por una frenada brusca, falta de visibilidad en curvas, cambios de rasante, etc.

En la Fig.12 se muestra tres vehículos cercanos sin mantener la distancia de seguridad mínima y con la posibilidad de producir una colisión en cadena.

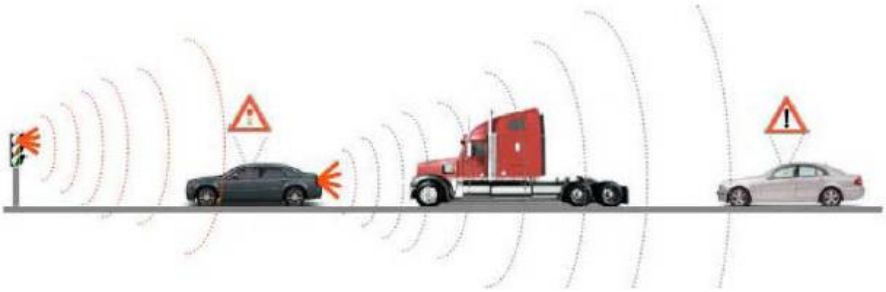


Fig. 12. Advertencia de Posible Colisión Trasera

Notificación de punto peligroso. Cualquier vehículo o RSU (Vehículos de recogida y transporte de residuos sólidos urbanos) informa a otros vehículos sobre la existencia de puntos peligrosos, tales como obstáculos en la carretera, obras o calzadas deslizantes.

En la Fig.13 se muestra la advertencia de un elemento o punto peligroso en la carretera.

Fig. 13. Aviso de Elemento o Punto Peligroso

Advertencias en intersecciones. En este caso, se detecta el riesgo de choque lateral para vehículos que se aproximan a una intersección. Los vehículos ya presentes en la intersección y/o los RSUs, detectan este peligro e informan a quienes se están aproximando.

Aviso de las condiciones del tráfico Cualquier vehículo que detecte una evolución/cambio rápido de la fluidez del tráfico informa sobre su situación a los demás vehículos y a los RSUs.

El siguiente caso de uso se corresponde con la aplicación desarrollada.

Violación de señal de tráfico Uno o más RSUs detectan violaciones de señales de tráfico. Esta circunstancia es retransmitida por los RSUs a todos los vehículos de los alrededores, de cara a advertirlos de un posible escenario peligroso.

En la Fig.14 se muestra una intersección que simula una VANET donde los vehículos son nodos de la red y establecen comunicaciones con el entorno de comunicación vehicular.

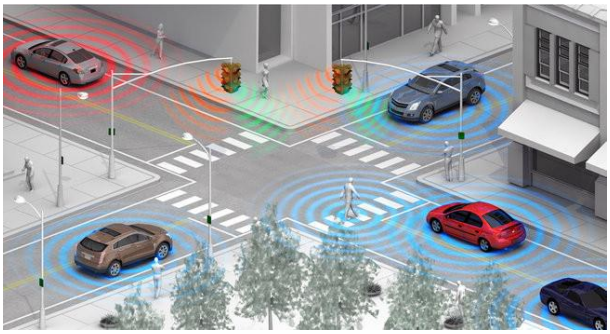


Fig. 14. VANET. Intersección

3.2 ITS

Se conoce como Sistemas Inteligentes de Transporte, o ITS [9] (Intelligent Transportation Systems), al conjunto de aplicaciones informáticas y sistemas tecnológicos creados con el objetivo de mejorar la seguridad y eficiencia en el transporte terrestre (carreteras y ferrocarriles), facilitando la labor de control, gestión y seguimiento por parte de los responsables. Estos sistemas obtienen la información de los diferentes elementos de interés de las carreteras, que una vez procesada y analizada, se utiliza para mejorar la seguridad de los conductores, mejorando el tráfico y la comodidad en los desplazamientos. La aplicación desarrollada consiste en un ITS.

En la Fig.15 se muestra un ejemplo de un sistema inteligente de transporte que se encuentra monitorizando todo el tráfico y evitando así una posible congestión.

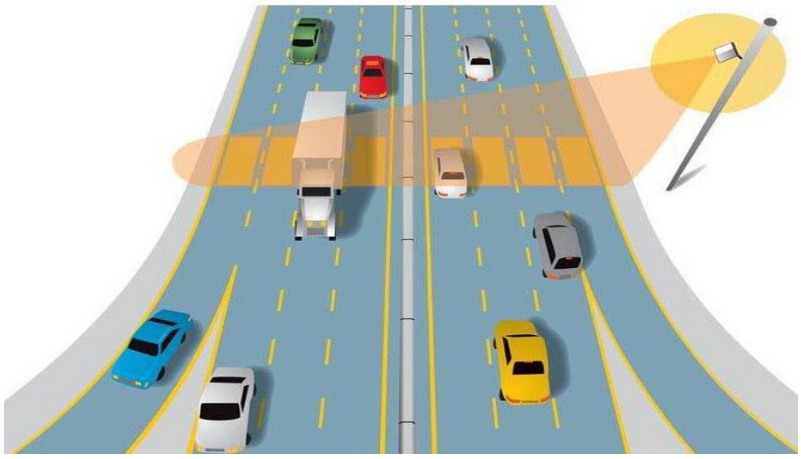


Fig. 15. Ejemplo de ITS

3.3 Bluetooth Low Energy

Tecnología que permite enviar información entre dispositivos electrónicos que tiene un consumo de energía considerablemente reducido mientras que mantiene un alcance de comunicación similar. La aplicación desarrollada está preparada para terminar de intercomunicar vía Bluetooth Low Energy [10] con el sensor integrado en los semáforos.

En la Fig.16 se muestra tecnología Bluetooth Low Energy aplicada para conectar sensores con smartphone o tabletas.

Fig. 16. Tecnología Bluetooth Low Energy

3.4 Android Studio

La aplicación móvil está desarrollada completamente en Android nativo haciendo uso de este IDE. Es una interfaz de desarrollo para la plataforma Android. Fue anunciado por Ellie Powers el 16 de mayo de 2013. Basado en IntelliJ IDEA de JetBrains, está diseñado específicamente para desarrollar para Android. En esta plataforma se encuentra nuestro proyecto, las carpetas del mismo, los archivos que hay en él, y todo lo necesario para acabar creando la aplicación.

En la Fig.17 se muestra el logo del entorno de trabajo utilizado para la realización de la aplicación.

Fig. 17. Logo Android Studio

3.5 NodeJS

Node.js es un entorno de programación del lado del servidor basado en el lenguaje de programación ECMAScript. Es el servidor de la aplicación. Será el encargado de procesar la información que recibe por parte de la aplicación móvil y notificar a los vehículos cercanos de que hay un vehículo saltándose un semáforo en los alrededores.

En la Fig.18 se muestra el logo del lenguaje utilizado para el desarrollo del servidor.

Fig. 18. Logo NodeJS

3.6 Express

Framework web mínimo y flexible para NodeJS que permite comunicarse con la aplicación gracias a las routes, ya que al crear éstas en la aplicación el servidor las interpreta y las entiende. Gestiona las rutas de la aplicación asociando cada ruta con una función encargada de controlar la acción a ejecutar.

En la Fig.19 se muestra el logo del framework utilizado con NodeJS para permitir la comunicación entre la aplicación y el servidor mediante las rutas que implementa dicho framework.

Fig. 19. Express

3.7 MongoDB

MongoDB es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto. La base de datos de la aplicación estará gestionada por MongoDB del lado del servidor donde guardaremos toda la información de la aplicación (señales, eventos y tipos de vehículos).

En la Fig.20 se muestra el logo del sistema de base de datos utilizado en el lado del servidor.

Fig. 20. Logo MongoDB

3.8 Google Cloud Messaging

El Servicio de mensajería en la nube de Google (GCM) es un servicio gratuito que ayuda a los desarrolladores a enviar datos de los servidores a las aplicaciones Android. Este servicio se encarga de enviar las notificaciones de que el usuario ha cometido la infracción al resto de vehículos cercanos.

En la Fig.21 se muestra el icono del servicio de Google Cloud Messaging encargado de enviar las notificaciones Push a los vehículos cercanos de la infracción cometida.

Fig. 21. Google Cloud Messaging

3.9 Criptografía de Clave Pública

Método criptográfico que usa un par de claves para el envío de mensajes. Las dos claves deben de pertenecer al emisor del mensaje.

Clave Pública: La conocen todos los usuarios y se puede entregar a cualquier persona.
Clave Privada: El propietario debe guardarla de modo que nadie tenga acceso a ella.

Características:

- Confidencialidad: el emisor usa la clave pública del receptor para encriptar un mensaje. Al llegar al receptor, decodifica el mensaje con su propia clave privada. De esta forma se mantiene la confidencialidad.
- Autenticidad: Si el receptor de un mensaje es capaz de descifrarlo con la clave pública del emisor, significa que el mensaje realmente fue enviado por el emisor, probando así su autenticidad. Esto es debido a que un mensaje enviado con una clave privada sólo puede descifrarse con su clave pública.

En la Fig.22 se muestra el funcionamiento del sistema de criptografía de clave pública [11].

Fig. 22. Criptografía de Clave Pública (Asimétrica)

4. Sensores Inteligentes en Semáforos

4.1 Definición

Un sensor inteligente [12] es un dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas. Estos sensores permiten conocer, en tiempo real, el estado de la circulación, de la vía o las tendencias de tráfico.
Los semáforos actuales están equipados con sensores inteligentes para la detección y cuenta de los vehículos que circulan.

Los sensores en los semáforos inteligentes se utilizan entre otras cosas para:

- Mejorar el rendimiento de los semáforos y ajustar el tiempo entre los diferentes colores de luz.
- Detectar peatones o ciclistas que cruzan por lo que los semáforos pueden cambiar en consecuencia.
- Lidiar con los atascos de tráfico. Si se produce un atasco de tráfico, el sistema puede activar los semáforos para cambiar antes de lo habitual para ayudar a dispersarlo.
- Dar prioridad al tráfico especial, como ambulancias o vehículos de la policía. Puede detectar un vehículo de emergencia que se aproxima con las sirenas en funcionamiento y puede cambiar las luces en consecuencia.
- Detección de vehículos preferentes se basan generalmente en transmisores inalámbricos, infrarrojos u ópticos que envían una petición al controlador de luces para cambiar cuando se detecta que un vehículo de emergencia se aproxima.
- Capaces de leer un determinado tipo de sirena y detectar ambulancias acercándose.
- Detección de vehículos y bicicletas con un borde de metal que se acercan a un paso de peatones o a un cruce. A continuación, cambian las luces en consecuencia.

En la Fig.23 se muestra una intersección donde podemos observar los vehículos emitiendo información gracias a los sensores que se encuentran en el entorno vehicular.

Fig. 23. Intersección. Comunicación Sensores

4.2 Características

Dentro de cada sensor inteligente radica uno o más sensores primitivos y la circuitería de soporte. Lo que hace a un sensor inteligente "inteligente" es la electrónica construida internamente adicional. Esta electrónica hace que estos sensores sean capaces de hacer una o más de las siguientes funciones:

- Pre-procesar los valores medidos en cantidades que posean algún significado.
- Comunicar las medidas con señales digitales y protocolos de comunicación.
- Orquestar las acciones de los sensores primitivos y sus circuitos para "tomar" mediciones.
- Tomar decisiones e iniciar alguna acción en base a las condiciones, de manera independiente al microcontrolador.
- Recordar la calibración o la configuración de sus parámetros.

