Titre: Title:	A Presence-Based Architecture for a Gateway to Integrate Vehicular Ad-Hoc Networks (VANETs), the IP Multimedia Subsystems (IMS) and Wireless Sensor Networks (WSNs)
Auteur: Author:	Mohab Aly
Date:	2013
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Aly, M. (2013). A Presence-Based Architecture for a Gateway to Integrate Vehicular Ad-Hoc Networks (VANETs), the IP Multimedia Subsystems (IMS) and Wireless Sensor Networks (WSNs) [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1070/

UNIVERSITÉ DE MONTRÉAL

A PRESENCE-BASED ARCHITECTURE FOR A GATEWAY TO INTEGRATE VEHICULAR AD-HOC NETWORKS (VANETs), THE IP MULTIMEDIA SUBSYSTEMS (IMS) AND WIRELESS SENSOR NETWORKS (WSNs)

MOHAB ALY

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

FÉVRIER 2013

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

A PRESENCE-BASED ARCHITECTURE FOR A GATEWAY TO INTEGRATE VEHICULAR AD-HOC NETWORKS (VANETs), THE IP MULTIMEDIA SUBSYSTEMS (IMS) AND WIRELESS SENSOR NETWORKS (WSNs)

présenté par : ALY Mohab

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de :

Mme BELLAÏCHE Martine, Ph.D., présidente

M. QUINTERO Alejandro, Doct, membre et directeur de recherche

M. PIERRE Samuel, Ph.D., membre

DEDICATION

To my parents “Prof. Hassan ElNaggar”, “Dr. Ebtessam Abd ElAal”, my sister “Dr. Menna ElNaggar” and my big family for their infinite love and support.

To the actors who were behind the scene, who gave me the love and courage I need every once in a while. To “Dr. Ahmed ElShafee” for sharing his knowledge whenever I need, to “Dr. Shahira ElAlfy” for her endless support, motivation and kindness and to “Dr. Sherif ElShafei” who provided me the good means of teaching during my undergraduate studies.

To my close friends who really care for me “Menna H.”, “A. Serag”, “I. Abou Hashim”, “Alaa Badr El-Deen”, “Mostafa M. Marzok”, “Islam Taha”, and “Salma ElShaarawi”

To my ”NU” colleagues, who care about my success every single time and motivate me to continue whenever I feel down, specially “Muhammad Daif”, “Heba Shalaby”, “Mona Ragab”, “Mohamed Naiel”, “Ahmed Othman”, “Ahmed M. Foda”, “Ahmed Zidan”, “Loay Elalfy”, “Mohamed Kilany”, “Walid Mohamed”, “Ahmed Gamal”, “Mostafa ElAttar”, “AbdAllah Motaal”, “Ahmed Salaheldin”, “Ahmed Ahmedin” and “Mohamed Samir ElBaz”.

To my “ACU” friends who were and still my source of motivation.

To my second family here in Canada, to “Ahmed Hamed”, “Yara Zayed”, “Dina Zayed”, “Salsabeel Zayed”, “Mohamed Naiel” and “Esraa Awad”. They encourage me to keep going on once I decided not to continue and return back to my beloved country. They helped me morally.

To “Dr. Iman M. AbdElAziz” for sharing her support and motivation to continue studying and do great while proceeding in this project and thesis.

I would like to thank my residence’s friends “Karim Nagi”, “Adham Ismail”, “Ahmed A. Aziz”, and “Yasser T. Shaaban” for the good times we spent all together.

To everyone who helped me finishing this work specially “Ahmed Ragab”, “Moussa Ouedraogo” and “Serge ElHelou”.

To all my “Polytechnique” colleagues who were there whenever I needed any help and advice.

ACKNOWLEDGEMENTS

To my director of research, Prof. Alejandro Quintero, who provided me the means, the supervision and the pieces of advice to successfully complete this project.

To “Prompt Lab” and “LARIM Lab” colleagues who share knowledge, wisdom and joy with me all the time while being here during my studies specially “Ryan Shayegan”, “Richard Jaramillo”, “Salah-Eddine Benbrahim”, and “Mahsima Rahimi”.

I really appreciate…

ABSTRACT

On one hand, IP Multimedia Subsystems (IMS) are a research area that has been gaining attention from the research community. It aims to provide cellular access to all Internet services. It is a control architecture on the top of the IP layer whose goal is dependent on the provision of the Quality of Service (QoS), integrated services and fair charging scheme throughout standard interfaces. On the other hand, Vehicular Ad-hoc Networks (VANETs) are a new communication paradigm that enables the wireless communication between vehicles moving with high speeds, as well as the vehicles and the road side equipments found along the roads. This opened the door to develop several new applications like, traffic engineering, traffic management, dissemination of emergency information to avoid critical situations, comfort and entertainment and other user applications. Moreover, VANETs are a sub-class of mobile ad-hoc networks; the performance of the communication depends on how better the routing takes place in the network. Routing of data depends on the routing protocols being used in the network. Combining the capabilities of IMS world with the VANET world opens the door to deploy a wide range of novel multimedia services. This dissertation proposes a presence-based architecture for the integration of IMS with VANETs. The presence of the middleware is used to make an instantaneous awareness of the VANETs changes as well as of the IMS format and to select the best delivery strategy between the two architectures. The gateway which is the heart of our architecture is an overlay built on the top of the IMS as well as the VANETs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

3GPP	Third Generation Partnership
ACK	Acknowledgement
AODV	Ad-hoc On-demand Distance Vector
CSCF	Call/Session Control Function
GW	Gateway
HSS	Home Subscriber Server
ICMP	Internet Control Messaging Protocol
IMS	IP Multimedia Subsystems
ITS	Intelligent Transportation System
LUNAR	Lightweight Underlay Network Ad hoc Routing
MANET	Mobile Ad-hoc Network
OBU	On Board Unit
PEDS	Persistent Encrypted Data Storage
QoS	Quality of Service
RSU	Road Side Unit
SIP	Session Initiation Protocol
UE	User Equipments
VANET	Vehicular Ad-hoc Network
WiFi	Wireless Fidelity

WSN Wireless Sensor Network

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Everything is becoming wireless. The fascination of mobility, accessibility and flexibility makes wireless technologies the dominant method of transferring all sorts of information. Satellite television, cellular phones and wireless Internet are well-known applications of wireless technologies.

Moreover, recently, applications and technologies are rarely seen isolated. Users expect to have access to a set of services without needing to know which system handles which service. Thus, systems integration and standardization has become a key-point towards ubiquitous access.

This project focuses on integrating two technologies together, the IP Multimedia Subsystem (IMS) and Vehicular Ad-hoc Networks (VANETs) in the presence of the Wireless Sensor Networks (WSNs). Both of them are evolving technologies and have gain more interest in the research community in recent years because of the great advantages and services they provide. As in this work, the integration of both worlds provides further services and applications in a standardized and decoupled environment. This work presents a promising gateway that interconnects the IMS world with the VANET world and introduces a tiny contribution to its research community.

The key element of the 3G network right now is the IP Multimedia Subsystems (IMS). It is deployed in such a way that it can provide multimedia services to the end-users on the top of the IP transport Layer [1]. Furthermore, it handles access from WiFi networks, and is continuing to be extended into an access-independent platform for service delivery, including broadband fixed-line access. Also it guarantees the provision of the seamless roaming between each of mobile, public WiFi and private networks as well for a wide range of services and devices. It is based on three main characteristics [2]:

1. The provision of the Quality of Service (Qos) to real-time sessions. That gives the operator the chance to control the service a user can get; considering the variety from one user to another.
2. The fair charging scheme to multimedia services is allowed. This is an important factor that can be considered since sessions can be created among users or services located in different networks.

3. Services integration throughout standard interfaces. IMS enables services such as instant messaging, conferencing and third party call to cellular user through standard interfaces.

On the other hand, Vehicular Ad-hoc Networks (VANETs) is a new challenging network environment that pursues the concept of ubiquitous computing for future [3]. Vehicles are equipped with wireless communication technologies and they act like computer nodes will be on the road soon and this will revolutionize the concept of travelling. They are considered as an off-shoot of Mobile Ad-hoc Networks (MANETs); however they have some distinguishing characteristics too. The solutions proposed for MANETs need to be evaluated carefully and then adapted in order to be used in VANET context. Besides, VANETs are also similar to MANETs in many ways [3]. For example, both networks are multi-hop mobile networks having dynamic topology. There is no central entity, and the nodes route the data themselves across the network. Both MANETs and VANETs are rapidly deployable, without the need of an infrastructure. Although, MANET and VANET, both are mobile networks, however, the mobility pattern of VANET nodes is such that they move on specific paths (roads) and hence not in random direction. This gives VANETs some advantages over MANETs as the mobility patterns of VANET nodes is predictable. Another difference is the highly dynamic topology of VANETs as vehicles may move at high velocities. This makes the lifetime of the communication links between the nodes quiet short. Node density in VANETs is also unpredictable; during rush hours, the roads are crowded with vehicles, whereas at other times, lesser vehicles are there. Similarly, some roads have more traffic than other roads [3].

Wireless Sensor Networks (WSNs) are technology which is becoming more mature and is gaining momentum as one of the enabling technologies for the future Internet. Therefore, it is being applied ubiquitously and, in particular, to Intelligent Transportation Systems (ITS). They consist of medium to large networks of inexpensive wireless sensor nodes capable of sensing, processing and distributing information acquired from the environment through the collaborative efforts of nodes [4]. WSNs provide significant advantages both in cost as well as in distribute intelligence. On the one hand, installation and maintenance expenses are reduced because of the use of cheap devices which do not required wiring. Furthermore, distributed intelligence enables

the development of diverse real-time traffic safety applications not feasible with centralized solutions.

Moreover, WSNs cannot be regarded just as stand-alone systems intended for ITS; on the contrary, they should be considered in the ITS context as additional components of a heterogeneous system, where they cooperate with other technologies such as VANET.

1.1 Definitions and Basic Concepts

This section presents the background information pertinent to this project. Its goal is to give the reader a better understanding as to what are the networks and the technologies used as base for the integration.

Firstly a brief survey of IMS is presented. Later on, VANET is introduced as technology and is explained in further details. Then a sufficient concern will be given for the WSN.

1.1.1. IP Multimedia Subsystems (IMS)

IMS [5] is a standardized network architecture that is designed to merge cellular networks and the Internet. The third generation partnership (3GPP) has standardized IMS. The purpose of this converged network is to provide a platform for all kinds of multimedia services, both basic calling services as well as enhanced services. The services include among others video sharing and Push-to-talk over Cellular [6]. IMS is designed to make it easy to develop and deploy new services. Additionally, two or more services can be integrated into one new service. IMS uses open standard IP protocols to enhance the compatibility between IMS and the Internet. The goal is to deliver multimedia services between the fixed- and the cellular networks and within the networks themselves, but without making these services available to the public internet or allowing new operators on the public internet to provide services to the fixed and mobile telecommunications users. The reason not to allow these new operators to offer services to these users is that the mobile access operators want to keep their monopoly or near monopoly positions [7]. 3GPP has defined a list of requirements that IMS must fulfill. Where IMS is defined as: “An architectural framework created for the purpose of delivering IP multimedia services to end-users.” [7]. These requirements state that the framework is needed to support IP Multimedia sessions, quality of service (QoS), interworking with the Internet and the circuit

switched network, roaming, operators' ability to strong control the users' services, and that new services do not need to be standardized [7]. The second requirement above concerning Quality of Service (QoS), means that the user is to be guaranteed a certain amount of bandwidth and bounded packet delay. Traditionally packet-switched networks only provided best-effort delivery, which means that the IP packets are not guaranteed to arrive at the destination without loss or corruption of packets or even within any specific time bound. Of course higher layer protocols can be used to provide reliability if it is desired, but this is the reverse to the usual model for circuit switched telecommunication where it is assumed that the network provides in order delivery of bits with a bounded delay furthermore the error rate is determined by the links used and error bounds are based upon engineering and management selections of the appropriate link. Another contrast is that of availability of services, in the circuit switched model you either have the service or you do not, while in the packet switched model you may have at least some low quality service even in the worst of conditions [7]. Cellular networks have emphasized roaming as a service. In IMS this means that a user can be in a foreign country (or non-home operator's network) and still be able to use his/her mobile operator's services (i.e. Video sharing, Push-to-talk over Cellular, etc.) [7].With the introduction of IMS, a mobile access network operator will have the ability to monitor each service which a user is using. This will make it easier for the operator to apply specific business model for each service as part of a use-by-use cost model, while controlling what kind of services the user is allowed to use. The advantage is that users will be able to compare the price of each service between the different operators, and this will enhance the competition between the operators to retain and attract subscribers. However, if the user is using a service that is not provided by the mobile network operator, then the operator will only be able to see how much data is sent over the packet network to/from the user. If the majority of users use these services, then these users may prefer a flat rate model. It is up to the operator which business model to use, such as: flat rate, time-based, service based, or QoS based. The signaling protocol that is used in IMS is SIP (Session Initiation Protocol), and it works over multiple transport protocols (e.g., TCP, UDP, and SCTP).

1.1.1.1. IMS Components

IMS is an overlay control layer on top of an IP layer. As shown in Figure 1-1, four logical layers are defined as part of the IMS architecture. The device layer represents the different networks that could access IMS when connecting to an IP network. The transport layer is responsible for initiating and terminating SIP sessions and providing conversion of data. In addition to the IP network, the transport layer allows IMS devices to make and receive calls to and from the PSTN.

The control layer is composed by the CSCF (Call/Session Control Function) and the HSS (Home Subscriber Server), which are essential entities in the IMS architecture. CSCF refers to a set of SIP servers (i.e. Proxy-CSCF, I-CSCF and S-CSCF) that process SIP signaling. And the HSS is a central repository that contains user-related information [1].

Finally, on top of the IMS network architecture is the service layer. At this layer, application servers are found. An application server is a SIP entity that hosts and executes IMS services.

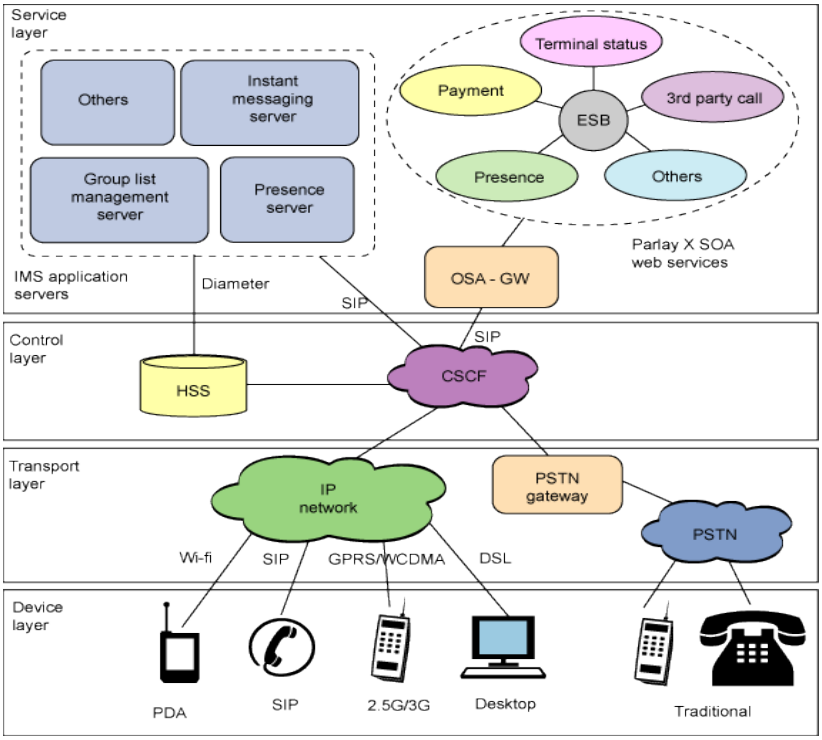


Figure 1-1: Multimedia Subsystem (IMS) CORE [2]

1.1.2. Vehicular Ad-hoc Networks (VANETs)

VANET is the technology of building a robust Ad-hoc network between mobile vehicles and each other, besides, between mobile vehicles and roadside units. As shown in Figure 1-2, there are two types of nodes in VANETs; mobile nodes as On Board Units (OBUs) and static nodes as Road Side Units (RSUs). An OBU resembles the mobile network module and a central processing unit for on-board sensors and warning devices. The RSUs can be mounted in centralized locations such as intersections, parking lots or gas stations. They can play a significant role in many applications such as a gate to the Internet.

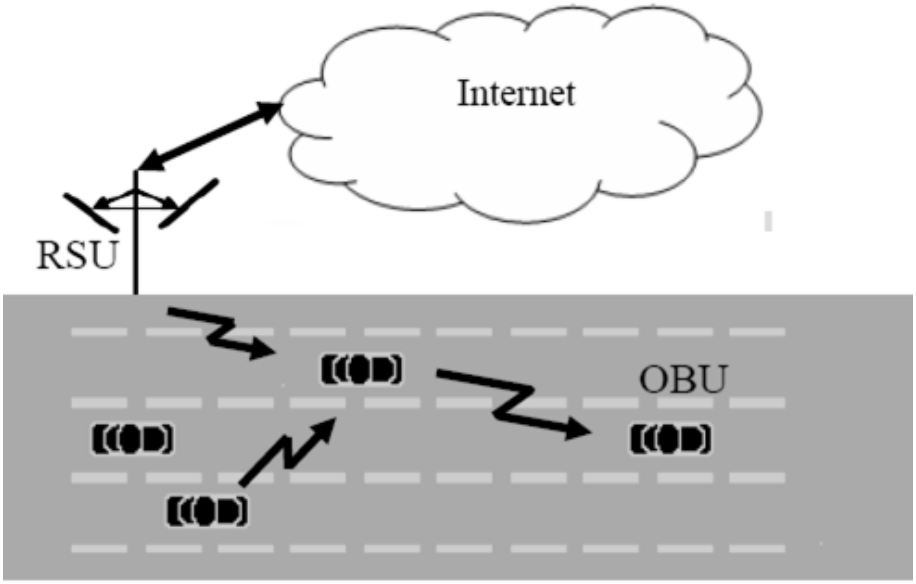


Figure 1-2: Node types in VANETs [8]

Referring to the introduction part, Vehicular Ad hoc Networks (VANET) is the subclass of Mobile Ad Hoc Networks (MANETs). It is one of the influencing areas for the improvement of Intelligent Transportation System (ITS) in order to provide safety and comfort to the road users. VANET assists vehicle drivers to communicate and to coordinate among themselves in order to avoid any critical situation through Vehicle to Vehicle communication e.g. road side accidents, traffic jams, speed control, free passage of emergency vehicles and unseen obstacles etc. Besides safety applications VANET also provide comfort applications to the road users. For example, weather information, mobile e-commerce, internet access and other multimedia

applications [9]. The most well known applications include, “Advance Driver Assistance Systems (ADASE2), Crash Avoidance Matrices Partnership (CAMP), CARTALK2000 and Fleet Net” that were developed under collaboration of various governments and major car manufacturers [9].

1.1.3. Wireless Sensor Networks (WSNs)

A sensor network is generally made up by a large number of sensor nodes deployed with or without a pre-established structured in a specific area in order to monitor and to collect information from the environment [10]. Since they do not necessarily rely on any pre-established infrastructure, sensor networks can be deployed randomly in inaccessible terrains.

Two basic entities are found in a sensor network: a sink and sensor nodes. The sink, generally centralized, gathers the information sent by the sensors; it also sends control information towards the nodes and acts as an interface to the user. Each sensor node is composed by, at least, four main components: A transmission or transceiver unit that provides network capabilities; A sensing unit that collects the information from the environment; A processing unit, which manages the procedures to collaborate with other nodes and has a small storage unit; And a power source that gives energy to the node and that is usually in the form of a battery. A wide range of applications has been enabled by sensor networks but two main categories can be established: monitoring and tracking. Monitoring applications include healthcare, environmental monitoring, power monitoring and traffic control among others. While tracking serves to object surveillance purposes such as: animal, human or vehicle tracking.

1.2. Aspects of the problem

Integrating the sensing capabilities of a WSN in the IP Multimedia Subsystem and giving the access to the IMS through a Vehicular Ad-hoc Network connectivity will open the door to a wide range of new multimedia services. Basically it allows the provisioning of new services and an optimized and efficient use of data.

On one hand, optimization and efficiency could be achieved once the information collected by the WSN reaches both IMS as well as VANETs. To further explain this case we could

analyze a scenario for integrating the sensing capabilities of a WSN in the IP Multimedia Subsystem and giving the access to the IMS through a Vehicular Ad-hoc Network connectivity. Information collected by the sensors is gathered by wireless nodes that a vehicle owner can use.

To further explain this case we could analyze the following scenario. It becomes important to support stable, high Internet access and multimedia services within VANET to obtain the powerful new generation intelligent vehicular networks and applications. Gateway here is known as the portal to these Internet services. Traditionally, gateways are facilitated between the given architectures. Due to the high speed of the vehicles, the vehicles quickly move into and out of the communication range of the gateway. Thus, it is difficult to stably access the Internet and maintaining the multimedia services as well via the specified gateway.

On the other hand, provisioning of new services or integrations could be developed since the data will already be available to be transmitted to the intended requester. Proactive actions exposed in [11] include: gateway discovery and selection that could be selected by the vehicles when integrating with the IMS and WSN as well. This is possible thanks to the standardization, unification and integration of services proposed by the gateway.

To successfully enable the above benefits, some minimum requirements need to be accomplished from the integration's point of view. Independence is one of them; the integration should require minimum changes in the three technologies involved. Additionally efficient translation needs to be done to merge the contextual information from IMS / WSN into VANET. Furthermore, to proactively react and optimize the use of information, the integration should consider storage and processing capabilities. And finally, since mobility is considered in WSN, VANETs, and in IMS users, the integration needs to allow nodes to join and leave the networks, thus scalability and fault-tolerance are required. Also another important aspect is to consider the quality of services (QoS) while transmission the multimedia services.

Several solutions have been proposed to bring information from sensor networks to other networks or technologies like IMS and VANETs. A common approach to the integration often used in the literature adds a centralized entity or gateway that connects all the technologies together [12, 13]. These models present some drawbacks mainly because scalability, fault tolerance, quality of service, and lightweight communication mechanisms are not supported or

not specified. Also they purpose the architecture is presented in a centralized data storage which can lead to single point of failure within the system.

The criteria of selection the gateway also is an important aspect that must be taken into the consideration, they did not mention the criteria for the selection of such gateway found.

1.3. Research Objectives

Our main goal is to design and build a gateway that will integrate IMS, VANET, and WSN, enabling multimedia dissemination between the IMS and interested users in the VANET world. More precisely, this project has the following goals:

1. Define a set of application-scenarios for the integration of VANET into traditional IMS and WSN to have the multimedia services enabled within the system.
2. Propose an architecture for the integration of the three technologies.
3. Develop a prototype of the gateway as a proof of concept.
4. Validate the proposed architecture by evaluating and comparing the output results.

1.4. Outline

The rest of this dissertation is organized as follows. Chapter II presents the state of the art regarding the integration of WSN with the Internet services and, more especially, with IMS. Also it contains the primary trials for achieving such integration between the VANETs and the IMS itself. In chapter III the proposed architecture is exposed in details. Chapter IV is devoted to the performance analysis of the proposed gateway and also simulating the interaction between the three different networks within the system. Finally, we conduct the conclusion and the future works.

CHAPTER 2 STATE OF THE ART

This chapter presents the state of art from the different technologies and integrations conceived in this project. Firstly, will further explain and describe the IMS architecture, after that we will explain in more details the VANETs and show some of the standards that have been proposed so far. Also we will pay attention for the wireless sensor networks and show some of its properties. The third section shows the scenario that is the base to analyze and define the requirements for the system. Then a brief survey on the integrations that have been done between the three technologies specifically connecting the VANETs with the IMS to gain the advantages of the multimedia services will be depicted. Later on, the benefits from integrating IMS, WSN, and VANETs are presented along with the proposals that have been done in the literature. Finally, benefits and weaknesses of the current overlay solutions for systems integration are presented.

2.1 IMS and the Presence Framework

As described before, IMS aims to integrate services and to provide QoS and a differential charging scheme. IMS architecture proposes several functional entities linked by standardized interfaces; meaning that two functional entities could be implemented in a single physical node.

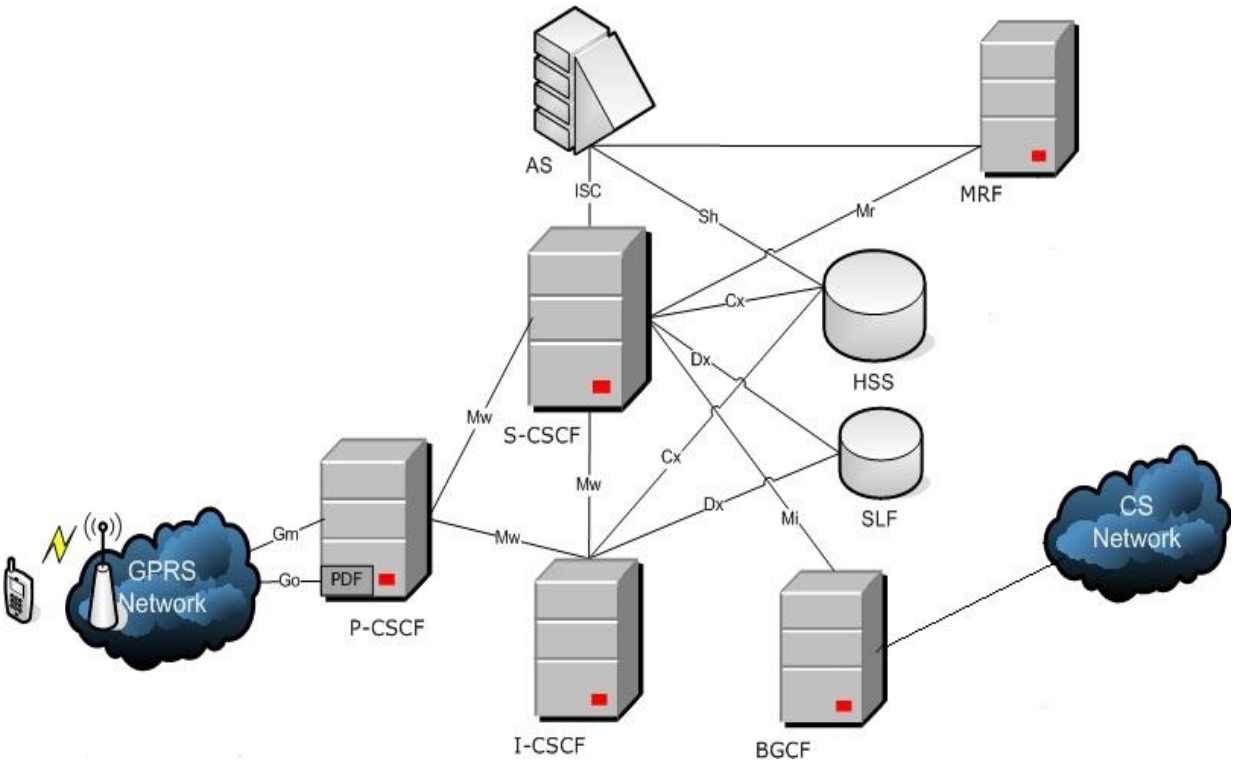


Figure 2-1: IMS simplified architecture overview [14]

Requirements

Referring to [15], IMS was conceived firstly to make more attractive the mobile Internet to users. So, the 3GPP Release 5 defines IMS as an architectural framework for delivering IP multimedia services to end-users. So the following requirements were needed:

- Support for establishing IP Multimedia Sessions, which means to support multimedia sessions over packet-switched networks.
- A mechanism to provide QoS.
- Support for interworking with other networks. Users must be able to reach other users regardless of what type of terminals they have or where they live.
- Support for roaming.
- Support for operators' control: from methods to provide the operator with service control to charging mechanisms.
- Support for secure communication. Security is a must nowadays. IMS takes it into account by including its own authentication and authorization mechanisms among other procedures.

The following releases and other groups' works added new requirements to support the so-called access-independence:

the fact that IMS provides support for different access networks (see [16]).

Entities

IMS is based on a number of entities which communicate among themselves. The most relevant ones are going to be described in the following sections.

Databases

The HSS (Home Subscriber Server) and the SLF (Subscription Locator Function) are the two databases in the IMS architecture. The first one is the main data repository for all subscribers and service-related data. It contains the user identities, registration information, access parameters and service triggering information.

