PROYECTO FINAL DE CARRERA

SMART CITY DESIGN FOR VEHICULAR NETWORKS

(DISEÑO DE UNA CIUDAD INTELIGENTE PARA REDES VEHICULARES)

Estudios: Ingeniería de Telecomunicaciones

Autor: Miguel Ángel Mateos Márquez

Directora: Mónica Aguilar Igartua

Co-Directora: Carolina Tripp Barba

Lugar y Fecha: Barcelona, Marzo 2012

AGRADECIMIENTOS

Primero que todo me gustaría agradecer a Mónica y Carolina por toda la ayuda, apoyo y tiempo dedicado a este proyecto, además de haberme dado la oportunidad de ser parte del mismo. También me gustaría agradecer al resto de proyectistas que formaron parte del grupo de trabajo durante el desarrollo de todos y cada uno de sus proyectos individuales, en especial a Pablo Regañas Soto, con el cual he compartido ideas, problemas y soluciones.

Seguidamente me gustaría agradecer a mis padres, hermanas y familiares cercanos. A mis padres por haberme dado una buena educación y la posibilidad para desarrollar mis habilidades y mi potencial al máximo, siempre siendo un apoyo incondicional durante todo el proceso. A mis hermanas por haberme ayudado durante toda mi vida a ser la persona que soy, y a estar siempre centrado en el lugar que debería de estar. A mis familiares cercanos por haberme dado acogida durante este tiempo de estudios en el exterior, abriéndome las puertas de su hogar y haciéndome parte de su vida.

Me gustaría agradecer también a una persona muy especial en mi vida, que a pesar de todas las situaciones siempre ha sabido ser constante y un apoyo incondicional. Gracias Isabella.

También me gustaría agradecer a mis amigos, cercanos, lejanos, venezolanos, españoles… Sin importar de dónde son o cómo los conocí me han permitido reír, llorar, disfrutar y tener un descanso de todo el proceso académico tan importante para mantener un equilibrio.

Por último, me gustaría agradecer a la Universidad Católica Andrés Bello por haberme brindado la oportunidad de ser uno de los pocos privilegiados de optar por la Doble Titulación como Ingeniero en Telecomunicaciones UCAB-UPC. No fue fácil, pero dio sus frutos.

A todos los nombrados anteriormente sólo puede decirles una cosa: ¡Muchísimas Gracias!...

Miguel Ángel Mateos Márquez

INDEX

INDEX OF FIGURES

INDEX OF TABLES

GLOSSARY OF ACRONYMS

A-STAR	Anchor based Street and Traffic Aware Routing
ACK	ACKnowledgment
AODV	Ad-hoc On Demand Distance Vector
AP	Access Point
API	Application Programming Interface
APTEEN	Adaptive Periodic Threshold-sensitive Energy Efficient sensor Network Protocol
ATT	Advance Transport Telematics
C2C-C	Car to car Communication Consortium
CAR	Connectivity Aware Routing
CAR-2-X	Car to both car and infrastructure
COMeSafety	Communications for eSafety
CPU	Central Processing Unit
CVIS	Co-operative Vehicle-Infrastructure Systems
DAB	Digital Audio Broadcast
DD	Directed Diffusion
DIR	Distance Routing
DRIVE	Dedicated Road Infrastructure for Vehicle Safety in Europe
DSDV	Dynamic Destination-Sequenced Distance-Vector
DSR	Dynamic Source Routing
DSRC	Dedicated Short Range Communications
DYMO	Dynamic On demand MANET Routing Protocol
ENoLL	European Networks of Living Labs
ENTEL	Departament d'ENginyeria TELemàtica
EU	European Union
EUREKA	European Research Coordination Agency
FIA	Fédération Internationale de l'Automobile
FP	Framework Program

FSR	Fisheye State Routing
GAF	Geographic Adaptive Fidelity
GEAR	Geographic and Energy Aware Routing
GEDIR	Geographic Distance Routing
GeOpps	Opportunistic Geographical Routing
GPCR	Greedy Perimeter Coordinator Routing
GPRS	General Packer Radio Service
GPS	Global Positioning System
GRUB	Grand Unified Bootloader
GSR	Geographic Source Routing
GUI	Graphical User Interface
HPAR	Hierarchical Power-aware Routing
HSVN	Hybrid Sensor and Vehicular Network
IARP	Intrazone Routing Protocol
ICT	Information and Communications Technology
ID	IDentification
IEEE	Institute of Electrical and Electronic Engineers
IERP	Interzone Routing Protocol
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
ITS	Intelligent Transportation System
LAR	Local Aided Routing
LDM	Local Dynamic Map
LEACH	Low Energy Adaptive Clustering Hierarchy
LRWPAN	Low Rate Wireless Personal Area Network
MAC	Media Access Control
MANET	Mobile Ad hoc NETwork
MCFA	Minimum Cost Forwarding Algorithm
MECN	Small Minimum Energy Communication Network
MFR	Most Forward within Radius

NCTUns	National Chiao Tung University Network Simulator
NOW	Network On Wheels
ODMRP	On Demand Multicast Routing Protocol
OS	Operating System
P2P	Peer to Peer
PC	Personal Computer
PEGASIS	Power-Efficient Gathering in Sensor Information Systems
PEWASUN	Performance Evolution of Wireless Ad hoc, Sensor Ubiquitous Networks
PHY	Physical
PPPP	Public-Private-People-Partnerships
PROMETHEUS	Programme for a European Traffic with Highest Efficiency and Unprecedented Safety
R&D	Research and Development
RERR	Route ERRor message
RREP	Route REPly message
RREQ	Route REQuest message
SAR	Spatial Aware Routing
SN	Sensor Node
SOP	Self-Organizing Protocol
SPIN	Sensor Protocols for Information via Negotiation
TBF	Trajectory Based Forwarding
TCP	Transfer Control Protocol
TEEN	Threshold-sensitive Energy Efficient Protocol
TTDD	Two-Tier Data Dissemination
TTL	Time To Live
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VADD	Vehicular Assisted Data Delivery
VANET	Vehicular Ad-hoc NETwork
VGA	Virtual Grid Architecture routing
VM	Virtual Machine

VoIP	Voice over Internet Protocol
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WiFi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WRP	Wireless Routing Protocol
WSN	Wireless Sensor Network
ZRP	Zone Routing Protocol

RESUM

La seguretat a la carretera ha esdevingut un problema principal pels governs i pels fabricants d'automòbils en els últims anys. El desenvolupament de noves tecnologies de vehicles ha afavorit a les empreses, els investigadors i les institucions que centrin els seus esforços a millorar la seguretat viària. Durant les últimes dècades, l'evolució de les tecnologies sense fils ha permès als investigadors a dissenyar sistemes de comunicació on els vehicles poden participar en les xarxes de comunicació. D'aquesta manera, es crea el concepte de Sistema de Transport Intel·ligent (STI), concepte utilitzat en parlar sobre les tecnologies de comunicació entre vehicles i infraestructura que milloren la seguretat vial en el transport, la seva millor gestió, l'eficiència mediambiental, etc.

A causa de l'alt cost econòmic de provar STI en situacions reals, l'ús de simuladors és realment útil a l'hora de desenvolupar STI. Així, en aquest projecte el simulador NCTUns ha estat modificat amb l'objectiu d'afegir noves possibilitats al simulador que ajudin a dissenyar STI a futurs usuaris. A més, un escenari d'una ciutat intel·ligent ha estat simulat amb l'objectiu d'avaluar com l'ús de la xarxa permet la recol·lecció i el càlcul d'estadístiques en temps real, a més de comprovar com funcionen els canvis realitzats en el simulador.

RESUMEN

La seguridad en la carretera se ha convertido en un problema principal para gobiernos y fabricantes de automóviles en los últimos años. El desarrollo de nuevas tecnologías vehiculares ha permitido a compañías, investigadores e instituciones a centrar sus esfuerzos para mejorar la seguridad vial. Durante las últimas décadas, la evolución de la tecnología de comunicación inalámbrica ha permitido a investigadores el diseño de sistemas de comunicación en los cuales los vehículos forman parte de la red de comunicación. De esta forma, se creó el concepto de Sistema de Transporte Inteligente (STI), concepto utilizado al hablar sobre las tecnologías de comunicación entre vehículos e infraestructura, que mejoran la seguridad vial en el transporte, su mejor gestión, eficiencia medioambiental, etc.

Debido al alto coste económico de probar STI en situaciones reales, el uso de simuladores es realmente útil a la hora de desarrollar este tipo de sistemas. Así, en este proyecto el simulador NCTUns ha sido modificado con el objetivo de añadir nuevas posibilidades al simulador que ayuden a diseñar STI. Además, un escenario de una ciudad inteligente ha sido simulado con el objetivo de evaluar como el uso de estas redes permite la recolección y el cálculo de estadísticas en tiempo real, además de comprobar cómo funcionan los cambios realizados en el simulador.

ABSTRACT

Road safety has become a main issue for governments and car manufacturers in the last twenty years. The development of new vehicular technologies has favored companies, researchers and institutions to focus their efforts on improving road safety. During the last decades, the evolution of wireless technologies has allowed researchers to design communication systems where vehicles participate in the communication networks. Thus, the concept of Intelligent Transportation Systems (ITS) appeared. This concept is used when talking about communication technologies between vehicles and infrastructure that improve transport safety, its management, environmental performance, etc.

Due to the high economic cost of real-life tests and experimentation, the use of simulators becomes really useful when developing ITS. Nonetheless, simulators not always include all the capabilities needed to simulate these kinds of networks. Thus, in this project the NCTUns simulator is modified in order to add new capabilities that allow users simulate ITS. Furthermore, smart city scenarios are simulated in order to evaluate how the use of these networks allows real-time statistic collection and calculation, and how modifications made in NCTUns work.

1. INTRODUCTION

Road safety has become a main issue for governments and car manufacturers in the last twenty years. The development of new technologies has made that companies and institutions focus their efforts on improving road safety. New concepts, such as smart cities and living labs [15] have appeared the last years and they all have relation with the concept of car-to-car communications or car-to-infrastructure communications. To transmit information about road conditions the driver must deal with and help him/her to make more adequate decisions.

Furthermore, the concept of smart cities considers that these cities will develop an intelligent traffic control (TIC) as well as TIC infrastructures reachable at any point. To test the possibilities of these future cities, living labs (cities in which the new system can be tested in real conditions) have been created all over Europe.

In recent years, the continuous advances in wireless communications have opened new research lines. It is in this situation where the concept of mobile networks was born. The constant mobility of individuals and the need of being always connected require focus research efforts on this field. Logically, connection and transmission of information between mobile nodes is one of the main objectives. In addition, the exponential growth of the number of vehicles in cities has led that one of the main focuses in which invests time and money is road safety and transportation efficiency.

During these last decades, the evolution of communication systems has allowed car manufacturers to improve communications within the different agents participating in the whole process. Thus, new types of networks have been created in order to facilitate communication between cars, such as Vehicular Ad-Hoc Networks (VANETs) and between cars

and infrastructure, such as the Hybrid Sensor and Vehicular Networks (HSVNs) that we use in this project.

VANETs are a particular case of mobile Ad-Hoc networks, in which mobile nodes are vehicles. The concept of vehicular technology uses moving cars as nodes in a network to create a temporary mobile network. Initially it seems simple, however, it presents many challenges both theoretical and technical (routing protocols, transmission ranges and frequencies, quality of service, etc.)

The study of VANETs is expensive and complicated because of the deployment that would be necessary to test in real environments. Thus, this arise the need of performing simulation as realistic as possible in order to study, from a computer, settings and situations as similar as possible to those obtained in real environments. In this project, the real time collection and processing of traffic statistics is proposed and simulated. At first, the traffic statistics this project will treat are traffic car density and traffic delay.

This project consists of eleven chapters whose contents are detailed in the following:

- The first chapter consists on a brief introduction to road security, vehicular networks and wireless communications.
- The second chapter summarizes the objectives of this project that would be creating a smart city framework to calculate and share real-time traffic statistics and modifying the simulator code to adapt the simulation settings as realistic as possible.
- The third chapter explains previous investigations on this research area highlighting the intelligent car initiative.
- The fourth chapter consists on an introduction to mobile and vehicular networks where are explained MANETs (Mobile Ad-Hoc Network), WSNs (Wireless Sensor Network), VANETs (Vehicular Ad-Hoc Network) and HSVNs (Hybrid Sensor and Vehicular Network).
- The fifth chapter explains the network architecture for all the mobile and vehicular networks from the previous chapter.
- The sixth chapter details the smart city framework used on this project.
- The seventh chapter lists the tools used in this project, being the most important the NCTUns network simulator.
- The eighth chapter indicates the changes made on the network simulator code.
- The nineth chapter details and explains the simulation environment and scenario of a city like Barcelona.
- The tenth chapter explains the results obtained in the project from the simulations.
- Finally, the eleventh chapter analyses the conclusions made from project's results and outlines the future lines of work.

2. OBJECTIVES

In this project, a real time traffic statistic collection and processing system is proposed and simulated.

To achieve these objectives, it will be necessary to modify the NCTUns simulator in order to simulate the traffic statistic system inside a VANET topology. Therefore, in this project the open source code of NCTUns will be modified in order to add some capabilities to the program and make it more realistic.

One of the objectives of this project is to modify the NCTUns simulator in order to obtain the following results:

- Real time statistics collection about traffic density and delay, on each node (vehicle) that belongs to the VANET.
- Real time statistics processing using a low pass filter (EWMA) to give greater weight to the historical values.
- Compatibility with future changes and updates of the statistic procedures.

Nevertheless, the modification of the open source code of NCTUns is not the only objective of this project. There will be simulated a HSVN and VANET scenario to determine the performance of this networks under traffic, using the updates this project proposes. Furthermore, a new type of message inside the routing protocol will be proposed to be used in future smart cities, so it could be possible having real time statistics of the city.

3. STATE OF THE ART

In this part of the project, the state of the art of vehicular networks programs is reviewed. The focus of this section is on the European projects and consortiums promoted in recent times.

Since nineteen eighties, the European Union is making efforts to improve road safety, funding projects with this objective. Some of these projects have been focused on VANET and HSVN, and many achievements have been made since the early beginnings.

The first predecessor of VANET and HSVN networks is the project called PROMETHEUS (*PROgramme for a European Traffic with Highest Efficiency and Unprecedented Safety*, 1987-1995) supported by the EUREKA (*European Research Coordination Agency*) and carried out by the Bundeswehr University of Munich and the car manufacturer Daimler-Benz [1]. This Research and Development project was the first approach to driverless cars, equipping cars with an autopilot system capable of driving from one point to another without human help. This project evolved through the years and, at the end, it shifted its focus on giving driver information by the use of in-vehicle systems such as the intelligent copilot. In Figure 3.1 it is shown the vision of PROMETHEUS Project of safe and efficient future traffic.

Figure 3.1 - The PROMETHEUS vision of safe and efficient future traffic [5]

Another project that can be considered the beginning of the car to car networks is the project DRIVE I (*Dedicated Road Infrastructure for Vehicle Safety in Europe*, 1988- 1991) that promoted the Advanced Transport Telematics (ATT) [1]. The objective of DRIVE I was connecting cars with the road-side infrastructure. This project had a continuation (1992- 1994) in a project called DRIVE II.

These projects supposed the first effort of European society to develop a car to both, car or infrastructure (CAR-2-X) communication, but the lack of appropriate technology didn't allow these research groups to find an effective solution [1]. The development of wireless technology (as the Institute of Electrical and Electronic Engineers (IEEE) 802.11 Wi-Fi) has allowed European organizations and companies to focus their research on CAR-2-X communication, with the clear objective of improving road safety.

3.1. THE INTELLIGENT CAR INNICIATIVE

The Intelligent Car Initiative, by the European Union (EU), promotes the use of the radio communications development in recent years to improve car user's quality of life. The main objective of the initiative is to develop an intelligent vehicle pursuing smarter, safer and cleaner roads and cars.

These intelligent cars projected should assist drivers in their driving functions, providing valuable information (about the state of roads, weather conditions, traffic conditions etc.) on real time and in a short-time future, helping in the choice of the route in an energy efficiency point of view.

This initiative promotes and coordinates the global efforts of the main actors that develop the prototypes of these future cars. These main actors are car manufacturers, road operators, telecommunication companies, transport service providers and European regulators. The

initiative also funds the research of proper technology (also providing information to main actors) and does social work informing consumers of the future implantation of this type of vehicles.

The European Commission has six future challenges related with vehicular networking [2]. These challenges should be solved with the following technologies:

- **eCall**, Emergency call system that should drastically reduce the time of response of emergency services in emergency situation. This automatically calls to the emergency services in case of accident or emergency, gaining a precious time that can help saving lives.
- **Intersection assistant**, System that will use sensors and CAR-2-X communications to detect cars in the neighborhoods, warning the driver of their presence. The sensors will also communicate with traffic lights, recommending uses a proper traffic speed in order to save energy.
- **Wireless local danger warning**, Technology that uses CAR-2-X wireless communication to detect unexpected events on the road.
- **Lane change assistant and blind spot detection**, System providing warning signals to user when he switches the lane of a rail in where another car is detected.
- **Dynamic traffic management**, Technology that provides real-time information from different sources such as sensors, GPS etc. to manage different strategies to select the best way and avoid traffic jams and unexpected obstacles in the road.
- **Adaptive cruise control**, Technology that, by the use of CAR-2-X communication, adapts vehicle´s speed depending of their neighbors' vehicles speed and distance.

3.2. RELEVANT RESEARCH AND DEVELOPMENT (R&D) PROJECTS

The European commission has developed a policy for sustainable mobility. The policy defines a political framework to ensure a high level of mobility, protection of humans and environment, technological innovation, and international cooperation. It is also defined the ambitious goal of reducing road fatalities by 50% by 2010. Hence the European Union funds projects with this idea through its Framework Programs (FP).

The major results of these framework programs have been achieved by projects funded by the 6th Framework Program (6FP) (such as GST, PReVENT, FleetNet or NOW) and in the 7th Framework Program (7FP) (such as CIVIS, SAFESPOT or COOPERS).

FleetNet: Internet on the Road (2000-2003)

Project developed by Daimler and funded by the German Ministry of Education and Research that studied the real possibilities of communication between cars. The project worked in the investigation of different radio technologies (in particular IEEE 802.11, data transmission by radar communications, etc.) [3] After an initial period of research, technology selected to

develop a prototype of car to car communication platform was IEEE 802.11. Due to the limited communication range of this type of radio communication the research was focused on multi-hop communication (position based to reduce difficulties in terms of scalability). In this system each node selects the next hop for a data packet reducing geographical distance to destination. These simulations concluded that position-based routing protocols improved topology-based routing protocols, such as Dynamic Source Routing (DSR), in highways and in terms of packet delivery ratio when multi-hop communication was required.

This project developed a Linux-based protocol router, using IEEE 802.11a and a planar antenna. The router used Global Positioning System (GPS) through an Ethernet connection to get GPS information [3].

Real simulations (with real cars and highways) of the project finished with promising results.

NOW: Network on Wheels (2004-2008)

The successor of FleetNet was born by the same actors with the idea of developing an open communication platform for road safety, traffic efficiency and infotainment applications, as well as analyzing strategies to introduce this car to car communication into European market (creating a European standard). This project mixes at same time traffic efficiency and safety, with applications of information and entertainment. This adds great difficulties to the project due to different types of requirements of these applications. Thus, NOW distinguished between two types of messages: the first type of them providing periodic information to users (such as, beacons or heartbeats) and the other type providing non-periodic information (such as car accidents or unexpected situations in the road).

The network architecture of the project used two different protocols; an ad hoc car to car protocol providing information for road safety applications and a traditional Internet Protocol (IP) for infotainment applications. The network architecture used IEEE 802.11p radio interface for road safety and an IEEE 802.11a/b/g radio interface for infotainment applications [4].

This project created a Linux-based communication system prototype of a CAR-2-X communication platform.

PreVENT (2004-2008)

PReVENT is a Research and Development integrated project funded by the European Commission to increase road safety. This project is divided into 13 subprojects, investigating each one different fields of road safety, affording the problem from different sides. Two of these projects included Vehicular Networking, WILLWARN and INTERSAFE [5].

WILLWARN uses different sensors to notify users of unexpected situations on the route. The scenarios covered by this project are four, as shown in Fig. 3.2.

1) Detection and warning of obstacles on the road, warning if one's own car is an obstacle for others.

2) Warning of emergency vehicles or slow vehicles.
3) Detection of reduced friction or reduced visibility through bad weather.
4) Warning of dangerous spots such as construction zones through electronic beacons

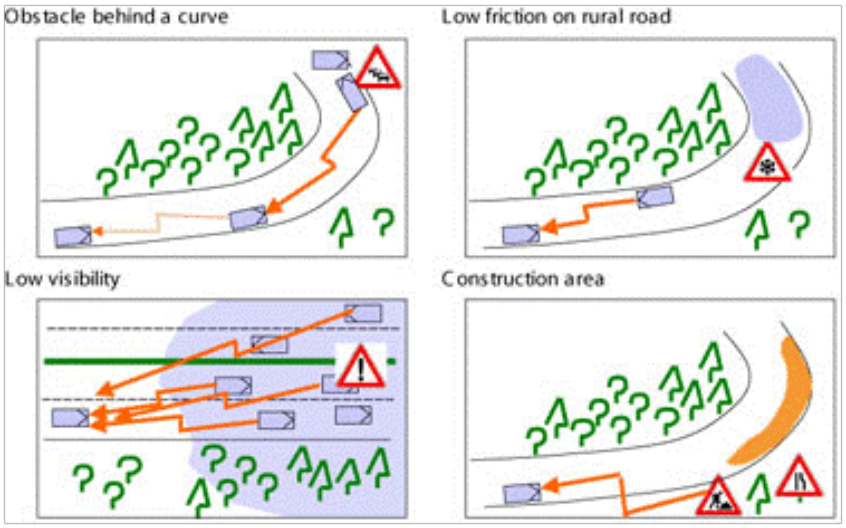

Figure 3.2 - Scenarios covered by the WILLWARN project [5]

To send their messages it uses the same wireless technology of NOW.

INTERSAFE subproject combined sensor and communication technologies to increase safety at intersections. This project developed an intersection driver warning system, providing warnings to driver based on precise localization (using on-board sensors), detection, classification and path prediction of all other objects at the intersection, communication with traffic lights and high-level map information. In Figure 3.3 it is shown an INERSAFE scenario.

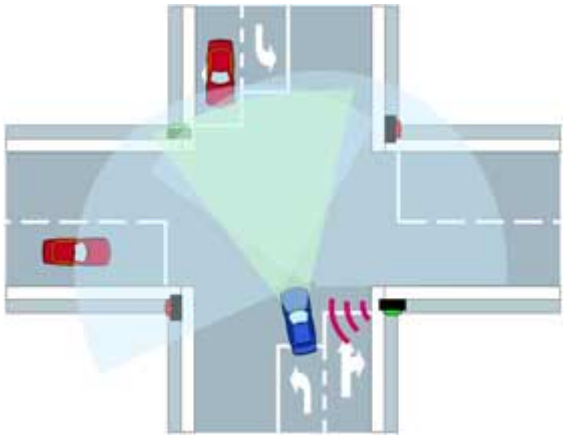

Figure 3.3 - INTERSAFE Simulation Scenario [5]

CARLINK (2006-2008)

Project developed by the University of Málaga and funded by the Spanish government and the European commission, with the main objective of developing an intelligent wireless traffic service platform between cars and supported by wireless transceivers beside the road.

The first objective of this project was not car to car communication, to avoid unexpected events, but real-time local weather data, urban transport traffic management through wireless communication between cars and local data base stations. To transmit this information, cars could communicate to each other as members of an ad-hoc network [6].

The radio communications systems being tested on this project were evolutionary extensions of Wireless Local Area Networks (WLANS), Worldwide Interoperability for Microwave Access (WiMAX) or mobile communications such as General Packet Radio Service (GPRS). These different systems were tested on different weather conditions and in different topologies.

SEVECOM (2006-2008)

Research and Development project of the European Commission focused on security and privacy aspects of CAR-2-X communication systems [7].

The difference between this project and the listed before is that this project made an effort in security architecture for CAR-2-X communication systems. In a second action, the project developed solutions for in-vehicle intrusion detection, malfunction detection and data consistency and secure positioning [7].

COMeSafety (2006-2009)

Communications for eSafety (COMeSafety) was born as a support platform for all European Commission road-safety stakeholders [8]. This platform provides coordination and consolidates results from the biggest R&D projects that we have talked before, as shown in Fig. 3.4.

Figure 3.4 - Communication between COMeSafety and R&D projects [8]

SAFESPOT (2006-2010)

Project funded by the European Commission combining car to car and car-to-infrastructure communications. This project develops a Safety Assistant that detects potentially risky situations in advance, providing information about surrounding environment in space and time, as shown in Fig. 3.5

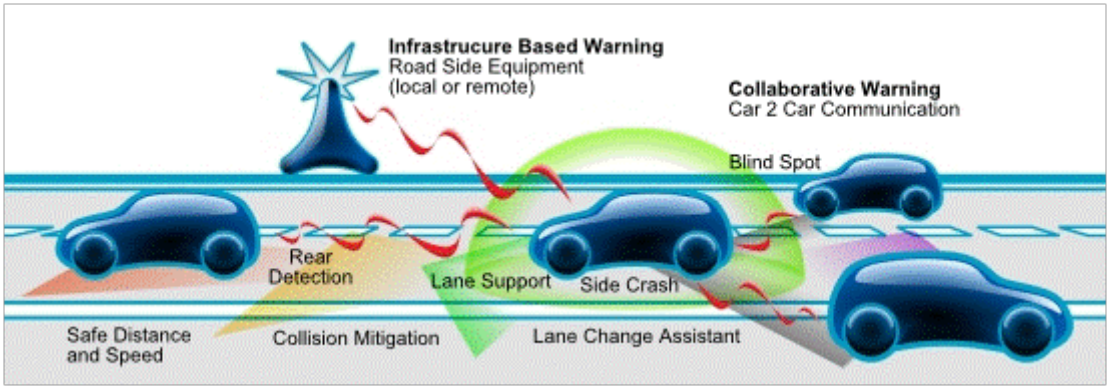

Figure 3.5 - Car communicating with both, car and infrastructure by SAFESPOT [9]

The project is based mainly on IEEE 802.11 wireless technology and the starting point is the NOW project. Different challenges of the project are to cooperate with the project called Co-operative Vehicle-Infrastructure Systems (CVIS) in the research of a European standard from the International Organization for Standardization (ISO) for CAR-2-X communications and to develop a reliable relative positioning system to improve CAR-2-X communications (considering systems such as GPS or Galileo) [9].

SAFESPOT has developed the concept of Local Dynamic Map (LDM) with the idea of share static and dynamic information. The LMD is a multilayered dynamic representation of the vehicle and everything that surrounds it, collecting information thanks to their sensors and providing users of unexpected events to react safely.

COOPERS (2006-2011)

Integrated R&D project funded by the European Commission. The project is focused on traffic management (infrastructure status, traffic jams) through wireless communication between cars, and between cars and infrastructures, as shown on Fig. 3.6. [10]

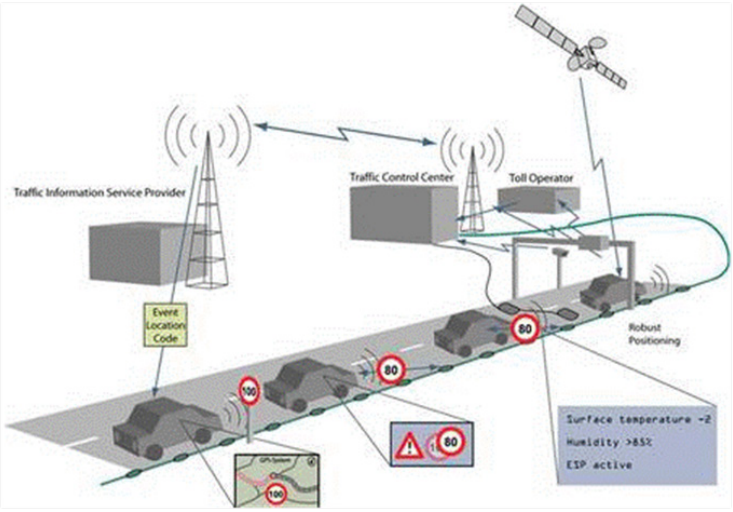

Figure 3.6 - Communication CAR-2-X developed by COOPERS [10]

The project is divided in three main blocks:

- The first block was focused on improving road sensor infrastructures.
- The second block is focused on developing the communication infrastructure to improve their reliability, robustness and real-time capability. The communication systems used with this propose are Digital Audio Broadcast (DAB), GPRS, Universal Mobile Telecommunications System (UMTS), WLAN and infrared and microwave technologies.
- The third part of the project (still not performed) will test the results of this project in main European highways.

CVIS (2006-2011)

The project called Co-operative Vehicle-Infrastructure Systems (CVIS) is an R&D project funded by the European Commission. The main objectives of this project are increasing road safety and traffic efficiency giving four new services to users. These systems are COMM (that allows CAR-2-X communication), POMA (providing positioning systems), COMO (providing monitoring services) and FOAM (linking vehicles with infrastructure).

CVIS considers a wide range of communications technologies such as cellular technologies (GPRS or UMTS), infrared, and wireless technologies [11]. The final results of this project are still not available.

3.3. EUROPEAN PROJECTS IN COLLABORATION WITH CAR MANUFACTURERS

CAR-2-CAR Communication Consortium

The Car to car Communication Consortium (C2C-C) is a consortium of major European vehicle manufacturers with the objective of creating a European standard for CAR-2-X communication. This standard is supposed to be based on Wireless communication technology, and it looks for an improvement of road safety and traffic efficiency. This consortium is divided in 6 groups:

WG PHY/MAC, focused on the Physical (PHY) and Media Access Control (MAC) protocol layers, WG NET, focused on network and transport protocols, WG ARCH (defines the architecture protocols for CAR-2-X communication), WG APP (identifies application requirements and protocols), WG SEC (works in the protocols security) and WG STA (groups that works with the standardization issues) [12].

The main car manufacturers that take part on this project are Audi, BMW Group, Daimler, Fiat, Honda, Opel, Renault, Volkswagen and VOLVO.

In Figure 3.7 it is shown a simulation of car to car communication on C2C-C.

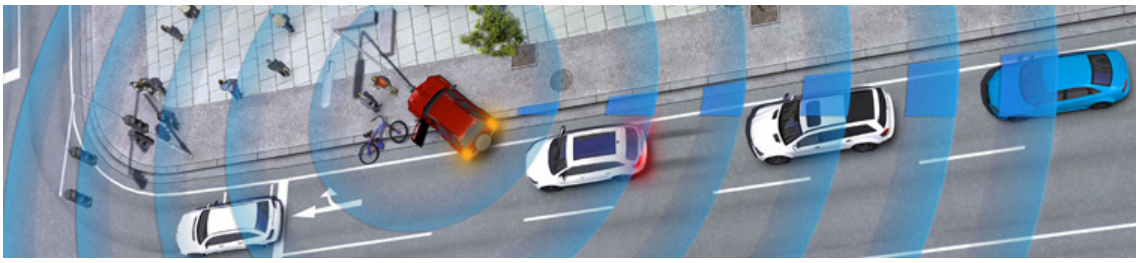

Figure 3.7 - Simulation of car to car communication on C2C-C [12]

EsafetyAware

Is a consortium that seeks to accelerate the market introduction of life-saving technologies by organizing information campaigns and dedicated events aimed at creating awareness of eSafety benefits among policy-makers and end-users. [13]

In this consortium there are car manufacturers such as Hyundai-Kia Motors or Toyota, tire manufacturers as Bridgestone or Continental and governments and User's associations such as the European Commission or the Fédération Internationale de l'Automobile (FIA).

Some of the systems that have been developed and promoted by eSafetyAware are the Blind Spot Monitoring (that indicates you if there is a car in the lane next to you, and that you are not seeing it), the Speed alert (indicates you if you exceed the speed limit) or the Adaptive Headlights (that optimizes the illumination of the road).

Living Labs

A Living Lab is a real-life test and experimentation environment where users and producers co-create innovations. The European Commission characterizes the Living Labs as Public-Private-People Partnerships (PPPP) for user-driven open innovation. Thus, a living Lab employs four main activities [14].

1) Co-Creation: co-design by users and producers
2) Exploration: discovering emerging usages, behaviors and market opportunities
3) Experimentation: implementing live scenarios within communities of users
4) Evaluation: assessment of concepts, products and services according to socio-ergonomic, socio-cognitive and socio-economic criteria.

With this intention, many European and worldwide cities are currently being used as Living labs. In the next section the main Living Lab associations are described, as well as an example of a Living Lab in the city of Barcelona.

ENoLL (2006-2011)

The European Network of Living Labs (ENoLL) is the international federation of benchmarked Living Labs in Europe and worldwide. Founded in November 2006 under the auspices of the Finnish European Presidency, the network has grown in 'waves' up to this day. To this date, 5 Waves have been launched; resulting in 274 accepted Living Labs. The ENoLL international

non-profit association, as the legal representative entity of the network, is headquartered in Brussels, at the heart of Europe [14].

There are three different kinds of memberships within ENoLL:

- Adherent members: Organizations representing a living lab. These members are included in the ENoLL communication channels, and have the right to be present and participate in the ENoLL activities.
- Associated members: Organizations which organizations which are involved in the object and activities of the association, but who are not selected according to the ENoLL selections process. These members pay the annual membership fee.
- Effective members: Adherent members can choose to become Effective members, and have a vote in the organization and strategic directions of the ENoLL association. Only approved Living Labs can become effective members.

Nowadays there are 21 effective members within the European Union, 4 of them in Spain (Andalusia, Catalonia, Valencia and Basque Country).

In Figure 3.8 there are shown the different waves of creation of Living Labs in Europe.

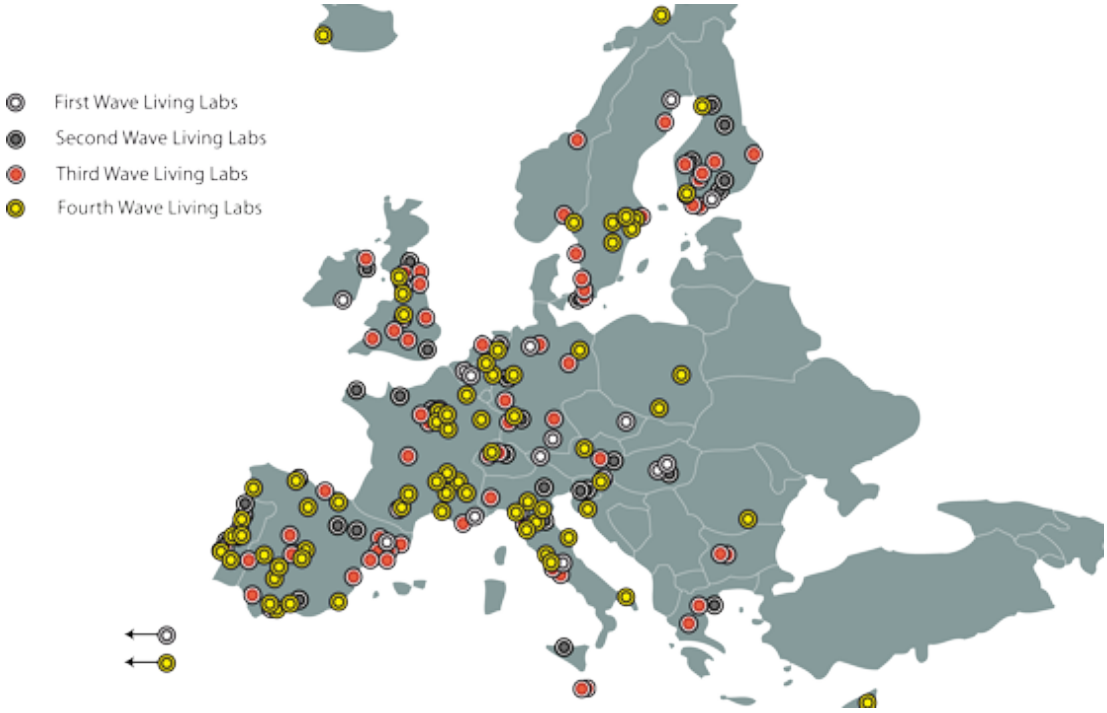

Figure 3.8 - Living Labs in Europe [14]

Living Labs Global (2003-2011)

Living Labs Global is a non-profit association based in Copenhagen (Denmark) with the objective to promote innovation in services and mobility in cities. It says that mobility is a paradigm shift in which the user, as a citizen, professional or visitor, is expecting public and private services to be tailored to his needs, delivered on demand, anywhere. At the same time,

the global market for innovative services in cities is obscured by a lack of knowledge about other experiences, technologies and business ideas.

In 2003, Living Labs Global began as an idea to provide cities with a practical tool for service innovation to address urban challenges in fields such as transport, healthcare, sustainability, social services and tourism. Since then, Living Labs Global has emerged as an independent, sustainable global initiative receiving strong support from the cities, technology institutes, associations and innovative small and medium-sized enterprises that comprise its membership of more than 160 organizations, as well as the European Commission [15].

Three different membership and partnership exists in the association:

- Showcase Membership: Organizations that provide innovative services and are accepted into the Living Labs Global Showcase receive a free annual Showcase Membership.
- Strategic Membership: Cities and organizations representing cities apply for Strategic Membership and are evaluated against the guidelines of the association. Two existing Strategic Members must endorse the application to proceed with Membership.
- Partnership: Living Labs Global maintains partnerships with other organizations via written Strategic Partnership agreements, regulating joint objectives and adding value to members. Any organization can apply for partnership, and opportunities will be evaluated on a case by case basis.

22@ Living Labs

It is a platform composed by companies and public and private institutions that origins and manage innovative processes of services and mobile technologies that involved companies are interested in developing. This project is led by 22@Barcelona in collaboration with the Barcelona Digital Foundation and it belongs to the European Network of Living Labs (ENoLL).

Also, there are several projects around the world researching about car-to-car communications and services for user's comfort. For example:

In Japan, a standard for vehicle-to-infrastructure communication was published in 2001 and denoted as 'Dedicated Short-Range Communication System' (ARIB 2001). The specified system operates in the 5.8 GHz frequency band, is based on time division multiple access and targets a range of about 30 m. The primary use of the system was seen in electronic toll collection but the system was generalized to support various other services. In 2008, more than 20 million on-board units for electronic toll collection were deployed in Japan. Based on the success on this 5.8 GHz DSRC system and on infrared- based vehicle-to-infrastructure communication, various ITS projects and activities are currently joining forces to demonstrate and enhance vehicle-to-infrastructure and vehicle-to-vehicle communication under the umbrella of Japan's national ITS Safety 2010 initiative. [16]

In the USA, there are several industry/government projects on-going. Vehicle Infrastructure Integration (VII), which has been rebranded as IntelliDrive [17], has recently completed a large

proof of concept demonstration. The majority of this testing environment was implemented in Detroit. This system comprised 55 Road Side Equipment (RSE) stations within 45 square miles and employed 27 vehicles. Seven applications were developed and tested:

- In-Vehicle Signage: RSEs trigger displays of advisory messages within the vehicle.
- Probe Data Collection: Vehicles provide historical data on their location/state and share with the RSE, which is then centrally compiled and analysed.
- Electronic Payments – Tolling.
- Electronic Payments – Parking.
- Traveller Information / Off-Board Navigation.
- Heartbeat: RSEs collect periodic status messages from vehicles including vehicle speed and location.
- Traffic Signal Indication: Broadcasts traffic light state.

Integrated Vehicle-Based Safety Systems (IVBSS) project [18] explores human-machine interface issues when several safety applications, with potentially overlapping or contradictory advisories, are operated simultaneously. The Cooperative Intersection Collision Avoidance System (CICAS) project [19], had three components: a Violation Warning project (demonstrated in Michigan), a Stop Sign Assist project (demonstrated in Minnesota), and a Signalized Left Turn Assist project (demonstrated in California).

4. INTRODUCTION TO MOBILE AND VEHICULAR NETWORKS

Over recent years, the considerable mobile services sector growth around the world was certainly the major phenomenon in telecommunications field. Wireless technology is capable of reaching virtually every location on the surface of the Earth. With such success of mobile communication demand it is hardly surprising that wireless technology has led to the development of new multimedia services and to evolution in user requirements in terms of throughput and universal mobility throughout different systems. Mobile communications are already applied to the realm of personal and business computing, making that people living habits and working ways evolve. [20]

Generally, there are two distinct approaches for enabling wireless mobile units to communicate each other:

Infrastructure or Centralize Network: Wireless mobile networks have traditionally been based on the cellular concept, where all devices are connected to a central node which is the acting agent for all communications, and relied on good infrastructure support. Typical examples of this kind of wireless networks are GSM, UMTS, WLAN, etc.

Infrastructureless Network: This mobile wireless network is commonly known as a mobile Ad-Hoc network or MANET. A MANET is a collection of wireless nodes that can dynamically be set up anywhere and anytime without using a pre-existing network infrastructure. It is an autonomous system in which mobile host connected by wireless link are free to move randomly and often act as routers at the same time.

The design of network protocols for MANETs is a complex and wide area of research with many challenges. Hence, during the last few years, mobile Ad-Hoc networks have become a very popular field of study within the research community.

4.1. INTRODUCING MANETs (MOBILE AD-HOC NETWORKS)

A MANET is a collection of wireless devices that can dynamically form a network with a very simple deployment capability, paving the way for new applications which have not been able to emerge until now and offers solutions in multiple environments that have no infrastructure. These devices or nodes can move in a random way and are capable of self-organize themselves arbitrarily, collaborating in order to communications succeed. Examples of devices are laptops, PDAs, mobile phones, handhelds and wearable computers. The most outstanding features of MANETs are detailed below:

Dynamic Network Topology

It is undoubtedly the characterizing element in MANETs. Since the nodes are mobile, the network topology may change rapidly and unpredictably and the connectivity among the terminal may vary with time. MANETs should adapt to the traffic and propagation conditions as well as the mobility patterns of the mobile networks nodes.

Autonomous Terminals and Self-Organization

In MANETs, each mobile terminal is an autonomous node, which may function as both, a host or a router, and are responsible for dynamically discovering other nodes to communicate or handle the network configuration e.g. addressing and position location issues.

Distributed Operation

Since there is no background network for the central control of the networks operation, the control management of the network is distributed among the devices. The nodes involved in a MANET should collaborate to each other and act as a relay as needed, to implement routing and security functions.

Multi-Hop Routing

As delivering data packets from a source to its destination out of the direct wireless transmission range, the packets should be forwarded via one or more intermediate nodes.

Fluctuating Link-Capacity

The nature of high bit error rates of wireless connections might be more critical in a MANET. The radio transmission rate is vulnerable to noise, fading, multiple access and interference conditions, and has less bandwidth than wired networks.

Light Weight Terminal

In most cases, the MANET nodes are mobile devices with limited processing capacity, small memory size and low power storage. Such devices need optimized algorithms to execute computing and communication functions.

Scalability

Sometimes the number of devices which set up the network can increase until dozens or hundreds. Since there isn't a central element which is in charge of network management, adding or rejecting nodes into the topology is a simple process.

4.1.1. CHALLENGES IN MANETS

Regardless of MANETs capabilities, possibilities and characteristics have risen a quickly spreading, that benefits lead to several changes as well, which must be studied carefully. Researching in the area of mobile Ah-Hoc networking is receiving more attention from academia, industry and government during the last few years. Almost every aspect of the network has been explored in some level of detail although no ultimate resolution to any of the problems is found yet and there are already many open issues for research and significant contributions.

Scalability Weakness

The numbers of network nodes can be large and finding the route to a destination also requires frequent exchange of routing control information among the nodes. Thus, the amount of the update traffic can be substantial, and it is even higher when nodes with increased mobility are present.

Routing

Routing in Ad-Hoc networks, which is quite different from traditional IP routing, is a particularly complex problem because of many factors including topology, selection of routers, locations of request initiator, resource limitations and unreliability of wireless links. A node at least need to know the reachability information to its neighbors for determinate a packet route, while the network topology can change quite often in a MANET. Thus, routes may change while in use and become no longer valid in a very short time. [21]

Since the arrival of Ad-Hoc network concepts, many proposals have been studied, simulated and evaluated. The same proposals have led to variations, specializations to given environments and optimizations. Ad-Hoc routing proposals can be classified in two main categories: proactive and reactive routing. Proactive or table-driven protocols are directly inspired by routing protocols deployed in the Internet and consist in maintaining a routing table for sending data to any node in the network. Instead, Ad-Hoc reactive routing algorithms research the vital information of a route between two nodes when a request for this route is

expressed by the higher protocols layers. The protocol of this class attempt to keep the routes used and only those as up to date as possible in order to minimize the use of control messages to a minimum to save bandwidth. We can add other generally hybrid proposals to these two families, which include both features.

Security

Research on security in addition to routing challenges has become a primary concern to mobile Ad-Hoc networks. Historically, network security has adopted a centralized, largely protective paradigm to satisfy aforementioned requirements. This is effective because the privileges of every node in the network are managed by dedicated machines, e.g. authentication servers. Membership in such networks allows individual nodes to operate in an open fashion because its simplicity guarantee that any malicious user from outside world won't be allow to access. Although these solutions have been considered very early in the evolution of Ad-Hoc networks, attempts to adapt similar client-server solutions to a decentralized environment have largely been ineffective.

Attempts to secure Ad-Hoc networks must remain being Ad-Hoc solutions: they must establish security without reference to any centralized node. Instead, security paradigms should be carried out by the cooperation of all available nodes in the network.

An implementation of an inefficient authentication protocol in Ad-Hoc networks may lead to a vulnerability increase and network will be compromised. Attacks from malicious nodes could range from message replay, passive eavesdropping to injecting erroneous messages or liable information into routing tables in order to make network congestion and denials of service by forwarding traffic to a black hole. Hence, security solutions need to consider malicious attacks not only from outside but also from within the network. So, key management and authentication procedure are issues that must be carefully considered.

Quality of Service (QoS)

As a wired network, the flows generated by applications supported by mobile Ad-Hoc networks have diverse characteristics such as type and volume of exchanged information, duration or interaction to name a few examples. These flows also have different QoS requirements: bandwidth requirements for video on demand or end-to-end requirements for voice over IP services. That is why uniform packet processing is not appropriate, and QoS support which considers the different QoS requirements is vital.

In MANETs the dynamic networks environment with continuous topology changes and the limited resources raise that problem of QoS support at different levels. [21]

Energy-Constrained Operations

Some or all the nodes in an Ad-Hoc network may rely on batteries or other exhaustible means for their energy. Therefore, energy conservative networks are becoming extremely popular within the Ad-Hoc network research.

The goals can be achieved either by developing better batteries, or by focusing on the devices networks interface, which is often the single largest consumer of power.

Energy efficiency at the network interface can be improved by developing transmission/reception technologies on the physical layer, but especially with specific networking algorithms. Nevertheless, energy conservation is currently being addressed in every layer of the protocol stack.

Much research has been carried out yet, however, there are still much more work to do be done. [21]

Interoperation

The self-organization of Ad-Hoc networks is a challenge when two independently formed networks come physically closed to each other. This is an unexplored research topic that has implications on all levels on the system design. The issue of joining two networks is not trivial: the networks may be using different synchronization, or even different MAC, routing or security protocol.

Another important issue comes into picture when we talk about all wireless networks. One of the most important aims of recent research on all wireless networks is to provide seamless integration of all types of networks. The issue raises questions on how the Ad-Hoc network could be designed so that they are compatible with, for instance, wireless LANs, 3G and 4G cellular networks. [21]

4.1.2. APPLICATIONS OF MANETS

With the increase of portable devices as well as progress in wireless communications, Ad-Hoc networking is gaining importance with the number of widespread applications. Ad-Hoc networking can be applied anywhere where there is little or no communication infrastructure or the existing infrastructure is expensive or inconvenient to use. Ad-Hoc networking allows the devices to maintain connections to the network as well as easily adding and removing devices to and from the network. The set of applications for MANETs is diverse, some well-known are: [21]

Community Network

For some business scenarios, the need for collaborative computing might be important outside office environments than inside a building. After all, it is the case where people do need to have outside meetings to cooperate and exchange information on a given project. It is also an interesting solution for neighborhood scenarios, stadiums, museums or airports.

Crisis-Management Applications

That includes emergency or rescue operations, as a result of natural disasters where the entire communications infrastructure is in disarray or inoperative (Tsunamis, hurricanes). Restoring communications quickly is essential. By using Ad-Hoc networks, an infrastructure could be set up in hours instead of days or weeks that a wire-line communication would require.

Personal Area Networking

A personal area network (PAN) is a short-range, localized network where nodes are usually associated with a given person. Bluetooth is an example of a technology aimed at supporting PANs by eliminating the need of wires between devices such as printers, cell phones and PDAs or laptop computers so on.

Military Battlefield Applications

MANETs were created for military purposes. Ad-Hoc networking would allow the military to take advantage of commonplace network technology to maintain an information network between soldiers, vehicles and military information headquarters.

Besides of these applications, two fields of study have become very interesting within the research community: wireless sensor networks (WSN) and vehicular Ad-Hoc networks (VANETs). The special features and advantages of both of them are detailed below.

4.2. INTRODUCING WIRELESS SENSOR NETWORKS (WSN)

In recent years, advances in wireless networking, micro-fabrication and integration embedded microprocessors have enabled a new technological vision possible: wireless sensor networks. WSN consist of a large number of sensor nodes which collects data and interoperate each other to carry out functions involving some kind of tracking, monitoring or controlling.

WSNs, which are considered as a special case of a MANET with reduced or no mobility, are expected to find increasing deployment in coming years, as they enable reliable monitoring and analysis of unknown and untested environments.

Development of wireless sensor networks was motivated by military applications such as battlefield surveillance and is now used in a variety of physical phenomena of interest: monitoring pedestrian or vehicular traffic in human-aware environments, report wildlife habitat conditions for environmental conservation, detect forest fires to aid rapid emergency response and monitoring industrial process of healthcare applications. [22]

In Figure 4.1 it is shown the typical WSN architecture.

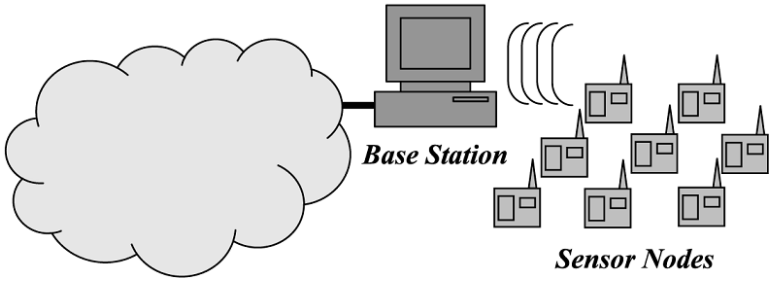

Figure 4.1 - WSN Architecture

A sensor node is basically a device that converts a sensed attribute (such as temperature of vibrations) into a form understandable by the users. It consist of a transducer to sense a given physically quantity with a predefined precision (sensing unit), an embedded processor for local processing, small memory unit for storage (processing and storage unit), a wireless transceiver to transmit or receive data (transceiver unit) and of course batteries to power the circuit (power unit), as shown in Fig. 4.2. [23]

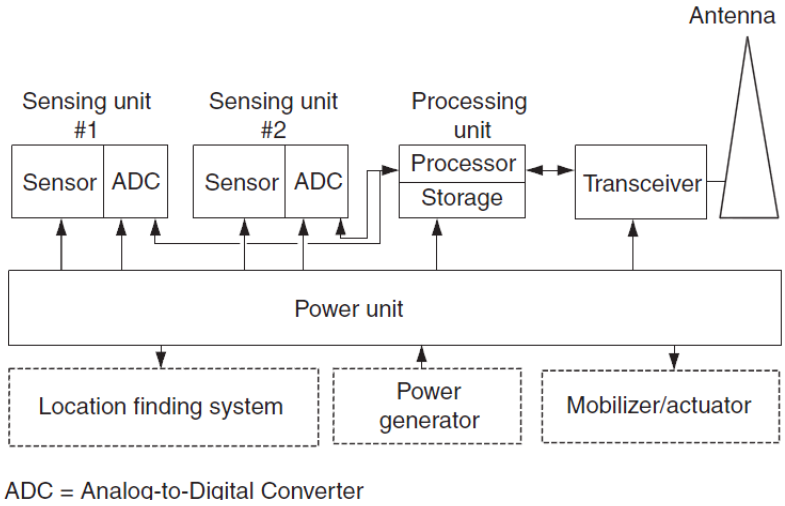

Figure 4.2 - Main Components in a WSN Node [23]

Once sensors have gathered and processed data, they have to transmit it to other nodes of the network. These data travels through the network using wireless links and multi hop routing, to a sink point (destination device). Usually, sensor nodes are organized in clusters to reduce power consumption. Energy used to transmit a message from a sensor to the clustering node is lower than the one that would be necessary to transmit the same message from the sensor nodes to the base station in a single hop.

Thus, most common WSN use the sequence presented in Fig. 4.3 to process data obtained by its sensor.

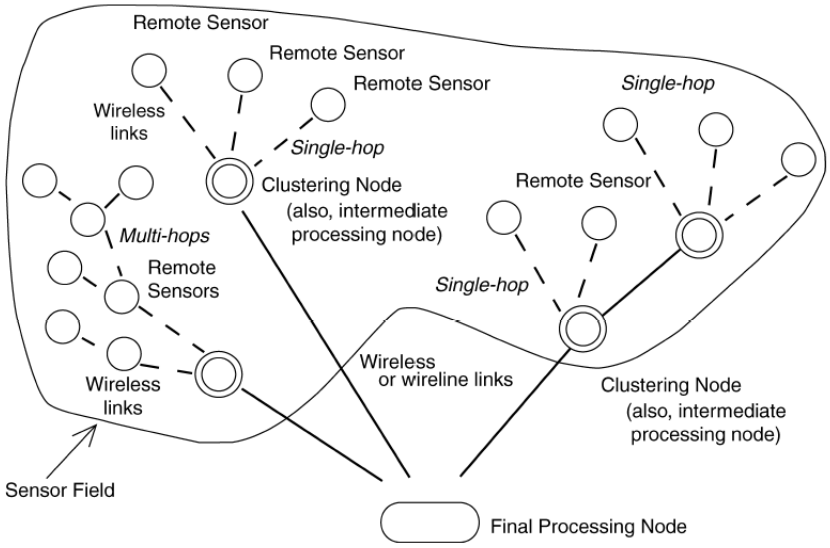

Figure 4.3 - WSN's Data Processing Sequence [23]

- The sensor node obtains data from its environment, perform some processing and send the data to a central point of information clustering.
- Clustering nodes receive, carry out further processing and forward the data to the next clustering node (multi hop) or to the base station.
- The final processing node, also called base station, receives and processes the information sent through the sensor network.
- The final processing node is the key node of WSN. This node must have enough memory, power and computational capacity to correctly perform the reception and processing of all data. It is usually responsible of connecting the sensor network with other networks, informing an administrator about the data collected.