En la Fig.24 se muestra un esquema de la estructura de un sensor inteligente.

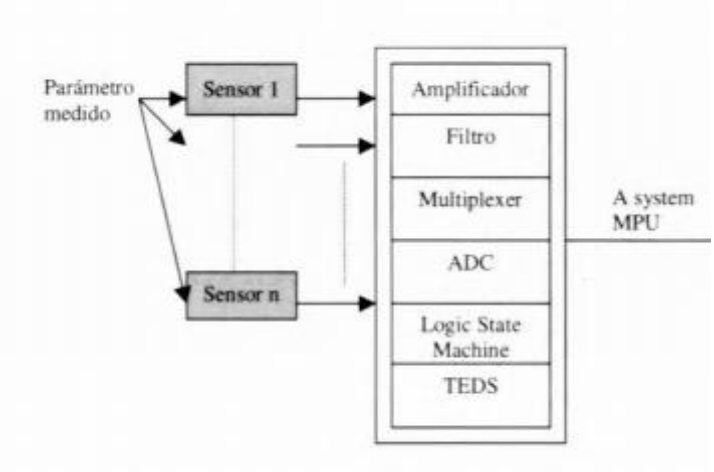


Fig. 24. Esquema Sensor Inteligente

4.3 Tipos

Los sensores integrados en los semáforos se construyen generalmente con sensores inductivos y funcionan mediante la detección de cambios en el campo electromagnético. Cuando un vehículo entra en el campo de los sensores magnéticos, hace que la luz cambie.

Estos sensores inductivos [13] son dispositivos electrónicos que utiliza una tecnología de sensor de proximidad para detectar objetos metálicos sin contacto físico. Son de gran utilización en la industria, tanto para aplicaciones de posicionamiento como para detectar la presencia de objetos metálicos en un determinado contexto (control de presencia o de ausencia, detección de paso, de atasco, de posicionamiento, de codificación y de conteo).

5. Semáforos Inteligentes

5.1 Definición

El desarrollo de modelos de semáforos inteligentes [14] ha sido una de las grandes propuestas que se han planteado para aumentar el flujo de vehículos por las calles de grandes ciudades, evitando así el congestionamiento vehicular.

Un semáforo inteligente es aquel que detecta la cantidad de flujo vehicular mediante sensores y con base a parámetros ya establecidos, van modificando los tiempos de paso y/o de detención. Es un sistema tecnológico que permite optimizar el funcionamiento de la red semafórica de la ciudad, mediante herramientas que entregan información sobre la cantidad de vehículos circulando, velocidades promedio y ocupación de la vía, entre otros indicadores.

5.2 Funcionamiento

El controlador de tráfico es autónomo e inteligente, toma las decisiones por si solo de acuerdo a las señales que les manden los sensores colocados estratégicamente para una mejor recepción de información, el sensor le mandara los datos a la computadora esta las procesara de manera rápida para tomar las decisiones de la manera más rápida y correcta posible.
Las computadoras o controladores que tiene el semáforo están perfectamente coordinadas con los sensores para que la información captada sea rápidamente procesada dando al conductor un rápido traslado de un lugar a otro. En la Fig. 25 se muestra el funcionamiento de un semáforo inteligente.

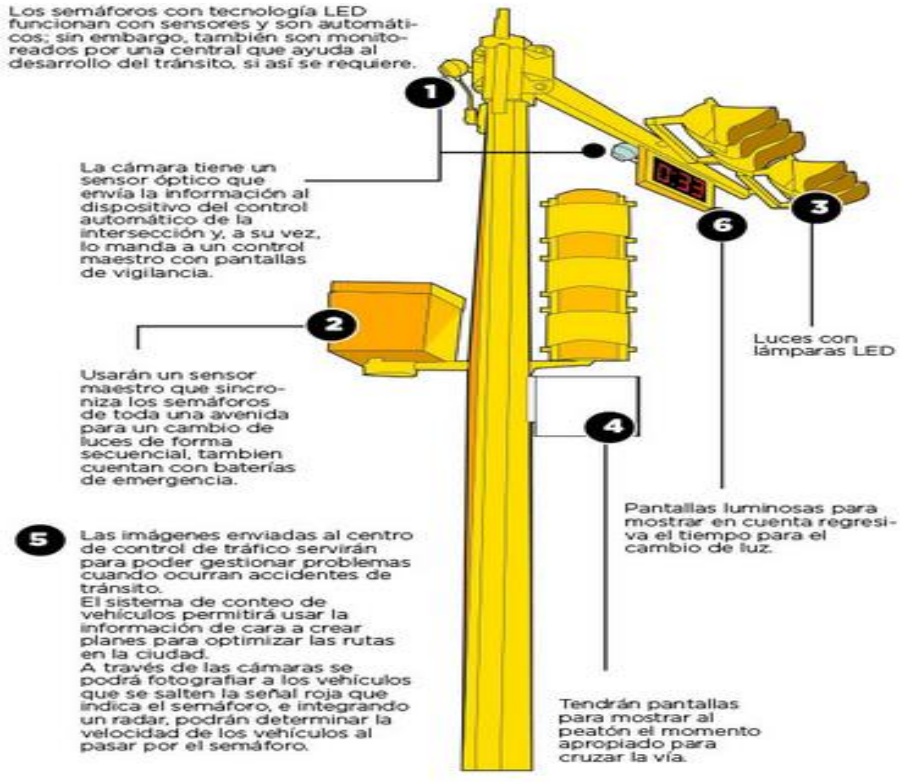


Fig. 25. Funcionamiento Semáforo Inteligente

5.3 Características

- Facilita el manejo directamente a los municipios. No dependen de terceros
- Consumen un 95% menos de energía.
- Permite actualización de software remotamente sin cambio de equipos
- Disminuyen los costos de mantenimientos entre un 30% y 50%
- Cuentan con una vida útil muy amplia
- Pueden funcionar sin energía en caso de cortes del fluido eléctrico sin afectar movilidad.
- El mantenimiento puede realizarse por personas que no sean especializadas y residan en la ciudad.
- El soporte y la casa matriz están directamente en la región.
- Se puede monitorizar su operación a través de internet o teléfonos móviles.
- Administrar la red semafórica según el tráfico vehicular.
- Maximizar los desplazamientos por la ciudad, reduciendo los tiempos de recorrido.
- Ordenar el tránsito.
- Minimizar las paradas de los vehículos.
- Reducir el tiempo de espera en los semáforos.
- Economizar combustible y reducir los mantenimientos al vehículo y el desgaste de sus piezas.
- Dar seguridad a los movimientos de los usuarios de la vía y proteger al peatón

5.4 Comunicación Semáforo – Aplicación

En la Fig.26 se muestra la comunicación entre el semáforo y el vehículo.

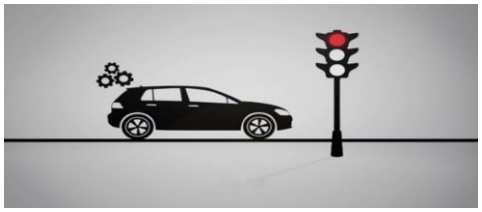


Fig. 26. Comunicación Semáforo – Aplicación

El semáforo se comunica con el vehículo gracias a los sensores Bluetooth Low Energy. El sensor en el semáforo está siendo realizado por otra universidad de Málaga y está todo preparado para en el futuro recibir en la aplicación por Bluetooth Low Energy esta información a través del sensor que irá colocado en los semáforos. Para el Trabajo de Fin de Grado se está realizando la acción del sensor de forma simulada

En la Fig.27 se muestra la conectividad entre el semáforo y vehículo (sensores BLE).

Fig. 27. Conectividad Sensores Bluetooth Low Energy

Suponemos que es un mock (objeto simulado), por tanto lo que se hace, es simular la recepción de estos datos, cuando el usuario pulsa el botón del menú izquierdo que indica el evento del Salto de Semáforo.
En primer lugar, una vez que el usuario selecciona la opción de generación del evento del salto de semáforo en el menú, automáticamente empieza la comunicación entre el vehículo y el semáforo que se ha saltado.
El semáforo se comunica con la aplicación (vehículo) enviándole una serie de parámetros que son los siguientes:

- Fecha Actual (Timestamp): Fecha en la que el vehículo ha cometido la infracción
- Estado del semáforo: Color del mismo.
- Posición del semáforo (GEO): La posición exacta del semáforo que el vehículo se ha saltado (Latitud y Longitud).
- ID del semáforo: Identificador que identifica al semáforo.

La posición GPS del semáforo, en principio se le asigna la misma posición que la del vehículo. Pero en un caso real, la posición GPS del semáforo la va a enviar el mismo gracias a los sensores. Como podemos observar a continuación hemos asignado la posición GPS del semáforo la misma que la del vehículo, donde Mapa.actual.getLatitude y Mapa.actual.getLongitude es la posición del vehículo.

```
location.setLatitude(Mapa.actual.getLatitude());
location.setLongitude(Mapa.actual.getLongitude());
```

Como hemos dicho al ser todo simulado, asumimos que la velocidad del vehículo no está en reposo cuando el semáforo le envía el estado del mismo (color) que estará en rojo a la aplicación cuando el vehículo se salte el semáforo. La velocidad y el color les hemos asignado un valor fijo, pero se podría programar en un futuro para establecer valores aleatorios tanto al color del semáforo como a la velocidad el vehículo.

```
String color = "TRAFFIC_LIGHT_RED";
location.setSpeed(10);
```

El semáforo también se encarga de generar y enviar un hash para la integridad de datos, es decir cuando se envíe al servidor, garantizar que han llegado intactos y no han sido manipulados por un tercero verificando en el servidor si el hash generado es el mismo que recibe por parte de la aplicación. Este hash está compuesto por:

Hash(TIMESTAMP|GPSSemaforo): Posición GPS Semáforo Concatenado con el TimeStamp (Fecha Actual de la infracción). Comprueba el hash recibido en el servidor y realiza el hash de los datos en claro. Si ambos hash coinciden, quiere decir que la información no se ha visto alterada durante la comunicación por un tercero

```
DateFormat df = new SimpleDateFormat("HH:mm:ss dd/MM/yyyy", Locale.getDefault());
String fecha_str = df.format(fecha);
String hash = fecha_str+"|"+latitud+"|"+longitud;
return hash;
```

Una vez que tenemos los datos que hemos recibido por parte del semáforo, en donde hemos visto que el vehículo se ha saltado el semáforo ya que estaba en rojo y su velocidad no estaba en reposo se genera el envío creando un webservice en la aplicación de tipo POST del evento de salto de semáforo al servidor:

6. Servidor

6.1 Funcionamiento

El servidor ha contado con la ayuda de Dephisit, proyecto nacional de investigación que ha cooperado para la realización de este proyecto en el que participa el grupo CryptULL. Al proyecto Dephisit se le ha integrado (desarrollado) la funcionalidad para la detección de un salto de semáforo y posterior notificación a los vehículos cercanos.