Depending on the number of subscribers, a network can contain more than one HSS. If it is the case, then the second database is needed: SLF. It permits other entities to know which is the HSS database that contains all the information about the user they need. So it simple maps user's addresses to HSSs.

CSCFs

CSCF stands for Call Session Control Function and it is an essential node in the IMS architecture. In fact it is not a node, but 4 differentiated entities: P-CSCF, S-CSCF, I-CSCF, E-CSCF (the last one is out of the scope of this project because it is related to emergency scenarios).

1. *Proxy Call Session Control Function (P-CSCF):*

   It is the first contact point for users within the IMS. This means that all SIP signaling from the UE will be sent directly to the P-CSCF and in the other way around.

   It has 4 tasks assigned: SIP compression, IPSec security association, interaction with Policy and Charging Rules and emergency session detection.

2. *Serving Call Session Control Function (S-CSCF):*

   It is the central point of the IMS framework and it implements a SIP Server plus a SIP Registrar. All SIP signaling an IMS terminal sends and receives passes through the

allocated S-CSCF, which inspects every SIP message and determines if it has to be sent to another entity towards the final destination. Furthermore, it is also responsible for handling registration processes.

3. *Interrogating Call Session Control Function (I-CSCF):*

   It is the contact point within an operator's network, located at the edge of an administrative domain. It has assigned three main tasks: obtaining the name of the next hop, assigning an S CSCF based on received capabilities from the HSS and routing incoming requests.
   The combination of the CSCFs and the HSS is also known as the IMS Core.

   AS
   An Application Server (AS) is an entity which provides value-added multimedia services. Its functions are to process and impact an incoming SIP session, generate SIP requests and send accounting information to the charging functions.

   There are three types of Application Servers: SIP AS (native), OSA-SCS (interface to the OSA framework) and IM-SSF (CAMEL services).

   MRF
   MRF stands for Media Resource Function, it permits the home network to have a media source to play announcements, mix media streams, transcode and do media analysis.

   BGCF
   The first entity, the Breakout Gateway Control Function (BGCF), is a SIP server which includes routing functionality based on telephone numbers; while the second one provides an interface towards a circuit-switched network, which permits that IMS terminals can make and receive calls to/from any circuit switched network.

Presence Framework

The presence service allows a user to be informed about the reachability, availability, and willingness of communication of another user. It can provide an extensive amount of information about a person to a set of interested users. Even more, services are able to read and analyze this information to provide further services.

The presence framework defines four roles:

- Presence Entity – "Presentity". It refers to the entity providing presence information (e.g. status, capabilities and location). It has several Presence User Agents (PUAs) which provide this information to the Presence Service. Each PUA can collect different pieces of information.
- Presence Agent (PA) - It gathers information sent by the PUAs and obtains an idea of the user's presence.
- Presence Server - It is a functional entity that acts as either a PA, as proxy server for SUBSCRIBE requests or as both. In IMS, this entity is represented as an Application Server that acts as a PA.
- Watcher - It refers the user that requests presence information from a presentity.

It is built on top of the SIP event notification framework; which is based in SUBSCRIBE/NOTIFY requests. A watcher subscribes to receive information from a presentity for a period of time or for requesting some specific information. The presentity's PA will send the information to the watchers using a SIP NOTIFY request. Presence information is sent in the body of the messages and it is a XML document called PIDF. The PIDF carries the semantics of presence information between presence entities or roles. It is protocol independent and highly flexible; in fact some extensions have already been proposed to overcome some limitations.

Presence service is divided in three processes. The first one is the publication process where presentity's PUAs send PIDF documents in a SIP PUBLISH message. IMS CSCFs forward the request to the Application Server that represents the Presence Server; which finally replies with an OK. The second process consists in the subscription of watchers. Through SIP SUBSCRIBE transaction, watchers request to receive information from a presentity, watchers can be users or even other services. Once new information reaches the Presence Server a SIP NOTIFY is sent to subscribed users and services that can exploit the information [2].

2.2 Vehicular Ad-hoc Networks (VANETs)

The basic concept of VANET is straightforward: take the widely adopted and inexpensive wireless local area network (WLAN) technology that connects notebook computers to each other and the Internet, and, with a few tweaks, install it on vehicles. Of course, if it were truly that straightforward, the active VANET research community would likely have never formed and this thesis would have never been written. If vehicles can directly communicate with each other and with infrastructure, an entirely new paradigm for vehicle safety applications can be created. Even other non-safety applications can greatly enhance road and vehicle efficiency. Second, new challenges are created by high vehicle speeds and highly dynamic operating environments. Third, new requirements, required by new safety-of-life applications, include new expectations for high packet delivery rates and low packet latency. Further, customer acceptance and governmental oversight bring very high expectations of privacy and security.

Even today, vehicles generate and analyze large amounts of data, although typically this data is self-contained within a single vehicle and with a VANET, the horizon of awareness for the vehicle or driver drastically increases. Communication in VANETs can be either done directly between vehicles as one-hop communication, or vehicles can retransmit messages, thereby enabling the so called multi-hop communication. In order to increase coverage or robustness of communication, relays at the road side units (RSU) can be deployed. Road side infrastructure can also be used as a gateway to the Internet and, thus, data and context information can be collected, stored and processed somewhere. It warrants repeating that the interest in vehicular inter-networks is strongly motivated by the wealth of applications that could be enabled. First of all, active safety applications, i.e., accident prevention applications, would benefit from this most direct form of communication. Second, by collecting traffic status data from a wider area, traffic flow could be improved, travel times could be reduced as well as emissions from the vehicles. As it was concisely stated as the tenet of the Intelligent Transportation System World Congress in 2008: save time, save lives. The application classes Safety and Efficiency can be used to classify applications based on their primary purpose (Cfr. chapter 4, "Data dissemination survey"). However, the aspects of safety and efficiency cannot be seen as completely disjoint sets of features [17].

Obviously, vehicle crashes can lead to traffic jams. A message reporting an accident can be seen as a safety message from the perspective of near-by vehicles. The same message can be seen by further-away vehicles as an input to calculate an alternative route within a transport efficiency application

While being conceptually straightforward, design and deployment of VANET is a technically and economically challenging endeavor. As described in the following part, key technical challenges include the following issues [17]:

- Inherent characteristics of the radio channel

  VANET present scenarios with unfavorable characteristics for developing wireless communications, i.e., multiple reflecting objects able to degrade the strength and quality of the received signal. Additionally, owing to the mobility of the surrounding objects and/or the sender and receiver themselves, fading effects have to be taken into account.

- Lack of an online centralized management and coordination entity

  The fair and efficient use of the available bandwidth of the wireless channel is a hard task in a totally decentralized and self-organizing network. The lack of an entity able to synchronize and manage the transmission events of the different nodes might result in a less efficient usage of the channel and in a large number of packet collisions.

- High mobility, scalability requirements, and the wide variety of environmental conditions

  The challenges of a decentralized self-organizing network are particularly stressed by the high speeds that nodes in VANET can experience. Their high mobility presents a challenge to most iterative optimization algorithms aimed at making better use of the channel bandwidth or the use of predefined routes to forward information.

- Security and privacy

  There is a challenge in balancing security and privacy needs. On the one hand, the receivers want to make sure that they can trust the source of information. On the other hand, the availability of such trust might contradict the privacy requirements of a sender.

- Standardization versus flexibility

  Without any doubt, there is a need for standardizing communications to allow VANET to work across the various makes and brands of original equipment manufacturers (OEMs). Yet, it is likely that OEMs will want to create some product differentiation with their VANET assets. These goals are somewhat in tension.

From an application and socio-economic perspective, key challenges are as follows:

- Analyzing and quantifying the benefit of VANET for traffic safety and transport efficiency. So far, relatively little work has been done to assess the impact of VANET as a new source of information on driving behavior. Clearly, the associated challenge in addressing the issue of impact assessment is the modeling of the related human factor aspects.
- Analyzing and quantifying the cost/benefit relationship of VANET. Because of the lack of studies on the benefits of VANET, a cost/benefit analysis can hardly be done.
- Designing deployment strategies for this type of VANET that are not based on a single infrastructure and/or service provider. Owing to the network effect, there is the challenge of convincing early adopters to buy VANET equipment when they will rarely find a communication partner.
- Embedding VANET in intelligent transportation systems architectures. VANET will be a part of an intelligent transportation system where other elements are given by traffic-light control or variable message signs. Also public and

individual transportation have to be taken into account in a joint fashion. Therefore, truly cooperative systems need to be developed [17].

As can be seen from the above lists of technical, application, and socio-economic aspects, the field of vehicular application and inter-networking technologies is based on an interdisciplinary effort in the cross section of communication and networking, automotive electronics, road operation and management, and information and service provisioning. VANET can therefore be seen as a vital part of Intelligent Transportation Systems (ITS).

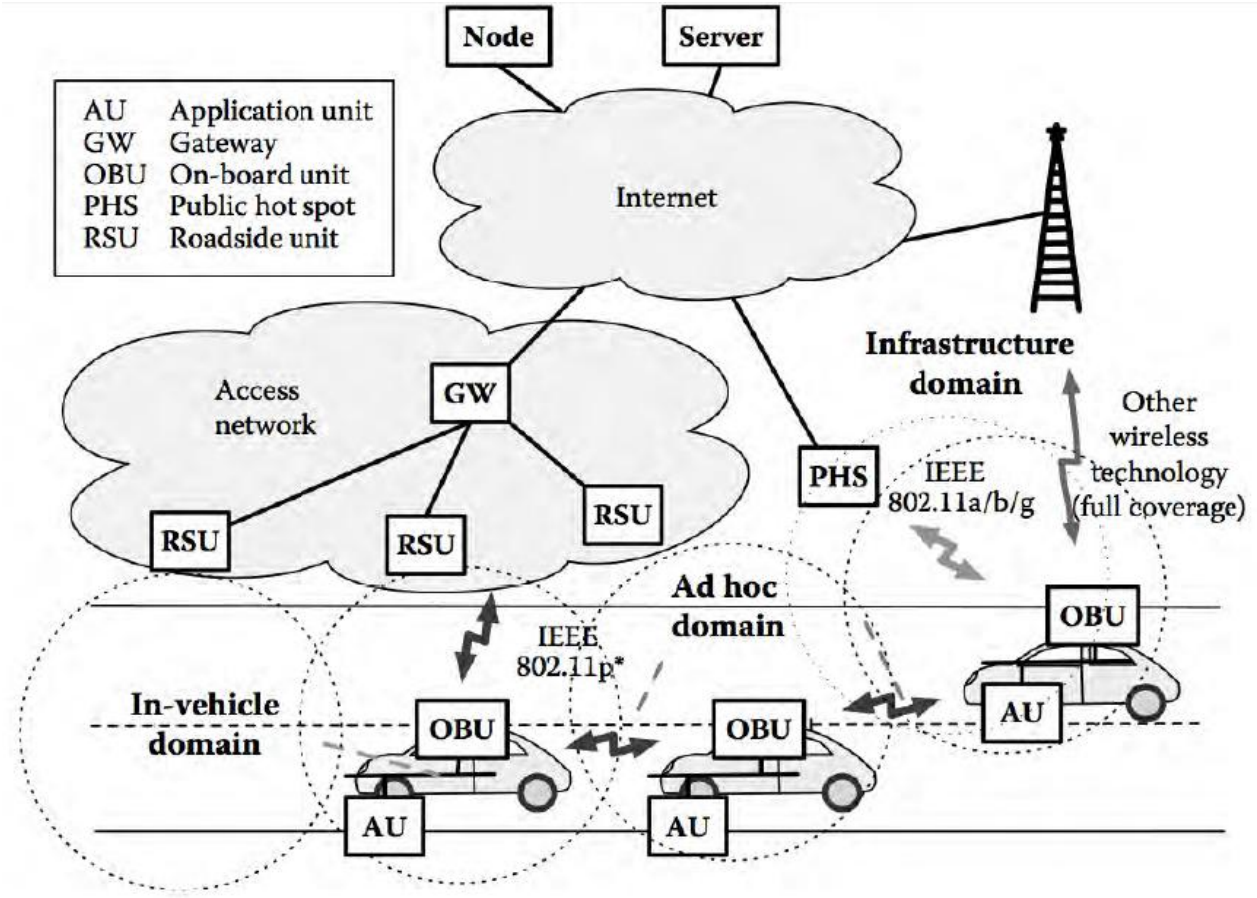


Figure 2-2: CAR-2-X communication scheme. A typical VANET scenario showing a CAR-2-X communication system and involved protocols of the IEEE 802 family. As shown in the figure, the CAR-2-X communication system consists of three domains: the in-vehicle domain, the ad hoc domain, and the infrastructure domain [17].

The concept of data dissemination is wide and meaningful, and within this work we generally refer to it as the process of spreading some amount of data over a distributed wireless network, which is a superset of a VANET.

Data exchanging on the roads is becoming more and more interesting, as the number of vehicles equipped with computing technologies and wireless communication devices is poised to increase dramatically. Communications between vehicles and within the same vehicle (inter-vehicle, or InV) is becoming a promising field of research and we are moving closer to the vision of intelligent transportation systems (ITS), which can enable a wide range of applications, such as collision avoidance, emergency message dissemination, dynamic route scheduling, real-time traffic condition monitoring and any kind of "infotainment" information spreading (i.e. movies, gaming and advertisement).

However, it is extremely important to consider several aspects when approaching to any kind of data transfer in a VANET, because nodes are not fixed but can move. Furthermore, in this scenario, other complications can easily arise because, unlike the well known mobile ad-hoc networks (MANETs), where nodes can freely move in a certain area, in VANETs, vehicles' movements are constrained by streets, traffic and specific rules [17].

The following are only some of the several issues which VANETs are affected by:

- High mobility:

  The environment in which vehicular networks operate is extremely dynamic, and includes extreme congurations: in highways, relative speed of up to 300 km/h may occur while density of nodes may be 1-2 vehicles per kilometer in low busy roads. Because of the relative movement of the vehicles, the connectivity among nodes could last only few seconds, and fail in unpredictable ways.

- Partitioned networks:

  Vehicular ad hoc networks will be frequently partitioned. The dynamic nature of traffic may result in large inter-vehicle gaps in sparsely populated scenarios, and in several isolated clusters of nodes. The degree to which the network is connected is highly dependent on two factors, such as the range of wireless links

and the fraction of participant vehicles, since only a fraction of vehicles on the road could be equipped with wireless interfaces. Maintaining end-to-end connectivity, packet routing, and reliable multi-hop information dissemination will become extremely challenging in such networks.

As it concerns specially the data transmission, in VANETs there are several additional issues to take into account:

- The signal fading, this becomes really fast due to the surrounding buildings;
- The strong interference and collision related to the high number of mobile transmitters (vehicles);
- The flapping links, caused by fading effect and vehicles' speed.

Furthermore, while traditional vehicular networks rely on specific infrastructures, such as road side traffic sensors reporting data to a central database, or cellular wireless communication between vehicles and a monitoring center, we want to focus our effort on completely decentralized data dissemination solutions, in order to avoid expensive infrastructures and increase the overall scalability of the system. In fact, how to exchange traffic information among vehicles in a scalable fashion is really an important problem to be solved in VANETs [17].

Now in the mean time for the multimedia services, multimedia services play an important role in the VANET world. Authors in reference [18] highlight the importance of the Live Multimedia Streaming (LMS) such world for their capability of providing comprehensive and user-friendly information. They focused on the fundamental challenges that face the VANET world while using minimal bandwidth resources, especially under highly dynamic topology of VANETs and the lossy nature of vehicular wireless communications.

They based their proposal on the *symbol-level network coding* (SLNC), which has been shown to be an effective approach to improve the efficiency of bandwidth utilization, by exploiting both wireless symbol level-diversity and the benefits of network coding. In their work, they introduced “CodePlay”, a new LMS scheme in VANETs that fully takes advantage of SLNC through a coordinated local push mechanism. Streaming contents are then actively disseminated from dedicated sources to interested vehicles via local coordination of destributively selected relays, each of which will ensure the smooth playback for vehicles

nearby. “CodePlay” is designed to simultaneously improve the performance of LMS service in terms of streaming rate, simulations show that CodePlay is potentially suitable for future LMS applications in VANET. The solution proposed considers the continuous availability of the multimedia services to the VANET systems.

2.3 Wireless Sensor Nodes (WSNs)

A wireless sensor network is generally composed by a large number of sensors that are deployed over an area to detect or sense a phenomenon. These sensor nodes are seen as sources of information, providing a rich set of contextual information such as: Spatial data (e.g. location), physiological data (e.g. heart rate, blood oxygen or blood pressure) or environmental data (e.g. temperature or soil properties). Additionally, they have the ability to compute simple tasks and to communicate with other nodes to transfer information or to perform other networking activities (e.g. routing).

The Sink or base station is the gateway or coordinator of the network. It collects information from sensor nodes and queries them if necessary. Besides the basic functionality, this functional entity is the entry point to the sensor network. Applications can create reports or analyze the data collected throughout the Sink. It has been generally conceived as a single and centralized entity in the WSN, however depending on the application several sinks and/or a decentralized architecture can be deployed.

Sensor networks have several features that made them unique. Among them, sensor nodes are densely deployed and prone to failures, the topology of the network varies in time due to mobility or failure factors, the nodes are limited in power, computational capacity and memory, and they may not have a unique global ID because they are densely deploy.

All these characteristics are being studied by researchers and several proposals for protocols, standards and algorithms that could fulfill them in a satisfying way have been made. Surveys in WSN [4, 10] have gathered this information and outline new challenges. Akyildiz et al. present in [4] a survey based on the communication architecture of wireless sensor networks and the proposals and challenges in the different layers (e.g. physical, data link, etc.). While Yick et al. [10] described the state of the art by dividing it in three lines of study: internal platform and operating system, communication protocol stack and network services. The following section

briefly describes some of the current standards for sensor networks, to further detail refer to [2, 10].

Standards

Wireless sensor networks requirements have been developed and design with low power consumption as a key characteristic. These requirements can change from different types of areas in sensor networks. Thus, several standards have been proposed. They include not only the functions but also the protocols necessary to interact with other nodes in the network. Some of the standards proposed for WSN are: IEEE 802.15.4, ZigBee, WirelessHART and IEEE 802.15.3.

IEEE 802.15.4 is a standard for low rate wireless personal area networks (LR-WPAN). LR-WPAN characteristics are ease of installation, reliable data transfer, short-range operation, extremely low cost battery life and a simple and flexible protocol. IEEE 802.15.4 allows the formation of either peer-to-peer or star network composed by the two types of devices defined by the standard; a full-function device (FFD) and a reduced-function device (RFD). The standard is designed for applications that require short range communication to maximize node lifetime, including residential, industrial and environmental monitoring, control and automation.

ZigBee is based on the IEEE 802.15.4 and defines the upper layers. It is designed for applications that require low data rate, long battery life, and secure networking. The standard defines three types of devices: ZigBee coordinator, ZigBee router and ZigBee end device. The ZigBee coordinator starts the network, can store information about it and can act as a bridge to other networks. ZigBee router links groups of devices and provides multi-hop communication. Finally, ZigBee end device limits its function to transmit information collected to the parent nodes; includes sensors, actuators or controllers. ZigBee has been proposed as standard for home automation, healthcare monitoring and industrial control.

WirelessHART as ZigBee is based in IEEE 802.15.4. It provides a communication protocol for process measurement, control and asset management applications. One of the main advantages is that it is compatible with existing devices, tools and systems. Their architecture is based on three main features: reliability, security and effective power management.

IEEE 802.15.3 is a physical and medium access control layers in high rate WPAN. It has been enhanced to address the specific needs of digital imaging and multimedia applications; audio and video.

The design of these platforms is tightly coupled with a range of applications, since energy efficiency, cost and other design challenges can dramatically change with the application requirements. Although these standards provide support for a wide range of applications, new applications could be easily be defined, thus, a more practical platform could be useful [2].

2.4 Use Cases and Requirements for the Integration

Multimedia services appear as a promising area of IMS thanks to their intrinsic features such as Internet provision and video / data streaming. IMS has been proposed to provide such services to users allowing real-time access to the information from the field.

Assisting the drivers in hazardous situations and decreasing road dangerous, has been extensively studied. Important processes like accident prevention and post-accident investigation take place to enhance and improve the drivers' awareness while getting in such dangerous situations. Festag, Hessler, Baldessari, Le, Zhang, and Westhoff in [19], for instance, propose a hybrid topology composed by road-side WSN – VANET that the vehicles are equipped with an On-board Unit (OBU) and two wireless network interfaces; namely IEEE 802.11p and IEEE 802.15.4 and also that the sensor data are stored in a distributed and redundant database in the sensor nodes. Moreover, it is difficult to design a common system architecture for both VANETs and WSN. For this reason, they proposed a hybrid system architecture that combines the best of the two worlds.

This proposal could be adapted to our case where the usage of IEEE 802.11p is applied in VANETs and then it can interact with WSN which in turn interacts with IMS using the proposed architecture in [2], as the presence service as entry point in the architecture ensures that the gateway is able to interact with the IMS and also is scalable.

Once the information is ready to be collected, this could be forwarded to the integration point with the VANETs. Depending on the role of each party within the environment, the integration point could publish different types of information. For example, the VANET user might request Internet access and video streaming from the IMS world. On the other hand, the

sensor nodes in the WSN monitor environmental data, store the collected data with the timestamp and sector information, and communicate the collected and processed data to passing vehicles via IEEE 802.15.4 ensuring its reliable collection.

WSN is a technology suitable to fit different requirements when monitoring critical and normal conditions. Sensors are deployed directly in the field with a network infrastructure that could run for years without man attendance [2]. In the sense of that, WSN aggregate the measured values and communicate their aggregated value to an approaching vehicle. The vehicle then generates messages and distributes it to all vehicles in a certain geographical region, potentially using wireless multi-hop communication [19].

Furthermore, [20] has proposed several proactive actions and scenarios that could be implemented in the future in IMS. Wireless Sensor Actuator Network (WSANs) for instance, could optimize some process. WSAN are sensor networks that integrate a new entity entitled actors that can perform appropriate actions upon the environment called actuation tasks [2].

Based on the previous scenario; some minimum requirements are defined to fulfill the benefits of the integration between the three technologies "IMS-VANETs-WSNs":

- *Independence.* The integration should be as independent as possible from both the IMS and VANET. This will ensure the minimal changes to the IMS and the VANET and in the integration point if different solutions want to be adapted.
- All services provided by the IMS "Internet access, video / data streaming provision, IP Multimedia services, etc." should be available to the VANET via the gateway.
- *Translation* from the information model used in each of the IMS, VANET, WSN and vice versa. Here we can say that the presence of the Middleware between each of them is an important factor that must be put into consideration.
- The presence of the appropriate provision of the Quality of Service (QoS) according to the function of each IMS and VANET, different parameters must be satisfied including reliability, availability, and priority, etc.
- Storage and processing capabilities to handle the requests from IMS as a way to support both synchronous and asynchronous mode to access its services.

- *Scalability and fault tolerance,* allowing the multiple nodes to join and leave the networks (IMS and mobile vehicles).
- *Determination of gateway location.* At the session setup, the VANET should be able to determine the location of the gateway; also in return, the IMS should be able to determine the gateway to send the requested services.

2.5 Integrating IMS and WSNs Architecture

In this section we are going to explain in details the integration methods between each of the IMS and WSNs as well as the WSNs with VANETs. Since most of the proposals found are dealing with these issues; we are going to extract the details and we will propose our architecture for integrating the three technologies together.

Referring to [2], IMS is a platform that has been selected as a key component of third generation (3G) networks. It is foreseen as a main component in service integration and provisioning. Moreover, it provides unified access to information, whose suppliers could include other networks like WSN.

A scalable framework on top of IMS [21] could facilitate the enhancement and development of more intelligent, invisible and autonomous applications that exploit WSN information (e.g. about people, places and objects). Strohbach et al. expose some of the benefits in building such a framework in IMS: message routing across administrative domains, access to heterogeneous networks, reusable core functionalities, extensive set of services and extended flexible session control. Furthermore, actual services such as the Presence Service can be extended to exploit WSN information.

Previously we described Presence Service. Under the basics of this framework, exploiting WSN information by extending and using it as an entry point in IMS becomes logical. Firstly, Presence already manages some rudimentary context information such as location and status and the PIDF defined is highly extensible to support other types of information. Furthermore, it allows not only users but also services (e.g. applications or other networks) to have access to sensor's information in real-time with a pre-established QoS. Thirdly and finally, already

existing services like SMS, MMS and IM can be used as they exist today and future integrations (e.g. with Actuator Networks) can be foreseen.

Integrating IMS and WSN

Integrating IMS and WSN is recently been advocated since its importance is been widely acknowledge as shown in the previous subsection. Gluhak and Schott in [13] present the e-SENSE architecture that integrates wireless sensor networks and IMS. The purpose of this architecture is to enable the delivery of context information from the sensors to the user in different application environments offering significant advances compared to ZigBee based systems.

Our focus is on the integration we will not explain the protocol stack of the e-SENSE system but in the integration done of this system into IMS. A new service called e-SENSE Service Enabler (SE) is added in IMS. This entity is in charge of providing sensor-based context information from several e-SENSE systems, thus processing and storing information. The entry point to integrate an e-SENSE system with the e-SENSE SE is the e-SENSE Gateway. This functional entity registers to IMS, buffers, schedules and prioritizes the data and enables the publish/subscribe services. However, when analyzed in the light of the requirements the solution is not suitable. Firstly, the proposed solution introduces a new entity in IMS that does not assure independence. Moreover, storage and processing are supported by this new entity and not by the gateway. Additionally, the adaptation of the information received by WSN into IMS is not explained. Scalability and fault tolerance are not considered and, since the gateway is a centralized unit without load-balancing or recovery mechanisms, we could assume that it is not supported.

Another work was exposed in [22] to integrate Personal Networks (PNs) and WSNs into IMS. Their objective is to enable the future creation of smart environments where different types of devices provide different services. The interconnection is done with a key component called the PN Gateway. It uses TinySIP since it is meant to work in any hardware with limited resources; TinySIP enables communication between a client on a traditional network and a sensor node. The PN Gateway has access to different technologies e.g. WLAN, Bluetooth and ZigBee. The gateway translates messages between TinySIP and SIP and it is directly connected to a SmartDust Enabler which routes SIP messages between IMS user agents and the different

PNs. Although this proposal offers integration between WSN and IMS, some of the requirements are not fully satisfied. It assures independency since minimal changes have to be done to assure that the WSN will locate the PN Gateway, it supports partial scalability, considers light weight communication mechanisms via TinySIP and somehow allows the publication of information. However, this architecture offers neither storage nor processing nor translation. It does not show how the different types of information and entities could be identified in IMS. Furthermore, this solution does not implement recovery mechanisms in case the PN gateway fails or is overloaded.

El Barachi et al. expose in [12] another architecture for the integration between the IMS and WSN. The integration is made by defining a generic gateway that offers several information management functions to acquire process, store and disseminate information and other support functions to enable a real internetworking. The gateway integrates both architectures by connecting with the WSN sink and acting in IMS as a Presence External Agent (PEA). The latter will actually publish the information captured by WSNs and send it to an extended Presence Server (capable of managing types of data and entities from WSN). The information is published using one of three methods defined: interval-based publication on regular-time intervals (e.g. every x seconds), event-based (certain events are detected) or trigger-based (information is published upon the direct request from the PS). Once information reaches the gateway it is processed and then stored. An event monitor is constantly analyzing the stored information to determine if it needs to be publish, additionally a trigger handler module is sensing to see if there is a request coming from IMS; if any of the above happens a publisher module formats the document into an extended-PIDF and send a SIP PUBLISH with the information.

In the light of stated requirements the WSN/IMS gateway meets most of them. First of all it is independent. The changes made to the PS are generic and are made to support the different types of data (i.e. spatial, physiological and environmental) and the types of entities (i.e. a user, a place and an object) that exist in WSN. Additionally, it stores, process, translates and publishes information.

However, it does not scale and it is not fault tolerant because it is centralized and no additional procedures have been defined for this purpose. The design will depend on one single node to pass the information to the P2P network, which could easily become a bottleneck.

Although this last proposal is not completely suitable some of the work that has been used is suitable to accomplish our objectives and will be considered further in the design process.