4.2.1. WSN PROPERTIES

The advancement in technology has made it possible to have a network of hundreds or even thousands of extremely small devices equipped with programmable computing and sensing and wireless communications capability, enhancing the reliability if the coverage area. Some of the features and challenges of WSN are as follows:

Easy of Deployment

These wireless sensors can be deployed (dropped from a plane or placed in a factory) at the site of interest without any prior organization, thus reducing the installation cost and time, and also increasing the flexibility of deployment.

Fault Tolerant

With macro-sensors, the failure of the node makes that area completely unmonitored until it is replaced. With wireless sensors, failure of one node doesn't affect the networks operation

substantially as there are other adjacent nodes collecting similar data. At most, the accuracy of data collected may be somewhat reduced.

Networking and Security Implementation

Sensor nodes have limited computing power and therefore may not be able to run sophisticated network protocols or authentications and encryption algorithms contrary to Ad-Hoc networks, leading to light weighted as simple versions of routing protocols and security implementations. Besides, two operational modes or states are defined for each node, awakened mode and slept mode, if the node must be active or not active, so the protocols and algorithm implementations must take into account this limitation too.

Data Centric

In traditional networks, data is requested from a specific node. WSN are data centric: data is requested based on certain attributes. An attribute-based address is composed of a set of attribute-value pair query. For instance, If the query is something like temperature > 35˚, then only those device sensing temperature greater than 35˚ need to respond and report their readings and other sensor nodes remain in the slept trance. Once an event of interest is detected, the system should be able to configure itself so as to obtain high quality results. [21]

Mobility

Since the wireless sensors are equipped with battery, they can possess limited mobility. Thus, if a region becomes unmonitored we can have the nodes rearrange themselves.

Energy Conservation

Sensor nodes can use up their energy supply carrying out computations and transmitting information in a wireless environment. As such, energy-conserving forms of communication and computation are crucial as the node lifetime shows a strong dependence on the battery life.

4.2.2. WSN STANDARDS

A suite of protocols and standards are needed at the physical, link, network and transport layers in WSN.

In the case of WSN the standard commonly used is ZigBee/IEEE 802.15.4 [24]. This standard defines the physical layer (PHY) and the Medium Access Control (MAC) sub-layer for Wireless Personal Area Networks (WPAN) focusing on low-cost, low-speed ubiquitous communication between low-rate devices, the Low Rate Wireless Personal Area Networks (LRWPAN). The emphasis is on very low cost communication of nearby devices such as sensors with little to no underlying infrastructure, intending to exploit this to lower power consumption even more.

The basic framework of this standard conceives a 10-meter communications range with a transfer rate of 250 kbps. Tradeoffs are possible to favor more radically embedded devices with even lower power requirements, through the definition of not one, but several physical layers. Lower transfer rates of 20 and 40 kbps were initially defined, with the 100 kbps rate being added in the current revision.

Even lower rates can be considered with the resulting effect on power consumption (really important in WSN). As already mentioned, the main identifying feature of 802.15.4 among WPANs is the importance of achieving extremely low manufacturing and operation costs and technological simplicity, without sacrificing flexibility or generality.

Important features include real-time suitability by reservation of guaranteed time slots, collision avoidance through Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) and integrated support for secure communications. Devices also include power management functions such as link quality and energy detection.

4.2.3. WSN CURRENT DEPLOYMENTS AND APPLICATIONS

Judging by the interest shown by military, academia and the media, dozen applications do exist for sensor networks such as weather monitoring, security and tactical surveillance, detecting ambient conditions or domotic and healthy applications. A brief description of some of them is presented below:

Remote Ecological Micro-Sensor Network

PODS [25] is a research project undertaken at the University of Hawaii that has built a wireless network of environmental sensors to investigate species of plants will grow in one area. They deployed camouflaged sensor nodes in the Hawaii Volcanoes National Park, where two types of sensor data are collected: weather data are collected every ten minutes and image data are collected once per hour. Users employ the Internet to access the data from a server in the University of Hawaii at Manoa. Figure 4.4 shows a diagram for this project.

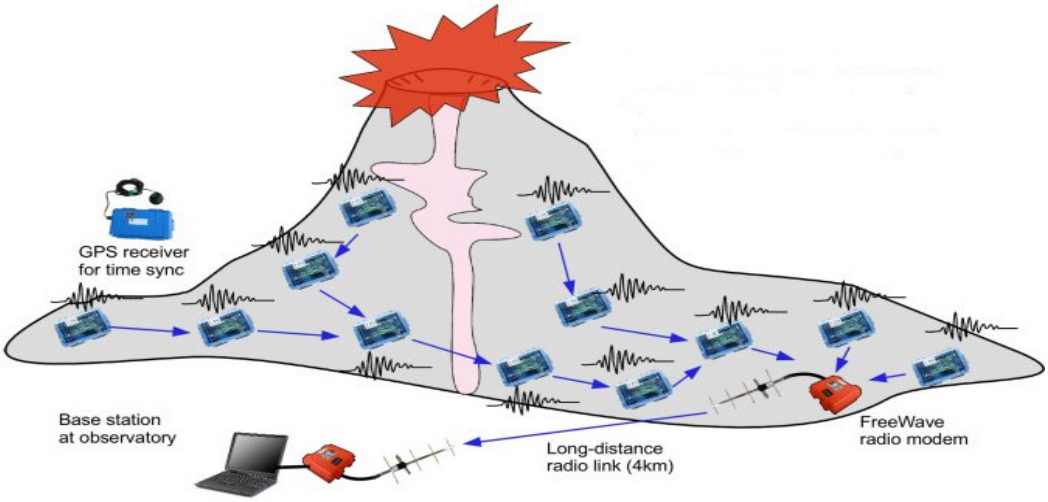

Figure 4.4 - Monitoring Volcanic Eruptions with WSN [26]

Environment Observation and Forecasting System

The Environment Observation and Forecasting System (EOFS) is a distributed system that spans large geographic areas and monitors, models and forecast physical processes such as environmental pollution or flooding. CORIE [27] is a prototype of EOFS for the Columbia River (Oregon, USA) which integrates a real-time sensor network, a data management system and advanced numerical models. Approximately thirteen stationary sensor nodes fixed to a pier are deployed across the Columbia River estuary, while on mobile sensors station drifts off-shore. The stationary are powered by a power grid, while the mobile station uses solar panel to harness solar energy. Sensor data are transmitted via wireless links towards on-shore master stations which, in turns, forward the data to a centralized server where it serves as input to a computationally physical environment model used to guide forecasting.

Disaster Relief Management

Novel sensor network architecture has been proposed in [28] that could be useful for major disasters including earthquakes, storms, floods, fires and terrorist attacks. The sensor nodes are deployed randomly at homes, offices, and other places prior to the disaster and data collecting nodes communicate with database server for a given sub are which are linked to a central database for continuous update. Based on the statistical data from Izmit earthquake in 1999, various performance curves are obtained to indicate required average number of active sensor nodes to detect a disaster, probability of disaster to be within the sensing range, total number of sensor nodes failed due to energy depletion.

Health Care Monitoring

An example of such application is the artificial retina developed within the Smart Sensors and Integrated Microsystems (SSIM) project [29], where a retina prosthesis chip consisting of one hundred micro sensors are built and implanted within the human eye allowing patients with

no vision or limited vision to see at an acceptable level. Wireless communication is required to suit the need for feedback control, image identification and validation.

DARPA Efforts towards Wireless Sensor Networks

The Defense Advanced Research Projects Agency (DARPA) has identified networked micro sensors technology as a key application for the future. There are many interesting projects and experiments going under the DARPA SensIT (Sensor Information Technology) program which aims to develop the software for distributed micro-sensors.

Vehicle type identification is important for defense applications and an experiment was performed for two weeks by placing sensors boards in the Marine Corps Air Ground Combat Centre in Twenty-nine Palms (California) for collecting acoustic data [30]. To detect presence of a vehicle, the sensor board is equipped with acoustic, seismic and passive Infra-Red sensors under program control and local processing is done to do local classification and storage.

4.3. INTRODUCING VANETS (VEHICULAR AD-HOC NETWORKS)

Traditional traffic management systems are based on centralized infrastructures where cameras and sensors implemented along the road collect information on density and traffic state and transmit this data to a central unit to process it and make appropriate decisions. This type of system is very costly in terms of deployment and is characterized by a long reaction time for processing and information transfer in a context where information transmission delay is vital and is extremely important this type of system. However, with the rapid development of wireless communication technologies a new decentralized architecture based on vehicle-to-vehicle communications (V2V) has created a very real interest these last few years for car manufactures, the R&D community and telecom operators. Thus, a new concept was born: a vehicular Ad-Hoc network (VANET), which is more than a specific application of traditional mobile Ad-Hoc network (MANET).

VANETs have recently emerged as a platform to support intelligent inter-vehicle communication to improve traffic safety. The road-constrained characteristics of these networks and the high mobility of the vehicles, their unbounded power source, and the emergence of the roadside wireless infrastructures make vehicular Ad-Hoc networks a challenging and promising research topic. An example for a VANET it is shown in the following figure (Figure 4.5).

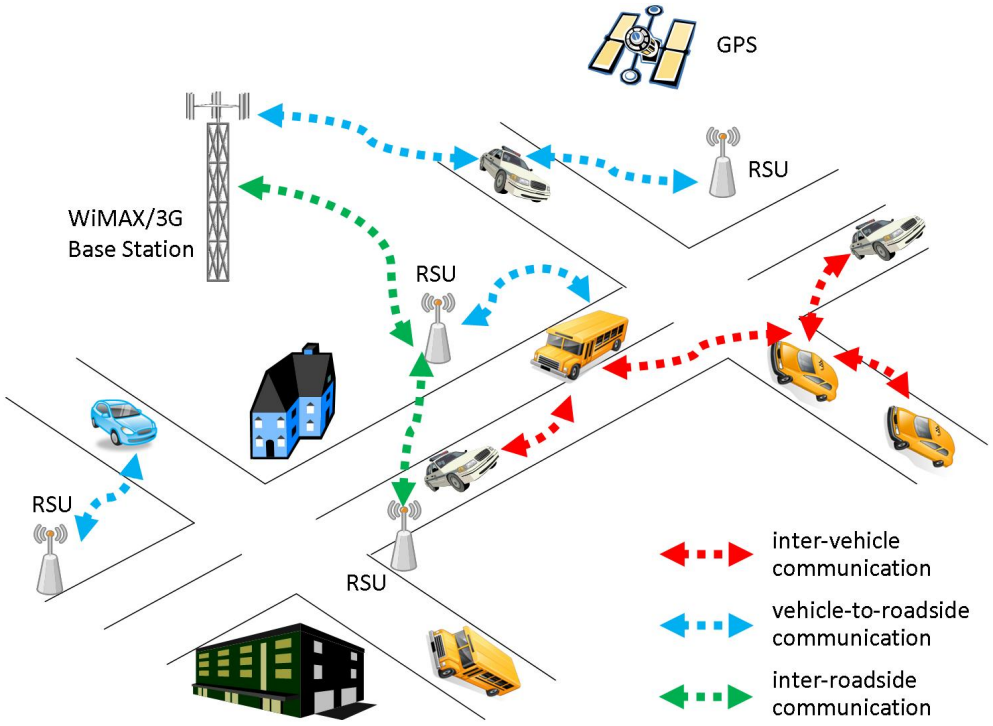

Figure 4.5 - Wireless Vehicular Networks

VANETs aim to provide our cars and roads with capabilities to make road more secure and to make our time on the road more enjoyable, enabling communications among nearby vehicles (car to car communication) as well as between vehicles and nearby fixed equipment (car to infrastructure communication). The following variety of applications is a typical example of an intelligent transportation system (ITS):

- **Safety**, in which a warning message will be broadcasted from a vehicle to its neighbourhood notifying about some event such as car collision or road surface conditions in order to decrease traffic accidents rates and enhance traffic flow control. It refers to applications or systems that increase the protection of the people in the vehicle as well as the vehicle itself.
- **Resource Efficiency**, referring to increase traffic fluency with data such as enhanced route guidance or parking spot locator services. Better efficiency results in less congestion and lower fuel consumption, helping to minimize environmental and economic impact.
- **Infotainment and Advance Driver Assistance Services (ADAS)**, combining *information* and *entertainment* and offering multimedia and Internet connectivity facilities for passengers.

The applications explained above are also shown in the Figure 4.6.

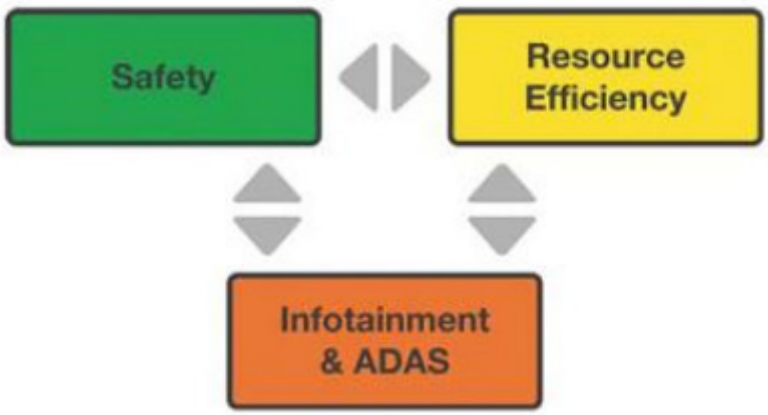

Figure 4.6 - Application Domains [31]

The huge potential of car-to-car connectivity is fundamentally due to the constant growth of automotive market and the increasing demand for the car safety. Some issues relating to architecture, routing, security, performance or QoS should be investigated. It is necessary to put special attention to ensure interoperability through the standardization of protocols and interfaces in order to allow the communication between vehicles from different manufacturers.

4.3.1. VANET PROPERTIES

As previously mentioned, a VANET represents a specific aspect of MANETs. Nevertheless, research works studied and carried out in the field of MANETs cannot be applied directly in the context of vehicular networks because of the characteristics of VANET making the application of MANET protocols and architecture inappropriate. In the following, the main properties and constraints related to the environment of vehicular Ad-Hoc networks are presented below.

Processing, Energy and Communication Capacity

Contrary to the context of mobile Ad-Hoc networks where the energy constraint represents one of the main challenges, vehicles in VANETs have no limit in terms of energy, have large processing capabilities and allow supporting several communication interfaces.

Environment and Mobility Model

Environments considered in Ad-hoc networks are often limited to open spaces or indoors. Vehicle movements are connected to road infrastructure, on highways, or within a metropolitan area. The constraints imposed by this time of environments, such as radio obstacles because of buildings, multipath and fading effects, considerably affect the mobility model and radio transmission quality.

Type of Information and Diffusion

Since one of the key VANETs applications is prevention and road safety, the type of communications will focus on message broadcast from a source to several receivers. Nevertheless, the vehicles concerned by such diffusion depend on their location and their degree of implication on the event. In such situations, communications are mainly unidirectional.

Network Topology and Connectivity

Contrary to Ad-Hoc networks, VANETs are characterized by very high mobility because of car speed. Thus, an element can quickly join or leave the network in a very short time, which makes the topology changes frequently. Solutions must consider this constraint where connectivity is one of the key parameter. In addition, properties inherent to VANETs, especially in terms of size, raise scaling problems and complete revision of existing solutions is required.

Security

Data sensitivity transmitted over a VANET demonstrates a high need for security. In fact, the importance of security in this context is vital because of the critical consequences resulting from a violation or attack. In addition, with a highly dynamic environment characterized by almost instant arrivals and departures of cars, the deployment of a security solution must cope with specific configuration en constraints.

4.3.2. CHALLENGES IN VANETS

When deploying of a vehicular network system, several issues have to be resolved. VANET characteristics – rapid topology changes, frequent fragmentation, variable and highly dynamic scale and network density, etc. – are opening some brand new lines of investigation and challenges for the scientific community.

In what follows, there is some briefly mentions about some issues related to VANETs:

Routing

In vehicular networks, mobility is constant. This fact causes extremely fast changes in network topology and involves the need to reconfigure the routing table of each node. Frequent network partitioning in VANETs requires a different approach, e.g. the 'carry and forward' idea [32], where, if there isn't a direct route, a packet is carried by a node until it could be forwarded to a node being closer to the destination, or the Delay Tolerant Networking.

Delay Tolerant Networking (DTN) is an approach to computer network architecture that wants to address the technical questions in heterogeneous networks, such as mobile networks, that could lack continuous network connectivity. The Delay Tolerant Network Research Group (DTNRG) [33] has defined an architecture based on a store and forward paradigm to

interconnect networks, even without end-to-end connectivity. Each DTN node may store packets and, when is appropriate, forward them towards the destination through intermediate nodes.

Security

Security is an issue that needs to be carefully addressed and assessed in the design of the vehicular communication system. In a wired network, user has to access to physical wire if s/he wants to access the network's information. However, wireless communications are weak from this point of view, because they use air as the transmission medium. This problem gets worse in vehicular networks due to the non-existence of infrastructure that provides security services centralization like user authentication or packet ciphering. The issue to be addressed includes trust – vehicles must be able to trust the messages they receive–, resiliency and efficiency –e.g. real-time message authentication–.

Privacy is also considered a major issue. Anonymity must be preserved making impossible tracking a vehicle for non-trusted parties. Not taking into account privacy could result in a multiple lawsuits after the network is deployed.

IEEE 802.11p dynamically assigns MAC addresses, along with a mechanism to duplicate MAC address discovery, thus vehicles and drivers would not be traceable for the MAC address.

As we can read in the online magazine DailyTech [34], Daimler-Chrysler is putting the finishing touches on a new system, *Wireless Local Danger Warning (Willwarn)*, that uses on-board ABS (Antilock Brake System), ESP (Electronic Stability Control), EBD (Electronic Brake Distribution) and GPS (Global Positioning System) systems to monitor hazardous road conditions or broken down vehicles. The information collected is then displayed to the driver so that proper precautions can be taken to avoid or safely navigate problem areas on the road. The Willwarn system is based on IEEE 802.11a/p and made use of the communication platform developed in the German *Network on Wheels (NOW)* project [35].

Quality of Service (QoS)

Quality of Service in wired networks is provided by different types of resources reservation mechanisms. However, executing these mechanisms is very complex due to the special features of VANET, such as high mobility nodes and large-scale node population. Nowadays, there exist some proposals; nevertheless, the majority of them are theoretical, simulated or implemented with fewer nodes.

Tarng et al. [36] proposed a method based on the stability from the radio propagation: signal strength and path loss; Sun et al. [37] proposed a grid based protocol. The digital map is pre-set with a grid. A routing path is selected based on the traffic features, such as the intersection, the number of vehicles, and roads, in a grid. Recently, Yan et al. [38] proposed a routing selection and maintenance based on the mobility of vehicles. The main ideas are to select and maintain one routing path and one backup routing paths based on the mobility of

vehicles (relative speed and direction). They reduce delay and response time in QoS terms because the retransmission caused by routing breakage is reduced and they improve the throughput in QoS terms.

Power Management

Power management in VANET is not concerned about energy efficiency, but rather about the transmission power – when too high, the on-going transmission could disrupt another transmission at a distant node due to interferences –. Thus, the denser the network is the lesser transmission power should be used. Several algorithms could be employed here, e.g. in [39] the power is adjusted to keep the number of neighbours within the maximum and minimum threshold. On the other hand, [40] concentrates on improving the 1-hop broadcast coverage by transmission power adjustments.

4.3.3. VANET STANDARDS

Different Ad-Hoc technologies and standards are integrated in VANET technology. Some of these technologies are Wireless Fidelity (WiFi) IEEE 802.11p, Wireless Access in Vehicular Environments (WAVE) IEEE 1609, WiMAX IEEE 802.16, Bluetooth, or ZigBee for easy, accurate, effective and simple communication between vehicles on dynamic mobility.

Vehicular Ad-Hoc Networks are expected to implement variety of wireless technologies and standards such as Dedicated Short Range Communications (DSRC), a one-way or two-way short- to medium-range wireless communication channels specifically designed for automotive use. Other wireless technologies that have been proposed to be used in VANETS are Cellular, Satellite, and WiMAX. Thus, VANET can be viewed as component of the Intelligent Transportation Systems (ITS).

4.3.4. VANET CURRENT DEPLOYMENTS AND APPLICATIONS

A VANET communication platform allows an enormous variety of applications aimed at administration, companies, drivers and people in the vehicle. These services will help and support topics as important as security driving, safety-related, traffic and fleet control as well as entertainment applications. Generally, from the connectivity point of view they could be divided into four groups: car-to-car traffic, car-to home, car-to-infrastructure and routing based applications.

4.3.4.1. SAFETY-RELATED APPLICATIONS

Safety-related applications are the most important kind of applications for VANETs due to its main objective: decrease of injuries and deaths due to vehicle accidents. In this context, the

European Commission is making an important effort to investigate, develop and implement these services in order to come into effect as soon as possible.

Cooperative Collision Avoidance

This service is about helping driving by detecting possible obstacles in the road. One such application would be emergency notifications. In case of an accident or sudden hard braking, a notification is sent to the following cars. This information could also be propagated by cars driving in the opposite direction and, thereby, conveyed to the vehicles that might run into the accident.

For a correct display of this service, it would be necessary a little installation in user's equipment which sends information about his/her position, trajectory or speed to the neighbours. Also, another system in the vehicle permanently listens to reliable information from the rest of vehicles and infrastructure.

Cooperative Driver Assistance System

This service exploits the exchange of sensor data or other status information among cars. The basic idea is to broaden the range of perception of the driver beyond his/her field of vision and further to assist the driver with autonomous assistance applications. By transmitting this data to cars following on the same road, drivers get information about hazards, obstacles or traffic flow ahead, resulting in more efficient and safe driving. Sensors could detect a danger and warn drivers with a brief description or even the driver could detect it and, through a vocal interface, describe the danger of the situation to the rest of users.

Information could be sent to every user in the network to inform, for example, that a traffic jam has started in a certain point of the road where we are driving. On the other side, it could be necessary a geocast message if, for example, we detect an oil puddle in one exit road; is necessary to send the information only for those vehicles that will take that exit.

eCall [41]

Is a project of the European Commission intended to bring rapid assistance to drivers involved in a collision anywhere in the European Union. In case of crash, an eCall-equipped vehicle automatically calls the nearest emergency centre. Even if no passenger is able to speak, e.g. due to injuries, a "Minimum Set of Data" is sent, which includes the exact location of the crash site. Shortly after the accident, emergency services therefore know that there has been an accident, and where exactly. eCall cuts emergency services response time. It goes down to 50% in the countryside and 60% in built-up areas. Annually, the quicker response will save around 2.500 lives in the European Union. The severity of injuries could be considerably reduced in 15% of cases. Drivers could also make an eCall by pushing a button in the vehicle. Witness of an accident can report it and automatically give the precise location.

4.3.4.2. COMFORT APPLICATIONS

The general aim of these applications is to improve passenger comfort and traffic efficiency. That could include nearest POI (Points of Interest) localization, current traffic or weather information and interactive communication. All kinds of applications, which may run on top of TCP/IP stack, might be applied here (online games or instant messaging). Another application is reception of data from commercial vehicles and roadside infrastructure about their businesses ('wireless advertising'), information about gas stations or enterprises which can set up stationary gateways to transmit marketing data to potential customers passing by, online help in case of breakdown, etc.

Furthermore, these services could be integrated with electronic payments, toll paying systems, etc. An important feature of comfort or commercial applications is that they should not interfere with safety applications. In this context traffic prioritizing and use of separate physical channels is a viable solution.

Optimum route calculation with real-time traffic data

This service could be used from the vehicle or from another point with an Internet connexion. The fact that, in the long term, all vehicles will be equipped with this system will make data-taking and data-publishing easier. This data, conveniently analysed, will inform about the state of the road, prevent traffic jams, etc.

4.3.4.3. APPLICATIONS FOR ADMINISTRATION

Vehicle identification

This service will provide a safe and fast way of information provision from vehicles without the need of stopping them. It will be necessary an appropriate legislation to allow that each vehicle stores the necessary information in an electronic format and its automatic transmission if it is required by an authorized device.

Vehicle identification service will help police in different ways: it would be possible to check if a vehicle and his/her driver have the necessary documentation or, if an infraction is detected, its report would be automatically processed, etc.

4.4. INTRODUCING HSVN (HYBRID SENSOR VEHICULAR NETWORK)

HSVN consist in making WSN and VANET work together to constitute a communication framework that can be used by vehicles in order to assist drivers to reduce road accidents, fatalities and injuries.

This network represents a new concept of road sensor deployment, and a new generation of network architectures has to be created. These new architectures have to be proposed to offer robust, reliable and cost-effective approaches for HVSN. The purpose of HSVN is that cars can monitor weather events conditions (such as rain or ice) or the amount of traffic density in remote road segments for potential users.

The idea is that WSNs gather information about weather (by its sensors deployed along the roadside) and about traffic density by passing cars. This information is later spread among all passing vehicles and later transmitted car by car using VANET. Thus, traditional VANET enlarge their scope, since other passing vehicles can recover information later and, furthermore, they receive information from WSN.

To allow communications between VANET and WSN a communication protocol has to be designed. This protocol has to satisfy efficiently some characteristics. For example, it has to permit both networks to share data carrying road information and short video streams. This data exchange has to be fast (due to connectivity between vehicles and WSN sink is limited on time).

One of the key points of HSVN is that communication between WSN and VANET makes possible to extend the transmission range from a VANET to a larger region, by the use of cooperation among Ad-Hoc nodes within both networks.

The content of exchanged messages regarding road safety has to be defined. Messages have to include information about road conditions and density, and can include a low quality image of the incoming intersections (permitting the driver or the co-driver to take a quick look to foresee real-time information of the road ahead).

An example of how this protocol could work, for the situation represented on Fig 4.7, is the following:

- Group leader vehicles (cars A and C) gather road information from the vehicles behind in their respective groups.
- The road information is interchanged between group leaders which travel in opposite directions (cars A and C).
- The new received information is spread from the group leader among the vehicles in the group.
- Passengers can access to the Internet by means of the Access Points APs spread along the roadside or also by means of public transportation with Internet connection.

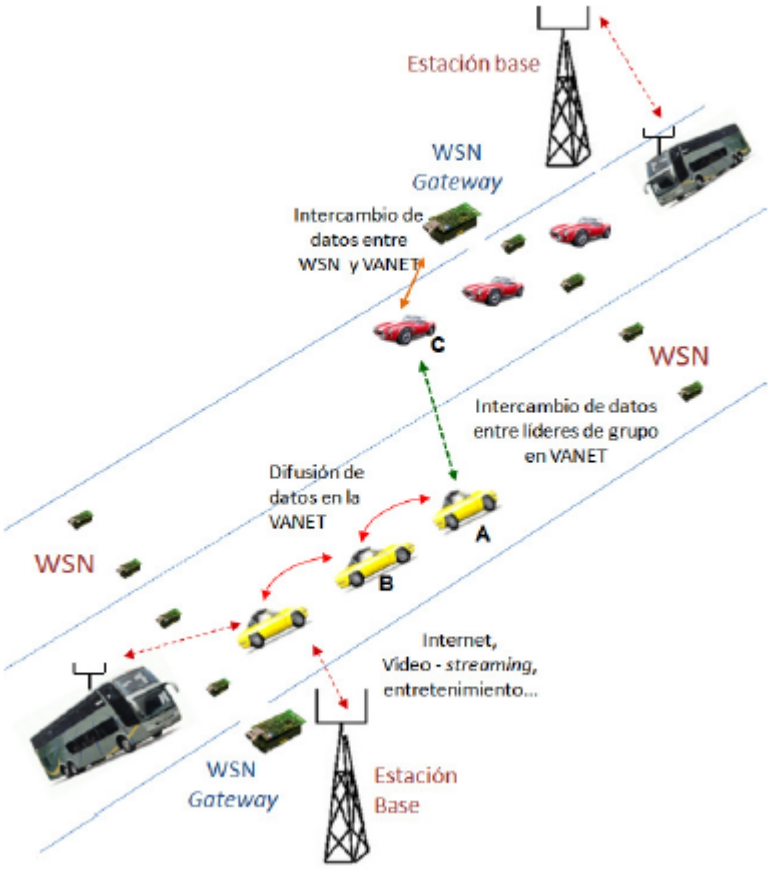

Figure 4.7 - General HSVN Scenario [42]

5. NETWORK ARCHITECTURE

In this chapter the architecture of HSVN is described, as well as a communication protocol [42] created by the same research group that this project. Finally, the most common routing protocols of all networks participating on smart cities previously described are reviewed. Nevertheless, the focus is on the routing protocol that will be used for further simulations, the Ad-hoc On-demand Distance Vector (AODV) [43].

5.1. HYBRID SENSOR AND VEHICULAR NETWORK ARCHITECTURE

Development of Hybrid Sensor and Vehicular Network requires an improvement of standard Wireless Sensor Networks architecture and protocols. Particularly, architecture changes from centralized architecture, where nodes self-organize at bootstrap phase for the delivery of data to the sink node, to a fully distributed one, where no pre-determined gateway can be identified. Furthermore, in order to prolong network life time, an energy-aware forwarding strategy and a delay-aware forwarding strategy are needed.

In energy-aware forwarding strategies, the expected consumed energy hop by hop is a key issue when taking the decision of sending a packet to one neighbor to another (and not only geographic information, as could be thought). In these strategies, consumed energy (in energy-aware forwarding) and delay (in delay-aware forwarding) weight the geographic distance. In such a way, more stressed nodes are avoided from a certain point on and performance is improved [44] [45] [46].

5.2. COMMUNICATION PROTOCOL

For communicating a WSN and a VANET a communication protocol is always needed. In this section a communication protocol proposed to communicate VANET and WSN by the same research group is described. This protocol is based on the one proposed by Carolina Tripp Barba in her PhD research work and presented on October 2010 in the international conference Performance Evolution of Wireless Ad hoc, Sensor and Ubiquitous Networks (PEWASUN). The communication algorithm between WSN and VANET proposed has the main purpose is to fulfill all the types of communications produced between vehicles and static road sensors [42]. Nevertheless, some minor differences have been considered by a parallel project developed by Pablo Regañas Soto [47].

Three types of communications are considered between WSNs and VANETs. These types of communication are described in the following subsections.

5.2.1. COMMUNICATIONS BETWEEN STATIC SENSORS AND VEHICLES (WSN-VANET)

The first type of communication described by the author is the communication between WSN and vehicles within a VANET. This means that WSN is sending and adding information to VANET and, at the same time, WSN can update its information thanks to information gathered by the vehicle from other WSN. This communication can be divided into the following steps:

(1) Communication from WSN to Vehicle

- The WSN gateway detects a vehicle within its transmission range.
- The WSN gateway sends a connection request (14 bytes) to the passing vehicle.

(2) Communication from Vehicle to WSN

- The vehicle sends back an acknowledgement (ACK) (14 bytes) to the WSN gateway. This ACK contains the coordinates of its destination (20 bits). This way the vehicle sets high priority to the information of its own interest. It also includes the identification (ID) of the vehicle.

(3) Communication from WSN to Vehicle

- The WSN gateway sends a packet including road information regarding all segments being in the path along the destination of the passing vehicle, as well as data regarding other paths. The packet contains the following information:
  - Two-bit header field per road segment including information about the traffic density/accident of each road segment along the path. The data codification is the following (Table 5.1):

Traffic Density / Accident Codification	
0	Free segment
1	Semi-congested segment
2	Very congested segment
3	Accident

Table 5.1 - Traffic Density / Accident Protocol Codification.

- Two-bit header field per road segment including information about the weather state of the roads. The data codification is the following (Table 5.2):

Weather Codification	
0	Good weather
1	Rain
2	Storm
3	Ice

Table 5.2 - Weather Protocol Codification.

- The WSN gateway sends a packet of a small image (50 Kbytes approx.) regarding the next cross of the path in the vehicle's destination. After that, other images regarding other intersections can be sent, as long as both nodes are in transmission range.

(4) Communication from Vehicle to WSN

- The vehicle sends to the WSN gateway road information (data and image) which was previously gathered from other vehicles or other WSN. The WSN updates that information (if newer) regarding the states of the roads. The content and codification of these packets were described above.

(5) Communication from Vehicle to WSN

- The vehicle reaches the coverage limit and the connection ends.

5.2.2. VEHICLE-TO-VEHICLE COMMUNICATIONS (SAME DIRECTION)

Another type of communication described by the Author corresponds to the VANET itself, in the case of vehicles moving in the same direction (see Figure 4.7). In this case, Vehicles share information gathered from the WSN gateways. This communication can be divided into the following steps:

(1) Communication from Vehicle A to Vehicle B

- Vehicle B is detected by vehicle A within its transmission range. Vehicle B is behind vehicle A in the same direction (Fig. 4.7).
- A connection request (14 bytes) is sent from vehicle A to vehicle B.

(2) Communication from Vehicle B to Vehicle A

- An ACK (14 bytes) in sent by vehicle B to vehicle A. It includes ID and its destination coordinates.

(3) Communication from Vehicle A to Vehicle B

- The Vehicle A sends a packet to Vehicle B regarding the state of the roads and traffic density of all the road segments that are in the destination path of vehicle B and also of the roads within other paths. The packet contains the following information:
  - Two-bit header field per road segment including information about the traffic density/accident of each road segment along the path. (See Table 5.1).
  - Two-bit header field per road segment including information about the weather state of the roads. (See Table 5.2).
- Transmission of a small image (50 Kbytes approx.) regarding the next cross of the path in the destination of vehicle B. After that, other images regarding other intersections can be sent, as long as both nodes are in transmission range.

(4) Communication from Vehicle A to Vehicle B

- Vehicle A reaches the coverage limit of vehicle B and the connection ends.

Following the same process, road information is spread backward through the group of vehicles. Thus, every new vehicle in the group sends its own road information forward till the leader, through the other relying vehicles in the group. This way, the leader gathers the whole road information of its group of vehicles and keeps always updated.

5.2.3. VEHICLE-TO-VEHICLE COMMUNICATIONS (OPPOSITE DIRECTIONS)

The last type of communication described by the Author corresponds with vehicles moving in opposite directions (see Figure 3.7). In this case, vehicles share information from both VANETs.

In case that the detected vehicle goes in opposite direction, the connection will only be established between the first vehicles of both groups (group leaders). The communication between vehicle A and vehicle C can be divided into the following steps:

(1) Communication from Vehicle A to Vehicle C

- Vehicle C is detected by vehicle A within its transmission range. Vehicle C moves opposite direction that vehicle A (Fig. 4.7).
- A connection request (14 bytes) is sent from vehicle A to vehicle C.

(2) Communication from Vehicle C to Vehicle A

- An ACK (14 bytes) in sent by vehicle C to vehicle A. It includes ID and its destination coordinates.

(3) Communication from Vehicle A to Vehicle C

- The Vehicle A sends a packet to Vehicle C regarding the state of the roads and traffic density of all the road segments that are in the destination path of vehicle B and also of the roads within other paths. The packet contains the following information:
  - Two-bit header field per road segment including information about the traffic density/accident of each road segment along the path. (See Table 5.1).
  - Two-bit header field per road segment including information about the weather state of the roads. (See Table 5.2).
- Transmission of a small image (50 Kbytes approx.) regarding the next cross of the path in the destination of vehicle B. After that, other images regarding other intersections can be sent, as long as both nodes are in transmission range.

(4) Communication from Vehicle A to Vehicle C

- Vehicle A reaches the coverage limit of vehicle C and the connection ends.

Finally, each group leader disseminates the new road information among the vehicles within its own group.

5.2.4. PERFORMANCE OF A COMMUNICATION PROTOCOL PROPOSED

Once the communication protocol has been defined, some considerations are done by the Author in order to check if this communication protocol is or is not implementable.

To control the amount of information needed, there may be a different number of road segments per destination. The number of segments needed to obtain an accurate description of the environment depends on the type of scenario and on the length of road to that destination. Three main possible scenarios are considered in order to determine the number of segments used. These three scenarios are urban scenario, rural scenario and highway scenario. The number of segments considered for each scenario is:

Urban scenarios: 200 m length segments are used. If the average trip for an urban scenario is a total of 5 km and road segments length is 200 m, a total number of 25 segments are used.

Rural scenario: 5 km length segments are used. If the average trip for a rural scenario is a total of 50 km and road segments length is 5 km, a total number of 10 segments are used.

Highway scenario: 10 km length segments are used. If the average trip for an urban scenario is a total of 500 km and road segments length is 10 km, a total number of 50 segments are used.

Once segment's length is determined, it has to be checked if there is enough time to fulfill the data interchange between the WSN gateway and a passing vehicle. In order to check it, the Author proposes a numerical example using the scenarios previously described.

In this example, the communication standard used to communicate both networks is the IEEE 802.11b MAC. This standard has a nominal link bandwidth of 11Mbps and a throughput (th) of around 7Mbps (for UDP like traffic).

Let us consider a maximum speed of the vehicle of "v" [m/s] and a transmission range of the WSN sink node of "r" [m]. Thus, the available time for data interchange, shown in Figure 5.1, is

$$t_{available} \approx \frac{2r}{v} \ [sec] \quad \text{(Eq. 1)}$$

Thus, $t_{available}$ is the time available to establish the communication between both networks and within this time, the transmission of all the packets must be done according to the messages interchange described in section 5.2.1. of this project.

Figure 5.1 shows the given situation for calculating the available time to establish communication.

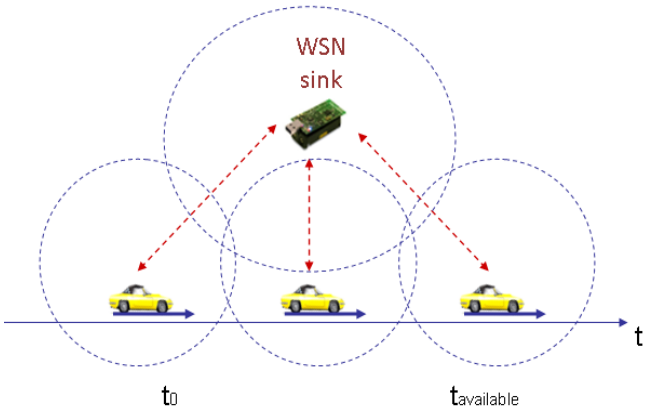

Figure 5.1 - Temporal Available Time to Interchange Messages [42]

The data information regarding all the road segments is transmitted in a single packet (p) of 1000 bytes. Thus, considering the throughput of IEEE 802.11b, the time needed to transmit this packet is:

$$t_{road\_segments} = \frac{p}{th} = 1.14 \ [msec] \quad \text{(Eq. 2)}$$

The transmission of an image of 50 Kbytes takes 50 packets and the time needed is:

$$t_{image} = 50 * t_{road\_segments} = 0.057 \ [sec] \quad \text{(Eq. 3)}$$

The total time required to send that information is:

$$t_{required} = t_{road\_segments} + t_{image} = 0.058 \ [sec] \quad \text{(Eq. 4)}$$

Thus, if:

$$t_{required} < t_{available} \quad \text{(Eq. 5)}$$

The exchange of information (data and images) between the car and the sensor sink can be successfully done. Furthermore, more images regarding other intersections could be sent as well within the available interval of time, $t_{available}$.

Then, for the scenarios previously considered, and with a transmission range of the WSN sink of 80 m, we can find de following results for $t_{available}$.

Urban scenario (v = 13.9 m/s): $t_{available}$ = 11.52 s

Rural scenario (v = 22.2 m/s): $t_{available}$ = 7.2 s

Highway scenario (v = 33.33 m/s): $t_{available}$ = 4.8 s

It can be noticed that with a $t_{required}$ of 0.058 sec, $t_{available}$ from all scenarios is more than enough to accomplish the communication. Thus, more additional images (t_{image}=0.057 sec) could be sent during the available period of time.

5.3. ROUTING PROTOCOLS IN AD HOC NETWORKS

In all communications networks, the choice of the routing protocol is a key issue. Routing protocols are protocols that specify how routers communicate to each other, disseminating information that enables them to select routes between any two nodes on a computer network. The choice of the route is always done by routing algorithms.

Traditional routing protocols used in wired networks are not suitable for Ad-Hoc wireless networks as VANET. These protocols are based on periodic route updating mechanisms that increase overhead and cannot efficiently handle topology changes (key issue of VANET).

Mobility of VANET nodes and topology changes within the network made routing more complex and requires more robust and flexible mechanisms to find and maintain the different routes. Protocols should be able of adapting dynamically their routes to these topology changes (even maintaining connectivity is hard and some nodes can disappear from the network).

Thus, routing protocols for these networks must deal with the following premises:

- Distributed operation**:** One of the bases of VANETs.
- Signaling reduction**:** To reduce energy consumption and enhancing network efficiency.
- Keeping the routes loop free**:** To avoid network congestion.
- Reduced processing time**:** To save resources of nodes.
- Management of asymmetric links**:** These are caused by different power levels among mobile nodes and other factors such as terrain conditions.

As it has been said, Hybrid Sensor Vehicular Network combines two types of Ad-Hoc wireless networks, WSN and VANET. One of the most important issues in HSVN is to create a new

routing protocol that can communicate both networks. Traditionally, these networks (VANETS and WSN) use different routing protocols due to particularities of each network. Some of the properties from these networks are listed on Table 5.3.

Property	WSN	VANET
Network size	Large	Large
Node's mobility	Mostly static	High, nonrandom
Energy limitations	Very high	Very low
Node's computation power	Very low	High
Node's memory capacity	Very low	High
Location dependency	High	Very high

Table 5.3 - Properties from WSN and VANET

Thus, this new protocol should combine and take into account the particularities listed above.

The objective of this project is not the implementation of this protocol (that could be done in the future). Nevertheless, a review of the current routing protocols of VANET and WSN is interesting, and for the simulations of this project Ad-hoc On-demand Distance Vector (AODV) protocol will be used.

5.3.1. DIFFERENCE BETWEEN ROUTING PROTOCOLS IN MANET AND VANET

Scalability

MANET supports a limited number of mobile nodes (about one or two hundred), so some routing protocols can store routes to all other nodes in the network within their routing tables. In the case of a VANET, storing routing tables for all vehicles is really impractical.

Full Connectivity

Full connectivity is not possible in vehicular networks. A destination node could be not reachable at the moment of sending a packet, a vehicle can move carrying the packet until the destination is eventually reached.

Mobility Prediction

Most MANET routing protocols assume random mobility patterns. In the case of VANETs, node mobility is restricted by the topology of the streets, speed limits or traffic signals.

Anticipation of Path Breakages

MANET uses adjustable timers on its routing protocols that can react in front of a routing break. In several VANET scenarios the information about the mobility patterns of neighboring nodes can help prevent path breakages before they happen.

Extensive use of Flooding

Most MANET routing protocols are based on flooding. Flooding consumes a lot of bandwidth due to control messages and limit performance in large networks. In VANETs the number of nodes is very high and flooding has to be limited.

Nonlocal Operation

To create and maintain routing paths in MANETs, it is required the effort of all nodes in the network. Localized routing solutions in which nodes only need information from their neighborhood are more appealing in terms of scalability, control overhead, and adaptation to different network conditions for VANETs.

Exploitation of Existing Knowledge

Currently VANET nodes are equipped with onboard units providing relevant information about expected environment. This information is not provided in MANET routing protocols.

5.3.2. ROUTING PROTOCOLS IN MANET

Although MANET protocols are not the most indicated to be used in VANET, these protocols have been used as a base of VANET routing protocols and it is important to know them before talking about proper VANET routing protocols. Due to software limitations and to simplify the work, MANET routing protocols will be used for the simulations.

A standard classification of Ad-Hoc network routing protocols has been done according to their different behavior. This classification divides routing protocols into proactive, reactive, hybrid and geographic routing protocols. The principal features of these protocols can be seen in Table 5.4.

	Proactive	Reactive	Hybrid	Geographic
Overhead	Very high	Low	Medium	Very low
Connection delay	Very low	Very high	Medium	Very low
Mobility impact	High	High	High	Very high
Scalability (nodes)	Very low	Low	Medium	Very high
Scalability (flows)	Very high	Very low	Medium	Very high

Table 5.4 - Principal Features for each Routing Protocol

Proactive Routing Protocols

These protocols issue periodic messages in order to learn the network topology and create routes to every other node present in the network. The main proactive protocols are Distance Sequenced Distance Vector (DSDV), Fisheye State Routing (FSR) and Wireless Routing Protocol (WRP).

Reacting Routing Protocols (On Demand)

These protocols work on the opposite way of proactive do. Reactive protocols do not issue any message to know the network topology until a data flow is about to be sent. The main reactive routing protocols are Dynamic Source Routing (DSR), Ad hoc On Demand Distance Vector (AODV), Dynamic On demand MANET routing protocol (DYMO), On Demand Multicast Routing Protocol (ODMRP) and Associativity Based Routing (ABR).

Hybrid Routing Protocols

Propose to proactively set up routes to the nodes inside a given zone, while letting the process of acquiring routes outside the zone operate on demand. The main hybrid routing protocols are Zone Routing Protocol (ZRP), Intrazone Routing Protocol (IARP), Interzone Routing Protocol (IERP).

Geographic Routing Protocols

Exploit location information to make forwarding decision only based on local information. In order to route a packet, a node must know its own position, the destination's position and the one hop neighbor's position. Global position can be learn by means of GPS and are exchanged between neighbors. The main geographic routing protocol is Location Aided Routing (LAR).

5.3.3. ROUTING PROTOCOLS IN VANET

New routing protocols have to be developed for VANET, due to its differences with MANET. The routing protocols in VANET can be divided into three different types: source routing protocols, geographic routing protocols and trajectory based protocols.

5.3.3.1. SOURCE ROUTING BASED PROTOCOLS

The main source routing based protocols in VANET are the following:

GSR (Geographic Source Routing)

The main idea of GSR protocol is to use the knowledge of the street map of the area where the nodes are moving using a static street map and location information about each node. This information is found of the navigator device of the vehicle [48]. The main problem of his protocol is that uses a global initial flooding to determine the location of the destination, thus, the scalability is not guaranteed.

SAR (Spatial Aware Routing)

SAR is a position based unicast routing protocol that predicts and avoids route recovery caused by permanent networks avoid. SAR relies upon the extraction of a static street map from an

external service such as GIS (Geographic Information System) to construct a spatial model for unicast routing. In SAR, a node determines its location on the spatial model and uses the street information to calculate a shortest path to a packet's destination. When this path is determined, the set of geographic locations to be traveled is embedded into the header of the packet [49].

A-STAR (Anchor based Street and Traffic Aware Routing)

This protocol utilizes city bus routes and a help to find path with high probability for delivery. The A-STAR protocol consists of including information about the traffic density of the street edges weights. The main idea is to determine a sequence of streets with a high probability of having enough vehicles to do easy the transmission of the data. This protocol uses a static street map to route messages around potential radio obstacles [50].

CAR (Connectivity Aware Routing)

Introduces the idea to solution some problems present in the other protocols. The unsuitable of applying pure geographic routing techniques and the assumption of connectivity between nodes inside the streets that some protocols make when computing the path to follow to the destination is solved. This protocol has the ability to maintain a cache of successful routes between various sources and destination pairs. Also predicts positions of destination vehicles, repairs routes as those positions change, and employs geographic marker messages. Use a preliminary flooding based phase to localize the destination node [51].

5.3.3.2. GEOGRAPHIC ROUTING BASED PROTOCOLS

This kind of protocols is the most used in VANET, but it has the problem that they only transmit messages between vehicles in the same street. These protocols do not use source routing, and the main idea is use geographic routing directly over the map streets, that helps to take decisions about new directions.

The main geographic routing based protocols are:

GPCR (Greedy Perimeter Coordinator Routing)

This protocol eliminates the precondition that assume that each node knows the complete street map, and does not use flooding. This protocol improves upon GSR by eliminating the requirements of an external static street map. To treat to minimize potential radio obstacles, this protocol modified the typical destination based greedy forwarding strategy such that only route messages along streets. In this way, routing decisions are only made at street intersections [52].

VADD (Vehicular Assisted Data Delivery)

VADD is based on the idea of carry and forward. The most important issue is to select a forwarding path with the smallest packet delivery delay. This protocol requires each vehicle to know its own position and also requires an external static street map that includes traffic statistics [53].

5.3.3.3. TRAJECTORY BASED PROTOCOLS

Currently most of the vehicles are equipped with GPS [54] [55] technology to determine position. Also with Internet connection these devices support periodical updates of information such traffic, street closed, weather, location of restaurant, etc. All these information is very useful for a routing protocol.

The information about the trajectory that the vehicle is following is very important for routing protocols, because allows taking routing decisions. Then, the trajectory based protocols are created, being the most important the following ones.

TBF (Trajectory Based Forwarding)

A novel method to forward packets in a dense Ad-Hoc network that makes it possible to route a packet along a predefined curve. The routing process consists of selecting as next relay neighbor closer to a point in the curve [56].

GeOpps (Opportunistic Geographical Routing)

The GeOpps protocol assumes that the source node already knows the position of the destination by some kind of location database or similar approach. Also assume that cars can interchange their expected routes using beacon messages.

The nodes in this protocol follow a predefined trajectory; use the expected trajectory of neighbors to take routing decisions. The general idea is that if a node finds a neighbor whose trajectory goes closer to the destination's position that its own, it send the packet to that node [57].

5.3.4. ROUTING PROTOCOLS IN WSN

As previously said, routing protocols in WSN are different to those in VANET. The WSN routing problem presents a very difficult challenge that can be posed as a classic trade-off between responsiveness and efficiency. This trade-off must balance the need to accommodate the limited processing and communication capabilities of sensor nodes against the overhead required to adapt to these [58].

Routing protocols for WSN can be divided into two different groups, those based on network structure and those depending on the protocol operation. In the following section, the differences between these two types of protocols are exposed.

5.3.4.1. ROUTING PROTOCOLS BASED ON THE NETWORK STRUCTURE

5.3.4.1.1. FLAT-BASED ROUTING

In the Flat-based routing protocols each node plays the same role and sensor nodes collaborate together to perform the sensing task. WSN can be composed of a large number of sensors (nodes), so it is not feasible to assign a global identifier to each node. Therefore, data centric routing is used, where the base station (BS) sends queries to certain regions and waits for data from the sensors located in the selected regions.

The main flat-based routing protocols are:

SPIN (Sensor Protocols for Information via Negotiation)

This family of protocols uses data negotiation and resource-adaptive algorithms. Nodes assign a high-level name to completely describe their collected data (called meta-data) and perform meta-data negotiations before any data is transmitted (to assure no redundant data sent throughout the network). SPIN has access to the current energy level of the node and adapts the protocol being run based on how much energy remain [58].

DD (Directed Diffusion)

Protocol that combines data coming from different sources (in-network aggregation) eliminating redundancy and minimizing the number of transmissions (saving energy and prolonging lifetime of the network). Sensors measure events and create gradients (attribute value and direction) of information in their respective neighborhoods. The base station requests data by broadcasting interests. Interest describes a task required to be done by the network. The interest is broadcasted by each node to its neighbors, and these reply setting up a gradient towards the senders. This process continues until gradients are setup from the sources back to the BS. The strength of the gradient may be different towards different neighbors resulting in different amounts of information flow [59].

Rumor Routing

Rumor routing comes from directed diffusion and is mainly intended for applications where geographic routing is not feasible.

The key idea is to route the queries to the nodes that have observed a particular event rather than flooding the entire network to retrieve information about the occurring events. When a node detects an event, it adds such event to its local table, called events table, and generates an agent. Agents travel the network in order to propagate information about local events to

distant nodes. When a node generates a query for an event, the nodes that know the route, reply to the query by inspecting its event table [60].

MCFA (Minimum Cost Forwarding Algorithm)

In this protocol, each node maintains the least cost path estimate from itself to the base-station. It does not require a unique ID or either maintain a routing table, the direction of routing is always known. The sensor node broadcasts the message to its neighbors. When these nodes receive the message, they check if it is on the least cost path between the source sensor node and the base-station. If this is the case, it rebroadcasts the message to its neighbors. This process is repeated until the base-station is reached [61].

Energy Aware Routing

The objective of this protocol is to increase the network lifetime. It differs from DD in the sense that it maintains a set of paths instead of maintaining one optimal path at higher rates. These paths are maintained and chosen by means of a certain probability, which value depends on how low the energy consumption of each path is. The protocol initiates a connection through localized flooding, which is used to discover all routes between source/destination pair and their costs; thus building up the routing tables. The high-cost paths are discarded and a forwarding table is built by choosing neighboring nodes in a manner that is proportional to their cost [62].

Routing Protocols with Random Walks

The objective is to achieve load balancing in a statistical sense and by making use of multipath routing in WSN. Only large scale networks where nodes have very limited mobility are considered. It is assumed that sensor nodes can be turned on or off randomly, each node has a unique identifier but no location information is needed. To find a route from a source to its destination, the location information is obtained by computing distances between nodes [63].

5.3.4.1.2. HIERARCHICAL-BASED ROUTING

These protocols are based on hierarchical architecture, where higher energy nodes can be used to process and send the information while low energy nodes can be used to perform the sensing in the proximity of the target.

Creating clusters and assigning special tasks to cluster heads can greatly contribute to overall system scalability, lifetime, and energy efficiency. Therefore, hierarchical routing is two-layer routing where one layer is used to select cluster-heads and the other layer is used for routing. The main hierarchical-based routing protocols are:

LEACH (Low Energy Adaptive Clustering Hierarchy)

Cluster-based that includes distributed cluster formation. LEACH randomly selects a few sensor nodes as cluster-heads (CH) and rotates this role to evenly distribute the energy load among the sensors in the network. The cluster-head nodes compress data arriving from nodes that belong to the respective cluster and send an aggregated packet to the base station in order to reduce the amount of information that must be transmitted to the base station [64].

PEGASIS (Power-Efficient Gathering in Sensor Information Systems)

It is an enhancement over LEACH protocol. In order to extend network lifetime, nodes need only communicate with their closest neighbors and they take turns in communicating with the base-station (reducing the bandwidth consumed. When a round of nodes communicating with the base station ends, a new round starts and so on. This reduces the power required to transmit data per round as the power draining is spread uniformly over all nodes [65].

TEEN and APTEEN (Threshold-sensitive Energy Efficient Protocols)

These two hierarchical routing protocols are proposed for time-critical applications.