El servidor se va a encargar de realizar diferentes funciones:

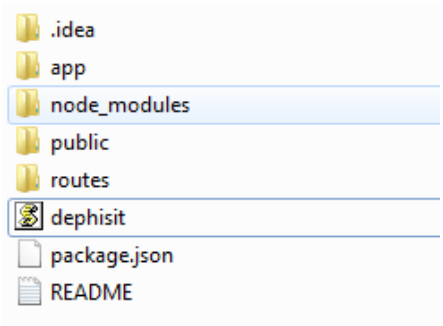
- Comunicarse con los diferentes vehículos y comunicarle que hay un conductor que se ha saltado un semáforo, gracias al servidor de gcm (Google Cloud Messaging), que es el servicio de Google que envía los mensajes a los destinos finales (vehículos).
- Buscar nuevos vehículos a los que comunicar la infracción que el usuario ha cometido.
- Gestionar la base de datos del lado del servidor MongoDB, donde se almacenan, todas las señales, eventos, tipos de vehículos y posición GPS del vehículo.
- Calcular la máxima distancia a la que se puede notificar a los vehículos de la infracción del salto de semáforo, que es aproximadamente unos 500 metros.
- Comunicarse con el servicio de Google Cloud Messaging que será el encargado de generar notificaciones push y enviarlas a los vehículos cercanos de la infracción cometida.
- Comprobar que la información que se envía de la aplicación al servidor no se ha visto alterada (manipulada), para ello se utiliza un hash como fingerprint.
- Enviar a la aplicación el id de registro del vehículo para que la aplicación pueda empezar a notificarle cosas (generar el evento del salto de semáforo).

En la Fig.28 se muestra un ejemplo de un servidor.

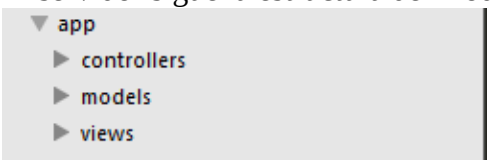
Fig. 28. Servidor

6.2 Estructura Servidor

El servidor está integrado completamente en NodeJS. No se contempla la interfaz (front-end) para este proyecto.



El servidor sigue la estructura del modelo-vista-controlador (MVC)



- La parte de **models** es donde se encuentra la información (datos) de la aplicación, por medio de la cual realiza consultas a través de los controladores en el lado del servidor.
- La parte de **views**, está disponible para la aplicación dephisit pero no se contempla para este proyecto.
- La parte de **controllers** es donde se centra todo el servidor. Responde a la información solicitada gracias a lo que le proporciona el modelo.

6.3 Web Services

```
var express = require('express'),
var app = express();
```

Como podemos observar utilizamos el framework express para permitir la comunicación entre la app y el servidor, a través de rutas que la aplicación envía al servidor y este es capaz de entender e interpretar. Estas rutas se encuentran en el servidor, concretamente en el fichero routes.js

```
routes = require('./routes/router')(app);
```

En este fichero se encuentran todas las rutas que el servidor entiende y se crean en la app. Express permite gestionar las rutas de la aplicación asociando cada ruta con una función encargada de controlar la acción a ejecutar.
Aquí tenemos algunas de las rutas que permiten la comunicación entre la aplicación y el servidor.

```
app.get('/register', utilities.login);

/* Other Routes */
require('../app/controllers/webservices/eventosWS')(app);
require('../app/controllers/webservices/gcmWS')(app);
```

Las dos últimas rutas son las más importantes que se encargan de gestionar los webservices de los eventos y el servicio gcm (google cloud messaging).

6.4 Base de Datos

El servidor es el que se encarga de conectarse y gestionar la base de datos usando **MongoDB**.

```
mongoose.connect('mongodb://localhost/dephisit_db');
```

El nombre de la base de datos como observamos se denomina **dephisit_db**
Tenemos que tener arrancado mongodb, para ello realizamos lo siguiente:
Nos dirigimos a la carpeta donde se encuentra instalado MongoDB y nos situamos dentro de la carpeta /bin. Una vez ahí ejecutamos el siguiente comando:

```
C:\Program Files\MongoDB\Server\3.0\bin>mongod
```

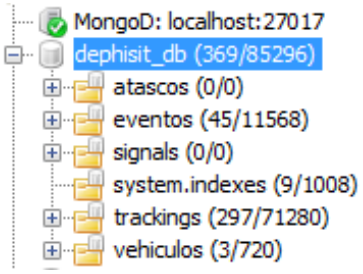
Una vez ejectutado el comando se arrancara el servicio de Mongodb.
A continuación arranacamos el servidor, que se escuchara en el puerto 3000, pero como ya hemos dicho al no tener interfaz web no podremos visualizar nada.

```
http.createServer(app).listen(3000);
```

Para arrancar el servidor ejecutamos el siguiente comando, una vez estemos situado en la carpeta del servidor:

```
C:\Users\Rushil\Desktop\Android\TFG\cryptull-dephisitserver-476e7f5423fa\cryptul
l-dephisitserver-476e7f5423fa>node dephisit.js
Lanzado en puerto 3000...
```

A continuación arrancamos la interfaz de mongodb para ver el schema y los atributos que contiene dicha base de datos:



Como podemos observar en la imagen tenemos los diferentes atributos que compone la base de datos (dephisit_db).

- Eventos
- Señales
- Vehículos
- Trackings

En el lado del servidor se encuentra el esquema de la base de datos (schema), donde encontramos toda la estructura de la base de datos, que describiremos con detalle más adelante.

```
var mongoose = require('mongoose'),
    Schema = mongoose.Schema;

var eventoSchema = new Schema({
    idpromotor: String,
    idtipoevento: String,
    iddestino: String,
    idtipovehiculo: String,
    detalles: String,
    fecha: Date,
    geo: {type: [Number], index: '2d'},
    sentido: { type: Number, default: 0 },
    activo: { type: Boolean, default: true }
});

//Export the schema
module.exports = mongoose.model('Evento', eventoSchema);
```

Estructura Evento:

- Id Promotor: El emisor,(infractor) que genera el evento
- Id TipoEvento: ID que referencia al tipo de evento a generar (salto de semáforo)
- IdDestino: ID que referencia a quien va dirigido
- IdTipoVehículo: ID que referencia al tipo de vehículo bien puede ser un vehículo pesado, de emergencia …Por defecto será un vehículo normal.
- Detalles: Información adicional del evento.
- Fecha: Fecha en que se ha generado el evento.
- Geo: Posición GPS de la zona en donde se ha generado el evento.
- Sentido: 0 a 180 º
- Activo: Indica si el evento está activo o no. Para conocer si ya ha caducado.

```
var mongoose = require('mongoose'),
    Schema = mongoose.Schema;

var trackingSchema = new Schema({
    idvehiculo: String,
    fecha: {type:  Date, default : Date.now() },
    geo: {type: [Number], index: '2d'},
    velocidad: { type: Number, default: 0 },
    sentido: { type: Number, default: 0 },
    emergencia: { type: Boolean, default: false }
});

//Export the schema
module.exports = mongoose.model('Tracking', trackingSchema);
```

Estructura Tracking:

- Id Vehículo: ID que referencia al tipo de vehículo
- Fecha: Fecha en la que se ha cometido la infracción
- Geo: Posición GPS del vehículo
- Velocidad: Velocidad del vehículo
- Sentido: 0-360º
- Emergencia: Si se trata de un vehículo de emergencia o no. En el proyecto actual no aplica, es para otra funcionalidad dentro de Dephisit, proyecto de investigación como hemos dicho anteriormente donde se engloba este TFG.

```
var mongoose = require('mongoose'),
    Schema = mongoose.Schema;

var vehiculoSchema = new Schema({
    idgcm: String, // id del móvil que te da google (MAC)
    idtipovehiculo: String,
    revocacion: Boolean,
    so: { type: String, default: 'ANDROID' }
});

//Export the schema
module.exports = mongoose.model('Vehiculo', vehiculoSchema);
```

Estructura Vehículo:

- IdGcm: ID del servicio google cloud messaging, único para cada smartphone
- IdTipoVehiculo: ID que referencia al tipo de vehículo que se ha saltado el semáforo que por defecto será un vehículo normal.
- Revocación: Este atributo indica si el vehículo esta revocado o no. Si se ha comportado mal y la autoridad certificadora lo ha revocado.
- SO: Sistema operativo utilizado que sería Android

6.5 Comunicación Servidor – Aplicación

En la Fig.29 se muestra un esquema sobre la comunicación servidor –aplicación.

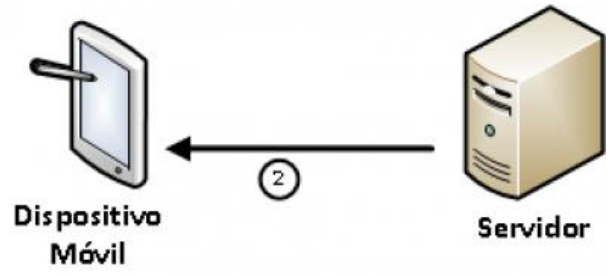


Fig. 29. Comunicación Servidor – Aplicación

La aplicación se comunica con el servidor generando un envío con/sin datos (POST o GET) gracias al framework express generando las rutas que el servidor entiende. En primer lugar una vez que el vehículo se ha registrado en el servidor, el servidor le devolverá a la aplicación un ID de registro único del vehículo para confirmar que ya ha sido registrado en el servidor, por tanto una vez que la aplicación lo recibe puede empezar a notificar cosas al servidor.

Una vez que el vehículo ha sido registrado en el servidor, se le asigna otro ID que le proporciona el servidor de Google Cloud Messaging (gcm) para poder enviar y recibir notificaciones a otros dispositivos. La aplicación envía el idgcm cada cierto tiempo por si ha cambiado y actualizar ese id en el servidor. Primero se busca en la base de datos de mongo si el vehículo está registrado en el servidor. Si efectivamente está registrado comprueba si el idgcm que se ha enviado desde la app es distinto, si es así asigna el nuevo idgcm al vehículo que puede ser que el servidor de Google Cloud Messaging lo haya cambiado. Por último envía un mensaje afirmativo para saber que le ha llegado correctamente el idgcm. Este ID se asigna desde la aplicación y se envía al servidor. Cada cierto tiempo se está verificando ese ID desde la aplicación ya que como hemos dicho puede cambiar y hay que actualizarlo en el servidor, en la base de datos de MongoDB.

```
app.post('/gcm/register/:idvehiculo',    register);
```

```
if ((vehiculo !== null) && (vehiculo !== undefined) && (vehiculo !== ''))
{
    console.log("Entra en el register del idgcm");
  if (req.body.idgcm !== undefined){
    vehiculo.idgcm=req.body.idgcm;
    //console.log(req.body.idgcm);
    vehiculo.save();
  }
  res.send("ok");
} else {
  res.send("ok");
```

A continuación se busca en la base de datos si el vehículo que se ha enviado desde la aplicación, ya se encuentra registrado con el idgcm, para saber si el idgcm ha cambiado que puede ser porque como se ha dicho anteriormente, Google a veces lo cambia en la aplicación, y hay que notificar al servidor de que el idgcm ha cambiado para asignarle el nuevo id. Si lo encuentra en la base de datos con el mismo idgcm enviamos de vuelta a la aplicación el id de registro del vehículo en el servidor (vehículo_id) por lo que la aplicación ya puede empezar a notificar cosas al servidor.

```
app.post('/nuevoVehiculo', nuevoVehiculo);
 Vehiculo.findOne({idgcm: req.body.idgcm}, function (err, vehiculo) {
     if ((vehiculo !== null) && (vehiculo !== undefined) && (vehiculo !== '')) {
         res.send(vehiculo._id);
```