Specifically, they propose and demonstrate how the Presence Service could be used as entry point for WSNs. Moreover, the already extended the standard XML-based PIDF to support WSN types of information and to allow the distinction between user and non-user related information. Thanks to [2] all of these pieces of information are provided in details.

2.6 Integrating VANETs and WSNs Architecture

Reference [19] presents the hybrid system architecture that combines the best of the VANET and the WSN worlds. Their objective is to enable the creation of smart environments where different types of networks provide different services. The interconnection is done with a hybrid VANET-WSN gateway. The gateway collected all sensor information, maintains the collected, aggregated data in a local database, and injects to data into the VANET. It uses different components to enable such connection between the two worlds. It uses the tinyLUNARInterface (tLI) – LUNAR means "Leightweight Underlay Network Ad hoc Routing", tinyLUNAR offers reactive routing for WSNs based on the label switching mechanism, and nodes maintain simple and small forwarding tables, while the overhead for data packets is maintained only with small size. In this mechanism, the process of discovering the routes is done by flooding route requests.

Another supported feature which is the Geocast, which is a networking protocol using geographical positions for addressing and routing. The core protocol components of the Geocast are beaconing, location service, and forwarding. One of the main featured also presented in such architecture is the middleware. They defined the middleware to be used only with the VANET environment. They defined it as the main data repository in a vehicle, which is a dynamic representation of the vehicle's environment. It maintains the fused sensor data, information exchanged with other vehicles and static data (such as a digital map). They used this featured component to utilize the stored data by control algorithms for driver assistance and communication applications. They referred to a typical example of a VANET middleware in [23]. In addition to the data storage, the middleware has important tasks for cross-layer communication exchange, security and privacy. In the light of our requirements the

VANET/WSN gateway meets most of them. First of all it is independent. The changes made to the architecture are generic and are made to support the different types of data (i.e. spatial, and environmental) and the types of entities (i.e. a VENT user, a place and an object) that exist in both the WSN and VANET. Additionally, it supports storage, process and publishing information.

This last proposal is quiet suitable to accomplish our objectives and will be considered further in the design process. Specifically, they propose and demonstrate two types of architectures, the centralized and the distributed modules deployed in both the RSU and the OBU respectively.

2.7 Architecture for System Integration

Here we have shown that how integrations between each of the IMS and WSN, as well as WSN and VANET – they all considered a gateway to interconnect the different types of networks. However this is not the only way to do it. In the following section we will show the available methods that can be used to interconnect IMS with VANET technology and will select the most suitable for our involved project.

There are mainly four layers: Application, Middleware, Geocast / tLI, and IEEE 802.11p and IEEE 802.15.4 that have been studied and compared in detailed in [18]. Also another methods and layers have been considered in [2]. These layers and approaches will be explained in further details below.

As in [19], they proposed the usage of IEEE 802.15.4 in which communication over small distances and geographical areas can be achieved. Also they used the IEEE 802.11p and applied it in the VANET. They made useful use of its feature of sending data over medium distances and distributing the information in geographical regions via the multi-hop communication. The sensor nodes in the WSN monitor the environmental data, store the collected information with timestamp and geo-information (sectors), and communicate the data to passing vehicles via the IEEE 802.15.4. The storage is encrypted and distributed over multiple nodes in a persistent way. For communication among the sensor nodes, the WSN is randomly divided into clusters, where each cluster is managed by a cluster head. The sensor nodes transmit data to their cluster heads, which transmit the aggregated data to other cluster heads. Data from

the WSN are injected into the VANET by vehicles in the communication range of a sensor. The data transmission from a sensor to a vehicle can be periodic, solicited by the passing vehicle, or both. Once the vehicle has received the sensor data, it can distribute the information to relevant in a geographical region by the Geocast protocol.

The OBU of a vehicle plays an important role in the architecture since it acts as a gateway between the WSN and the VANET and decides about injection and forwarding of relevant sensor data. As an alternative to the WSN with a distributed data storage, a RSU acts as a gateway between the WSN and VANET. The RSU collects all sensor information, maintains the collected, aggregated data in a local database, and injects the data into the VANET.

A vehicle's OBU has a dual protocol stack. For communication in the VANET the OBU executes Geocast and IEEE 802.11p beneath the VANET middleware and application; where the middleware is the main data repository in a vehicle, which is a dynamic representation of the vehicle's environment. It maintains the fused sensor data, information exchanged with the other vehicles and static data (such as a digital map). The stored data are utilized by control algorithms for driver assistance and communication applications. The typical example of the VANET middleware is the local dynamic map. In addition to the data storage, the middleware has important tasks for cross-layer communication exchange, security and privacy. Moving to the other point, the applications implement the application protocol, typically based on SAE J2735 [24] for message encoding and TPEG for event encoding [25].

A sensor node executes IEEE 802.15.4 and tinyLUNAR, as radio and networking protocol, the middleware tinyPEDS and applications. For the defined protocol stacks, the tinyLUNAR offers a low-overhead, topology based, reactive routing for WSNs. Based on the label switching mechanism, nodes maintain simple and small forwarding tables, while the overhead for data packets is only one byte. Routes are discovered by flooding route requests, which can address nodes according to flexible criteria; e.g. node role, content, position, etc.

tinyPEDS is a distributed data collection and storing scheme with security enhancement for WSNs. It used concealed data aggregation techniques in order to minimize the size of the data transmitted in the network, thus increasing the system lifetime. tinyPEDS has further reliability and security enhancements, such as access control and sensor reading outliers detection.

And as in [2], they purpose a P2P overlay gateway for the integration of the mobile sink-based WSNs and IMS. They categorized the architecture of the gateway into five sets. The first set interacts with the IMS Presence Service, the second one interacts with the mobiles sinks. The third set handles the storage, the fourth one processes the information, and the fifth set enables the interaction between the four mentioned sets. Also they defined the roles of each component in the gateway. The Sink Entry Point (SEP) allows the connectivity with the mobile sinks; to interact with the IMS Presence Service they defined the Presence Service Entry Point. The presence of the processor within the model to transform, aggregate or compress information, storage acts as the peer that handles the storage process in the overlay.

Furthermore, they introduced three super peers; the first one is the Super Sink Entry Point (SSEP) which is in charge of the SEPs, also it interprets and analyzes information from the mobile WSN to determine whether it should be stored, processed or sent directly to the presence service. The second super peer is the Super Data Management (SDM) which manages the storage and processor peers. Finally, they defined the SPSE. It handles Presence Service Entry Point (PSE) peers and publishes the information received to the presence service in IMS.

2.8 Integrating VANETs and IMS “3G technologies”

Reference [26] presents the guidelines for integrating VANET with the 3G technologies. To enable such integrated architecture, vehicles are clustered according to their different metrics. A minimum number of adequate vehicles is selected to serve as a liaison between VANET and the 3G technologies. By considering the following scenario of two different tracks over a particular road (e.g., highway), with a track for each direction. The key components of the architecture are IEEE 802.11p-based VANET vehicles, a Universal Mobile Telecommunication System (UMTS) “Node B” and the main components for the UMTS core network. Communication over the VANET network is multi-hop and on a peer-to-peer basis. VANET is linked to UMTS via selected VANET mobile gateways using the Universal Terrestrial Radio Access Network (UTRAN) interface. As mentioned before the vehicles are equipped with both IEEE 802.11p and UMTS interfaces, lying within or moving into the 3G region, are called Gateway Candidates (GWCs). The rest of the vehicles, that do not lie in the 3G Active Region, are not equipped with the UMTS interface, or do not have their UTRAN interfaces enabled, and

that they are called Ordinary Vehicles (OVs). Among the gateway candidates, a minimum number of Cluster Heads (CHs) per direction are elected as optimal gateways (GWs) using different metrics. The number of the gateways is required to be minimum, so as to avoid the bottleneck at the UMTS Base Station (BST) and save UTRAN resources. Gateway candidates are grouped in clusters using dynamic clustering mechanisms, and only the selected gateways will have their 3G UTRAN interfaces activated. However, the IEEE 802.11p interface is enabled and activated on all the VANET vehicles. The Dynamic Clustering Operation is mentioned in details (**See** [26]).

Gateway management and selection is also performed in a dynamic manner using different metrics. It consists of three mechanisms, namely "multi-metric mobile gateway selection", "gateway handover" and "gateway discovery/advertisement" mechanisms. The gateway selection mechanism is used to select the minimum number of adequate gateways to optimally communicate with the backhaul UMTS network. It is based on the Simple Additive Weighting (SAW) technique using metrics such as the mobility speed of the Cluster Head (CH), its UMTS Received Signal Strength (RSS), and the stability of its link with the source vehicles. The first vehicular source broadcasts a Gateway Solicitation (GWSOL) message within the VANET, using the Time To Live value (TTLs). The performance of the overall architecture was evaluated using computer simulations and interesting results were obtained.

Now that we have described the possible methods to interconnect each of the three technologies, we could say that the proxy is a suitable solution for our requirements but it needs to overcome the fault-tolerance and scalability issues. Middleware offers a solution for fault-tolerance in the network by introducing the concept of translation between different formats; a logical layer on top of the protocol stack. Combining these two approaches could be a solution for our integration.

A gateway made up by considering the middleware and some specific specs seems like a possible solution that will help integrate the three technologies together.

CHAPTER 3 PROPOSED ARCHITECTURE

This chapter presents the architecture designed to integrate VANET, IMS and WSN together. Firstly, we could analyze the next scenarios to attain optimization and efficiency of such systems together.

Multimedia services in all the form of video stream, voice, Internet access, etc. within the VANET system have always been considered, a lot of efforts have been undertaken to try having these services in the VANET world and in improving their quality of service (QoS), and their availability.

Good to mention that vehicles are connected via continuous wireless communication with the road infrastructure on motorways, exchange data and information relevant for specific road segments and multimedia services to increase the overall road safety and enable co-operative traffic management as well as the allowance of the multimedia services to be accessible throughout the vehicle while moving. So we are dealing with the Vehicle to Infrastructure (V2I) communication.

Scenarios and Use Cases

Scenarios are the predominant basis for evaluating the applicability of the proposed architecture in the given environment. Two scenarios are being considered in our case.

The First Scenario:

Roads have always been dangerous, and a lot of efforts have been taken to improve their safety. Vehicles, education, road signs have been improved throughout generations. Nevertheless, dangers remain and with the rise of computer and wireless technologies, new solutions are available to assist the driver in hazardous situations and to decrease road dangers.

Vehicles are equipped with wireless devices, so that they can communicate with each other. The primary application of this technology is to let vehicles exchange about their current context. In detail, the information exchanged can be two types, (i) periodic exchange status messages among the vehicles in direct communication range and (ii) safety messages triggered

by a critical event and distributed in a geographical region. We foresee that WSN roadside islands will be installed in specific dangerous locations to support drivers with current road and weather condition. In this case, connection between VANETs and WSNs will be considered.

Typically, WSN technologies help where neither the vehicle's sensors nor the driver can detect the danger, e.g. very localized road condition, animal crossing the road our of the forest, etc. The roadside WSN islands significantly extend the sensing range of a vehicle. Hence, either the driver or the vehicle itself could initiate appropriate reactions according to the current environmental conditions with the overall aim to increase the driver's safety.

The scenario of a combined WSN and VANET architecture aims at the provisioning of two complementary services:

1) **Accident prevention**. When a car passes by a sensor network, it retrieves fresh environmental data collected by the roadside sensors. Data can include various physical quantities, such as temperature, humidity and light, and also detect moving obstacles (such as animals). The received information are processed in the vehicle's OBU and potentially displayed to the driver. Hence, wireless sensor nodes complement other sensors installed in a car (such as radar). However, wireless sensor nodes are external devices that in principle can measure road conditions data more accurately than an on-board sensor. In addition, the data of the wireless sensor node may include a set of data covering a period of quantities collected over a time-span and make the data more plausible.

   Once a vehicle has processed the sensor data, it may interpret the data as a dangerous situation and trigger a safety warning message. For this message, the vehicle determines a geographical region defined by a geometric shape and broadcasts the message to its neighbor vehicles. The communication system of the vehicle ensures that the data packets are reliably distributed to all vehicles located within a region. As a result, vehicles that receive the information are warned about dangerous spots ahead of time and can take appropriate countermeasures.

2) **Post-accident investigation**. In this use case, sensor nodes continuously measure and store the environmental data. These data include the collected quantities (e.g. temperature) and also event data, such as previously detected obstacles and vehicles. Storing these information over a long period may be of interest for a forensic team. In contrast to the accident prevention service, such a liability service will be limited to a well specified group of end users, e.g. insurance companies or the road portal. These authorized users can retrieve the sensor data from the roadside WSN island from (nearly) any time in the past for forensics purposes. Typical examples are retroactive discovery of accident causes and assessment of drivers' behavior with respect to the road conditions at the time of the accident.

The two complementary services described above pose various functional and performance-related requirements for the data communication and storage. A fundamental assumption is that a communication system for vehicle-to-roadside communication will only be rolled out if the costs for the roadside equipment, installation, and maintenance are minimal. This leads to a system architecture with extremely low cost autonomous sensor networks and without the deployment of dedicated roadside units. Since sensor nodes may disappear over time due to their restricted energy capabilities, both communication among sensors and data storage need to be distributed and redundantly organized. Likewise, sensor nodes' data transmission to approaching vehicles and dissemination of data for persistent storage require energy-efficient and the trustworthiness of the data begin communicated from the WSN to the vehicles. In addition, as the data stored for a relative long duration within the roadside WSN, they shall not be stored in plain-text. In turn, in order to minimize costs, software-based security solutions are preferred over costly hardware components or tamper-resistant modules on sensor nodes.

The Second Scenario:

Cellular networks are deployed at various locations on the road and that the IMS is connected to them to supply the vehicular users with the various forms of its multimedia services. As the vehicles have various and different speeds, we have to make sure of the arrival of such real time services to them. In this case, connection between VANETs and IMS will be considered.

In the real time applications, the data gathered become important and useful only if users can have access to it in the same time they are needed and anywhere they are requested. These two specifications can be fulfilled by the presence of the gateway, where the multimedia services are already available when the connection between the IMS and VANET is established.

Our aim is to combine them all together (VANETs, WSNs, IMS) to successfully enable the above benefits and scenarios, so some minimum requirements need to be accomplished from the integration's point of view. We will list these in order to satisfy our architecture and approach.

3.1 Architecture Design

General Purposes

REQ 1. The gateway shall be designed to be independent. The integration should be as independent as possible from both the IMS and VANET. This will ensure the minimal changes to the IMS and the VANET and in the integration point if different solutions want to be adapted.

REQ 2. All services provided by the IMS "Internet access, video / data streaming provision, IP Multimedia services, etc." should be available to the VANET via the gateway.

REQ 3. Determination of gateway location. At the session setup, the VANET should be able to determine the location of the gateway; also in return, the IMS should be able to determine the gateway to send the requested services.

REQ 4. The gateway shall be designed hiding the implementation details to applications. However, it should provide feedback about what is happening on the network, avoiding unnecessary details.

REQ 5. When the maximum time set for a message between WSN and VANET expires, the gateway shall select an alternative way to send the messages to the intended destination.

REQ 6. The presence of the appropriate Quality of Service (QoS) according to the function of each IMS and VANET, different parameters must be satisfied including reliability, availability, and priority, etc.

REQ 7. The gateway inherits the main characteristics of the peer to peer system architectures: scalability, fault tolerance and the resilience as well.

Extensible

REQ 8. The gateway should be designed to easily adaptive to the required functionalities.

End-to-End delay

REQ 1. Information transmitted shall be delay-constrained. It means that the total time a message takes to reach a target shall not exceed a time-constraint.

REQ 2. When the maximum time set for a message expires, the gateway shall select an alternative way to send the messages to the intended destination.

The goal of the design is to accomplish all the requirements exposed in chapter III. We admit that the gateway is a simple and straightforward solution to interconnect two or more technologies together [2].

As mentioned in the previous requirements, the gateway inherits the main characteristics of the peer to peer systems which are a part of the required features. In addition to that, storage capabilities have already been proven to be feasible in the VANET architecture ***in case of*** dealing with the data collected from the sensor nodes ***only*** [27].

By using the vehicles acting as mobile nodes in the VANET world and the sensors acting as collectors in the WSN and the presence of the user agents in the IMS. Requirements can be satisfied and a solution can be implemented with the existing infrastructure to build out the gateway.

We propose a gateway to fulfill the defined requirements and needs, and we design the solution from this decision. Once the architecture is presented; a full detailed analysis on how the requirements are met by the proposal will be done in chapter IV.

3.1.1. Assumptions

Our idea is to provide a gateway that will interconnect VANET, IMS and WSN all together; however there are some assumptions that need to be done at this point as a way to present achievable at this stage in the design process.

Concerning the entire system, we assume that all entities of the system (i.e. WSN, VANET, gateway, etc.) are configured to know each other. However, during the design of the architecture we try to ensure that minimal changes and adjustments will have to be made in case the format changes.

Architecture Principles

The principles to our architecture are based on the gateway which will be the central point of the integration. Figure 3-1 depicts the general architecture, including the overlay gateway which is divided into six groups. Each one of these groups is in charge of providing one or more services and has one leader node.

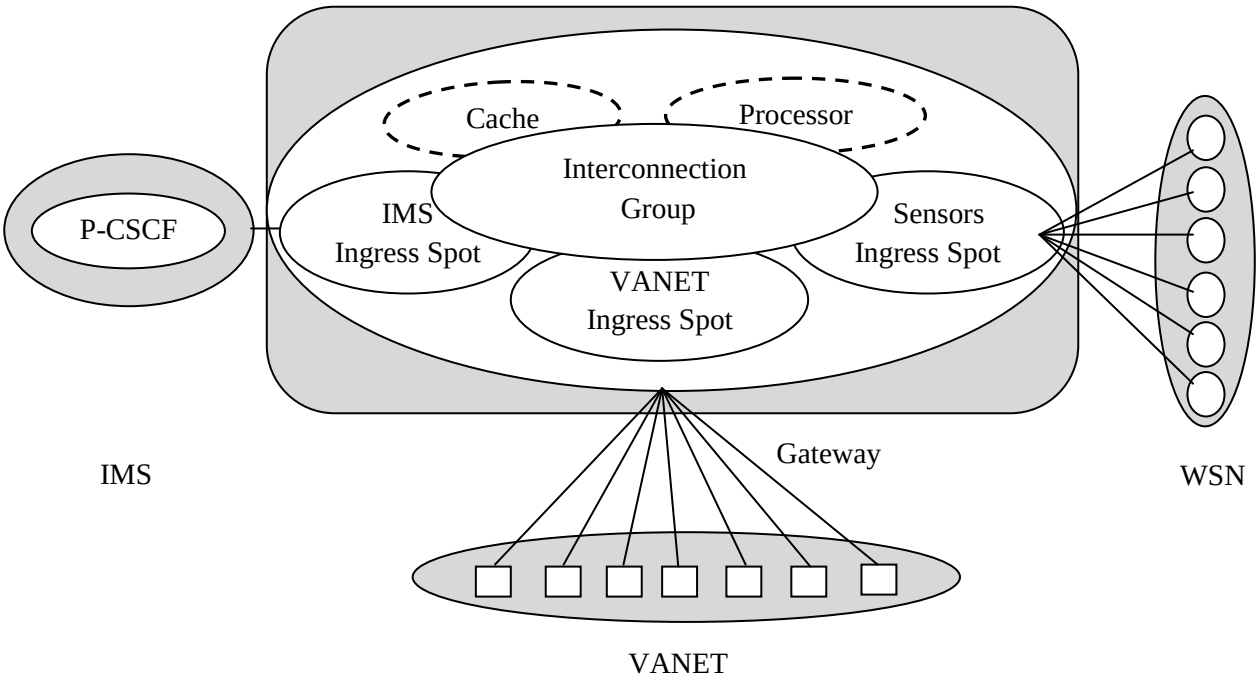


Figure 3-1: General architecture

(The dotted line in the figure indicates that these two criteria can be deployed within the gateway or to not be considered according to the needs/importance of the information collected by the sensor nodes).

The first group is in charge of the interaction with the VANET world, meaning that it manages all the integration with this technology. This group is the entry point to the VANET system; it is in charge of publishing the information and to process the requests of the VANET towards the IMS and WSN.

The second group is similar to the previous one, but offers connectivity to the IMS world. In this group, the interaction with the IMS Presence Service is done, meaning that it manages all the integration with this technology. This group is the entry point to the IMS; it is responsible for connecting with SIP Servers (CSCFs) in the IMS, and to publish the information to the Presence Server and to process requests from IMS towards both the VANET and the sensor networks (SN).

The third group is responsible for the interaction of the WSN with the whole system. This group interacts and connects the sensor nodes with the gateway. Moreover, it receives the

information sent by the nodes and is able to send requests to the sensors when requests for specific information arrived.

The fourth and fifth groups deal with handling data management processes when the information from sensor nodes is ready [these groups are working only while dealing with VANETs and WSNs]. They are in charge of caching and processing information respectively. Caching information allows VANET users to have an instant copy of the small data segment collected by the sensors (referring to the first scenario introduced). Processing of information is done through predefined parameters; it consists for example in the aggregation of the data collected from sensors in a specific area.

Finally, there is one last group whose role is to interconnect services in the gateway. It manages all interactions between the other groups mentioned, allowing the communication between them. This group is formed by the leader node from each group.

3.1.2. Architecture topology and interactions

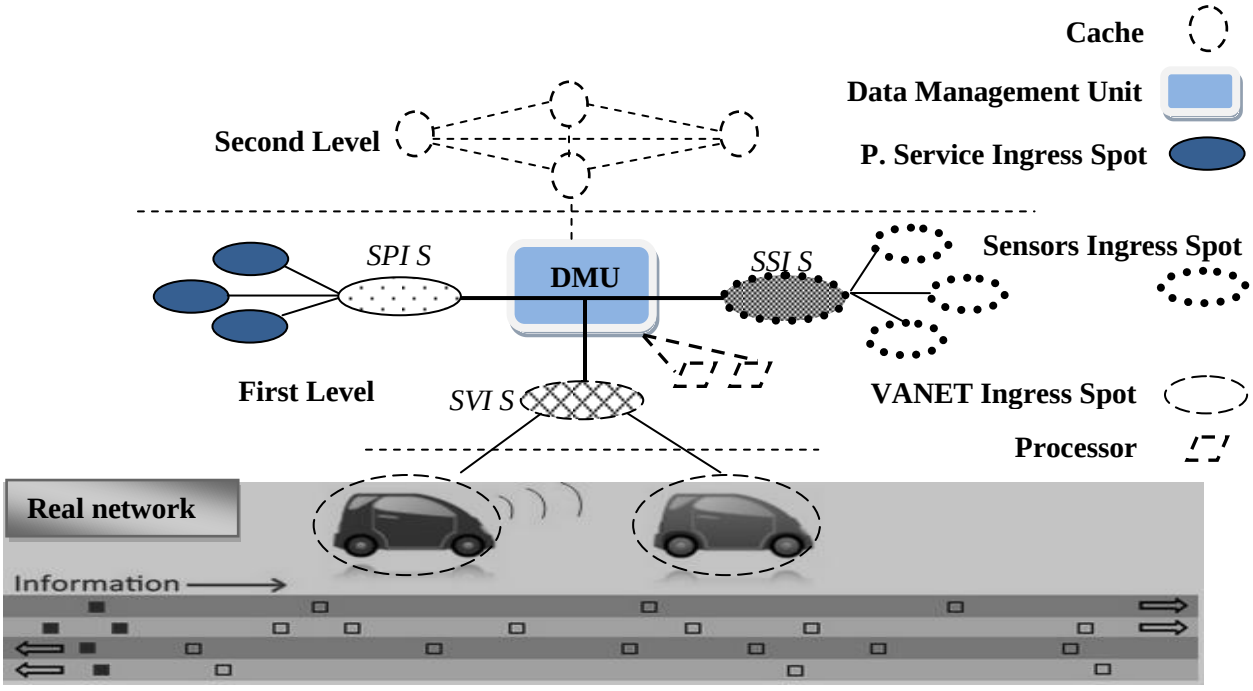


Figure 3-2: Overlay Gateway

Figure 3-2 shows the topology that has been designed based on the key principles of the architecture. This topology is depicted as a two-tire topology where the first level provides all other services, and the second level is exclusively for caching information from sensor nodes. Each of the six groups defined in the previous section had two or more participant roles interacting to accomplish their functions; we will explain in more details the following subsections. Once the roles and the interactions are described, the topology will be further analyzed and presented.

Factors

When describing the principles of our design and architecture, we describe the six groups that compose it. Each one has a leading role that enables the services as was previously defined. A functionality is an identification that determines the services and the logic provided by each node in the overlay.

The first group is in charge of providing an integration point with the IMS. This group has two participant roles; the Presence Service Ingress Spot (PSIS) and the Super Presence Service Ingress Spot (SPSIS). The PSIS acts strictly as an entry point, it provides connection to IMS CSCF's servers, enabling the communication between the IMS and the gateway. This role interacts exclusively with the SPSIS. The SPSIS is the leader of the group; but it also has the ability to provide the connection with the IMS. Moreover, it sends requests and responses to and from IMS and other groups in the gateway.

The second group is the one in charge of the connection with the sensor network; made exclusively through the sensor nodes. This group includes the Sensor Ingress Spot (SIS) and the Super Sensor Ingress Spot (SSIS). SIS allows the connectivity with the sensor nodes, enabling the transference of messages between networks. The SSIS is the leader node of this group. It is in charge of SIS nodes, interprets and analyzes information received from the WSN and is capable to determine whether this data should be stored, processed or sent directly to the VANET users. As the SPSIS, this leader also has the ability to play the basic role, connecting the sensor nodes with the gateway. Although and SIS could play the role, at a given time only one can do it.

The third group is in charge of the connection with the VANET world; made through the moving vehicles on the road. This group includes the VANET Ingress Spot (VIS) and the Super

VANET Ingress Spot (SVIS). SVIS allows connectivity with the moving vehicles; enabling the transference of the messages between the VANET users. The SVIS is the leader node of this group. It is in charge of VIS nodes.

The fourth and the fifth groups, as defined in the architecture, handle caching and processing of information, they share the same leader. The leader (i.e. DMU) manages the group and the interaction between its groups and the rest of the groups in the gateway.

The processing group is composed by the Processing role that can transform, aggregate or compress information and the DMU; as mentioned before, the DMU is also the leader.

The final group is composed of the Super nodes (i.e. SVIS, SPIS, and SSIS) from the seven groups previously stated. They collaborate with each other to offer their services to the overlay and allow access to the other groups found within the architecture.

Interactions

In this part, the interactions that are made inside the group members will be highlighted. The communication and message exchange depends in the source and the destination of the messages.

The first group is the IMS Ingress Spot that offers connection with the IMS. As explained before, the SPIS connects all the PISs to the gateway. When a message is received by the PIS from the IMS, it is forwarded by the leader to the SPIS. Equally, when a message is received from another entity in the leader's group, the SPIS sends the message to one of the PISs so it can be forwarded to the IMS.

The second group is the Sensor Ingress Spot group that connects with the sensors WSN. The SSIS receives data from and sends requests to the WSN through the SISs. Each SIS is connected to one or more sensors. Once information is collected from the sensors WSN, the SIP receiving the information, forwards it to the SSIS. It decides where the data should go (whether to store, to process or to publish), thus it sends a request to one of the other nodes in the leader group. Additionally, when a request is sent to the group to retrieve information from the WSN, the SSIS will forward the request to all SISs.

The third group is the VANET Ingress Spot that gives the connection with the VANET world entities. As mentioned, the SVIS connects all the vehicles to the gateway. When a message is ready to be received; it is transferred by the SVIS to the VIS.

The fourth and fifth groups are related to caching and processing of information gathered from the sensor nodes. Firstly, the DMU receives requests from the SPIS, SSIS and the SVIS. If these requests are related to some basic information about the environment; these requests will be resent to the caching nodes in the second level of the architecture which will later send a response with the information to the DMU if a copy is found there. Secondly, if processing of data is demanded, the DMU forwards the request to one of the processor nodes that will perform the action and return the transformed data.

Finally, interactions inside the Interconnection Group are explained. We mentioned before that the SSIS decides whether the information should be stored, processed or published. If storage or processing needs to be done, then the SSIS sends a message to the DMU and if it needs to be published then it is routed to the SVIS.

The SPIS handles publication towards the IMS Ingress Spot group, thus it receives the requests and disseminate it according to the definitions inside the group. When information is requested from IMS and reaches the SPIS, it forwards the request to the DMU.