In TEEN, the sensor nodes sense the medium continuously, but the data transmission is done less frequently. In this case a cluster head sensor sends its members *Hard threshold* or *Soft threshold*. When hard threshold, it tries to reduce the number of transmissions by allowing the nodes to transmit only when the sensed attribute is in the range of interest. When soft threshold, a small change in the value of the sensed attribute triggers the node to switch on its transmitter and transmit. The soft threshold further reduces the number of transmissions that might have otherwise occurred when there is little or no change in the sensed attribute [66].

In APTEEN, the threshold value change upon time, according to the user needs and the type of the application [67].

MECN (Small Minimum Energy Communication Network)

This protocol computes an energy-efficient sub-network by using low power GPS. It identifies a relay region for every node. The relay region consists of nodes in a surrounding area where transmitting through those nodes is more energy efficient than direct transmission [68].

SOP (Self Organizing Protocol)

A self-organizing protocol and application taxonomy are used to build architecture able to support heterogeneous sensors. Some sensors probe the environment and forward the data to a designated set of nodes that act as routers. Router nodes are stationary and form the backbone for communication. Collected data are forwarded through the routers to the more powerful BS nodes. Each sensing node should be able to reach a router in order to be part of the network [69].

Sensor Aggregates Routing

The objective is to collectively monitor target activity in a certain environment (target tracking applications). A sensor aggregate comprises those nodes in a network that satisfy a grouping predicate for a collaborative processing task. Sensors in a sensor field are divided into clusters according to their sensed signal strength, so that there is only one peak per cluster. Then, local cluster leaders are elected. One peak may represent one target, multiple targets, or no target in case the peak is generated by noise sources [70].

VGA (Virtual Grid Architecture Routing)

It uses data aggregation and in-network processing to maximize the network lifetime. It arranges nodes in a fixed topology. A GPS-free approach is used to build clusters that are fixed, equal, adjacent, and non-overlapping with symmetric shapes [71].

HPAR (Hierarchical Power-Aware Routing)

The protocol divides the network into groups of sensors. All groups of sensors in geographic proximity are clustered together as a zone and each zone is treated as an entity. To perform routing, each zone is allowed to decide how it will route a message hierarchically across the other zones such that the battery lives of the nodes in the system are maximized. Messages are routed along the path which has the maximum over all the minimum of the remaining power, called the max-min path. The motivation is that using nodes with high residual power may be expensive compared to the path with the minimal power consumption [72].

TTDD (Two-Tier Data Dissemination)

This protocol provides data delivery to multiple mobile base-stations. Each data source proactively builds a grid structure which is used to disseminate data to the mobile sinks by assuming that sensor nodes are stationary and location-aware. Therefore, in TTDD sensor nodes are stationary and location-aware while sinks may change their locations dynamically. Once an event occurs, sensors surrounding it process the signal and one of them becomes the source to generate data reports [73].

5.3.4.1.3. LOCATION-BASED ROUTING

Location-based routing protocols address sensor nodes by means of their locations. Relative coordinates of neighboring nodes can be obtained by exchanging such information between neighbors. Alternatively, the location of nodes may be available directly by communicating with a satellite, using GPS (Global Positioning System), if nodes are equipped with a small low power GPS receiver.

To save energy, some location based schemes demand that nodes should go to sleep if there is no activity. More energy savings can be obtained by having as many sleeping nodes in the network as possible. Some examples of location-based routing protocols are:

GAF (Geographic Adaptive Fidelity)

In this protocol the network area is first divided by the use of a virtual grid. Inside each zone, nodes collaborate with each other to play different roles. For example, nodes will elect one sensor node to stay awake for a certain period of time and then they go to sleep. This node is responsible of monitoring and reporting data to the BS on behalf of the nodes in the zone. Thus, sensor nodes have three different state possibilities, *Discovery* (determining the neighbors in the grid), *Active* (reflecting participation in routing) or *Sleep* (when the radio is turned off) [74].

GEAR (Geographic and Energy Aware Routing)

It uses energy aware and geographically-informed neighbor selection heuristics to route a packet towards the destination region. The key idea is to restrict the number of interests in directed diffusion by only considering a certain region rather than sending the interests to the whole network. By doing this, GEAR can conserve more energy than directed diffusion [75].

MFR, DIR and GEDIR

These protocols deal with basic distance, progress, and direction based methods. The key issues are forward direction and backward direction. A source node or any intermediate node will select one of its neighbors according to a certain criterion.

MFR (Most Forward within Radius) is an algorithm that has the same path than the greedy method [76].

GEDIR (The Geographic Distance Routing) is a greedy algorithm that always moves the packet to the neighbor of the current vertex whose distance to the destination is minimized. The algorithm fails when the packet crosses the same edge twice in succession [76].

DIR (Distance Routing) algorithm chose as best neighbor this that has the closest direction (that is, angle) toward the destination. That is, the neighbor with the minimum angular distance from the imaginary line joining the current node and the destination is selected [76].

5.3.4.2. ROUTING PROTOCOLS DEPENDING ON THE PROTOCOL OPERATION

Routing protocols for WSN can also be based on the protocol operation. Depending on the protocol operation, routing protocols can be multipath-based, query based, negotiation-based, QoS-based or Coherent-based. Therefore, the protocol description of each type in [77]:

5.3.4.2.1. MULTIPATH-BASED ROUTING PROTOCOL

These routing protocols use multiple paths rather than a single path in order to enhance the network performance. Network reliability is increased at the expense of increased the overhead produced by maintaining the alternate paths.

The key idea is to split the original data packet into sub-packets and then send each sub-packet through one of the available multi-paths. It has been proved that even if some of these sub-packets are lost, the original message can still be reconstructed.

5.3.4.2.2. QUERY-BASED ROUTING PROTOCOL

In query based routing protocols the destination nodes propagate a query asking for some specific data (sensing task). This query is propagated from this node to the network. If a node having the data matching with the query is found, it sends this data back to the requesting node.

5.3.4.2.3. NEGOTIATION-BASED ROUTING PROTOCOL

Negotiation-based routing protocols use high level data descriptors in order to eliminate redundant data transmissions through negotiation.

The main idea of negotiation based routing in WSNs is to suppress duplicate information and prevent redundant data from being sent to the next sensor or the base-station by conducting a series of negotiation messages before the real data transmission begins.

5.3.4.2.4. QUALITY OF SERVICE-BASED ROUTING PROTOCOL

In QoS-based routing protocols, network balances between energy consumption and data quality.

Networks using this type of routing protocols have to satisfy certain QoS metrics in terms of delay, energy consumed, bandwidth, etc. when nodes deliver data to the base station.

5.3.4.2.5. NON-COHERENT AND COHERENT-BASED ROUTING PROTOCOLS

Non-coherent and coherent based routing protocols are based in the way data is processed in the operation within wireless sensor networks. These two data processing techniques work in the following way:

Non-coherent data processing routing: Nodes process raw data locally before sending it to other nodes for further processing. The nodes that perform further processing are called aggregators. Three phases take part in non-coherent processing:

- Target detection, data collection and pre-processing.
- Membership declaration.
- Central node election.

Coherent data processing routing: The data is forwarded to aggregators after minimum processing. The minimum processing typically includes tasks like time stamping, duplicate suppression, etc. This is an energy efficient routing.

5.3.5. ROUTING PROTOCOL SIMULATED IN THIS PROJECT: AD HOC ON DEMAND DISTANCE VECTOR (AODV)

In this project the routing protocol used during the simulations is the Ad hoc On Demand Distance Vector (AODV). The choice of this protocol was made after the results obtained in the project, where it was proved that with the simulator NCTUns worked much better with AODV than with DSR.

The Ad hoc On-Demand Distance Vector (AODV) routing protocol is designed for use in Ad-Hoc mobile networks. AODV is a reactive protocol: the routes are created only when they are needed. It uses traditional routing tables, one entry per destination, and sequence numbers to determine whether routing information is up-to-date and to prevent routing loops.

An important feature of AODV is the maintenance of time-based states in each node: a routing entry not recently used is expired. In case of a route is broken the neighbors can be notified.

Route discovery is based on query and reply cycles, and route information is stored in all intermediate nodes along the route in the form of route table entries. The following control packets are used: Route REQuest message (RREQ) is broadcasted by a node requiring a route to another node, Route REPly message (RREP) is unicasted back to the source of RREQ, and Route error message (RERR) is sent to notify other nodes of the loss of the link. HELLO messages are used for detecting and monitoring links to neighbors [43].

5.3.5.1. MERITS OF AODV

- The AODV routing protocol does not need any central administrative system to control the routing process. Reactive protocols like AODV tend to reduce the control traffic messages overhead at the cost of increased latency in finding new routes [43].
- AODV reacts relatively fast to the topological changes in the network and updates only the nodes affected by these changes [43].

- The HELLO messages supporting the routes maintenance are range-limited, so they do not cause unnecessary overhead in the network [43].
- The AODV routing protocol saves storage place as well as energy. The destination node replies only once to the first request and ignores the rest. The routing table maintains at most one entry per destination [43].
- If a node has to choose between two routes, the up-to-date route with a greater destination sequence number is always chosen. If routing table entry is not used recently, the entry is expired. A not valid route is deleted: the error packets reach all nodes using a failed link on its route to any destination [43].

5.3.5.2. DRAWBACKS OF AODV

- Valid routes can expire. Determining of a reasonable expiry time is difficult, because the nodes are mobile, and source's sending rates may widely differ and can change dynamically from node to node [43].
- AODV can gather only a very limited amount of routing information. Route learning is limited only to the source of any routing packets being forwarded. This causes AODV to rely on a route discovery flood more often, which may carry significant network overhead. Uncontrolled flooding generates many redundant transmissions which may cause so-called broadcast storm problem [43].
- The performance of the AODV protocol without any misbehaving nodes is poor in larger networks. The main difference between small and large networks is the average path length. A long path is more vulnerable to link breakages and requires high control overhead for its maintenance. Furthermore, as a size of a network grows, various performance metrics begin decreasing because of increasing administrative work, so-called administrative load [43].
- AODV is vulnerable to various kinds of attacks, because it is based on the assumption that all nodes will cooperate. Without this cooperation no route can be established and no packet can be forwarded. There are two main types of uncooperative nodes: malicious and selfish. Malicious nodes are either faulty and cannot follow the protocol, or are intentionally malicious and try to attack the network. Selfishness is noncooperation in certain network operations, such as dropping of packets which may affect the performance, but can save the battery power [43].

6. SMART CITIES

In a near future, cities will communicate to people and objects that populate it. Future cities could communicate with people, vehicles and urban furniture making life more comfortable to all these agents. These future cities will be smart cities. Traditional smart cities can be identified along six main axes [78]. These axes are: smart economy, smart mobility, smart environment, smart people, smart living and smart governance. These six axes connect with traditional regional and neoclassical theories of urban growth and development. In particular, the axes are based on theories of regional competitiveness, transport and Information and Communications Technology (ICT) economics, natural resources, human and social capital, quality of life, and participation of citizens in the governance of cities [78].

In this chapter a smart city framework is proposed. On it, these six axes are covered. Nonetheless, the focus on this framework is on the Information and Communications Technology. In this smart city framework different Ad-Hoc networks will be implemented. Therefore, these networks are described in this chapter.

6.1. SMART CITY FRAMEWORK

This project has been running in parallel and in the same research group that the project developed by Pablo Regañas Soto [47]. Hence, both projects are related and share this framework.

This project was in charge of the smart city framework development; meanwhile the other project was in charge of the warning message capability design. With the previous projects

joint the goal was to develop a smart city that could be simulated on a network simulator as realistic as possible.

In the proposed framework traditional traffic lights become Intelligent Traffic Lights (ITL). These ITLs can communicate information to passing vehicles (such as weather conditions and traffic statistics) and other Ad-Hoc networks, for instance Wireless Sensor Networks (WSN). At the same time, these ITLs can receive messages from the passing vehicles and collect statistics from these vehicles (such as the traffic density or traffic delay). These ITLs also form a sub-network that allow ITLs to share the information collected by each ITL and create statistics of the whole city. The following sections describe how this smart city is designed and which use the ITLs will have.

6.1.1. SMART CITY DESIGN

Figure 6.1 shows the design of the smart city projected. In this design, blocks have square design and buildings on its four sides. The obstacles in every block shown in Figure 6.1 represent these buildings. The Intelligent Traffic Lights (represented as traditional traffic lights on the Figure 6.1) are responsible to manage the traffic of the vehicles forming a Vehicular Ad-Hoc Network. These ITL do not have to be located on each intersection. Within all the traffic lights that are normally located in a city, only a few will be replaced by Intelligent Traffic Lights. This is because each ITL covers a whole intersection and the 4 streets that converge on the intersection. To cover all this area the antenna pattern used is an omni-directional propagation pattern. Therefore, each ITL receives data from all passing vehicles on their cover range (the four streets and the intersection mentioned). Not having an ITL on each intersection is more economic when implemented this framework. Figure 6.1 also shows where these ITL have to be located. Finally, the represented vehicles are also equipped with wireless communication technology that permits to communicate car to car and car to ITL.

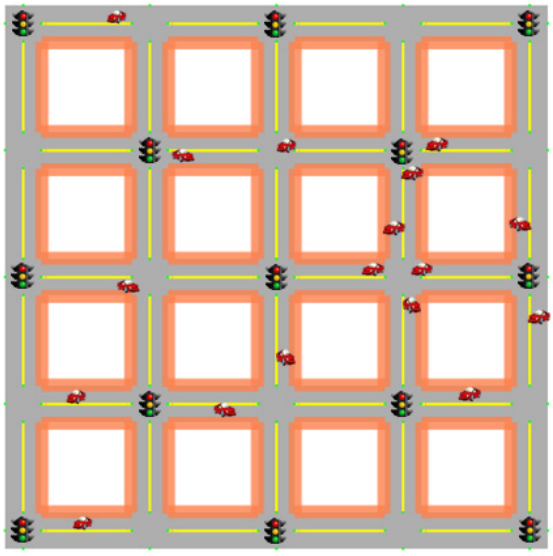

Figure 6.1 - Smart City Design

Every ITL is equipped with two different interfaces, eth1 and eth2; each one of them configured with sub networks operating in Ad-Hoc mode, using AODV as routing protocol, as explained before. The first of them, 1.0.2.X/24 (eth1), used for ITL-ITL communications and the second one, 1.0.3.X/24 (eth2), used for Vehicle-ITL and ITL-Vehicle communications.

As mentioned before, the interface with the 1.0.2.X/24 subnet, eth1, is used by the Intelligent Traffic Light to communicate to other ITLs and share statistics about traffic density and traffic delay. Thus, there could be stored a database with statistics of the entire city. This would allow an ITL to access data that do not belong to its own coverage area and send it to a vehicle, for example.

The other interface (eth2) configured with the 1.0.3.X/24 subnet is used by both, vehicles and ITLs to send or receive, respectively, data associated with statistic gathering such as traffic density or traffic delay. In the future more statistics could be shared using this interface. The subnet configuration and distribution are shown in Figure 6.2.

Also in Figure 6.2 it is shown how an ITL (Intelligent Traffic Light) does not mean that the regular traffic lights on every converging street to the intersection are removed. In fact, the design implies that an ITL gathers the four regular traffic lights of the intersection and associate them to the same IP address. This will ensure traffic control and gathering traffic statistics.

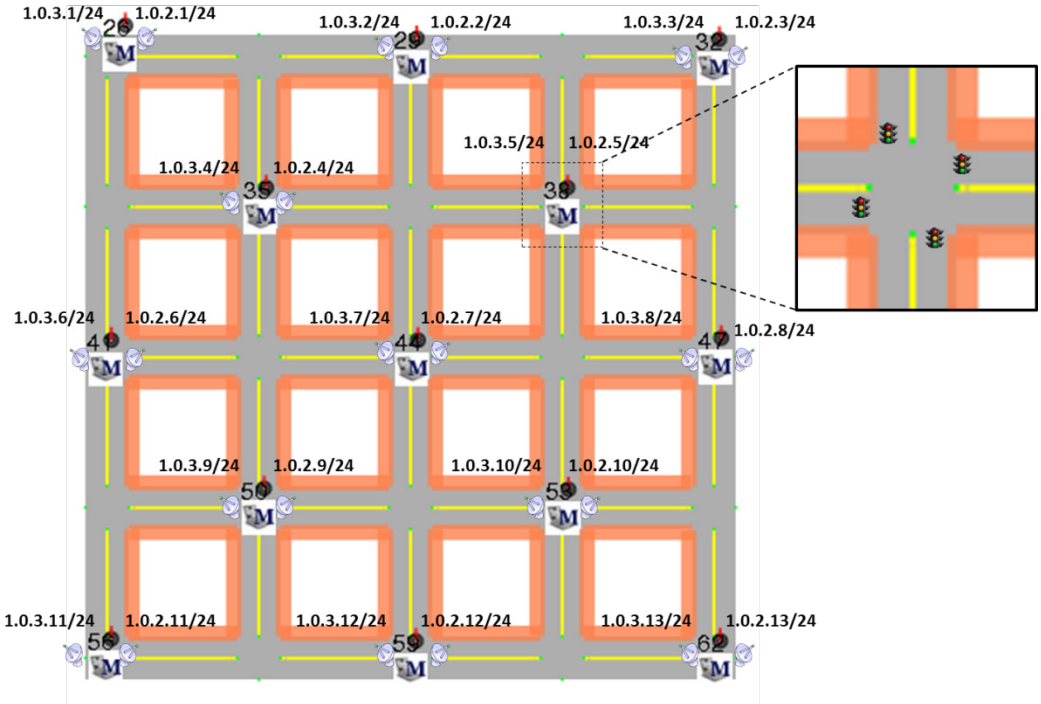

Figure 6.2 - Smart City Sub Network Configuration

In Figure 6.3 is shown the Intelligent Traffic Light distribution across the city and the ID associated with each of them depending of its position inside the city.

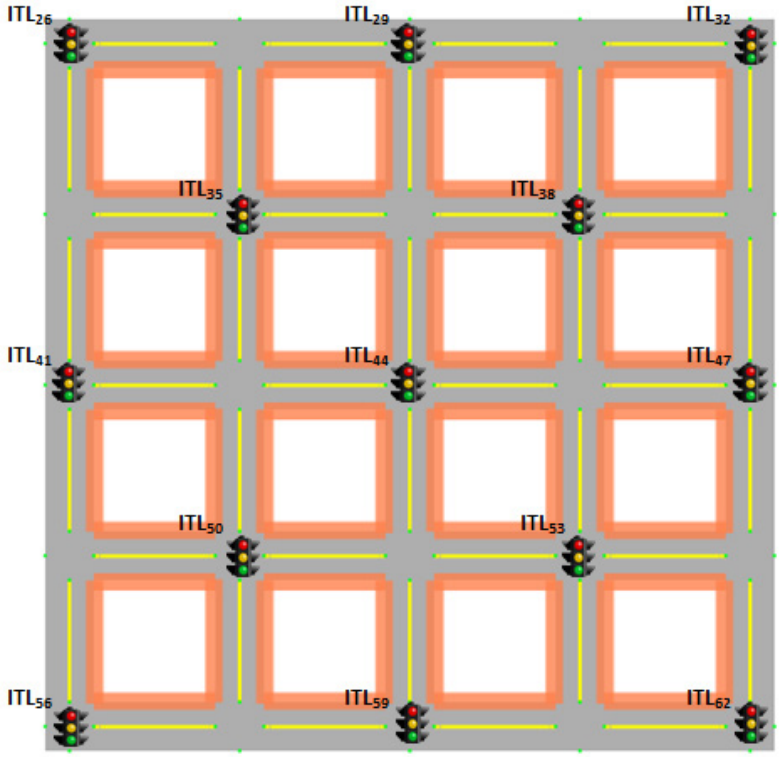

Figure 6.3 - ITL Deployment on the Smart City

The city design was inspired by the city of Barcelona, where the project had been developed. For this reason the length of streets was 100 meters and the length of intersections was 40 meters. The measures described before could be seen in Figure 6.4.

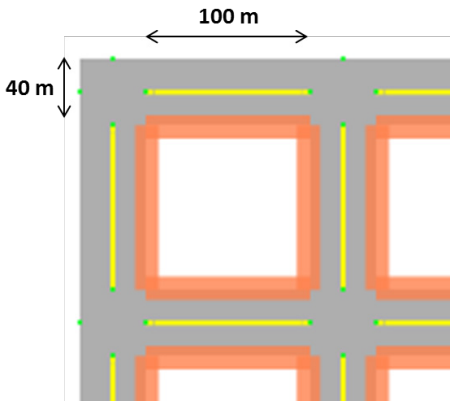

Figure 6.4 - Smart City Measures

Knowing the distance of the streets and intersections that conforms the city it was possible to determine the distance between adjacent ITLs, or the maximum coverage area desired for an ITL.

By design it was established that a single ITL covered the four converging streets on its own intersection. Besides, it was already known that the street measures were 100 meters, so if the ITL was placed in the middle of the intersection to cover the farthest place of the street it

would need a coverage area of 120 meters (as shown in Figure 6.5). It was decided to give an extra margin for the proper system operation, so the radius of coverage of the interface eth2 was set on 130 meters with an omni-directional propagation pattern.

Using the same logic it was determined that adjacent ITLs were separated two complete streets, two complete intersections and two half intersections (as shown in Figure 6.5), giving a total of 280 meters. In this case the radius of coverage of the interface eth1 was set on 300 meter, also with and omni-directional propagation pattern.

In Figure 6.5 it is explained graphically how the coverage areas were estimated. On red colour it is shown the interface eth1 (300 meters of coverage) of the ITL1. It could be appreciated that this interface could communicate without any problem with the similar one of the ITL2 because they have direct line of sight (Assuming all the ITLs were configured equally, as they were). This interface (eth1) of the ITL1 could not communicate with the interface of the ITL3, because of the obstacles, represented by orange walls, between them. Also it could be appreciated from the Figure 6.5 how the interface eth2 (130 meters of coverage) of the ITL3 covers the four streets that converge on its intersection, so it could establish communication with any vehicle that is on its surrounding area.

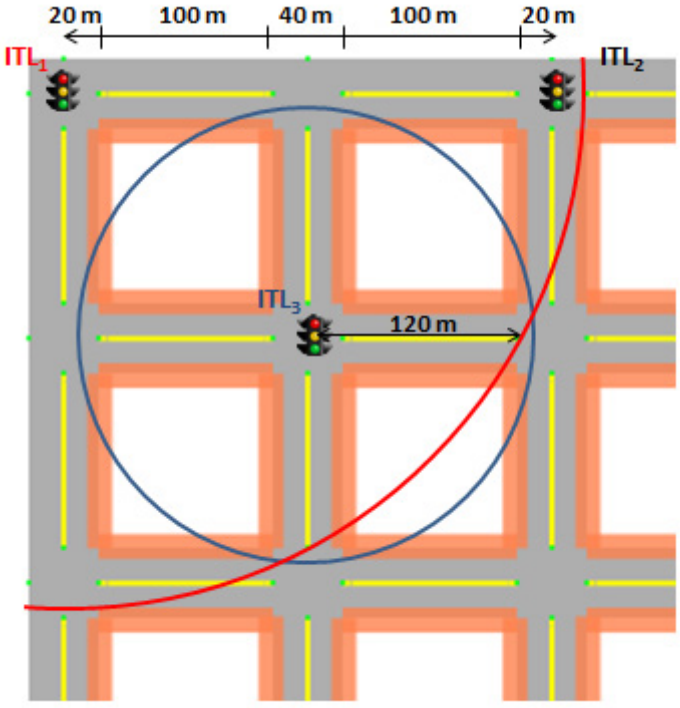

Figure 6.5 - Interfaces Coverage Area on ITLs

The vehicles on the smart city were represented using *ITS Cars* equipped with an 802.11 b interface on Ad-Hoc mode. This car has the characteristic that is controlled by a mobility agent that moves it through the city respecting the streets, crosses and traffic lights. It was configured with AODV routing protocol, modified to collect statistics. As this car is only

supposed to connect with interface (eth2) of the Intelligent Traffic Light and other surrounding cars, it was configured to cover 130 meters in an omni-directional pattern.

6.1.2. MANAGEMENT OF TRAFFIC DENSITY

The smart city framework includes ITLs set in the crossroads. These ITLs collect real-time traffic data from the passing vehicles and calculate traffic statistics such as the vehicles density. The way each ITL collects the traffic density of its area is as follows. Each vehicle travelling on the city counts the number of neighbors (close cars) every two seconds and sends this number to the closest ITL. Then, the ITL update its statistics properly, calculating the average number of neighbors per vehicle. With this average number, two adaptive thresholds will determine the traffic density of the area.

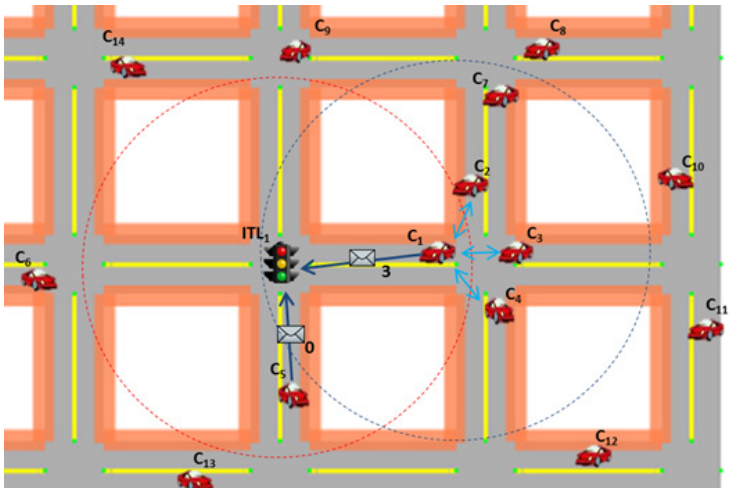

Figure 6.6 - Traffic Density Statistics Gathering Process Example

On Figure 6.6, an example of traffic density gathering is shown. In this example, vehicle C_1 counts three neighbors (C_2, C_3, and C_4). Although C_7 is inside its cover range they cannot establish any communication because buildings represented by obstacles on the simulation block the communication. On the other hand, vehicle C_5 does not have any neighbor around so it sends a zero to the same ITL (the closest one).

6.1.2.1. SENDING TRAFFIC STATISTICS (VEHICLE-ITL)

The first step on traffic statistics gathering is the one that perform every vehicle every two seconds. This step will consist of taking advantage of the routing protocol characteristics, in this case AODV.

The first thing that must be taken into account is that AODV is a reactive protocol; this means that only needed routes will be created and stored among the routing information. This

information consist on a neighbour list, where, as its name says, are stored all the neighbors of the node for a particular moment.

As we are working with Vehicular Networks (VANETs), the nodes have the capability to move around the scenario changing the topology, breaking existing routes and creating new ones when need it. For this reason AODV protocol must be adaptive. To achieve this goal AODV associates every new entry on the neighbor list with and expiration time, assuring that not recent entries will expire.

Knowing the protocol functioning it is now possible to explain how the nodes (vehicles) count their surrounding neighbors, taking advantage of the routing protocol they are already using. This process consists in the following steps:

- Counting Neighbors: The node (vehicle) consults its neighbor list and counts every entry which has an ID that belongs to a vehicle (not being itself) and an expiration time greater than the time the consultation is made, assuring the route is valid. The vehicle will only count the entries that belong to the same subnet, in this case the VANET subnet.
- Determining Destination: The node (vehicle) consults on its neighbor list all the entries with an ID that belongs to an ITL, comparing their expiration time. The vehicle will only get the IP address of the ITL with the biggest expiration time, but ensuring this time is greater than the consulting time, to assure it is the nearest Intelligent Traffic Light.
- Creating and Sending Stat Message: Once the number of neighbors and the IP address of the nearest ITL are known, the vehicle assembles a STAT_msg with the structure of the Table 6.1. Finally the message in sent to the nearest Intelligent Traffic Light.

STAT_msg			
Field	**Description**	**Data Type**	**Default Value**
Type	Statistic Message	Char	"AODV_STAT"
stat_type	Traffic Density	Int	0
	Traffic Delay		1
stat_my_id	Car Sending Statistics	Int	C_i
stat_neighbors	Number of Neighbors (NoN)	Int	NoN_i
stat_time	Moment of Computation	u_int64_t	t_i
stat_dst	ITL IP Address	u_long	ITL_i
stat_delay	Delay Time	u_long	Del_i

Table 6.1 - Stat Message "Stat_Msg" Configuration

The type of the message is used by AODV to differentiate this type of message from the rest, such as "AODV_RREQ", "AODV_RREP" or "AODV_RERR". The stat_type is a field that differentiate the statistic that contains the message, it could be traffic density if this field is set to zero, or traffic delay if is set to one. In the future there could be more statistics added to this type of message. The stat_my_id field contains the ID of the vehicle who sends the statistics. The stat_neighbors contains the number of surrounding neighbours the vehicle had on that determined time; if the Stat message is a traffic delay type this field is set to zero. The stat_time field contains the moment in which the statistic was computed. The stat_dst field

contains the IP address of the nearest ITL. Finally, the stat_delay contains the value of time the vehicle expend from the beginning of the street to the end of it; if the Stat message is a traffic density type this field is set to zero.

This process is repeated by every vehicle every two seconds. This time was set considering the maximum speed of the cars, the length of the streets and the number of average statistics message sent from each vehicle. First it was decided by the workgroup that every vehicle at least should send five different traffic density messages on every street. Assuming the worst case scenario, which is when the vehicle always travels with the maximum speed, the time in which the process would be repeated, was calculated. The maximum speed by default on NCTUns ITS cars is 10 m/sec and the street length is already known (100 meters).

$$t_{minimum} = \frac{100\ m}{10\ m/sec} = 10\ [sec] \qquad \text{(Eq. 6)}$$

$$t_{repetition} = \frac{10\ sec}{5\ messages} = 2\ [sec/message] \qquad \text{(Eq. 7)}$$

6.1.2.2. RECEIVING TRAFFIC STATISTICS (ITL-VEHICLE)

The only nodes that could receive traffic statistics would be the ITLs, and only using the interface eth2 which belongs to the VANET sub network (1.0.3.X/24). This is because it was designed on the sending process that the destination must be an IP address associate with an ITL and also that belongs to the VANET.

This fact allows ensuring that no one but an ITL is going to receive statistic messages. Thus, the process that manages the statistical data processing was designed taking into consideration that only the ITLs will execute it. The steps of the process are as follows:

- Receiving: First the ITL receives a statistic message and differentiate which data contains (traffic density or traffic delay).
- Updating Statistics: Once was checked the message contained traffic density information the ITL should update the stats. As this step may be extensive it is going to be divided in sub steps:
  - Once the ITL knows the instant value that a vehicle sends it, the first thing that should be done is checking if there is any previous information or statistic (historical value).
  - The ITL will update and store the statistic using an Exponential Weight Mean Average (EWMA). This, consist on a low pass filter which, depending on the value of Alpha (α ϵ [0,1]) it gives more weight to the historical value (when closer to zero) or the instantaneous one (when closer to one). The EWMA equation is shown in the following formula, where N_i represents the historical value of the statistics in the time *i*:

$$EWMA:\ \overline{N_i} = \overline{N_{i-1}} * (1-\alpha) + N_{instant} * \alpha \qquad \alpha \in [0,1] \qquad \text{(Eq. 8)}$$

- The ITL could calculate the statistics using different values of Alpha and store the results. One of them could be used for historical stats purposes and the other one to check the traffic density on that instant moment (possible application).
- Once the calculations are done the ITL store the information and waits to receive another statistic message.

- Sharing Statistics: The ITL periodically shares the information about traffic density statistics of its own are with the other ITLs using the interface eth1 that belongs to the subnet of ITLs.

The day has been divided into five periods due to the usually variable traffic densities in a city throughout the day. Thus, every ITL updates the traffic density per periods: $TDst_{6\text{-}9}$, $TDst_{9\text{-}12}$, $TDst_{12\text{-}15}$, $TDst_{15\text{-}18}$, $TDst_{18\text{-}21}$. For instance, $TDst_{6\text{-}9}$ gathers the average traffic density in the city, during week days, from 06:00 AM to 09:00 AM. The value $TDst_{6\text{-}9}$ will continuously be updated using equation 9 with a small α value to smooth out isolated deviations. ~~$TDst_{6\text{-}9,i}$~~ is the updated average in iteration *i* and $TDst_{6\text{-}9}$ is the last value received by that ITL. The same computation will be done for the other periods of the day.

$$\overline{TDst}_{6-9,i} = \overline{TDst}_{6-9,i-1} * (1-\alpha) + TDst_{6-9,i-1} * \alpha \qquad \alpha \in [0,1] \qquad \text{(Eq. 9)}$$

6.1.3. MANAGEMENT OF TRAFFIC DELAY

As said before, ITLs collect real-time traffic data from the passing vehicles and calculate traffic statistics. In addition to the traffic density statistics the ITLs also collect traffic delay statistics. The traffic delay consists on the time the vehicle expends travelling a whole street from beginning to the end. It would be a statistic directly related with traffic density and a way to the determine navigation routes for the travel assistants in a near future.

To implement traffic delay statistics on NCTUns it was discussed with the working group a general idea on how it must function. This basic idea is shown in Figure 6.7 and explained below:

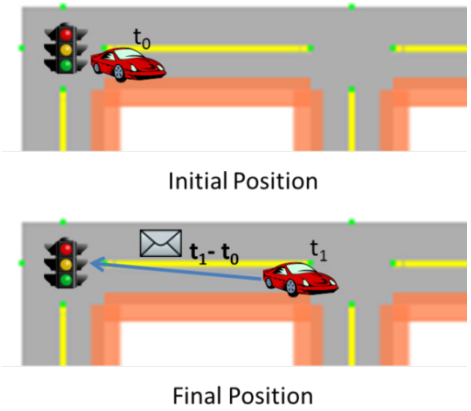

Figure 6.7 - Traffic Delay Gathering Original Idea

As seen in Figure 6.7 the idea consists that the vehicle knowing its position inside the city (a driver in the real world know where he is) it could calculate the time difference between the beginning of the street and the end of it. For example, in Figure 6.7 the initial position it is associated with an initial time t_o, meanwhile it is also associated a final time t_1 to the final position. As the vehicle is still inside the cover range of the ITL at the end of the street, it would send to the nearest ITL the statistic information, in this case the delay time that it expended travelling the street. To accomplish this idea inside the simulator it is important to know the position of every vehicle the whole simulation time, so it could be possible to know when they are on a beginning of a street or in the end of it. Therefore, it must be known and understood how the vehicles move around the city during the simulation.

In NCTUns VANET vehicles are represented with ITS Cars. These nodes have an important characteristic: when they are deployed inside streets they can move randomly according to a mobility agent. There are different mobility agents in NCTUns to simulate different behaviours of the vehicle during simulations, but in this project it is going to be studied and modified the "CarAgent.cc" agent.

This agent is responsible of moving all the vehicles around the city taking in count street turns, traffic lights, other cars and accidents. Hereby, this agent must modify every vehicles velocity depending on the surrounding situation of each one of them, besides it must take the decision of which street it is going to be the following, when a vehicle reaches an intersection.

As the "CarAgent.cc" had total control over the mobility model of the vehicles, it was concluded that this agent must be an important part for implementing traffic delay statistics. The fact that the agent knows when the decision of turning is made, gives the possibility to associate a time with this process and subtract it with the previous turning decision. Therefore the implemented idea it is shown on Figure 6.8.

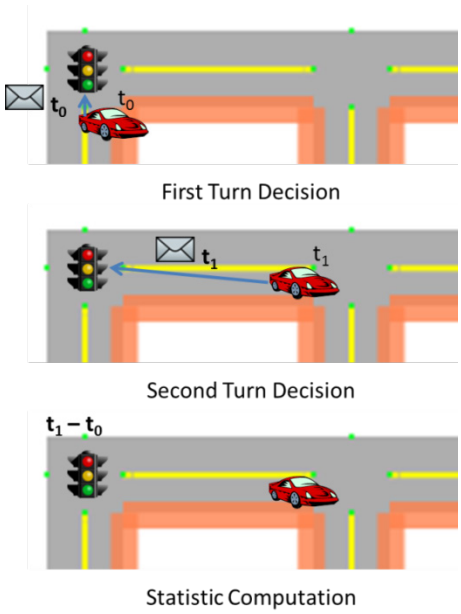

Figure 6.8 - Traffic Delay Idea Implemented

The first thing which could be appreciated from Figure 6.8 is that the vehicle does not send a time difference anymore, it only sends the time when a turn decision is in progress. Now the time difference is computed by the ITL, so it is responsible to subtract consecutive times of the same vehicle. Now, it is fundamental to differentiate messages by vehicles so it could be possible to efficiently calculate delay statistics for every vehicle.

It is important to highlight that the original idea for traffic delay could be implemented in real life, because the vehicle would have all the tools necessary to compute this time difference on its own, and only send the statistic to the ITL. The vehicle would have a considerable processing power, would not have any energy constraint and could base the location of the beginning or the end of a street on its travel assistant.

6.1.4. MANAGEMENT OF WARNING MESSAGES

Warning Messages management was designed and implemented by Pablo Regañas Soto on his Career Final Project named “Study of Vehicular Networks in Urban and Interurban Scenarios” [47].

The statistics about traffic density described in the section above are used when creating a new type of message named “warning message”. Warning messages will contain information about traffic density and weather conditions. Thus, data referring to traffic density is gathered using the traffic statistics collected by the ITLs. In case of accident, the vehicle itself communicates this circumstance to the ITL, using its own sensors. Regarding to weather, data is gathered by a Wireless Sensor Network (WSN) that periodically transmits it to the ITL. In case that the WSN is not operative, the forecast proportioned by local public weather services could be used.

7. TOOLS USED IN THIS PROJECT

This project consists on a VANET simulation environment in which a statistical acquisition and processing system will be implemented and tested. The network simulator used for this purpose is NCTUns (National Chiao Tung University Network Simulator) on its 6.0 version.

NCTUns 6.0 is free and open-source code software. It has a core technology based on the novel kernel re-entering methodology invented by Prof. S. Y. Wang when he was pursuing his Ph.D. degree at Harvard University [79]. Due to this novel methodology, NCTUns provides many unique advantages that cannot be easily achieved by traditional network simulators.

This network simulator needs UNIX operating system to be used. NCTUns 6.0 is usually run with Fedora 11 operating system. In this project the installation of the referred operating system in a personal computer (PC) was avoided and a virtual machine was created to run both the operating system and the network simulator.

A virtual machine (VM) is a software implementation of a computer that executes programs like a physical machine [80]. The VM used in this project is a system virtual machine. A system virtual machine provides a complete system platform which supports the execution of a complete operating system (OS), Fedora 11 in this case. An essential characteristic of a virtual machine is that the software running inside is limited to the resources and abstractions given by the virtual machine. Thus, a VM allows the user to execute an operating system different the host machine presents, without installing it.

In order to create and use a virtual machine a VM Player is required. There are many virtualization software like VirtualBox [81], or VMWare. In this project the used software for virtualization was VMWare Player [82].

In this part of the project, the software used, as well as the installation steps, are described and detailed below.

7.1. NETWORK SIMULATOR (NCTUNS 6.0) CHARACTERISTICS

As said, NCTUns is a high-fidelity and extensible network simulator and emulator capable of simulating various protocols used in both wired and wireless IP networks.

NCTUns uses a novel kernel re-entering simulation methodology and as a result it provides several unique advantages traditional networks simulators cannot achieve easily.

Traditionally, simulators are limited by their reduced resources. Simulators used to present simplified versions of the results compared to real devices, and protocols were simulated with few details to reduce complexity and cost of their developments.

In addition, applications must be written to use the internal Application Programming Interface (API) of the simulator. Otherwise it has to be re-compiled for the simulator, creating a big and complex program. To overcome these two limitations, NCTUns propose a new simulation method to re-enter into and modify the UNIX kernel. Using this method, a real implementation of the protocol stack is created, and the simulator can provide more realistic results. This means that protocols from the simulated network are really executed by the UNIX kernel (and not by the simulator) and that NCTUns will work as an emulator too.

As a free software open-source code new applications can be created, as done in this project. Furthermore, objects simulated by NCTUns are not contained in a unique program, but they are distributed in multiple and independent components that run concurrently in a UNIX machine, improving the written code efficiency.

Finally, NCTUns allows attenuation and bandwidth measurements in any type of network, and can also simulate several types of under-development networks, such as optical networks.

7.1.1. SYSTEM REQUIREMENTS

To install and use NCTUns 6.0 some requirements in terms of hardware and software have to be satisfied. These requirements are shown in the table below (Table 7.1):

Operating System	Hardware	Software
➢ Fedora 7.0	➢ 1.6 GHz Processor ➢ 256 Mb RAM ➢ 300 Mb Hard Disk	➢ gcc Compiler ➢ Administrator log-in

Table 7.1 - Hardware and Software Requirements on NCTUns

In this project, the computer used for simulations had the following characteristics:

- Processor: Intel® Core™ i3 CPU M 330 @ 2.13 GHz
- RAM Memory: 4.0 GB (3.87 GB usable)
- Hard Disk Free Memory: 260 GB

When running the virtual machine for the first time, it must be chosen the amount of RAM memory dedicated to the VM. In this case 2 GB were chosen as recommended by the virtualization software.

7.1.2. ARCHITECTURE AND SIMULATION METHODOLOGY OF NCTUNS

This chapter is based on [83]. NCTUns is mainly composed of six components, shown on Fig. 7.1 below:

- Graphical User Interface (GUI)
- Dispatcher
- Coordinator
- Simulation Engine
- Application Programs
- Patches to the kernel Transmission Control Protocol (TCP) / User Datagram Protocol (UDP) / IP Protocol stacks

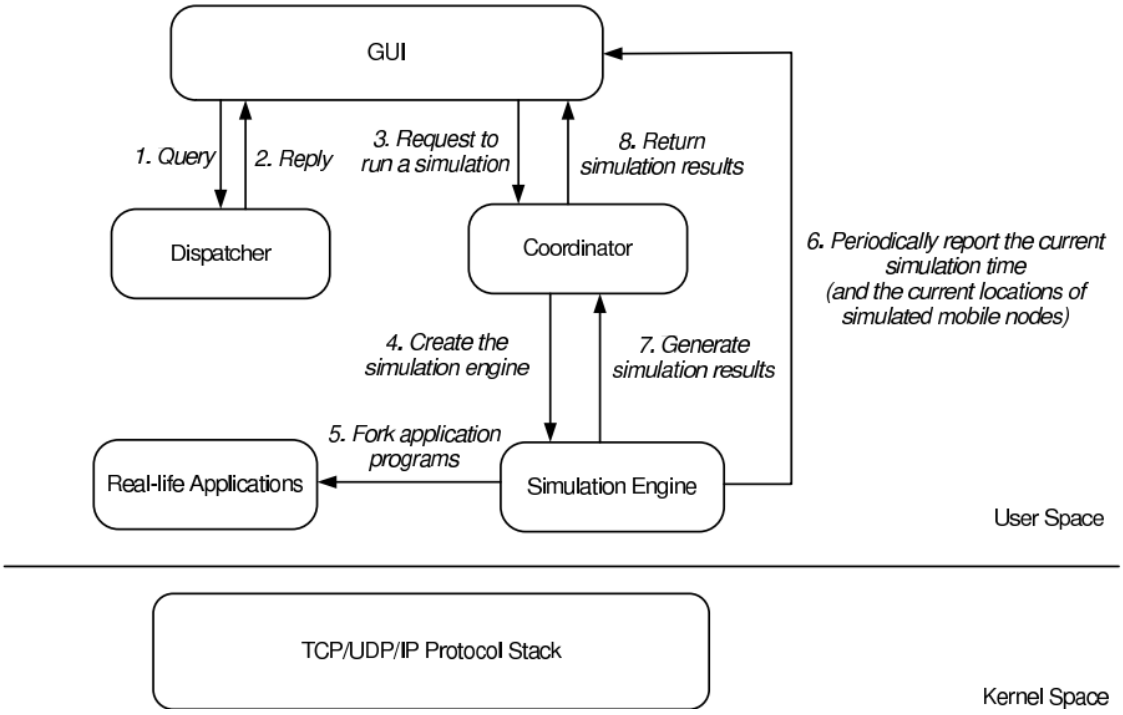

Figure 7.1 - NCTUns Architecture [83]

The main functions of the components listed above are:

Graphical User Interface (GUI)

NCTUns provides a front-end GUI program called "NCTUnsclient" in its package, which provides useful facilities for users to efficiently create simulation and emulation cases. According to user's common needs, it groups the operations of generating a simulation / emulation case into fours modes that will be described in another section below:

- Draw Topology
- Edit Property
- Run Simulation
- Play Back

Dispatcher

NCTUns provides flexible simulation architecture, by which the GUI program and the simulation engine program can be run on different machines. In NCTUns, the GUI program need not find a simulation server to run the simulation engine program for a simulation. Instead, it sends the Dispatcher program an inquiry message to know which simulation server is currently available. The Dispatcher program is responsible for monitoring the statuses of the simulation servers that it manages and selecting an available simulation server (if one exists) to serve the simulation request issued from the GUI program.

Coordinator

The Coordinator program has the following four tasks:

- Processing the commands sent from Dispatcher.
- Creating a simulation engine process to perform a simulation.
- Reporting the status of the created simulation engine process to the Dispatcher program.
- Collecting the simulation results produced by its created simulation engine process and sending then to the GUI program.

Before starting any simulation on a simulation server, a Coordinator program should be run up.

Simulation Engine

It is composed of a set of various protocol modules and an event scheduler. The former is responsible for simulating protocol behaviours while the latter is responsible for scheduling events in a non-decreasing order based on their timestamps. In addition, during simulation the simulation engine process will periodically report the current simulation time to the GUI program.

Application Program

Application programs are responsible for generating network traffic in a simulated network. Most real-life application program can be directly run up on a node simulated by NCTUns to generate realistic network traffic.

Kernel Patches

NCTUns uses the real-life Linux network protocol stack to "simulate" transport-layer and network-layer protocols, such as TCP, UDP and IP. Minor modifications to Linux kernel timers are required so that the timers used by the in-kernel protocol stack of each simulated node can advance their times based on the simulated clock (controlled by NCTUns) rather than the real-world clock.

7.1.3. SEAMLESS INTEGRATION OF EMULATION AND SIMULATION

NCTUns can be turned into an emulator easily. In emulation, nodes in a simulated network can exchange real packets with real-world machines via the simulated network.

That is, the simulated network is seamlessly integrated with the real-life network so that simulated nodes and real-life nodes can exchange their packets across the integrated simulated and real-life networks. This capability is very useful for testing the functions and performances of a real-life device, such as a Voice over Internet Protocol (VoIP) phone, under various network conditions. In NCTUns emulation case, an external real-life device can be a fixed host, a mobile host, or a router.

NCTUns supports distributed emulation of a large network over multiple machines. If the load of an emulation case is too heavy so that it cannot be carried out in real time on a single machine, this approach can simultaneously use the Central Processing Unit (CPU) cycles and main memory of multiple machines to carry out a heavy emulation case in real time.

7.2. NCTUNS 6.0 INSTALLATION

In this project, NCTUns 6.0 is run by a virtual machine installed in a Personal Computer (PC) with Windows 7 Professional operating system. The steps followed to install the virtual machine in this project were:

Virtualization Software Download (VMWare Player)

The first step was to download the virtual machine VMware player, free desktop application that allows running multiple operation systems at the same time on a PC. VMware Player can be downloaded for free in the VMware website [84].

To install VMware, getting starting guide (Chapter 4) provided by VMware provides all information needed. This guide can be found in the website of VMware [85]. Once this software was installed, an icon of VMware appeared in the desktop, allowing running the software.

Fedora 11 Virtual Machine with NCTUns 6.0 Download

The second step was to download the virtual machine. The virtual machine was created by Daniel Navarro, and was downloaded from a server of "Departament d'Enginyeria Telemàtica" (Entel). The location of this virtual machine was [86].

This virtual machine is a Fedora 11 image (Fedora is an operating system based on a Linux Kernel, and it is the official operating system supported by NCTUNS 6.0). Once the virtual machine was downloaded, it can be run with VMware player.

Run Virtual Machine

After downloading the virtual machine, it was unzipped (File can be unzipped with the free software WinRAR [87]). Therefore, the virtual machine was ready to be executed through the virtual machine player of VMware.

To play the virtual machine, the player must be opened. Once played, some advices about the compatibility of the virtual machine and the current operating system were given by VMware player. They were accepted.

Once the virtual machine is played, a GNU GRUB (GRand Unified Bootloader) appears in the window. A GNU is the reference implementation of the Multiboot Specification, which provides a user the choice to boot one of multiple operating systems installed on a computer or select a specific kernel configuration available on a particular operating system's partitions. The kernel configuration that has to be chosen is "NCTUNS". The user name and password are shown in Table 7.2:

User	nctuns
Password	nctuns

Table 7.2 - VM Session User and Password

7.3. STEPS IN SIMULATIONS

In order to simulate different VANET in this project, the first thing to do is run NCTUns. To execute NCTUns 6.0, a command terminal has to be opened.

NCTUns can only be run with root privileges. To obtain root privileges it have to be introduced the "$su" command in the terminal and then the password that is "admintid33", as shown in the following figure (Figure 7.2):

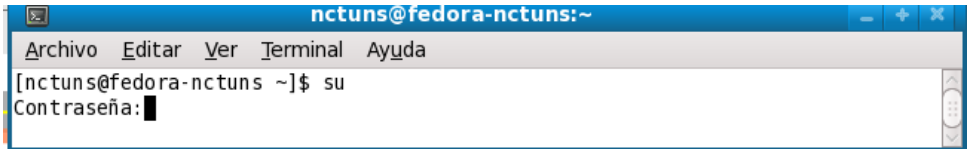

Figure 7.2 - "su" Command to Obtain Root Privileges

To run NCTUns in order to simulate a VANET, three of the components of NCTUns have to be run. These are the dispatcher, the coordinator and the NCTUnsclient (GUI). To execute these components, the following commands have to be introduced in the commands terminal:

- #dispatcher&
- #coordinator&
- #nctunsclient&

An example of this commands being executed on a terminal are shown in the following two figures: Figure 7.3 and 7.4 respectively. On those images root privilege session is initialized and then NCTUns in run.

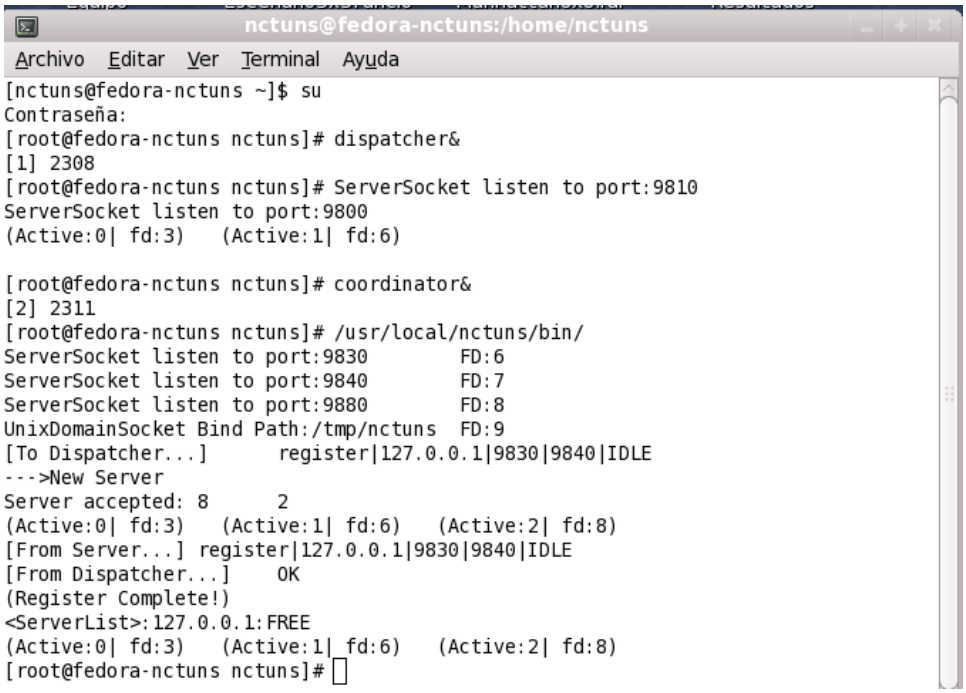

Figure 7.3 - NCTUns Initiation by Terminal

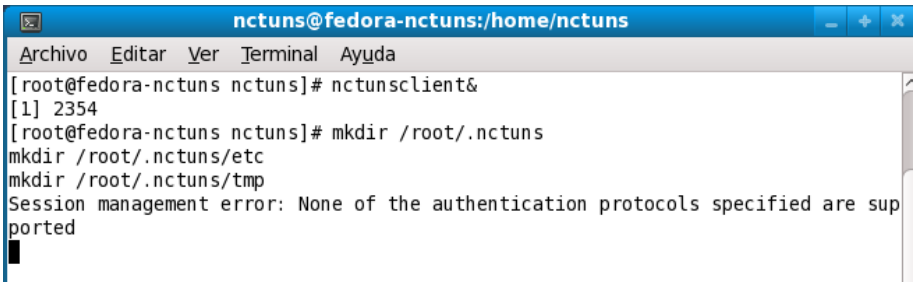

Figure 7.4 - NCTUns Initiation by Terminal

Any simulation generated with NCTUns has to complete four stages. Each stage has to be done with each mode of the GUI that would be described in the next sections.

7.3.1. DRAW TOPOLOGY

In this mode, one can insert network nodes, create network links, and specify the locations and moving paths of mobile nodes. In addition, the GUI program provides a complete tool kit for users to construct road networks, which is fundamental to wireless vehicular network simulations, where many Peer to Peer (P2P) researchers are proposing to run P2P applications. A screenshot of the draw topology mode it is shown on Figure 7.5.