En caso contrario, que el vehículo no se encuentre registrado en la base de datos de mongo, lo registramos con los datos que nos envía la app que son los siguientes:

- ID Tipo Vehículo
- IDGCM
- Velocidad
- Revocación
- S.O

Lo guardamos en la base de datos de mongo y enviamos a la aplicación el id de registro único del vehículo en el servidor, como podemos observar:

```
else {
  var vehiculoNuevo = new Vehiculo({
      idtipovehiculo: req.body.idtipovehiculo,
      idgcm: req.body.idgcm,
      velocidad: req.body.velocidad,
      revocacion: false,
      so: req.body.so
  });
  vehiculoNuevo.save();
  res.send(vehiculoNuevo._id);
```

A continuación enviamos la posición GPS del vehículo al servidor

```
app.post('/nuevoTracking/:idvehiculo', nuevaPosicion);
```

Registra la posición GPS del vehículo (tracking) en la base de datos de mongo. Los datos que envía la aplicación son los siguientes:

- Id Vehículo: Id único del vehículo que indica que se encuentra registrado en el servidor
- Fecha: Fecha actual
- Posición GPS del vehículo (GEO)

- Sentido: Sentido del vehículo (0 a 180º)

Se almacena la posición del vehículo en el servidor en la base de datos de mongo concretamente en la tabla tracking y le responde al servidor con un mensaje afirmativo para que sepa que todo ha ido bien y que ha recibido la posición del vehículo.

```
var trackingNuevo = new Tracking({
    idvehiculo: req.params.idvehiculo,
    fecha: Utilities.parseDate(req.body.fecha),
    geo: [parseFloat(req.body.longitud), parseFloat(req.body.latitud)],
    sentido: req.body.sentido
});
trackingNuevo.save();
res.send("ok");
```

Por último tras haber registrado el vehículo y su posición GPS en la base de datos se genera el webservice de tipo POST del evento de salto de semáforo en la aplicación.

```
app.post('/nuevoEventoSemaforo/:idvehiculo/:idtipoevento', nuevoEventoSemaforo);
```

Se recibe el evento del salto de semáforo que hemos enviado desde la aplicación. Mostramos al usuario la generación del nuevo evento para que el usuario sepa que la comunicación entre el servidor y la aplicación ha podido establecerse de forma exitosa y se ha generado el evento de salto de semáforo de forma correcta.

```
 console.log("Nuevo evento SALTA SEMAFORO :" + req.body.longitudS + ", " + req.body.latitudS);
```

A continuación se hace la comprobación si ya hay un evento generado, es decir si el evento ya se encuentra registrado en la base de datos que gestiona mongodb, si la respuesta es afirmativa cierra la conexión con la aplicación enviándole un ok. En el caso de que no se encuentre en la base de datos, crea el evento y lo registra en la base de datos y le informa al resto de vehículos.

```
Evento.findOne({$and: [{geo: [parseFloat(req.body.longitud), parseFloat(req.body.latitud)]}, {idtipoevento: req.params.idtipoeve
    function (err, nevento) {
        if ((nevento !== null) && (nevento !== undefined) && (nevento !== '')) {
            res.send("ok");
        } else {
            crearEventoSeInformar(req, res, 500);
        }
    }
);
```

Sí no se encuentra registrado el evento lo creamos, y lo guardamos en la base de datos de mongodb: Le pasamos los siguientes datos del evento al servidor para almacenarlo en mongo:

- ID Tipo Evento: Tipo Evento que será el del salto de semáforo
- ID Promotor: Nulo, ya que quien envía el evento es anónimo
- ID Destino: Nulo, ya que no se envía a uno sólo sino a todos los cercanos
- ID Tipo Vehículo: Vehículo Normal considerado por defecto
- Detalles: Nulo, ya que está gestionado por el servidor gcm
- Fecha: Fecha de generación de la infracción
- Latitud Vehículo: Latitud del infractor
- Longitud Vehículo: Longitud del infractor

Por último informamos el evento a los vehículos cercanos que encontremos. Para ello hacemos una consulta a la base de datos para ver si tenemos registrada la posición GPS (tracking) del vehículo en la base de datos que hemos enviado desde la aplicación al servidor que cumpla la condición de que se encuentren a 500 m como mucho.

```
Tracking.find({
        geo: {
            $near: [parseFloat(req.body.longitud), parseFloat(req.body.latitud)],
            $maxDistance: distance / 111000.12
        }, fecha: {$gte: now}
```

Si encontramos la posición de algún vehículo registrado en mongodb que cumpla las condiciones de estar como mucho a 500 metros guardamos su id para posteriormente informarle de que alguien a su alrededor se ha saltado un semáforo indicando la posición GPS del infractor.

```
if ((trackings !== null) && (trackings !== undefined)) {
    var idvehiculos = trackings.map(function (x) {
        return x.idvehiculo
    });
```

A continuación buscamos los vehículos en la base de datos que coincidan con el id del vehículo que previamente habíamos guardado.

```
Vehiculo.find({_id: {$in: idvehiculos}}, function (err, vehiculos) {
```

Si encuentra los vehículos a los que notificar en la base de datos entonces procedemos a activar el servicio gcm que es el encargado de avisar a dichos vehículos la infracción cometida. Para ello realizamos un push, es decir enviamos la notificación del servicio gcm .Enviamos el mensaje con los parámetros siguientes:

- IDs de los vehículos registrados en el servicio gcm
- El evento que se ha generado que será el del salto de semáforo
- Y un mensaje (“ok) a la aplicación para que sepa que todo ha ido bien y se han encontrado vehículos a los que informar la infracción del salto de semáforo.

```
if ((vehiculos !== null) && (vehiculos !== undefined) && (vehiculos !== [])) {
    var registrationIds = [];
    vehiculos.forEach(function (v) {
        registrationIds.push(v.idgcm);
    });
    Utilities.sendMessage(registrationIds, eventoNuevo, res);
    res.send("ok");
```

El mensaje que el servidor de gcm va a enviar a los vehículos cercanos contiene la siguiente información:

```
msg.idpromotor = evento.idpromotor;
msg.idtipoevento = evento.idtipoevento;
msg.detalles = evento.detalles;
msg.idtipovehiculo = evento.idtipovehiculo;
msg.fecha = evento.fecha;
msg.latitud = evento.geo[1];
msg.longitud = evento.geo[0];
msg.sentido = evento.sentido;
```

- Id Promotor: El usuario que ha generado el evento, es decir el que se saltó el semáforo
- Tipo Evento: Evento de salto de semáforo
- Detalles: Información adicional sobre el evento, que será nulo

- Tipo Vehículo: El tipo de vehículo que se saltó el semáforo. Vehículo normal por defecto
- Fecha: Fecha en que se produjo la infracción
- Posición (Latitud, Longitud): Posición del infractor que se ha saltado el semáforo.

Si por el contario no encontramos ningún vehículo le enviamos al usuario un mensaje en el lado del servidor que no se ha podido encontrar ningún vehículo y a la aplicación un mensaje de error (res.send("err")) indicándole que no ha habido éxito con la búsqueda de vehículos .

```
else {
  console.log("Fallo al encontrar VEHICULOS");
  res.send("err");
```

7. Aplicación

7.1 Funcionamiento

La aplicación se llama Dephisit como ya hemos mencionado anteriormente, se trata de un proyecto nacional de investigación orientado hacia aplicaciones vehiculares que ha colaborado para la realización de este proyecto.

La aplicación se va a encargar de realizar las siguientes funciones:

- Comunicación con el servidor, encargada de generar el evento del salto de semáforo y enviárselo a la aplicación
- Se encarga de generar el idgcm que le asigna el servidor de Google y se lo envía al servidor. Está cada cierto tiempo comprobando si el idgcm ha cambiado ya que el servicio de Google Cloud Messaging lo puede cambiar
- Se encuentra la interfaz del usuario, donde el usuario puede acceder a la aplicación y generar el evento
- Calcula la posición GPS del usuario durante todo momento. Se la envía al servidor cada 10 segundos
- Recibe únicamente por parte del servidor el id de registro del vehículo en el servidor. Una vez que la aplicación lo recibe empieza a notificarle cosas al servidor
- Calcula la posición GPS del vehículo
- Se comunica con el semáforo mediante los sensores integrados en los semáforos BLE (Bluetooth –Low Energy)
- Se encarga de pintar en el mapa, la posición GPS del infractor
- Se comunica con el servidor IDGCM para solicitarle su id
- Envía al servidor el hash generado en la aplicación para la integridad de datos, es decir una vez que el servidor genere el hash con los datos en claro recibidos por parte de la aplicación verifica que es el mismo hash, es decir que los datos no han sido manipulados por un tercero

En la Fig.30 se muestra el logo de Dephisit, proyecto nacional de colaboración con este TFG.

Fig.30. Proyecto DEPHISIT

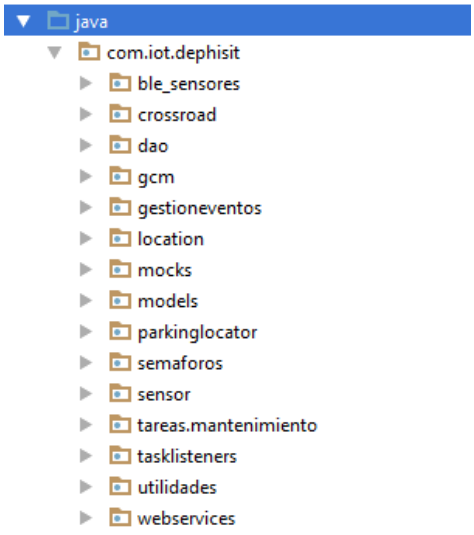
7.2 Estructura de la Aplicación

La aplicación se ha desarrollado en Android nativo (java) para la lógica de la aplicación (back-end) y xml para la vista de la aplicación (front-end).
La aplicación en Android está dividida en actividades:
Por un lado se encuentra la parte lógica y por otro lado la parte gráfica.

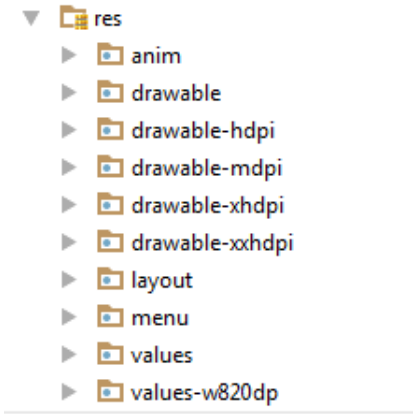
- La parte lógica (java) permite manipular, interactuar y colocar el código de la actividad (back-end).
- La parte gráfica (XML) contiene todos los elementos que se ven en la pantalla (móvil). Es el diseño de la aplicación (front-end).

Carpeta /app/src/main/java

Esta carpeta contendrá todo el código fuente de la aplicación, clases auxiliares,… La estructura en jerarquía de la aplicación es la que se muestra a continuación:



Carpeta /app/src/main/res/
Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, layouts, cadenas de texto, etc. Es la parte que el usuario puede ver, donde se encuentra la interfaz de la aplicación y el diseño de la misma.