Finally, there is the DMU which responds to requester nodes asking to store, to process or to retrieve any information. Storage and process are handled inside their respective groups; however retrieval of information is first analyzed by the node. If history data is required or if the time of the stored data is below the demanded time threshold, information is retrieved from the Storage group; otherwise the DMU sends the request to the SSIS so the retrieval is made directly from the sensor nodes WSN.

Global Architecture

Now that the gateway architecture, its roles and its topology have been explained in a detailed way, we will show how we connect and integrate the gateway with the vehicles, IMS, and the sensor nodes. At this point, the topology and the basic of the integration will be described, the process on how they are connected to each other is explained below in the next subsection. The following figure [figure 3-3] shows the gateway connected with the vehicles and in this case using the traditional IMS architecture.

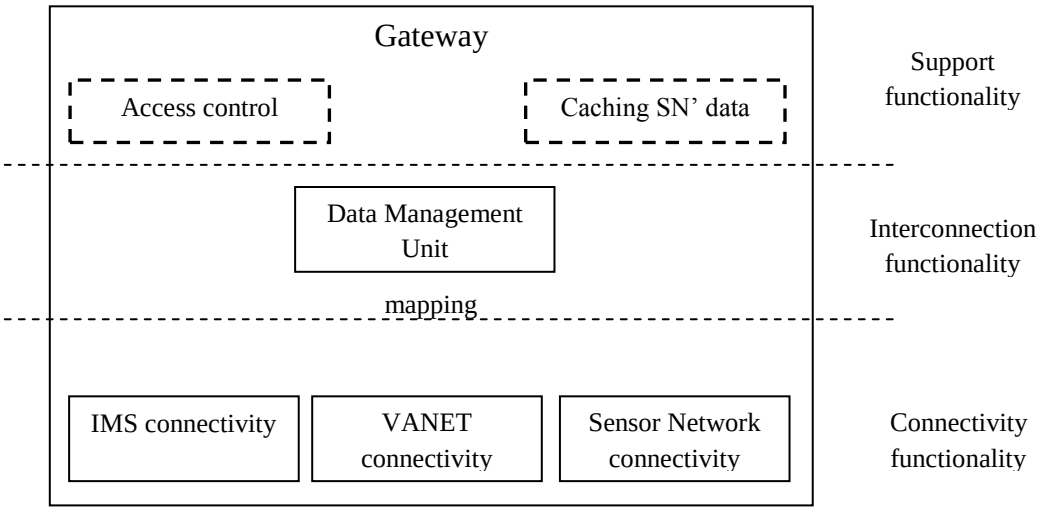


Figure 3-3. Global topology of the gateway

3.2 Overlay Rules

This section presents the rules governing the architecture described in the above section. This includes the information models used inside the gateway, the protocol used by nodes to communicate with each other and the description to different procedures supported by the overlay.

3.2.1 Protocols

Protocols allow nodes in the network to communicate with each other in a standard language. A protocol defines the rules of the syntax, semantics and the synchronization within the network. It not only defines the messages that can be exchanged between nodes but also their structure and the rules for exchanging them (e.g. the answer expected).

To become a suitable protocol for the gateway, it should accomplish the requirements defined before. Basically the protocol needs to:

- Enable the self-organizing and self-recovery mechanisms. This means it should allow nodes (i.e. vehicles / sensor nodes) to join / leave the network and form / leave groups.
- Allow information caching, processing, retrieving and publication (i.e. in case of data collected by sensor nodes).
- Be as simple as possible. The protocol should be simple and low consuming since it is intended to be installed in VANET, WSN, and the gateway with limited capacity.
- Enable the scalability. It should enable more nodes (i.e. vehicles / sensor nodes) to join the network without overloading.
- If possible, be standard. Even more, available standard protocols should be analyzed to determine whether or not they can be reused.

We selected the Session Initiation Protocol (SIP) to be the protocol for the gateway since it meets our requirements. Firstly, SIP is a standards protocol, widely deployed and easily extensible. It has already been implemented to work in devices with limited capacity and its structure is simple.

Secondly, the different procedures are enabled by some of the existing extensions. Here we present a brief introduction about the characteristics of this protocol. The SUBSCRIBE / NOTIFY methods included within the RFC3265 (It's about the Session Initiation Protocol – Specific Event Notification) [28]. It provides an option for asynchronous notification of events. The event-notification framework is proposed to enable entities in the network to subscribe to receive the current state or updates from a remote node. If the state changes in the remote node, NOTIFY messages are used to inform it to subscribers. The Presence Framework, for instance, uses this extension to control the vehicles registered to each "presentity". This event-notification framework enables the creation of groups inside the gateway where the information needs to be disseminated from the super node "leader" to one or several participants in an asynchronous way.

Additionally, the INFO method described within the RFC2976 (It's about the SIP INFO Method) [29] enables the exchange of the application level information. Its goal is not to change the state or the parameters of the session but to send information required by applications. The advantage behind this extension is that it allows the application to determine the message body.

Moreover, as the self-organizing is enabled by SIP multicast addressing, since in the beginning, the nodes do not know who their super node "leader" is, they send a message to a multicast address that will allow its location.

3.2.2 Procedures

In the previous section we described the protocol that rules the exchange of messages among the gateway, in this part we show how those messages are used by different entities in the architecture to enable the effective functioning of it. We described each of the main procedures; the roles involved in the process and the message exchanges between them.

Self-Organization

Self-organization is mandatory since we are considering an ad-hoc network built out from moving vehicles as well as the sensor nodes that do not know each other at the creation of the network.

The mechanisms defined for discovering the other nodes is the SIP multicast addressing. A multicast address is assigned to all leader nodes and known by all nodes in the network. We assume that each node knows the functionality it can play and provide in the architecture and with the gateway and thus, the groups it needs to connect to. The joining process depicted in Figure 3-2 explained here below

The gateway should be functional when at least one vehicle / one sensor node is available, so we decided that first node to join the network is the SVIS and / or SSIS. Once it starts to play its role, it creates two groups; the first one is for being leader node, meaning that it enables the reception through the multicast address. And the second one is for the Vehicle

Ingress Spots and the Sensor Ingress Spot respectively, offering the connectivity for the moving vehicles and the sensor nodes.

When another node joins the network, it determines the roles it can play and the groups it should join. Then, it sends a SIP INFO message to the multicast address requesting the address of their leading node in charge of its group (the selected leader). The answer is sent by any of the leading peers in another SIP INFO message indicating whether there is a leading node (sending its address) or not.

If there is no leader yet, the node becomes the leader node for the group. This implies that the multicast address is assigned to it and that the reception of the SIP SUBSCRIBE messages from nodes belonging to its groups is enabled.

Sequence Diagram

Sequence diagram in (figure 3-4) present a dynamic view of the system. Two main processes are described, presenting the main capabilities the system offers. On one hand, a detailed presentation and description of the nodes (i.e. vehicles / sensor nodes) joining the networks. It shows how the components participate in order to find the leader of the group and interact with it. On the other hand, if the leader is not present, the new joining node will be the leader for that group and will accept new nodes to join in the network. In the next subsection, a detailed description of the self-organization process is elaborated.

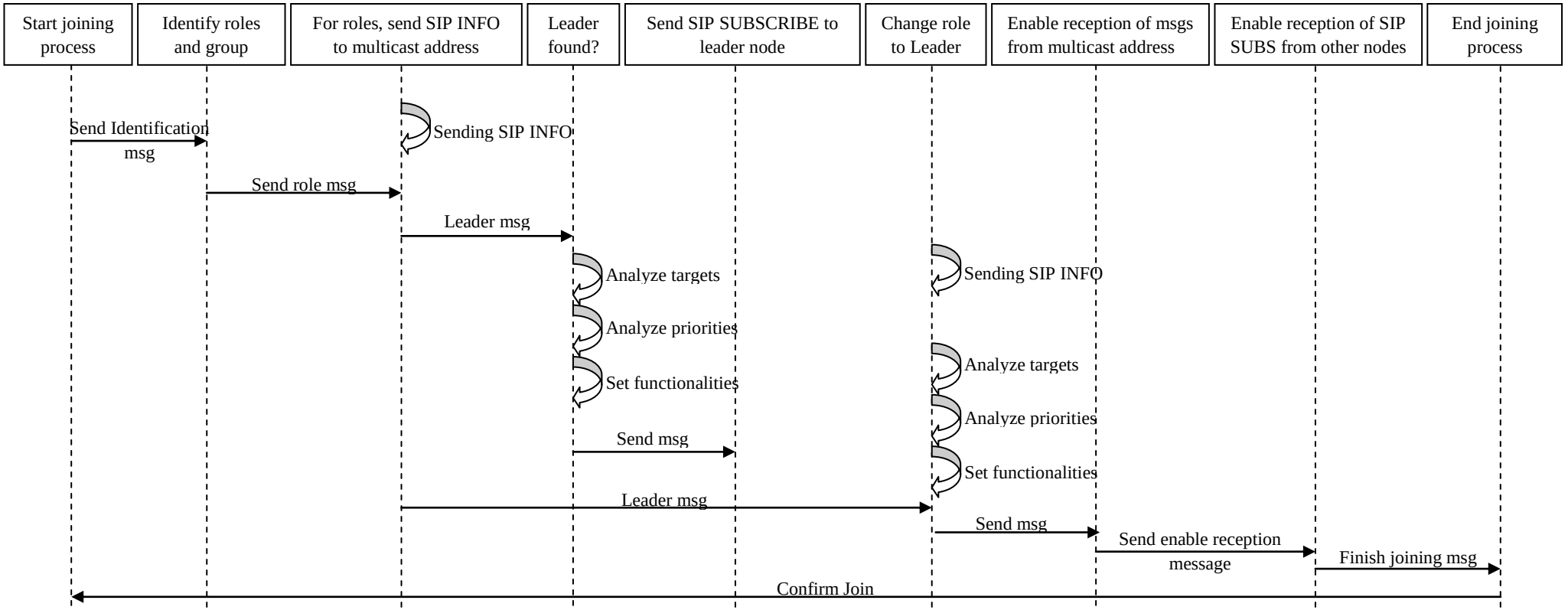


Figure 3-4: Self-Organization Process

For example, if a node joins the network (i.e. VANET) and identifies that it should play the role of leading other nodes. The node takes for each role the groups to which it should be part of. First for the leading role, it sends a multicast SIP INFO requesting the address of the SVIS that handles the VANET group; if there is no leader then this node changes its role and becomes the SVIS. Once this happens, the node enables the reception of SIP INFO message through the multicast address and of SIP SUBSCRIBE request from other joining and participating nodes. Otherwise, if the leader node has already been selected, it sends a SIP SUSBCRIBE to it and after all, it got the confirmation back that it has joined successfully the network.

For self-recovery purposes, we decided that all participating nodes (i.e. vehicles / sensors) within a group should be aware of the group members. So, each time a group changes (i.e. a node joins or leaves the group) the leader sends a SIP INFO message with "infoGroup" action and the list of all members to all nodes registered in the group.

In some roles (e.g. VIS, SIS) information is received by the simple nodes and they need to communicate it to the leader. For these cases, once a node joins the group, the leader peers sends a SIP SUBSCRIBE request to the node which allow that each time the peer gets information it forwards it with a SIP NOTIFY message.

Once the gateway is set up (there is at least one node playing each role), the SVIS, and SSIS registers to the VANET, and WSN respectively by sending a SIP REGISTER. The VIS, and SIS roles can be played by the leader nodes, thus implying that the gateway could be considered as set up even with the two leader nodes and no other nodes in these two groups.

Self-Recovery

There are three possible scenarios where self-recovery is required for the system. The first one is when a node (i.e. vehicles / sensors) leaves voluntarily, the second one when it is unexpected, and the third one when a node is forced to leave the system. At this point in the project we will only consider the voluntarily departure from the overlay.

Each node is responsible for informing other participating members of its departure. To do this, it uses a SIP INFO message with a "leaving" action in the message body. The following

presents how self-recovery process is done depending on the role of the node leaving as depicted in figure 3-5.

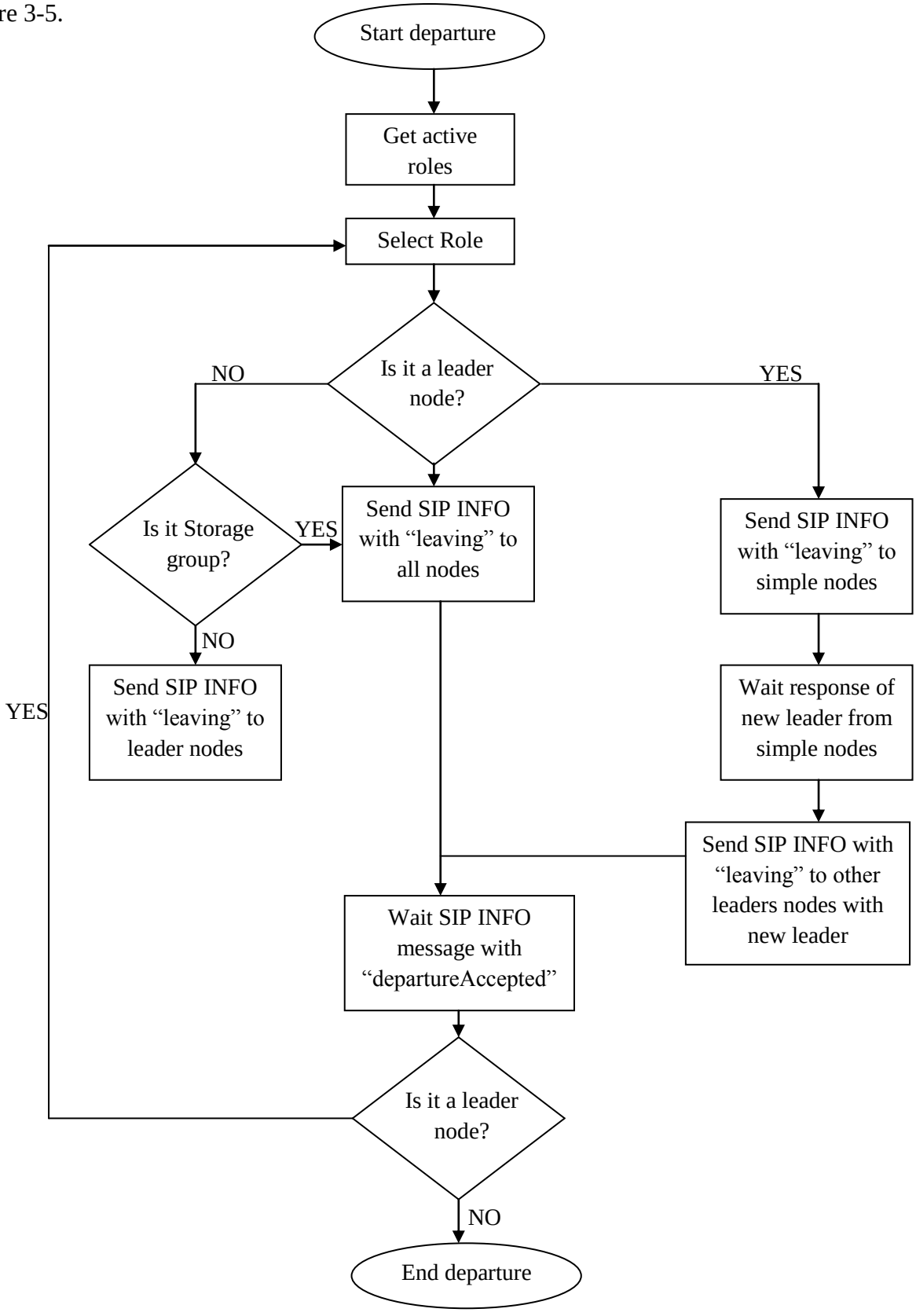


Figure 3-5(a): Node leaving the network

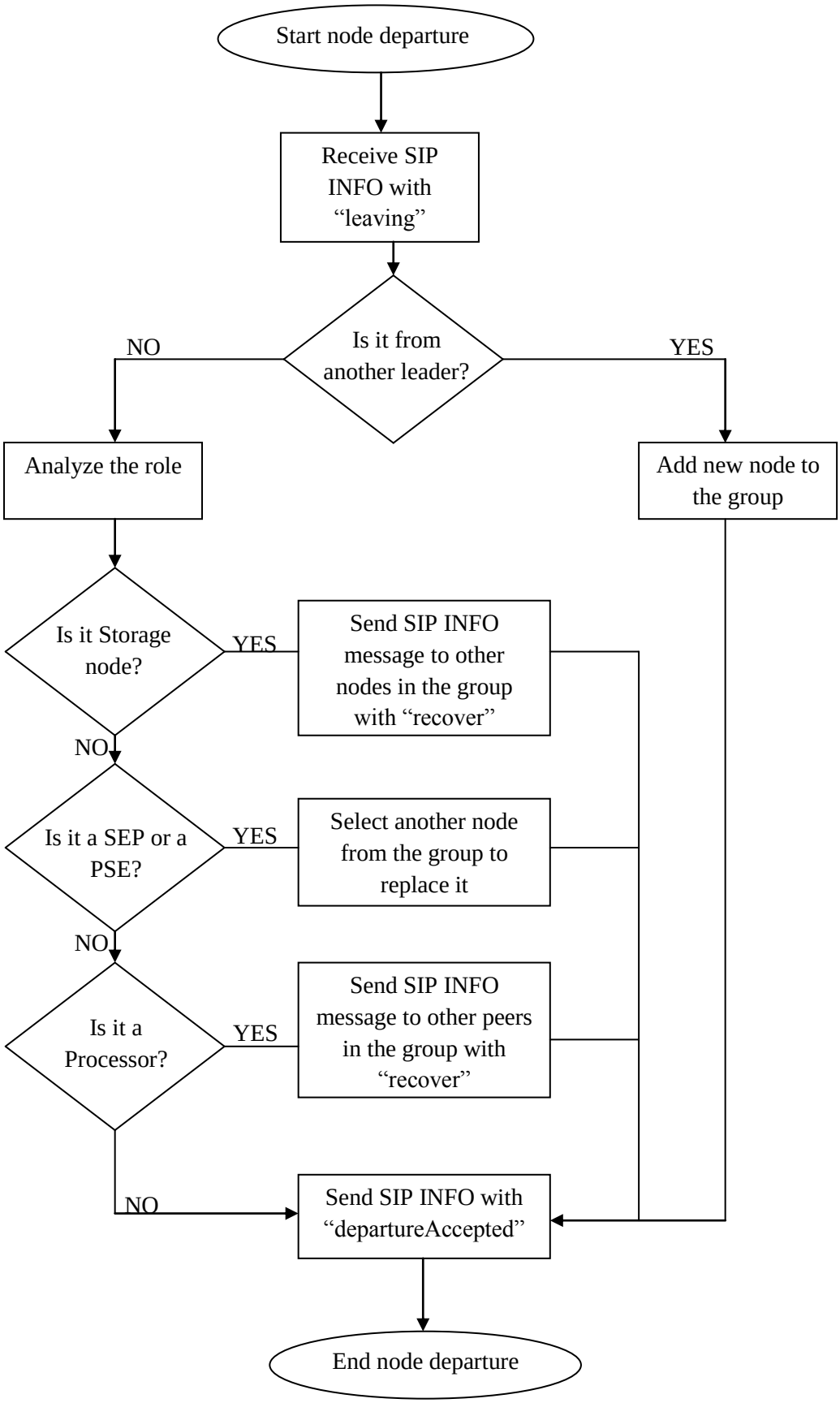


Figure 3-5(b): Leader node receiving the message

When a node is leaving, as shown in figure 3-5(a), the first thing it needs to verify and identify at the same time is the role that is actually playing in the overlay. For each node it should analyze if it is a simple node acting in the network or a leader node. If it is a participating node acting normally within the network, it sends a SIP INFO message to its leader node informing that the departure process has begun. In case it is a cache or processing node, they should send the message to their leader node as well as to other nodes within the group, since they are all connected in a fully meshed way. When the node itself is the leader of the group, it sends the message to all of the normal participating nodes in its groups and waits for a reply from one of the nodes found that will be playing as a new leader to such a group. Alternatively, in the case of the cache or processing node departure, the normal nodes responsible for the selection of the leader are the cache and processing group within the network. Once they receive the message; they forward the decision to the other leader nodes in the leader group. Afterwards, the nodes wait to receive a SIP INFO message with "departure accepted" to remove the role and do the same procedure with all roles till there is no active role. At the end, it gets disconnected.

On the other hand, figure 3-5(b), the leader node when it receives a SIP INFO message with leaving part on it, if the message is from another leader node then a new leader has to be selected in order to replace it and the node just updates the references only. But if it comes from a normal node participating within the network, it determines the group to which the node belongs to and then, the role it is participating with. When a processing node is leaving, the leader node sends a "replace processor" request to all nodes within the group. Moreover, when a cache node will be left, the leader node sends a "retrieve and restore" action to all nodes found within this group where a restore measure should be executed to get the pieces of information store inside it. When it is one of the Ingress Spots "i.e. VIS, PIS, or SIS", the leader node will choose another node to replace this leaving node, in case of the SIS connected with a sensor node. It sends a SIP INFO message with "restore" to the selected node, informing the actions the actions that should be taken into consideration and waits for the SIP INFO message "restoreGranted" that means that it has been accepted and established a new connection. Otherwise, it is a processing node in which case the leader node only tries to get any information still in the node. Last, once actions have been decided, the leader node sends the SIP INFO message to the node leaving with the expected "departureGranted".

3.3 Scenario

Now we have shown the proposed architecture design of the gateway, its procedures, factors, and the rules governing interactions and roles. This subsection will show how the architecture comes together in a specification scenario. It explores the scenario briefly described in chapter III in greater details.

Firstly, we assume that the overlay is already designed and organized and that there is at least one node in each role. Connections with the vehicles, IMS, and sensor nodes have also been established. Finally, moving vehicles are subscribed in the PS to receive both the multimedia services and also the contextual information from the sensor WSN.

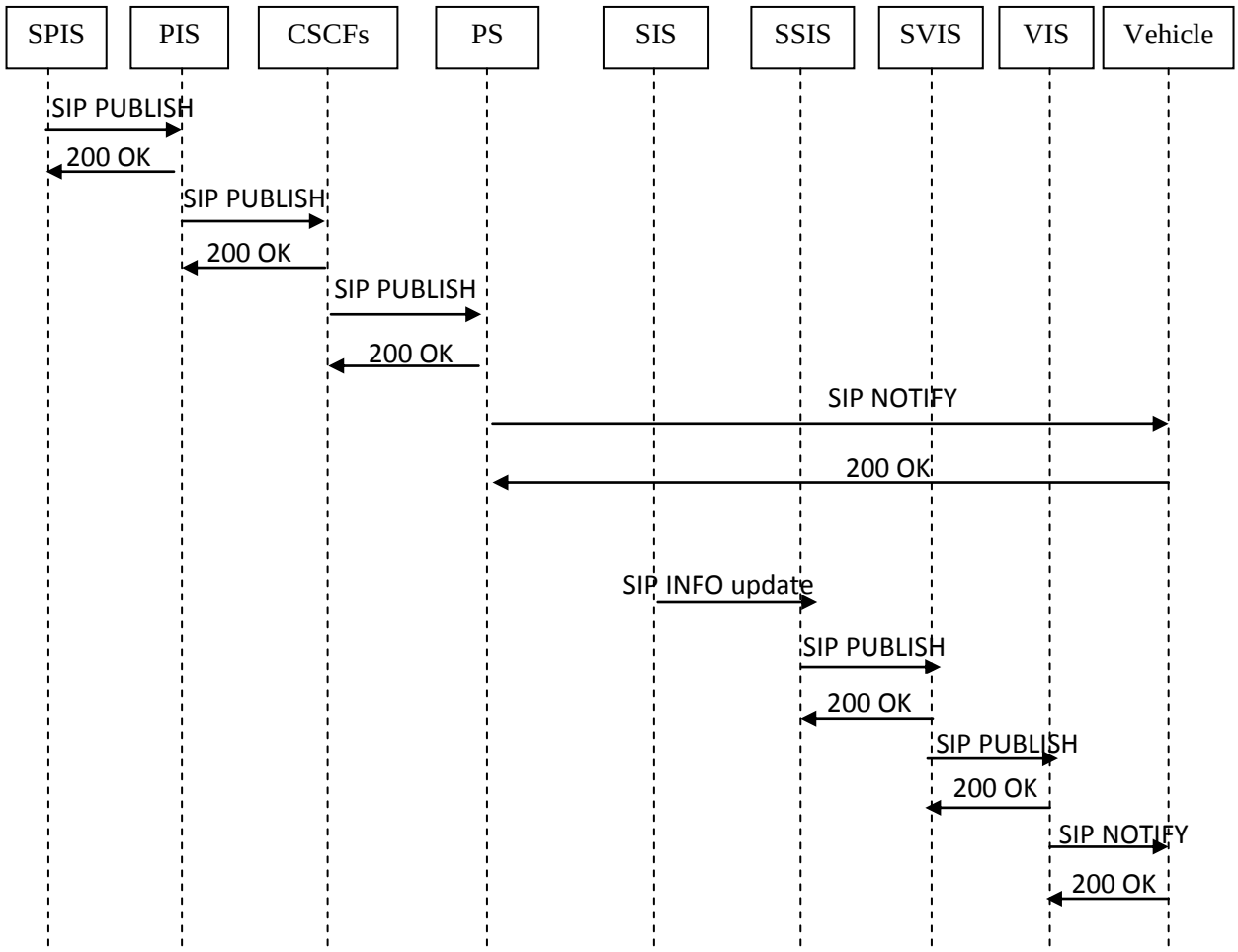


Figure 3-6: Sequence Diagram for multimedia services & sensed data present to the vehicles

Figure 3-6 shows the exchange between functional entities within the integration for the scenario where the multimedia services and the sensed data are ready and presented to the VANET architecture. The SPIS receives the information “multimedia services” and analyzes the request. When it determines that the information is to be published, the SPIS translates the data from the gateway information model to the corresponding format agreed before. Once this done, the node sends a message to one of the PSI in a SIP PUBLISH message. The PIS receives the message and forwards it the IMS CSCFs servers that are already configured to send the presence SIP PUBLISH message to the Presence Server (PS). A SIP NOTIFY is sent from the PS to the vehicle containing the services requested from the vehicles.

On the other hand, Sensors in WSN are periodically sending updates then they resend the information towards the gateway through the SIS. Since the leader node has previously subscribed to the SIP, it received a SIP NOTIFY message, after that, it analyzes the information and in this case determines that the services are already presented and ready to be sent so the data should be published in both the IMS and the sensor nodes. It sends a SIP NOTIFY to the SPIS which has also subscribed before.

Up to this point the information has reached IMS and the sensor nodes and is available to the vehicles’ users. As the example scenario shows, a SIP NOTIFY is sent from the PS to the moving nodes “vehicles” and the multimedia services are sent accordingly.

CHAPTER 4 IMPLEMENTATION AND VALIDATION

In the previous chapter, the detailed architecture intended to meet the system requirements was presented. Now, this work is validated in order to guarantee that it certainly matches and achieves the requirements.