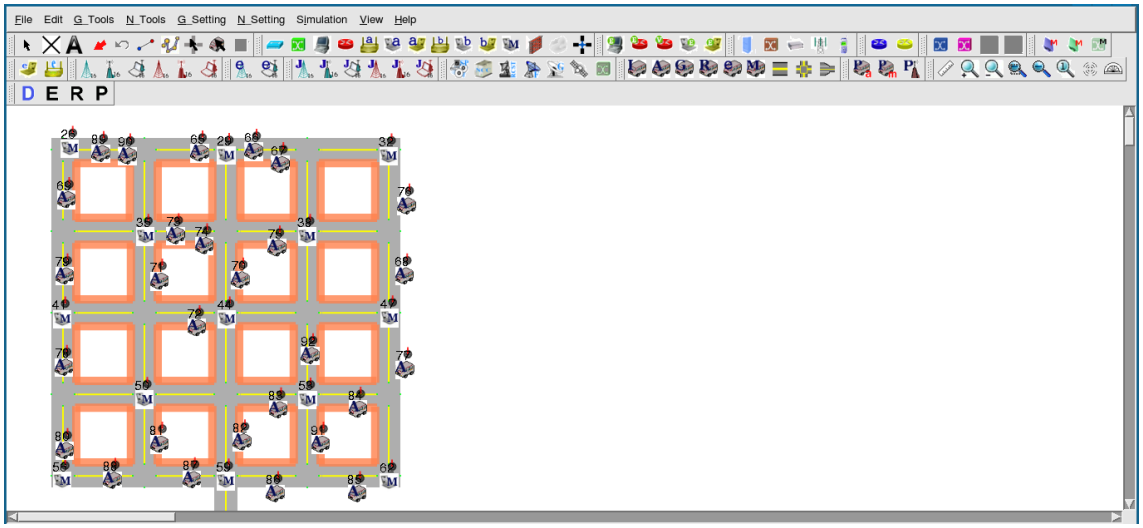

Figure 7.5 - NCTUns Draw Topology Mode Screenshot

7.3.2. EDIT PROPERTY

In this mode, the icon of a network node can be double-clicked to configure its properties (such as, the network protocol stack used in this node, the applications to be run on this node during simulation as seen on Figure 7.6, the physical layer and channel model parameters as seen on Figure 7.7 and other parameters).

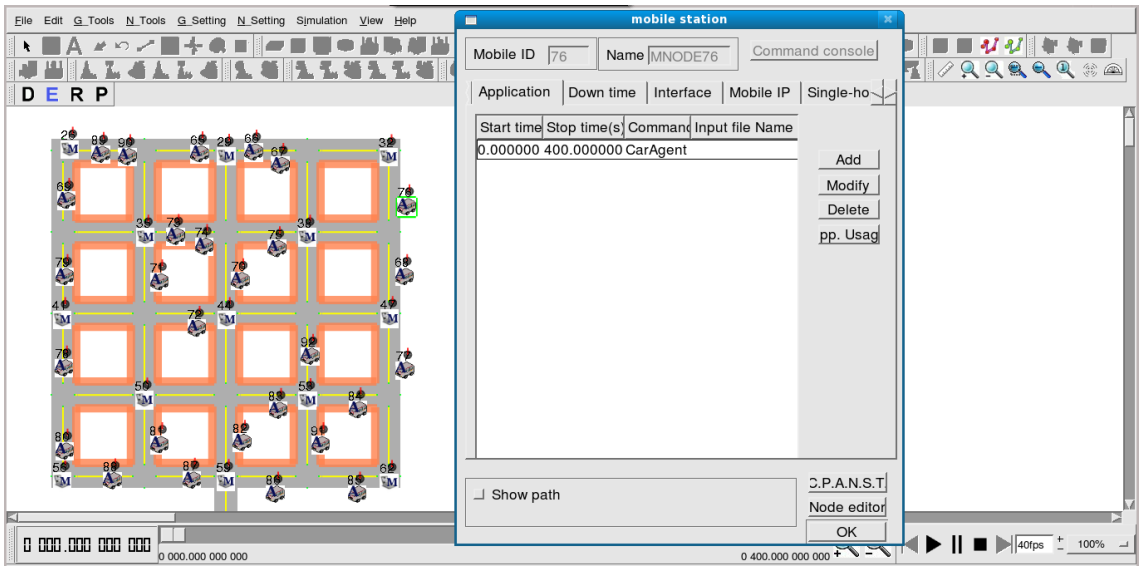

Figure 7.6 - NCTUns Edit Property Mode Screenshot – Mobile Station Edition

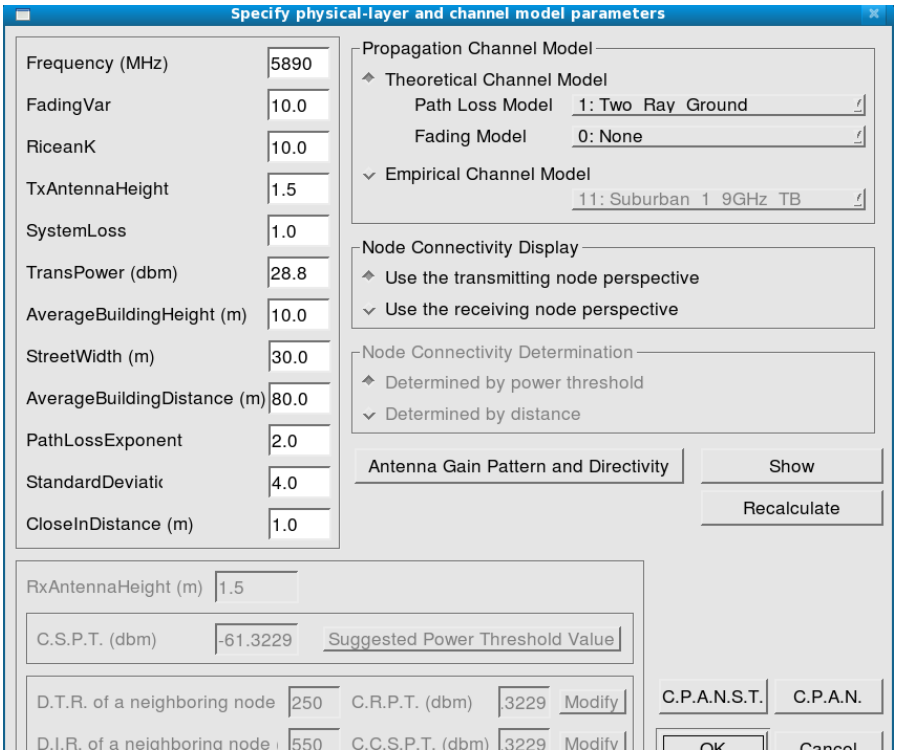

Figure 7.7 - NCTUns Edit Property Mode Screenshot – Physical layer and Channel Model Parameters Edition

7.3.3. RUN SIMULATION

In this mode, the GUI program provides users with a complete set of commands to start/pause/stop a simulation. The progress of a simulation can easily be controlled by simply pressing a button on the GUI control panel. This mode it is shown in Figure 7.8.

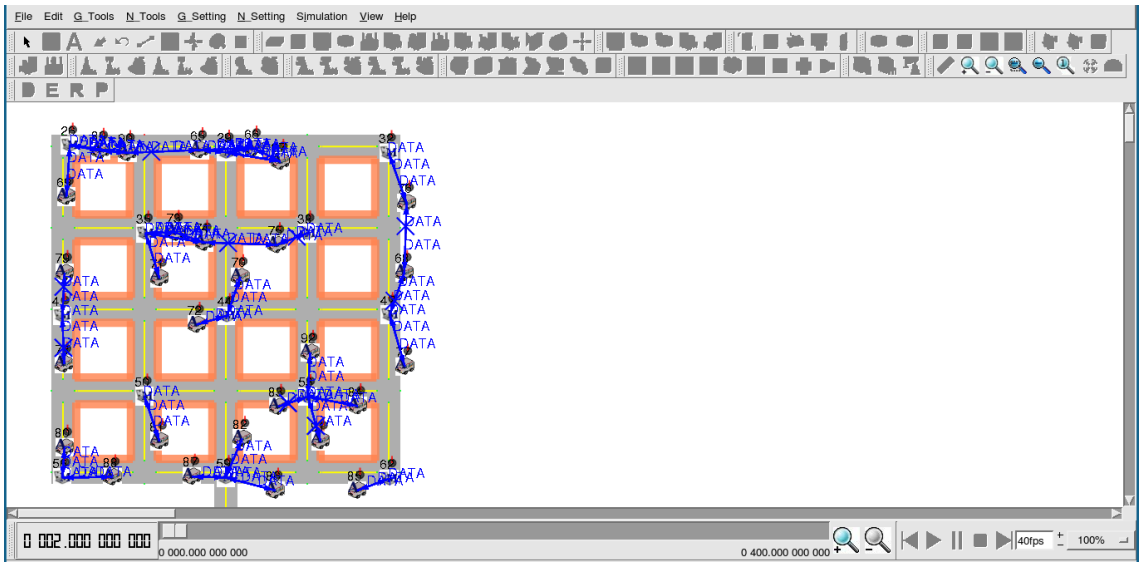

Figure 7.8 - NCTUns Run Simulation Mode Screenshot

7.3.4. PLAYBACK

After a simulation is finished, the GUI program will automatically switch itself into the "Play Back" mode and read the packet trace file generated during the simulation. In this mode, one can use the GUI program to replay a node's packet transmission/reception operations in an animated manner. This mode it is shown in Figure 7.9.

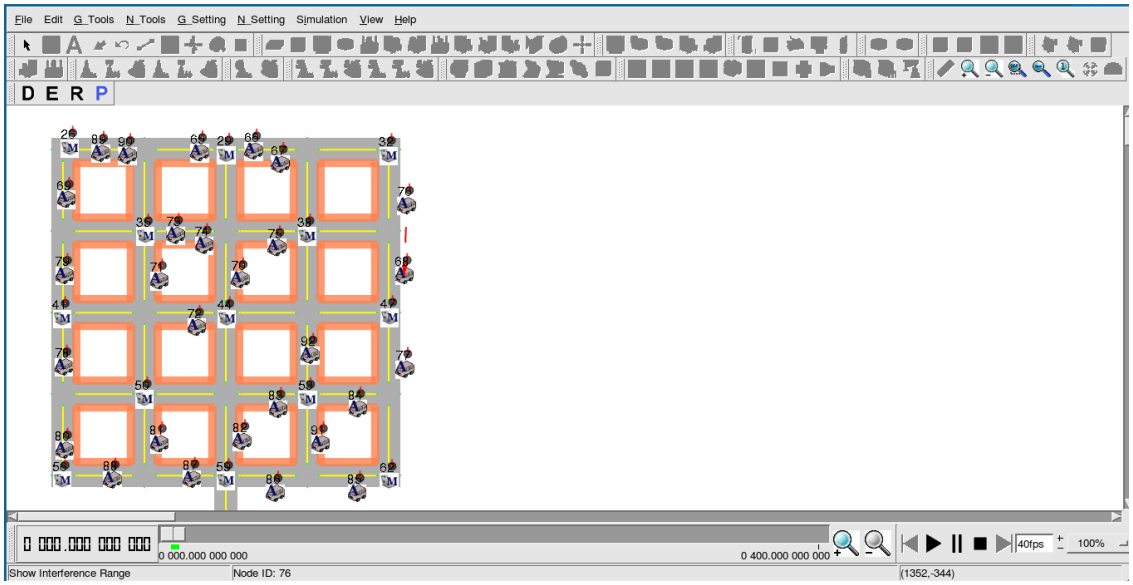

Figure 7.9 - NCTUns Play Back Mode Screenshot

8. CHANGES MADE IN NCTUNS FOR THIS PROJECT

NCTUns is an open source software that allows developers to modify its code in order to add new functions and capabilities to the simulator. Therefore, different NCTUns users can create new functions and protocols that the simulator engine will use on its simulations.

The main objective of this project is not only to simulate a VANET but to modify the NCTUns source code to add new possibilities that make the simulations with NCTUns more realistic. Thus, in this project four modules of the simulator have been modified, three of them related to the AODV routing protocol and the last one related to the mobility agent described before. The modified modules were: “AODV.cc”, “AODV.h”, “AODVrt.cc” and finally “CarAgent.cc”. The first three modules were allocated in the following directory “NCTUns-6.0/src/nctuns/module/route/aodv”, while the last module was allocated in the next directory “NCTUns-6.0/tools/tacticMANET/lib”.

With these modifications AODV is able to support, from the nodes running this protocol, calculation and processing of traffic statistics, such as traffic density (known from the number of neighbors around a determined node) or traffic delay (time between consecutive turn decisions). Every node running AODV as a routing protocol will be able to build up traffic statistic (vehicles) or process it (ITLs).

On the VANET scenario design process it was decided that traffic statistic collection should rely on every vehicle that forms part of the network, and leave to a central device the statistic processing. Naturally, with one central processing device was difficult to cover an entire city, so it was decided that a processing node will cover a determined area, and it should be calculated the amount of processing devices that should be deployed in the city to cover it. The best solution was to take advantage of the fact that in every street intersection were four

traffic lights and grant this structure the capability of data processing, obtaining as a result an Intelligent Traffic Light (ITL).

In this chapter is going to be explained how modules have been modified to obtain the results expected as well as the files created in the whole process.

8.1. CAPABILITIES ADDED TO NCTUNS

Within the objective of improving the quality and to obtain more realistic simulations with NCTUns, the open source code of the simulator has been modified. Therefore, future users of the simulator could use these improvements to obtain more realistic results of their simulations with VANETs and HSVNs. The future users of NCTUns will find the following capabilities added to the simulator.

The modifications to the code of NCTUns were made taking into account the scenario that would be simulated, so there are some lines of the code that are exclusively for the smart city framework detailed earlier and the scenarios that will be discussed in the next chapters. As a future work it could be proposed a project that takes the idea shown on this project and implement a general solution, based on the work done. It would consist on getting from the GUI all the nodes that forms part of the VANET, differentiate vehicles from ITLs and regular traffic lights and finally combine the code of this project with that information, achieving the code adaptation to any scenario proposed.

As it was mentioned briefly before, the traffic statistic collecting relies on every vehicle of the scenario running AODV as the routing protocol. For the traffic density, the vehicles will count the amount of neighbors surrounding it and send this information to the nearest ITL (Intelligent Traffic Light); this process will be repeated every 2 seconds. For the traffic delay the vehicle will send the time when a turning decision is in progress to the nearest ITL and the second one will compute the delay statistics; in this case the data will be sent every time a vehicle makes a turn decision.

On the contrary, the traffic statistic processing relies on ITLs, assuming they will have enough computational power and memory to store this statistics. This part of the process consists on receiving from any vehicle a new type of AODV created for this purpose named “STAT_msg”, which contains the statistic information from the source node either traffic density or traffic delay. The ITL will store this data using an exponential weight mean average (EWMA) giving more weight to the historical values or the instantaneous value depending on the situation. This EWMA acts as a low pass filter achieving a slow variation of the statistics and tending them to the real situation around the ITLs.

8.2. NCTUNS MODIFIED MODULES

As described in Table 6.1, the statistic messages (STAT_msg) will include different type of information. This information regards the traffic density of each road segment along the path, the traffic delay of the vehicles when travelling streets and the weather conditions of the roads.

Thus, in this project this information has to be added when a packet is sent. Also, the receptor of the packet has to be capable of understanding the type of information that is being received and the module controlling the vehicle must be a part of the traffic delay gathering process.

In order to avoid affecting the simulator behavior, all the code of AODV was read, understood and analyzed in order to have good information of where the new lines of code should be added. How the different modules taking part on this process have been modified and how new users can use them is described in the sections below.

8.2.1. AODV.CC MODULE

The original "AODV.cc" module of NCTUns was in charge of the tasks for the routing protocol: creating routes, deleting routes when obsolete, maintaining a neighbors list, sending periodical HELLO messages, among others.

The modified module was added with a new type of AODV message for statistical purposes (AODV_STAT). In addition, this module was added with functions in charge of neighbors counting (executed by the vehicles) and statistics update using EWMA (executed by ITLs).

Summarizing, the changes made in the code on this module were the following:

- Function called every two seconds to count the amount of surrounding neighbors. Then, the vehicle sends the value to the nearest Intelligent Traffic Light. (CheckNeiList_Stat).
- Create a new type of AODV Message described previously in Table 6.1 (STAT_msg).
- Updating Statistic process for ITLs. This process consist in calculating the statistics using a low pass filter (EWMA) and store it on an individual file for every Intelligent Traffic Light and another global file to track the general statistics in the whole area.
- Comment a part of the code that prevents sending HELLO messages when there were less than 2 hops on the routing table. This action was made because it was needed that the vehicles continuously connects (send hello packets) between neighbors and ITLs. If there are less than two neighbors the vehicle won't send any hello packets, situation that could be possible on the scenario and we wanted to avoid.
- Create a new function that assembles a "STAT_msg" message and send it to the closest Intelligent Traffic Light.

8.2.2. AODV.H MODULE

The original file contains all the functions, variables and constant declarations used in the "AODV.cc" and "AODVrt.cc" modules. The changes made on this file were simple: adding the declaration of the new functions and constants used in both modules detailed before.

8.2.3. AODVRT.CC MODULE

The original "AODVrt.cc" module was implemented to modify and manipulate the tables and lists of the routing protocol. The modification added to this module was a function which checks the neighbor list counting neighbors which expiration time is greater than the consulting time and in the same time get the IP address of the nearest ITL inside the list. The results of this function were passed to the function that was in charge of sending the statistic message.

8.2.4. CARAGENT.CC MODULE

This module of NCTUns is in charge of the mobility of the vehicles during the simulation. Every action taken by the vehicles is made by the mobility agent. The change made in the code for the mobility agent consists that the vehicle alerts the nearest ITL when it is making a turn decision. The ITL will be in charge of the statistic calculation process. To establish who is the nearest ITL of the vehicle was used the same logic that was used in traffic density statistics.

8.3. FILES CREATED IN THIS PROJECT

In this project, the "AODV.cc", "AODV.h", "AODVrt.cc" and "CarAgent.cc" modules have been modified, and within these modifications different output files will be created when using these four modules. These files created and the information they provide are described in the following sections. In addition some AWK files used for result filtering are detailed below.

8.3.1. TRAFFIC-STAT-ID FILE

Each of the ITLs generates this type of file to store and update the statistics of traffic density of its own area. As seen in Figure 8.1, the ITL receives information every two seconds as was explained in previous chapters. In addition to this individual files for traffic density statistics, the simulator creates another global file where all the content of this individual files are stored.

The content of this type of file are organized by columns. The first one of them contains the simulation time in which the statistic message was received and in the second column the updated statistic for that specific time using a EWMA.

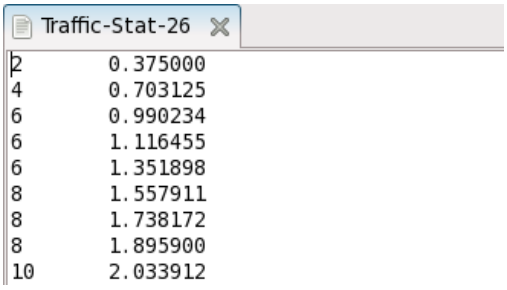

Figure 8.1 - Traffic-Stat-26 File Screenshot

8.3.2. GLOBAL-TRAFFIC-STAT FILE

This file is generated by the simulator to store in one single file the information about traffic density of all the ITLs of the simulation. The reason of creating this file is to filter the results getting only the latest statistic of every second of every ITL using an AWK filter. These filtered results were used to obtain graphics which are going to be explained in the following chapters.

The content of this file also is organized by columns. The First of them shows the Time when the statistic message was received, the second column indicates the ITL which received the message, the third column contains the statistic that ITL presents on that certain moment and the fourth and last column shows the number of neighbors received in the last statistic message. All the previous description can be observed in Figure 8.2.

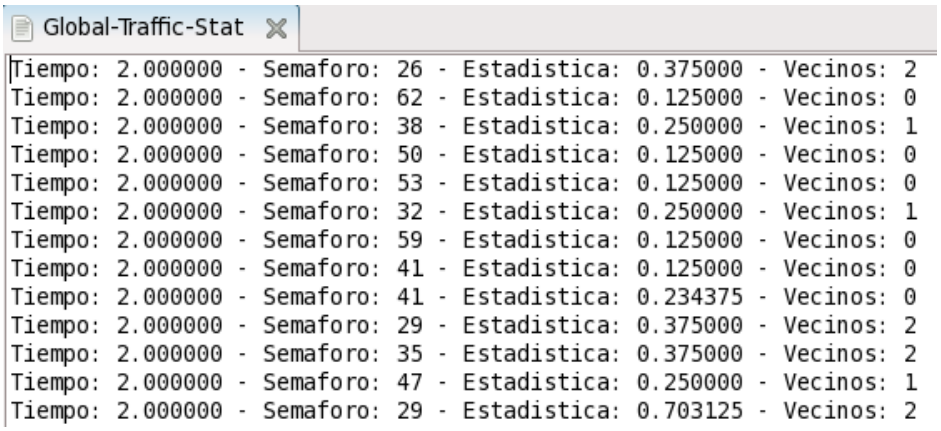

Figure 8.2 - Global-Traffic-Stat File Screenshot

8.3.3. DELAY-STAT-ID FILE

Each of the ITLs generates this type of file to store and update the statistics of traffic delay of its own area. When an ITL receives a statistic message with traffic delay data it must verify which was the last time for the vehicle sending the message. This verification is made on an individual file that any vehicle has, to keep track of the time when it makes a turn decision.

The content of this file is organized in columns as shown in Figure 8.3. The first column is the time when the ITL receives the statistic message, the second one contains the vehicle that sends the message and the third column shows the subtraction of the actual time with the last time that the vehicle has on its own individual file, in other words the delay of the vehicle.

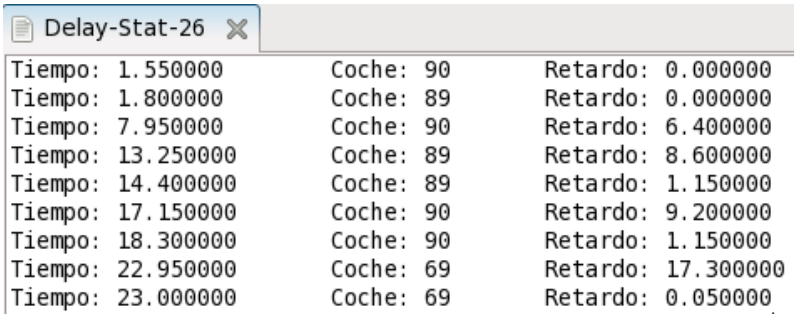

Figure 8.3 - Delay-Stat-26 File Screenshot

8.3.4. DELAY-CAR-ID FILE

Each of the vehicles generates this type of file to keep the record of every time it makes a turn decision. This file is important for the simulation because every time an ITL computes a subtraction to calculate the delay of a certain vehicle, the ITL first checks the individual file of that vehicle.

The content of this file is simple and consist on a unique column that shows the time when a turn decision is made by the mobility agent for the vehicle. (Figure 8.4)

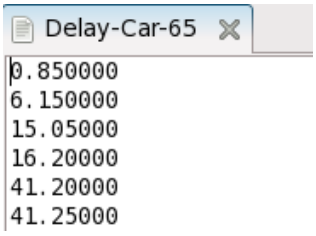

Figure 8.4 - Delay-Car-65 File Screenshot

8.3.5. GLOBAL-DELAY-STAT FILE

This file is generated by the simulator to store in one single file the information about traffic delay of all the ITLs of the simulation. The reason of creating this file is to filter the results by ITL using an AWK filter. These filtered results were used to obtain graphics which are going to be explained in the following chapters.

The content of this file also is organized by columns. The First of them shows the Time when the statistic message was received, the second column indicates the ITL which received the message, the third column contains the vehicle that sent the message and the fourth and last column the statistic that ITL presents on that certain moment. All the previous description can be observed in Figure 8.5.

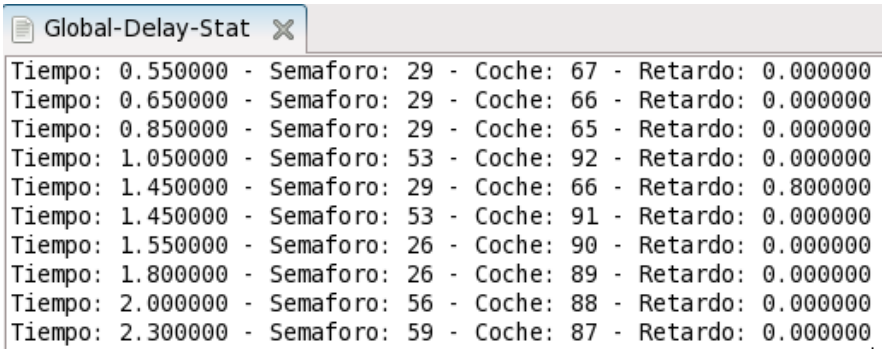

Figure 8.5 - Global-Delay-Stat File Screenshot

8.3.6. AWK FILTER "TRAFFIC-FILTER.AWK" FILE

This file was programmed with the objective of filtering the global traffic density file and obtaining results that could be put in graphics. The output files of this filter are ".txt" files that are perfectly capable for importing to Windows Operative System. This AWK file creates an individual file for each ITL selecting the last density statistic of each second of simulation. For example, if the ITL 26 received 3 different density messages on the second 14, it would update the statistic three times; The project is only interested on the latest update for statistical purposes so that is what this filter do.

8.3.7. AWK FILTER "DELAY-FILTER.AWK" FILE

As the previous AWK file, this was programmed with the objective of filtering the global traffic delay file and obtaining results that could be put in graphics. The main purpose was to differentiate the delay statistics by ITL, creating individual ".txt" files that could be imported to Windows Operative System.

9. SIMULATION ENVIRONMENT AND SCENARIO

The selected scenario for simulations and tests of the Smart City is a Manhattan style map with streets that form 4x4 blocks. The Smart City has obstacles in every block that represent buildings, and traffic lights every cross which are responsible to manage the traffic of the ITS (Intelligent Transportation System) cars. There are only a few ITLs (Intelligent Traffic Lights) among the regular traffic lights, and the idea is to cover several streets with one intelligent light, taking advantage of the omni-directional propagation pattern of the antenna. In this case, an ITL receives data from any vehicle which is on any of the four streets it covers. Following this design to cover every road it is not necessary to have ITLs on every cross, so resources can be saved. In chapter six it was fully explained which is the project concept for a Smart City, its characteristics and how it works.

Intelligent Traffic Lights were represented using *Multi-Interface Mobile Node* nodes, with two Ad-Hoc interfaces configured in two different subnets. A sub-network for the interface (eth2) will communicate with ITS cars and the other interface (eth1) will communicate with the other ITLs.

The vehicles on the simulation were represented using *ITS Cars* equipped with an 802.11 b interface on Ad Hoc mode. This car has the characteristic that is controlled by an agent that moves it through the city respecting streets, crosses and traffic lights. It was configured with AODV routing protocol which has been modified to collect statistics. As this car is only supposed to connect with interface (eth2) of the ITLs and other surrounding cars, it was configured to cover 130 meters in an Omni-directional pattern.

9.1. SIMULATION DESCRIPTION

The simulation consists of a given number (N) of vehicles (*ITS Cars*) moving around the city and establishing communications with the nearest ITL to send the data of the amount of surrounding neighbors, data collected every 2 seconds, and the time it takes to travel the streets (traffic delay). Every time an ITL receives data from a vehicle it calculates and updates the statistic of car density and delay on its area, stores it on an individual file and prints it on a global trace. All resulting files from ITLs will be stored on a folder on the Desktop named "Results".

The results expected with this simulation are real-time calculation, from every ITL, of traffic density and delay on its own area. It is essential that the statistics of every ITL adjust well to the statistics of the simulated scenario. This way we will be sure of the system's reliability applied on a smart city of the real world.

9.2. SIMULATED SCENARIO

The simulated scenario consists in a Smart City with a certain amount of vehicles moving around the city. In a particular moment (t_1) the numbers of cars inside the city will be increased, because a group of vehicles enter from the city entrance, located on the south where is deployed the ITL_{59} as shown in Figure 9.1. Another group of cars will also enter to the city in a second moment (t_2), to increase again the number of vehicles. This means the total number of vehicles moving around the city will be increased two times. This augmentation will represent the peak hours of any city which represent more density of cars and more delays. As in any real city there must be a return to a normal situation so, it is also simulated when the vehicles arrive to their respective destinations, reducing the number of vehicles. The reduction is also made in two different groups on different times, t_3 and t_4 respectively.

On Table 9.1 it is shown how the group of vehicles in the city increase and decrease throughout time.

Simulation Time Range	Number of Vehicles
0 – 40 sec	30
41 – 100 sec	42
101 – 250 sec	56
251 – 300 sec	44
301 – 400 sec	30

Table 9.1 - Number of Vehicles vs Simulation Time

In Figure 9.1 it is shown where the vehicles get to the city, and a possible situation of simulation. The ITS cars (vehicles) are represented with Ferraris and ITLs are represented with a traffic light. In this scenario three ITLs will be studied rigorously to understand the behavior and adaptation of the code we have implemented. These ITLs would be ITL_{59} (deployed on the city entrance), ITL_{44} (deployed on the city downtown) and ITL_{29} (deployed on the city outskirts).

These ITLs will represent all the possible situations that could happen on the scenario, so the code will be tested with different circumstances inside the same city. It is expected that the code adapts quickly and efficiently to all possible situations of the proposed scenario.

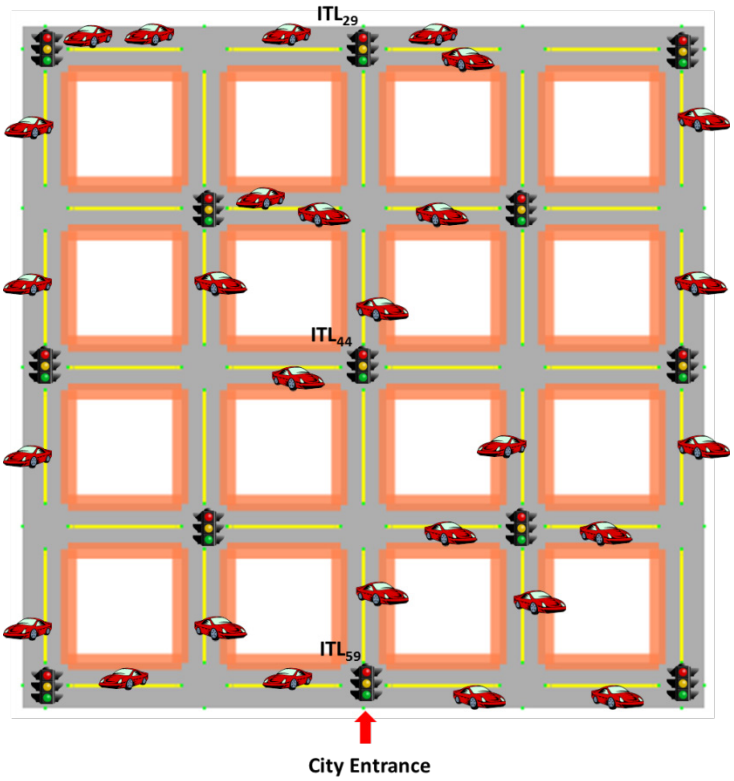

Figure 9.1 - Simulated Scenario Representation

In Figure 9.2 it is shown the actual scenario simulated on NCTUns to obtain the results that are going to be presented in the next chapter. A highway is connected to the city entrance at the south, with two different groups of vehicles waiting to enter the city. As it was said before, the first group activates at second 40 of simulation (t_1) and starts moving towards the city. The same thing is done by the second group but from the second 100 (t_2). To simulate the vehicles leaving the city or arriving to the destination, the same vehicles of the two previous groups were configured as broken, so they stop being part of the simulation. The first group gets broken at second 250 of simulation (t_3), meanwhile the second group gets broken at second 300 (t_4).

In the NCTUns simulator, once a car gets broken it returns to its original position and remains there until the end of the simulation. That is, the cars leave the city (in the afternoon) and return to the highway from which the come in the morning.

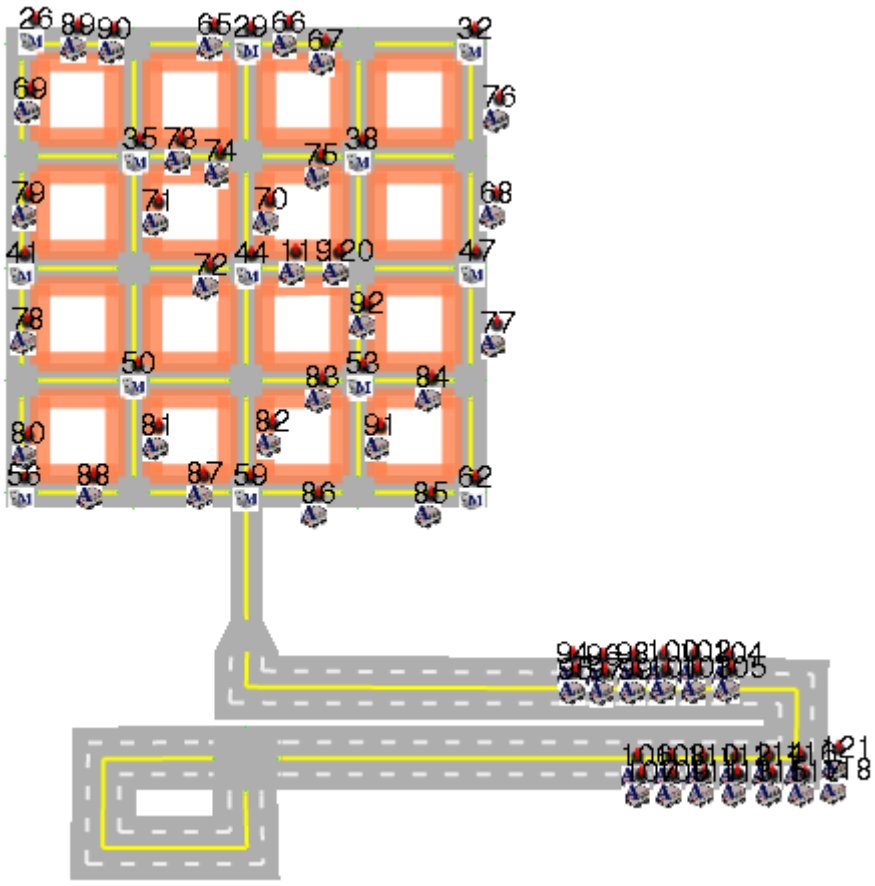

Figure 9.2 - Real Scenario Simulated

Finally, in Table 9.2 there are all the simulation parameters that were used to obtain the results shown in chapter 10. All modified files were explained in chapter 8.

Simulation Settings	
Medium Capacity	11 Mbps
Packet Size	256 Kbytes
Transmission Range	ITL Interface eth1: 300 m
	ITL Interface eth2: 130 m
	Vehicles: 130 m
Carrier Sense	300 m: -68.5865 dBm
	130 m: -61.3229 dBm
Simulation Time	400 sec
MAC Specification	802.11b
Area	500x500
Average Speed	10 m/sec
Number of Nodes	ITLs: 13
	Vehicles: 56
Mobility Model	CarAgent* (NCTUns)
Routing Protocol	AODV*

*: Modified in this Final Degree Project

Table 9.2 - Simulation Settings

10. SIMULATION RESULTS

The scenarios described in chapter 9 have been simulated with NCTUns [90], and the results of these simulations are presented in this section. The objective of these simulations is to test and evaluate how the use of Intelligent Transportation Systems (ITS) may help to improve road safety.

Furthermore, these simulations have also to test and evaluate that the changes made in the simulator during the work of this Final Degree Project. Also, they will show that it is possible to get accurate real-time statistics for traffic density and delay.

All the graphs that will be presented below which indicates a Confidence Interval (CI) are obtained from 10 simulation repetitions, and represented using a 90% of confidence interval.

10.1. ITL_{59} RESULTS (CITY ENTRANCE)

As explained before, one of the three ITLs that are going to be studied is ITL_{59}. This ITL's location is in the city entrance, so there will be a bottleneck of vehicles waiting to enter the city. This will result in an augmentation of car density and delay on that specific area.

First, it is going to be analysed the results of traffic density and after that the results of traffic delay.

10.1.1. TRAFFIC DENSITY FOR ITL_{59}

Figure 10.1 shows how the traffic density statistic of ITL_{59} evolves during the simulation time.

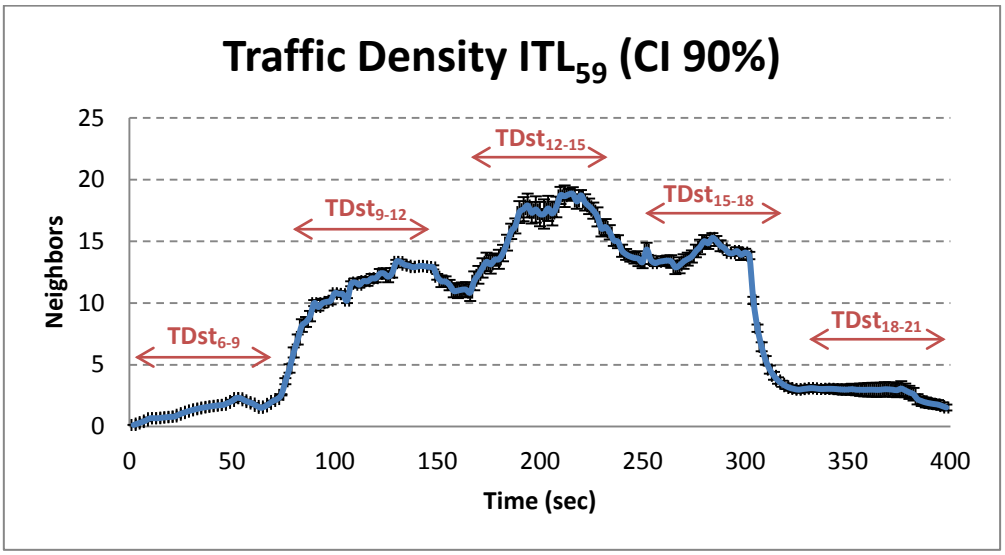

Figure 10.1 - ITL_{59} Traffic Density

If we have in mind Figure 9.2, it can be noticed that in a first moment ITL_{59} only has three surrounding vehicles so the statistics for the first seconds of simulation should be very low. In addition, and remembering Table 9.1, it is known that the first group of vehicles that enters the city start moving from second 40, but they are distanced from the city entrance so they have to travel the highway spending time on it. The actual entrance of the vehicles to the city is evidenced on the graph (Figure 10.1) around second 75, being that the expected.

The graph presents the same behaviour as was described in Table 9.1: There are 5 delimited areas on the graph that can be associated with the entering groups of vehicles to the city. The first area is the initial condition, only 30 vehicles are inside the city but well distributed, being only three vehicles around ITL_{59} at first. In the second area (from second 75 to 170) it is shown an augmentation of vehicle density, as the first group of entering cars arrived the city entrance and wait for its turn to get in. The third area (from second 170 to 250) shows when the second group of vehicles intend to enter the city, but there are some vehicles of the first group still trying to get to the city, so the groups now form a bigger group and the density increases again, reaching a maximum. The fourth area (from second 250 to 300) shows how the density decreases due to the configuration of the vehicles of the first group as broken, although the vehicles from the second group are still in the city. The fifth and last area shows when the vehicles of the second group are configured as “broken” (i.e. the leave the city) so the city is left only with the 30 initial vehicles well distributed around the city, making the density falls.

The entire simulation has a duration of 400 seconds, which aims to simulate a day in a city like Barcelona from 6:00 to 21:00, where it is more important to study the statistics proposed in this project. These time intervals of the city are represented on Figure 10.1.

10.1.2. TRAFFIC DELAY FOR ITL_{59}

Once the traffic density results have been analysed, the traffic delay results are going to be explained. These results regarding delay should be related with the previous results regarding density, as they were obtained from the same simulation.

By delay, we mean the average delay it takes a vehicle to travel in a street, between consecutives crossroads.

It is important to remark that the minimum delay time expected for this results are 10 seconds, because the vehicles are set with a maximum velocity of 10 m/sec and the street length is 100 meters. Assuming the worst case scenario for delay, the vehicle will travel with its maximum velocity during the whole process, spending 10 seconds in travelling the car. From this premise, any delay of less than 10 seconds should be considered negligible data.

In the following figures (Figure 10.2 and Figure 10.3) it is shown how the ITL_{59} traffic delay statistics adapt to changes on the scenario. In Figure 10.2 the low pass filter (EWMA) of the ITL is configured with an alpha value of 0.125, meanwhile in Figure 10.3 the alpha value was set on 0.250. In both figures are shown in green the instantaneous delay received by the ITL and in red the historical statistic of delay.

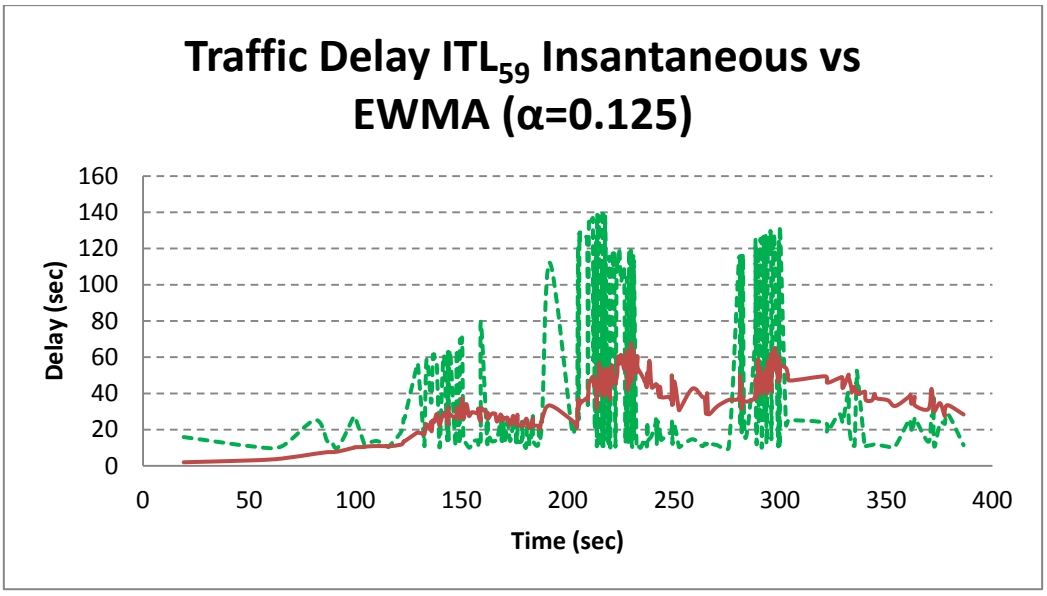

Figure 10.2 - ITL_{59} Traffic Delay Instantaneous vs EWMA (α=0.125)

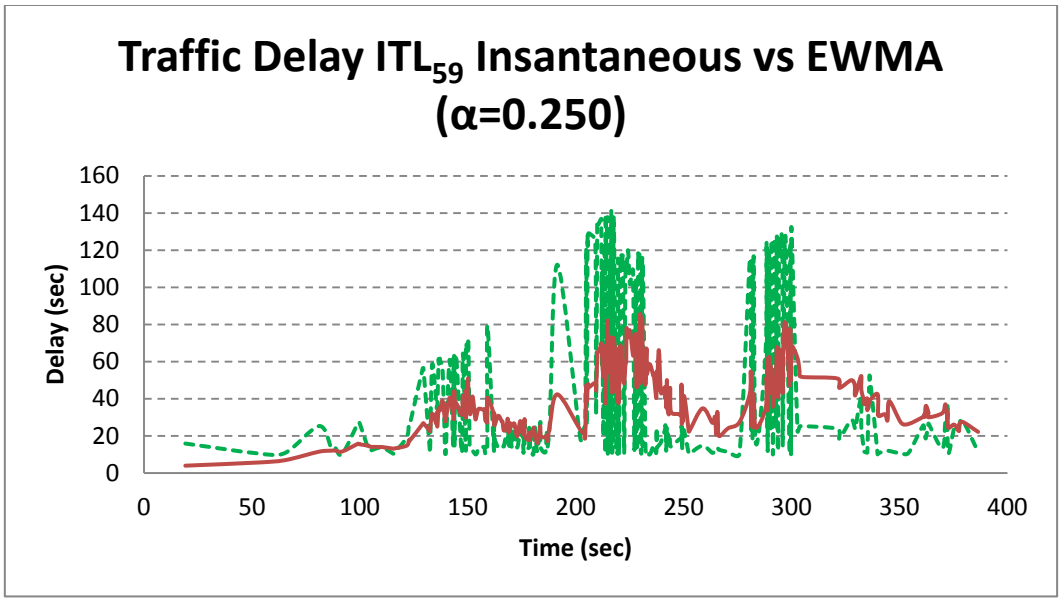

Figure 10.3 - ITL_{59} Traffic Delay Instantaneous vs EWMA (α=0.25)

In both figures it can be seen that the instantaneous delay matches the behaviour of the traffic density graph: it also presents five delimited areas. The second area (from second 125 to 160) and the third area (from second 180 to 230) corresponds to the entering of the first and the second group of vehicles, respectively, into the city, justifying the augmentation of the delay on the area. As was explained for traffic density results, when the first group of vehicles reach the city entrance they have to wait to get in to the city, which is why the delay increases. When the second group of vehicles arrive to the city entrance, not all the vehicles from the previous group have entered yet to the city, so they make together a bigger group and the delay increases even more, reaching a maximum value.

Then, in the fourth area (from second 275 to 310) the delay decreases but it still is considerable big. This situation is due to the configuration of the vehicles of the first group as "broken" (i.e. they leave the city), although the vehicles from the second group are still in the city. Finally, the vehicles of the second group are set with a broken configuration and the delay drops to the initial state.

Finally, comparing Figure 10.2 and Figure 10.3 it can be observed that when the ITL_{59} computes the statistical values using an alpha value of 0.25 the statistic reacts and adapts faster to changes on the scenario than using 0.125. Mathematically, it can be explained since alpha determines the weight that the instantaneous and historical values will have. When alpha is set with a low value the historical value is benefited, however when is set with a value near to 1 the instantaneous value is benefited. As 0.25 is bigger than 0.125 it is logical to assume the statistics will react faster to the instantaneous variations as indeed it happens.

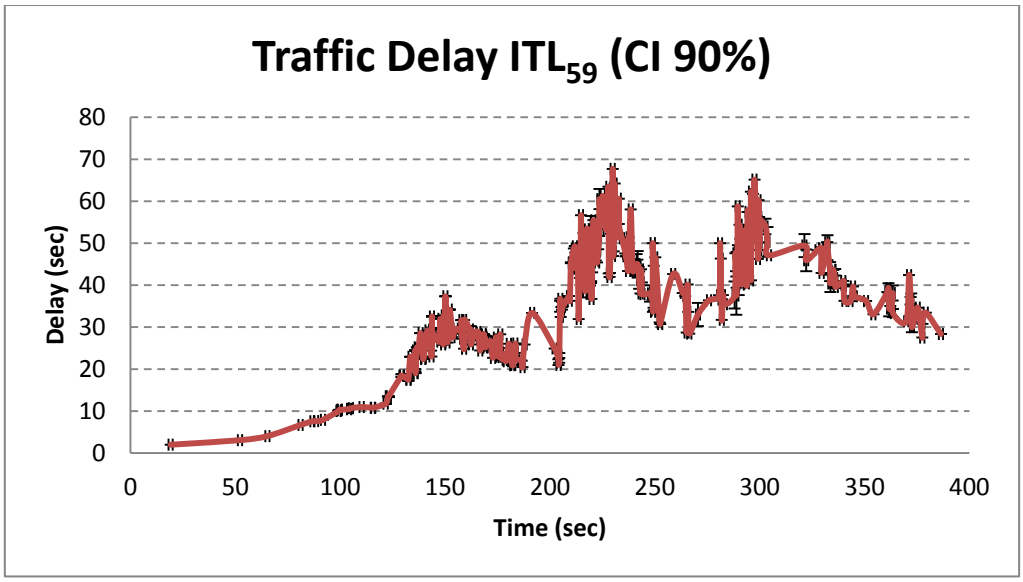

Figure 10.4 - ITL_{59} Traffic Delay CI 90%

Figure 10.4 shows the traffic delay statistics of ITL_{59} using an alpha value of 0.125 with a confidence interval of 90%. In this graph it can be appreciated the maximum values of delay that the statistics reach. The maximum of delay reached by the statistics is around 70 seconds and corresponds to the third area (200 – 250 seconds), when the two groups of entering vehicles coincide at the city entrance.

10.2. ITL_{44} RESULTS (CITY DOWNTOWN)

The ITL_{44} is located on the city's downtown (see Figure 9.1). This could represent a constant traffic density due to a higher probability for the vehicles to pass by this area at any time of the day. In other words, a vehicle could reach the city's downtown from any part of the city (north, south, east or west), so it is more likely that a vehicle travels through the downtown than through the outskirts of the city.

10.2.1. TRAFFIC DENSITY FOR ITL_{44}

In Figure 10.5 is shown how the traffic density statistic of ITL_{44} evolves during the simulation time. If we remember Figure 9.2, it can be noticed that in a first moment ITL_{44} only has four surrounding vehicles, so the statistics for the first seconds of simulation should tend to this value until such vehicles leave the coverage area. In the early stages of simulation it could be seen how the traffic density statistics tend to a value of two neighbors, but the vehicles already left the coverage area, so the statistics start decreasing again.

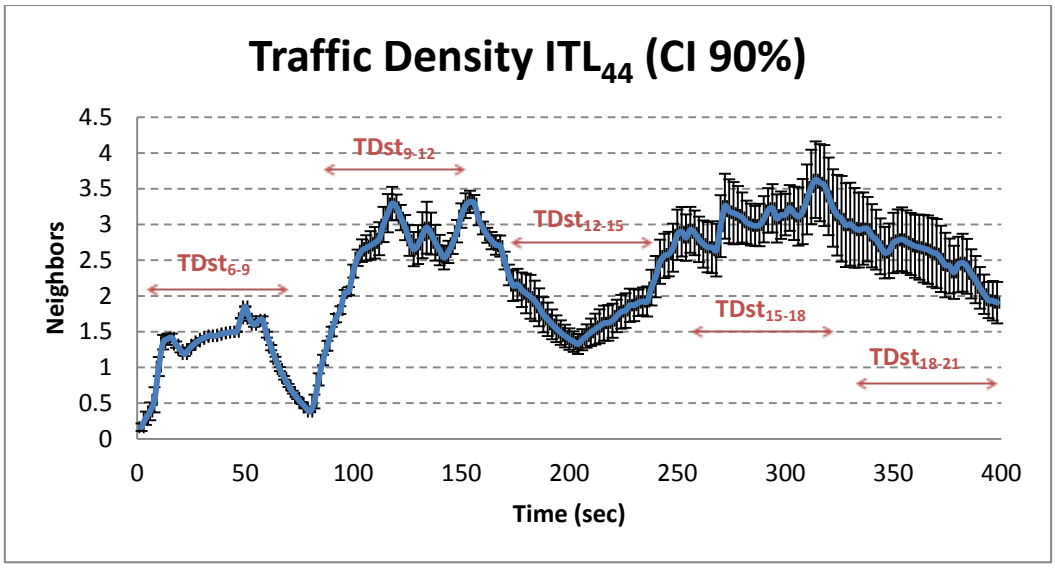

Figure 10.5 - ITL_{44} Traffic Density

Unlike ITL_{59}'s traffic density results, Figure 10.5 does not present five delimited areas for the analysis; instead it presents what it could be considered as three differentiated areas. The first of them (from second 0 to 80), represents the period of time when the downtown is not affected by the entrance of the first group of vehicles. If we check the results of ITL_{59}, from the second 75 (approx.) the new vehicles start entering the city, so it is reasonable to expect these vehicles will travel the city's downtown a couple seconds later. Indeed it is what is obtained: the traffic density of ITL_{44} increases considerably from the second 80 of simulation.

Around second 200 is evidenced the contribution of the second group of vehicles, maintaining a stable value of traffic density statistic. To remark the difference between a downtown ITL and an outskirt ITL the average value of density of each case are going to be compared, for that the Table 10.1 is presented.

Average Traffic Density Value ITL 44	
Complete Simulation	2.24696
From Second 80	2.50639

Table 10.1 - ITL_{44}'s Average Traffic Density Value

As shown in the previous table, the average value of traffic density that presents the ITL_{44} is bigger when it is only considered the time of simulation when the two groups of vehicles entered the city than the complete simulation time.

Also, it could be seen in Figure 10.5 that from the start of the second area there is a pseudo stable behaviour. Although the two groups of vehicles left the city from the second 300 the traffic density of the ITL_{44} remains stable. As previously mentioned, the initial vehicles that were in the city would tend to move to the downtown instead to the outskirts (according to our simulation settings), so at the end of the simulation it is more likely to be a greater number of vehicles in the downtown than in the outskirts.

10.2.2. TRAFFIC DELAY FOR ITL_{44}

If a constant traffic density was expected for this Intelligent Traffic Light (ITL_{44}), it would be logical to expect a constant traffic delay statistic. The most important thing is that the traffic delay results obtained from simulation are consistent with the traffic density ones.