7.3 Comunicación Aplicación - Servidor

En la Fig.31 se muestra el esquema de la comunicación aplicación – servidor.

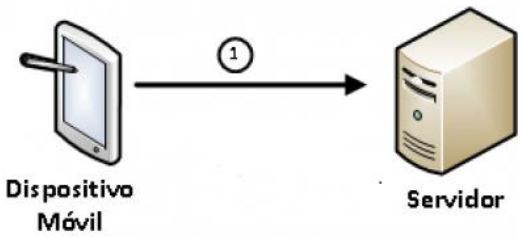


Fig. 31. Comunicación Aplicación – Servidor

La aplicación está desarrollada en Android Nativo, concretamente en Java (back-end). Una vez que el usuario inicia la aplicación, la aplicación se comunica con el servidor registrando el vehículo en el servidor, una vez que esté le devuelve el ID de registro en el servidor que es el mismo que el de mongo ya que el servidor gestiona la base de datos, la aplicación puede empezar a notificar cosas al servidor.

Se crea el webservice de tipo POST del nuevo vehículo que vamos a registrar en el servidor, donde establecemos la ruta que va a recibir el servidor en el fichero routes.js que va entender gracias al framework express.

```
Red.postWS(context, "nuevoVehiculo/", callback, data);
```

Suponemos que se trata de un tipo de vehículo normal aunque el proyecto está preparado para aceptar cualquier tipo de vehículo (emergencia, lento…)

```
if (crearVehiculo(ctx, "TV_NORMAL", null, null)){
    System.out.println("Entramos a crearvehiculo");
    CommonUtilities.counter = 0;
    WebServices.enviarVehiculo(ctx, new ResponseServer_PostNuevoVehiculo(ctx, 0), DataEvent.getDataNuevoVehiculo(BBDD
```

En el JSON que vamos a enviar al servidor del nuevo vehículo enviamos los siguientes datos:

- IdTipoVehiculo: Que será un vehículo normal como hemos establecido a la hora de crear el vehículo.
- Idgcm: Que va a ser el id del servicio de Google para recibir notificaciones de que se ha cometido la infracción.

El IDGCM se asigna al vehículo una vez se inicia la aplicación de la siguiente forma:

```
try {
    if (gcm == null) {
        gcm = GoogleCloudMessaging.getInstance(context);
    }
    regid = gcm.register(SENDER_ID);
    msg = "Device registered, registration ID =" + regid;
```

Aquí asignamos el IDGCM en la base de datos.

```
GCMRegistrar.register(ctx.getApplicationContext() , CommonUtilities.SENDER_ID);
```

Una vez que se ha registrado el vehículo en el servidor y recibamos el ID de registro del vehículo en el servidor ya podemos empezar a notificar cosas al servidor.
A continuación le enviamos el idgcm del vehículo, para ello generamos el envío mediante el webservice de tipo POST el idgcm de ese vehículo para almacenarlo en el servidor:

```
WebServices.enviarIdgcm(context, new ResponseServer_PostSetIdgcm(context, 0, DataEvent.getDataIdgcm(regid), BBDD.quienSo
```

A continuación le enviamos la posición GPS del vehículo, para saber en todo momento donde se encuentra el vehículo en un JSON que contendrá lo siguiente:

- Fecha: Fecha actual
- Latitud: Latitud del vehículo
- Longitud: Longitud del vehículo
- Sentido: Sentido del vehículo

La posición GPS del vehículo se envía **cada 10 segundos** al servidor para mantenerlo controlado en todo momento, primero comprobamos si hemos recibido el ID de registro del vehículo por parte del servidor, si es así creamos y enviamos el WebService de tipo POST al servidor con el JSON mencionado anteriormente

```
if ((BBDD.quienSoyIdServer(Principal.context) != null) && !(BBDD.quienSoyIdServer(Principal.context).equals(""))) {
    if ((timeLastTrack == null) || (!Mapa.esActualSec(timeLastTrack, 10))) {
        WebServices.enviarTracking(Principal.this, new ResponseServer_PostTracking(Principal.this, 0, DataEvent.getDataTrackir
    }
```

A continuación se comunica con el semáforo, que ya hemos explicado esa comunicación anteriormente.

Una vez recibimos el color por parte del semáforo lo analizamos. Si el color del semáforo es rojo, generamos el evento de Salta Semáforo pasándole como parámetro la velocidad 5 o 10 para la realización de pruebas en parado.

```
else if (color.equals ("TRAFFIC_LIGHT_RED")) {
  return saltaSemaforo(5);
else if (color.equals ("TRAFFIC_LIGHT_FLASHING_RED")) {
  return saltaSemaforo(10);
```

Por último tras ver que el vehículo se ha saltado el semáforo se genera el evento del salto de semáforo de forma totalmente anónima gracias al uso de la firma digital como se verá más adelante en la parte de seguridad y se lo comunica al servidor.

```
DetectSemaforo.ultimaSaltoSemaforo = fecha;
com.iot.dephisit.mocks.Evento.generarEnvio(ctx, "SEM.A28", null, null, fecha, location);
```

A continuación creamos el webservice de tipo POST y lo enviamos al servidor para informar del salto del semáforo como podemos observar a continuación:

```
else if ((tipo.equals("SEM.A28")))  {
  WebServices.informarSaltoSemaforo(ctx, new ResponseServer_PostNuevoEvento(ctx, 0, tipo, DataEvent.getDataSemaforo
```

Enviamos el JSON del evento que se ha generado:

- Idtipovehículo: Que será un vehículo normal

- Fecha: Fecha en que se ha cometido la infracción
- LatitudV: Latitud Vehículo
- LongitudV: Longitud Vehículo
- LatitudS: Latitud Semáforo
- LongitudS: Longitud Semáforo
- Hash: Concatenación entre la posición GPS del semáforo y la fecha actual (timestamp).

Una vez que le llega al servidor el evento, lo genera como vimos en el capítulo anterior y busca vehículos a los que notificar el evento generado en la aplicación mediante el servicio gcm que ya vimos anteriormente.

En el caso de que seamos los receptores de la notificación, el funcionamiento sería el siguiente.
Una vez que se comunica el servidor gcm con la aplicación para notificarle la infracción cometida por alguien alrededor nuestro,se pinta en la aplicación la posición GPS del infractor con un semáforo como indicador y nos advierten de que hay un conductor que ha cometido una infraccion a sus alrededores, que nos mantengamos alerta, marcandonos en el lado del servidor la posicion GPS de dicho vehículo.

```
if (marker != null)
    Mapa.marcadoresPintados.put(ev, marker);
Voz.hablar(EventosTypes.getMsg(ctx, ev.getIdtipoevento()));
```

7.4 Interfaz

La interfaz del usuario ha sido diseñada mediante XML (parte gráfica). En la carpeta layout del proyecto es donde se encuentran los archivos de diseño de todas las actividades.

En la Fig.32 se muestra en la pantalla izquierda, el icono por defecto de la aplicación y en la pantalla derecha la vista de bienvenida al usuario de la aplicación.

Fig. 32 Icono App vs Activity_splash_screen.xml

Una vez que el usuario inicia la aplicación se invoca al menú principal como observamos a continuación:

```
setContentView(R.layout.activity_principal);
```

En el menú principal, una vez que se ha iniciado la aplicación y tenemos conectividad mediante Wi-Fi o por datos móviles (3G o 4G) y activado el servicio de GPS, la aplicación se conecta con los servicios de Google Maps para cargar los planos del sitio donde nos encontremos. Nos marca la posición GPS donde nos encontremos.

En la Fig.33 se muestra la interfaz con los servicios de Google Maps indicando la posición GPS del vehículo y marcando dicha posición. En la pantalla derecha se muestra las opciones de configuración que tiene activada el smartphone.

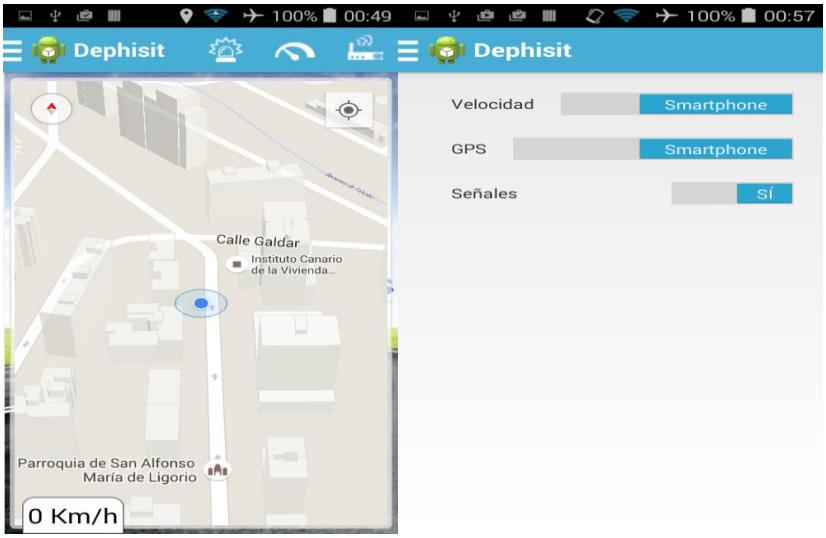


Fig. 33. Activity_Principal.xml

```
Mapa.map = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map)).getMap();
public final com.google.android.gms.maps.GoogleMap getMap()
```

Como podemos observar, una vez que el usuario selecciona en la esquina superior izquierda se despliega el menú

En la Fig.34 se muestra el menú de la aplicación Dephisit.

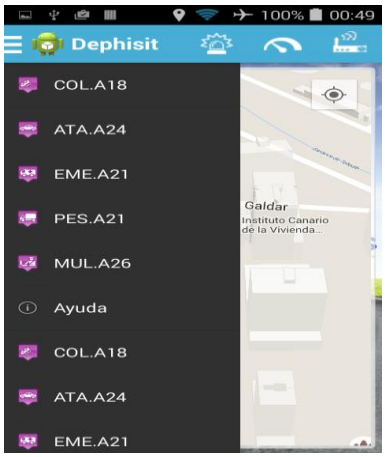


Fig. 34 Menú Aplicación

Una vez que la aplicación nos marca la posición GPS actual, se envía la aplicación la posición GPS

del vehículo (nuestra posición) al servidor cada 10 segundos, siempre cuando éste le haya devuelto el ID de registro en el servidor para que pueda empezar a notificar cosas al servidor. Como podemos observar aparecen varias posiciciones ya que cada 10 segundos esta verificando si el vehículo se ha movido para notificarselo al servidor y mandarle la nueva posición.

```
C:\Users\Rushil\Desktop\Android\TFG\cryptull-dephisitserver-476e7f5423fa\cryptul
l-dephisitserver-476e7f5423fa>node dephisit.js
Lanzado en puerto 3000...
Nuevo tracking en -16.2449606, 28.4806185 y sentido = 0
Nuevo tracking en -16.2449814, 28.4806041 y sentido = 0
Nuevo tracking en -16.2449635, 28.4806093 y sentido = 0
Nuevo tracking en -16.244957, 28.480611 y sentido = 0
Nuevo tracking en -16.2449616, 28.4806095 y sentido = 0
Nuevo tracking en -16.2449697, 28.4806068 y sentido = 0
Nuevo tracking en -16.2449712, 28.4806066 y sentido = 0
Nuevo tracking en -16.2449723, 28.480606 y sentido = 0
Nuevo tracking en -16.2449735, 28.480605 y sentido = 0
Nuevo tracking en -16.2449741, 28.480603 y sentido = 0
```

Suponemos que estamos en la situación en donde nos encontramos en mitad de la carretera frente a un semáforo y nos lo hemos saltado por tanto generamos el evento del salto del semáforo.