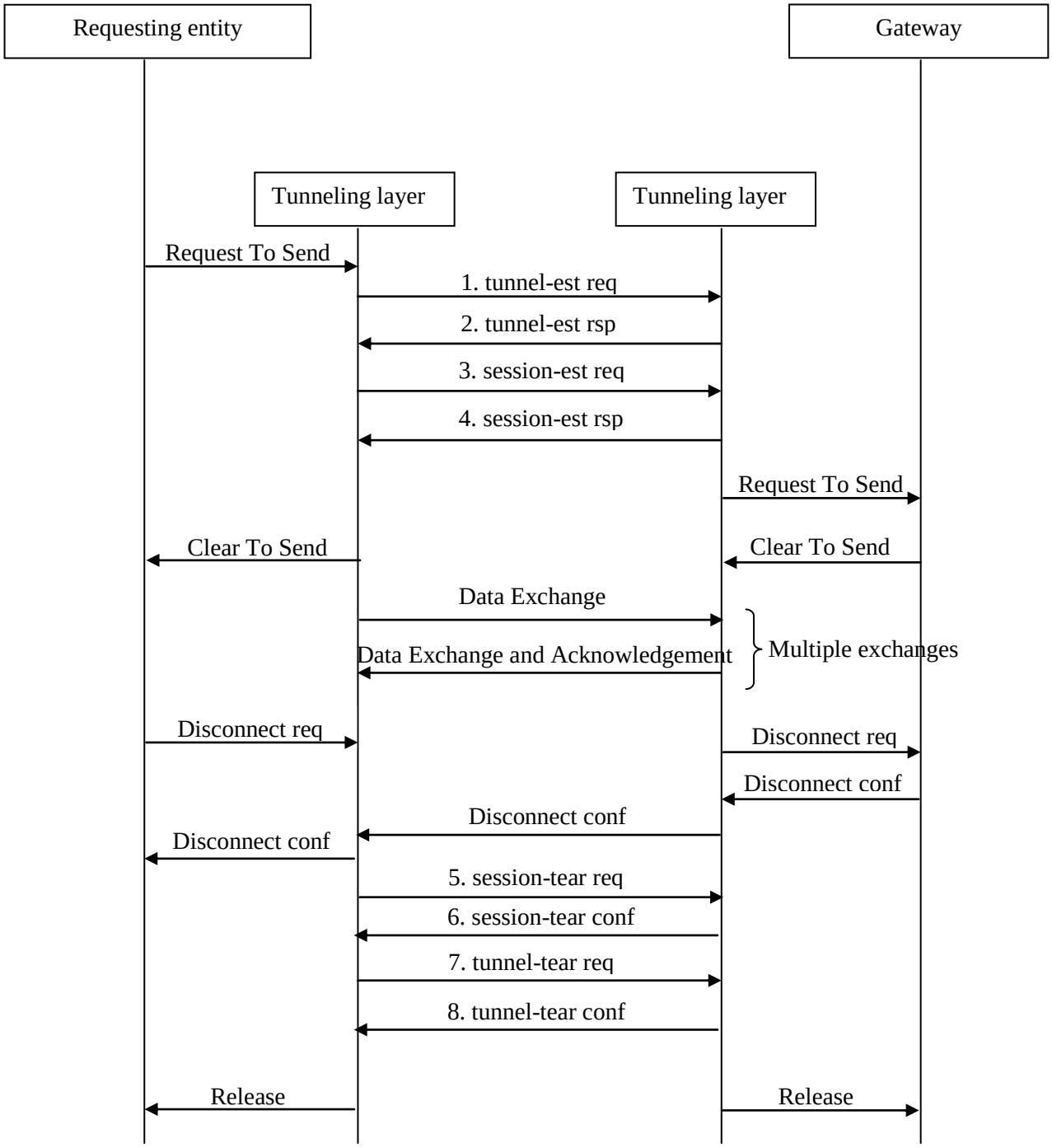
This chapter shows the process of the implementation and validation of the overlay gateway proposed and depicted in the previous chapter. First of all, as implementation, we analyzed the performance of our architecture as *compared* to the other architectures in the literature. As explained before, while different solutions can be found in the literature - the bulk of these solutions are based on a fixed architecture for a proactive integration of even Vehicular Ad-hoc Networks (VANETs), or Wireless Sensor Networks (WSNs) and the services they provide in the IP Multimedia Subsystems (IMS) architecture, as opposed to the reactive architecture that we propose. For this reason, we will use the same analytical model for the proactive architectures presented in [30] and compare it to the model developed for our solution.

We will compare the performance of these two types of architectures with respect to first the *average number of packets generated per requested services*, second - *the number of services found per service request.*

Based on [30], an extension to the IMS architecture to support dynamic discovery of WSNs and the services is provided, independently from their location and the protocol stack they implement. The solution used User Equipments (UEs) as Gateways (GWs) towards the targeted area where WSNs could be existed through the implementation of a *tunneling layer*. They analyzed the performance of the proposed architecture in a comparative study which we will base our study and analysis on. The same concepts and techniques will be applied as we are dealing with WSNs in our architecture and moreover we are going to add the vehicles to the main study which are the main aspect of our research.

4.1 Procedures

In this section we will highlight the discovery and integration procedures, describing messages exchange between the requesting entities (i.e. vehicles, sensor nodes, and IMS) and the gateway (GW) through the *tunneling layer*. The message flow between the involved entities is represented in the following figure.



req: Request

rsp: Respond

est: Establish

conf: Confirm

Figure 4-1: Message flow between entities of the architecture

- **Tunnel / Session Establishment and Tear Down the Connection between the Gateway and the External World (i.e. vehicles, sensor nodes, IMS):**

For each User Equipment (UE) - Gateway (GW) in the targeted area, the tunneling layer in the previous architecture (figure 4-1) stars by sending a tunnel establishment request (tunnel-est req, *message 1*) containing the locally assigned tunnel ID to which the tunneling layer of the gateway responds with a tunnel establishment response (tunnel-est rsp, *message 2*) also containing the locally assigned tunnel ID. Once this response is received the tunnel is established and the tunneling layer assigned to the vehicles/sensors/IMS sends a session establishment request (session-est req, *message 3*) containing the destination tunnel ID and a unique session ID for both sides of the tunnel, to which the tunneling layer in the gateway responds with a session establishment response (session-est rsp, *message 4*) containing the destination tunnel ID and the same session ID. The reception of this response at the tunneling layer concludes the tunnel and session establishment procedures between the external world and the gateway and triggers the discovery procedures between both of the two worlds.

Once the communication with the gateway terminated, the session and the tunnel are silently dropped. Optionally a tear down procedure can take place starting with closing the session by sending a session tear down request (session-tear req, *message 5*) containing the tunnel and session IDs, then closing the tunnel by sending a tunnel tear down request (tunnel-tear req, *message 7*) containing the ID of the tunnel and 0 as session ID. Each of these requests is answered by the tunneling layer on the other side of the tunnel with a tear down confirm (session-tear conf, *message 6* and tunnel-tear conf, *message 8* respectively). And finally the release of the two communication tunnels will be occurred.

Based on the previous indications, right now we are going to assess the performance of the system.

4.2 Performance Analysis

We define the following parameters to achieve our objectives:

n_v number of vehicles in the selected region

n_{WSN} number of sensors in the targeted area

N_{Req} average number of *packets* per requesting service (*packet per unit time*)

f_{Req} frequency of *packets* sent to request service(s) (*packet per unit time*)

f_{Serv} frequency of service *packets* responded (*packet per unit time*)

P_{ims} average probability that a IMS offers the requested service

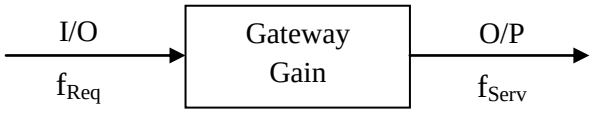
P_{WSN} average probability that a WSN offers the requested service

4.2.1 Number of packets generated per requested service

Here we are considering the following: The punctual services that are answered only once. For the punctual services after analyzing the presented works in [31], the average number of packets per unit time (i.e., "second") to get the aimed service(s) for the proactive architectures N_{Proact} is dependent on the average number of packets per request service(s) N_{Req} from each of the vehicles and sensors, the frequency of packets sent to get the service(s) f_{Req} and the responded packets for that service(s) f_{Serv} to/from the gateway respectively.

N_{Proact} can be defined as the number of entities to be serviced (vehicles or sensors) multiplied by the average number of packets per requesting service (i.e., for both the vehicles and sensors) multiplied by the gain of the gateway.

The gain of the gateway can be identified as the ratio of the serviced frequency to the requested service one.



The above sentence can be formulated mathematically as:

N_{Proact} = (no. of vehicles + no. of sensors) × average number of packets to request × gateway's gain

Applying the above general form to the notations that we defined previously:

$$N_{Proact} = (n_v + n_{WSN}) \times N_{Req} \times \left(f_{Serv}/f_{Req}\right) \quad (1)$$

Considering the ***zero impact case***, we assume that all the services are already presented in the core network when a request(s) is/are sent and that all the entities found will get a response when they send requests to the gateway (f_{Req} *will not be equal zero*), a response (i.e., service) will be available when a request(s) is/are popped-up.

On the other hand, for our reactive architecture the average number of packets per unit time (i.e., "second") N_{React} is independent of the frequency of both the requested and service packets responded; the service retrieval is done once the request(s) is/are sent to the gateway. In other words, this is a different case rather than that of the proactive architecture which already includes the services in the gateway, but here once a request(s) is/are sent, the gateway will process it and then retrieve the service, finally it responds with the packets of such service.

Also it depends on the packets sent to start making connection establishment between the requesting entity and the gateway (i.e., tunnel establishment request and session establishment request) plus the average number of packets per requesting service to the gateway.

N_{React} can be defined as that each entity (i.e., vehicle or sensor) sends to request a connection establishment with the gateway as mentioned above and also considering the average number of packets sent to get the required service(s). The above statement can be formulated mathematically as:

N_{React} = (requesting entity) × [average number of packets to req. + tunnel est. req. + session est. req.]

Applying the above general form to the notations that we defined previously:

$$N_{React} = (n_v + n_{WSN}) \times [N_{Req} + 2] \quad (2)$$

By consulting (figure 4-1) again "page 63", "*2*" represents the main processes between any entity and the gateway throughout the "*tunneling and session establishment's requests*". Once they are ready and established the flow of requesting services and services response can be taken into action.

From equation (1) and (2) we can calculate and compute the *ratio* of proactive to reactive number of packets per service as in the following equation:

$$RPKTS_{Punct} = (n_v + n_{WSN}) \times N_{req} \times (f_{Serv}/f_{Req}) \,/\, (n_v + n_{WSN}) \times [N_{Req} + 2] \quad (3)$$

Using Matlab we plotted these equations in figure 4-2. Figures 4-2.a and 4-2.b represent equations (1) and (2) respectively. In equation (1) we vary the total average requested packet's number N_{req} between 1 and 20, and the service request frequency (f_{Serv}/f_{Req}) between 0.1 and 20.

In equation (2) we vary only the relevant parameter which is the number of vehicles and sensor nodes $(n_v + n_{WSN})$ between 1 and 10.

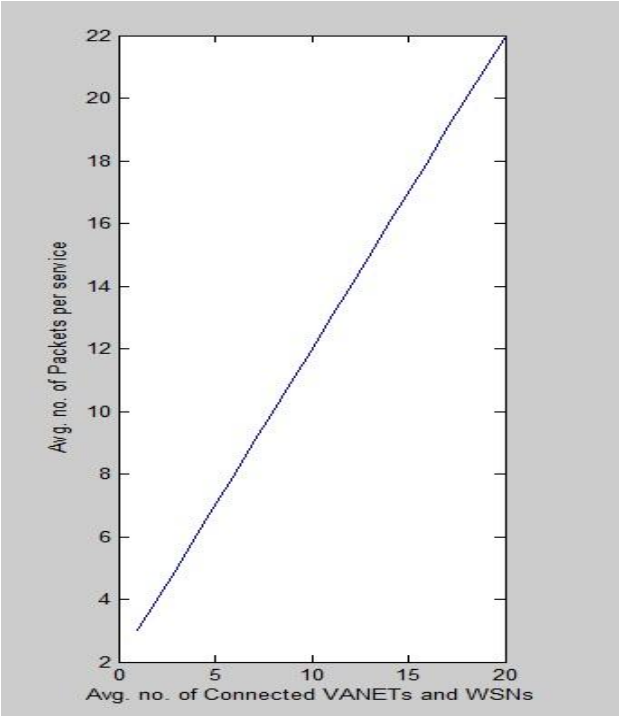
Finally figures 4-2.c and 4-2.d represent equation (3) for the number of existed networks between 1 and 5 respectively. In equation (3), we vary the number of connected vehicles and

sensors $(n_v + n_{WSN})$ between 1 and 10, which varies the proactive to reactive total packets' number ratio $((n_v + n_{WSN}) \times N_{Req} / (n_v + n_{WSN}) \times [N_{Req} + 2])$ between 0.05 and 0.2.

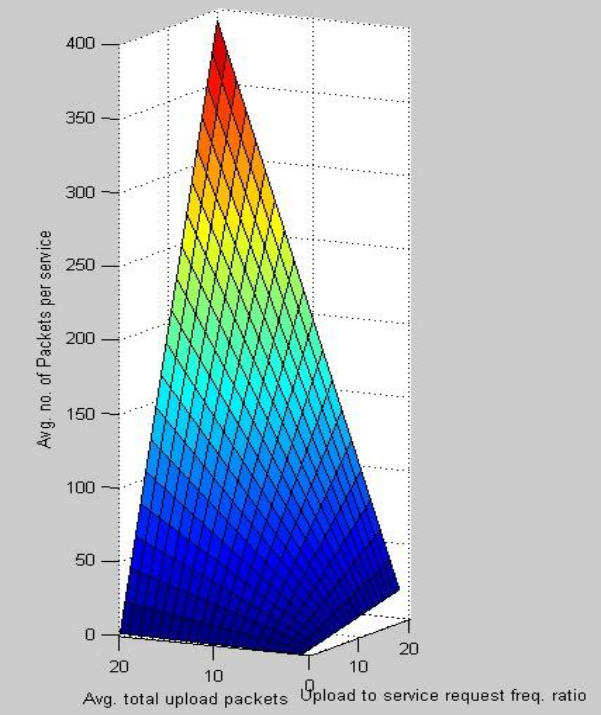
Here for simplicity, we consider the number of vehicles and sensor nodes $(n_v + n_{WSN})$ is the same for both the proactive and reactive architectures. Also in this equation we vary the requests to services ratio as for equation (1).

The results in figure 4-2 show clearly that the proactive architecture yields a lower packet overhead when the average number of packets per request of the vehicles and sensor nodes is equal 1, but this advantage is quickly lost as compared to the reactive architecture when the average number of packets per request to service frequency increases. This result is normal because the packet overhead in the proactive architectures depends on the number of vehicles, sensor nodes, and the average number of packets per request and the service provided frequency ratio. In the reactive architecture, this overhead depends solely on the number of vehicles and sensor nodes.

a. Proactive architecture



b. Reactive architecture



c. $N_{Req} = 1$ d. $N_{Req} = 5$

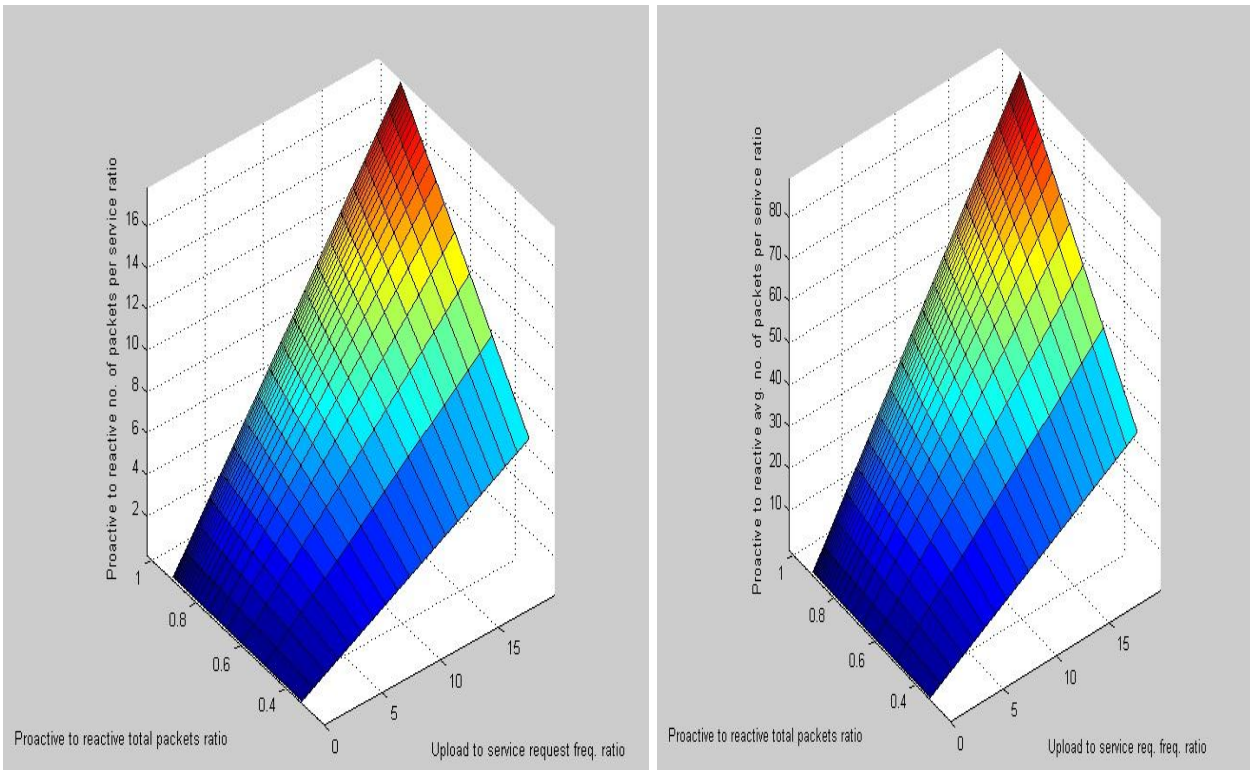


Figure 4-2: Proactive to Reactive total packets ratio for punctual services

4.2.2 Number of Services found per Service Request

In the case a proactive architecture, there is no actual service discovery, since all available services are already registered in the central server. Still we will represent the average number of discovered services $N_{ServProact}$ per unit time (i.e., "second") as the number of connected vehicles n_v and sensors n_{WSN} according to the average probability of IMS and/or WSN to offer the requested services P_{ims}, P_{WSN}

$N_{ServProact}$ can be viewed as the number of vehicles multiplied by the average probability of the IMS which provides the requested service(s) plus the number of sensors multiplied by the average probability of WSN to provide the requested information. The reason for choosing the average probability for each of the service's provider is justified since we do not know the exact probability of providing the service(s) or not for each type (IMS, WSN) to each of the requested entities (i.e., vehicles and/or sensors).

The above statement can be formulated mathematically as:

$$N_{ServProact} = n_v \left[\frac{1}{n_v} \sum_{i=1}^{n_v} P_{ims,i} \right] + n_{WSN} \left[\frac{1}{n_{WSN}} \sum_{j=1}^{n_{WSN}} P_{WSN,j} \right]$$

$$N_{ServProact} = \sum_{i=1}^{n_v} P_{ims,i} + \sum_{j=1}^{n_{WSN}} P_{WSN,j} \quad (4)$$

On the other side, in case of the reactive architecture that we purpose, the average number of discovered services $N_{ServProact}$ per unit time (i.e., "second") depends on the number of the discovered vehicles and sensors $n_{v,} n_{WSN}$ according to the average number of requests sent from the vehicles and sensors N_{Req} to the gateway and the average probability of getting this service requested from the requested network P_{ims}, P_{WSN}. So the average number of discovered services for our reactive architecture $N_{ServReact}$ can be represented mathematically as:

$$N_{ServReact} = \sum_{i=1}^{n_v} P_{ims,i} + \sum_{j=1}^{n_{WSN}} P_{WSN,j} \times N_{Req} \quad (5)$$

The Matlab plotting of these equations (4, and 5) is represented in figure 4-3. Naturally as the reactive to proactive deployment level ratio increases, the number of services discovered by the first architecture is higher than the second.

From equation (4) and (5) we can have the *ratio* of proactive to reactive number of packets per service as:

$$RSER_{Punct} = \sum_{i=1}^{n_v} P_{ims,i} + \sum_{j=1}^{n_{WSN}} P_{WSN,j} \; / \; \sum_{i=1}^{n_v} P_{ims,i} + \sum_{j=1}^{n_{WSN}} P_{WSN,j} \times N_{Req} \quad (6)$$

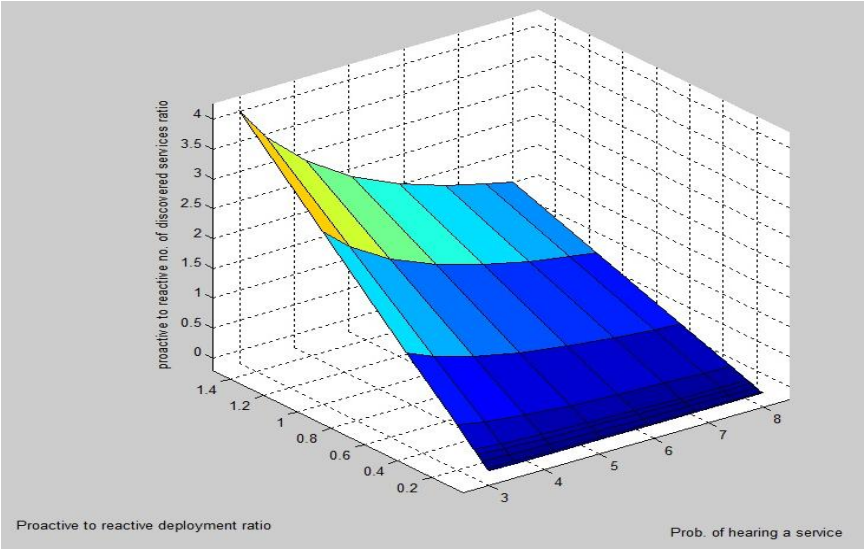


Figure 4-3: Proactive to reactive average number of discovered services

4.3 Validation

The validation process is done from three perspectives: firstly we will show how the architecture meets the requirements, secondly we will compare our proposal with the architectures that integrate VANETs, IMS and WSN and were described in the state of the art; finally, it is our goal to prove through a simulation that using the proposed gateway as part of the solution increases scalability and fault-tolerance when compared with a centralized gateway solution.

4.3.1 Requirements vs. Architecture

Before verifying how the architecture assesses the requirements, let us review what they are as defined in chapter II:

- Independence. The integration should be as independent as possible from both the VANET, IMS and WSN.
- All types of information related to all types of entities sensed by a WSN should be made available to IMS.
- All types of multimedia services of all types should be available to the VANET users throughout the IMS.
- Translation from the information model used in the sensor network to the information model used in IMS and vise versa.
- Translation from the information model used in IMS to the information model used in VANET and vise versa.
- Publication of the information into IMS.
- Scalable and fault-tolerance.
- Support lightweight communication mechanisms.

Although the architecture was defined in the light of these requirements, it is important to validate which features of the architecture satisfy which requirement. Table 4.3.1 presents a matrix where each requirement is achieved by one or several features presented in the proposed architecture.

Table 4.3.1: Requirements vs. Architecture

	Gateway as integration solution	Service definition	Extended PIDF	Information Model	Vehicles
1. Independence	*X*			*X*	
2. Information and entities from WSN			*X*		
3. Translation		*X*		*X*	
4. Publication		*X*			
5. Storage		*X*			
6. Processing		*X*			
7. Traditional and P2P IMS	*X*				
8. Scalability		*X*			
9. Fault-tolerance		*X*			
10. Light-weight communication					*X*

Independence, for instance is achieved by several features including the use of a gateway and defining a new information model. Using a gateway as an integration point provides independence from the two technologies merged as was shown in chapter II section 2.7, it requires minimal or no changes in the architectures involved since all translation is done by the gateway. Additionally, independence from the gateway point of the view is also achieved by the information model used inside. It assures that minimal changes will need to be done in case the information model from the VANET, WSN or the presence service in IMS change.

The provisioning of *all types of information and entities* from VANET or WSN in IMS is accomplished by the extended PIDF that we have chosen to use in our architecture. This extension allows the modification of presence information to add non-user related information.

Translation is provided as explained in the previous chapter by three services that were defined in the overlay. The first one is the service that translates between the information model from the VANET and the gateway. The second one is the service that translates between the information model from the mobile sink WSN and the one from the gateway. And the third one is between the extended-PIDF and the information model from the gateway. The information

model is flexible enough to support both models. Thus, the features included in the architecture to meet the requirement are the defined services and the information model.

Publication is supplied by the service defined as part of the IMS entry point group. This service allows the publication of information in IMS by using SIP PUBLISH. Similarly, *Processing* is done in the service that has been defined for this purpose; which is supplied by the processing group and the information model.

Data *Storage* is met by two features, the P2P architecture and the service definition. The service definition as shown before with the other information management processes, is achieved through the service that has been for this purpose and that is supplied from a specific group (i.e. Storage group). However, this is also possible thanks to the P2P overlay; since the feasibility to provide distributed storage in P2P networks has been proved.

Traditional and P2P IMS were considered all along the design. Having a P2P overlay as gateway makes the future integration with P2P IMS more natural. Additionally, this feature also helps achieved the *Scalability* and *Fault-Tolerance*, which, as seeing before, are characteristics in P2P systems. However, it is important to clarify that in our project we do not consider unexpected departure, which is also important to improve the fault-tolerance in the solution.

Finally, the fact that we build our gateway with the presence of user agents obliges us to work with *lightweight communication mechanisms.*

4.3.2 Architecture comparison

Once it has been proved that the proposed architecture meets the requirements defined according to the motivations, we will compare the solution with other gateways proposed for the integration of these three technologies. The proposals that have been chosen for the comparison are WSN/IMS gateway from El Barachi et al. [12], the e-SENSE architecture proposed by Gluhak and Schott in [13] and the PN gateway to integrate Personal Area Networks and Sensor Networks with IMS described in [22] by Arbanowski et al. Moreover, the VANET-WSN gateway proposed by Festag et al. in [19].

Table 4.3.2 depicts the comparison between proposed architectures; it shows for each requirement whether it is Supported, Not Supported, Partially Supported or Not Considered in the design.

Table 4.3.2: Comparison between proposed gateways for integrating VANET, WSN and IMS
S: Supported, N: Not supported, P: Partially supported and N.C: Not considered

Requirement	**Overlay Gateway**	**WSN/IMS gateway**	**e-SENSE architecture**	**PN Gateway**	**VANET-WSN gateway**
1. Independence	*S*	*S*	*N*	*S*	*S*
2. All types of Information and entities	*S*	*S*	*S*	*N*	*P*
3. Translation	*S*	*S*	*N*	*N*	*S*
4. Publication	*S*	*S*	*S*	*N*	*S*
5. Storage	*S*	*S*	*S*	*N*	*N*
6. Processing	*S*	*S*	*S*	*N*	*S*
7. Enable traditional and P2P IMS	*S*	*N.C.*	*N.C.*	*N.C.*	*N.C.*
8. Scalability	*S*	*N*	*N*	*P*	*N*
9. Fault-tolerance	*P*	*N*	*N*	*N*	*N*
10. Lightweight communication mechanism	*S*	*N*	*N*	*S*	*S*

The PN gateway proposed by Arbanowski et al. considers some of the non functional factors that other proposals do not. The design is meant to work in hardware devices with limited resources like wireless sensors or other tiny smart devices. For this purpose they use TinySIP (standard protocol for any kind of resource constrained device) to enable interactions between nodes through SIP messages. It is independent from IMS since the connection is made through an entity called SmarDust Enable or via specific device that connect directly to CSCFs servers. However, this solution does not explain how the information will be processed, stored and publish in IMS or other entities.

The e-SENSE architecture supports most of the information management procedures we defined, except for translation which is not needed in this proposal. As mentioned in chapter two, the functional entity added in IMS is the e-SENSE Service Enabler SE that provides sensor-based information, thus including all types of information in IMS. It additionally offers storage and processing of information. This entity receives the information from one or several e-SENSE Gateways. The gateway offers publication of information to the e-SENSE SE. Translation is not

supported because there is no need of translation, the information is delivered as sensor-based information and not in other format.

However, this solution does not support the non functional requirements related to scalability, fault-tolerance and lightweight communication mechanisms. The main drawback of this solution is that it is not independent from IMS. It adds a new server to IMS called the e-SENSE SE, instead of using one of the existents such as the presence service. Furthermore, storage and processing is supported by the e-SENSE SE and not by the gateway.

WSN/IMS gateway proposed by El Brachi et al. supports all information management requirements. It is a generic gateway that proposes minimal changes to the Presence Server in order to support diverse types of information. These types of information are also included in the proposal of the extended PIDF. Additionally, the gateway supports processing, storage, translation and publication of information.

Scalability, fault-tolerance and lightweight communication mechanisms are not considered as part of the solution. Additionally, P2P IMS is not considered for future scenarios and the integration may not be done straightforward since it can become a bottleneck.

VANET-WSN gateway proposed by Festag et al. supports most of the information management procedures we defined, except for storage which is not needed in this proposal. Scalability, fault-tolerance are not supported for this proposal and they did not considered at all the multimedia services presented by the IMS. Furthermore, the gateway supports Processing, translation and publication of information as well.