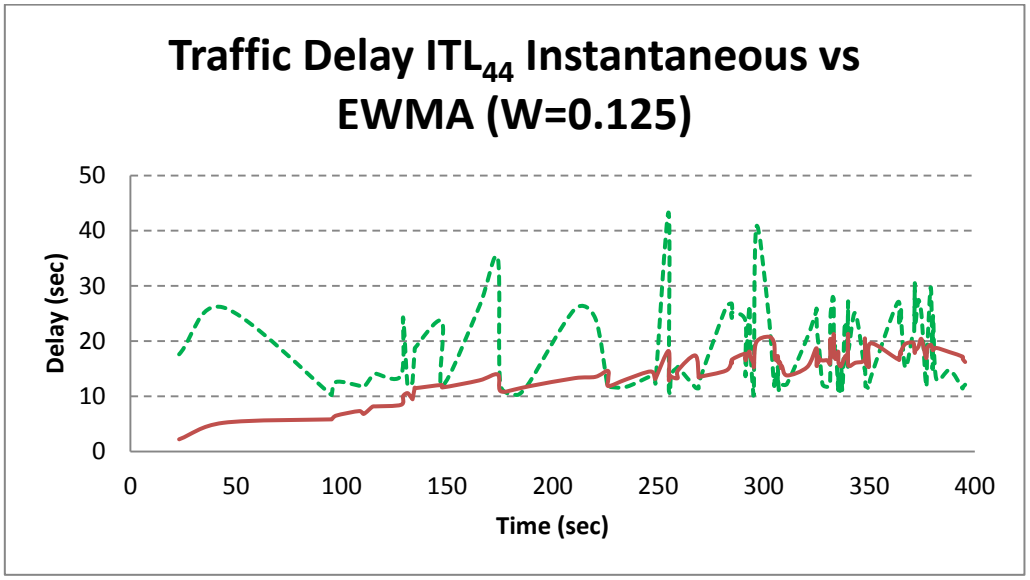

Figure 10.6 - ITL_{44} Traffic Delay Instantaneous vs EWMA (α=0.125)

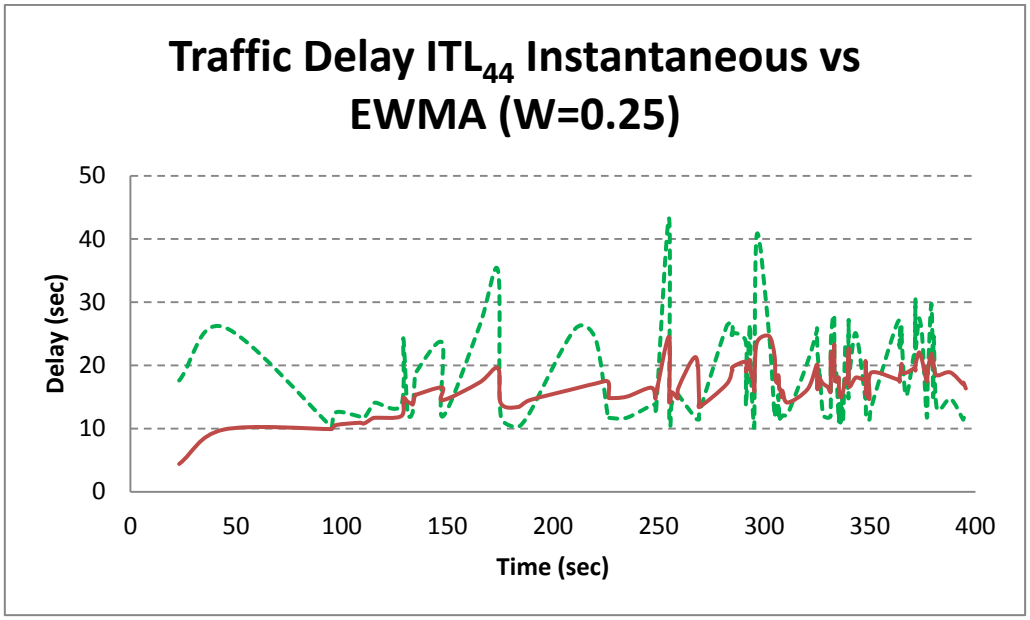

Figure 10.7 - ITL_{44} Traffic Delay Instantaneous vs EWMA (α=0.25)

Once again, Figure 10.6 and Figure 10.7 compare how the ITL_{44} adapts to the instantaneous values of delay using different alphas on the low pass filter EWMA. In Figure 10.6 alpha is set on 0.125, meanwhile in Figure 10.7 alpha is 0.25.

In both graphics it could be seen how the delay statistics change according to the variation of traffic density along the simulation. With each increase in the traffic density statistics it can be

seen a similar change in traffic delay statistics, confirming that the delay and density are closely related.

Figure 10.8 shows the traffic delay statistics of ITL_{44} using an alpha value of 0.125 with a confidence interval of 90%. In this graph could be appreciated the maximum values of delay that the statistics reach. The maximum of delay reached by the statistics is 22 seconds and corresponds to the third area mentioned above.

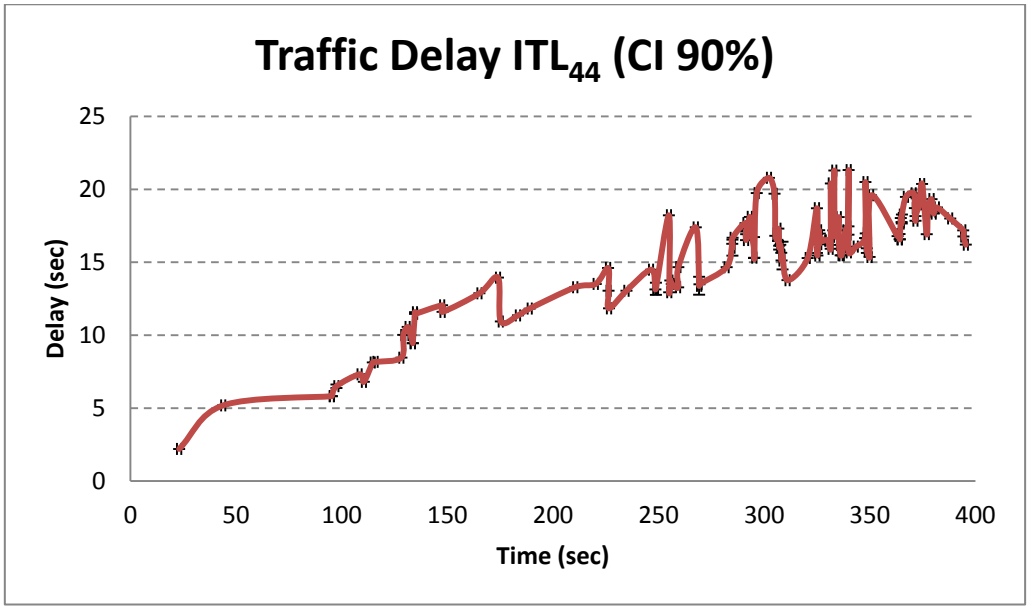

Figure 10.8 - ITL_{44} Traffic Delay CI 90%

10.3. ITL_{29} RESULTS (CITY OUTSKIRTS)

The last of the three ITLs that are going to be studied is the ITL_{29} that is located in the city's outskirt, at north (see Figure 9.1). As was explained before, in a city it is more likely for the vehicles to travel in the city's downtown than the city outskirts, since an outskirt refers to remote locations of the city and hence less population, pedestrians and vehicles itself.

The expected results for this ITL are a low statistic register for traffic density and delay, even though it is possible sporadic situations characterized by an increase of density and delay.

10.3.1. TRAFFIC DENSITY FOR ITL_{29}

In Figure 10.9 it is shown how the traffic density statistic of ITL_{29} evolves during the simulation time. Referring to Figure 9.2, it can be noticed that in a first moment ITL_{29} only has three surrounding vehicles, so the statistics for the first seconds of simulation should tend to this value until such vehicles leave the coverage area. In the early stages of simulation it could be seen how the traffic density statistics tend to a value of two neighbours, but the vehicles

already left the coverage area, so the statistics start decreasing again. From that time in advance, the statistics will depend directly to the vehicles that travel through ITL_{29}'s coverage area.

This figure has a direct relation with the outcomes for ITL_{44} due to a geographical proximity. For this, the comparison for results will be done taking in count ITL_{44} despite ITL_{59}, where the bottleneck really happens.

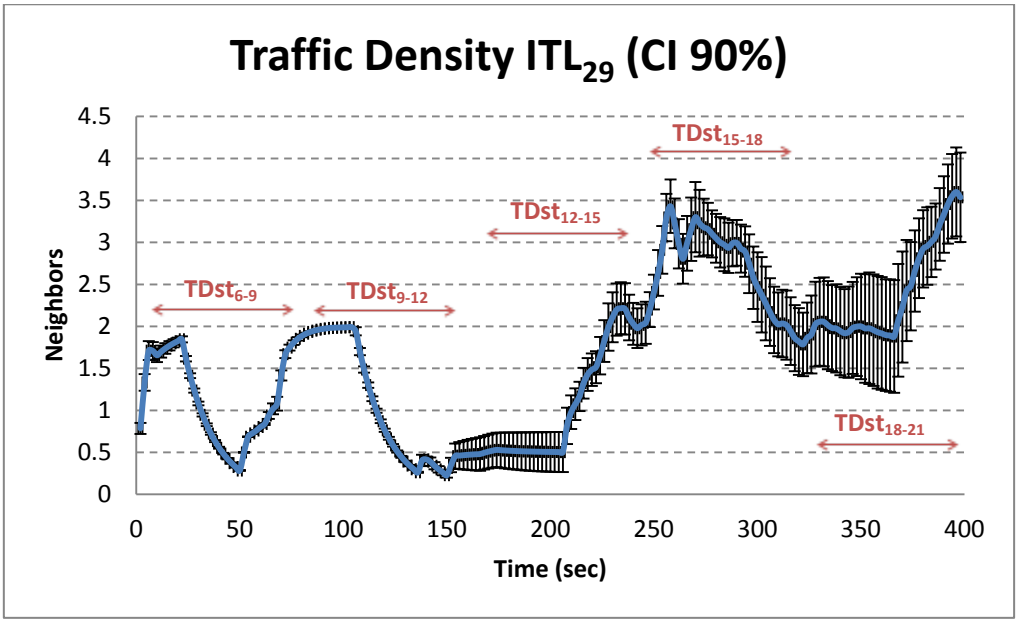

Figure 10.9 - ITL_{29} Traffic Density

As shown in Figure 10.9, the main increase on traffic density starts at second 200 of simulation. Consulting Table 9.1 this augmentation coincides with the period when the two groups of vehicles are already inside the city or at least intending it. A relevant aspect is that the density increase occurs almost 120 seconds after the first group of vehicles are recorded in the statistics of ITL_{44}; this fact can be justified on the grounds that vehicles spent time moving from downtown to ITL_{29}'s area.

Despite that in this ITL was not expected a considerable statistic record as it was deployed at the city outskirt; it is observed that the behaviour is similar to the ITL_{44} in terms of results, but with a time lag. This leads to think that while it was thought of ITL_{29} as an outskirt Intelligent Traffic Light it really works as a near-downtown ITL. The remoteness of the Intelligent Traffic Light is not enough so that its behaviour is as was expected initially.

Average Traffic Density Value ITL_{29}	
Complete Simulation	1.641119
From Second 200	2.285358

Table 10.2 - ITL_{29}'s Average Traffic Density Value

Comparing Table 10.1 and 10.2 it could be appreciated when it is calculated the average traffic density value for the time in that vehicles are part of ITLs statistic of ITL_{44} and ITL_{29} that are

practically equal. This corroborates the idea that both Intelligent Traffic Lights have the same behavior.

Although the ITL_{29} does not behave as expected and it could not be analysed the three different behaviours proposed for ITLs at the beginning of the chapter, it has been possible to check the general functioning of the code regarding traffic density statistics.

10.3.2. TRAFFIC DELAY FOR ITL_{29}

Once proven that the traffic density behaviour of ITL_{29} is indeed similar to ITL_{44}, it is expected that the traffic delay results of both Intelligent Traffic Lights are similar.

As the previous analysis, in Figure 10.10 and Figure 10.11 is shown the adaptation of the ITL_{29} regarding traffic delay statistics, using an Exponential Weight Mean Average (EWMA) with different values for alpha (0.125 and 0.25 respectively).

In both graphs it could be appreciated that the delay increases at the same simulation time than the density does, maintaining consistency in the results.

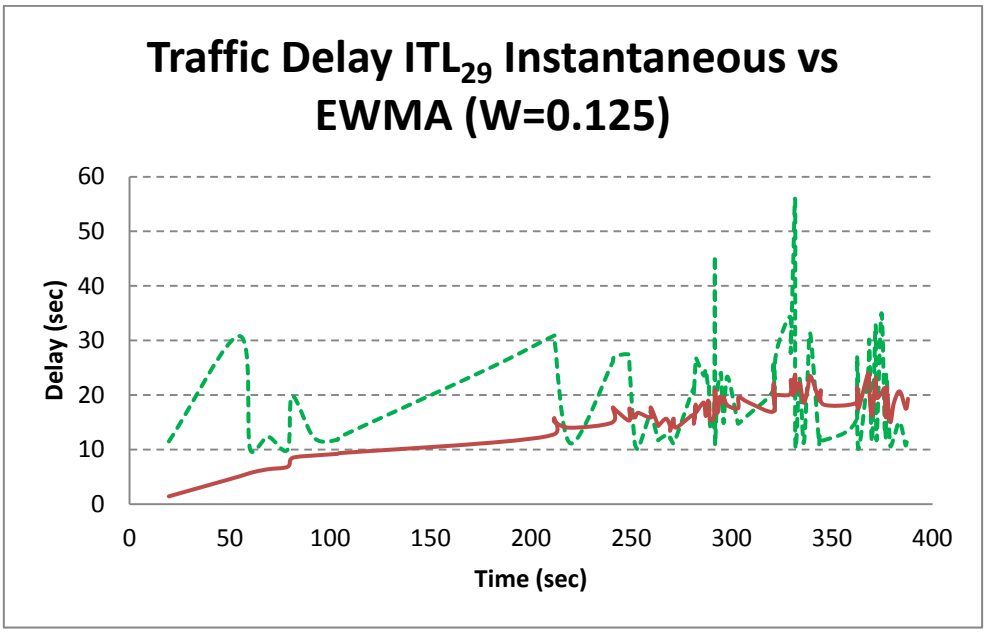

Figure 10.10 - ITL_{29} Traffic Delay Instantaneous vs EWMA (α=0.125)

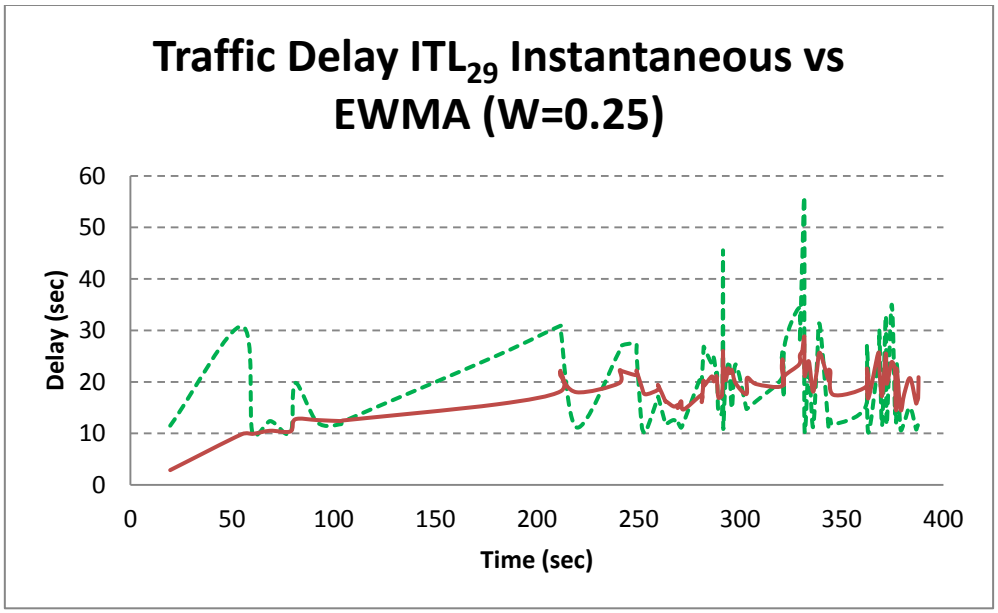

Figure 10.11 - ITL_{29} Traffic Delay Instantaneous vs EWMA (α=0.25)

Figure 10.12 shows the traffic delay statistics of ITL_{29} using an alpha value of 0.125 with a confidence interval of 90%. In this graph could be appreciated the maximum values of delay that the statistics reach. The maximum of delay reached by the statistics is 24 seconds, a similar value obtained in ITL_{44}.

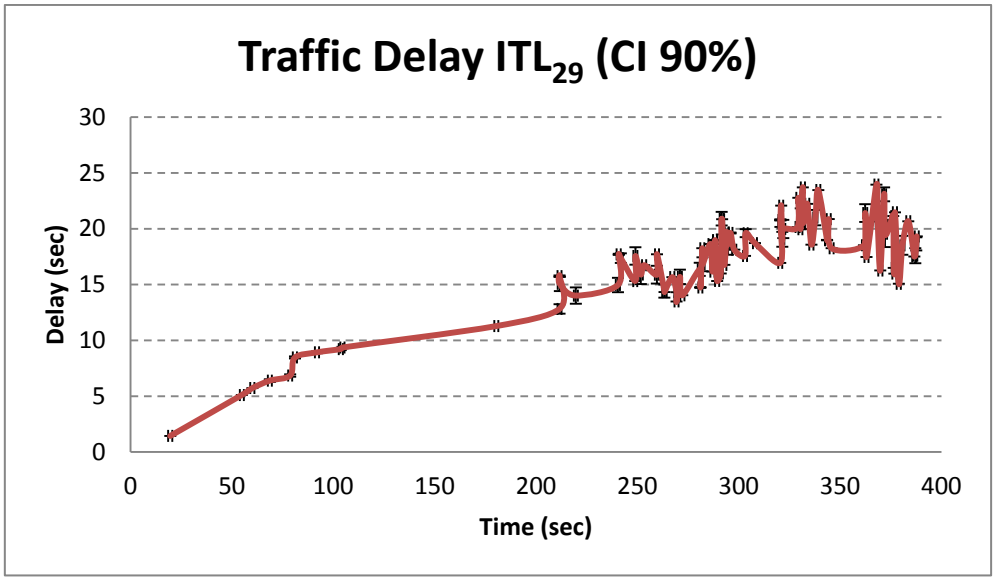

Figure 10.12 - ITL_{29} Traffic Delay CI 90%

11. CONCLUSIONS AND FUTURE WORK

The introduction of Intelligent Transportation Systems (ITS) applied to communications between transport infrastructure and vehicles seeks to improve transport safety in the roads. Public organizations such as the European Union as well as private car manufacturers are focusing their efforts on the development of this kind of communication technologies in order to deploy them during the next years. Thus, cities as Barcelona are being used as living labs [14] to test these technologies in real life situations. Nonetheless, testing these technologies in real situations and with real technologies is expensive and difficult (hundreds of cars equipped with these technologies are needed) so the use of simulators in order to develop these technologies before its actual deployment becomes mandatory.

In this project, a smart city framework has been presented in order to explain how future cities will communicate with vehicles travelling around it. In this smart city framework a set of Intelligent Traffic Lights (ITLs) deployed around the city will gather statistic information, regarding traffic density and delay, from passing vehicles and will share this data with an ITL's sub-network. At the same time, this sub-network will periodically send warning messages to passing vehicles informing about traffic density and weather conditions [47], in order to improve road safety, save fuel and reduce trip time. This smart city scenario has been evaluated in this project with the use of a network simulator, NCTUns [90], specifically modified to simulate smart cities.

The first conclusion that can be drawn from this project is that the inclusion of new capabilities to the simulator (such as the possibility of gathering real-time traffic statistics regarding density and delay) improves the quality of simulations. This quality is demonstrated by the adaptation of the statistics on real time to the changes on the scenario, translated in the

mobility of the vehicles managed by the CarAgent module. The statistics obtained in this project's simulations could be used for packet routing protocols as well as for traffic routing decisions that would help the driver to avoid traffic jams or accidents, making travel times shorter, achieving less CO_2 emissions and obtaining a sustainable smart city.

These changes made in the code of the simulator have been tested in this project with a general objective: to check if the changes made in the simulator worked, and it can be concluded that they definitively work. As seen, intelligent infrastructure can communicate with each other to collect, compute and share real-time statistics.

Despite that the new capabilities added to the network simulator functioned as expected, not all the ideas proposed in the smart city framework did. Specifically ITL_{29}, which had been initially proposed as an outskirt city Intelligent Traffic Light, actually behaved as an ITL adjacent to the city downtown due to its proximity. To obtain the behavior of an outskirt ITL, it should be implemented another type of scenario with an area distanced from the downtown simulating a suburb or an outskirt. As for the other ideas presented on the smart city framework, all showed similar behavior to the one that was expected.

Finally, it can be concluded that the use of ITS makes roads safer, more environmentally efficient and reduce the trip time.

Future lines in Hybrid Sensor and Vehicular Networks (HSVN) and in the development of simulation tools keep open based on this project.

In the side of NCTUns simulator, this document includes the necessary information to modify the NCTUns simulator. It is described how different modules can communicate to each other through the use of files and how the main modules work. Hence, these modules can be easily modified and different forms of communication between cars and infrastructure can be created.

In the side of HSVN, they can be used not only to increase road safety but also to improve drivers' quality of life. Thus, an application of HSVN that could be studied and implemented in NCTUns is the use of these networks to find a free parking spot. In this case, ITLs communicate to passing vehicles where to find free parking spots on their cover range. With this information the driver assistant would indicate the driver where to park. Furthermore, if the parking spot is a paying parking spot, the driver assistant could be use also to pay. The implementation of this HSVN could be as follows. A Wireless Sensor Network gets data about the location of free parking spots and it communicates this information to the ITLs. Then, the intelligent traffic lights could use a "parking message" to communicate to passing vehicles the location of the free parking spots. Finally, the driver assistant uses this data to draw in the map the location of the free spots.

Also, statistics collected by the ITLs can improve data routing protocols selecting the path that offers a higher chance to forward a packet to the destination successfully. In the Polytechnic University of Catalonia (UPC) is in progress a project whose objective is the design and

simulation of a VANET routing protocol that considers the framework to update traffic statistics that has been developed in this Final Degree Project.

As seen, future lines keep open after this project. Nevertheless, future projects involving car-to-infrastructure communication could use this project as base and make further implementations or add different type of statistics to the gathering and computing process.

ANNEX 1. PROPOSAL ARTICLE FOR 2012 IEEE INTELLIGENT VEHICLES SYMPOSIUM (IV'12)

The work done for this project is part of the work of a research group of ENTEL [88] department from the Universitat Politècnica de Catalunya (UPC). Information extracted from this project has been used as part of a paper that will presented for the 2012 IEEE Intelligent Vehicles Symposium (IV'12- 3-7 June 2012, Alcalá de Henares, Spain) [89].

Smart city for VANETs using warning messages, traffic statistics and intelligent traffic lights

Carolina Tripp Barba, Miguel Ángel Mateos, Pablo Regañas Soto, Ahmad Mohamad Mezher, Mónica Aguilar Igartua
Departament d'Enginyeria Telemàtica.
Universitat Politècnica de Catalunya (UPC). Barcelona, Spain
{ctripp, monica.aguilar, ahmad.mezher}@entel.upc.edu, {miguel.4.mateos, pauregas}@gmail.com

***Abstract*—Road safety has become a main issue for governments and car manufacturers in the last twenty years. The development of new vehicular technologies has favoured companies, researchers and institutions to focus their efforts on improving road safety. During the last decades, the evolution of wireless technologies has allowed researchers to design communication systems where vehicles participate in the communication networks. Thus, new types of networks, such as Vehicular Ad Hoc Networks (VANETs), have been created to facilitate communication between vehicles themselves and between vehicles and infrastructure. New concepts where vehicular networks play an important role have appeared the last years, such as smart cities and living labs [1]. Smart cities include intelligent traffic management in which data from the TIC (Traffic Information Centre) infrastructures could be reachable at any point. To test the possibilities of these future cities, living labs (cities in which new designed systems can be tested in real conditions) have been created all over Europe. The goal of our framework is to transmit information about the traffic conditions to help the driver (or the vehicle itself) take adequate decisions. In this work, the development of a warning system composed of Intelligent Traffic Lights (ITLs) that provides information to drivers about traffic density and weather conditions in the streets of a city is proposed and evaluated through simulations.**

***Index Terms*—Vehicular Ad Hoc Networks (VANETs), Traffic Information Centre (TIC), Smart Cities, Intelligent Transportation System (ITS).**

I. Introduction

During the last few years, progress in wireless communications has offered new research fields, providing network connectivity in environments where wired solutions are impossible. Among these, vehicular ad hoc networks (VANETs) are attracting a growing attention due to the promising important applications, from road safety to traffic control and entertainment for passengers. Smart cities would like to plan how to minimize their transportation problems due to the increasing population that produces congested roads. VANETs aim at helping to alleviate this issue improving vehicles' mobility, increasing road safety and also seeking to have more sustainable cities.

At the beginning of the development of vehicular technologies, the main goal was to have more efficient and safer roads. Nowadays, thanks to the huge development of wireless technologies and their application in vehicles, it is possible to use Intelligent Transportation System (ITS) that will change our way to drive, improve road safety, and help emergency services. VANETs may soon allow vehicles to easily communicate among themselves and also with fixed infrastructure. This will not only improve road safety, but also raise new commercial opportunities such as infotainment for passengers.

Car accident prevention, safer roads, pollution and congestion reduction are some of the goals of VANETs. The deployment of an efficient system to manage warning messages in VANETs has important benefits, from the perspective of both road operators and drivers. Efficient traffic alerts and updated information about traffic incidents will reduce traffic jams, increase road safety and improve the driving in the city. Furthermore, from the sustainable and economic perspective, real-time traffic alerting will reduce trip time and fuel consumption and therefore decrease the amount of CO_2 emissions [2].

In this work, a smart city framework has been developed in where intelligent traffic lights (ITLs) set in the crossroads of a city are involved. These ITLs are in charge of gathering traffic information (e.g. traffic density) from passing vehicles, updating traffic statistics of the city and reporting those statistics to the vehicles. Also, ITLs will send warning messages to vehicles in case of accidents to avoid further collisions.

The rest of the paper is organized as follows. Section II gives a state of the art of some relevant works of VANETs using ITS. Section III introduces our smart city framework. Section IV presents our proposal to calculate the vehicles' density in the city using ITLs. Section V summarizes our proposed scheme of warning messages. Section VI shows simulation results. Finally, section VII gives conclusions and future work.

II. State of the Art

During the last decades, Intelligent Transportation Systems (ITS) have emerged as an efficient way to improve the performance of the flow of vehicles in the roads. The goals of ITLs are road safety, comfortable driving and distribution of updated information about the roads. Many proposals about ITS have been presented in recent years. In this section some works about ITS in smart cities are highlighted.

The work in [3] is a survey about multifunctional data-driven intelligent transportation system (D^2ITS), which are supported by a large amount of data collected from various

resources: Vision-Driven ITS (input data collected from video sensors and used recognition including vehicle and pedestrian detection); Multisource-Driven ITS (e.g. inductive-loop detectors, laser radar and GPS); Learning-Driven ITS (effective prediction of the occurrence of accidents to enhance the safety of pedestrians by reducing the impact of vehicle collision); and Visualization-Driven ITS (to help decision makers quickly identify abnormal traffic patterns and accordingly take necessary measures).

In [4] and [5] two adaptive traffic light systems based on wireless between vehicles and fixed controller nodes deployed at intersections are designed and developed. These systems improve traffic fluency, reduce the waiting time of vehicles at intersections and help to avoid collisions.

The e-NOTIFY [6] system was designed for automated accident detection, reporting to the Emergencies Coordination Center, and assistance of road accidents using the capabilities offered by vehicular communication technologies. e-NOTIFY focus on improving post collision care with a fast and efficient management of the available emergency resources, which increases the chances of recovery and survival for those injured in traffic accidents.

In this work, we focus on the development of a smart city framework using intelligent infrastructure in the streets, in our case intelligent traffic lights (ITLs). ITLs provide warning messages to the passing vehicles to inform drivers about traffic and weather conditions of the different streets of the city. This way, the smart city framework can help drivers to have a better trip, reducing time to destination, preventing accidents and traffic jams and also saving petrol and reducing pollution. As [3], our proposal manages traffic information seeking to avoid accidents, although the information here is gathered from the vehicles themselves so no further infrastructure is needed. Also our proposal could easily be used by the traffic information centre to design an adaptive traffic light system similar to [4] and [5].

III. Smart City Framework

The smart city framework we have designed includes ITLs set in some of the crossroads. These ITLs collect real-time traffic data from the passing vehicles and calculate traffic statistics such as traffic density in the adjacent streets (between consecutive crossroads). At the same time, these ITLs can communicate the traffic information to passing vehicles and alert them with warning messages in case of accidents. These ITLs also form a sub-network that allows ITLs to share the collected information and calculate statistics of the whole city. Thus, vehicles are well informed of the traffic situation in the city. The following sections describe how this smart city framework is designed and which use the ITL will have.

In the smart city projected, blocks have a regular square design and buildings on its four sides. ITLs are responsible of managing the traffic of the vehicles, which form a VANET. These ITLs do not have to be located at each intersection. Within all the traffic lights that are traditionally located in a city, only a few will be replaced by ITLs. This is because each ITL covers a whole intersection and the 4 streets that converge on this intersection. ITLs are placed as shown in Fig. 1. To cover all this area the antenna pattern used is an omnidirectional propagation pattern. Therefore, each ITL receives data from all passing vehicles on its cover range (the four streets and the intersection). Not having an ITL on each intersection is more economic when implementing this framework.

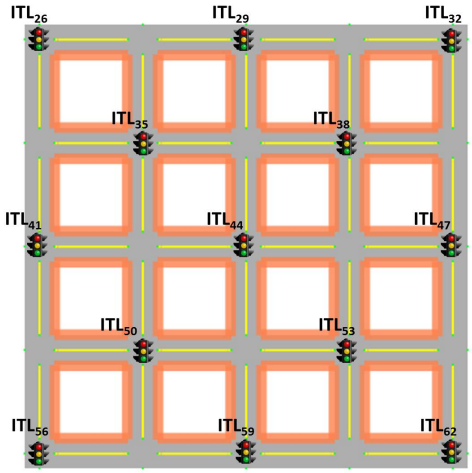

Fig. 1. Intelligent Traffic Lights distribution

It is assumed that vehicles have a global positioning system (GPS) device, a driver assistant device, full map information of the city including the position of the ITLs. Thus, vehicles can easily select which is the nearest ITL.

Every ad-hoc node (i.e., ITLs and vehicles) set on the scenario was configured with Ad hoc On-Demand Distance Vector (AODV) [8] routing protocol. AODV was selected because of its simplicity. Although it is well known that AODV is not suitable as routing protocol of general use in VANETs, there are some applications that might work well with AODV. The advantage of AODV is its simplicity and widespread use. The main drawback is that AODV needs end-to-end paths for data forwarding, which is difficult to handle because in VANETs end-to-end paths last not much due to high speeds of vehicles. Other routing protocols that use other strategies like greedy forwarding and geographical routing. For instance, GPSR (Greedy Perimeter Stateless Routing) [9] and GOSR (Geographical Opportunistic Source Routing) [10] have shown good performance in VANETs, but at the cost of greater complexity and increased delay. Nonetheless, for some applications that require a short delay AODV can perform well. In this paper we are considering smart city services where vehicles send warning messages (weather conditions and traffic density) to the closest ITL, so it is not necessary to establish long paths that last long. Instead, vehicles need to establish very short paths (1-2 hops) to the nearest ITL. Besides, the communication must be quickly since vehicles move fast and the period in coverage range of the ITL is short. Thus, AODV is suitable for our purposes.

TABLE I
FORMAT INFORMATION OF MESSAGE STAT

Type	Statistic Message (SM)	STAT
stat_type	Traffic density (TD_{st})	0
stat_my_id	Car sending statistics	C_i
stat_neighbours	Number of neighbours (NoN)	NoN_i
stat_time	Time of statistics report	t_i
stat_dst	ITL IP address	ITL_i

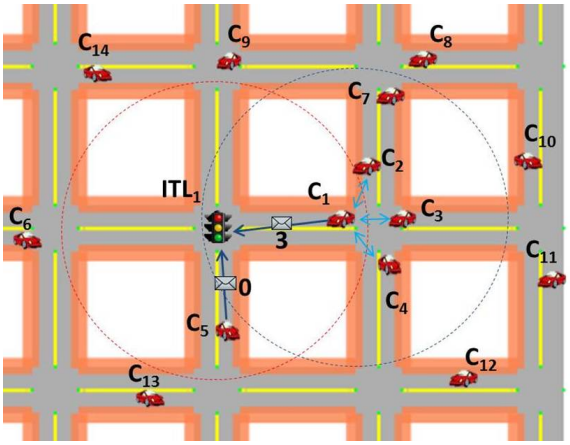

Fig. 2. ITL obtaining traffic statistics in its intersection

IV. MANAGEMENT OF TRAFFIC DENSITY

In this work we focus on the analysis of traffic density, although similar analysis could easily be done for other traffic statistics (e.g. number of passengers, trip time).

The messages sent by each vehicle to an ITL include the type of message (a new message called Statistic Message, SM), the identification of the vehicle (C_i) sending the message, the current value of the number of neighbours (NoN_i) in its coverage range at that moment, the moment in which the message was sent (t_i) and the IP address of the ITL destination (ITL_i). This message is sent by the vehicles each 2 sec. This way, a car (v=40 km/h) sends 5 messages while it crosses a 100 m. street. The format of this message is shown in Table I. The ITL will update the traffic statistics upon the reception of each new message, as it is explained below.

Fig. 2 shows the procedure of getting and sending traffic statistics from the vehicles to the ITLs. Each vehicle interchanges hello messages (HM) with its neighbours and this way it knows the amount of vehicles on its transmission range. Then, the vehicle sends a Statistic Message (SM) with the number of neighbours to the nearest ITL. For example, C_1 counts with three neighbours (C_2, C_3, and C_4). Notice that although C_7 is inside its range they cannot establish any communication because of the buildings that represent obstacles. The car C_5 does not see any neighbour around so it sends a SM to the nearest ITL with a zero on it.

ITL_1 will receive the SMs and will update the traffic density statistics by using an exponential weighted moving average (EWMA) to average current and historical values. Then, ITL_1 will store the results properly and will share its statistics with the others ITLs in the city through the sub-network they form. The day has been divided into five periods due to the usually variable traffic densities in a city throughout the day. Thus, every ITL updates the traffic density per periods: $TDst_{6-9}$, $TDst_{9-12}$, $TDst_{12-15}$, $TDst_{15-18}$, $TDst_{18-21}$. For instance, $TDst_{6-9}$ gathers the average traffic density in the city, during week days, from 06:00 AM to 09:00 PM. The value $TDst_{6-9}$ will continuously be updated using Eq. (1), where w is a small weight (e.g. w=0.25) to smooth out isolated deviations, $\overline{TDst}_{6-9,i}$ is the updated average in iteration i and $TDst_{6-9}$ is the last value received by that ITL. The same computation will be done for the other periods of the day.

$$\overline{TDst}_{6-9,i} = w \cdot \overline{TDst}_{6-9,i-1} + (1-w) \cdot TDst_{6-9} \quad (1)$$

The ITLs of the city share that traffic information and after that, each ITL will send back to each passing vehicle a message with the updated traffic statistics of the city on that period of time. With this information, the driver's assistant device can take proper trip decisions (e.g. avoiding congested roads). Also, data routing protocols may use that information to take suitable forwarding decisions (e.g. forward the packet through denser streets where there are more possible forwarding nodes).

V. MANAGEMENT OF WARNING MESSAGES

In the promising smart cities of near future, communications between vehicles and the city will be constant, including infrastructure-to-infrastructure, car-to-car and infrastructure-to-car communications, by means of city infrastructure and Traffic Information Centres (TICs). These packets will contain different type of information and should be prioritized accordingly. For instance, packets containing information about an accident have to be prioritized over those containing other kind of data such as entertainment data.

Upon the reception of a warning message, a vehicle should consider its current distance to the initial source of the warning message and act consequently. For instance, a car being a long distance away from an accident will not act the same way (i.e., will not brake) when receiving a warning about the accident since it does not affect the immediate security of that car. Nonetheless, that warning message will inform the driver of that car (actually, the driver's assistant device), who may vary the trip plan consequently.

We have implemented a simple warning service to prevent further collisions by alerting drivers about accidents and dangerous road conditions. To achieve that goal, vehicles send short warning messages once one of the situations depicted in Table II has been detected. This information can be obtained from different sources. Regarding weather, data can be collected by a Wireless Sensor Network (WSN) that periodically transmits the weather conditions to the nearest ITL. Also, from small weather stations set in a few ITLs of the city. This information is spread through the city using

TABLE II
WARNING MESSAGES: TRAFFIC AND WEATHER CONDITIONS

Traffic density (2-bit)	Weather (2-bit)	Warning message: reduce speed
Free road segment	Sun	U
	Rain	85% · U
	Storm	65% · U
	Ice	40% · U
Semi-congested road segment	Sun	75% · U
	Rain	50% · U
	Storm	25% · U
	Ice	10% · U
Very congested road segment	Sun	50% · U
	Rain	40% · U
	Storm	30% · U
	Ice	10% · U
Accident	Sun	0
	Rain	0
	Storm	0
	Ice	0

U: Initial driver speed

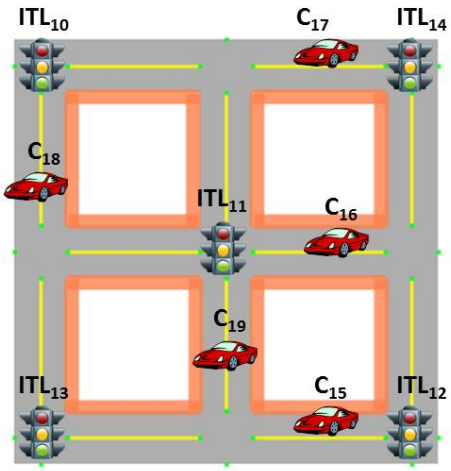

Fig. 3. Simulation scenario of a car accident

the sub-network formed by the ITLs. Complementarily, the forecast proportioned by local public weather services could be used as well. The sub-network of ITLs could share that forecast information obtained from an Internet access point set in one of the ITLs. In case of accident, the vehicle itself (using sensors that detect that the car suffered an accident) communicates this situation to the closest ITL and to the neighbouring vehicles.

To know the traffic density, each ITL uses the statistics collected by the network of ITLs in the city (as explained in the previous section) regarding the average number of neighbours per vehicle in the streets along the day. Thus, depending on the average number of neighbours, two adaptive thresholds determine the traffic density of the road: free, semi-congested or very congested (see Table II).

We use a 4-bit field in the warning messages to code traffic density (2 bits) and weather information (2 bits). The warning message also includes a field with the location of the initial place of the warning message. Nearby vehicles that receive such message will reduce their speed depending on the warning message according to Table II. For instance, in a very congested road with rain condition, warning messages inform nearby vehicles to reduce their speed to 20% of the initial driver speed (U in Table II). The driver's assistant device in the vehicle will make the vehicle brake accordingly.

VI. Simulation Results

In this work, we evaluate the performance of the vehicles of a VANET in a smart city using warning messages and traffic statics managed by the ITLs set in some crossroads of the city. To achieve this evaluations we use the network simulator NCTUns 6.0 [7]

A. Configuration of Intelligent Traffic Lights (ITLs)

ITLs are implemented using Multi-Interface Mobile nodes with two wireless ad-hoc interfaces configured in two different sub-networks. One of the interfaces of the ITL will communicate with vehicles and the other interface will communicate with the other ITLs in the city.

To ensure node connectivity between ITLs, interface two (used to connect with other ITLs) is determined by the distance needed to cover 300 m between crossroads. Interface one (used to connect with vehicles) communicates with the passing vehicles. As the streets have a length of 100 m, and the intersection of 40 m, this interface is configured to cover 130 m. Both interfaces have an omnidirectional propagation diagram. As each vehicle is only supposed to connect to interface one of the ITL and also to other cars, the transmission range of vehicles is configured to cover 130 m in an omnidirectional pattern.

To ensure that it could be possible to differentiate data communication of the two wireless ad-hoc interfaces, they are configured with two different sub-networks. The sub-network used by the ITLs to collect, send and calculate statistics from the vehicles is "1.0.3.XX/24", whereas the sub-network used to communicate ITLs among themselves is "1.0.2.XX/24".

There are only a few ITLs among the regular traffic lights. In this case, an ITL receives data from any passing vehicle from any of the four streets covered by that ITL.

B. Benefits of Using Warning Messages after Accidents

To evaluate the operation of warning messages, we use an urban scenario to show how the vehicles react under different traffic and weather conditions. Fig. 3 shows a neighbourhood of the city where a car accident will happen. The simulation recreates a Manhattan 280x280 m^2 scenario. The length of the streets is 100 m, and the size of every cross is 40 m. These values were chosen to emulate the regular streets in the city of Barcelona, Spain. In the scenario there are 5 vehicles and 5 ITLs. During the simulation, vehicle C_{18} has an accident and remains broken close to the intelligent traffic light ITL_{11} situated in the centre, see Fig. 3. Vehicles C_{15}, C_{16}, C_{17} and C_{19} are all travelling towards ITL_{11}. In this simulation it is shown how the use of ITL helps to avoid collision

TABLE III
TRAFFIC AND WEATHER CONDITIONS DURING THE SIMULATION

Traffic conditions	Weather conditions	Period of time	Average vehicle velocity
Free segment	Sun	30 s	40 km/h
Accident	Sun	50 s	0 km/h

TABLE IV
SIMULATION SETTINGS

Parameter	Value
Medium capacity	11 Mbps
Packet size	256 bytes
Transmission range	130 m
Carrier sense range	180 m
Simulation time	80 sec
MAC specification	IEEE 802.11b
Area	280 x 280 m^2
Maximum Average speed	40 km/h
Number of nodes	5 ITLs and 8 vehicles
Mobility model	CarAgentMod (NCTUns)
Routing protocol	AODV

among the other vehicles and the broken vehicle, thanks to our warning scheme that makes them brake beforehand. Table III summarizes the traffic and weather conditions during the simulation.

The accident will occur in the second 30. The traffic lights number ITL_{10}, ITL_{12}, ITL_{13} and ITL_{14} will send messages of *good weather conditions* and *free traffic segment* during the 80 sec of this simulation. The traffic light number ITL_{11} will send during 30 sec messages of *good weather conditions* and *free traffic segment*. After that, ITL_{11} will send *good weather conditions* and *accident* during the next 50 sec. Each of the ITLs sends these packets to the vehicles in the four streets that go from the crossroads where they are located to the next 4 closest crossroads.

ITLs will broadcast 256 bytes messages every 0.2 seconds (i.e. 5 messages each second) with information about traffic conditions, weather conditions and accident warnings. Vehicles move randomly through the streets at an average velocity of 40 km/h (it automatically decreases when approaching an intersection and the vehicle turns). Simulation settings are summarized in Table IV.

The cars in the simulation were represented using smart vehicles equipped with IEEE 802.11b interface on ad-hoc mode. These vehicles are controlled by a program called *agent* (CarAgent.cc) that makes vehicles move through the city respecting streets, crossroads and traffic lights.

TABLE V
DRIVER'S REACTION TIME AND DISTANCE TRAVELLED

	Use of ITLs	Non use of ITLs
Driver's reaction time	0,084 s	1 s
Distance travelled till reaction	0,93 m	11,11 m
Braking period of time	1,355 s	1,355 s
Distance travelled during braking	7,52 m	7,52 m
Total distance travelled	8,45 m	18,63 m

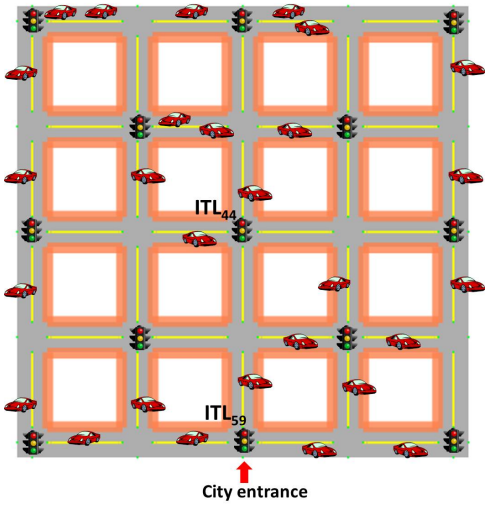

Fig. 4. Traffic density simulation scenario

As it was said in Section III, ad-hoc nodes (i.e. ITLs and vehicles) use AODV. We simply modified AODV to be able to collect traffic statistics while establishing the routing paths. To do that, we use modified RREQ messages that carry SMs (see Table I).

The objective of these simulations is to evaluate if the use of ITLs reduce the driver's reaction time after accidents. According to the *Dirección General de Tráfico* (DGT) [11], responsible of the transportation policy in Spain, the average reaction time of a driver is 1 sec, so a car (v = 40 km/h) before start braking still travels 18,63 m. Using our framework the driver's reaction time was 0,084 sec, which represents that the distance travelled will be reduced to 8,45 m. Table V shows the time and distances that a vehicle, in average, travels with and without the use of our smart city framework. In this case, it can be appreciated that the safety distance from the car to the obstacle has been reduced around 55% from 18,63 m without the use of ITLs to 8,45 m using them, which increases road safety notably.

C. Measure of the Traffic Density in the Smart City

To evaluate the operation of the traffic statistics system, we use a Manhattan scenario with streets that form 5x5 blocks (Fig. 4). It has obstacles that represent buildings, and ITLs which are responsible to manage the traffic of the vehicles that form the VANET.

The simulation consists on a random number of smart vehicles moving around the city and establishing communications with the nearest ITL to send information of the current number of neighbours. Traffic statistics are updated as explained in section IV and according to eq. (1). This data is collected every 2 sec. Every time an ITL receives data from a passing car it updates the statistics of traffic density on its surrounding area, stores it on an individual file and shares it with the rest of the ITLs of the city.

Fig. 5 shows the behaviour of the statistics collected by

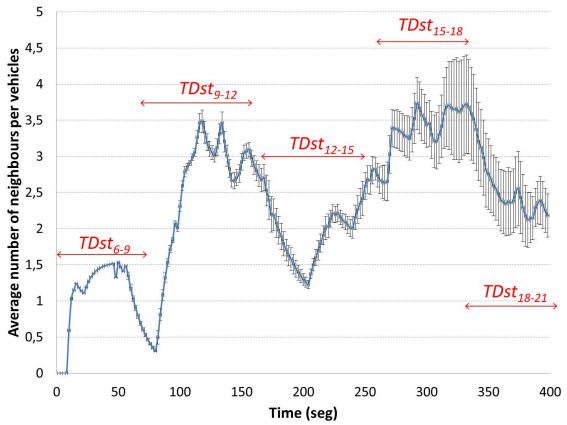

Fig. 5. Average number of neighbours per vehicle measured by ITL_{44}, set in downtown

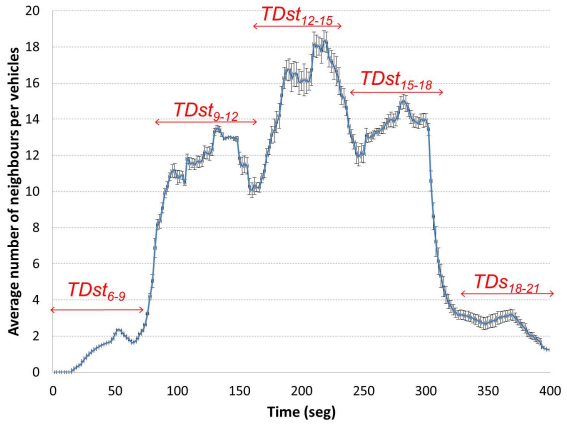

Fig. 6. Average number of neighbours per vehicle measured by ITL_{59}, set in the city entrance

ITL_{44} (set in downtown, see Fig. 5). Here, simulations show 400 sec (i.e. 15 h from 6 AM to 9 PM), so that 27 sec in the simulations represents 1 h. The results show the density of cars in downtown along the day. With this information drivers can obtain which are the roads more congested in each part of the city. Fig. 6 shows the results obtained by ITL_{59}, which is located in the entrance of the city. We can see the behaviour in one day, where the more congested periods of time are between 12-15 PM. Streets are almost free between 6-9 AM. and 18-21 PM.

VII. CONCLUSION AND FUTURE WORK

In this work we have designed a smart city framework for VANETs that include intelligent traffic lights (ITLs) that transmit warning messages and traffic statistics. We have implemented the framework in the NCTUns 6.0 [7] simulator. Simulation results show that the use of ITLs in smart cities can not only improve road safety but also the driver's quality of life. We have explained how the ITLs gather traffic and weather conditions of the roads and how they update those statistics. The goal is that the driver's assistant device can take proper trip decisions, for instance to avoid congested roads, and therefore reducing the trip time and pollution as well. Besides, our smart city framework includes warning messages sent by possible broken vehicles to make approaching vehicles brake beforehand and thus avoid more collisions. Simulation results show the effectiveness of this scheme, reducing the distance to brake and the driver's reaction time.

As a near future work, ITLs could communicate to passing vehicles indicating where are the free parking spots in the city. With this information, the driver assistant device could indicate the driver where free spots are located. This system could use a WSN to get the data about free parking spots and communicate it to the nearest ITLs. The ITLs could share that information though the sub-network they form. This would save trip time, petrol and CO_2 as a consequence, which helps to have sustainable smart cities.

Also, statistics collected by the ITLs can improve data routing protocols selecting the path that offers a higher chance to forward a packet to the destination successfully. We will design a VANET routing protocol that considers those statistics in its operation.

ACKNOWLEDGEMENTS

This work has been partially funded by the Spanish Ministry of Science and Education under the project CICYT CONSEQUENCE (TEC2010-20572-C02-02) and partially supported by the Comissionat per a Universitats i Recerca del DIUE from the Generalitat de Catalunya and the Social European Budget (Fons Social Europeu) with the grant FI-AGAUR; and by the Autonomous University of Sinaloa, México.

REFERENCES

[1] European Network of Living Labs (ENoLL), *http://www.openlivinglabs.eu/*.
[2] Ferrari, G., Busanelli, S., Lotti, N., Kaplan, Y., "Cross-Network Information Dissemination in VANETs", 11th International Conference on ITS Telecommunications, pp. 351-356, 2011.
[3] Junping, Z., Fei-Yue, W., Kunfeng, W., Wei-Hua, L., Xin, X., Cheng, C., "Data-Driven Intelligent Transportation Systems: Survey", IEEE Transactions on Intelligent Transportation Systems, Vol. 12, Issue 4, pp. 1624-1639, 2011.
[4] Maslekar, N., Boussedjra, M., Mouzna, J., Labiod, H., "VANET based Adaptive Traffic Signal Control", IEEE 73rd Vehicular Technology Conference (VTC Spring), pp. 1-5, 2011.
[5] Gradinescu, V., Gorgorin, C., Diaconescu, R., Cristea, V., Iftode, L., "Adaptive Traffic Light Using Car-to-Car communications", IEEE 65th Vehicular Technology Conference (VTC Spring), pp. 21-25, 2007.
[6] Fogue, M., Garrido, P., Martinez, F. J., Cano, J. C., Calafate, C. T., Manzoni, P., Sanchez, M., "Prototyping an Automatic Notification Scheme for Traffic Accidents in Vehicular Networks", Wireless Days (WD) IFIP, pp. 1-5, 2011.
[7] The NCTUns 6.0 Network Simulator and Emulator. Available at *http://NSL.csie.nctu.edu.tw/nctuns.html*.
[8] Perkins, C.E., Belding-Royer, E. M., Das, S.R., "Ad hoc on-demand distance vector (AODV) routing", IEEE Personal Communications, pp. 16-28, 2001.
[9] Karp, B., Kung, H. T., "GPSR: Greedy Perimeter Stateless Routing for wireless Networks", MobiCom 2000.
[10] Zhongyi 1., Tong, Z., Wei, Y., Xiaoming, L., "GOSR: geographical opportunistic source routing for VANETs", ACM SIGMOBILE Mobile Computing and Communications, Vol. 13, Issue 1, 2009.
[11] Dirección General de Tráfico "Revista Tráfico 192", 2011, *http://www.dgt.es/revista/num192/pages/infografias.html*.