En la Fig.35 se muestra los eventos que pueden generar la aplicación y la selección por parte del usuario del evento del salto de semáforo.

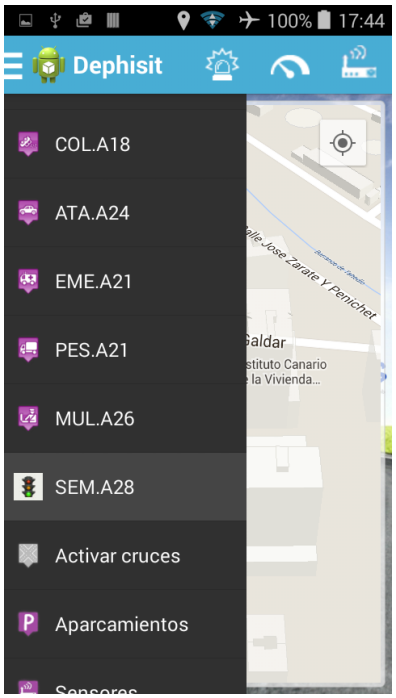


Fig. 35. Evento Salto de Semáforo

Por tanto una vez el usuario selecciona la opción de generación del evento del salto de semáforo (SEM.A28), inmediatamente comienza la comunicación con el semáforo mediantes los sensores Bluetooth Low Energy (BLE) como vimos en el capítulo de la comunicación semáforo-vehículo.

El semáforo envía la información siguiente a la aplicación:

- Posición GPS del semáforo (Latitud, Longitud)
- Color del semáforo
- ID Semáforo
- Hash (Posicion GPS Semaforo | Timestamp)

Una vez que la aplicación recibe la información por parte del semáforo se genera el envío del salto de semáforo gracias al servidor de GCM explicado anteriormente, y notifica a los vehículos cercanos que encuentre enviándole la posición GPS de dicho vehículo. En este caso no hemos encontrado a ningún vehículo que se encuentre a menos de 500 metros a quien notificarle que alguien a su alrededor se ha saltado el semáforo que hemos sido nosotros, en donde nuestra identidad permanecerá siempre anónima.

```
Nuevo evento SALTA SEMAFORO :-16.2449507, 28.4806289
{"idpromotor":null,"idtipoevento":"SEM.A28","detalles":null,"idtipovehiculo":"TU
_NORMAL","fecha":"2015-06-30T18:50:17.000Z","latitud":28.4806289,"longitud":-16.
2449507,"sentido":-1}
Los receivers (vehiculos) son
[]
```

7.5 Caso de Uso

Suponemos que el vehículo se ha saltado el semáforo y ya ha generado el evento del salto de semáforo.

En la Fig.36 se muestra un caso de uso de la comunicación tras la generación del evento del salto del semáforo.

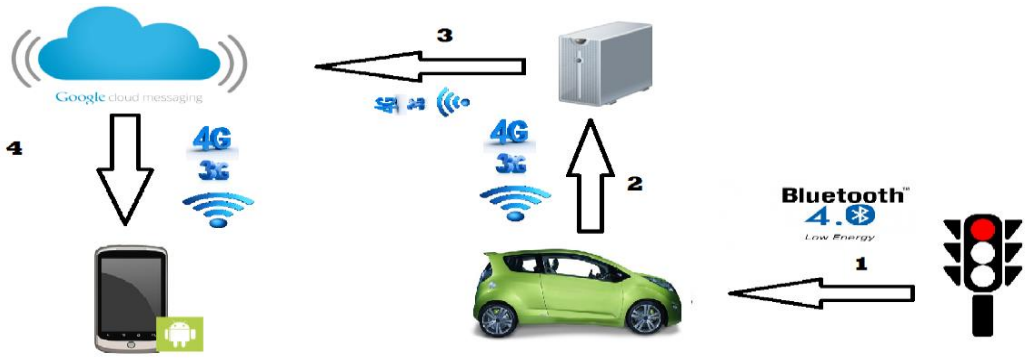


Fig. 36. Caso de Uso

1) Comunicación Semáforo Vehículo

El semáforo se comunica con el vehículo mediante los sensores Bluetooth Low Energy integrados en el semáforo. Una vez que recibe la información correspondiente del semáforo, inicia la comunicación con el servidor.

2) Comunicación Aplicación - Servidor

El vehículo ya se encuentra registrado en el servidor, ya que es lo primero que realiza al iniciarse la aplicación. La comunicación se realiza mediante Wi-Fi/3G/4G .Genera el webservice de tipo POST

y le envía la información correspondiente del evento del salto de semáforo.

- Posición GPS Semáforo
- Posición GPS Vehículo
- Tipo Evento
- Sentido Vehículo
- Id Promotor
- Id destino
- Detalles
- Fecha

3) Comunicación Servidor – GCM
Una vez que el servidor recibe la información por parte de la aplicación, inicia la comunicación mediante 3G/4G/Wi-Fi con el servicio de Google Cloud Messaging que va a ser el encargado de notificarle el mensaje a los vehículos cercanos. El servicio GCM comprueba que los vehículos a los que notificar la infracción se encuentran registrados en dicho servicio con su ID correspondiente. Una vez que lo confirma consultando la base de datos de mongo en el servidor activa el servidor GCM para notificar a los vehículos correspondientes.

4) Comunicación GCM – Vehículo
El servidor de Google Cloud Messaging genera un push con los datos correspondientes que le ha enviado el servidor que son los que mencionamos anteriormente en la comunicación aplicación servidor y se lo envía a los vehículos cercanos mediante Wi-Fi /3G/4G. Una vez que recibe la notificación recibe también por parte del servidor todos los datos donde se ha cometido la infracción del evento generado y marca en el mapa de la aplicación la posición GPS del infractor con el icono de un semáforo para que pueda anticiparse ante posibles accidentes de tráfico.

8. Seguridad de la Aplicación

8.1 SSL

Definición

SSL [15] (Security Socket Layer) es un protocolo criptográfico de uso común empleado para establecer conexiones mediante un canal seguro entre un cliente y un servidor. Permite establecer una sesión segura que requiere una mínima intervención por parte del usuario final. Cifra y protege los datos transmitidos utilizando el protocolo HTTPS. Garantiza la integridad de las comunicaciones en las redes mediante certificados digitales que implican procesos de encriptación, autenticación y verificación. Es decir, proporciona autenticación de las partes que participan en las transacciones en línea y encripta las sesiones de comunicación. Garantiza a los usuarios de su sitio web que sus datos no serán interceptados de manera fraudulenta proporcionando de esta forma privacidad e integridad.

Funcionamiento

- El cliente y el servidor entran en un proceso de negociación, conocido como handshake [16]. Este proceso sirve para realizar la conexión de forma segura.
- El cliente solicita al servidor web que se identifique
- El servidor envía al cliente una copia de su certificado SSL
- El cliente comprueba la veracidad del certificado SSL. Si es así envía un mensaje al servidor
- El servidor devuelve una confirmación mediante firma digital para comenzar una sesión con cifrado SSL (los datos encriptados se comparten entre cliente y servidor). Se codifica y descodifica todo lo que sea enviado hasta que la conexión se cierra.

En la Fig.37 se muestra el funcionamiento del protocolo SSL

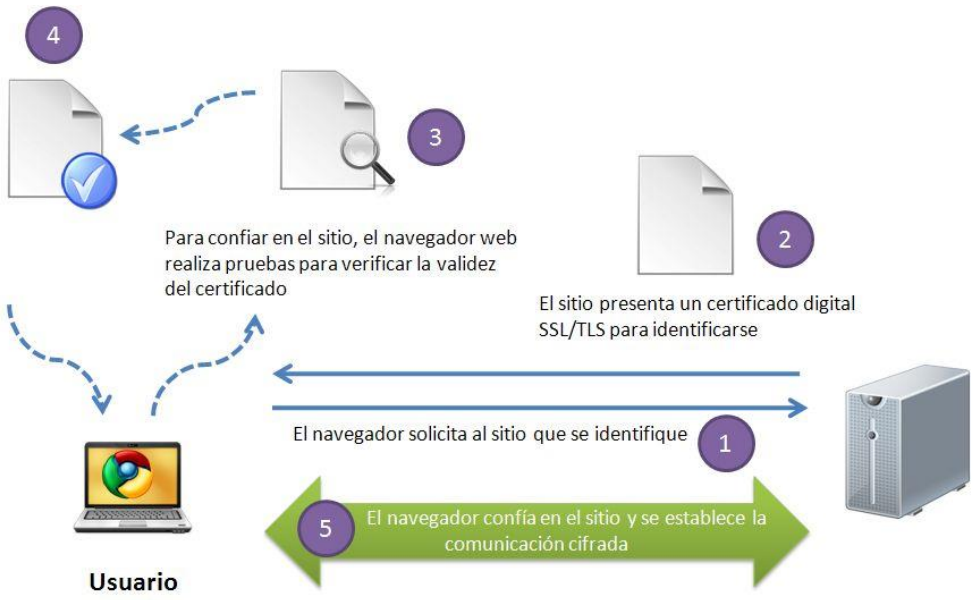


Fig. 37. Funcionamiento del Protocolo SSL

Protocolo Handshake

Permite tanto al cliente como al servidor autenticarse mutuamente y negociar el cifrado antes de intercambiar datos. Negocia y establece la conexión. Sus características principales son:

- Autenticación asimétrica de al menos una de las dos partes
- Negociación del secreto compartido (simétrico) seguro, mediante técnicas de encriptación de clave pública (asimétricos). Se derivará una clave simétrica común con la que se cifrarán los datos.
- Negociación confiable: no es posible una modificación sin que ninguna de las dos partes se dé cuenta de ello.
- Proporciona los parámetros de seguridad a la capa de registro.

A continuación se muestra el intercambio inicial necesario para establecer una conexión lógica entre el cliente y el servidor. El intercambio se divide en cuatro fases:

En la Fig.38 se muestra las fases que sigue el protocolo handshake [16].