As can be seen in the analysis, our architecture meets all the requirements; it was previously exposed which features in the architecture contributed to their accomplishment. The only one that is partially supported is the fault-tolerance, due to the lack of definition for an unexpected departure of a node in the overlay. However the procedure can be later extended and is flexible enough to support the changes.

4.3.3 Vast simulation and results

In this section we evaluate the performance of the proposed solution through extensive simulation. We simulate our solution on Network Simulator 2 (NS-2).

Simulation is distinct as the process of designing a model of a real system and conducting experiments with this model for the function of understanding the behavior of the system and/or evaluating various strategies for the operation of the system. NS-2 is a packet level simulator and essentially a centric discrete event scheduler to schedule the events such as packet and timer expiration [32]. Centric event scheduler cannot accurately emulate "event handled at the same time" in real world, that is, events are handled one by one. The C++ classes of NS-2 network components or protocols are implemented in the subdirectory "NS-2", and the TCL library in the subdirectory of "tcl". NS-2 is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless networks functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS-2. In general NS-2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors [32, 33].

NS-2 provides users with executable command ns which take on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS-2 executable command ns. NS-2 provides the Network component like Node, Link, Queue, etc. they are created from the corresponding C++ classes; The other are compound components, that is, they are composed multiple simple C++ classes like Link are composed of Delay (emulating propagation delay) and Queue. We can say that in NS-2, all network components are created, plugged and configured from TCL. NS-2 provides the Event Scheduling that is associated with time. class Event is defined by {time, uid, next, handler}, where time is the scheduling time of the event, uid is the unique id of the event, next is the next scheduling event in the event queue that is a link list, and handler points to the function to handle the event when the event is scheduled. Events are put into the event queue sorted by their time, and scheduled one by one by the event scheduler [32, 34].

NS-2 has four main components: NS itself, Network animator (NAM), Pre-processing and post-processing. The primary component is ns, the actual simulator. This provides the software backup for programming networks models. The programming environment provides an interactive mode because the frontend OTcl interpreter allows for the direct programming of the simulator. It is also possible, and often times more efficient, to simply use a text editor to write scripts and run them from a terminal's command line [32, 35].

The second component is the network animator or NAM. This is a simple animator with two-dimensional (2D) graphics that help the user visualize and monitor the simulation both as a whole, and as individual components. The GUI only requires the trace files (created from the simulation scripts) as input. Pre-Processing and Post-Processing are also important components of NS-2. Examples of Pre-Processing are traffic and topology generator. An example of Post-Processing is simple trace analysis, i.e., xgraph and/or gnuplot, often developed using scripting languages such as *AWK*, *Perl* and *Tcl*. NS-2 provides the substantial support to simulate bunch of protocols like TCP, UDP, FTP and HTTP. NS-2 is discrete event simulator i.e. timing of events is maintained in a scheduler. Using xgraph/gnuplot (plotting programs) we can create graphical representation of simulation results. All the work done is under the Linux platform.

Simulation Configuration

The beginning of a simulation script is primarily used to configure the network components of a simulation. For example, components such as the topography, queues, wireless channel, routing protocols, etc. are usually configured at the top of a simulation scripts.

Typically, an array, *opt*, is used to store various elements that control a facet of the simulation. A block of code is used to initialize array elements of *opt*. For example, the command set *opt(x) 1000* assigns *1000* to the array element *x*. When a topography object is later created, the size of *x* component of the topography is assigned the value contained in *opt(x)*. By putting all of the options in a block of code, the user can easily go back and make any changes to the simulation. The following sample of code is used to set some of the elements of *opt* that are later used when creating network objects:

```
# Set simulation options

set opt(chan)       Channel/WirelessChannel          ;# channel type

set opt(prop)        Propagation/TwoRayGround        ;# radio-propagation model

set opt(ant)         Antenna/OmniAntenna              ;# antenna type

set opt(x) 1000                                       ;# x size for topography

set opt(y) 1000                                       ;# y size for topography

....
```

Many of the network objects are configurable through the OTcl linkage, as discussed in the previous section. For example, the following code sample configures the *802.11*protocl's slot-time to *13 μs* and SIFS to *32 μs*. In addition, the wireless physical layer settings are also configured through the OTcl linkage.

```
# set the 802.11 parameters.

Mac/802_11 set SlotTime_   0.000013            ;# 13 us

Mac/802_11 set SIFS_       0.000032            ;# 32 us

....

Phy/WirelessPhy set freq_ 5.9e+9               ;# 5.9 GHz

Phy/WirelessPhy set bandwidth_ 6.0e6           ;# 6 Mbps
```

Numerous other aspects of simulation can be specified at the start of the simulation script. To use the same physical layer setting as DSRC, the wireless channel's frequency is set to *5.9 GHz* and the bandwidth is set to 6 *mbps*. By setting the variables identical to those specified by DSRC standards, a simulation is able to realistically approximate a VANET.

The file *ns-2.35/tcl/lib/ns-default.tcl* sets various simulator components to their default values; this file can also be used to identity what simulation components a user is able to configure.

Creating Network Objects

After all of the configuration options have been set, the next step is to create the necessary network objects. To simulate a wireless network, a number of network components must be created. Some of the components needed to simulate a wireless network are *Simulator*, *Topography*, *god*, and *WirelessChannel*. The following code is used to create a new instance of the ns simulator and to create some of the standard objects that are needed for a wireless simulation:

```
# create a new simulator object
set ns_ [new Simulator]


# create the topography
set topo [new Topography]
$topo  load_flatgrid $opt(x) $opt(y)


# create god, the god object must
# be created for a wireless network
set god_  [ create-god $opt(nn) ]


# create a channel object
set chan_  [new Channel/WirelessChannel]
....
# Continue setting up the simulation.
```

Most wireless simulation will follow the same format as the source code listed above. This code first creates an instance of the simulator. Next, the topography of the simulation is created. It is followed by the creation of a god object, and finally a channel object is instantiated.

After the basic network objects have been created, the next step is to configure the mobile nodes. The mobile nodes are configured by issuing a command similar to the one that follows:

```
# configure the nodes of the simulation
$ns_ node-config    -adhocRouting $opt(rp) \
                    -llType $opt(ll) \
                    -macType $opt(mac) \
                    -ifqType $opt(ifq) \
                    -ifqLen $opt(ifqlen) \
                    -antType $opt(ant) \
                    -propInstance [new $opt(prop)] \
                    -phyType $opt(netif) \
                    -topoInstance $topo \
                    -channel $chan_ \
                    -agentTrace   $opt(at) \
                    -routerTrace  $opt(rt) \
                    -macTrace     $opt(mact) \
                    -movementTrace $opt(movt)
```

As previously discussed, the elements of *opt* are set at the beginning of the simulation script. Most of the parameters used to configure a mobile node are from the *opt* array. The NS-2 documentation describes the configuration option for a mobile node. After the nodes have been configured, the user is free to actually create mobile nodes. The following code creates an array of mobile nodes:

```
# Create the mobile nodes for the simulation.
for {set i 0} {$i < $opt(nn)} {incr i} {
set node_($i) [$ns_ node]
```

```
$ns_   initial_node_pos    $node_($i)    10

}
```

After the mobile nodes are created, the network events are scheduled to complete the simulation. An example of an event that must be scheduled is the transmission of a packet. In addition, the user will typically add some functions such as one that performs clean up when the simulation is completed. The execution of clean-up function is scheduled as are all the other events that occur in a simulation. After all of the simulation is configured, the final step is to run the simulation. A simulation is executed with the following command:

```
....

# Run the simulation;        this command is typically

# found at the very end of a simulation script.

$ns_ run
```

Creating Movement and Traffic

In the case of VANET, a mobility model is needed for the movement of nodes. It is important that the simulation of a VANET is as realistic to an actual VANET as possible. Typically, a separate file contains all of the commands needed to move the nodes during the length of the simulation. To insert the code contained in another Tcl file into the main simulation script, the *source* command is used. The following commands are required to execute all of the commands contained in the file *mobility.tcl*:

```
set   opt(sc) "mobility.tcl"          ;# mobility scenario

....

# Source the mobility scenario file to execute

# all of the commands with the mobility file

source $opt(sc)
```

Section 4.3.3.1 contains a detailed description of the mobility model used for the simulations. The user has a number of options available for creating network traffic (transmitting packets). To being, all of the commands needed to send packets could be included in the main simulation script. The drawback to this approach is a large number of lines of code may need to be added to the simulation script, and this approach would reduce the readability and manageability of the simulation.

Another alternative is to use a program such as constant bit rate generator "*cbrgen.tcl"* to generate all of the traffic. In this case, the user supplies a few command line options to the program, and the output of the "*cbrgen"* program is written to a file. Another choice is for the user to write his/her own program to generate the network traffic. This solution offers the user to the greatest control over the generation of traffic.

If a separate file is used that contains all the commands needed to transmit packets, the file containing the network traffic must be also sourced, in the same manner as the mobility scenario. The following code is used to source a traffic file:

```
# Source the file that contains the broadcast traffic.

puts "sourcing traffic file"

source $opt(cp)
```

Section 4.3.3.2 contains a detailed description of the network traffic used in the simulations.

Running a Simulation

To run a simulation, the user types *ns wireless-simulation.tcl*, and the ns interpreter executes all of the commands contained in the *wireless-simulation.tcl* script. The time it takes to execute a simulation can range from a few minutes, to several hours, to days, depending on the complexity of the simulation.

A number of command-line options are available to the user when conducting a simulation with the *wireless-simulation.tcl* simulation script. The following table contains a list of the command-line options for *wireless-simulation.tcl.*

For many of the simulations that a user will run, a single parameter of the simulation is varied, and a number of simulations are run to determine the effect that the changing the parameter has on the simulation results. In this scenario, it is beneficial to create another script with *bash* or *Perl or AWK* to launch consecutive simulations and automate as much of the simulation process a possible.

Table 4.3.3.1: Command-Line Options for wireless-simulation.tcl

Option	***Description***
nn	The number of nodes contained in the simulation.
stop	The time to end the simulation.
seed	The seed for the random number generator.
sc	The name of the mobility scenario file.
cp	The name of the traffic connection pattern file.
tr	The name of the trace file to write the output to.
nam_tr	The name of the nam-trace file to write the output to.
x	The size of the x-axis of the topography.
y	The size of the y-axis of the topography.
pkt_size	The size of the periodic broadcast packets.
help	Displays help to the user.

Table 4.3.3.2: Data Contained in an NS-2 Trace

Event	**Time**	**From node**
To node	Packet type	Packet size
Flags	Source address	Source port
Destination address	Destination port	Sequence number

As a simulation executes, the results of the simulation are written to the trace file. A trace contains detailed account of the events that occurred during a simulation. A trace is also used to record the time at which specific events occurred, and each event is written on separate line of the trace to make parsing the file easy. In addition, the user can specify what events are written to the trace. Some of the events that can be written to a trace are routing events, MAC layer events,

and mobility events (e.g., the position and movements of nodes). Furthermore, the simulation time can be improved if tracing is turned off for events of no interests.

The typical data contained on a line of a trace is given in Table 4.3.3.2. The text-based data of the trace is usually parsed with a script, which is typically written with *Perl* or *AWK*, to evaluate a simulation's performance.

NAM: Network Animator

The NAM: Network Animator is a visualization tool that can be used to further examine a simulation. The NAM program is written with Tcl/TK, and NAM is used for visually displaying a simulation. To use NAM, the user must enable the creation of a *nam-trace* file during the simulation. As a simulation executes, the results are written to a *nam-trace*. The format of a *nam-trace* is similar to a regular trace generated by the NS-2 simulator. When a simulation is complete, the results of the simulation will be written to the *nam-tace*; it is then used as input to the NAM program.

The following commands are issued to enable the creation of a nam-trace:

```
set opt(x)    1000                      ;# x size for topology

set opt(y)    1000                      ;# y size for topology

set opt(nam_tr)     "nam-trace.tr"      ;# name of the nam-trace file

....

# Set the file to write nam trace to and enable

# writing to the nam trace.

set namtracefd   [open $opt(nam_tr) w]

$ns_ namtrace-all-wireless       $namtracefd $opt(x) $opt(y)
```

In order to execute NAM, at the command line interface, the user has to type *nam nam-trace.tr*, where the *nam* executes the NAM program and the *name-trace.tr* is the name of *nam-trace file*. Once NAM has started, the interface to the NAM is similar to a video player. For example, the user can either fast-forward or rewind a simulation any time. Also, the user can control time resolution and will be capable of either speed up or slow down the simulation.

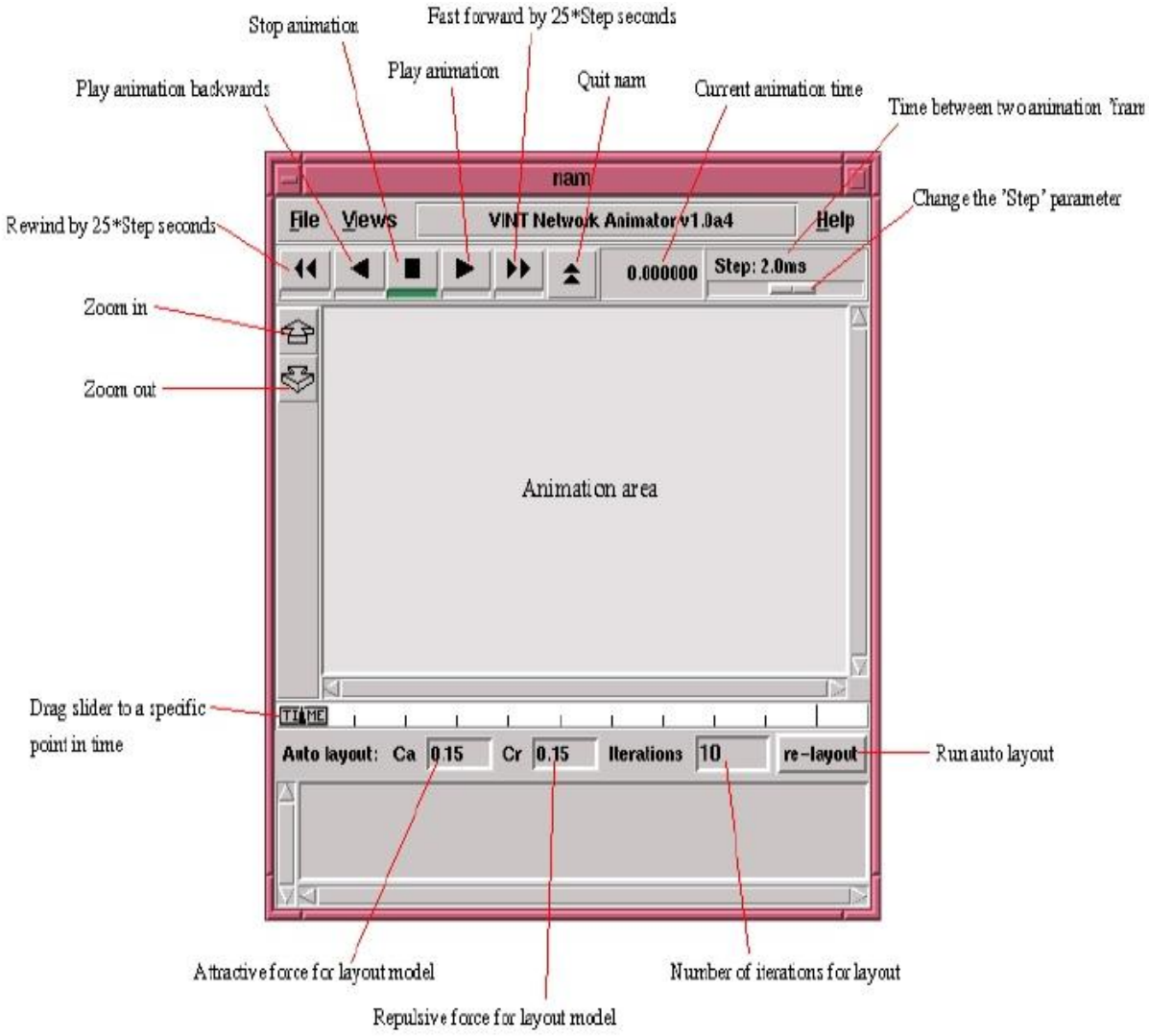


Figure 4-4: NAM controls

NAM gives the user some visual feedback for the simulation which already done, this is an advantage for it, but on the other hand, the drawback of it is that it does not present the user with quantitative results.

4.4 Mobility Model

Most of the mobile ad-hoc simulations are using the random waypoint mobility model; with the usage of this model, each node in the network will randomly choose a location to move towards, and it will randomly select a velocity with which to move towards that destination.

The velocity is selected randomly from [0, $velocity_{max}$], where $velocity_{max}$ is the maximum velocity allowed for a node. Subsequently, the node will continue moving towards its intended destination until it reaches there. When it arrives, it will pause for the moment of time specified by its "pause timer". Then, the node will randomly choose another destination to travel towards. The random waypoint model is provided by the *setdest* tool within the NS-2.

While the random waypoint model is frequently used to model the ad-hoc networks, this model is not so good to model a VANET because of the vehicle's movement uniqueness. Vehicles, in a VANET, cannot haphazardly travel anywhere with the topography. They are only able to move where there is an adequate road. Since a vehicle's movement is constrained by the structure of the road, the random waypoint model is considered to be unrealistic model for the VANET world. For this reason, an alternative mobility model must be used for the given VANET.

4.4.1 Freeway Mobility Model

The freeway mobility model is one of the possible models that can be used as a foundation of the node's movement. The freeway model emulates the movement of the vehicles on a freeway. The freeway model uses maps to create the mobility of the nodes. Nodes are only able to move where the road is identified by the map. With the given model, a map can contain several and different freeways, and each one of them may have multiple lanes. Moreover, the lanes within a freeway can be travelled in either one or two directions.

Freeway Model Characteristics

The following model is characterized by the following:

- Each node is restricted to move only within its own lane of the freeway. To simplify the complexity of this model, the model sacrifices some realism. As a result, vehicles do not have the ability to change lanes as they would on a real freeway.

- A safety distance will be maintained so that the node cannot exceed the velocity of the node in front of it, if they are within the safety distance. Formally, the safety distance can be defined as the following: if $distance_{i,j} < safety\_disntace$, then $velocity_i(t) < velocity_j(t)$, if *j* is ahead of *i* in its lane.

USC Mobility Generator Software

The USC Mobility Generator [36] software implements the freeway model, also it creates the NS-2 mobility traces for a number of different mobility models, including the freeway mobility model. The source code for the USC Mobility Generator is available from the project's website for free. After the *freeway.cpp* source code is obtained, the source code must be compiled. When we execute the *freeway* program, the user will be prompted the following information:

1. The number of number within the simulation.
2. The node's acceleration speed (the software authors recommended setting the acceleration of the nodes to 10% of the maximum velocity).
3. The map's name that contains the description of the freeways.
4. The name of the file to create. (Where the NS-2 mobility trace will be written).

After these four items are specified, the *freeway* program will be executed and then writes an NS-2 mobility trace to a file that will be used afterwards to get the information. The mobility trace file can be later sourced within an NS-2 simulation script.

4.4.2 Map File

A Topography file is a critical component that determining the topology of the network we are using. Hence, the map file is defining and determining all the freeways that exist in the simulation. As a result, it is possible to model any of the freeways by supplying the correct data (in the form of a map file) to the *freeway* program.

The following is the specification of the map file format. The specification was taken from the *manual.txt* file of the USC Mobility Generator program:

```
FREEWAY

FREEWAY_NUM <total_number_of_freeways>

LANE_NUM <total_number_of_lanes>

LANE_BEGIN <freeway_id> <lane_id_in_this_freeway>

           <lane_id_in_all_freeway> <direction>

           <number_of_phases_of_this_lane>

PHASE <phase_id> (<phase_start_x0,phase_start_y0>)

                 (<phase_end_x1,phase_end_y1>) <v_min> <v_max>

PHASE <phase_id> (<phase_start_x1,phase_start_y1>)

                 (<phase_end_x2,phase_end_y2>) <v_min> <v_max>

....
```

The documentation that comes with the USC Mobility Generator thoroughly describes all of the above fields. To create a map file for the simulation you would like to run, the program *create-map.pl* was written with Perl language. The *create-map.pl* program, available in the Appendix, generates an approximation of a circular map. The inner-radius is specified for the map, and the inner-radius is the distance, in meters, from the center of the map to the inner-lane of the freeway. The default size of the inner-radius is *300 m*. Freeway lanes are then placed each *5 m* a part. Examples showing the previous explanation, if the inner-radius of the freeway is *300* m, the next lane of the freeway is *305 m* followed by another lane placed at *310 m*. Lanes are added to the freeway until the number of lanes specified for the freeway is reached.

By default, the program creates an eight-lane freeway, with four lanes traveling in each direction. Moreover, the user can change the inner-radius of the circle to decrease or increase the freeway's size. Also, the user can specify the number of points that are going to be used to construct the freeway. A map becomes more circular as more points are used to construct the map file.

The following description is a portion of a map file generated with the *create-map.pl* program:

```
FREEWAY

FREEWAY_NUM 1

$node_(0) set X_ 797.63441
$node_(0) set Y_ 537.59997
$node_(0) set Z_ 0.0
$node_(0) radius   230.81019207872924
$node_(1) set X_ 790.57495
$node_(1) set Y_ 574.60697
$node_(1) set Z_ 0.0
$node_(1) radius   230.81019207872924

      ....

$node_(4) set X_ 742.70510
$node_(4) set Y_ 676.33558
$node_(4) set Z_ 0.0
$node_(4) radius   230.81019207872924
$node_(5) set X_ 718.69059
$node_(5) set Y_ 705.36413
$node_(5) set Z_ 0.0
$node_(5) radius   230.81019207872924

      ...

$node_(20) set X_ 232.72646
$node_(20) set Y_ 646.93487
$node_(20) set Z_ 0.0
$node_(20) radius   230.81019207872924

      ...

$node_(40) set Y_ 224.02775
$node_(40) set Z_ 0.0
$node_(40) radius   230.81019207872924
```

```
      ...
$node_(0) set X_ 807.55556
$node_(0) set Y_ 538.85330
$node_(0) set Z_ 0.0
$node_(0) radius   230.81019207872924
$node_(1) set X_ 800.26078
$node_(1) set Y_ 577.09387
$node_(1) set Z_ 0.0
$node_(1) radius   230.81019207872924
      ...
$node_(8) set X_ 631.99158
$node_(8) set Y_ 780.49639
$node_(8) set Z_ 0.0
$node_(8) radius   230.81019207872924
      ...
$node_(49) set X_ 810.00000
$node_(49) set Y_ 500.00000
$node_(49) set Z_ 0.0
$node_(49) radius   230.81019207872924
      ...
$node_(34) set X_ 398.02439
$node_(34) set Y_ 813.84865
$node_(34) set Z_ 0.0
$node_(34) radius   230.81019207872924
      ...
$node_(49) set X_ 835.00000
$node_(49) set Y_ 500.00000
$node_(49) set Z_ 0.0
$node_(49) radius   230.81019207872924
```

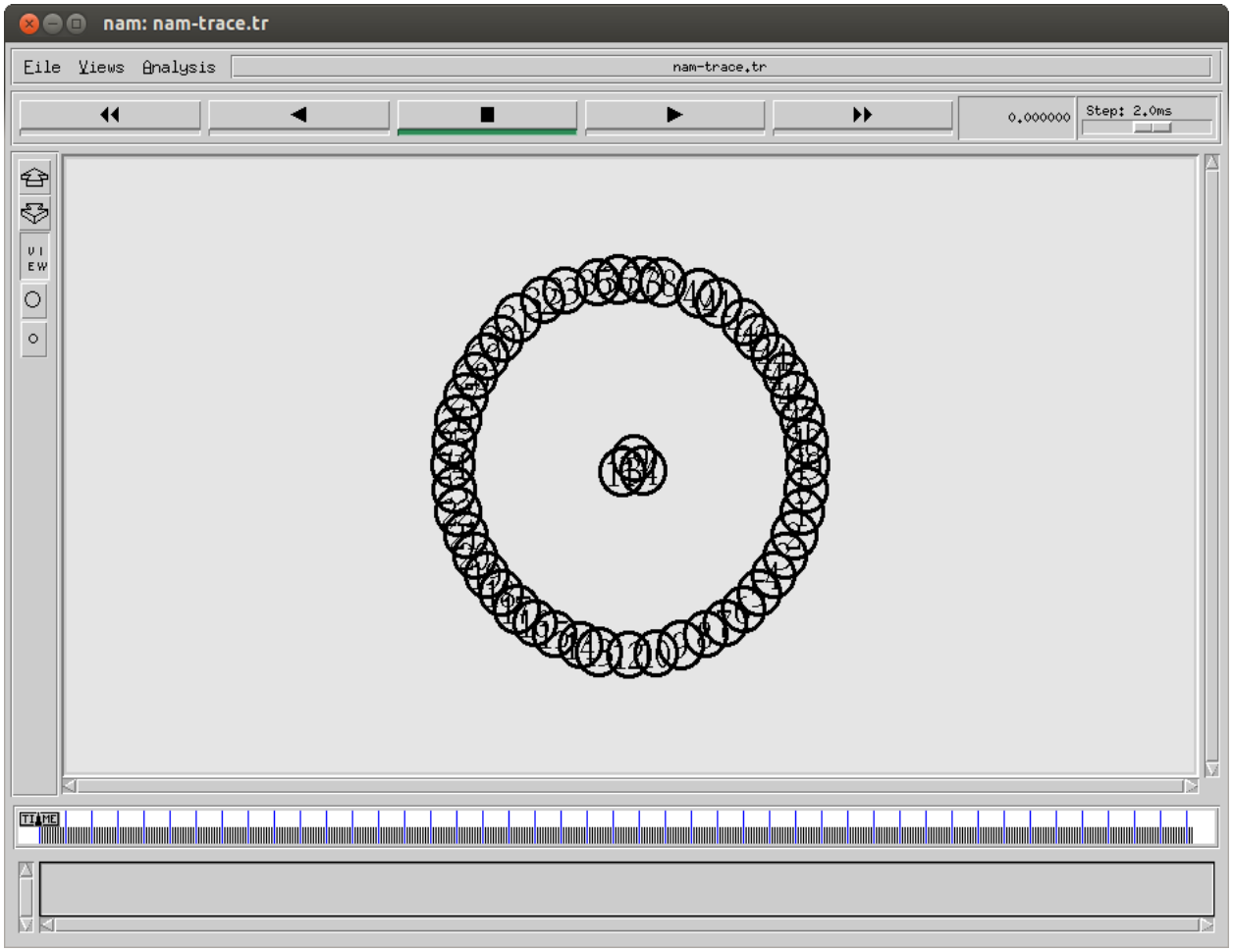


Figure 4-5: Simulation topology with the gateway in the middle.

The sample map file displayed above contains one freeway that is made up of circular road. Furthermore, in each circular path, the vehicles will be adopted to be organized in a circular way. The above values indicate these points in the map. The velocity of the vehicles on this freeway can be defined within the *create-map.pl*. For each single vehicle within the freeway, it is possible to assign different values to the $velocity_{min}$ and $velocity_{max}$. Within that case, the minimum and maximum velocity are set throughout this freeway by these values: $velocity_{min}$ is *17.0* m/s and $velocity_{max}$ is *25.0* m/s. Figure 4-6 contains a screen shot from the NAM program when the defined map is used for our simulation.

The *create-map.pl* starts by providing each node an initial x, y, or z coordinate. As the current implementation of NS-2 version 2.35, the z coordinate has no effect on the simulation, but the z coordinate must be mentioned. After the node's initial positions are set, the nodes are then put into motion. In this case, node (0) is set to travel towards the x coordinate `797.63441` and the y coordinate `537.59997` at velocity of approximate *17 m/s*. Node(0) will continue to travel to its destination until periodically node(0)'s destination and velocity will be altered by the trace generated to send it to a new destination point.

The file provides the movement of the mobile node within the simulation. The radius of the freeway was made large enough so that the transmission on one-side of the road would not affect a concurrent transmission on the other side of the road. Also, the diameter of the inner lane of the freeway is *600 m*, and the circumference of the inner-lane is approximately *1880 m*. One difference that was considered was the portion of the road to model for that simulation. If the diameter of the road was increased, more nodes would be considered and added to the network to have the same amount of the network traffic that would result in increasing the length of the simulation.

4.5 Network Traffic

Different kinds of networks exist, and the traffic pattern of each of the existing networks differs from one to another. Moreover, the simulation's traffic pattern should resemble the actual VANET's traffic.