ANNEX 2. AODV.CC MODIFIED CODE

```
/*
 * Copyright (c) from 2000 to 2009
 *
 * Network and System Laboratory
 * Department of Computer Science
 * College of Computer Science
 * National Chiao Tung University, Taiwan
 * All Rights Reserved.
 *
 * This source code file is part of the NCTUns 6.0 network simulator.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation is hereby granted (excluding for commercial or
 * for-profit use), provided that both the copyright notice and this
 * permission notice appear in all copies of the software, derivative
 * works, or modified versions, and any portions thereof, and that
 * both notices appear in supporting documentation, and that credit
 * is given to National Chiao Tung University, Taiwan in all publications
 * reporting on direct or indirect use of this code or its derivatives.
 *
 * National Chiao Tung University, Taiwan makes no representations
 * about the suitability of this software for any purpose. It is provided
 * "AS IS" without express or implied warranty.
 *
 * A Web site containing the latest NCTUns 6.0 network simulator software
 * and its documentations is set up at http://NSL.csie.nctu.edu.tw/nctuns.html.
 *
 * Project Chief-Technology-Officer
 *
 * Prof. Shie-Yuan Wang <shieyuan@csie.nctu.edu.tw>
 * National Chiao Tung University, Taiwan
 *
 * 09/01/2009
 */

/*
 *    Ad hoc On-Demand Distance Vector (AODV) Routing
 *    reference: draft-ietf-manet-aodv-12.txt
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/uio.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <assert.h>
#include <object.h>
```

```
#include <event.h>
#include <regcom.h>
#include <scheduler.h>
#include <nodetype.h>
#include <timer.h>
#include <nctuns_api.h>
#include <ethernet.h>
#include <ip.h>
#include <random.h>
#include <packet.h>
#include <route/aodv/AODV.h>
#include <mbinder.h>

using namespace AODVd;

#define LINK_LAYER_RETRY
/* #define LINK_LAYER_DROP */

extern   RegTable        RegTable_;
extern   typeTable       *typeTable_;
extern   scheduler       *scheduler_;

FILE     *fdtA;
                  //M.Mateos
FILE     *fdtD;
                  //M.Mateos
FILE     *fdtE;
                  //M.Mateos
FILE     *fdtF;
                  //M.Mateos
FILE     *fdtG;
                  //M.Mateos
FILE     *fdtH;
                  //M.Mateos
FILE     *fdtI;
                  //M.Mateos
FILE     *fdtJ;
                  //M.Mateos
FILE     *fdtK;
                  //M.Mateos

MODULE_GENERATOR(AODV);

AODV::AODV(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
          : NslObject(type, id, pl, name)
{
     s_flowctl = DISABLED;
     r_flowctl = DISABLED;

          rreq_id = 0; /* ? */
          link_fail_list.slh_first = 0;
     bcache.slh_first = 0;

          acc_rreq = 0;
          acc_rerr = 0;
```

```
        qcur_ =0;
        rd_head = rd_tail = 0;
        qmax_ = 5;

        /* bind input file name */
        vBind("HELLO_INTERVAL",      &HELLO_INTERVAL);
    vBind("ALLOWED_HELLO_LOSS",   &ALLOWED_HELLO_LOSS);
        vBind("ACTIVE_ROUTE_TIMEOUT", &ACTIVE_ROUTE_TIMEOUT);
        vBind("DELETE_PERIOD",       &DELETE_PERIOD);
        vBind("NET_DIAMETER",        &NET_DIAMETER);
        vBind("NODE_TRAVERSAL_TIME",  &NODE_TRAVERSAL_TIME);
        vBind("RREQ_RETRIES",        &RREQ_RETRIES);
        vBind("RREQ_RATELIMIT",      &RREQ_RATELIMIT);
        vBind("RERR_RATELIMIT",      &RERR_RATELIMIT);
}

AODV::~AODV() {
}

int AODV::init() {

        mip = GET_REG_VAR(get_port(), "IP", u_long *);

        Rt_entry *r = new Rt_entry;

        // the first entry is for its own
        r->rt_dst     = *mip;
        r->rt_nexthop = *mip;
        r->rt_valid_dst_seqno = true;
        r->rt_seqno   = 1;
        r->rt_hopcount= 0;
        r->rt_flags   = RTF_VALID;
        r->rt_time    = INFINITY_LIFETIME;

        rtable.insert(r);

        MILLI_TO_TICK(hello_interval_,  (u_int64_t)HELLO_INTERVAL);
        MILLI_TO_TICK(active_route_timeout_, (u_int64_t)ACTIVE_ROUTE_TIMEOUT);
        MILLI_TO_TICK(node_traversal_time_, (u_int64_t)NODE_TRAVERSAL_TIME);
        MILLI_TO_TICK(delete_period_, (u_int64_t)DELETE_PERIOD);

        MY_ROUTE_TIMEOUT = 2 * ACTIVE_ROUTE_TIMEOUT;
        MILLI_TO_TICK(my_route_timeout_, (u_int64_t)MY_ROUTE_TIMEOUT);
        NET_TRAVERSAL_TIME = (3 * NODE_TRAVERSAL_TIME * NET_DIAMETER /2);
        MILLI_TO_TICK(net_traversal_time_, (u_int64_t)NET_TRAVERSAL_TIME);
        PATH_DISCOVERY_TIME = (2 * NET_TRAVERSAL_TIME);
        MILLI_TO_TICK(path_discovery_time_,  (u_int64_t)PATH_DISCOVERY_TIME);

        MILLI_TO_TICK(route_check_timer_,  (u_int64_t)ROUTE_CHECK);
        MILLI_TO_TICK(rreq_check_timer_,  (u_int64_t)PENDING_RREQ_CHECK);
        MILLI_TO_TICK(recent_rreq_list_timer_,  (u_int64_t)RECENT_RREQ_LIST_CHECK);
        MILLI_TO_TICK(accumulated_rreq_rerr_timer_,
(u_int64_t)ACCUMULATED_RREQ_RERR_TIMER);
        MILLI_TO_TICK(nei_list_check_timer_,  (u_int64_t)NEI_LIST_CHECK);
        MILLI_TO_TICK(nei_list_stat_check_timer_, (u_int64_t)NEI_LIST_STAT_CHECK);
                        //M.Mateos
```

```
MILLI_TO_TICK(delay_stat_check_timer_, (u_int64_t)DELAY_STAT_CHECK);
                    //M.Mateos
MILLI_TO_TICK(sendhello_timer_,  (u_int64_t)SENDHELLO_TIMER);
MILLI_TO_TICK(link_fail_list_check_timer_,  (u_int64_t)LINK_FAIL_LIST_CHECK);
/* MILLI_TO_TICK(printLoc_timer_,  (u_int64_t)5000); */

BASE_OBJTYPE(type);
type = POINTER_TO_MEMBER(AODV, sendHello);
SendHello_timer.setCallOutObj(this, type);
SendHello_timer.start((sendhello_timer_ + Random()%100000), 0);

BASE_OBJTYPE(type_r);
type_r = POINTER_TO_MEMBER(AODV, RTTimer);
RT_timer.setCallOutObj(this, type_r);
RT_timer.start(route_check_timer_, 0);

BASE_OBJTYPE(type_rreq);
type_rreq = POINTER_TO_MEMBER(AODV, RREQ_retry);
SendRREQ_timer.setCallOutObj(this, type_rreq);
SendRREQ_timer.start(rreq_check_timer_, 0);

BASE_OBJTYPE(type_recent_rreq);
type_recent_rreq = POINTER_TO_MEMBER(AODV, CheckRecentRREQ);
RecentRREQ_timer.setCallOutObj(this, type_recent_rreq);
RecentRREQ_timer.start(recent_rreq_list_timer_, 0);

BASE_OBJTYPE(type_acc_rreq_rerr);
type_acc_rreq_rerr = POINTER_TO_MEMBER(AODV, ClearAccRREQ_RERR);
AccRREQ_RERR_timer.setCallOutObj(this, type_acc_rreq_rerr);
AccRREQ_RERR_timer.start(accumulated_rreq_rerr_timer_, 0);


/*
 * If an AODV network has greedy traffic flows, the hello
 * packets will frequently collide with data packets and
 * not be received successfully. This will casue the
 * CheckNeiList function to frequently detect the neighbor-
 * losing event so that many RRER packets are sent out.
 * The uncessary RRER packets may seriously reduce the
 * AODV network's connectiviy; therefore, we disable this
 * function here.
 */
/*
BASE_OBJTYPE(type_nei_list);
type_nei_list = POINTER_TO_MEMBER(AODV, CheckNeiList);
Nei_List_timer.setCallOutObj(this, type_nei_list);
Nei_List_timer.start(nei_list_check_timer_, 0);
*/

BASE_OBJTYPE(type_nei_list_stat);
                    //M.Mateos
type_nei_list_stat = POINTER_TO_MEMBER(AODV, CheckNeiList_Stat);
                    //M.Mateos
Nei_List_Stat_timer.setCallOutObj(this, type_nei_list_stat);
          //M.Mateos
```

```
	Nei_List_Stat_timer.start(nei_list_stat_check_timer_, 0);
		//M.Mateos

	BASE_OBJTYPE(type_delay_stat);
			//M.Mateos
	type_delay_stat = POINTER_TO_MEMBER(AODV, CheckDelay_Stat);
			//M.Mateos
	Delay_Stat_timer.setCallOutObj(this, type_delay_stat);
			//M.Mateos
	Delay_Stat_timer.start(delay_stat_check_timer_, 0);
			//M.Mateos

	BASE_OBJTYPE(type_link_fail_list);
	type_link_fail_list = POINTER_TO_MEMBER(AODV, CheckLinkFailList);
	RecentRREQ_timer.setCallOutObj(this, type_link_fail_list);
	RecentRREQ_timer.start(link_fail_list_check_timer_, 0);

	/* BASE_OBJTYPE(type_loc);
	type_loc = POINTER_TO_MEMBER(AODV, PrintLoc);
	PrintLoc_timer.setCallOutObj(this, type_loc);
	PrintLoc_timer.start(printLoc_timer_, 0);  */

	return (1);
}

int AODV::RTTimer(){

	u_int64_t nowtime = GetCurrentTime();

	// skip the route entry for its own
	Rt_entry *p_pre = rtable.rt_getHead();
	Rt_entry *p_rt  = rtable.rt_getHead()->next;
	Rt_entry *p_tmp;

	while(p_rt) {
		if(p_rt->rt_flags == RTF_VALID) {
			// turn into INVALID state
			if(p_rt->rt_time < nowtime) {
				p_rt->rt_flags = RTF_INVALID;
				p_rt->rt_time = nowtime + delete_period_;

				// if the entry is for a neighbor(hopcount=1),
				// we delete the corresponding entry in the
				// neighborr-list.
				/*if(p_rt->rt_hopcount == 1) {
					nei_list.remove(p_rt->rt_dst);
				}*/

			}

			p_pre = p_rt;
			p_rt  = p_rt->next;

		}else if(p_rt->rt_flags == RTF_INVALID) {
			if(p_rt->rt_time < nowtime) {
				p_tmp = p_rt->next;
```

```
                                rtable.removeEntry(p_rt, p_pre);
                                p_rt  = p_tmp;
                        }else {
                                p_pre = p_rt;
                                p_rt  = p_rt->next;
                        }
                }else {
                        p_pre = p_rt;
                        p_rt  = p_rt->next;
                }

                //p_pre = p_rt;
                //p_rt  = p_rt->next;
        }

        BASE_OBJTYPE(type_r);
        type_r = POINTER_TO_MEMBER(AODV, RTTimer);
        RT_timer.setCallOutObj(this, type_r);
        RT_timer.start(route_check_timer_, 0);

        return (1);
}

int AODV::RREQ_retry(){

        u_int64_t nowtime = GetCurrentTime();
        Ctrl_entry *p_ctrl = ctrl_table.getHead();
        Ctrl_entry *p_ctrl_deleted;

        while(p_ctrl) {
                if(p_ctrl->rreq_lifetime <= nowtime) {

                        if(p_ctrl->rreq_retries < (unsigned int)RREQ_RETRIES) {

                                if(acc_rreq <= RREQ_RATELIMIT){
                                        /* send retry RREQ */
                                        sendRREQ(p_ctrl->dst_ip, NET_DIAMETER);
                                        acc_rreq++;

                                        p_ctrl->rreq_retries += 1 ;

                                        p_ctrl->rreq_lifetime = nowtime + net_traversal_time_;
                                }
                        }else{
                                /* free all queued pkts and remove the ctrl entry */
                                struct buf_list *b_list = p_ctrl->buffer_list;
                                for(; b_list != NULL ; b_list = b_list->next){
                                        freePacket(b_list->queued_pkt);
                                }

                                p_ctrl_deleted = p_ctrl;
                                p_ctrl = p_ctrl->next;
                                ctrl_table.remove(p_ctrl_deleted->dst_ip);
                                continue;
                        }
                }
```

```
                p_ctrl = p_ctrl->next;
        }

        LocalRepair_entry *p_local_rep = local_repair_table.getHead();
        LocalRepair_entry *p_local_rep_deleted;

        // for local repair
        while(p_local_rep) {
                if(p_local_rep->local_repair_lifetime <= nowtime){
                /* local repair fails, free all queued pkts and remove the ctrl entry */

                        // turn the flag of corresponding route entry
                        // from RTF_BEING_REPAIRED into RTF_INVALID
                        Rt_entry *p_rt =rtable.rt_get(p_local_rep->dst_ip);
                        if(p_rt) {
                                p_rt->rt_flags = RTF_INVALID;

                                // shortage of rfc.3561
                                // Without the decrease, after local-repair
                                // fails, no AODV-pkt can update this route.
                                // Since it is equipped with the highest dst
                                // seqno of the network
                                p_rt->rt_seqno--;
                        }

                        struct buf_list *b_list = p_local_rep->buffer_list;
                        for(; b_list != NULL ; b_list = b_list->next){
                                freePacket(b_list->queued_pkt);
                        }


                        // for the link-layer upcall, we generate
                        // the unreach-list, then send RERR
                        if(p_local_rep->brokenlink_node != NO_USE) {
                                Unreach_list *unr_list = new Unreach_list();

                                Rt_entry *p_rt = rtable.rt_getHead();
                                while(p_rt) {
                                        if(p_rt->rt_nexthop == p_local_rep->brokenlink_node)
                                        {
                                                if(p_rt->rt_flags == RTF_VALID){
                                                        p_rt->rt_seqno++;
                                                        p_rt->rt_flags = RTF_INVALID;
                                                        p_rt->rt_time  = nowtime +
delete_period_;
                                                }
                                                unr_list->insert(p_rt->rt_dst, p_rt->rt_seqno);
                                        }

                                        p_rt = p_rt->next;
                                }

                                if(acc_rerr < RERR_RATELIMIT) {
                                        bcastRERR(unr_list);
                                        acc_rerr++;
```

```
                                        }

                                        delete unr_list;
                                }

                                p_local_rep_deleted = p_local_rep;
                                p_local_rep = p_local_rep->next;
                                local_repair_table.remove(p_local_rep_deleted->dst_ip);
                                continue;
                        }
                        p_local_rep = p_local_rep->next;
                }

                BASE_OBJTYPE(type_rreq);
                type_rreq = POINTER_TO_MEMBER(AODV, RREQ_retry);
                SendRREQ_timer.setCallOutObj(this, type_rreq);
                SendRREQ_timer.start(rreq_check_timer_, 0);

                return (1);
}
/* end of revise */

int AODV::CheckRecentRREQ() {
                BroadcastID *b;
                u_int64_t    now = GetCurrentTime();

                SLIST_FOREACH(b, &bcache, nexB){
                        if(b->lifetime < now)
                                SLIST_REMOVE(&bcache, b, BroadcastID, nexB);
                }


                BASE_OBJTYPE(type_recent_rreq);
                type_recent_rreq = POINTER_TO_MEMBER(AODV, CheckRecentRREQ);
                RecentRREQ_timer.setCallOutObj(this, type_recent_rreq);
                RecentRREQ_timer.start(recent_rreq_list_timer_, 0);

                return (1);
}

// clear the accumulated count for RRER/RERR to 0 every second
int AODV::ClearAccRREQ_RERR() {
                acc_rreq = 0;
                acc_rerr = 0;

                BASE_OBJTYPE(type_acc_rreq_rerr);
                type_acc_rreq_rerr = POINTER_TO_MEMBER(AODV, ClearAccRREQ_RERR);
                AccRREQ_RERR_timer.setCallOutObj(this, type_acc_rreq_rerr);
                AccRREQ_RERR_timer.start(accumulated_rreq_rerr_timer_, 0);

                return (1);
}

// regularly check link connection of the neighbors
int AODV::CheckNeiList() {
                Nei_entry *p_nei   = nei_list.getHead();
```

```
Nei_entry *p_nei_deleted;
u_int64_t  nowtime = GetCurrentTime();


while(p_nei) {
        // no Hello is received for ALLOWED_HELLO_LOSS*HELLO_INTERVAL
        if(p_nei->nei_time < nowtime) {
                Unreach_list *unr_list = new Unreach_list();

                Rt_entry *p_rt_nei = rtable.rt_get(p_nei->nei_addr);
                if(p_rt_nei) {

                        /* if(p_rt_nei->rt_valid_dst_seqno == true)
                                p_rt_nei->rt_seqno++; */
                        // invalidate route entry for the lost neighbor
                        p_rt_nei->rt_flags = RTF_INVALID;
                        p_rt_nei->rt_time  = nowtime + delete_period_;
                }

                Rt_entry *p_rt = rtable.rt_getHead();
                // insert those route entries which use the lost
                // neighbor as next hop into unreach-list
                while(p_rt) {
                        if(p_rt->rt_nexthop == p_nei->nei_addr) {
                                unr_list->insert(p_rt->rt_dst, p_rt->rt_seqno);
                                // link is broken, mark RTF_REPAIRABLE
                                p_rt->rt_flags = RTF_REPAIRABLE;
                                p_rt->rt_time  = GetCurrentTime() + delete_period_;
                        }

                        p_rt = p_rt->next;
                }
                if(acc_rerr <= RERR_RATELIMIT) {
                        bcastRERR(unr_list);
                        acc_rerr++;
                }

                delete unr_list;

                p_nei_deleted = p_nei;
                p_nei = p_nei->next;
                nei_list.remove(p_nei_deleted->nei_addr);

                /*
                 * The local repair fuction is not provided yet.
                 */
                // local repair
                //sendRREQ();
                continue;
        }
        p_nei = p_nei->next;
}

BASE_OBJTYPE(type_nei_list);
type_nei_list = POINTER_TO_MEMBER(AODV, CheckNeiList);
Nei_List_timer.setCallOutObj(this, type_nei_list);
```

```
	Nei_List_timer.start(nei_list_check_timer_, 0);

	return 0;
}

int AODV::CheckNeiList_Stat() {

	u_int64_t nowtime = GetCurrentTime();
					//M.Mateos
	int neighbors = 0, my_id = 0, light_id = 0;
				//M.Mateos
	u_long light_addr = 0;
					//M.Mateos

	my_id = get_nid();
					//M.Mateos
	nei_list.stat(nowtime, my_id, &neighbors, &light_addr, &light_id);
				//M.Mateos
	if(/*neighbors > 0 &&*/ light_addr != 0)
				//M.Mateos
		sendSTAT(1,my_id,neighbors,nowtime,light_addr,0);
					//M.Mateos

	BASE_OBJTYPE(type_nei_list_stat);
					//M.Mateos
	type_nei_list_stat = POINTER_TO_MEMBER(AODV, CheckNeiList_Stat);
					//M.Mateos
	Nei_List_Stat_timer.setCallOutObj(this, type_nei_list_stat);
				//M.Mateos
	Nei_List_Stat_timer.start(nei_list_stat_check_timer_, 0);
				//M.Mateos

	return 0;
					//M.Mateos
}
					//M.Mateos


int AODV::CheckDelay_Stat() {

	Nei_entry *p_nei   = nei_list.getHead();
				//M.Mateos

	int		my_id = 0, Position = 0, Pos = 0, id = 0;
				//M.Mateos
	double		nowtime = 0;
					//M.Mateos
	u_long		light_addr = 0, reference_light_addr = 0;
				//M.Mateos
	char		*Prueba,Time_[9],Name[60],Name1[60],Boolean_[2];
					//M.Mateos
	double		Time,delay,Boolean;
					//M.Mateos
	u_int64_t	light_time = 0, reference_light_time = 0;
				//M.Mateos
```

```
my_id = get_nid();
                        //M.Mateos

if(my_id >= 65 && my_id <= 121) {
                //M.Mateos
        sprintf(Name, "/home/nctuns/Escritorio/Resultados/Delay/C-%i",my_id);
                //M.Mateos
        fdtH = fopen(Name,"a+");
                //M.Mateos
        fseek(fdtH,0,SEEK_END);
                //M.Mateos
        Pos = ftell(fdtH);
                //M.Mateos
        if(Pos > 0) {
                //M.Mateos
                fseek(fdtH,0,SEEK_SET);
                //M.Mateos
                fscanf(fdtH,"%s",Boolean_);
                //M.Mateos
                fseek(fdtH,0,SEEK_SET);
                //M.Mateos
                fprintf(fdtH,NULL);
                //M.Mateos
                fclose(fdtH);
                //M.Mateos
                Boolean = strtod(Boolean_,&Prueba);
                //M.Mateos
                if(Boolean == 1) {
                //M.Mateos
                        nowtime = GetCurrentTime();
                        //M.Mateos
                        sprintf(Name1, "/home/nctuns/Escritorio/Resultados/Delay/Delay-
Car-%i",get_nid());             //M.Mateos
                        fdtI = fopen(Name1,"a+");
                //M.Mateos
                        fseek(fdtI,0,SEEK_END);
                        //M.Mateos
                        if(ftell(fdtI) == 0) {
                        //M.Mateos
                                fprintf(fdtI,"%lf\n",(double)(nowtime*TICK/1000000000.0));
                //M.Mateos
                                delay = 0;
                        //M.Mateos
                        }
                        //M.Mateos
                        else {
                        //M.Mateos
                                Position = ftell(fdtI) - (sizeof(double) + 1);
                //M.Mateos
                                fseek(fdtI,Position,SEEK_SET);
                        //M.Mateos
                                fscanf(fdtI,"%s",Time_);
                //M.Mateos
                                Time = strtod(Time_,&Prueba);
                        //M.Mateos
```

```
                                        delay = (double)(nowtime*TICK/1000000000.0) - Time;
                                //M.Mateos
                                        fseek(fdtI,0,SEEK_END);
                                //M.Mateos
                                        if((double)(nowtime*TICK/1000000000.0) < 10) {
                                //M.Mateos

        fprintf(fdtI,"%lf\n",(double)(nowtime*TICK/1000000000.0));                      //M.Mateos
                                        } else if ((double)(nowtime*TICK/1000000000.0) >= 10 &&
(double)(nowtime*TICK/1000000000.0) < 100) {

        fprintf(fdtI,"%2.5f\n",(double)(nowtime*TICK/1000000000.0));
        //M.Mateos
                                        } else if ((double)(nowtime*TICK/1000000000.0) >= 100 &&
(double)(nowtime*TICK/1000000000.0) < 1000) {

        fprintf(fdtI,"%3.4f\n",(double)(nowtime*TICK/1000000000.0));
        //M.Mateos
                                        }
                                //M.Mateos
                                }
                                //M.Mateos
                                fclose(fdtI);
                                //M.Mateos
                                while(p_nei) {
                                //M.Mateos
                                        if(p_nei->nei_time >= nowtime) {
                        //M.Mateos
                                                id = ipv4addr_to_nodeid(p_nei->nei_addr);
                        //M.Mateos
                                                if(id == 26 || id == 29 || id == 32 || id == 35 || id ==
38 || id == 41 || id == 44 || id == 47 || id == 50 || id ==
                        53 || id == 56 || id == 59 || id == 62) {                               //M.Mateos
                                                        light_addr = p_nei->nei_addr;
                                        //M.Mateos
                                                        light_time = p_nei->nei_time;
                                        //M.Mateos
                                                        if(light_time > reference_light_time) {
                                        //M.Mateos
                                                                reference_light_time = light_time;
                        //M.Mateos
                                                                reference_light_addr = light_addr;
                        //M.Mateos
                                                        }
                                        //M.Mateos
                                                }
                                        //M.Mateos
                                        }
                                //M.Mateos
                                        p_nei = p_nei->next;
                                //M.Mateos
                                }
                                //M.Mateos
                                if(reference_light_addr != 0) {
                                //M.Mateos
```

```
                                        sendSTAT(2,my_id,0,nowtime,reference_light_addr,delay);
                                //M.Mateos
                                }
                                //M.Mateos
                        }
                                //M.Mateos
                }
                                //M.Mateos
                else {
                                //M.Mateos
                        fclose(fdtH);
                                //M.Mateos
                }
                                //M.Mateos
                remove(Name);
                                //M.Mateos
        }
                                //M.Mateos

        BASE_OBJTYPE(type_delay_stat);
                                //M.Mateos
        type_delay_stat = POINTER_TO_MEMBER(AODV, CheckDelay_Stat);
                                //M.Mateos
        Delay_Stat_timer.setCallOutObj(this, type_delay_stat);
                                //M.Mateos
        Delay_Stat_timer.start(delay_stat_check_timer_, 0);
                                //M.Mateos

        return 0;
                                //M.Mateos

}

int AODV::CheckLinkFailList() {
        struct Link_fail_entry *l;
        u_int64_t    now = GetCurrentTime();

        SLIST_FOREACH(l, &link_fail_list, next){
                if(l->lifetime < now)
                        SLIST_REMOVE(&link_fail_list, l, Link_fail_entry, next);
        }

        BASE_OBJTYPE(type_link_fail_list);
        type_link_fail_list = POINTER_TO_MEMBER(AODV, CheckLinkFailList);
        RecentRREQ_timer.setCallOutObj(this, type_link_fail_list);
        RecentRREQ_timer.start(link_fail_list_check_timer_, 0);

        return 0;
}

int
AODV::recv(ePacket_ *pkt) {

        u_long dst_ip, src_ip;

        Packet *p;
```

```
struct AODV_packet *my_pkt;
Rt_entry *dst_route;
/* int lastseqno; */

assert(pkt&&(p=(Packet *)pkt->DataInfo_));
GET_PKT(p, pkt);

char pkttype[5];
strncpy(pkttype, p->pkt_get(), 4);
pkttype[4]='\0';

my_pkt = (struct AODV_packet *)p->pkt_get();


if(strcmp(pkttype,"AODV") == 0){
dst_ip = my_pkt->dst_ip;
        src_ip = my_pkt->src_ip;
}
else {
        IP_DST(dst_ip, p->pkt_sget());
        IP_SRC(src_ip, p->pkt_sget());

}

/*  Receive normal packet, we must help it forward to
*   next node ,or if it is my packet, I pass it to interface layer.
*/
if(bcmp(my_pkt->pro_type, "AODV", 4) != 0 ) {

        /* hwchu: moved to below
        struct ip *iphdr;
iphdr = (struct ip *)p->pkt_sget();
--iphdr->ip_ttl;
if (iphdr->ip_ttl == 0) {
                return (put(pkt, recvtarget_));
        }
        */

        // use the src_ip of the normal pkt as index,
        // to update the lifetime of corresponding route entry
        /* Rt_entry *p_rt_src = rtable.rt_get(src_ip);
        p_rt_src->rt_flags = RTF_VALID;
        p_rt_src->rt_time  = GetCurrentTime() + active_route_timeout_; */

/*
*   if I am dst or it's a broadcast pkt, pass to upper layer
        */
if((dst_ip == *mip) || is_ipv4_broadcast(get_nid(), dst_ip)){
                return (put(pkt, recvtarget_));
        } else {
                /* hwchu:
                 *   There is no chance for this packet to enter the kernel,
                 *   so we decrement its TTL here.
                 */
                u_char ttl;
```

```
                GET_IP_TTL(ttl, p->pkt_sget());
                if (ttl <= 1) {
                        return put(pkt, recvtarget_);
                }
                IP_DEC_TTL(p->pkt_sget());
        }

        int lookup_result = rtable.rt_lookup(dst_ip);
 if(lookup_result == RTF_VALID){
                dst_route = rtable.rt_get(dst_ip);

                // each time the route is used for forwarding,
                // update its lifetime
                dst_route->rt_time = GetCurrentTime() + active_route_timeout_;

                p->rt_setgw(dst_route->rt_nexthop);
                p->pkt_setflow(PF_SEND);

                return (sendToQueue(pkt));
        }
/* else {
                if (ctrl_table.ifExist(dst_ip) < 0) {
                ctrl_table.addEntry(dst_ip, 0);
                }
                ctrl_table.attachPKT(dst_ip,pkt);

                sendRERR(dst_ip);
                return(1);
    } */
 else if(lookup_result == RTF_INVALID){

                Unreach_list *unr_list = new Unreach_list();
                Rt_entry *rt_unreach   = rtable.rt_get(dst_ip);

                unr_list->insert(dst_ip, rt_unreach->rt_seqno);

                Nei_entry *p_nei = rt_unreach->rt_preclist->getHead();
                // sendRERR to each precursors of rt_entry for
                // the dst_ip of the received data pkt.
                while(p_nei) {
                        if(acc_rerr <= RERR_RATELIMIT) {
                                sendRERR(p_nei->nei_addr, unr_list);
                                acc_rerr++;
                        }

                        p_nei = p_nei->next;
                }

                delete unr_list;

        }else if(lookup_result == RTF_REPAIRABLE){
                dst_route = rtable.rt_get(dst_ip);

        if (!local_repair_table.ifExist(dst_ip)) {
```

```
                                local_repair_table.insert(dst_ip,
GetCurrentTime()+net_traversal_time_, NO_USE);
                                if(local_repair_table.attachPkt(dst_ip, pkt) < 0){
                                        freePacket(pkt);
                                        return 1;
                                }

                                if(acc_rreq <= RREQ_RATELIMIT) {
                                        // local repair
                                        dst_route->rt_seqno++;
                                sendRREQ(dst_ip, dst_route->rt_hopcount + TTL_THRESHOLD);
                                        acc_rreq++;

                                        dst_route->rt_flags = RTF_BEING_REPAIRED;
                                }
                                /* cclin: need to be examined to see if
                                 * we should free the packet before this function
                                 * returns.
                                 */
                                freePacket(pkt);
                                return (1);

                }else{
                                if(local_repair_table.attachPkt(dst_ip, pkt) < 0)
                                        freePacket(pkt);
                                return (1);
                        }
                }else if(lookup_result == RTF_BEING_REPAIRED){

                        // These pkts being received will be queued,
                        // may be dropped after repairing timeout.
                if (!local_repair_table.ifExist(dst_ip)) {
                                local_repair_table.insert(dst_ip,
GetCurrentTime()+net_traversal_time_, NO_USE);
                                if(local_repair_table.attachPkt(dst_ip, pkt) < 0)
                                        freePacket(pkt);

                                return (1);
                }else{
                                if(local_repair_table.attachPkt(dst_ip, pkt) < 0)
                                        freePacket(pkt);
                                return (1);
                        }

                }else if(lookup_result == RT_NOT_EXIST){
                }

                /* cclin: need to be examined to see if
                 * we should free the packet before this function
                 * returns.
                 */
                freePacket(pkt);
                return (1);
        }
```

```
if(bcmp(my_pkt->pro_type, "AODV_RREQ", 10) == 0){

        struct RREQ_msg  *my_rreq;

        my_rreq = (struct RREQ_msg *)((char *)my_pkt+sizeof(AODV_packet));

        // create/update immediate reverse route to previous hop
        updateSimpleRRoute(src_ip);

        // not entirely conform to the spec ><
        if(my_rreq->rreq_src_addr == *mip) {
                freePacket(pkt);
                return (1);
        }

        // We must update the broadcast id cache.
        bool found_in_bcache = false;
        BroadcastID *b;
        SLIST_FOREACH(b, &bcache, nexB){
                if(b->addr == my_rreq->rreq_src_addr) {

                        found_in_bcache = true;

                        if(b->bid > my_rreq->rreq_id) {
                                freePacket(pkt); // free duplicate RREQ
                                return (1);
                        }else if(b->bid == my_rreq->rreq_id) {
                                // Let lower-hopcnt RREQ pass through
                                // to achieve shortest-path routing.
                                if(b->hopcnt <= my_rreq->rreq_hopcount){
                                        freePacket(pkt);
                                        return (1);
                                }else
                                        b->hopcnt = my_rreq->rreq_hopcount;
                        }else {
                                b->bid = my_rreq->rreq_id;
                        }
                }
        }

        if(!found_in_bcache) {
        BroadcastID *b = new struct BroadcastID;

                b->addr = my_rreq->rreq_src_addr;
                b->bid = my_rreq->rreq_id;
                b->hopcnt = my_rreq->rreq_hopcount;
                b->lifetime = GetCurrentTime() + path_discovery_time_;
                SLIST_INSERT_HEAD(&bcache, b, nexB);
        }


        my_rreq->rreq_hopcount++;

        u_int64_t minimalLifetime = GetCurrentTime() + 2*net_traversal_time_ - 2*(my_rreq-
>rreq_hopcount)*node_traversal_time_;
```

```
		// add route to RREQ originator(ie. reverse route )
		if((rtable.rt_lookup(my_rreq->rreq_src_addr)) == RT_NOT_EXIST) {
			Rt_entry *r = new Rt_entry;

			r->rt_dst		= my_rreq->rreq_src_addr;
			r->rt_nexthop		= src_ip;
			r->rt_valid_dst_seqno = true;
			r->rt_seqno		= my_rreq->rreq_src_seqno;
			r->rt_hopcount		= my_rreq->rreq_hopcount;
			r->rt_flags		= RTF_VALID;
			r->rt_time		= minimalLifetime;

			rtable.insert(r);
		}
		else {
			Rt_entry *rt1 = rtable.rt_get(my_rreq->rreq_src_addr);
			/* if((rt1.rt_hopcount > my_rreq->rreq_hopcount)||
			(rt1.rt_seqno < my_rreq->rreq_src_seqno)) {
				updateRT(src_ip,my_rreq,lastip, lastseqno);
			} */

			rt1->rt_nexthop		= src_ip;
			if(rt1->rt_valid_dst_seqno)
				rt1->rt_seqno = (my_rreq->rreq_src_seqno > rt1->rt_seqno) ?
my_rreq->rreq_src_seqno : rt1->rt_seqno;
			else
				rt1->rt_seqno = my_rreq->rreq_src_seqno;

			rt1->rt_valid_dst_seqno = true;

			rt1->rt_hopcount	= my_rreq->rreq_hopcount;
			rt1->rt_flags		= RTF_VALID; // ?
			rt1->rt_time		= (minimalLifetime > rt1->rt_time) ? minimalLifetime : rt1-
>rt_time;
		}

	processBuffered(my_rreq->rreq_src_addr);
	processBuffered(src_ip);

		// If I am the dst, I reply RREP.
		if(my_rreq->rreq_dst_addr == *mip){
			if(!(my_rreq->U)) {
				// receive RREQ from local repairing node
				if(my_rreq->rreq_dst_seqno == (rtable.myOwnSeqno() + 1))
					rtable.incMyOwnSeqno();
			}

			sendRREP(my_rreq->rreq_dst_addr, my_rreq->rreq_src_addr, src_ip, 0,
rtable.myOwnSeqno(), my_route_timeout_);

			freePacket(pkt);

			return (1);
		}
		// If intermediate node has "fresh enough" route, it may
		// reply RREP.
```

```
                else if((rtable.rt_lookup(my_rreq->rreq_dst_addr) == RTF_VALID)
                    && (rtable.rt_get(my_rreq->rreq_dst_addr)->rt_valid_dst_seqno)
                    && (rtable.rt_get(my_rreq->rreq_dst_addr)->rt_seqno >= my_rreq-
>rreq_dst_seqno)
                    && (!(my_rreq->D)))
                {
                        Rt_entry *rt_d = rtable.rt_get(my_rreq->rreq_dst_addr);
                        Rt_entry *rt_s = rtable.rt_get(my_rreq->rreq_src_addr);


                        rtable.updatePrecursorList(my_rreq->rreq_dst_addr, src_ip);
                        rtable.updatePrecursorList(my_rreq->rreq_src_addr, rt_d->rt_nexthop);
                        sendRREP(my_rreq->rreq_dst_addr, my_rreq->rreq_src_addr, src_ip, rt_d-
>rt_hopcount, rt_d->rt_seqno, (rt_d->rt_time - GetCurrentTime()));

                        // gratuitous RREP to the dst
                        //if(my_rreq->G & 0x1)
                        if(my_rreq->G) {
                                sendRREP(my_rreq->rreq_src_addr, my_rreq->rreq_dst_addr, rt_d-
>rt_nexthop, rt_s->rt_hopcount, my_rreq->rreq_src_seqno, (rt_s->rt_time - GetCurrentTime()));

                        }

                        freePacket(pkt);

                        return (1);
                }
                else {
                        if(--my_pkt->ttl != 0)
                        forwardRREQ(my_rreq, my_pkt->ttl);

                        freePacket(pkt);

                        return (1);
                }
    }
    else if(bcmp(my_pkt->pro_type, "AODV_RREP", 10) == 0){

                struct RREP_msg * my_rrep;
                my_rrep = (struct RREP_msg *)((char *)my_pkt+sizeof(AODV_packet));

                // Hello message!!!
                if(dst_ip == inet_addr("1.0.255.255")) {
                        int lookup_result = rtable.rt_lookup(my_rrep->rrep_dst_addr);
                        if(lookup_result == RT_NOT_EXIST) {
                                Rt_entry *r = new Rt_entry;

                                r->rt_dst          = my_rrep->rrep_dst_addr;
                                r->rt_nexthop      = src_ip;
                                r->rt_valid_dst_seqno = true;
                                r->rt_seqno        = my_rrep->rrep_dst_seqno;
                                r->rt_hopcount     = my_rrep->rrep_hopcount + 1;
                                r->rt_flags        = RTF_VALID;
                                r->rt_time         = GetCurrentTime() + my_rrep->rrep_lifetime;

                                // insert new entry
```

```
                        rtable.insert(r);

                }
                else {
                        Rt_entry *r = rtable.rt_get(my_rrep->rrep_dst_addr);
                        if(!r)
                                printf("AODV error.\n");

                        r->rt_nexthop       = src_ip;
                        r->rt_valid_dst_seqno = true;
                        r->rt_seqno         = my_rrep->rrep_dst_seqno;
                        r->rt_hopcount      = my_rrep->rrep_hopcount + 1; // actually 1
                        r->rt_flags         = RTF_VALID;
                        r->rt_time          = GetCurrentTime() + my_rrep->rrep_lifetime;
                }

                // update the neighbor list
                nei_list.update(my_rrep->rrep_dst_addr, (GetCurrentTime() +
(ALLOWED_HELLO_LOSS * hello_interval_)));


                /* cclin: need to be examined to see if
                 * we should free the packet before this function
                 * returns.
                 */

                freePacket(pkt);
                return 0; // Hello msg won't be forwarded.
        } // HELLO pkt

        // create/update immediate reverse route to previous hop
        updateSimpleRRoute(src_ip);


        my_rrep->rrep_hopcount++;

        // add route to RREP-sender(ie. reverse route )
        int lookup_result = rtable.rt_lookup(my_rrep->rrep_dst_addr);
        if(lookup_result == RT_NOT_EXIST) {
                Rt_entry *r = new Rt_entry;

                r->rt_dst           = my_rrep->rrep_dst_addr;
                r->rt_nexthop       = src_ip;
                r->rt_valid_dst_seqno = true;
                r->rt_seqno         = my_rrep->rrep_dst_seqno;
                r->rt_hopcount      = my_rrep->rrep_hopcount;
                r->rt_flags         = RTF_VALID;
                r->rt_time          = GetCurrentTime() + my_rrep->rrep_lifetime;

                // insert new entry
                rtable.insert(r);
        }
        else {
                Rt_entry *r = rtable.rt_get(my_rrep->rrep_dst_addr);
                if(!r)
                        printf("AODV error.\n");
```

```
                bool need_update = false;

                if(r->rt_valid_dst_seqno == false) {
                        need_update = true;
                }
                if((my_rrep->rrep_dst_seqno > r->rt_seqno) &&
                  (r->rt_valid_dst_seqno == true)) {
                        need_update = true;
                }
                if(my_rrep->rrep_dst_seqno == r->rt_seqno) {
                        if(r->rt_flags != RTF_VALID) // ?
                                need_update = true;
                        if(my_rrep->rrep_hopcount < r->rt_hopcount)
                                need_update = true;
                }

                // update the exist routing
                if(need_update) {
                        r->rt_flags = RTF_VALID;
                        r->rt_valid_dst_seqno = true;
                        r->rt_nexthop = src_ip;
                        r->rt_hopcount = my_rrep->rrep_hopcount;
                        r->rt_time = GetCurrentTime() + my_rrep->rrep_lifetime;
                        r->rt_seqno = my_rrep->rrep_dst_seqno;
                }
        }
        /* else if(lookup_result == RTF_VALID){
                Rt_entry *r = rtable.rt_get(my_rrep->rrep_dst_addr);

                if(my_rrep->rrep_dst_seqno > r->rt_seqno) {
                }
        }
        else if(lookup_result == RTF_INVALID){
        }
        else if(lookup_result == RTF_REPAIRABLE){
        }
        else if(lookup_result == RTF_BEING_REPAIRED){
        } */

        // process the buffered pkts with the new routes
        processBuffered(my_rrep->rrep_dst_addr);
        processBuffered(src_ip);

        if(my_rrep->rrep_ori_addr == *mip){
                freePacket(pkt);
        return (1);
        } else {
                Rt_entry *rt_s = rtable.rt_get(my_rrep->rrep_ori_addr);

                if(rt_s) {
                        // only forwarding nodes need to update precursor list
                        rtable.updatePrecursorList(my_rrep->rrep_dst_addr, rt_s-
>rt_nexthop);
                        rtable.updatePrecursorList(src_ip, rt_s->rt_nexthop);
                        //rtable.updatePrecursorList(my_rrep->rrep_ori_addr, src_ip);
```

```
                                        // rt_s->rt_time = max(existingLT, now + active_route);
                                        if(rt_s->rt_time < (GetCurrentTime() + active_route_timeout_))
                                                rt_s->rt_time = GetCurrentTime() + active_route_timeout_;

                                        if(--my_pkt->ttl != 0)
                                        forwardRREP(my_rrep, my_pkt->ttl);
                                }

                        freePacket(pkt);
                        return (1);
                        }
                }

                else if(bcmp(my_pkt->pro_type, "AODV_STAT",10) == 0) {
                                        //M.Mateos

                        struct STAT_msg *my_stat;
                                        //M.Mateos

                        my_stat = (struct STAT_msg *)((char *)my_pkt+sizeof(AODV_packet));
                                        //M.Mateos
                        float   Stat = 0;
                                //M.Mateos
                        int     Position,Position1,i,Cycles = 0,time_check = 0, Time_Trace;
                                //M.Mateos
                        char
                Name[60],Name1[60],Name2[60],Stat_[9],Time_Trace1[2],Time_Trace2[3],Time_Trace3[4];
                        //M.Mateos
                        char    *Prueba;

                        //stat_type = 1 Neighbor Stats
                        if (my_stat->stat_type == 1){
                                        //M.Mateos
                                sprintf(Name1, "/home/nctuns/Escritorio/Resultados/Traffic-Stat-
%i",get_nid());                                 //M.Mateos
                                fdtF = fopen(Name1,"a+");
                                        //M.Mateos
                                fseek(fdtF,0,SEEK_END);
                                        //M.Mateos
                                if(ftell(fdtF) == 0) {
                                        //M.Mateos
                                        Time_Trace = 0;
                                        //M.Mateos
                                }
                                        //M.Mateos
                                else {
                                        //M.Mateos
                                        if ((double)(my_stat->stat_time*TICK/1000000000.0) < 10) {
                                //M.Mateos
                                                Position1 = ftell(fdtF) - (sizeof(double)+3);
                                //M.Mateos
                                                fseek(fdtF,Position1,SEEK_SET);
                                        //M.Mateos
                                                fscanf(fdtF,"%s", Time_Trace1);
                                        //M.Mateos
```

```
                                        Time_Trace = (int)strtod(Time_Trace1,&Prueba);
                                    //M.Mateos
                                    }
                                    //M.Mateos
                                    else if ((double)(my_stat->stat_time*TICK/1000000000.0) >= 10 &&
(double)(my_stat->stat_time*TICK/1000000000.0) < 100) {
                                        Position1 = ftell(fdtF) - (sizeof(double)+4);
                                //M.Mateos
                                        fseek(fdtF,Position1,SEEK_SET);
                                    //M.Mateos
                                        fscanf(fdtF,"%s", Time_Trace2);
                                    //M.Mateos
                                        Time_Trace = (int)strtod(Time_Trace2,&Prueba);
                                    //M.Mateos
                                    }
                                    //M.Mateos
                                    else if ((double)(my_stat->stat_time*TICK/1000000000.0) >= 100 &&
(double)(my_stat->stat_time*TICK/1000000000.0) < 1000) {
                                        Position1 = ftell(fdtF) - (sizeof(double)+5);
                                //M.Mateos
                                        fseek(fdtF,Position1,SEEK_SET);
                                    //M.Mateos
                                        fscanf(fdtF,"%s", Time_Trace3);
                                    //M.Mateos
                                        Time_Trace = (int)strtod(Time_Trace3,&Prueba);
                                    //M.Mateos
                                    }
                                    //M.Mateos
                                }
                                    //M.Mateos
                                fclose(fdtF);
                                    //M.Mateos
                                time_check = 0;
                                    //M.Mateos

                                if((int)(my_stat->stat_time*TICK/1000000000.0) - Time_Trace > 2) {
                                //M.Mateos
                                    Cycles = (int)(((int)(my_stat->stat_time*TICK/1000000000.0) -
Time_Trace)/2);                     //M.Mateos

                                    sprintf(Name1, "/home/nctuns/Escritorio/Resultados/Traffic-Stat-
%i",get_nid());                     //M.Mateos
                                    fdtF = fopen(Name1,"a+");
                                    //M.Mateos
                                    fseek(fdtF,0,SEEK_END);
                                    //M.Mateos
                                    Position1 = ftell(fdtF) - (sizeof(double)+1);
                                //M.Mateos
                                    fseek(fdtF,Position1,SEEK_SET);
                                    //M.Mateos
                                    fscanf(fdtF,"%s",Stat_);
                                //M.Mateos
                                    Stat = strtod(Stat_,&Prueba);
                                    //M.Mateos
                                    fclose(fdtF);
                                    //M.Mateos
```

```
                    for(i=1; (i<Cycles); i++) {
                //M.Mateos
                    Stat = (1-ALPHA)*Stat;
                    //M.Mateos
                    Time_Trace = Time_Trace + 2;
                    //M.Mateos

                    sprintf(Name1,
"/home/nctuns/Escritorio/Resultados/Traffic-Stat-%i",get_nid());          //M.Mateos
                    fdtF = fopen(Name1,"a+");
                    //M.Mateos
                    fseek(fdtF,0,SEEK_END);
                    //M.Mateos
                    if(Stat >= 10.0) {
                //M.Mateos
                        fprintf(fdtF,"%i\t%2.5f\n",Time_Trace, Stat);
                    //M.Mateos
                    }
                    //M.Mateos
                    else {
                    //M.Mateos
                        fprintf(fdtF,"%i\t%f\n",Time_Trace, Stat);
                //M.Mateos
                    }
                    //M.Mateos
                    fclose(fdtF);
                    //M.Mateos

                    fdtE = fopen("/home/nctuns/Escritorio/Resultados/Global-
Traffic-Stat","a+");          //M.Mateos
                    fseek(fdtE,0,SEEK_END);
                    //M.Mateos
                    fprintf(fdtE,"Tiempo: %lf - Semaforo: %i - Estadistica: %f -
Vecinos: 0 \n", (double)Time_Trace, get_nid(), Stat);
                    fclose(fdtE);
                    //M.Mateos
                    }
                    //M.Mateos
                }
                    //M.Mateos
                sprintf(Name, "/home/nctuns/Escritorio/Resultados/Traffic-Stat-
%i",get_nid());          //M.Mateos
                fdtD = fopen(Name,"a+");
                //M.Mateos
                fseek(fdtD,0,SEEK_END);
                    //M.Mateos
                if(ftell(fdtD) == 0) {
                    //M.Mateos
                    Stat = (my_stat->stat_neighbors + 1)*ALPHA;
                    //M.Mateos
                    fprintf(fdtD,"%i\t%f\n",(int)(my_stat-
>stat_time*TICK/1000000000.0), Stat);          //M.Mateos
                }
                    //M.Mateos
```

```
                        else {
                                //M.Mateos
                                Position = ftell(fdtD) - (sizeof(double)+1);
                        //M.Mateos
                                fseek(fdtD,Position,SEEK_SET);
                                //M.Mateos
                                fscanf(fdtD,"%s", Stat_);
                        //M.Mateos
                                Stat = strtod(Stat_,&Prueba);
                                //M.Mateos
                                Stat = ((my_stat->stat_neighbors + 1)*ALPHA+(1-ALPHA)*Stat);
                                //M.Mateos
                                if (Stat >= 10.0) {
                        //M.Mateos
                                        fseek(fdtD,0,SEEK_END);
                                //M.Mateos
                                        fprintf(fdtD,"%i\t%2.5f\n",(int)(my_stat-
>stat_time*TICK/1000000000.0), Stat);            //M.Mateos
                                }
                                //M.Mateos
                                else {
                                //M.Mateos
                                        fseek(fdtD,0,SEEK_END);
                                //M.Mateos
                                        fprintf(fdtD,"%i\t%f\n",(int)(my_stat-
>stat_time*TICK/1000000000.0), Stat);            //M.Mateos
                                }
                                //M.Mateos
                        }
                                //M.Mateos
                        fclose(fdtD);
                                //M.Mateos
                        fdtE = fopen("/home/nctuns/Escritorio/Resultados/Global-Traffic-Stat","a+");
                        //M.Mateos
                        fprintf(fdtE,"Tiempo: %lf - Semaforo: %i - Estadistica: %f - Vecinos: %i \n",
(double)(my_stat->stat_time*TICK)/1000000000.0, get_nid(), Stat, my_stat->stat_neighbors);

        //M.Mateos
                        fclose(fdtE);
                                //M.Mateos
                }
                                //M.Mateos

                //stat_type = 2 Delay Stats
                else if (my_stat->stat_type == 2) {
                        //M.Mateos
                        sprintf(Name2,"/home/nctuns/Escritorio/Resultados/Delay-Stat-
%i",get_nid());                         //M.Mateos
                        fdtK = fopen (Name2,"a+");
                                //M.Mateos
                        fprintf(fdtK,"Tiempo: %lf\tCoche: %i\tRetardo: %lf\n",(double)(my_stat-
>stat_time*TICK)/1000000000.0,my_stat->stat_my_id,my_stat->stat_delay);
                        fclose(fdtK);
                                //M.Mateos
```

```
                        fdtA = fopen ("/home/nctuns/Escritorio/Resultados/Global-Delay-Stat","a+");
                        //M.Mateos
                        fprintf(fdtA,"Tiempo: %lf - Semaforo: %i - Coche: %i - Retardo:
%lf\n",(double)(my_stat->stat_time*TICK)/1000000000.0,get_nid(),my_stat->stat_my_id,my_stat-
>stat_delay);
        //M.Mateos
                        fclose(fdtA);
                                //M.Mateos

                }
                                //M.Mateos

                //Could add more statistics adding more types to the code...

        }

        else if(bcmp(my_pkt->pro_type, "AODV_RERR", 10) == 0){

                struct RERR_msg  *my_rerr;

                my_rerr = (struct RERR_msg *)((char *)my_pkt+sizeof(AODV_packet));
                Unreach_list *unr_list = new Unreach_list();

                struct unreach_tuple *p_tuple = &(my_rerr->unreach_e);
                u_char num_tuple          = my_rerr->destCount;

                // generate my own unreach_list according to the received RERR
                while(num_tuple > 0) {
                        Rt_entry *p_rt = rtable.rt_get(p_tuple->unreach_dip);
                        if((p_rt != NULL) && (p_rt->rt_nexthop == src_ip)) {
                                unr_list->insert(p_tuple->unreach_dip, p_tuple->unreach_dseq);
                                // invalidate those route entries which use
                                // the node(sending RERR) as next hop.
                                p_rt->rt_flags = RTF_INVALID;
                        }

                        num_tuple--;
                        p_tuple++;
                }

                //sendRERR to every precursors of each entry of the unreach_list
                struct unreach_entry *p_ue = unr_list->getHead();
                while(p_ue) {
                        Rt_entry *p_rt = rtable.rt_get(p_ue->unreach_dip);
                        if(p_rt != NULL) {
                                Nei_entry *p_nei = p_rt->rt_preclist->getHead();
                                while(p_nei) {
                                        if(acc_rerr <= RERR_RATELIMIT) {
                                                sendRERR(p_nei->nei_addr, unr_list);
                                                acc_rerr++;
                                        }
                                        p_nei = p_nei->next;
                                }
                        } else {
                        }
```

```
                        p_ue = p_ue->next;
                }

                delete unr_list;

                freePacket(pkt);
                return (1);
    }

    // not RREQ,RREP,RERR packet
    else{
                printf("[%u]: receive AODV_unknown pkt (type:%s) at tick=%llu\n",
                  get_nid(), my_pkt->pro_type, GetCurrentTime());
                //assert(0);
                return 1;
    }

    freePacket(pkt);
    return (1);

}


int
AODV::send(ePacket_ *pkt) {

    Packet          *p;
    u_long          dst_ip, src_ip;

    assert(pkt&&(p=(Packet *)pkt->DataInfo_));

    GET_PKT(p, pkt);

    IP_DST(dst_ip, p->pkt_sget());
    IP_SRC(src_ip, p->pkt_sget());

    if (is_ipv4_broadcast(get_nid(), dst_ip)){
            // It's a broadcast pkt. Just send it.
            sendToQueue(pkt);
            return 1;
    }

    int lookup_result = rtable.rt_lookup(dst_ip);
    if(lookup_result != RTF_VALID) {

        if (!ctrl_table.ifExist(dst_ip)) {

                        ctrl_table.insert(dst_ip, GetCurrentTime()+net_traversal_time_);
                        if(ctrl_table.attachPkt(dst_ip, pkt) < 0) {
                                freePacket(pkt);
                                return (1);
                        }

                        if(acc_rreq <= RREQ_RATELIMIT) {
                        sendRREQ(dst_ip, NET_DIAMETER);
                                acc_rreq++;
```

```
                        }
                        return (1);

        }else{
                    if(ctrl_table.attachPkt(dst_ip, pkt) < 0)
                        freePacket(pkt);
                    return (1);
                }
    }
    else {
        Rt_entry *rt0 = rtable.rt_get(dst_ip);

                // each time the route is used for transmission,
                // update its lifetime
                rt0->rt_time = GetCurrentTime() + active_route_timeout_;

                p->rt_setgw(rt0->rt_nexthop);

        sendToQueue(pkt);
                return (1);
    }
}



int AODV::sendToQueue(ePacket_ *pkt) {

    int             (NslObject::*upcall)(MBinder *);
            /* Do flow control for myself with s_queue */

    if ( sendtarget_->get_curqlen() > 0 ) {
        /* Insert to rd_queue */
            if (qcur_ < qmax_ ){
                if (qcur_ == 0){
                        /* The first ePacket I want to send */
                            rd_head = pkt;
                            rd_tail = pkt;
                            pkt->next_ep = 0;
             }else{
                        rd_tail->next_ep = pkt;
                            rd_tail = pkt;
                    }/* if (qcur_ == 0) */
                    qcur_++;
            }else{
                    /* queue is full, drop the ePacket */
                        freePacket(pkt);
            }
        return (1);
    }else{
        /* call put() method */
            upcall = (int (NslObject::*)(MBinder *))&AODV::push;

            BASE_OBJTYPE(type);
            type = POINTER_TO_MEMBER(AODV,LinkLayerCall);
            Packet      *p;
        GET_PKT(p, pkt);
```

```
                p->pkt_setHandler(this,type);
                sendtarget_->set_upcall(this, upcall);
                return (put(pkt, sendtarget_));
        }
}


int AODV::push(void) {

        ePacket_     *pkt;
        int          (NslObject::*upcall)(MBinder *);
        if (qcur_ > 0){
                pkt = rd_head;
                rd_head = rd_head->next_ep;
                qcur_--;

                upcall = (int (NslObject::*)(MBinder *))&AODV::push;
                sendtarget_->set_upcall(this, upcall);
                assert(put(pkt, sendtarget_) > 0);
        }
        return (1);
}


int AODV::processBuffered(u_long new_rt_addr) {
        Packet     *p;
        Ctrl_entry *p_c = ctrl_table.getEntry(new_rt_addr);
        struct buf_list *p_buf, *p_pre;
        Rt_entry *new_rt;
        int queuecnt = 0;


        if(p_c) {
                p_buf = p_c->buffer_list;
                if(rtable.rt_lookup(new_rt_addr) == RTF_VALID) {
                        new_rt = rtable.rt_get(new_rt_addr);

                        while(p_buf) { // FIFO
                                GET_PKT(p, p_buf->queued_pkt);
                                p->rt_setgw(new_rt->rt_nexthop);
                                sendToQueue(p_buf->queued_pkt);
                                queuecnt++;

                                p_pre = p_buf;
                                p_buf = p_buf->next;

                                //free(p_pre);
                        }
                } else {
                         return 0;
                }
        }


        ctrl_table.remove(new_rt_addr);
```

```
        LocalRepair_entry *p_local_rep = local_repair_table.getEntry(new_rt_addr);
        p_buf = NULL; p_pre = NULL;
        new_rt = NULL;
        queuecnt = 0;

        if(p_local_rep) {
                p_buf = p_local_rep->buffer_list;
                if(rtable.rt_lookup(new_rt_addr) == RTF_VALID) {
                        new_rt = rtable.rt_get(new_rt_addr);

                        while(p_buf) { // FIFO
                                GET_PKT(p, p_buf->queued_pkt);
                                p->rt_setgw(new_rt->rt_nexthop);
                                sendToQueue(p_buf->queued_pkt);
                                queuecnt++;

                                p_pre = p_buf;
                                p_buf = p_buf->next;

                                //free(p_pre);
                        }
                } else {
                         return 0;
                }
        }


        local_repair_table.remove(new_rt_addr);

        return 0;
}



/*
 *   Regulary Send Broadcast HELLO message.
 */
int AODV::sendHello() {

        // Helllo msg will be sent if there is one or
        // more than one active-routes(hopcount >= 2) in the route table.
        // ps: Since receiving Hello pkt may generate route with 1 hopcount,
        //     to avoid infinite sendHello, we take those route with at least
        //     2 hopcount as active.
        /*if(rtable.rt_activeCnt() == 0) {
                int tmprandom = Random();
                int sign=0;
                if (tmprandom%2 == 0)
                sign = 1;
                else
                sign = -1;

                tmprandom = tmprandom%10000 * sign;

                BASE_OBJTYPE(type);
                type = POINTER_TO_MEMBER(AODV, sendHello);
```

```
                SendHello_timer.setCallOutObj(this, type);
                SendHello_timer.start((sendhello_timer_ + tmprandom), 0);

                return 0;
        }*/

        struct AODV_packet *mypkt;
        struct RREP_msg    *my_hello;


        ePacket_   *pkt;
        Packet *p = new Packet;
        pkt = createEvent();

        mypkt = (struct AODV_packet *)p->pkt_malloc(sizeof(struct AODV_packet)
                        + sizeof(struct RREP_msg));
        strcpy(mypkt->pro_type,"AODV_RREP");
        mypkt->dst_ip = inet_addr("1.0.255.255");
        mypkt->src_ip = *mip;
        mypkt->ttl = 1;

        my_hello = (struct RREP_msg *)((char *)mypkt+sizeof(AODV_packet));
        my_hello->rrep_hopcount = 0;
        my_hello->rrep_dst_addr = *mip;
        my_hello->rrep_dst_seqno = rtable.myOwnSeqno();
        //my_hello->rrep_ori_addr = ;
        my_hello->rrep_lifetime = ALLOWED_HELLO_LOSS * hello_interval_;


        p->pkt_addinfo("isAODV", "yes", 4);

        p->rt_setgw(inet_addr("1.0.255.255"));
        p->pkt_setflow(PF_SEND);

        ATTACH_PKT(p, pkt);

        int tmprandom = Random();
        int sign=0;
        if (tmprandom%2 == 0)
        sign = 1;
        else
        sign = -1;

        tmprandom = tmprandom%10000 * sign;

        BASE_OBJTYPE(type);
        type = POINTER_TO_MEMBER(AODV, sendHello);
        SendHello_timer.setCallOutObj(this, type);
        // send hello every HELLO_INTERVAL
        SendHello_timer.start((hello_interval_ + tmprandom), 0);

        return (put(pkt, sendtarget_));
}

/* insert the reverse route to the previous hop from which AODV msg received */
int AODV::updateSimpleRRoute(u_long prevhop_ip) {
```

```
        if((rtable.rt_lookup(prevhop_ip)) == RT_NOT_EXIST) {

        Rt_entry  *r = new Rt_entry;

                r->rt_dst = prevhop_ip;
                r->rt_flags = RTF_VALID;
                r->rt_nexthop = prevhop_ip;
                r->rt_valid_dst_seqno = false; /* seqno field is invalid */
        r->rt_hopcount = 1;
                r->rt_time = GetCurrentTime() + my_route_timeout_; /* ? */

                rtable.insert(r);
        }else{
                Rt_entry *tmp;

                tmp = rtable.rt_get(prevhop_ip);

                tmp->rt_flags = RTF_VALID;
                tmp->rt_nexthop = prevhop_ip;
                //tmp->rt_valid_dst_seqno = false;
        tmp->rt_hopcount = 1;
                tmp->rt_time = GetCurrentTime() + my_route_timeout_;
        }

        return 0;
}

int AODV::updateRT(u_long dst, u_long nexthop, u_int32_t seqno, u_int16_t hopcount, u_int64_t
lifetime) {

     Rt_entry  *r = new Rt_entry;

          r->rt_dst = dst;
          r->rt_flags = RTF_VALID;
          r->rt_nexthop = nexthop;
          r->rt_seqno = seqno;
     r->rt_hopcount = hopcount;
          r->rt_time = lifetime;

          rtable.insert(r);

          return 1;
}

int AODV::sendRREQ(u_long dst, const u_char ttl) {

          struct AODV_packet *mypkt;
          struct RREQ_msg * my_rreq;

          ePacket_   *pkt;
          Packet *p = new Packet;
          pkt = createEvent();


          mypkt = (struct AODV_packet *)p->pkt_malloc(sizeof(struct AODV_packet)
```

```
                    + sizeof(struct RREQ_msg));
strcpy(mypkt->pro_type, "AODV_RREQ");
mypkt->dst_ip = inet_addr("1.0.255.255"); // broadcast
mypkt->src_ip = *mip;
mypkt->ttl    = ttl;

my_rreq = (struct RREQ_msg *)((char *)mypkt+sizeof(struct AODV_packet));
my_rreq->rreq_dst_addr = dst;
my_rreq->rreq_src_addr = *mip;

/* my_rreq->rreq_dst_seqno = 1; *//* ? */

my_rreq->G = 0; // ?
my_rreq->D = 0; // ?
if(rtable.rt_lookup(dst) == RT_NOT_EXIST){
        my_rreq->U = 1;
}else{

        Rt_entry *tmp = rtable.rt_get(dst);

        if(tmp->rt_valid_dst_seqno) {
                my_rreq->U = 0;
                my_rreq->rreq_dst_seqno = tmp->rt_seqno;
        } else {
                my_rreq->U = 1;
        }
}

my_rreq->rreq_src_seqno = rtable.myOwnSeqno();

rreq_id++; // increment by 1 for each attempt
my_rreq->rreq_id = rreq_id;

my_rreq->rreq_hopcount = 0; /* 0? */

p->pkt_addinfo("isAODV", "yes", 4);
p->rt_setgw(inet_addr("1.0.255.255"));
p->pkt_setflow(PF_SEND);

// insert or update the bcache
bool found_in_bcache = false;
BroadcastID *b;
SLIST_FOREACH(b, &bcache, nexB){
        if(b->addr == my_rreq->rreq_src_addr) {

                found_in_bcache = true;
                // update to the greatest rreq_id
                b->bid = (b->bid > my_rreq->rreq_id) ? b->bid : my_rreq->rreq_id;
                b->lifetime = GetCurrentTime() + path_discovery_time_;
                break;
        }
}

if(!found_in_bcache) {
// buffer its own ip and rreq_id for path_discovery_time
BroadcastID *b = new struct BroadcastID;
```

```
                b->addr = *mip;
                b->bid = rreq_id; // already +1
                b->lifetime = GetCurrentTime() + path_discovery_time_;
                SLIST_INSERT_HEAD(&bcache, b, nexB);
        }

        ATTACH_PKT(p, pkt);
        return (put(pkt,sendtarget_));

}


int AODV::forwardRREQ(RREQ_msg * my_rreq, const u_char cur_ttl) {

        struct AODV_packet *mypkt;
        struct RREQ_msg *my_rreq_f;

        ePacket_   *pkt;
        Packet *p = new Packet;
        pkt = createEvent();


        mypkt = (struct AODV_packet *)p->pkt_malloc(sizeof(struct AODV_packet)
                        + sizeof(struct RREQ_msg));
        strcpy(mypkt->pro_type,"AODV_RREQ");
        mypkt->dst_ip = inet_addr("1.0.255.255");
        mypkt->src_ip = *mip;
        mypkt->ttl = cur_ttl;

        my_rreq_f = (struct RREQ_msg *)((char *)mypkt+sizeof(AODV_packet));
        my_rreq_f->G          =my_rreq->G;
        my_rreq_f->D          =my_rreq->D;
        my_rreq_f->U          =my_rreq->U;
        my_rreq_f->rreq_dst_addr = my_rreq->rreq_dst_addr;
        my_rreq_f->rreq_src_addr = my_rreq->rreq_src_addr;
        my_rreq_f->rreq_dst_seqno = my_rreq->rreq_dst_seqno;
        my_rreq_f->rreq_src_seqno = my_rreq->rreq_src_seqno;
        my_rreq_f->rreq_id = my_rreq->rreq_id;
        my_rreq_f->rreq_hopcount = my_rreq->rreq_hopcount; //already increased

        p->pkt_addinfo("isAODV", "yes", 4);

        p->rt_setgw(inet_addr("1.0.255.255"));
        p->pkt_setflow(PF_SEND);

        ATTACH_PKT(p, pkt);

        return (put(pkt,sendtarget_));
}


int AODV::sendRREP(u_long dst,     u_long src,     u_long toward,
                u_int8_t hopcount, u_int32_t seqno, u_int64_t lifetime){
        struct AODV_packet *mypkt;
```

```
        struct RREP_msg * my_rrep;


        ePacket_   *pkt;
        Packet *p = new Packet;
        pkt = createEvent();

        mypkt = (struct AODV_packet *)p->pkt_malloc(sizeof(struct AODV_packet)
                        + sizeof(struct RREP_msg));
        strcpy(mypkt->pro_type,"AODV_RREP");
        mypkt->dst_ip = toward;
        mypkt->src_ip = *mip;
        mypkt->ttl = NET_DIAMETER;

        my_rrep = (struct RREP_msg *)((char *)mypkt+sizeof(AODV_packet));
        my_rrep->rrep_hopcount = hopcount;
        my_rrep->rrep_dst_addr = dst;
        my_rrep->rrep_dst_seqno = seqno;
        my_rrep->rrep_ori_addr = src;
        my_rrep->rrep_lifetime = lifetime;

        p->pkt_addinfo("isAODV", "yes", 4);

        p->rt_setgw(toward);
        p->pkt_setflow(PF_SEND);

        ATTACH_PKT(p, pkt);

        return (put(pkt, sendtarget_));
}


int AODV::forwardRREP(struct RREP_msg *my_rrep, const u_char cur_ttl) {

        struct AODV_packet *mypkt;
        struct RREP_msg * my_rrep_f;

        Rt_entry *rt_ori;
        if(rtable.rt_lookup(my_rrep->rrep_ori_addr) == RTF_VALID)
                rt_ori = rtable.rt_get(my_rrep->rrep_ori_addr);
        else {
                return (-1);
        }

        ePacket_   *pkt;
        Packet *p = new Packet;
        pkt = createEvent();

        mypkt = (struct AODV_packet *)p->pkt_malloc(sizeof(struct AODV_packet)
                        + sizeof(struct RREP_msg));
        strcpy(mypkt->pro_type,"AODV_RREP");
        mypkt->dst_ip = rt_ori->rt_nexthop;
        mypkt->src_ip = *mip;
        mypkt->ttl = cur_ttl;

        my_rrep_f = (struct RREP_msg *)((char *)mypkt+sizeof(AODV_packet));
```

```
        my_rrep_f->rrep_dst_addr  = my_rrep->rrep_dst_addr;
        my_rrep_f->rrep_dst_seqno = my_rrep->rrep_dst_seqno;
        my_rrep_f->rrep_ori_addr  = my_rrep->rrep_ori_addr;
        my_rrep_f->rrep_hopcount  = my_rrep->rrep_hopcount; // already +1
        my_rrep_f->rrep_lifetime  = my_rrep->rrep_lifetime;

        p->pkt_addinfo("isAODV", "yes", 4);

        p->rt_setgw(rt_ori->rt_nexthop);
        p->pkt_setflow(PF_SEND);

        ATTACH_PKT(p, pkt);

        return (put(pkt,sendtarget_));
}

int AODV::sendSTAT(int type, int id, int neighbors, u_int64_t time, u_long dst, double delay){
                //M.Mateos
        struct AODV_packet *mypkt;
                        //M.Mateos
        struct STAT_msg * my_stat;
                        //M.Mateos

        ePacket_   *pkt;
                //M.Mateos
        Packet *p = new Packet;
                        //M.Mateos
        pkt = createEvent();
                        //M.Mateos

        mypkt = (struct AODV_packet *)p->pkt_malloc(sizeof(struct AODV_packet)
                        //M.Mateos
                        + sizeof(struct STAT_msg));
        strcpy(mypkt->pro_type,"AODV_STAT");
                        //M.Mateos
        mypkt->dst_ip = dst;
                        //M.Mateos
        mypkt->src_ip = *mip;
                        //M.Mateos
        mypkt->ttl = NET_DIAMETER;
                        //M.Mateos

        my_stat = (struct STAT_msg *)((char *)mypkt+sizeof(AODV_packet));
                //M.Mateos
        my_stat->stat_type = type;
                        //M.Mateos
        my_stat->stat_my_id = id;
                //M.Mateos
        my_stat->stat_neighbors = neighbors;
                        //M.Mateos
        my_stat->stat_time = time;
                        //M.Mateos
        my_stat->stat_delay = delay;
                        //M.Mateos
```

```
        p->pkt_addinfo("isAODV", "yes", 4);
                        //M.Mateos

        p->rt_setgw(dst);
                //M.Mateos
        p->pkt_setflow(PF_SEND);
                        //M.Mateos

        ATTACH_PKT(p, pkt);
                        //M.Mateos

        return (put(pkt, sendtarget_));
                        //M.Mateos
}


// unicast RERR, fill the field according to the unreach_list
int AODV::sendRERR(u_long precursor_ip, Unreach_list *p_ulist) {
        struct AODV_packet *mypkt;
        struct RERR_msg    *my_rerr;


        ePacket_   *pkt;
        Packet *p = new Packet;
        pkt = createEvent();

        // First unreach_tuple is in struct RERR_msg
        mypkt = (struct AODV_packet *)p->pkt_malloc(
                  sizeof(struct AODV_packet)
                + sizeof(struct RERR_msg)
                + (p_ulist->unreach_count() - 1)* sizeof(struct unreach_tuple));
        strcpy(mypkt->pro_type,"AODV_RERR");
        mypkt->dst_ip = precursor_ip;
        mypkt->src_ip = *mip;

        my_rerr = (struct RERR_msg *)((char *)mypkt+sizeof(AODV_packet));
        my_rerr->destCount = p_ulist->unreach_count();

        struct unreach_tuple *p_utuple = &(my_rerr->unreach_e);

        // fill in tuples of unreach ip and seq
        struct unreach_entry *p_ue = p_ulist->getHead();
        while(p_ue) {
                p_utuple->unreach_dip  = p_ue->unreach_dip;
                p_utuple->unreach_dseq = p_ue->unreach_dseq;

                p_utuple++;
                p_ue = p_ue->next;
        }

        p->pkt_addinfo("isAODV", "yes", 4);

        p->rt_setgw(precursor_ip); // ?
        p->pkt_setflow(PF_SEND);

        ATTACH_PKT(p, pkt);
```

```
	return(put(pkt, sendtarget_));
}

// broadcast RERR, fill the field according to the unreach_list
int AODV::bcastRERR(Unreach_list *p_ulist) {
	struct AODV_packet *mypkt;
	struct RERR_msg    *my_rerr;


	ePacket_   *pkt;
	Packet *p = new Packet;
	pkt = createEvent();
	// First unreach_tuple is in struct RERR_msg
	mypkt = (struct AODV_packet *)p->pkt_malloc(
		  sizeof(struct AODV_packet)
		+ sizeof(struct RERR_msg)
		+ (p_ulist->unreach_count() - 1)* sizeof(struct unreach_tuple));
	strcpy(mypkt->pro_type,"AODV_RERR");
	mypkt->dst_ip = inet_addr("1.0.255.255");
	mypkt->src_ip = *mip;

	my_rerr = (struct RERR_msg *)((char *)mypkt+sizeof(AODV_packet));
	my_rerr->destCount = p_ulist->unreach_count();

	struct unreach_tuple *p_utuple = &(my_rerr->unreach_e);

	// fill in tuples of unreach ip and seq
	struct unreach_entry *p_ue = p_ulist->getHead();
	while(p_ue) {
		p_utuple->unreach_dip  = p_ue->unreach_dip;
		p_utuple->unreach_dseq = p_ue->unreach_dseq;

		p_utuple++;
		p_ue = p_ue->next;
	}

	p->pkt_addinfo("isAODV", "yes", 4);

	p->rt_setgw(inet_addr("1.0.255.255"));
	p->pkt_setflow(PF_SEND);

	ATTACH_PKT(p, pkt);

	return(put(pkt, sendtarget_));
}


// mac layer reports transmission error and return the transmitted pkt.
int AODV::LinkLayerCall(ePacket_ *pkt){

	Packet *p;
	u_long dst;
	struct Link_fail_entry *p_link_f;
	bool   found=false;
#ifdef LINK_LAYER_RETRY
```

```
        bool retry=false;
#else
        bool drop=false;
#endif

        GET_PKT(p, pkt);
        IP_DST(dst, p->pkt_sget());


        if (!local_repair_table.ifExist(dst)) {
                SLIST_FOREACH(p_link_f, &link_fail_list, next) {
                        if(p_link_f->dst_ip == dst) {
                                found = true;

                                p_link_f->acc_cnt++;
                                p_link_f->lifetime = LINK_FAIL_LIFETIME;

                                if(p_link_f->acc_cnt < LINK_FAIL_THRESHOLD) {
#ifdef LINK_LAYER_RETRY
                                        retry = true;
#else
                                        drop = true;
#endif
                                } else {
#ifdef LINK_LAYER_RETRY
                                        retry = false;
#else
                                        drop = false;
#endif
                                }

                                break;
                        }
                }

                if(!found) {
                        p_link_f = new struct Link_fail_entry;
                        p_link_f->dst_ip = dst;
                        p_link_f->acc_cnt = 1;
                        p_link_f->lifetime = LINK_FAIL_LIFETIME;
                        //p_link_f->next = NULL;

                        SLIST_INSERT_HEAD(&link_fail_list, p_link_f, next);

#ifdef LINK_LAYER_RETRY
                        retry = true;
#else
                        drop  = true;
#endif
                }

                // If the LinkLayerCall does not reach LINK_FAIL_THRESHOLD,
                // we try to send back to link-layer for outgoing again.
#ifdef LINK_LAYER_RETRY
                if(retry) {
                        return(put(pkt, sendtarget_));
```

```
                }
#else
                if(drop) {
                        freePacket(pkt);
                        return 0;
                }
#endif

        local_repair_table.insert(dst, GetCurrentTime()+net_traversal_time_, p->rt_gateway());
                if(local_repair_table.attachPkt(dst, pkt) < 0){ // queue full
                        freePacket(pkt);
                        return 0;
                }

        } else {
                if(local_repair_table.attachPkt(dst, pkt) < 0){ // queue full
                        freePacket(pkt);
                }

                return 0; // local-repair already took place before
        }

        // local repair
        Rt_entry *p_rt_broken = rtable.rt_get(dst);
        if(p_rt_broken) {
                // local repair only once
                if(p_rt_broken->rt_flags != RTF_BEING_REPAIRED) {
                        p_rt_broken->rt_seqno++;   // dst_seqno++ to avoid route loop

                        p_rt_broken->rt_flags = RTF_BEING_REPAIRED;
                        sendRREQ(dst, p_rt_broken->rt_hopcount);
                }
        }

        return 0;
}
```