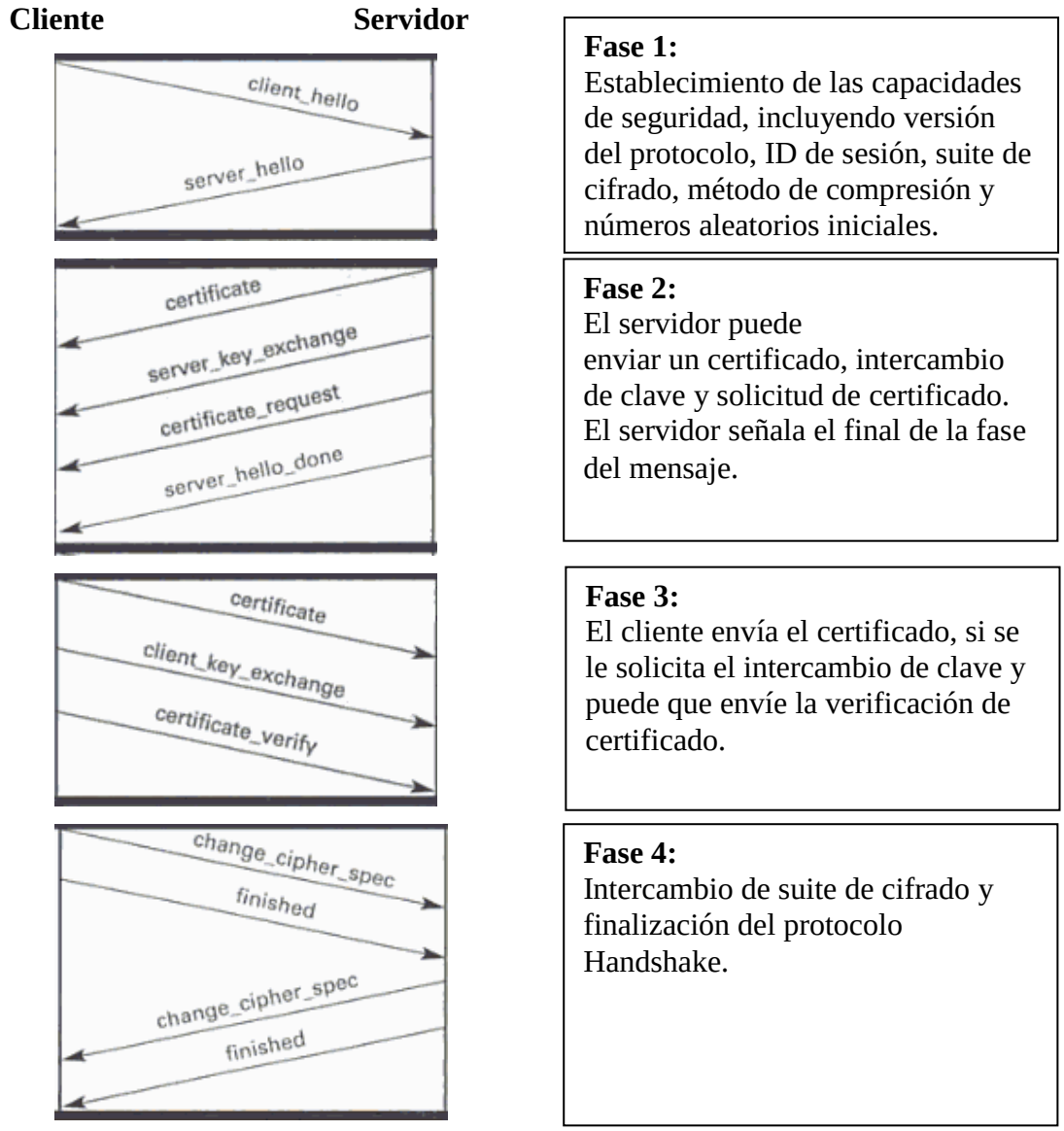


Fig. 38. Acción del protocolo Handshake

OpenSSL

OpenSSL [17], es una implementación criptográfica muy potente de código libre del protocolo SSL Proporciona un entorno adecuado para encriptar los datos enviados a otro ordenador dentro de una red y a su vez descifrarlo adecuadamente por el receptor, evitando así, el acceso a la información por intrusos con la utilización de sniffer.

Características:

- Creación de claves DSA y RSA
- Creación de certificados x.509
- Calcular "hash" de ficheros
- Cifrado y descifrado de datos
- Pruebas SSL/TLS tanto en el lado del cliente como el lado del servidor

OpenSSL dispone los mejores algoritmos de encriptación simétrica y asimétrica, así como de potentes funciones hash. Cabe destacar los siguientes:

- Algoritmos de hash

  md2, md5, mdc2, rmd160, sha, sha1, sha224, sha256, sha384, sha512

- Algoritmos de cifrado

  Base 64, Blowfish, CAST5, DES, Triple-DES, IDEA, RC2, RC4, RC5, AES

El algoritmo de hash SHA-512 ha sido el utilizado en este proyecto para la integridad de datos, ya que es el hash lo que se envía al receptor firmado digitalmente junto al texto en claro. Si el receptor recibe el mismo hash que genera junto a los textos en claro, se han enviado los datos sin modificaciones de algún intruso.

En la Fig.39 se muestra la suite utilizada Open-SSL para la generación de certificados tanto en el lado del servidor como en el lado del cliente.

Fig. 39. Logo de OpenSSL

8.2 Firma Digital

Es un mecanismo criptográfico que permite al receptor de un mensaje firmado digitalmente determinar la entidad originadora de dicho mensaje, es decir que ha sido realizado por la persona u

organización que afirma haberlo creado.

Suelen ser archivos con extensión SIG o ASC que resultan de firmar criptográficamente con la clave privada del autor el hash de un fichero.

Está compuesta por una clave pública y otra privada, donde la clave pública se suele distribuir por una autoridad de certificadora que es la entidad en la que todos confían, tanto el receptor como el emisor, y es también la encargada de generar el certificado digital.

Si se comprueba, a través de la clave pública, que el fichero firmado concuerda con la firma, es que se está ante un fichero realmente creado por quien dice haberlo hecho, y no modificado desde que se firmó. Una vez más, esto no garantiza en ningún modo las intenciones del archivo, sólo su origen.

En la Fig.40 se muestra el funcionamiento que se lleva a cabo de la firma digital [18].

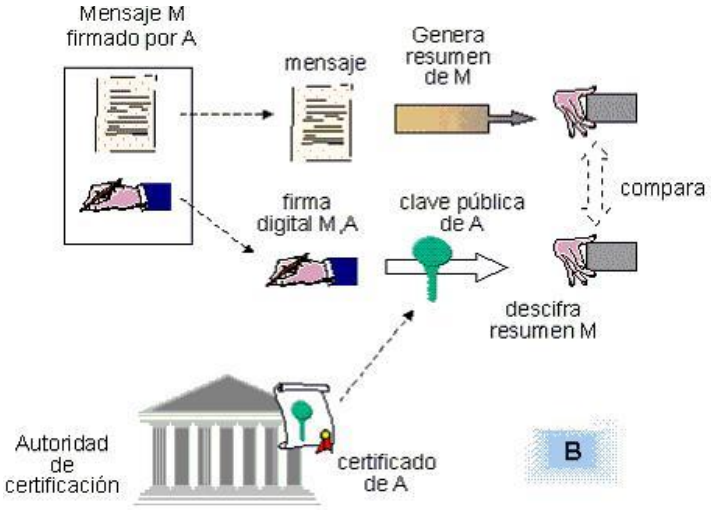


Fig. 40. Funcionamiento de la Firma Digital

El esquema de funcionamiento sería el siguiente:

- El emisor genera un resumen del fichero ya que la firma no se realiza sobre el fichero completo sino en un resumen del mismo o hash.
- El emisor firma su resumen encriptándolo con la clave privada propia.
- El emisor envía el fichero y su resumen firmado al receptor.
- El receptor genera también un resumen del documento que ha recibido usando la misma función que el emisor. Al mismo tiempo descifra el resumen recibido con la clave pública que el emisor ha publicado. Si los resúmenes coinciden la firma será validada.

ECDSA: Definición

El algoritmo de firma digital para curvas elípticas está basado en el estándar de firma digital DSA. Este algoritmo ofrece un esquema que permite firmar documentos y verificar las firmas. Es una modificación del algoritmo DSA que emplea operaciones sobre puntos de curvas elípticas en lugar de las exponenciaciones que usa DSA .La principal ventaja de este esquema es que requiere números de tamaños menores para brindar la misma seguridad que DSA o RSA

Aplicación ECDSA: Proyecto

En el caso del proyecto desarrollado, se utiliza un algoritmo ECDSA [19] gracias a la herramienta Open-SSL que ofrece algunas opciones para trabajar con curvas elípticas. Los datos que se envían de la aplicación al servidor van a ir firmados digitalmente para comprobar la veracidad de la identidad del emisor cuando lo reciba el servidor y viceversa, ya que se intercambian datos mutuamente como hemos visto en capítulos anteriores. Todos los datos se envían con su firma digital.

Anonimato

Para garantizar el anonimato del que genera el evento un grupo de usuarios comparte el mismo certificado para poder firmar, de manera que cuando se envíe la información firmada digitalmente y la verifiquemos en el servidor y comprobemos si la firma es válida, sabremos que la fuente es fiable ya que viene de un grupo que conocemos pero no sabremos exactamente la persona en particular que envío la información firmada digitalmente.

El usuario firma el mensaje con su clave privada y el servidor comprueba la firma con la clave pública del usuario. La firma es una por cada grupo de usuario.

8.3 Hash

Definición

Los hash [20] o funciones de resumen son algoritmos que consiguen crear a partir de una entrada (ya sea un texto, una contraseña o un archivo, por ejemplo) una salida alfanumérica de longitud normalmente fija que representa un resumen de toda la información que se le ha dado (es decir, a partir de los datos de la entrada crea una cadena que solo puede volverse a crear con esos mismos datos).

En la Fig.41 se muestra un ejemplo de un hash donde a partir de una entrada (texto, contraseña…) se genera una salida alfanumérica que representa toda la información dada.

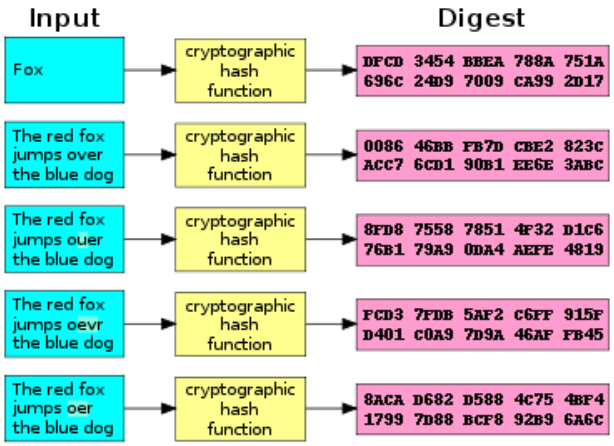


Fig. 41. Ejemplo Hash

En seguridad de la información, se utilizan funciones hash criptográficas en procesos de autenticación, o de comprobación de integridad de datos.

Lo que se obtiene al aplicar una función hash criptográfica a un mensaje (flujo de datos, o más usualmente, un archivo) se llama resumen criptográfico, huella digital o message digest.

Aplicación del Hash en el Proyecto

Comprobar la integridad de un fichero consiste en averiguar si algún dato del archivo ha variado desde su creación. El archivo puede haber sido modificado por error, por cortes en la comunicación o, en el peor de los casos, porque un atacante haya inyectado código malicioso. Para ello se utiliza el algoritmo HASH SHA-512, para verificar que la información llega intacta y sin modificaciones al receptor.

El procedimiento que se sigue en el proyecto es el siguiente:

1) La aplicación envía el texto en claro y el hash que genera a partir del texto en claro.

2) El servidor recibe los datos en claro y el hash. Con los datos en claro, realiza el hash y comprueba con el hash que recibió del servidor, es el mismo. Si coinciden es que el 'Texto En Claro' no ha sido modificado.

3) Un posible intruso puede interceptar el mensaje y querer cambiar el 'Texto en Claro', pero el hash nunca podrá modificarlo para obtener un hash nuevo, ya que se trata de una operación muy compleja.