Most network favor certain applications and most network application favor the use of a specific transport protocol. For example, the majority of the traffic on one network may use TCP while another network found maybe use the UDP as a primary traffic. The transport protocols used will affect the traffic profile of the network. Another example is that the transport protocol used to deliver VoIP messages in a VANET is UDP. Because of the nature of the VoIP traffic, no transport-layer connection is established before transmitting.

The network applications used also affects the traffic profile. For instance, some applications have bursty traffic, while other applications provide a constant stream of packets to the network. Typically, applications that use the HTTP protocol have bursty traffic, and VoIP applications transmit a steady stream of bytes.

The NS-2 simulator comes with a number of stand-alone programs that generate simulation traffic. The problem with using the applications provided by NS-2 for creating network traffic is they only create *unicast* traffic. The program *cbrgen.tcl* is one of the famous programs for creating network traffic. For instance, the *cbrgen.tcl* program creates random TCP/UDP connections between nodes and uses the FTP application protocol for generating constant bit rate traffic in the TCP connection for example. In case of a VANET, TCP connections are usually not established before application data is transmitted. For this reason, an additional piece of application was created *traffic-w-jitter.pl* to generate the network traffic for the simulations.

4.5.1 Creating Network Traffic

Agent are got attached to the node after creating that node in the simulation script. Agents represent endpoints where network layer packets are constructed and consumed, and they are used in the implementation protocols at various layers [37]. A number of different agents exist for ns-2 including UDP Agent, TCP Agent, Null Agent, MessagePassing Agent, etc. To create a new TCP agent, the next piece of code is used:

```
# Add Transport agents

set udp [new Agent/UDP]

$ns_ attach-agent $n(0) $udp

set udpsink [new Agent/LossMonitor]

$ns_ attach-agent $n(1) $udpsink

$udp connect $udpsink
```

The code listed above creates a new UDP agent object and establishes a connection between the nodes *n(0)* and *n(1).*

After a transport agent is attached to the node, an application agent is then typically attached to a transport agent. Some of the application agents that are available for NS-2 are CBR, FTP, Worm, etc. The following code is used to attach a CBR application to a UDP agent:

```
# Add application

set cbr [new Application/Traffic/CBR]
```

```
$cbr set packetSize_ 128
$cbr attach-agent $udp
```

After the CBR application is attached to a transport agent, a node can then schedule the application.

In terms of the VANET simulation, dealing with the streaming options, the MessagePassing agent is one of the agents that can be used for sending and transmitting the packets requesting the streaming services. The benefit of using this agent is that the MessagePassing agent allows the users to specify both the address and a port for a transmission and a reception.

For the simulations, the port number to which a packet is sent provides a way to determine what application sent which packets.

The MessagePassing agent is the base agent class used in the simulations. It is an extended class, and a new class is created for each access category used in the simulation. For instance, the class VoIP extends the MessagePassing class. The VoIP class defines two methods, *recv* and *send_voice*, that are used to receive and transmit voice.

The following code defines a new class, VoIP that extends the base class Agent/MessagePassing:

```
# Voice class used to send Voice
# messages to a node(s) “gateway” one-hop neighbors.
Class Agent/MessagePassing/Voice
-superclass Agent/MessagePassing
Agent/MessagePassing/ Voice instproc recv
                          {source sport size data} {
# This empty function is needed so receive works.
}


Agent/MessagePassing/ Voice instproc send_message {} {
$self instvar node_
global ns_ Voice _PORT ValidPort_ADDR opt
```

```
# send the Voice message
$self sendto $opt(pkt_size_emer) 0 $Voice_ADDR $VoIP_PORT
}
```

The following code creates Voice agents and attaches an agent to each of the node in the simulation. Each node attaches the agent to a determined port, in this case *Voice_port* is port 17.

```
# Attach a new Agent/MessagePassing/Voice
# to each node on port $Voice_PORT, 17
for {set i 0} {$i < $opt(nn)} {incr i} {
set em_agent($i) [new Agent/MessagePassing/Voice]
$em_agent($i) set class_ 0
$em_agent($i) set prio_ 0
$node_($i) attach $em_agent($i) $Voice_PORT
}
```

4.5.2 The Network Traffic Programs: *AWK* throughput latency scripts

Because there is no standard NS-2 program that creates network traffic identical to a VANET's traffic profile, the programs *throughput.awk and latency.awk*, listed in the Appendix, was created to calculate the network performance in regards to its throughput and latency.

The programs *throughput.awk and latency.awk* use the variables of time in which these variables determine the duration of the transmission for the simulation. The result of the execution of this program is a file containing all the commands to generate network traffic performance for the throughput and the latency of the simulation.

The following result was the generated traffic file after executing the "*wireless-simulation.tr*":

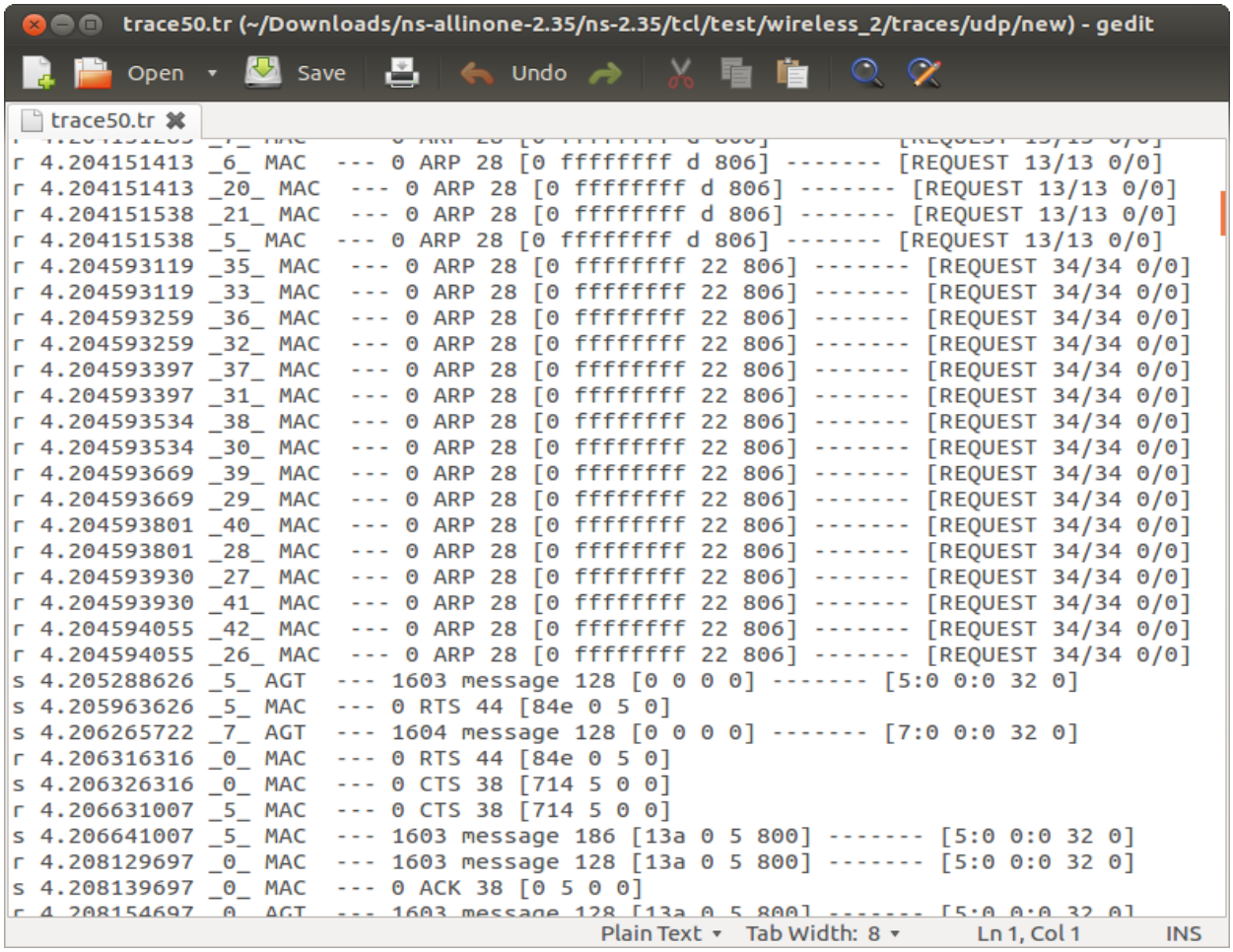


Figure 4-6: Part of the trace file as an output of the tcl script.

The next outputs "*throughput* and *latency of the network*" were generated by running the *throughput.awk and latency.awk* scripts on the pervious trace file, and it results the following:

throughput50 (~/Downloads/ns-allinone-2.35/ns-2.35/tcl/test/wireless_2/throughput) - gedit

Open Save Undo

throughput50

```
0.100000 0.000000
0.200000 0.000000
0.300000 0.000000
0.400000 0.000000
0.500000 0.000000
0.600000 0.000000
0.700000 0.000000
0.800000 0.000000
0.900000 0.000000
1.000000 0.000000
1.050094 0.001577
1.105015 0.003277
1.155453 0.013759
1.206741 0.015442
1.257024 0.028996
1.308272 0.036529
1.360030 0.055334
1.410475 0.067778
1.460722 0.093478
1.512531 0.104537
1.563246 0.148594
1.613565 0.155726
1.663606 0.207193
1.714150 0.212169
1.764521 0.255785
1.815697 0.274465
1.865794 0.346647
1.922033 0.316809
1.972561 0.426791
2.022966 0.437717
```

latency50 (~/Downloads/ns-allinone-2.35/ns-2.35/tcl/test/wireless_2/latency) - gedit

Open Save Undo

latency50

```
0.000000 1.000920
1.000920 1.000920
2.001840 1.000921
3.002761 1.001978
4.004740 1.001979
5.006718 1.001979
6.008697 1.001979
7.010676 1.002030
8.012706 1.004163
9.016869 1.012079
10.028948 1.012080
11.041028 1.012720
12.053747 1.012720
13.066467 1.012720
14.079187 1.014200
15.093386 1.014871
16.108257 1.014871
17.123128 1.014871
18.137999 1.014871
19.152871 1.019562
20.172433 1.019563
21.191996 1.019563
22.211559 1.019563
23.231122 1.020090
24.251211 1.020404
25.271615 1.021102
26.292718 1.021417
27.314134 1.022596
28.336730 1.022596
29.359326 1.022596
```

Plain Text Tab Width: 8 Ln 1, Col 1 INS

Figure 4-7: Throughput and latency trace files

4.6 Performance Metrics

A metric provides a standard measure for accessing the performance of a specific subject. To evaluate the simulations, metrics are needed to determine the effectiveness of the used protocols to send and receive voice and/or video as well as other data. The two quantitative metrics [Throughput and Latency] are used to evaluate the performance of the simulation found.

Some details about the Throughput and the Latency can be found in the following table.

Table 4.6.1: Performance Metrics

Metric	***Description***
Throughput **Latency**	-The percentage of packets successfully received at a specific distance. -The amount of time it takes from when a packet is passed down the requesting entity until it is placed on the gateway.

Throughput measures the percentage of packets successfully received at a given distance. Because of the unreliable nature of the voice/video transmission, a percentage of the frame transmitted will fail to be delivered. As discussed before, the probability that a frame will be received is affected by the distance between the sender "vehicle" and the receiver "gateway". Therefore, the distance between the requesting entity and the gateway is considered when the reception rate is calculated. As the distance between nodes increases, the probability of reception decreases. The metric reception rate is used to determine how far a protocol's performance is from the best case scenario.

Latency measure the amount of time it takes from when a packet is passed down from the requesting entity until it got reached the gateway.

Two *AWK* scripts are created, *throughput.awk* and *latency.awk*, for the purpose of evaluating the simulations. The program *throughput.awk* is used to define the overall reception rates at specific distance. Furthermore, the program *latency.awk* measures the access delay for the simulation. At the completion of the simulation, the programs *throughput.awk* and *latency.awk* parse the simulation trace and calculate the performance metrics.

4.6.1 NS-2 Simulation Trace Format

The simulation results are written to a trace file. To measure the quantitatively the performance of the network protocol used, the appropriate data must be extracted from this trace file. Consequently, the format of the trace file follows a specific style with fields being placed in specific locations and the "NS-2 manual" describes in detail the format of a trace [43].

An *ns-trace* is divided into fields; each trace field is delimited by an obvious white space. As a result of the presence of the separation of the fields by white space, the file will be easily parsed with a language such as *Perl* or *AWK*. Also, to simplify parsing a trace, each trace field is in a specific location. For example, the first trace field records the type of the event that is occurring. Some of the valid simulation events are: ***s*** for send, ***r*** for receive, ***d*** for drop, and ***M*** for mobility. Moreover, the second field records the time that the event occurred. In addition, the third field is the node ID that is used to uniquely identify each node. The format of data contained in the trace file is described in the NS-2 documentation. Numerous other fields, that can be extracted, also contain information about the simulation. While some of the field's content is always the same (e.g., the second field always contains the time of an event), the content of many of the fields varies based on the type of the event. For instance, the fields after the third column have different meanings for mobility events as compared to send events.

The following is a sample output from a trace:

```
...
M 1.00000 48 (1178.05, 245.00, 0.00), (1200.69, 245.00), 22.64
M 1.00000 49 (2681.49, 255.00, 0.00), (2660.85, 255.00), 20.64
s 1.001393522 _34_ AGT --- 0 message 200 [0 0 0 0] ------- [34:42 -1:42 32 0]
s 1.001488522 _34_ MAC --- 0 message 260 [0 ffffffff 22 0] ------- [34:42 -1:42 32 0]
s 1.003523553 _37_ AGT --- 1 message 200 [0 0 0 0] ------- [37:42 -1:42 32 0]
r 1.003568779 _9_ MAC --- 0 message 200 [0 ffffffff 22 0] ------- [34:42 -1:42 32 0]
r 1.003568890 _1_ MAC --- 0 message 200 [0 ffffffff 22 0] ------- [34:42 -1:42 32 0]
...
```

4.6.2 Reception Rate

The program *throughput.awk,* listed in the Appendix, calculates the reception rate metric for each access category involved in an NS-2 simulation. In terms of the calculation of the reception rate, the distance between the nodes and the gateway in the simulation must be maintained. The reception rate of an access category is the percentage of the packets that are successfully received at a specific distance from the sender for the length of the simulation. In addition, reception rate is evaluated on the percentage of packets received at certain intervals.

The following is a sample of the output generated from the execution of the *AWK* script *throughput.awk* program (moreover, the *--file* switch is used to specify the name of the file to save results in a format to easily generate a graph)

```
./throughput.awk --file=throughput.awk trace50.tcl

--------------------------------------------------------------------------

Number of nodes 50

--------------------------------------------------------------------------
1.154210 0.012875
1.206741 0.015077
1.257024 0.028996
1.308272 0.036529
1.360030 0.055334
1.410475 0.067778
1.460722 0.093478
1.512531 0.104537
1.563246 0.148594
1.613565 0.155726
....
2.128669 0.520727
2.180389 0.604732
2.230531 0.684612
2.284945 0.676777
2.335876 0.813641
2.386517 0.844861
2.436932 0.918861
2.487327 0.976639
2.539740 1.011423
2.589993 1.116604
....
3.259042 2.173040
3.311033 2.270844
3.363914 2.409066
```

```
3.414016 2.566810
3.464476 2.741328
3.514562 2.786706
3.567352 2.835908
3.617489 3.063929
3.667984 3.199017
3.720016 3.182816
....
4.029718 4.030891
4.080281 4.219450
4.130424 4.410329
4.181370 4.464330
4.239665 4.039495
4.292671 4.564871
4.342709 5.067826
4.392724 5.137279
4.444738 5.170435
4.498722 5.045807
....
```

Within an NS-2 simulation script, a user can turn on the tracing mobility events and specify how often the location of a node is written into the trace. The more often a node's position is written within the trace, the more accurate the position is when determining the location of the node. The drawback of frequently updating the location of a node is that it increases the run-time of the simulation being held. For the simulations, the node's locations are written to the trace file every *100 ms*. As a result, the location of the node within the topography must be maintained when calculating the rate of the reception. Each single line that starts with an ***M*** specifies and signifies a mobility event. When a trace line starts with an ***M*** is read, the node location is updated within the program *reception.pl*

When the *reception.pl* program begins the execution, the trace file is first opened. To open the trace, the command-line option --trace is performed. After the trace is opened, the program starts to read and analyze each line within the file. The only events of interest are those that occur at the *s* for send, *r* for receive and *d* for drop.

A line starting with a *s* indicates that a packet is sent. For each packet sent from the node, the neighboring nodes within the transmission range of the sender are recorded. When a packet is transmitted, the ID of the packet and the gateway that should receive the packet are recorded. In addition, a line that begins with a *r* signals that a packet was received. When a packet is received for a packet ID, the receive count at the distance between the nodes and the gateway is

incremented by one. Also, a *d* indicates that a packet was received in error, the percentage of packets successfully received at each distance is calculated, and the results are displayed to the user for the interpretation process.

4.6.3 Delay

The program *latency.awk*, listed in the Appendix, calculates the delay metric for the simulation. A number of different access categories may be used in the simulation, so the access delay must be determined for each access category. The program, *latency.awk*, calculates the average access delay and the number of frames dropped for each access category.

The following is a part of the output generated from the execution of this program:

```
./latency.awk --file=throughput.awk trace50.tr
1.001380 0.000000
1.001380 1.001380
1.001381 2.002760
1.002158 3.004141
1.002159 4.006300
1.002159 5.008458
....
2.273925 2222.044764
2.276815 2224.318689
2.276815 2226.595504
2.276815 2228.872318
2.276815 2231.149133
....
3.120341 4618.551937
3.120341 4621.672278
3.120341 4624.792619
3.120699 4627.912960
3.125947 4631.033659
....
4.814757 11936.560486
4.815071 11941.375243
4.816570 11946.190315
4.816595 11951.006884
4.816884 11955.823479
....
```

Upon execution of the *latency.awk*, the program begins by opening a trace file, and starts reading each line of the trace file. The only events of interest in determining the delay are send events (e.g., trace lines that starts with *s*). Each line contained in a trace is read by the *latency.awk* program. If a line begins with *s*, the event for the line starting with an *s* is processed by the program.

4.7 Simulation Results And Analysis

To determine the effectiveness of the gateway we proposed in Chapter III, a number of simulations were conducted. The simulations compared the performance of gateway for providing its services against the ratio of requests from the requesting entities found within the architecture.

The simulations used the freeway mobility model, which was discussed in Section 4.4. In addition, the metrics used to evaluate the performance of such gateway were those discussed in Section 4.6. Also the network traffic was generated with the *traffic-w-jitter.pl* program that was discussed in Section 4.5. Furthermore, the length of the VANET simulations was set to *60 s*.

A number of different simulations were conducted. Each simulation varied the number of nodes contained within the freeway. By increasing the number of nodes, one is able to observe the effect of increasing the traffic of the network has on the performance of the gateway. The simulation varied the number of nodes in the simulation by *50* nodes, and simulations were executed for the freeway containing the following number of nodes: *50*, *100*, *200* and *400*.

Simulation for *50* Nodes

For the simulation conducted with *50* nodes, this configuration results in a relatively sparse network. For this reason, modifying the rate of requests has a minor impact on both of the reception and response rates of the gateway.

In this simulation, because not too many nodes are trying to simultaneously send requests to the gateway, the access delay remains short. Overall, there is no big difference in the gateway's reception rate as the reception rate was already fairly well.

Simulation for *100* Nodes

Figure 4-8 contains the reception rate for a simulation with *100* nodes. In this simulation, the reception rate of the gateway slightly increases. One surprising finding is the reception rate actually became worse at large distances between nodes "vehicles" and the gateway. The access category of the gateway maintained an acceptable access delay and the performance of this aspect of the simulation is acceptable too.

Simulation for *200* Nodes

Figure 4-9 contains the reception rate for a simulation with *200* nodes. The results of the simulation for *200* nodes were barely near to the simulation for *100* nodes. The reception rate increased for the gateway access category when vehicles maintain distance closer to it. One thing that is observable from the simulation is that as the traffic on the network increases, the difference in the reception rate between access categories increases. The access delay for all of the access categories remained acceptable.

Simulation for *400* Nodes

A final simulation was conducted with *400* nodes. This simulation models the case where the network becomes fully saturated with network traffic "requests and responses". The results of the reception rate for the simulation were worse than that of the simulations carried out with *200* nodes. And in terms of delay, responses from the gateway to these *400* nodes are the worst amongst the other simulations.

The following figure groups the average throughput of each of the carried simulations (*50, 100, 200* and *400*). It indicates that the average throughput of the gateway for handling *50* nodes "vehicles" is the best among other numbers of nodes. As the number of nodes is small, they could be served better and get the aimed services quickly with a small delay. Also it shows that the performance of the gateway degrades when serving *400* nodes "vehicles".

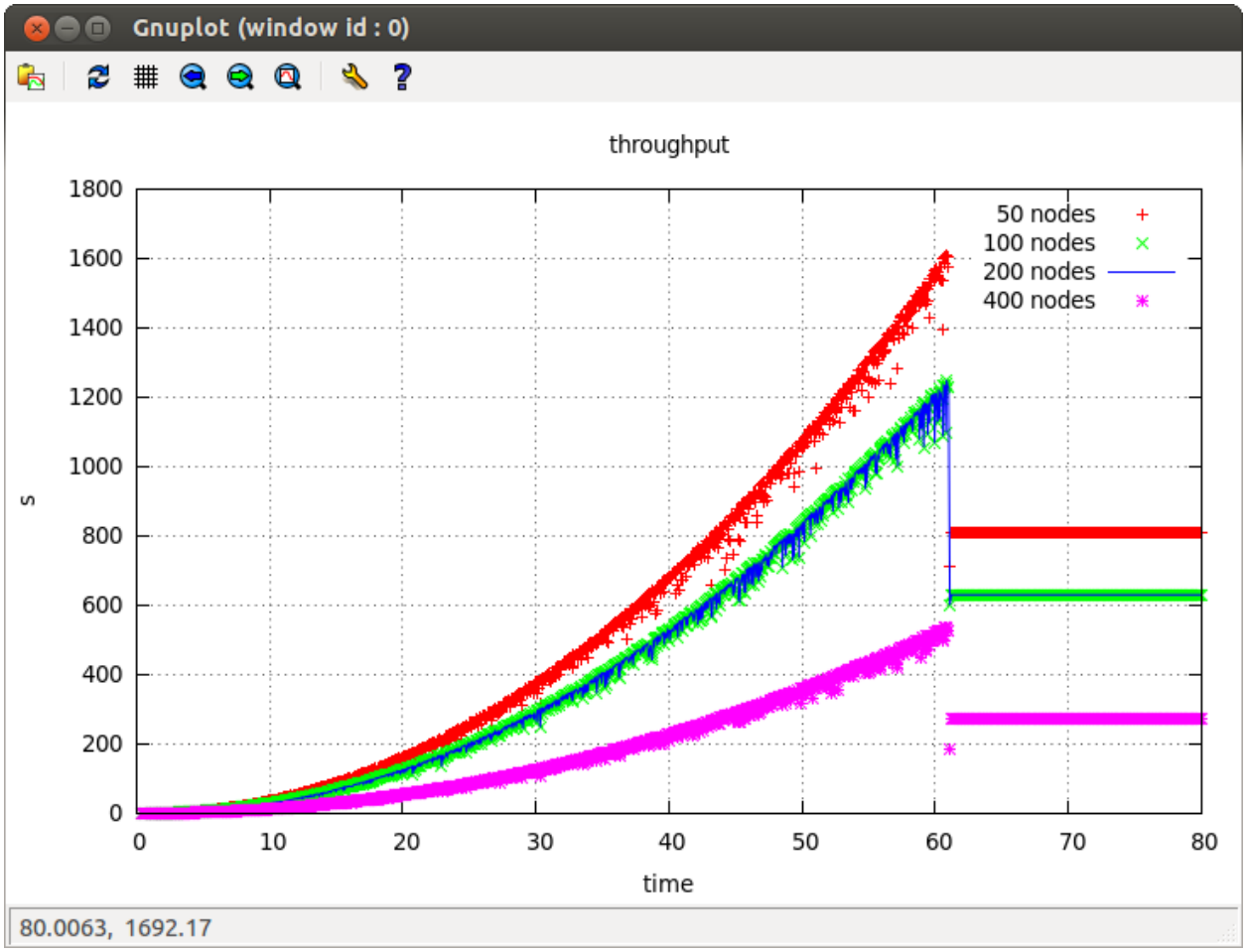


Figure 4-8: Aggregated throughput for all the simulated nodes.

The next figure shows the average latency between the gateway and the nodes to be serviced with the real time application(s). It shows that the *50* nodes have a small latency to get the requested service(s) compared with the case of the *400* nodes, this is obvious because serving small number of nodes will result in a shorter time rather than that of serving larger number of nodes.

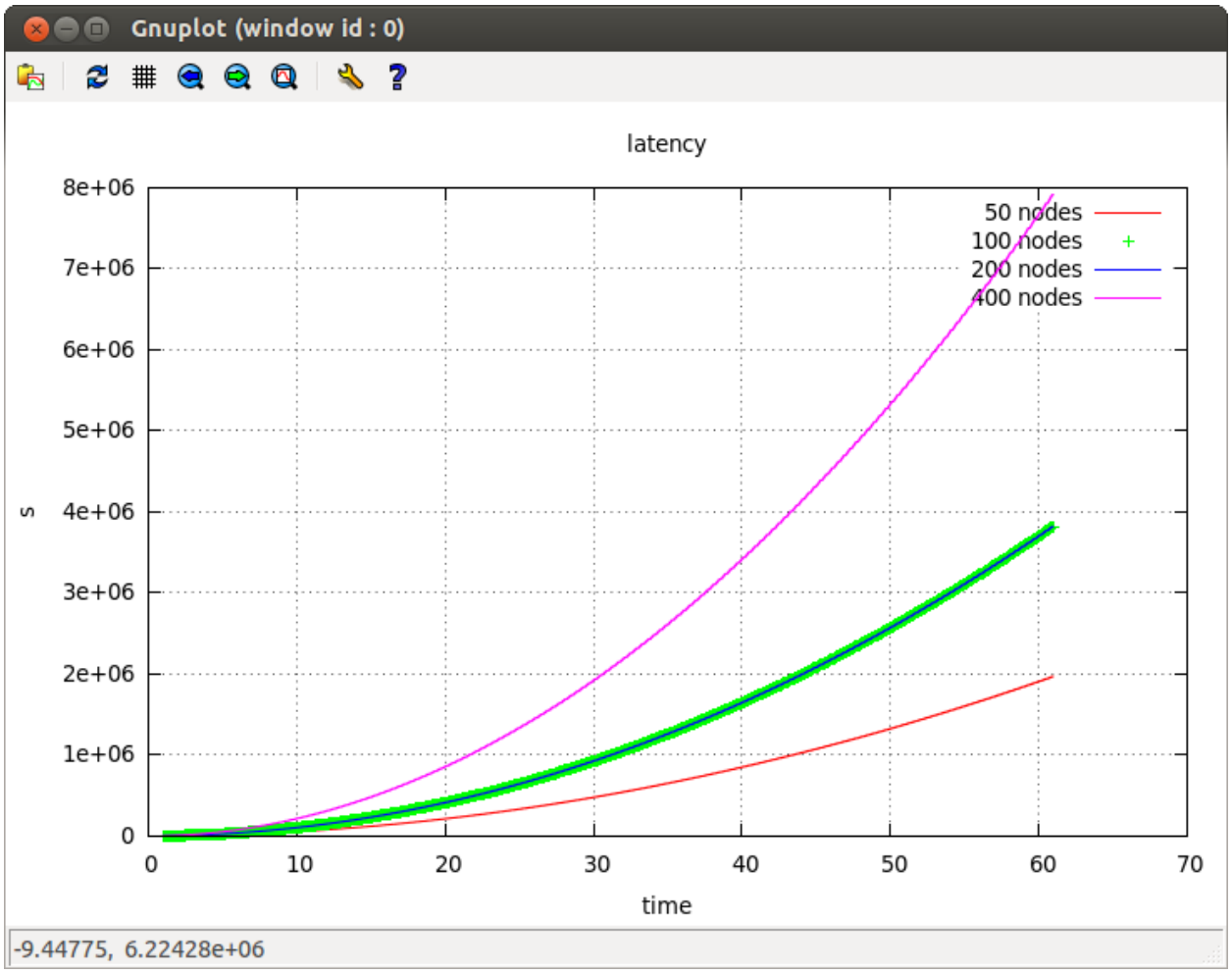


Figure 4-9: Aggregated latency for all the simulated nodes

CONCLUSION

In this dissertation we have proposed an architecture that services as a gateway to connect VANETs, IMS and WSNs. This architecture allows the provisioning of new services to users "vehicles" and an optimized and more efficient use of the multimedia services collected in the IMS as well as the information collected in the WSN.