ANNEX 3. AODV.H MODIFIED CODE

```
/*
 * Copyright (c) from 2000 to 2009
 *
 * Network and System Laboratory
 * Department of Computer Science
 * College of Computer Science
 * National Chiao Tung University, Taiwan
 * All Rights Reserved.
 *
 * This source code file is part of the NCTUns 6.0 network simulator.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation is hereby granted (excluding for commercial or
 * for-profit use), provided that both the copyright notice and this
 * permission notice appear in all copies of the software, derivative
 * works, or modified versions, and any portions thereof, and that
 * both notices appear in supporting documentation, and that credit
 * is given to National Chiao Tung University, Taiwan in all publications
 * reporting on direct or indirect use of this code or its derivatives.
 *
 * National Chiao Tung University, Taiwan makes no representations
 * about the suitability of this software for any purpose. It is provided
 * "AS IS" without express or implied warranty.
 *
 * A Web site containing the latest NCTUns 6.0 network simulator software
 * and its documentations is set up at http://NSL.csie.nctu.edu.tw/nctuns.html.
 *
 * Project Chief-Technology-Officer
 *
 * Prof. Shie-Yuan Wang <shieyuan@csie.nctu.edu.tw>
 * National Chiao Tung University, Taiwan
 *
 * 09/01/2009
 */

#ifndef __NCTUNS_AODV_h__
#define __NCTUNS_AODV_h__

#include <event.h>
#include <object.h>
#include <mylist.h>
#include <timer.h>
#include <packet.h>
//#include <route/aodv/mstate.h>

#define INFINITY_HOPCNT  0xff
#define INFINITY_LIFETIME  0x7fffffff
#define AODV_MAXQUEUELEN 30

#define TTL_THRESHOLD 7

#define LINK_FAIL_LIFETIME          500   // ms
```

```
#define LINK_FAIL_THRESHOLD         4     // times

// millisec
#define SENDHELLO_TIMER             50
#define ROUTE_CHECK                 300
#define PENDING_RREQ_CHECK          50
#define RECENT_RREQ_LIST_CHECK      300
#define ACCUMULATED_RREQ_RERR_TIMER   1000
#define NEI_LIST_CHECK              300
#define NEI_LIST_STAT_CHECK         2000
                        //M.Mateos
#define DELAY_STAT_CHECK            50
                        //M.Mateos
#define LINK_FAIL_LIST_CHECK        1000
#define ALPHA                       0.125
                        //M.Mateos


namespace AODVd{

class AODV;

/*
 *   Neighbor entry and Neighbor List
 */
class Nei_entry {
public:
    u_long                      nei_addr;    // ip address
    u_int64_t           nei_time;    // expire time

    Nei_entry           *next;
public:
    Nei_entry(u_long addr, u_int64_t lifetime);
    ~Nei_entry();
};

class Neighbor {
private:
        Nei_entry         *nei_head;

public:
        Neighbor();
        ~Neighbor();

        //int       isExist(u_long addr);
        // If not exist, new entry is created, otherwise update the lifetime.
        int       update(u_long addr, u_int64_t lifetime);
        int       remove(u_long addr);
        int           stat(u_int64_t currenttime, int my_id, int *neighbors_, u_long *light_addr_, int
*light_id_);        //M.Mateos
        Nei_entry*   getEntry(u_long addr);
        Nei_entry*   getHead();
};

struct Link_fail_entry {
        SLIST_ENTRY(Link_fail_entry) next;
```

```
        u_long      dst_ip;
        u_char      acc_cnt;
        u_int64_t   lifetime;
};

/*
 *  Broadcast ID and BroadcastID List
 */
struct BroadcastID {
    SLIST_ENTRY(BroadcastID) nexB;
    u_long                   addr;
    u_int32_t                bid;
    u_int8_t                 hopcnt;
    u_int64_t                lifetime;
};



class Rt_entry {
public:
    u_long                   rt_dst;    // destination ip
    u_long                   rt_nexthop; // nexthop
    bool                     rt_valid_dst_seqno;// dst seqno is valid of not
    u_int32_t                rt_seqno;   // destination sequence number
    u_int8_t                 rt_hopcount;// hopcount
#define RTF_INVALID        1
#define RTF_VALID          2
#define RTF_REPAIRABLE     3
#define RTF_BEING_REPAIRED   4
    u_int8_t                 rt_flags;    // valid/invalid/repairable/repaired
    Neighbor                 *rt_preclist; // precurosr list
    u_int64_t                rt_time;    // expire time

    Rt_entry                 *next;

    Rt_entry();
    ~Rt_entry();
};



class AODV_RtTable {

/* friend class Neighbor; */

private:
    Rt_entry                 *rt_head;
    Rt_entry                 *rt_tail;

public:
    AODV_RtTable();
    ~AODV_RtTable();

    int                      insert(Rt_entry *r);
    //int                      update();
    int                      remove(u_long addr);
```

```
    int             removeEntry(Rt_entry *target, Rt_entry *pre);
    int             updatePrecursorList(u_long dst_addr,u_long nei_addr);
#define RT_NOT_EXIST 0
    int             rt_lookup(u_long ip);
    Rt_entry*       rt_get(u_long ip);
    Rt_entry*       rt_getHead();
    u_int32_t       myOwnSeqno();
    int             incMyOwnSeqno();

    // It will return count for the active-routes(except the first entry for
    // its own) in the route table. If the count is equal or greater than
    // one, Hello msg should be broadcasted periodically.
    int             rt_activeCnt();

};
/*
 *  AODV's queue and its entry
 */
struct buf_list {
    ePacket_        *queued_pkt;
    struct buf_list*    next;
};

/*
 *   every destination has a pkt queue list
 */
class Ctrl_entry {
        public:
    u_long          dst_ip;
    u_int           buf_pkt_cnt;
    struct buf_list*   buffer_list;

    u_int64_t       rreq_lifetime;
    u_int           rreq_retries;

    Ctrl_entry      *next;

        public:
        Ctrl_entry();
        ~Ctrl_entry();
};

class CtrlTable {

   private:
     Ctrl_entry*        ctrl_head;

   public:
     CtrlTable();
        ~CtrlTable();

        int             insert(u_long dst, u_int64_t rreq_lt);
        int             remove(u_long dst);
        int             attachPkt(u_long dst, ePacket_*);
    bool                    ifExist(u_long);
        Ctrl_entry*         getHead();
```

```
        Ctrl_entry*         getEntry(u_long);

};

class LocalRepair_entry {
public:
    u_long          dst_ip;
    u_int           buf_pkt_cnt;
    struct buf_list*   buffer_list;

    u_int64_t       local_repair_lifetime;
    u_int           local_repair_retries;
    // brokenlink_node is used for local-repair(linkLayer upcall),
    // It stores the pkt_gateway(i.e. the disconnected next-hop).
#define NO_USE     0
    u_long          brokenlink_node;

    LocalRepair_entry       *next;

public:
        LocalRepair_entry();
        ~LocalRepair_entry();
};

class LocalRepairTable {

private:
    LocalRepair_entry*        local_repair_head;

        /* Ctrl_entry*         getHead(void); */

public:
    LocalRepairTable();
        ~LocalRepairTable();

        int             insert(u_long dst, u_int64_t rreq_lt, u_long disc_node);
        int             remove(u_long dst);
        int             attachPkt(u_long dst, ePacket_*);
    bool                        ifExist(u_long);
        LocalRepair_entry*    getHead();
        LocalRepair_entry*    getEntry(u_long);

};


struct RREQ_msg {
        char   type;

        u_short  J:1;
        u_short  R:1;
        u_short  G:1;
        u_short  D:1;
        u_short  U:1;
        u_short  Reserved:11;

        u_int8_t        rreq_hopcount;
```

```
        u_int32_t       rreq_id;
        u_long          rreq_dst_addr;
        u_int32_t       rreq_dst_seqno;
        u_long          rreq_src_addr;
        u_int32_t       rreq_src_seqno;

};


struct RREP_msg {

        char   type;

        u_short  R:1;
        u_short  A:1;
        u_short  Reserved:9;
        u_short  prefix_size:5;

        u_int8_t        rrep_hopcount;
        u_long          rrep_dst_addr;
        u_int32_t       rrep_dst_seqno;
        u_long          rrep_ori_addr;
        u_int64_t       rrep_lifetime;

};

struct STAT_msg {
                        //M.Mateos

        char   type;
                        //M.Mateos

        int                stat_type;
                        //M.Mateos
        int                stat_my_id;
                        //M.Mateos
        int                stat_neighbors;
                //M.Mateos
        u_int64_t          stat_time;
                        //M.Mateos
        u_long             stat_dst;
                        //M.Mateos
        double             stat_delay;
                        //M.Mateos

};


struct unreach_tuple {
        u_long      unreach_dip;
        u_int32_t   unreach_dseq;
};

struct RERR_msg {
        char  type;
```

```
        u_short N:1;
        u_short Reserved:15;

        u_char      destCount;

        struct unreach_tuple  unreach_e;
};

struct unreach_entry {
        u_long      unreach_dip;
        u_int32_t    unreach_dseq;

        struct unreach_entry *next;
};

class Unreach_list {
        u_char              unr_count;

        struct unreach_entry   *unr_head;

        public:
        Unreach_list();
        ~Unreach_list();

        int                 insert(u_long, u_int32_t);
        struct unreach_entry*   getHead();
        u_char              unreach_count();
};

/*
 *  AODV's pkt format: {
 *    protocol type => { AODV_RREQ or AODV_RREP or AODV_RERR or AODV_HELLO};
 *    destination ip addr;
 *    source ip addr;
 *    who send this pkt to me; => last ipaddr;
 *    who send this pkt to me(it's seqno): => last sequence number;
 *    pointer point to RREQ or RREP or RERR or HELLO;
 *  }
 */


struct AODV_packet {
   char           pro_type[11];
   u_long                  src_ip;
   u_long                  dst_ip;
   u_char         ttl;
   char *                  point_msg;
};

struct Result {
   int               neighbors;
   u_long                  light_addr;
};


class AODV : public NslObject {
```

```
                    // default
    int  HELLO_INTERVAL;       //1000
    int  ALLOWED_HELLO_LOSS ;   //2
    int  ACTIVE_ROUTE_TIMEOUT;  //3000
    int  DELETE_PERIOD;        //3000
         int  NET_DIAMETER;         //15
         int  NODE_TRAVERSAL_TIME;   //40 ms
         int  RREQ_RETRIES;         //5
         int  RREQ_RATELIMIT;       //10/per sec
         int  RERR_RATELIMIT;       //10/per sec

    int  MY_ROUTE_TIMEOUT;
         int  NET_TRAVERSAL_TIME;
         int  PATH_DISCOVERY_TIME;

private:
         timerObj                   SendHello_timer;
         timerObj                   DelHello_timer;
         timerObj           RT_timer;
         timerObj           SendRREQ_timer;
         timerObj           RecentRREQ_timer;
         timerObj           AccRREQ_RERR_timer;
         timerObj           Nei_List_timer;
         timerObj                   Nei_List_Stat_timer;
                           //M.Mateos
         timerObj                   Delay_Stat_timer;
                           //M.Mateos
         timerObj           PrintLoc_timer;

         // interval tmp variable
         u_int64_t          hello_interval_;
         u_int64_t          delete_period_;

         u_int64_t          active_route_timeout_;
         u_int64_t          my_route_timeout_;
         u_int64_t          node_traversal_time_;
         u_int64_t          net_traversal_time_;
         u_int64_t          path_discovery_time_;

         u_int64_t          route_check_timer_;
         u_int64_t          rreq_check_timer_;
         u_int64_t          recent_rreq_list_timer_;
         u_int64_t          accumulated_rreq_rerr_timer_;
         u_int64_t          nei_list_check_timer_;
         u_int64_t                  nei_list_stat_check_timer_;
                           //M.Mateos
         u_int64_t                  delay_stat_check_timer_;
                  //M.Mateos
         u_int64_t          sendhello_timer_;
         u_int64_t          link_fail_list_check_timer_;
         u_int64_t          printLoc_timer_;

         u_long                     *mip;          // my IP address

         int                rreq_id;        // my rreq ID
```

```
        AODV_RtTable        rtable;         // my Routing Table
        CtrlTable           ctrl_table;     // my AODVqueue table
        LocalRepairTable    local_repair_table;
        SLIST_HEAD( ,Link_fail_entry)  link_fail_list;
        SLIST_HEAD( ,BroadcastID)  bcache;      // Broadcase ID cache
        Neighbor            nei_list;       // neighbor list

        // accumulated count for RREQ/RERR, will be clean to 0 every second
        int             acc_rreq;
        int             acc_rerr;

        int             qmax_;          // AODVqueue's max
        int             qcur_;          // AODVqueue current count
        ePacket_        *rd_head;       // AODVqueue's head
        ePacket_        *rd_tail;


        public:

        AODV(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~AODV();

        int             init();
        int             recv(ePacket_ *pkt);
        int             send(ePacket_ *pkt);
        int                     sendHello();
        int                     miew();

        int                     HelloTimer();
        int             RTTimer();
        int             RREQ_retry();
        int             CheckRecentRREQ();
        int             ClearAccRREQ_RERR();
        int             CheckNeiList();
        int                     CheckNeiList_Stat();
                //M.Mateos
        int                     CheckDelay_Stat();
                //M.Mateos
        int             CheckLinkFailList();


        int                     UpdateHello(struct HELLO_msg *);
        int                     routing(u_long dst, Packet *p);
        int             updateSimpleRRoute(u_long prevhop_ip);
        int                 updateRT(u_long dst, u_long nexthop, u_int32_t seqno, u_int16_t
hopcount, u_int64_t lifetime);

        int                     sendRREQ(u_long dst, u_char ttl);
        int                     forwardRREQ(struct RREQ_msg *my_rreq, u_char cur_ttl);
        int                     sendRREP(u_long dst, u_long src, u_long toward, u_int8_t hopcount,
u_int32_t seqno, u_int64_t lifetime);
        int                     forwardRREP(struct RREP_msg *my_rrep, u_char cur_ttl);
        int                     sendSTAT(int type, int id, int neighbors, u_int64_t time, u_long dst,
double delay);    //M.Mateos
        int                     sendRERR(u_long delip, Unreach_list *);
        int                     bcastRERR(Unreach_list *);
```

```
        int                     push(void);
        int                     sendToQueue(ePacket_*);
        int                     processBuffered(u_long);
        int                     LinkLayerCall(ePacket_ *);
        int                     PrintIP(u_long);
};

}; //namespace AODVd

#endif  /* __NCTUNS_AODV_h__ */
```

ANNEX 4. AODVRT.CC MODIFIED CODE

```
/*
 * Copyright (c) from 2000 to 2009
 *
 * Network and System Laboratory
 * Department of Computer Science
 * College of Computer Science
 * National Chiao Tung University, Taiwan
 * All Rights Reserved.
 *
 * This source code file is part of the NCTUns 6.0 network simulator.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation is hereby granted (excluding for commercial or
 * for-profit use), provided that both the copyright notice and this
 * permission notice appear in all copies of the software, derivative
 * works, or modified versions, and any portions thereof, and that
 * both notices appear in supporting documentation, and that credit
 * is given to National Chiao Tung University, Taiwan in all publications
 * reporting on direct or indirect use of this code or its derivatives.
 *
 * National Chiao Tung University, Taiwan makes no representations
 * about the suitability of this software for any purpose. It is provided
 * "AS IS" without express or implied warranty.
 *
 * A Web site containing the latest NCTUns 6.0 network simulator software
 * and its documentations is set up at http://NSL.csie.nctu.edu.tw/nctuns.html.
 *
 * Project Chief-Technology-Officer
 *
 * Prof. Shie-Yuan Wang <shieyuan@csie.nctu.edu.tw>
 * National Chiao Tung University, Taiwan
 *
 * 09/01/2009
 */

#include <stdlib.h>
#include <module/route/aodv/AODV.h>
#include <nctuns_api.h>
#include <string.h>

//#include <dmalloc.h>
//
/* #define DEBUG_HSIYUN */

FILE       *fdtC;

using namespace AODVd;

/// Nei_entry ////////////////////////////////
Nei_entry::Nei_entry(u_long addr, u_int64_t lifetime) {
        nei_addr = addr;
        nei_time = lifetime;
```

```
	next = NULL;
}

Nei_entry::~Nei_entry() {
}

/// Neighbor ////////////////////////////////
Neighbor::Neighbor() {
	nei_head = NULL;
}

Neighbor::~Neighbor() {
	Nei_entry *n = nei_head;
	Nei_entry *tmp;

	while(n){
		tmp = n;
		n = n->next;
		delete tmp;
	}
}

// create new entry or update the existing lifeitme
int Neighbor::update(u_long addr, u_int64_t lifetime) {

	Nei_entry *p_nei = getEntry(addr);
	if(!p_nei) {  // not exist
		Nei_entry *n_e = new Nei_entry(addr, lifetime);
		if(n_e == NULL) return -1;

		// insert into the head
		n_e->next = nei_head;
		nei_head  = n_e;
		return 0;
	}else {
		p_nei->nei_time = lifetime;

		return 1;
	}

}

int Neighbor::remove(u_long addr) {
	Nei_entry *p_nei = nei_head;
	Nei_entry *p_pre = NULL;

	while(p_nei) {
		if(p_nei->nei_addr == addr) {
			if(p_pre == NULL) {
				nei_head = p_nei->next;
			}else {
				p_pre->next = p_nei->next;
			}

			delete p_nei;
```

```
                        return 0;
                }
                p_pre = p_nei;
                p_nei = p_nei->next;
        }

        // NOT found
        return -1;
}

int Neighbor::stat(u_int64_t currenttime, int my_id, int *neighbors_, u_long *light_addr_, int *light_id_)
{
        Nei_entry *p_nei = nei_head;
        int neighbors = 0, id = 0, light_id = 0, reference_light_id = 0;
        u_long light_addr = 0, reference_light_addr = 0;
        u_int64_t light_time = 0, reference_light_time = 0;
        char *addr[6];

        if(my_id != 26 && my_id != 29 && my_id != 32 && my_id != 35 && my_id != 38 && my_id != 41
&& my_id != 44 && my_id != 47 && my_id != 50 &&                          my_id != 53 &&
my_id != 56 && my_id != 59 && my_id != 62) {
                while(p_nei) {
                        if(p_nei->nei_time >= currenttime) {
                                id = ipv4addr_to_nodeid(p_nei->nei_addr);
                                if(id != 26 && id != 29 && id != 32 && id != 35 && id != 38 && id != 41
&& id != 44 && id != 47 && id != 50 && id != 53 && id != 56 &&
id != 59 && id != 62)
                                        neighbors++;
                                else {
                                        ipv4addr_to_str_subnet(p_nei->nei_addr, *addr);
                                        /*fdtC = fopen ("/home/nctuns/Escritorio/Addr", "a+");
                                        fprintf(fdtC, "Tiempo: %lf - Coche: %i - Subred del Vecino: %s
- ID Vecino: %i \n", (double)(currenttime*TICK)/1000000000.0, my_id, *addr, id);
                                        fclose(fdtC);*/
                                        if(strcmp(*addr, "1.0.3") == 0) {
                                                light_addr = p_nei->nei_addr;
                                                light_time = p_nei->nei_time;
                                                light_id = id;
                                                if(light_time > reference_light_time) {
                                                        reference_light_time = light_time;
                                                        reference_light_addr = light_addr;
                                                        reference_light_id = light_id;
                                                }
                                        }
                                }
                        }
                        p_nei = p_nei->next;
                }
        }
        *neighbors_ = neighbors;
        *light_addr_ = reference_light_addr;
        *light_id_ = reference_light_id;

        return 0;
}
```

```
Nei_entry* Neighbor::getHead() {
	return nei_head;
}

Nei_entry* Neighbor::getEntry(u_long addr) {
	Nei_entry *p_nei = nei_head;

	while(p_nei) {
		if(p_nei->nei_addr == addr)
			return p_nei;

		p_nei = p_nei->next;
	}

	// NOT found
	return NULL;
}

/// Rt_entry //////////////////////////////////
Rt_entry::Rt_entry() {
	rt_preclist = new Neighbor;
	next = NULL;
}

Rt_entry::~Rt_entry() {
	delete rt_preclist;
}



/// AODV_RtTable //////////////////////////////
AODV_RtTable::AODV_RtTable() {
	rt_head = NULL;
	rt_tail = NULL;

}

AODV_RtTable::~AODV_RtTable() {
	Rt_entry *r = rt_head;
	Rt_entry *tmp;

	while(r) {
		tmp = r;
		r = r->next;
		delete tmp;
	}
}

int AODV_RtTable::insert(Rt_entry *r){
	if((rt_head == NULL) && (rt_tail == NULL)) {
		rt_head = r;
		rt_tail = r;
	}else {
		// insert into tail
		rt_tail->next = r;
		rt_tail = r;
```

```
	}

	return 0;
}

int AODV_RtTable::remove(u_long addr){
	Rt_entry *r   = rt_head;
	Rt_entry *pre = NULL;

	while(r) {
		if(r->rt_dst == addr) {
			if(pre == NULL) {
				//the first entry(for its own) should not be deleted
				printf("AODV error.\n");
				//rt_head = r->next;
				//delete r;
			}else {
				pre->next = r->next;
				if(r == rt_tail)
					rt_tail = pre;

				delete r;
			}
			return 0; // found and deleted
		}
		pre = r;
		r = r->next;
	}

	return -1; // entry not found
}

int AODV_RtTable::removeEntry(Rt_entry *target, Rt_entry *pre){

	pre->next = target->next;
	if(target == rt_tail)
		rt_tail = pre;
	delete target;
	//target->~Rt_entry();

	return 0;
}

int AODV_RtTable::updatePrecursorList(u_long dst_addr, u_long nei_addr){
	Rt_entry *tmp = rt_head;

	while(tmp) {
		if(tmp->rt_dst == dst_addr) {
			// lifetime is not necessary for precursor
			tmp->rt_preclist->update(nei_addr, 0);
			return 0;
		}

		tmp = tmp->next;
	}
	// corresponding route entry not found
```

```
	return -1;
}


int AODV_RtTable::rt_lookup(u_long ip) {
	Rt_entry *tmp = rt_head;

	while(tmp) {
		if(tmp->rt_dst == ip) {
			return tmp->rt_flags;
		}

		tmp = tmp->next;
	}

	return RT_NOT_EXIST;
}
Rt_entry* AODV_RtTable::rt_get(u_long ip) {
	Rt_entry *tmp = rt_head;

	while(tmp) {
		if(tmp->rt_dst == ip) {
			return tmp;
		}

		tmp = tmp->next;
	}

	return NULL;
}

Rt_entry* AODV_RtTable::rt_getHead() {

	return rt_head;
}

// the first routing entry is for its own(including its own seqno)
u_int32_t AODV_RtTable::myOwnSeqno() {
	return rt_head->rt_seqno;
}

int AODV_RtTable::incMyOwnSeqno() {
	rt_head->rt_seqno++;

	return 0;
}

int AODV_RtTable::rt_activeCnt() {
	int ret = 0;

	// The first entry for its own is excluded.
	Rt_entry *tmp = rt_head->next;

	while(tmp) {
		if((tmp->rt_flags == RTF_VALID) && (tmp->rt_hopcount >=2))
```

```
                        ret++;

                tmp = tmp->next;
        }

        return ret;
}

/// Ctrl_entry //////////////////////////
Ctrl_entry::Ctrl_entry() {
        buf_pkt_cnt = 0;
        buffer_list = NULL;
        rreq_retries = 0;

        next = NULL;
}

Ctrl_entry::~Ctrl_entry() {
        struct buf_list *p_buf = buffer_list;
        struct buf_list *p_tmp;

        while(p_buf) {
                p_tmp = p_buf;
                p_buf = p_buf->next;
                free(p_tmp);
        }
}

/// CtrlTable //////////////////////////
CtrlTable::CtrlTable() {
        ctrl_head = NULL;
}

CtrlTable::~CtrlTable() {
        Ctrl_entry *c = ctrl_head;
        Ctrl_entry *tmp;

        while(c) {
                tmp = c;
                c = c->next;
                delete tmp;
        }
}

int CtrlTable::insert(u_long dst, u_int64_t rreq_lt) {
        Ctrl_entry *c = new Ctrl_entry;


        c->dst_ip = dst;
        c->rreq_lifetime = rreq_lt;

        c->next = ctrl_head; // insert into head
        ctrl_head = c;

        return 0;
}
```

```
int CtrlTable::attachPkt(u_long in_dst_ip, ePacket_ *pkt) {


   if (!ifExist(in_dst_ip))
      return (-1);   /* not found control entry for this IP */

   Ctrl_entry *p_c = ctrl_head;
   // determine the correspoding entry
   while(p_c) {
            if(p_c->dst_ip == in_dst_ip)
                     break;
            p_c = p_c->next;
   }

   if(p_c->buf_pkt_cnt >= AODV_MAXQUEUELEN)
            return (-1);

   struct buf_list  *p_buf = p_c->buffer_list;

   // traverse to the tail
   while(p_buf) {
            if(p_buf->next == NULL)
                     break;
            p_buf = p_buf->next;
   }

   if(!p_buf) { // list is empty
         p_buf = (struct buf_list*)malloc(sizeof(struct buf_list));
         p_buf->queued_pkt = pkt;
         p_buf->next = NULL;

         p_c->buffer_list = p_buf;
   }else {
         // attach to the tail (for FIFO processing queued pkts)
         p_buf->next = (struct buf_list*)malloc(sizeof(struct buf_list));
         p_buf->next->queued_pkt = pkt;
         p_buf->next->next = NULL;
   }

   p_c->buf_pkt_cnt++;
   // p_c->rreq_enable = true;

   return (1);
}

int CtrlTable::remove(u_long dst) {
         Ctrl_entry *c   = ctrl_head;
         Ctrl_entry *pre = NULL;

         while(c) {
                  if(c->dst_ip == dst) {
                           if(pre == NULL) { //delete first entry
                                    ctrl_head = c->next;
                                    delete c;
                           }else {
```

```
                                pre->next = c->next;
                                delete c;
                        }
                        return 0; // found and deleted
                }
                pre = c;
                c = c->next;
        }

        return -1; // entry not found
}

bool CtrlTable::ifExist(u_long addr) {
        Ctrl_entry *tmp = ctrl_head;

        while(tmp) {
                if(tmp->dst_ip == addr)
                    return true;
                tmp = tmp->next;
    }

    /* NOT found */
    return false;
}

Ctrl_entry* CtrlTable::getHead() {
        return ctrl_head;
}

Ctrl_entry* CtrlTable::getEntry(u_long addr) {
        Ctrl_entry *ret = ctrl_head;

        while(ret) {
                if(ret->dst_ip == addr)
                        return ret;
                ret = ret->next;
        }

        // NOT found
        return NULL;
}


/// LocalRepair_entry ////////////////////////////
LocalRepair_entry::LocalRepair_entry() {
        buf_pkt_cnt = 0;
        buffer_list = NULL;
        next = NULL;
}

LocalRepair_entry::~LocalRepair_entry() {
        struct buf_list *p_buf = buffer_list;
        struct buf_list *p_tmp;

        while(p_buf) {
                p_tmp = p_buf;
```

```
                p_buf = p_buf->next;
                free(p_tmp);
        }
}

/// LocalRepairTable ////////////////////////////
LocalRepairTable::LocalRepairTable() {
        local_repair_head = NULL;
}

LocalRepairTable::~LocalRepairTable() {
        LocalRepair_entry *c = local_repair_head;
        LocalRepair_entry *tmp;

        while(c) {
                tmp = c;
                c = c->next;
                delete tmp;
        }
}

int LocalRepairTable::insert(u_long dst, u_int64_t rreq_lt, u_long disc_node) {
        LocalRepair_entry *c = new LocalRepair_entry;


        c->dst_ip = dst;
        c->local_repair_lifetime = rreq_lt;
        if(disc_node != NO_USE)
                c->brokenlink_node = disc_node;

        c->next = local_repair_head; // insert into head
        local_repair_head = c;

        return 0;
}

int LocalRepairTable::attachPkt(u_long in_dst_ip, ePacket_ *pkt) {


   if (!ifExist(in_dst_ip))
      return (-1);   /* not found control entry for this IP */

   LocalRepair_entry *p_c = local_repair_head;
   // determine the correspoding entry
   while(p_c) {
             if(p_c->dst_ip == in_dst_ip)
                       break;
             p_c = p_c->next;
   }

   if(p_c->buf_pkt_cnt >= AODV_MAXQUEUELEN)
             return (-1);

   struct buf_list  *p_buf = p_c->buffer_list;

   // traverse to the tail
```

```
  while(p_buf) {
        if(p_buf->next == NULL)
            break;
        p_buf = p_buf->next;
  }

  if(!p_buf) { // list is empty
        p_buf = (struct buf_list*)malloc(sizeof(struct buf_list));
        p_buf->queued_pkt = pkt;
        p_buf->next = NULL;

        p_c->buffer_list = p_buf;
  }else {
        // attach to the tail (for FIFO processing queued pkts)
        p_buf->next = (struct buf_list*)malloc(sizeof(struct buf_list));
        p_buf->next->queued_pkt = pkt;
        p_buf->next->next = NULL;
  }

  p_c->buf_pkt_cnt++;
  // p_c->rreq_enable = true;

  return (1);
}

int LocalRepairTable::remove(u_long dst) {
        LocalRepair_entry *c   = local_repair_head;
        LocalRepair_entry *pre = NULL;

        while(c) {
            if(c->dst_ip == dst) {
                if(pre == NULL) { //delete first entry
                    local_repair_head = c->next;
                    delete c;
                }else {
                    pre->next = c->next;
                    delete c;
                }
                return 0; // found and deleted
            }
            pre = c;
            c = c->next;
        }

        return -1; // entry not found
}


bool LocalRepairTable::ifExist(u_long addr) {
        LocalRepair_entry *tmp = local_repair_head;

        while(tmp) {
            if(tmp->dst_ip == addr)
                return true;
            tmp = tmp->next;
    }
```

```
    /* NOT found */
    return false;
}

LocalRepair_entry* LocalRepairTable::getHead() {
        return local_repair_head;
}

LocalRepair_entry* LocalRepairTable::getEntry(u_long addr) {
        LocalRepair_entry *ret = local_repair_head;

        while(ret) {
                if(ret->dst_ip == addr)
                        return ret;
                ret = ret->next;
        }

        // NOT found
        return NULL;
}
Unreach_list::Unreach_list() {
        unr_count = 0;
        unr_head = NULL;
}

Unreach_list::~Unreach_list() {
        struct unreach_entry *p_ue = unr_head;
        struct unreach_entry *p_pre = NULL;

        while(p_ue) {
                p_pre = p_ue;
                p_ue = p_ue->next;
                free(p_pre);
        }
}

int Unreach_list::insert(u_long addr, u_int32_t seqno) {
        struct unreach_entry *p_unr = (struct unreach_entry *)malloc(sizeof(struct unreach_entry));

        p_unr->unreach_dip  = addr;
        p_unr->unreach_dseq = seqno;

        // insert into the head
        p_unr->next = unr_head;
        unr_head    = p_unr;

        unr_count++;

        return 0;
}

struct unreach_entry*
Unreach_list::getHead() {
        return unr_head;
}
```

```
u_char Unreach_list::unreach_count() {
        return unr_count;
}
```