Las características que ofrece este algoritmo son las siguientes:

- Disponibilidad. El sistema y los datos tienen que ser accesibles por los usuarios autorizados en todo momento.
- Integridad. Nadie puede cambiar, recortar o falsificar ilegítimamente los datos.

8.4 Implementación Open-SSL

En el proyecto se ha hecho uso de la suite de Open-SSL.
Para generar las claves en el servidor y en la aplicación se realiza lo siguiente:

Generación de certificados del lado del servidor

Creamos la clave privada en PEM
openssl genpkey -algorithm <nombrealgoritmo> -pkeyopt <algorithm>_keygen_bits:2048 -out server.private.pem;

La cambiamos a DER con PKCS#8 para que Android la pueda entender cuando intercambiamos claves
openssl pkcs8 -topk8 -inform PEM -in server.private.pem -outform DER -out server.private.der -nocrypt

Generamos la clave pública en DER a partir de la privada en PEM
openssl <algorithm> -in server.private.pem -pubout -outform DER -out server.public.der

Los certificados se generan con Open-SSL en el servidor. Se crea una clave pública y una clave privada para cada grupo de usuarios. La clave privada se envía de forma manual al Smartphone que tenga instalada la aplicación. La clave pública se almacena en el servidor.

Para calcular el hash de los datos con Open-SSL se utiliza el siguiente comando:
openssl dgst [-sha] <nombre_archivo>

Para obtener el valor del hash se utiliza el siguiente comando:
openssl x509 -noout -in cert.pem –hash

Para ver la huella digital MD5:
openssl x509 -noout -in cert.pem –fingerprint

Para firmar el archivo digitalmente en la aplicación sería:
openssl dgst -c -sign privada.key -out firmado.sig entrada.txt

Para verificar la firma en el servidor:
openssl dgst -c -verify publica.key -signature firmado.sig entrada.txt

9. Presupuesto

Este capítulo incluye los cálculos estimados de costes que supondría el desarrollo de este trabajo en un entorno real. Para ello se distinguen costes en tres categorías: personal, componentes y licencias.

9.1 Costes de personal

En la Tabla 1 se muestra el coste de personal, en este caso un único ingeniero para las diferentes tareas.

Tarea	Horas	Coste	Coste/Hora
Investigación, documentación y análisis de las tecnologías	50	1400	28
Sensores integrados en los semáforos	30	840	28
Estudio de esquemas curvas elípticas	10	280	28
Análisis y diseño de la aplicación móvil	40	1120	28
Desarrollo de aplicación	120	3360	28
Desarrollo de servidor	80	2240	28
Total	**330**	**9240**	**168**

Tabla 1. Relación de tareas y coste estimado

9.2 Costes de componentes Hardware

La Tabla 2 muestra los costes del hardware necesario para el desarrollo de este trabajo.

Equipo	Fabricante	Unidades	Coste
Lenny Wiko	Wiko	1	110
Ordenador portátil	Dell XPS 15	1	1200
Total			**1310**

Tabla 2. Costes de componentes Hardware

9.3 Costes de componentes Software

La Tabla 3 muestra los costes de componentes y licencia software necesario para el desarrollo de este trabajo.

Componentes Software y licencias	Coste	Coste /Mes
Hosting Amazon EC2	120	10
Licencia Desarrollador Android	18.75	18.75/25 Años
Total		

Tabla 3. Costes de componentes y licencia Software

9.4 Coste total del proyecto

Se muestra en la Tabla 4 el presupuesto total estimado para la realización de este trabajo.

Concepto	Coste
Personal	9240
Hardware	1310
Software	138.75
Total	**10688,75**

Tabla 4. Coste total del proyecto

Teniendo en cuenta todo lo anterior, el coste de la realización de este trabajo de final de grado asciende a un total de 10.688,75

10. Conclusiones y Líneas Futuras

Desde antaño, los sistemas inteligentes de transporte se han ido incorporando poco a poco a la sociedad actual. Se trata de un conjunto de aplicaciones prácticas que integran las tecnologías de la información y de las comunicaciones, tanto en los vehículos como en las carreteras, de manera que dichos avances tecnológicos proporcionan grandes mejoras para la seguridad vial y la gestión del tráfico y del transporte, y ofrecen capacidad al usuario para elegir de forma inteligente de qué manera prefiere hacer su viaje.

Los sistemas inteligentes de transporte permiten reducir el grado de incertidumbre, antes y durante el viaje, gracias al conocimiento de la ruta y de los posibles incidentes. Merece la pena destacar que dichos sistemas permiten proporcionar información al viajero, incluyendo datos de las condiciones de tráfico en tiempo real, y de sistemas de asignación dinámicas de tráfico y guiado automático en ruta. Por tanto, la creación de un sistema inteligente de detección y aviso sobre infracciones llevadas a cabo por infringir semáforos, basado en al ecosistema de las VANETs y desarrollado mediante teléfonos inteligentes, es un gran avance en la seguridad vial.

Este proyecto provee una nueva aplicación para detectar cuándo un vehículo se ha saltado un semáforo, y notificar dicho hecho de forma anónima, rápida y veraz a todos los vehículos cercanos. Con dicho objetivo, se ha hecho uso de sensores inteligentes de Bluetooth Low Energy que se sitúan en los semáforos y emiten constantemente su estado. Los vehículos captan esta señal y corroboran si han infringido el semáforo o no, para notificarlo, de forma anónima gracias al uso de algoritmos criptográficos que protegen la privacidad, a un servidor remoto que es el encargado de notificar al resto de vehículos, mediante notificaciones push instantáneas de la negligencia que está ocurriendo en esa zona.

En particular, hemos elaborado una aplicación cliente bajo la plataforma Android, con objeto de identificar e interactuar a los diferentes vehículos implicados del sistema de manera que cualquier usuario con un simple smartphone puede ser partícipe del ecosistema que se genera. Para la parte del servidor se han utilizado tecnologías javascripts y bases de datos no relacionales, que permiten un gran escalado de forma dinámica. Todas las tecnologías usadas son Open Source, así como las herramientas de desarrollo utilizadas para la implementación.

Este trabajo incluye el diseño y la implementación de dicha aplicación, así como la configuración de los niveles de seguridad requeridos tanto en la aplicación como en el servidor, haciendo uso de la suite de OpenSSL.

El proyecto deja pendientes varias futuras funcionalidades que podrían añadirse a lo que ya se ha implementado, gracias al diseño e implementación modular que se ha seguido. Como líneas futuras de este trabajo destacan: la elección de parámetros óptimos para detectar que un usuario se ha saltado un semáforo, la posibilidad de configurar esos parámetros de forma dinámica y personalizable para cada semáforo, la creación de una interfaz en el servidor para que se pueda ver todo lo que sucede en tiempo real y la comparación de la propuesta con otras similares.

11. Conclusions and Open Problems

Since ancient times, intelligent transportation systems have been incorporated gradually into today's society. They form a set of practical applications that integrate information technology and communications, both in vehicles and on the roads, so that these technological advances provide significant improvements to road safety and traffic management and transport and offer the user ability to choose wisely how it would like to make the trip.

Intelligent transportation systems can reduce the uncertainty, before and during the journey, thanks to the knowledge of the route and possible incidents. It is worth noting that these systems can provide traveller information, including data on traffic conditions in real time and of dynamic allocation systems and automatic guided traffic en route. Therefore, creating an intelligent detection and warning system on infringements carried out for breaking traffic lights, based on the ecosystem of the VANETs and developed by smartphones, it is a breakthrough in road safety.

This project provides a new application to detect when a vehicle jumped a red light, and notify that fact anonymous, fast and true to all nearby vehicles. With this objective, we have made use of smart Bluetooth Low Energy sensors that are placed at traffic lights and are constantly emitting their status. Vehicles capture this signal and confirm whether they have violated the traffic lights or not, to notify anonymously through the use of cryptographic algorithms to protect privacy, to a remote server that is responsible for notifying the other vehicles through push notifications snapshots the infraction that is happening in that area.

In particular, we have developed a client application on the Android platform, in order to identify and interact with the various vehicles involved in the system so that any user with a simple smartphone can be part of the ecosystem that is generated. For the server side, JavaScript technologies and non-relational databases have been used, allowing a large scale dynamically. All the used technologies are Open Source, as well as the development tools used for implementation.

This project includes the design and implementation of the application, as well as the setting of the levels of security required in both the application and the server, using the OpenSSL suite.

The project has left different possible functionalities that could be added to what has already been implemented, thanks to the modular design and implementation that has been followed. As future lines of this project are: the choice of optimal parameters for detecting that a user has jumped a traffic light, the ability to set these parameters dynamically and customizable for each traffic light, creating an interface on the server to see everything that happens in real time, and the comparison between the proposal and other similar proposals.

12. Bibliografía

[1] AudiTravolution:http://www.miscocheselectricos.com/audi-travolution-comunicacion-semaforos-vehiculos-2354.html (Accedida en Abril de 2015)

[2] Jen Hsun «Huang Visual Computing: The Road Ahead » 2015.

[3] Jen Hsun «Huang Visual Computing: The Road Ahead » 2015.

[4] Jen Hsun «Huang Visual Computing: The Road Ahead » 2015.

[5] Honda:http://www.autopista.es/tecnologia/todo-tecnologia/articulo/honda-prueba-sistema-comunicacion-coche-semaforos-100066 (Accedida en Marzo de 2015)

[6] IBM:http://www.cookingideas.es/ibm-quiere-un-semaforo-que-detenga-el-motor-de-los-coches-a-distancia-20100525.html (Accedida en Marzo de 2015)

[7] V2V: http://www.clicktaller.es/blog/sistema-v2v/ (Accedida en Abril de 2015)

[8] Cruz, A., & David, J. (2009). Estudio y simulación de una red ad-hoc vehicular VANET.

[9] Collado, J. M., Hilario, C., Armingol, J., & de la Escalera, A. (2003). Visión por computador para vehículos inteligentes. XXIV Jornadas de Automática, León, España.

[10] BLE:http://www.libelium.com/bluetooth-low-energy-ble-4-0-smart-connect-sensors-smartphone/ (Accedida en Junio de 2015)

[11] Paredes, G. G. (2006). Introducción a la Criptografía. Revista Digital Universitaria, 7(7).

[12] Zabler, E. (2002). Los sensores en el automóvil. Reverte.

[13] Fernández Amador, G. (2005). Sensores magnéticos e inductivos.

[14] Santamaría, M. V. B., & Moscol, M. F. R. (2014). Semáforos Inteligentes para la Regulación Del Tráfico Vehicular. Ingeniería: Ciencia, Tecnología e Innovación, 1(1).

[15] Cobas, J. D. G. (2005). Secure Sockets Layer (SSL).

[16] William Stalling (2004). Fundamentos de seguridad en redes

[17] Young, E. A., Hudson, T. J., & Engelschall, R. S. (2001). OpenSSL. World Wide Web, http://www. openssl. org/, Last visited, 9.

[18] Mendillo, V. (2009). Firma Digital y Sellado de Tiempo.

[19] Belingueres, G. (2000). Introducción A Los Criptosistemas de Curva Elíptica. Obtenido en la Red Mundial el, 5.

[20] Escalona, S. B., & Inclán, L. V. (2012). Funciones resúmenes o hash. Revista Telem@ tica, 10(1).