ANNEX 5. CARAGENT.CC MODIFIED CODE

```
/*
 * Copyright (c) from 2000 to 2009
 *
 * Network and System Laboratory
 * Department of Computer Science
 * College of Computer Science
 * National Chiao Tung University, Taiwan
 * All Rights Reserved.
 *
 * This source code file is part of the NCTUns 6.0 network simulator.
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation is hereby granted (excluding for commercial or
 * for-profit use), provided that both the copyright notice and this
 * permission notice appear in all copies of the software, derivative
 * works, or modified versions, and any portions thereof, and that
 * both notices appear in supporting documentation, and that credit
 * is given to National Chiao Tung University, Taiwan in all publications
 * reporting on direct or indirect use of this code or its derivatives.
 *
 * National Chiao Tung University, Taiwan makes no representations
 * about the suitability of this software for any purpose. It is provided
 * "AS IS" without express or implied warranty.
 *
 * A Web site containing the latest NCTUns 6.0 network simulator software
 * and its documentations is set up at http://NSL.csie.nctu.edu.tw/nctuns.html.
 *
 * Project Chief-Technology-Officer
 *
 * Prof. Shie-Yuan Wang <shieyuan@csie.nctu.edu.tw>
 * National Chiao Tung University, Taiwan
 *
 * 09/01/2009
 */

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <fcntl.h>
#include <math.h>
#include <time.h>
#include "math_fun.h"
#include "tactic_api.h"
#include "sock_skel.h"
#include "road.h"
#include <deque>

#define ITS_NET 1
#define ITS_COLLISION_AVOIDANCE 1
```

```
#define ITS_SHOW_MSG 0

#define MOTION_LINEAR 0x30
#define MOTION_TAKING_TURNS_STEP_1 0x40
#define MOTION_TAKING_TURNS_STEP_2 0x41
#define MEGA 1000000 //micro secs
#define SLEEPING_PERIOD 100000 //micro secs
#define VISIBILITY_SCALE_IN_DEGREE 40.0 //degree
#define VISIBILITY_SCALE_IN_DISTANCE 100.0 //meters

FILE	*fdtA;

enum EventType
{
	APPROACHING_INTERSECT,
	IN_INTERSECT,
	LEAVING_INTERSECT
};

class msgSequence
{
	public:
		u_int32_t nid;
		int seqNum; // sequence number
};

//If car agent reach to the position (x, y), it change its direction into "the direction of class Event"
class Event
{
	public:
		double x, y;
		double direction; // the direction of class Event
		EventType type;
};

timeval now;
//-----Global variables for IPCs and APIs
int mynid, myTCPsockfd, myUDPsockfd, socketfd2;
int agentUDPportNum = 4000;
int oldCollisionCarID = 0;

//-----Global variables for car's performances.
double MaxAcceleration, MaxDeceleration, MaxVelocity;

//Global variables for car's condition
//1 m/s = 3.6 km/hr
double CurrentVelocity = 0, CurrentAcceleration, CurrentDirection;
double CurrentPOS_x, CurrentPOS_y, CurrentPOS_z = 0;
double ExpectedDirection;
double brokenCarDirection; // It will be a problem if there are many broken cars.
double roadWidth;
Event CurrentEvent;
int WarningDecelerate = 0;
int WakedTimes = -1;
int agentReceivedBrokenCarMsg = 0;
int brokenCarID = 0;
```

```
//About the traffic light
int SeenTheTrafficLightOrNot = 0, SigLight, SigGID = -1, SignalIndex = -1;
double SigPOS_x, SigPOS_y;
vector<msgSequence> msgSeq;

int CacheOrNot = 1;
deque<Event> EventQueue;

CacheRecord myCache;
CacheRecord otherCache;

void FillTheQueue(int NumOfTurns, double *turningPOS_x, double *turningPOS_y, double *directions,
double FirstPointAfterTheTurn_x, double FirstPointAfterTheTurn_y, double ExpectDirection)
{
	//This function fulls the queue when agent just has reached a new road block.
	//After agent calls the takeATurn function, it uses "pass by reference parameters" to full the
queue by this function.
	Event temp;
	for (int i = 0; i<NumOfTurns; i++){
		//put every turning points into the queue.
		temp.x = turningPOS_x[i];
		temp.y = turningPOS_y[i];
		temp.direction = directions[i];
		EventQueue.push_back(temp);

	}
	//Finally, put the end point of the turning into the queue.
	temp.x = FirstPointAfterTheTurn_x;
	temp.y = FirstPointAfterTheTurn_y;
	temp.direction = ExpectDirection;
	EventQueue.push_back(temp);

}

void reportMyStatusToAGroupOfNode()
{
	static int seqNum=0;
	double lastTime = (double) now.tv_sec * 1000000 + (double) now.tv_usec;
	int value;
	socklen_t len;

	agentClientReportStatus *msg;
	msg = new agentClientReportStatus;

	msg->x = CurrentPOS_x;
	msg->y = CurrentPOS_y;
	msg->type = AGENT_CLIENT_REPORT_STATUS;
	msg->nid = mynid;
	msg->moreMsgFollowing = 1;
	msg->acceleration = CurrentAcceleration;
	msg->speed = CurrentVelocity;
	msg->direction = CurrentDirection;
	msg->seqNum = seqNum;
	msg->timeStamp=lastTime;
	msg->TTL=3;
```

```
		seqNum++;
		sockaddr_in cli_addr;

		//Broadcast start

		value = 1;
		setsockopt(myUDPsockfd , SOL_SOCKET , SO_BROADCAST , &value , sizeof(value));
		len =  sizeof(cli_addr);
		memset(&cli_addr, 0, sizeof(cli_addr));
		cli_addr.sin_family = AF_INET;
		cli_addr.sin_port   = htons(agentUDPportNum);
		cli_addr.sin_addr.s_addr = inet_addr("1.0.1.255");
		int n = sendto(myUDPsockfd, msg, sizeof(struct agentClientReportStatus), 0, (struct sockaddr *)
&cli_addr, len);
		if (n < 0) {
			printf("Agent (%d) sendto failed\n", mynid);
		}
		else{
			//printf("Car Agent (%d) broadcast pkt, Size %d, seqNum %d, packet type==%d\n",
mynid, sizeof(struct agentClientReportStatus), msg->seqNum, msg->type);
		}
		//Broadcast end
		free(msg);
}

void receiveMsg()
{
		socklen_t len;
		int n = 1;
		sockaddr_in cli_addr;
		while(n > 0){
			typeChecker p;
			len = sizeof(cli_addr);
			n = recvfrom(myUDPsockfd, (char *) &p, sizeof(struct typeChecker), MSG_PEEK, (struct
sockaddr *) &cli_addr, &len);
			if(n == 0){
				//printf("Car Agent: %d UDP socket error\n", mynid);
				return;
			}
			if(p.type == RSUAGENT_REPORT_WARNING){
				RSUAgentReportWarning msg;
				n = recvfrom(myUDPsockfd, (char *)&msg, sizeof(struct
RSUAgentReportWarning), 0, (struct sockaddr *) &cli_addr, &len);
				printf("Agent %d : Receive Warning from RSU %d\n", mynid, msg.RSUnid);
				if(msg.AccelerationOrDeceleration == -1)
					WarningDecelerate = -10;
				continue;
			}
			else if(p.type == AGENT_CLIENT_REPORT_STATUS || p.type ==
AGENT_CLIENT_IS_A_BROKEN_CAR){
				agentClientReportStatus msg;
				n = recvfrom(myUDPsockfd, (char *)&msg, sizeof(struct
agentClientReportStatus), 0, (struct sockaddr *) &cli_addr, &len);
				if(n > 0){
					if(msg.seqNum > 0){
						int nidExists = 0;
```

```
                                        for (int i=0 ; i<(int)(msgSeq.size()); i++)
                                        {
                                                if(msg.nid == msgSeq[i].nid)
                                                {
                                                        nidExists=1;
                                                        if(msg.seqNum > msgSeq[i].seqNum)
                                                        {
                                                                //double twoDistance =
Distance_BetweenTwoNode(CurrentPOS_x, CurrentPOS_y, msg.x, msg.y);
                                                                //printf("Agent %d receive msg
from agent %d, distance %lf\n", mynid, msg.nid, twoDistance);
                                                                msgSeq[i].seqNum = msg.seqNum;
                                                                if(msg.type ==
AGENT_CLIENT_IS_A_BROKEN_CAR){

        agentReceivedBrokenCarMsg = 1;
                                                                        if(ITS_SHOW_MSG)

        printf("CarAgent(%d) received a broken car report from Agent (%d): %d\n", mynid, msg.nid,
msg.seqNum);


        if(ITS_COLLISION_AVOIDANCE){

        brokenCarDirection = msg.direction;
                                                                                brokenCarID =
msg.nid;
                                                                }
                                                                break;
                                                        }
                                                        if(ITS_SHOW_MSG)
                                                                printf("CarAgent(%d)
received a status report from Agent (%d): %d\n", mynid, msg.nid, msg.seqNum);

                                                }
                                                else{
                                                        //      printf("CarAgent(%d):
Message is already received from agent (%d)_%d\n", mynid, msg.nid, msg.seqNum);
                                                }
                                                break;

                                        }

                                }
                                if(nidExists == 0)
                                {
                                        msgSequence temp;
                                        temp.nid = msg.nid;
                                        temp.seqNum = msg.seqNum;
                                        msgSeq.push_back(temp);
                                        //double twoDistance =
Distance_BetweenTwoNode(CurrentPOS_x, CurrentPOS_y, msg.x, msg.y);
                                        //printf("Agent %d receive msg from agent %d,
distance %lf\n", mynid, msg.nid, twoDistance);
                                        if(msg.type == AGENT_CLIENT_IS_A_BROKEN_CAR){
                                                agentReceivedBrokenCarMsg = 1;
```

```
                                                        if(ITS_SHOW_MSG)
                                                                printf("CarAgent(%d) received a
broken car report from Agent(%d): %d\n", mynid, msg.nid, msg.seqNum);

                                                        if(ITS_COLLISION_AVOIDANCE)
                                                                brokenCarDirection =
msg.direction;
                                                }
                                                else {
                                                        if(ITS_SHOW_MSG)
                                                                printf("CarAgent(%d) received a
status report from Agent(%d): %d\n", mynid, msg.nid, msg.seqNum);
                                                }
                                        }
                                }
                        }
                }
        }
}

double RandomDirection(int NumOfDirections, double *directions)
{
        int RandInt = rand() % NumOfDirections;
        return directions[RandInt];
}

void init()
{
        int n;
        char portNumStr[32];
        double CorrectedPOS_x, CorrectedPOS_y;

        // get car node id
        mynid = getMyNodeID();
        srand(mynid);

        // create IPC connection with SE and set agent in group 1
        myTCPsockfd = createTCPSocketForCommunicationWithSimulationEngine(mynid, 1, -1, -1, -1, -
1, -1, -1, -1, 0, socketfd2, PROCESS_TYPE_AGENT, 1);

        // create UDP socket for listening broadcasted message
        sprintf(portNumStr, "%d", agentUDPportNum);
        myUDPsockfd = passiveUDP(portNumStr);
        printf("Agent(%d) created myUDPsockfd %d\n", mynid, myUDPsockfd);
        if (myUDPsockfd < 0) {
                printf("Agent(%d): Creating myUDPsockfd failed\n", mynid);
                exit(0);
        }
        n = fcntl(myUDPsockfd, F_SETFL, O_NONBLOCK);

        // Get driving behavior from car profile
        n = getProFileData(mynid, MaxVelocity, MaxAcceleration, MaxDeceleration, 1);
        if(n == 0){
                printf("Car %d open car profile fail. Use default value\n", mynid);
                MaxVelocity = 18;
                MaxAcceleration = 1;
```

```
		MaxDeceleration = 4;
	}

	// Construct the whole road map by using this function
	constructRoadMapofTheWholeField(1);

	n = getInitialNodePosition(mynid, CurrentPOS_x, CurrentPOS_y, CurrentPOS_z);
	CurrentPOS_z = 0;

	/* correct my position into the middle of the road */
	n = selfCorrectness(CurrentPOS_x, CurrentPOS_y, CorrectedPOS_x, CorrectedPOS_y,
CacheOrNot, &myCache);
	if(n == 1)
	{
		CurrentPOS_x = CorrectedPOS_x;
		CurrentPOS_y = CorrectedPOS_y;
	}
	n = setCurrentWaypoint(myTCPsockfd, mynid, CurrentPOS_x, CurrentPOS_y, CurrentPOS_z, 1);
	/* default parameter */
	//CurrentVelocity = 0;
	CurrentAcceleration = 0; /* acceleration = 1 m/sec^2 */
	//setCurrentMovingSpeed(myTCPsockfd, mynid, CurrentVelocity, 1);

	getRoadWidth(roadWidth);
}

inline int ReachTheNextTriggerPointOrNot()
{
	if(((int) EventQueue.size()) == 0)
		return -1;// Error!!

	double tempAngle = fmod(atan2(CurrentEvent.y- CurrentPOS_y, CurrentEvent.x -
CurrentPOS_x)/ PI* 180+ 360, 360);
	tempAngle = fmod(360- tempAngle, 360);

	double angleDiff = fabs(tempAngle- CurrentDirection);
	if(angleDiff > 180)
		angleDiff = 360- angleDiff;

	if(angleDiff >= 90){
		// move over the event point, which means reach the event point
		if(EventQueue.empty() != true)
			EventQueue.pop_front();
		return 1; // reach Event point
	}
	else
		return 0; // unreach
}

inline double SelectMinimum(double a, double b)
{
	if(a <= b)
		return a;
	else
		return b;
}
```

```
inline double SelectMaximum(double a, double b)
{
	if(a >= b)
		return a;
	else
		return b;
}

/*
 * S: distance in meter
 * v0: current velocity in meter/s (> 0)
 * vt: target velocity in meter/s (> 0, because movng in the same direction with v0)
 * a: acceleration in meter/(s^2)
 * t: time in second
 *
 * E1: vt = v0 + a*t
 * E2: S = v0*t + (1/2)*a*(t^2)
 *
 * From E1, we get a = (vt - v0)/t
 * Put a into E2, we get S = 0.5*(v0 + vt)*t
 * Therefore, we get
 * E3: t = 2 * S / (v0 + vt)
 */
void DetermineVc(double &BufferTime, double &vt)
{
	double NearestPrecedingCarPOS_x, NearestPrecedingCarPOS_y, NearestPrecedingCarPOS_z,
NearestPrecedingCarDirection;
	double NearestPrecedingCarSpeed = 999999;
	double DesiredMaxSpeed, DistanceToNextEventPoint; // in meter/secs.
	double SAFETY_INTERVEHICLE_DISTANCE = 10;
	double BufferTime1, BufferTime2, BufferTime3 = 999999;
	int NearestPrecedingCarID, n = 1;

	//Desired max speed should be lower at the corner.
	DistanceToNextEventPoint = Distance_BetweenTwoNode(CurrentEvent.x, CurrentEvent.y,
CurrentPOS_x, CurrentPOS_y);

	/* Cosider what to do when close to the end of a road block */
	if((DistanceToNextEventPoint != 0) && (CurrentVelocity != 0))
	{
		BufferTime3 = roundf(DistanceToNextEventPoint / CurrentVelocity);
	}

	if(DistanceToNextEventPoint >= 30)
	{
		DesiredMaxSpeed = MaxVelocity; // meter/second
	}
	else
	{
		// If close to the next event point, don't drive too fast.
		DesiredMaxSpeed = rand() % 6 + 5; // 10 m/s => 36 km/hr at maxmimum
	}

	n =
getNearestNodePositionAlongTheSpecifiedDirectionAndRangeViewedOnGUIScreen(myTCPsockfd,
```

```
mynid, CurrentPOS_x, CurrentPOS_y, 0, CurrentDirection, VISIBILITY_SCALE_IN_DEGREE,
VISIBILITY_SCALE_IN_DISTANCE, 1, NearestPrecedingCarPOS_x, NearestPrecedingCarPOS_y,
NearestPrecedingCarPOS_z, NearestPrecedingCarDirection, NearestPrecedingCarID, 1);

        double vt1, vt2;

        vt1 = DesiredMaxSpeed;
        vt2 = DesiredMaxSpeed;

        // set default time, make car run as fast as possible
        /* Put a = MaxAcceleration into E1 and get BufferTime1 */
        BufferTime1 = fabs(vt1 - CurrentVelocity) / MaxAcceleration;
        BufferTime2 = BufferTime1;

        if(n == 0)
        {
                /* There's a car in front of me. */

                n = checkIfOnTheSameLane(CurrentPOS_x, CurrentPOS_y, NearestPrecedingCarPOS_x,
NearestPrecedingCarPOS_y, CacheOrNot, &myCache, &otherCache);
                if(n != 0)
                {
                        /* The preceding car is on the same lane with me. */

                        double distance = 999999;

                        // Get the distance between me and the preceding car.
                        distance = Distance_BetweenTwoNode(CurrentPOS_x, CurrentPOS_y,
NearestPrecedingCarPOS_x, NearestPrecedingCarPOS_y);
                        if((distance <= 0.3) && (oldCollisionCarID != NearestPrecedingCarID)){
                                printf("Collision of Car %d (%lf, %lf) and Car %d (%lf, %lf).\n",
                                                mynid, CurrentPOS_x, CurrentPOS_y,
                                                NearestPrecedingCarID,
NearestPrecedingCarPOS_x,
                                                NearestPrecedingCarPOS_y);
                                oldCollisionCarID = NearestPrecedingCarID;
                        }

                        // Get preceding car's velocity.
                        getCurrentMovingSpeed(myTCPsockfd, NearestPrecedingCarID,
NearestPrecedingCarSpeed, 1);

                        if(agentReceivedBrokenCarMsg == 1){
                                if((brokenCarDirection != ExpectedDirection) && (brokenCarID ==
NearestPrecedingCarID)){
                                        /* Broken car is on the same lane but I will switch to other
lane.
                                         * So, the preceding car shouldn't affect me.
                                         */
                                        NearestPrecedingCarSpeed = 999999;
                                        distance = 9999999;
                                        BufferTime1 = 999999;
                                }
                        }

                        /* Too close to the preceding car. */
```

```
                        if(distance <= 3)
                        {
                                BufferTime = 0;
                                vt = 0;
                                return;
                        }

                        /* Consider the effects of the preceding car's velocity and position. */
                        if(NearestPrecedingCarSpeed == 0)
                        {
                                vt1 = NearestPrecedingCarSpeed;

                                if(CurrentVelocity == 0)
                                {
                                        vt = 0;
                                        BufferTime = 0;
                                        return;
                                }
                                /* Set vt to 0 and put vt into E3.
                                 * We get t = 2 * S / v0
                                 */
                                BufferTime1 = 2 * (distance - 3) / CurrentVelocity;
                        }
                        else
                        {
                                if(distance < 2 * SAFETY_INTERVEHICLE_DISTANCE)
                                {
                                        /* Set the target velocity to NearestPrecedingCarSpeed. */
                                        double tmpV;

                                        tmpV = CurrentVelocity + NearestPrecedingCarSpeed;
                                        BufferTime1 = 2*(distance -
SAFETY_INTERVEHICLE_DISTANCE)/tmpV;
                                        vt1 = NearestPrecedingCarSpeed;
                                }
                                else
                                {
                                        /* Preceding car is far from me.
                                         * Using default vt1 and BufferTime1.
                                         */
                                }
                        }
                }
        }

        /* Cosider what to do when close to the traffic light */
        if((SeenTheTrafficLightOrNot == 1) && (SigLight == RED || SigLight == YELLOW))
        {
                double DistanceToTrafficLight = 9999999;

                DistanceToTrafficLight = Distance_BetweenTwoNode(SigPOS_x, SigPOS_y,
CurrentPOS_x, CurrentPOS_y);
                if(DistanceToTrafficLight <= 2*SAFETY_INTERVEHICLE_DISTANCE){
                        // Getting close to the traffic light, need to slow down a little
                        vt2 = SelectMinimum(0 - (DistanceToTrafficLight -
SAFETY_INTERVEHICLE_DISTANCE) / 10 , DesiredMaxSpeed);
```

```
		}
		if((DistanceToTrafficLight != 0) || (CurrentVelocity != 0))
		{
			if((BufferTime2 = DistanceToTrafficLight / CurrentVelocity) < 0)
			{
				BufferTime = 0;
				return;
			}

			/*
			 * From E1, put vt = 0 and we get 0 = v0 + at
			 * Ignore (1/2)*a*(t^2) part in E2 because t is very small.
			 * We get
			 * E1': 0 = v0 + a*t
			 * E2': S = v0*t
			 *
			 * From E1' and E2' we get v0 = sqrt(a * S)
			 */
			double scv1 = 0; // safe current velocity

			scv1 = sqrt(MaxDeceleration * DistanceToTrafficLight);

			/* safe check */
			if(CurrentVelocity >= scv1){
				/* Dangerous Velocity!! Need to slow down fast. */
				BufferTime = 0;
				return;
			}
		}
	}

	BufferTime = SelectMinimum(BufferTime1, BufferTime2);
	BufferTime = SelectMinimum(BufferTime, BufferTime3);

	vt = SelectMinimum(vt1, vt2);
}

double DetermineAcceleration()
{
	/*
	 * The following implementation is based on
	 * VATSIM: A Simulator for Vehicles and Traffic ,
	 * Jia Lei Keith Redmill Umit Ozguncr ,
	 * Department of Electrical Engineering, The Ohio State University,
	 * 2001 IEEE Intelligent Transportation Systems Conference Proceedings
	 *
	 *
	 * vt:		the target velocity
	 * bfTime: the buffer time to achieve vt.
	 */
	double vt, bfTime;

	DetermineVc(bfTime, vt);

	double acc;
```

```
	if(bfTime > 0)
	{
		/* Equation: vt = CurrentVelocity + acc * bfTime */
		acc = roundf((vt- CurrentVelocity) / bfTime * 10);
		acc /= 10;
	} else {
		/* Emergent Stop */
		acc = -MaxDeceleration;
	}

	//Our behavior was changed due to warning from RSU
	if(WarningDecelerate < 0)
		return -MaxDeceleration;
	if(acc > MaxAcceleration)
		return MaxAcceleration;
	if(acc < -MaxDeceleration)
		return -MaxDeceleration;
	else
		return  acc;
}

int main()
{
	int sleepingPeriod = 100000;
	char Name[60];
	init();
	while(1)
	{
		if(ITS_NET)
			reportMyStatusToAGroupOfNode();
		fflush(stdout);

		/* FIXME: If necessary, make every node wake up in different time.
		 * (Avoid collision of broadcast pkt on mac)
		 */
		//usleepAndReleaseCPU(myTCPsockfd, mynid, SLEEPING_PERIOD, 1);
		usleepAndReleaseCPU(myTCPsockfd, mynid, sleepingPeriod, 1);
		WarningDecelerate++;
		WakedTimes++;
		gettimeofday(&now, 0);

		if(ITS_NET){
			// receive msg from other nodes
			receiveMsg();
		}

		int n = 0;
		double *CandidateDirection = NULL;

		n = getCurrentPosition(myTCPsockfd, mynid, CurrentPOS_x, CurrentPOS_y,
CurrentPOS_z, 1);

		// See the Traffic light signal.
		n = getTheNearestTrafficLightInfrontOfMe(myTCPsockfd, mynid, CurrentPOS_x,
CurrentPOS_y, CurrentDirection, 100.0, SigGID, SigLight, SigPOS_x, SigPOS_y, SignalIndex, 1);
		if(n == 1)
```

```
		{
			SeenTheTrafficLightOrNot = 1;
		}
		else
			SeenTheTrafficLightOrNot = 0;

		// get next moving event (point) from event queue
		CurrentEvent = EventQueue.front();

		getCurrentMovingSpeed(myTCPsockfd, mynid, CurrentVelocity, 1);
		CurrentAcceleration = DetermineAcceleration();

		setCurrentSpeedAcceleration(myTCPsockfd, mynid, CurrentAcceleration, 1);

		if(EventQueue.empty() == false)
		{
			n = ReachTheNextTriggerPointOrNot();
			// n = 1; Reach the event point.
			// n = 0; Unreach the event point.
		}
		else
		{
			n = 1;
		}

		if(n == 0)
		{
			//Not reach the event point, do nothing, keep going.
			sleepingPeriod = 100000;
			continue;
		}
		else if(n == 1)
		{
			sleepingPeriod = 10000;
			/* create new event queue */
			if(((int)EventQueue.size()) == 0)
			{
				/* End of taking turns, so the queue became empty */
				int numOfDirections;
				double CorrectedPOS_x, CorrectedPOS_y;
				double a, b, c;

				// correct my position into the middle of the road
				n = selfCorrectness(CurrentPOS_x, CurrentPOS_y, CorrectedPOS_x,
CorrectedPOS_y, CacheOrNot, &myCache);
				if(n == 1)
				{
					CurrentPOS_x = CorrectedPOS_x;
					CurrentPOS_y = CorrectedPOS_y;
				}
				n = setCurrentWaypoint(myTCPsockfd, mynid, CurrentPOS_x,
CurrentPOS_y, CurrentPOS_z, 1);

				/* get front node block ID which connected by current edge.
				 * Note:
				 *    node block ID is the same as signal group ID
```

```
                                        */
                                        getFrontNID(CurrentPOS_x, CurrentPOS_y, SigGID, CacheOrNot,
&myCache);

                                        int RoadType;
                                        double endPOS_x, endPOS_y;

                                        n = getCurrentRoadInformation(CurrentPOS_x, CurrentPOS_y,
numOfDirections, CandidateDirection, a, b, c, endPOS_x, endPOS_y, RoadType, CacheOrNot,
&myCache);
                                        if(n <= 0){
                                                printf("Warning_1_2: Node is not on the lane!! Node %d
position(%lf, %lf), current direction %lf, eventPos (%lf, %lf), eventDirection %lf\n",
                                                                mynid, CurrentPOS_x, CurrentPOS_y,
CurrentDirection, CurrentEvent.x, CurrentEvent.y, CurrentEvent.direction);
                                                fflush(stdout);
                                                stopSimulation(myTCPsockfd, mynid);
                                        }

                                        if(RoadType == ROAD_TYPE_NODE)
                                        {
                                                double distanceToNextEventPoint =
Distance_BetweenTwoNode(CurrentPOS_x, CurrentPOS_y, CurrentEvent.x, CurrentEvent.y);
                                                double angleDiff = fabs(CurrentDirection -
CurrentEvent.direction);
                                                if(angleDiff > 180)
                                                        angleDiff = 360 - angleDiff;

                                                if(((CurrentEvent.x <= 0.000001) && (CurrentEvent.y <=
0.000001)) ||
                                                        ((distanceToNextEventPoint >= roadWidth) &&
(angleDiff < 90)))
                                                {
                                                        /*
                                                         * Node is in the node block at beginning ||
                                                         * Node skips one lane and directly enters into the
node block
                                                         * (It happens when the road length is very small)
                                                         */
                                                        double exitPOS_x, exitPOS_y, exitDirection;
                                                        double nextDirection;

                                                        n = getCurrentNodeExit(CurrentPOS_x,
CurrentPOS_y, exitPOS_x, exitPOS_y, exitDirection, nextDirection, CacheOrNot, &myCache);
                                                        if(n < 0)
                                                        {
                                                                printf("Node[%d] getCurrentNodeExit
error\n", mynid);
                                                        }

                                                        CurrentDirection = exitDirection;
                                                        // set current moving direction viewed on GUI
screen
                                                        n =
setCurrentMovingDirectionViewedOnGUIScreen(myTCPsockfd, mynid, CurrentDirection, 1);
```

```
                                        FillTheQueue(0, NULL, NULL, NULL, exitPOS_x,
exitPOS_y, nextDirection);
                                }
                                else
                                {
                                        /* After the turn, if car is still on the previous road,
                                         * make this car go to the final point of the previous
road
                                         * one more time.
                                         */
                                        FillTheQueue(0, NULL, NULL, NULL, CurrentEvent.x,
CurrentEvent.y, CurrentEvent.direction);
                                }
                                free(CandidateDirection);
                                CandidateDirection = NULL;
                                continue;
                        }

                        CurrentDirection = RandomDirection(numOfDirections,
CandidateDirection);
                        free(CandidateDirection);
                        CandidateDirection = NULL;

                        n = setCurrentMovingDirectionViewedOnGUIScreen(myTCPsockfd,
mynid, CurrentDirection, 1);

                        n = getNextRoadInformation(CurrentPOS_x, CurrentPOS_y,
CurrentDirection, numOfDirections, CandidateDirection, a, b, c, CacheOrNot, &myCache);
                        if(n <= 0){
                                printf("Warning_1_3: Cannot find the next road!! Node %d
position(%lf, %lf)\n", mynid, CurrentPOS_x, CurrentPOS_y);
                                fflush(stdout);
                                stopSimulation(myTCPsockfd, mynid);
                        }

                        if(numOfDirections == 0){
                                printf("Error: numOfDirections can't be zero\n");
                                fflush(stdout);
                                stopSimulation(myTCPsockfd, mynid);
                        }

                        ExpectedDirection = RandomDirection(numOfDirections,
CandidateDirection);
                        if(agentReceivedBrokenCarMsg == 1){
                                if(numOfDirections >= 2){
                                        while(ExpectedDirection == brokenCarDirection){
                                                ExpectedDirection =
RandomDirection(numOfDirections, CandidateDirection);
                                        }
                                }
                        }
                        free(CandidateDirection);
                        CandidateDirection = NULL;

                        int numOfTurns;
```

```
                                double *turningPOS_x = NULL, *turningPOS_y = NULL,
*DirectionQueue = NULL;
                                double FirstPointAfterTheTurn_x, FirstPointAfterTheTurn_y;

                                n = takeATurn(CurrentPOS_x, CurrentPOS_y, CurrentDirection,
ExpectedDirection, numOfTurns, turningPOS_x, turningPOS_y, FirstPointAfterTheTurn_x,
FirstPointAfterTheTurn_y, DirectionQueue, CacheOrNot, &myCache);
                                if(n <= 0)
                                {
                                        printf("Error: cannot get the taking turns info, error no. %d,
curPos (%lf, %lf), curDirection %lf, expectedDirection %lf\n", n, CurrentPOS_x, CurrentPOS_y,
CurrentDirection, ExpectedDirection);
                                        fflush(stdout);
                                        stopSimulation(myTCPsockfd, mynid);
                                }
                                FillTheQueue(numOfTurns, turningPOS_x, turningPOS_y,
DirectionQueue, FirstPointAfterTheTurn_x, FirstPointAfterTheTurn_y, ExpectedDirection);

                                delete turningPOS_x;
                                delete turningPOS_y;
                                delete DirectionQueue;

                                sprintf(Name,"/home/nctuns/Escritorio/Resultados/Delay/C-
%i",mynid);                     //M.Mateos
                                fdtA = fopen(Name,"a+");
                        //M.Mateos
                                fseek(fdtA,0,SEEK_END);
                        //M.Mateos
                                if(ftell(fdtA) == 0) {
                        //M.Mateos
                                        fseek(fdtA,0,SEEK_SET);
                        //M.Mateos
                                        fprintf(fdtA,"1");
                        //M.Mateos
                                }
                        //M.Mateos
                                fclose(fdtA);
                        //M.Mateos

                        }
                        else if(((int)EventQueue.size()) > 0)
                        {
                                //Pop an item from the queue
                                CurrentDirection = CurrentEvent.direction;
                                CurrentPOS_x = CurrentEvent.x;
                                CurrentPOS_y = CurrentEvent.y;
                                n = setCurrentWaypoint(myTCPsockfd, mynid, CurrentPOS_x,
CurrentPOS_y, CurrentPOS_z, 1);
                                n = setCurrentMovingDirectionViewedOnGUIScreen(myTCPsockfd,
mynid, CurrentDirection, 1);
                        }
                }
                else
                {
                        printf("Node[%d]: Error for reached or not, n %d\n", mynid, n);
                        fflush(stdout);
```

```
                    stopSimulation(myTCPsockfd, mynid);
                }
            }
}
```

ANNEX 6. "TRAFFIC-FILTER.AWK" CODE

```
BEGIN {
        salida="Light-26-Traffic.txt"               #Archivo de Salida Semaforo 26
        salida1="Light-29-Traffic.txt"              #Archivo de Salida Semaforo 29
        salida2="Light-32-Traffic.txt"              #Archivo de Salida Semaforo 32
        salida3="Light-35-Traffic.txt"              #Archivo de Salida Semaforo 35
        salida4="Light-38-Traffic.txt"              #Archivo de Salida Semaforo 38
        salida5="Light-41-Traffic.txt"              #Archivo de Salida Semaforo 41
        salida6="Light-44-Traffic.txt"              #Archivo de Salida Semaforo 44
        salida7="Light-47-Traffic.txt"              #Archivo de Salida Semaforo 47
        salida8="Light-50-Traffic.txt"              #Archivo de Salida Semaforo 50
        salida9="Light-53-Traffic.txt"              #Archivo de Salida Semaforo 53
        salida10="Light-56-Traffic.txt"             #Archivo de Salida Semaforo 56
        salida11="Light-59-Traffic.txt"             #Archivo de Salida Semaforo 59
        salida12="Light-62-Traffic.txt"             #Archivo de Salida Semaforo 62
        tiempo_26=0                     #Tiempo de Comparacion Semaforo 26
        valor_26=0                      #Valor de Estadistica a Imprimir Archivo 26
        tiempo_29=0                     #Tiempo de Comparacion Semaforo 29
        valor_29=0                      #Valor de Estadistica a Imprimir Archivo 29
        tiempo_32=0                     #Tiempo de Comparacion Semaforo 32
        valor_32=0                      #Valor de Estadistica a Imprimir Archivo 32
        tiempo_35=0                     #Tiempo de Comparacion Semaforo 35
        valor_35=0                      #Valor de Estadistica a Imprimir Archivo 35
        tiempo_38=0                     #Tiempo de Comparacion Semaforo 38
        valor_38=0                      #Valor de Estadistica a Imprimir Archivo 38
        tiempo_41=0                     #Tiempo de Comparacion Semaforo 41
        valor_41=0                      #Valor de Estadistica a Imprimir Archivo 41
        tiempo_44=0                     #Tiempo de Comparacion Semaforo 44
        valor_44=0                      #Valor de Estadistica a Imprimir Archivo 44
        tiempo_47=0                     #Tiempo de Comparacion Semaforo 47
        valor_47=0                      #Valor de Estadistica a Imprimir Archivo 47
        tiempo_50=0                     #Tiempo de Comparacion Semaforo 50
        valor_50=0                      #Valor de Estadistica a Imprimir Archivo 50
        tiempo_53=0                     #Tiempo de Comparacion Semaforo 53
        valor_53=0                      #Valor de Estadistica a Imprimir Archivo 53
        tiempo_56=0                     #Tiempo de Comparacion Semaforo 56
        valor_56=0                      #Valor de Estadistica a Imprimir Archivo 56
        tiempo_59=0                     #Tiempo de Comparacion Semaforo 59
        valor_59=0                      #Valor de Estadistica a Imprimir Archivo 59
        tiempo_62=0                     #Tiempo de Comparacion Semaforo 62
        valor_62=0                      #Valor de Estadistica a Imprimir Archivo 62
}

{
        semaforo=$5                     #ID Semaforo
        tiempo=$2                       #Tiempo
        estadistica=$8                  #Estadistica Acumulada

                if (semaforo == "26")
                {
                        if(tiempo_26 == 0)
                        {
                                tiempo_26=tiempo
```

```
                valor_26=estadistica
        }
        else
        {
                if(tiempo_26 == tiempo)
                {
                        valor_26=estadistica
                }
                else
                {
                        printf("%d\t%f\n",tiempo_26,valor_26) >> salida
                        tiempo_26=tiempo
                        valor_26=estadistica
                }
        }
}
if (semaforo == "29")
{
        if(tiempo_29 == 0)
        {
                tiempo_29=tiempo
                valor_29=estadistica
        }
        else
        {
                if(tiempo_29 == tiempo)
                {
                        valor_29=estadistica
                }
                else
                {
                        printf("%d\t%f\n",tiempo_29,valor_29) >> salida1
                        tiempo_29=tiempo
                        valor_29=estadistica
                }
        }
}
if (semaforo == "32")
{
        if(tiempo_32 == 0)
        {
                tiempo_32=tiempo
                valor_32=estadistica
        }
        else
        {
                if(tiempo_32 == tiempo)
                {
                        valor_32=estadistica
                }
                else
                {
                        printf("%d\t%f\n",tiempo_32,valor_32) >> salida2
                        tiempo_32=tiempo
                        valor_32=estadistica
                }
```

```
		}
	}
	if (semaforo == "35")
	{
		if(tiempo_35 == 0)
		{
			tiempo_35=tiempo
			valor_35=estadistica
		}
		else
		{
			if(tiempo_35 == tiempo)
			{
				valor_35=estadistica
			}
			else
			{
				printf("%d\t%f\n",tiempo_35,valor_35) >> salida3
				tiempo_35=tiempo
				valor_35=estadistica
			}
		}
	}
	if (semaforo == "38")
	{
		if(tiempo_38 == 0)
		{
			tiempo_38=tiempo
			valor_38=estadistica
		}
		else
		{
			if(tiempo_38 == tiempo)
			{
				valor_38=estadistica
			}
			else
			{
				printf("%d\t%f\n",tiempo_38,valor_38) >> salida4
				tiempo_38=tiempo
				valor_38=estadistica
			}
		}
	}
	if (semaforo == "41")
	{
		if(tiempo_41 == 0)
		{
			tiempo_41=tiempo
			valor_41=estadistica
		}
		else
		{
			if(tiempo_41 == tiempo)
			{
				valor_41=estadistica
```

```
                        }
                        else
                        {
                                printf("%d\t%f\n",tiempo_41,valor_41) >> salida5
                                tiempo_41=tiempo
                                valor_41=estadistica
                        }
                }

        }

        if (semaforo == "44")
        {
                if(tiempo_44 == 0)
                {
                        tiempo_44=tiempo
                        valor_44=estadistica
                }
                else
                {
                        if(tiempo_44 == tiempo)
                        {
                                valor_44=estadistica
                        }
                        else
                        {
                                printf("%d\t%f\n",tiempo_44,valor_44) >> salida6
                                tiempo_44=tiempo
                                valor_44=estadistica
                        }
                }

        }
        if (semaforo == "47")
        {
                if(tiempo_47 == 0)
                {
                        tiempo_47=tiempo
                        valor_47=estadistica
                }
                else
                {
                        if(tiempo_47 == tiempo)
                        {
                                valor_47=estadistica
                        }
                        else
                        {
                                printf("%d\t%f\n",tiempo_47,valor_47) >> salida7
                                tiempo_47=tiempo
                                valor_47=estadistica
                        }
                }

        }
        if (semaforo == "50")
```

```
{
        if(tiempo_50 == 0)
        {
                tiempo_50=tiempo
                valor_50=estadistica
        }
        else
        {
                if(tiempo_50 == tiempo)
                {
                        valor_50=estadistica
                }
                else
                {
                        printf("%d\t%f\n",tiempo_50,valor_50) >> salida8
                        tiempo_50=tiempo
                        valor_50=estadistica
                }
        }

}
if (semaforo == "53")
{
        if(tiempo_53 == 0)
        {
                tiempo_53=tiempo
                valor_53=estadistica
        }
        else
        {
                if(tiempo_53 == tiempo)
                {
                        valor_53=estadistica
                }
                else
                {
                        printf("%d\t%f\n",tiempo_53,valor_53) >> salida9
                        tiempo_53=tiempo
                        valor_53=estadistica
                }
        }

}
if (semaforo == "56")
{
        if(tiempo_56 == 0)
        {
                tiempo_56=tiempo
                valor_56=estadistica
        }
        else
        {
                if(tiempo_56 == tiempo)
                {
                        valor_56=estadistica
                }
```

```
                        else
                        {
                                printf("%d\t%f\n",tiempo_56,valor_56) >> salida10
                                tiempo_56=tiempo
                                valor_56=estadistica
                        }
                }

        }
        if (semaforo == "59")
        {
                if(tiempo_59 == 0)
                {
                        tiempo_59=tiempo
                        valor_59=estadistica
                }
                else
                {
                        if(tiempo_59 == tiempo)
                        {
                                valor_59=estadistica
                        }
                        else
                        {
                                printf("%d\t%f\n",tiempo_59,valor_59) >> salida11
                                tiempo_59=tiempo
                                valor_59=estadistica
                        }
                }

        }
        if (semaforo == "62")
        {
                if(tiempo_62 == 0)
                {
                        tiempo_62=tiempo
                        valor_62=estadistica
                }
                else
                {
                        if(tiempo_62 == tiempo)
                        {
                                valor_62=estadistica
                        }
                        else
                        {
                                printf("%d\t%f\n",tiempo_62,valor_62) >> salida12
                                tiempo_62=tiempo
                                valor_62=estadistica
                        }
                }

        }
}

END {
```

```
        printf("%d\t%f\n",tiempo_26,valor_26) >> salida
        printf("%d\t%f\n",tiempo_29,valor_29) >> salida1
        printf("%d\t%f\n",tiempo_32,valor_32) >> salida2
        printf("%d\t%f\n",tiempo_35,valor_35) >> salida3
        printf("%d\t%f\n",tiempo_38,valor_38) >> salida4
        printf("%d\t%f\n",tiempo_41,valor_41) >> salida5
        printf("%d\t%f\n",tiempo_44,valor_44) >> salida6
        printf("%d\t%f\n",tiempo_47,valor_47) >> salida7
        printf("%d\t%f\n",tiempo_50,valor_50) >> salida8
        printf("%d\t%f\n",tiempo_53,valor_53) >> salida9
        printf("%d\t%f\n",tiempo_56,valor_56) >> salida10
        printf("%d\t%f\n",tiempo_59,valor_59) >> salida11
        printf("%d\t%f\n",tiempo_62,valor_62) >> salida12
        close(salida)
        close(salida1)
        close(salida2)
        close(salida3)
        close(salida4)
        close(salida5)
        close(salida6)
        close(salida7)
        close(salida8)
        close(salida9)
        close(salida10)
        close(salida11)
        close(salida12)
}
```

ANNEX 7. "DELAY-FILTER.AWK" CODE

```
BEGIN {
        salida="Light-26-Delay.txt"             #Archivo de Salida Semaforo 26 para Retardo
        salida1="Light-29-Delay.txt"            #Archivo de Salida Semaforo 29 para Retardo
        salida2="Light-32-Delay.txt"            #Archivo de Salida Semaforo 32 para Retardo
        salida3="Light-35-Delay.txt"            #Archivo de Salida Semaforo 35 para Retardo
        salida4="Light-38-Delay.txt"            #Archivo de Salida Semaforo 38 para Retardo
        salida5="Light-41-Delay.txt"            #Archivo de Salida Semaforo 41 para Retardo
        salida6="Light-44-Delay.txt"            #Archivo de Salida Semaforo 44 para Retardo
        salida7="Light-47-Delay.txt"            #Archivo de Salida Semaforo 47 para Retardo
        salida8="Light-50-Delay.txt"            #Archivo de Salida Semaforo 50 para Retardo
        salida9="Light-53-Delay.txt"            #Archivo de Salida Semaforo 53 para Retardo
        salida10="Light-56-Delay.txt"           #Archivo de Salida Semaforo 56 para Retardo
        salida11="Light-59-Delay.txt"           #Archivo de Salida Semaforo 59 para Retardo
        salida12="Light-62-Delay.txt"           #Archivo de Salida Semaforo 62 para Retardo

# Este fichero esta disenado para una distancia de carreteras o calles de 100 y una velocidad maxima de
10 m/seg por lo que el retardo minimo posible sera el conseguido al recorrer la calle con la velocidad
maxima (t = 100m/(10m/s) = 10 seg)
}

{
        semaforo=$5                     #ID Semaforo
        tiempo=$2                       #Tiempo
        retardo=$11                     #Retardo

                if (retardo >= 10)
                {
                        if(semaforo == "26")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida
                                }
                        else if (semaforo == "29")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida1
                                }
                        else if (semaforo == "32")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida2
                                }
                        else if (semaforo == "35")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida3
                                }
                        else if (semaforo == "38")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida4
                                }
                        else if (semaforo == "41")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida5
                                }
                        else if (semaforo == "44")
```

```
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida6
                                }
                        else if (semaforo == "47")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida7
                                }
                        else if (semaforo == "50")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida8
                                }
                        else if (semaforo == "53")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida9
                                }
                        else if (semaforo == "56")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida10
                                }
                        else if (semaforo == "59")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida11
                                }
                        else if (semaforo == "62")
                                {
                                        printf("%f\t%f\n",tiempo,retardo) >> salida12
                                }
                }
}

END {
        close(salida)
        close(salida1)
        close(salida2)
        close(salida3)
        close(salida4)
        close(salida5)
        close(salida6)
        close(salida7)
        close(salida8)
        close(salida9)
        close(salida10)
        close(salida11)
        close(salida12)
}
```

REFERENCES

[1] S. Olariu and M. Weigle, “Vehicular Networks, From Theory To Practice”, CRC Press, Taylor & Francis Group, 2009

[2] European Comission CORDIS, [Online] January 2012, http://cordis.europa.eu

[3] NEC Laboratory Europe, FleetNet – Internet on the Road: Ad Hoc Radio Network for Inter-Vehicle Communications, January 2002

[4] Network on Wheels (NOW), [Online] January 2012, http://www.network-on-wheels.de

[5] PReVENT, [Online] January 2012, http://www.prevent-ip.org

[6] CARLINK, [Online] January 2012, http://carlink.lcc.uma.es/

[7] SEVECOM, [Online] January 2012, http://www.sevecom.org/

[8] COMeSafety: Communications for eSafety, [Online] January 2012, http://www.comesafety.org/

[9] SAFESPOT, [Online] January 2012, http://www.safespot-eu.org/

[10] COOPERS, [Online] January 2012, http://www.coopers-ip.eu/

[11] CVIS project, [Online] January 2012, http://www.cvisproject.org/

[12] CAR 2 CAR Communication Consortium, [Online] January 2012, http://www.car-to-car.org/

[13] ESAFETYAWARE, [Online] January 2012, http://www.esafetyaware.eu/

[14] Open Living Labs, [Online] January 2012, http://www.openlivinglabs.eu

[15] Living Labs Global, [Online] January 2012, http://www.livinglabs-global.com

[16] ARIB Project, ‘Dedicated Short-Range Communication System’ English Translation, [Online], January 2012 http://www.arib.or.jp/english/html/overview/doc/5-STD-T75v1_0-E2.pdf

[17] IntelliDrive, [Online] January 2012, http://www.intellidriveusa.org/

[18] Integrated Vehicle-Based Safety Systems (IVBSS), [Online] January 2012, http://www.its.dot.gov/ivbss/

[19] Cooperative Intersection Collision Avoidance System (CICAS), [Online] January 2012, http://www.its.dot.gov/cicas/

[20] Houda Labiod, "Wireless Ad Hoc and Sensor Networks" Wiley-ISTE, 2008, ISBN: 1848210035

[21] D. Prakash, C. de Morais “Ad Hoc & Sensor Networks”, World Scientific – 2006

[22] F. Zhao, L. J. Guibas “Wireless Sensor Networks”, Morgan Kaufmann – 2004

[23] K. Sohraby, D. Minoli and T. Znati, “Wireless Sensor Networks. Technology, Protocols and Applications”, John Wiley & Sons, Inc., 2007

[24] IEEE, Computer Society, "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPAN's)”, New York, 2006

[25] E. Biagioni, K. Bridges and B. J. S. Chee, "PODS: A remote ecological Micro sensor network", EURASIP Journal on Wireless Communications and Networking archive Volume 2005 Issue 5 - October 2005

[26] Harvard Sensor Network Lab, [Online] January 2012, http://fiji.eecs.harvard.edu/Volcano

[27] T. Dang, S. Frolov, N. Bulusu, A. Baptista “Near Optimal Sensor Selection in the COlumbia RIvEr (CORIE) Observation Network for Data Assimilation Using Genetic Algorithms” (DCOSS'07 Proceedings of the 3rd IEEE international conference on Distributed computing in sensor system – Berlin 2007)

[28] E. Cayirci and T. Coplu, “SENDROM: sensor networks for disaster relief operations management”, Journal Wireless Network Volume 13 Issue 3 – 2007

[29] L. Schwiebert, S. K. S. Gupta and J. Weinmann, “Research challenges in Wireless Networks of biomedical sensors”, Research challenges in Wireless Networks of biomedical sensors – 2001

[30] M. F. Durate and Y. H. Hu, “Vehicle classification in Distributed Sensor Networks”,Journal of Parallel and Distributed Computing Volume 64 Issue 7 – Orlando July 2004

[31] R. Popescu, I. Radusch, R. A. Rigani “Vehicular-2-X Communications: State-of-the-Art and Research in Mobile Vehicular Ad hoc Networks”, Springer – 2010

[32] W. Chen, S. Cai, "Ad Hoc Peer-to-Peer Network Architecture for Vehicle Safety Communications", IEEE Communications Magazine, pp. 100-107, April 2005.

[33] Delay-Tolerant Networking Research Group (DTNRG), [Online] January 2012, http://www.dtnrg.org/

[34] Brandon Hill, "Daimler-Chrysler Tests WiFi Car-to-Car Communication System" , [Online] January 2012, http://www.dailytech.com/article.aspx?newsid=5257

[35] Network On Weels (NOW), [Online] January 2012, http://www.network-on-wheels.de/

[36] J. Tarng, B. Chuang, and F. Wu, “A novel stability-based routing protocol for mobile ad-hoc,” IEICE Transactions on Communications, vol. E90-B, no. 4, pp. 876–884, April, 2007.

[37] W. Sun, H. Yamaguchi, K. Yukimasa and S. Kusumoto, "GVGrid: A QoS Routing Protocol for Vehicular Ad Hoc Networks," Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on , vol., no., pp.130-139, 19-21 June 23006

[38] Yan Gongjun, D. B. Rawat and B. B. Bista, "Provisioning Vehicular Ad Hoc Networks with Quality of Service," Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on , vol., no., pp.102-107, 4-6 Nov. 2010

[39] G. Caizzone, P. Giacomazzi, L. Musumeci and G. Verticale, "A power control algorithm with high channel availability for vehicular ad hoc networks," Communications, 2005. ICC 2005. 2005 IEEE International Conference on , vol.5, no., pp. 3171- 3176 Vol. 5, 16-20 May 2005

[40] Li Xiaoyan, T. D. Nguyen and R. P. Martin, "Using adaptive range control to maximize 1-hop broadcast coverage in dense wireless networks," Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on , vol., no., pp. 397- 405, 4-7 Oct. 2004

[41] Europe's Information Society Thematic Portal, [Online] January 2012, http://ec.europa.eu/information_society/activities/esafety/ecall/index_en.htm

[42] C. Tripp, K. Ornelas, G. D. Delgado and M. Aguilar (Universitat Politècnica de Catalunya), "Evaluación de prestaciones de una red híbrida vehicular y de sensores para mejorar la seguridad vial", Barcelona 2010

[43] G. Sklyarenko; "AODV Routing Protocol", Seminar Technische Informatik, Freie Universitat Berlin, 2006

[44] D. Tacconi, I. Carreras, D. Miorandi, I. Chlamtac, F. Chiti and R. Fantacci, "Supporting the sink mobility: a case study for wireless sensor networks", Proc. of IEEE ICC, Scotland, 2007

[45] D. Tacconi, I. Carreras, D. Miorandi, F. Chiti, A. Casile and R. Fantacci, "A system architecture supporting mobile applications in disconnected sensor networks" Proc. of GLOBECOM, Washington, USA, 2007

[46] D. Tacconi, D. Miorandi, I. Carreras, F. Chiti and R. Fantacci, "Using wireless sensor networks to support intelligent transportation systems", Florence, Italy, January 2010

[47] P. Regañas, "Study of vehicular networks in urban and interurban scenarios", Career Final Project, UPC 2012.

[48] C. Lochert et al. "A Routing Strategy for Vehicular Ad Hoc Networks in City", IEEE, Intelligent Vehicles Symposium '03. 2003. pp. 156-161

[49] J. Tian, L. Han and K. Rothermel, "Spatially Aware Packet Routing for Mobile Ad Hoc Inter-Vehicle Radio Networks", IEEE Intelligent Transportation Systems, Shanghai, China : s.n., 2003. pp. 1546-1551

[50] S. See et al, "A-STAR: A Mobile Ad Hoc Routing Strategy for Metropolis", 3rd International Networking Conference IFIP-TC6 (IFIP '04), 2004

[51] V. Naumov and T. Gross, "Connectivity-Aware Routing (CAR) in Vehicular Ad Hoc Networks", IEEE INFOCOM 2007, 2007

[52] C. Lochert et al. "Geographic Routing in City Scenarios", ACM SIGMOBILE Mobile Computing and Communications Review, 2005, Vol. 9, pp. 69-72

[53] J. Zhao and J. Cao, "VADD: Vehicle-assisted data delivery in vehicular ad hoc networks", 25th IEEE International Conference on Computer Communications, INFOCOM 2006, 2006, pp.1-12

[54] B. Hofman-Wellwnhof, H. Lichtenegger and J. Collins, "Global Positioning System: Theory and Practice", Springer, 2001, 5th Edition

[55] E. Kaplan and C. Hegarty, "Understanding GPS: Principles and Applications", Boston : Artech Hourse, 2006, 2nd Edition

[56] D. Niculescu, and B. Nath, "Trajectory Based Forwarding and Its Applications", 9th International Conference on Mobile Computing and Networking (MOBICOM '03), September 2003, pp. 260-272

[57] I. Leontiadis and C. Mascolo, "GeOpps: Geographical Opportunistic Routing for Vehicular Networks", World of Wireless, Mobile and Multimedia Networks, WoWMoM 2007, 2007

[58] K. Sohraby, "Wireless sensor networks: technology, protocols and applications", United States of America, 2007

[59] C. Intanagonwiwat, R. Govindan, D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks" In: Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MOBICOM), Boston, MA, USA, August, 2000

[60] D. Braginsky and D. Estrin, "Rumor Routing Algorithm for Sensor Networks", in the Proceedings of the First Workshop on Sensor Networks and Applications (WSNA), Atlanta, GA, October 2002

[61] F. Ye, A. Chen, S. Liu and L. Zhang, "A scalable solution to minimum cost forwarding in large sensor networks", Proceedings of the tenth International Conference on Computer Communications and Networks (ICCCN), pp. 304-309, 2001

[62] R. C. Shah and J. M. Rabaey, "Energy Aware Routing for Low Energy Ad Hoc Sensor Networks" In Proceedings of the Wireless Communications and Networking Conference (WCNC), Orlando, FL, USA, March, 2002; pp. 350–355

[63] I. Mabrouki, X. lagrange and G. Froc, "Random Walk Based Routing Protocol for Wireless Sensor Networks", ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium 2007

[64] M. Younis, M. Youssef and K. Arisha, "Energy-Aware Routing in Cluster-Based Sensor Networks", in the Proceedings of the 10th IEEE/ACM(MASCOTS2002), Fort Worth, TX, October 2002

[65] S. Lindsay and C. Raghavendra, "PEGASIS: Power-Efficient Gathering in Sensor Information Systems", international Conf. on Communications, 2001

[66] A. Manjeswar and D. P. Agrawal, "TEEN: A protocol for enhanced efficiency in wireless sensor networks", In Proceedings of 1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, San Francisco, CA, USA, 2001, p. 189

[67] A. Manjeswar and D. P. Agrawal, "APTEEN: A hybrid protocol for efficient routing and comprehensive information retrieval in wireless sensor networks", In Proceedings of 2nd

International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, Fort Lauderdale, FL, USA, April 15–19, 2002, pp. 195-202

[68] V. Rodoplu and T. H. Ming, "Minimum energy mobile wireless networks," IEEE Journal of Selected Areas in Communications, Vol. 17, No. 8, pp. 1333-1344, 1999

[69] L. K. Subramanian, "An architecture for building selfconfigurable systems", in First Annual Workshop on Mobile and Ad Hoc Networking and Computing 2000. MobiHOC, 2000

[70] Q. Fang, F. Zhao, and L. Guibas, "Lightweight Sensing and Communication Protocols for Target Enumeration and Aggregation," Proc. 4th ACM MOBIHOC, 2003, pp. 165–76

[71] J. N. Al-karaki et al. "Data Agggregation in Wireless Sensor Networks – Exact and Approximate Algorithms," Proc. IEEE Wks. High Perf. Switching and Routing 2004, Phoenix, AZ, Apr. 18-21, 2004

[72] S. Lindsey and C. Raghavendra, "PEGASIS: Power-Efficient Gathering in Sensor Information Systems", IEEE Aerospace Conference Proceedings, 2002, Vol. 3, 9-16 pp. 1125-113

[73] F. Ye et al. "A Two-Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks," Proc. ACM/IEEE MOBICOM, 2002

[74] Y. Xu, J. Heidemann and D. Estrin, "Geography-informed energy conservation for ad hoc routing", in Proc. IEEE Ann. Int. Conf. Mob. Comp. Netw., Rome, Italy, 2001

[75] Y. Yu, R. Govindan and D. Estrin, "Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks", UCLA Computer Science Department UCLA-CSD TR-01-0023, May, 2001.

[76] I. Stojmenovic and X. Lin, "GEDIR: Loop-Free Location Based Routing in Wireless Networks", In International Conference on Parallel and Distributed Computing and Systems, Boston, MA, USA, Nov. 3-6, 1999

[77] A. K. Dwivedi and O. P. Vyas, "Network Layer Protocols for Wireless Sensor Networks: Existing Classifications and Design Challenge", International Journal of Computer Applications (0975 – 8887), Volume 8– No.12, October 2010

[78] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Pichler-Milanovic and E. Meijers, "Smart cities – Ranking of European medium-sized cities", http://www.smart-cities.eu/, Vienna: Centre of Regional Science, Retrieved 2009-11-11.

[79] S. Y. Wang and H. T. Kung, "A Simple Methodology for Constructing Extensible and High-Fidelity TCP/IP Network Simulators", IEEE INFOCOM'99, March 21-25, 1999, New York, USA

[80] J. E. Smith and R. Nair, "The Architecture of Virtual Machines", Computer (IEEE Computer Society) 38 (5): 32–38 (2005)

[81] Virtual Box Virtualization Software, [Online] January 2012, https://www.virtualbox.org/

[82] VMWare Virtualization Software, [Online] January 2012, http://www.vmware.com/es/

[83] S. Y. Wang, C. C. Lin and C. C. Huang, "NCTUns Tool for Evaluating the Performances of Real-life P2P Applications," a chapter of the "Peer-to-Peer Networks and Internet Policies" book, (ISBN 978-1-60876-287-3, published by Nova Science Publishers in 2010)

[84] VMWare Player, [Online] January 2012, http://www.vmware.com/products/player/

[85] Getting Started Guide VMware Player 3.1, [Online] January 2012, http://www.vmware.com/pdf/vmware_player310.pdf

[86] ENTEL server, [Online] January 2012, http://bowie.upc.es/vmware/vm-fedora.tgz

[87] WinRar Software, [Online] January 2012, http://www.winrar.es/

[88] Departament d'Enginyeria Telemàtic, [Online] January 2012, http://www.entel.upc.edu/

[89] 2012 IEEE Intelligent Vehicles Symposium: IV'12- 3-7 June 2012, Alcalá de Henares, Spain, [Online] February 2012, http://www.robesafe.es/iv2012/

[90] NCTUns 6.0 Network Simulator and Emulator, [Online] February 2012, http://nsl.csie.nctu.edu.tw/nctuns.html