The main contribution opens the door to a wide set of current and future applications in IMS to be provided to the vehicles via the presence of the gateway.

This last chapter presents a summary of the work that has been exposed in this project. Later on, future applications and possible extensions to overcome the limitation of the current proposal are described.

5.1 Summary of the work

Gateway has been proposed as a solution to integrate the three different mentioned networks and improve the overall VANET life time by implementing efficient data collection and data querying among the IMS and WSN.

This gateway enables applications such as multimedia services, environmental monitoring, accidents detection and prevention, etc., to be transmitted to the moving vehicles and hence make good use of collected services. In the scenarios exposed before, for instance, the data gathered is important and useful if it can be accessed in real-time. Thus, it could be further enhanced by using IMS enablers such as messaging and multimedia sessions. Our architecture was designed based on several requirements defined in chapter two and obtained from the analysis of the scenarios stated above in chapter III.

This proposal also not only offers the possibility to access information from one or more sensors but it also stores and processes data when required (in case of dealing with the information presented by WSN).

Moreover, during the design process, independence was constantly considered to ensure minimal changes in the involved technologies or when proposing future extensions. Independency is presented in the architecture from three different perspectives; firstly no changes are needed from the sensors in the WSN and/or vehicles side and only minimal are required in the presence services.

Secondly, we propose the use of an independent information model to transport data inside the gateway. And finally, the architecture was designed in components which make the proposal easily flexible to be overlay middleware-and protocol-independent.

Moreover, the gateway here proposed considers factors such as scalability and fault-tolerance; which have hardly been considered in current proposals.

The integration with IMS was done through and already existing IMS service, the presence framework. This service already offers some basic contextual information and its information model (i.e. PIDF) was extended to support the non-user related data that could be recovered from a cellular network or a WSN.

To prove our concept, the architecture for a peer in the overlay was designed and a prototype based on this architecture was implemented. Also it was mathematically proved using Matlab and then Network Simulator 2 (NS-2).

The creation of a gateway to use the sensing capabilities from WSNs and the requests from vehicles in the presence service exposed in the IMS architecture opens the door to a wide range of applications and other services in IMS, providing effective and real-time feedback to users. Furthermore, IMS services can be also connected with the gateway and receive VANET and WSN requests and/or information through several PSEs.

Additionally, the architecture of the gateway in which one or more cellular networks cooperate to deliver information "multimedia services", gives better results regarding self-recovery and scalability.

5.2 Publications

This work done in this project has been submitted to the 9th International Wireless Communications and Mobile Computing Conference (IWCMC 2013), Cagliari, Sardinia, Italy – and it's under review right now. The conference will be held on July 1st to July 5th, 2013.

5.3 Limitation of the work

The work that has been proposed presents several limitations that should be taken into consideration when using this proposal and when defining a future research path. Initially, the gateway extensively relies on a data dissemination protocol to provide the status of the disseminated information. If the protocol does not offer such information, it automatically

becomes useless in the architecture. Secondly, we did not pay too much attention about the topology being used inside the simulations; we used the simplest topology which is the circular way/road topology where the vehicles are adopted and organized in a circular motion and that the simulations should also be focused on the performance analysis of the protocol in real scenarios (i.e., rural, city, urban and highways) in order to see the efficiency of such gateway on the rate of requests it faces from each of the VANET users and WSNs subscribes. Moreover, we have to support different speeds of the vehicles while being on the road so we can adopt the different changes found while being on the road.

Furthermore, in this architecture, we put an assumption that each type of the tunneling layers found that receives a message has a way to send back an acknowledgement about the successfully or unsuccessfully reception. If this functionality in such types is missing, the delivery cannot be guaranteed.

5.4 Future Works

At this stage, our proposal only includes a procedure of self-recovery for voluntarily departure. A future extension would include the procedure when a node (vehicle, sensor or ingress spot) fails. The recovery process should consider the different roles present in the overlay, since the actions could differ from one another. This extension will increase the fault-tolerance of the architecture, complementing the already defined self-recovery process.

The implemented prototype that was presented in this thesis is a subset of the architecture. A future phase will be extending they prototype to support more functionalities.

Processing of information should be further detailed to determine when and how data should be sent to the Data Management Unit and the processor group. The storage process in case of WSN systems should also be further analyzed, data distribution is especially important is these types of networks since it should be determined where data is stored and how efficiently retrieve it.

Flexibility when considering these new functions should be constantly present, since each service could change depending on the target application. It is important to define a clear way of indicating when information should be stored, processed or published and where should the retrieval process takes place (e.g. from the stored information or directly from the cellular network or the WSN).

In a final phase the efficiency of the proposed architecture concepts will be verified in the field test.

The previous is applicable in case of the WSNs, moving to the VANET world. The following are some areas of future work to improve the performance of the gateway and the UDP protocol for a VANET system:

- **Adaptive transmission rate,** An algorithm that throttles the rate at which the vehicle's state is transmitted.
- **Adaptive transmission range,** the transmission range can be adjusted to keep the network load on the channel below a certain threshold.
- **Extensive mathematical and simulation mode,** larger model could be developed to determine the maximum improvement that can be expected from adjusting the gateway and the rate of its connecting the entities and the transmission between the requesting node and the other party that provides the requested service(s).

5.4.1 Adaptive Transmission Rate Control

Due to the hidden terminal problem and other interference, it is unrealistic to achieve a 100% delivery rate without retransmission in a wireless network even for example Safety messages typically need to be repeatedly transmitted at a certain rate to ensure reliable delivery.

If the network is highly loaded, the rate of transmission should be decreased eventually within the network regarding the requests from the vehicles side. The authors of [38] propose the VCWC protocol which is only based on application-specific properties to help controlling the channel congestion and when the network seems to be kind of free (i.e., after some calculations are done), the algorithm will notify the nodes to augment the rate of requests whenever they would like to.

We will utilize the aforementioned channel feedback, packet collision rate and number of nodes within the transmission range, to effectively adjust the transmission rates for all the traffic classes. For example, when the packet loss rate is larger than some threshold, we then first minimize the transmission rate of all traffic classes. If this is not enough, we will further drop all non-safety related messages and reduce the transmission rate of low priority safety messages, and so on.

5.4.2 Dynamic Transmission Power Control

Controlling the communication range between the gateway and the requesting entity, by adjusting the power of the transmission, can be used to mitigate the adverse effects caused by nodes being densely populated. The choice of the communication range has a high impact on a fundamental property of an ad-hoc network, the connectivity. In a VANET, a static transmission range cannot maintain the network connectivity due to the non-homogenous conditions. It is shown in [39, 40] that a dynamic transmission range is needed to maintain connectivity in non-homogenous networks to take advantage of power saving and increased capacity.

The transmission range of all nodes can be adjusted, using the power control, to keep the network load below a certain threshold. By adjusting the transmission range to the minimum range required by a multimedia (i.e., real time application), the load on the channel can be reduced as a result of having highly accurate information of gateway.

In a final phase, the efficiency of the proposed architecture concepts will be verified in the field test.

REFERENCES

[1] G. Camarillo and M. A. Garcia-Martin, *The 3G IP multimedia subsystem (IMS): merging the Internet and the cellular worlds*: Wiley, 2011.

[2] M. V. Pulgarin, R. Glitho, and A. Quintero, "An Overlay Gateway for the Integration of IP Multimedia Subsystem and Mobile Sink Based-Wireless Sensor Networks," in *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, pp. 1-5, 2010.

[3] I. Khan, "Performance evaluation of Ad hoc routing protocols for Vehicular ad hoc networks," *Mohammad Ali Jinnah University*, 2009.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *Communications Magazine, IEEE*, vol. 40, pp. 102-114, 2002.

[5] (March, 2012). *Wikipedia, IP Multimedia Subsystem*. Available: http://en.wikipeida.org/wiki/IP_Multimedia_Subsystem#Media_Servers

[6] "3GPP TSG SA WG3 Security-S3#31, MMS Security Considerations," ed.

[7] G. Camarillo and M. A. Garcia-Martin, *The 3G IP Multimedia Susbsystem (IMS): Merging the Internet and the cellular worlds, Second edition Wiley, Copyright 2006,*, 2006.

[8] M. M. I. Taha, "Broadcasting Protocols in Vehicular Ad-Hoc Networks (VANETs)," MSc., Electrical Engineering, Assuit University, 2008.

[9] T. Taleb, E. Sakhaee, A. Jamalipour, K. Hashimoto, N. Kato, and Y. Nemoto, "A stable routing protocol to support ITS services in VANET networks," *Vehicular Technology, IEEE Transactions on*, vol. 56, pp. 3337-3347, 2007.

[10] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer networks*, vol. 52, pp. 2292-2330, 2008.

[11] Y. W. Lin, J. M. Shen, and H. C. Weng, "Gateway Discovery in VANET Cloud," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pp. 951-954, 2011.

[12] M. El Barachi, A. Kadiwal, R. Glitho, F. Khendek, and R. Dssouli, "The design and implementation of a gateway for ip multimedia subsystem/wireless sensor networks interworking," in *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, pp. 1-5, 2009.

[13] A. Gluhak and W. Schott, "A WSN system architecture to capture context information for beyond 3g communication systems," in *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, pp. 49-54, 2007.

[14] (March, 2012). *Once upon a release - IP Multimedia Subsystem (IMS) architecture story*. Available: http://ictbackyard.com/archives/tag/ims

[15] R. Martínez García, "Diseño y desarrollo de una CNG orientado a Linux Embedded," 2011.

[16] G. Camarillo, M. A. García-Martín, and M. A., *The 3G Multimedia Subsystem [online]: Merging the Internet and the Cellular World. United Kingdom: John Wiley & Sons Ltd.*, 2004.

[17] G. Grilli, "Data dissemination in vehicular networks," PhD thesis University of Rome" Tor Vergata", Rome, Italy, 2010.

[18] Z. Yang, M. Li, and W. Lou, "Codeplay: Live multimedia streaming in vanets using symbol-level network coding," in *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pp. 223-232, 2010.

[19] A. Festag, A. Hessler, R. Baldessari, L. Le, W. Zhang, and D. Westhoff, "Vehicle-to-Vehicle and Road-Side sensor communication for enhanced road safety," in *Proceedings of the 15th World Congress on Intelligent Transport Systems*, 2008.

[20] J. Burrell, T. Brooke, and R. Beckwith, "Vineyard computing: Sensor networks in agricultural production," *Pervasive Computing, IEEE*, vol. 3, pp. 38-45, 2004.

[21] M. Strohbach, J. Vercher, and M. Bauer, "A case for IMS," *Vehicular Technology Magazine, IEEE*, vol. 4, pp. 57-64, 2009.

[22] S. Arbanowski, L. Lange, T. Magedanz, and L. Thiem, "The dynamic composition of personal network services for service delivery platforms," in *Circuits and Systems for Communications, 2008. ICCSC 2008. 4th IEEE International Conference on*, pp. 455-460, 2008.

[23] Z. Papp, C. Brown, and C. Bartels, "World modeling for cooperative intelligent vehicles," in *Intelligent Vehicles Symposium, 2008 IEEE*, pp. 1050-1055, 2008.

[24] J. B. Kenney, "Dedicated short-range communications (DSRC) standards in the United States," *Proceedings of the IEEE*, vol. 99, pp. 1162-1182, 2011.

[25] (April, 2012). *Traveller Information Services Association*. Available: http://www.tisa.org

[26] T. Taleb and A. Benslimane, "Design guidelines for a network architecture integrating vanet with 3g & beyond networks," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pp. 1-5, 2010.

[27] N. K. Warambhe and S. Dorle, "Implementation of Protocol for Efficient Data Storage and Data Dissemination in VANET," *International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)*, vol. 1, pp. 65-71, 2012.

[28] (March, 2012). *"RFC3265"*. Available: http://www.ietf.org/rfc/rfc3265.txt

[29] (March, 2012). *"RFC2976"*. Available: http://ietf.org/rfc/rfc2976.txt

[30] S. Helou, A. Quintero, and F. Khendek, "Architecture for the architecture for the reactive discovery and integration of WSNs and their services with IMS," unpublished|.

[31] A. Outtagarts and O. Martinot, "iSSEE: IMS Sensors Search Engine Enabler for Sensors Mashups Convergent Application," *International Journal of Computer Science Issues, IJCSI*, vol. 6, pp. 1-7, 2009.

[32] R. P. Gupta, V. K. Sharma, and V. M. Shrimal, "Investigation of Different Parameters of Dynamic Source Routing with varied Terrain Areas and Pause Time for Wireless Sensor Network."

[33] D. B. J. D. A. Maltz and J. Broch, "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks," *Computer Science Department Carnegie Mellon University Pittsburgh, PA*, pp. 15213-3891, 2001.

[34] S. Panda and R. Mohapatra, "Implementation and Comparison of Mobility Models In Ns-2," 2009.

[35] K. Fall and K. Varadhan, "the VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC, the NS Manual," ed.

[36] F. Bai, N. Sadagopan, and A. Helmy, "User manual for important mobility tool generators in NS-2 simulator," *University of Southern California,* 2004.

[37] "The NS manual (formally NS notes and documentation)," ed, 2006.

[38] X. Yang, L. Liu, N. H. Vaidya, and F. Zhao, "A vehicle-to-vehicle communication protocol for cooperative collision warning," in *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pp. 114-123, 2004.

[39] J. Gomez and A. T. Campbell, "A case for variable-range transmission power control in wireless multihop networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, pp. 1425-1436, 2004.

[40] R. Ramanathan and R. Rosales-Hain, "Topology control of multihop wireless networks using transmit power adjustment," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, pp. 404-413, 2000.

APPENDIX 1– Simulation Script

1.1 wireless-simulation.tcl

```
#======================================================================
#
#  Wireless simulation, a simulation of a UDP (and/or broadcast)
#  protocol
#  for vehicular ad hoc networks (VANETs).
#
#  Nathan Balon
#  University of Michigan - Dearborn
#  CIS 695
#
#  Modified by:
#  Mohab Aly
#  Ecole Polytechnique de Montreal - Montreal
#  Computer Engineering Department
#======================================================================
#
#======================================================================
# Options
#======================================================================

set opt(chan)       Channel/WirelessChannel   ;# channel type
set opt(prop)       Propagation/TwoRayGround  ;# radio-propagation
model
#set opt(prop)       Propagation/Shadowing
set opt(ant)        Antenna/OmniAntenna       ;# antenna type
set opt(ll)         LL                        ;# link layer
set opt(ifq)        Queue/DropTail/PriQueue   ;# interface queue
set opt(ifqlen)     100                        ;# max packet in ifq
set opt(netif)      Phy/WirelessPhy           ;# network interface
type
set opt(mac)        Mac/802_11                ;# mac type
set opt(rp)         DumbAgent                 ;# routing protocol
set opt(nn)         50                        ;# number of mobile
nodes
set opt(pkt_size)   128                       ;# size of UDP packet
(or broadcast message)
set opt(cp)         /home/mohab/Downloads/ns-allinone-2.35/ns-
2.35/tcl/mobility/traffic2.tcl ;# connection pattern traffic file
#set opt(cp)         /home/mohab/Downloads/ns-allinone-2.35/ns-
2.35/indep-utils/cmu-scen-gen/cbr-100-test    ;#connection pattern
traffic file
set opt(sc)         /home/mohab/Downloads/ns-allinone-2.35/ns-
2.35/tcl/mobility/topology_50_2500            ;#mobility scenario
#set opt(sc)        /home/mohab/Downloads/ns-allinone-2.35/ns-
2.35/tcl/mobility/scene/scen-3-test           ;#mobility scenario
```

```
#set opt(sc)         /home/mohab/Downloads/ns-allinone-2.35/ns-
2.35/tcl/mobility/map50.txt  ;#mobility scenario  (circular road)
set opt(tr)         trace.tr                    ;# trace file
set opt(nam_tr)     nam-trace.tr                ;# name trace file
set opt(seed)        1.5          ;# seed the random number generator
set opt(stop)       80.0                     ;# time to end simulation
set opt(x)          1000                        ;# x size for topology
set opt(y)          1000                        ;# y size for topology
set opt(lm)         ON                          ;# log movement
set opt(at)         ON                          ;# agent trace
set opt(rt)         OFF                         ;# routing trace
set opt(mact)       ON                          ;# mac trace
set opt(movt)       ON                          ;# movement trace
set opt(cw)         20
#set opt(modified)    2                     ;# use modified broadcast
set opt(sliding)     0.1                    ;# threashold to slide CW
#=====================================================================

# slot times
#Mac/802_11 set CWMax_          1023
#Mac/802_11 set SlotTime_       0.000013        ;# 20us
#Mac/802_11 set SIFS_           0.000032        ;# 10us

# physical layers headers and rates
#Mac/802_11 set PreambleLength_         32      ;# 144 bit
#Mac/802_11 set PLCPHeaderLength_       40      ;# 48 bits
#Mac/802_11 set PLCPDataRate_   6.0e6           ;# 1Mbps
#Mac/802_11 set dataRate_       6.0e6
#Mac/802_11 set basicRate_      6.0e6

# Queue/DTail set drop_front_ false
# Queue/DTail set summarystats_ false
# Queue/DTail set queue_in_bytes_ false
# Queue/DTail set mean_pktsize_ 250
# Queue/DTail/PriQ set Prefer_Routing_Protocols  1
# Queue/DTail/PriQ set Max_Levels     4
# Queue/DTail/PriQ set Levels    4

# Mac/802_11e set cfb_ 0

# Mac/802_11 set RTSThreshold_           3000      ;# bytes
# Mac/802_11 set ShortRetryLimit_        7         ;# retransmittions
# Mac/802_11 set LongRetryLimit_         4         ;# retransmissions

# added parameters to 802.11e
# Mac/802_11e set update_interval_    0.4 ;# interval to adjust the CW
# Mac/802_11e set timeout_entries_    0.8 ;# remove nodes if not heard
from in this time
# Mac/802_11e set WMA_alpha_      0.8 ;# alpha used in the weighted
average
# Mac/802_11e set scaling_factor_ 2.0 ;# how much the CW should be
increased by
```

```
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 2.5118864e-13    ;# -96 dBm
Phy/WirelessPhy set RXThresh_ 1.0e-12          ;# -90 dBm
Phy/WirelessPhy set bandwidth_ 6.0e6
Phy/WirelessPhy set Pt_ 0.0003754
Phy/WirelessPhy set freq_ 5.9e+9
Phy/WirelessPhy set L_ 1.0

#Propagation/Shadowing set pathlossExp_ 2.7
#Propagation/Shadowing set std_db_      4.0
#Propagation/Shadowing set seed_        0
#Propagation/Shadowing set dist0_       1.0

# Unity gain, omni-directional antennas
# Set up the antennas to be centered in the node and 1.5 meters above
# it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 2.0
Antenna/OmniAntenna set Gt_ 4.5
Antenna/OmniAntenna set Gr_ 4.5

#===================================================================

set BROADCAST_ADDR -1        ;# broadcast address
set MESSAGE_PORT 42          ;# periodic message port
set WARNING_PORT 33          ;# warning message
set EMER_PORT 21             ;# port to listen for emergency messages

#===================================================================
#     Functions
#===================================================================

# Display program usage
proc usage { argv0 } {
puts "Usage: $argv0"
puts "\tmandatory arguments:"
puts "\t\t\[-x MAXX\] \[-y MAXY\]"
puts "\t\t\[-cp conn pattern\] \[-sc scenario\] \[-nn nodes\]"
puts "\toptional arguments:"
     puts "\t\t\[-seed seed\] \[-stop sec\] \[-tr tracefile\]\n"
     puts "\t\t\[-pkt_size size\] \[-nam_tr nam trace\]\n"
puts "\t\t\[-modified 1 or 0\] \[-sliding threashold\]\n"
}

# set the options from the command line arguments
proc getopt {argc argv} {
global opt
lappend optlist cp nn seed sc stop tr x y pkt_size pkt_size_emer
nam_tr modified sliding
```

```
for {set i 0} {$i < $argc} {incr i} {
set arg [lindex $argv $i]

if {[string range $arg 0 0] != "-"} continue

if {[string range $arg 0 0] != "-"} continue

set name [string range $arg 1 end]
set opt($name) [lindex $argv [expr $i+1]]
}
}

Class Agent/MessagePassing/PeriodicBroadcast -superclass
Agent/MessagePassing
Agent/MessagePassing/PeriodicBroadcast instproc recv {source sport
size data} {
    # This empty function is needed so receive works.
}

Agent/MessagePassing/PeriodicBroadcast instproc send_message {} {
    $self instvar node_
    global ns_ MESSAGE_PORT BROADCAST_ADDR opt

    #puts "[$node_ node-addr] sending message"
    # send the broadcast message
    #$self sendto $opt(pkt_size) 0 $BROADCAST_ADDR $MESSAGE_PORT
    set udp [new Agent/UDP]
    $udp set class_ 1
    set sink [new Agent/LossMonitor]
    $self sendto $opt(pkt_size) 128 $udp $sink
}

# perform clean up at the end of the program
proc finish {} {
    global ns_ tracefd namtracefd opt
    $ns_ flush-trace
    close $tracefd
    if {$opt(nam_tr) != ""} {
        close $namtracefd
    }
    $ns_ halt
    exit 0
}

# set the options from the command line arguments
proc getopt {argc argv} {
    global opt
    lappend optlist cp nn seed sc stop tr x y bc_size nam_tr cw

    for {set i 0} {$i < $argc} {incr i} {
        set arg [lindex $argv $i]
        if {[string range $arg 0 0] != "-"} continue
```

```
        set name [string range $arg 1 end]
        set opt($name) [lindex $argv [expr $i+1]]
    }
}
# log the movement of a node every 0.1 seconds
proc log-movement {} {
        global logtimer ns_ ns
        set ns $ns_
        source /home/mohab/Downloads/ns-allinone-2.35/ns-
2.35/tcl/mobility/timer.tcl
        Class LogTimer -superclass Timer
        LogTimer instproc timeout {} {
                global opt node_;
                for {set i 0} {$i < $opt(nn)} {incr i} {
                        $node_($i) log-movement
                }
                $self sched 0.1
        }
        set logtimer [new LogTimer]
        $logtimer sched 0.4
}

#===================================================================
#     Main Program
#===================================================================

#get command line arguments
getopt $argc $argv

 if { $opt(nn) == 0 || $opt(sc) == "" || $opt(cp) == ""} {
 usage $argv0
 exit 1
 }

# Mac/802_11 set CWMin_ $opt(cw)
# puts "x: $opt(x), y: $opt(y)"
# puts "CW: $opt(cw)"

# Mac/802_11e set number_nodes_ $opt(nn)    ;# use modified broadcast
# algorithm
# Mac/802_11e set modified_ $opt(modified)   ;# use modifed algorithm
# Mac/802_11e set sliding_threshold_ $opt(sliding) ;# threshold to
slide CW

if {$opt(seed) > 0} {
    puts "\n Seeding Random number generator with $opt(seed)\n"
    ns-random $opt(seed)
}
# create a new simulator
set ns_ [new Simulator]
$ns_ color 1 Blue
$ns_ color 2 Red
```

```
# set up the traces
set tracefd [open $opt(tr) w]
$ns_ trace-all $tracefd


# set the topology
set topo [new Topography]
$topo load_flatgrid $opt(x) $opt(y)

# Create god, the god object must
# be created for a wireless network
set god_ [ create-god $opt(nn) ]

# set the channel object
set chan_ [new Channel/WirelessChannel]

# Set up the nam trace if desired.
if {$opt(nam_tr) != ""} {
    set namtracefd [open $opt(nam_tr) w]
    $ns_ namtrace-all-wireless  $namtracefd $opt(x) $opt(y)
}

# configure the nodes of the simulation
$ns_ node-config -adhocRouting $opt(rp) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propInstance [new $opt(prop)] \
                 -phyType $opt(netif) \
                 -topoInstance $topo \
                 -channel $chan_ \
                 -agentTrace $opt(at) \
                 -routerTrace $opt(rt) \
                 -macTrace $opt(mact) \
                 -movementTrace $opt(movt)

set y 0
set x 0

# Source the mobility scenario file.
#source $opt(sc)

# Create the mobile nodes for the simulation.
for {set i 0} {$i < $opt(nn)} {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 0  ;#disable random motion between nodes
}

# Source the mobility scenario file.
```

```
source $opt(sc)

# Source the file that contains the broadcast traffic.
#source $opt(cp)

# Define node initial position in nam
for {set i 0} {$i < $opt(nn)} {incr i} {
     $ns_ initial_node_pos $node_($i) 80 ;#80 difines the node size in
nam.
}

# Attach a new Agent/MessagePassing/PeriodicBroadcast to each node on
port $MESSAGE_PORT
for {set i 0} {$i < $opt(nn)} {incr i} {
    set udp [new Agent/UDP]
    $udp set class_ 1
    set sink [new Agent/LossMonitor]

    set bc_agent($i) [new Agent/MessagePassing/PeriodicBroadcast]
    #$node_($i) attach  $bc_agent($i) $MESSAGE_PORT
    $node_($i) attach  $bc_agent($i) $udp
    $node_($i) attach  $bc_agent($i) $sink
}

# log movement
if { $opt(lm) == "ON" } {
    puts "Logging movement..."
    log-movement
}

# Source the file that contains the UPD packets (or broadcast
traffic).
source $opt(cp)

$ns_ at $opt(stop) "finish"

# Add to the trace simulation parameters.
puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp $opt(rp)"
#puts $tracefd "M 0.0 cp $opt(cp) seed $opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"

puts "Starting Simulation..."
$ns_ run
```

APPENDIX 2 – Mobility

2.1 create-map.pl

```
#!/usr/bin/perl -w
#
#  create-map.pl: creates a circular road map for the freeway model.
#  The map is then read by the mobility generator found at
#  http://nile.usc.edu/important/software.htm, to create the mobility
#  for nodes in an ns2 simulation.
#
#  Nathan Balon
#  University of Michigan - Dearborn
#
#  Modified by Mohab Aly
#  Ecole Polytechnique de Montreal - Montreal
#

use strict;
use warnings;

use constant PI => 3.1415926535;
use constant LANE_DIST => 5;   # distance between lane

# parameter used to generate a circular road
my $nodes = 50;                # the number of points on the circle
my $inner_edge_dist = 300;     # the distance to the inner lane
my $traffic_direction = 2;     # direction of traffic (either 1 or 2)
my $lanes = 8;                 # number of lanes
my $lane_dist = 0;             # distance between the inner lane
my @center = (500,500);        # the center point of the circular road
my $min_velocity = 20.0;       # minimum velocity of the nodes
my $max_velocity = 30.0;       # maximum velocity of the nodes

my $theta_const = 2 * PI / $nodes;  # angle between nodes
my $theta = 0;                      # theta value for the current node

# Check that an even number of lanes exist if traffic
# flows in both directions on the road.
if($lanes % 2 != 0 && $traffic_direction == 2){
    die "ERROR: if traffic flows in both direction, you must " .
        "have an even number of lanes\n";
}



# display the information about the freeway
#print "FREEWAY\n";
#print "FREEWAY_NUM 1\n";
#print "LANE_NUM $lanes\n";
```

```
for(my $i = 0; $i < $lanes; $i++,$lane_dist += LANE_DIST){
    $theta = 0;
    my $direction = 1;
    # set the phase of the traffic for traffic to
    # flow in the opposite direction for half the lanes
    if($i/$lanes >= 0.5 && $traffic_direction == 2){
        $direction = -1;
    }

    # create a new lane in the map
    #print "LANE_BEGIN 0 $i $i $direction $nodes\n";

    # determine the position for two nodes connected by an edge
    # and display the results
    for(my $j = 0; $j < $nodes; $j++){
        my $next_x = 0;
        my $next_y = 0;

        my $initial_x = $center[0] + cos($theta) * ($inner_edge_dist +
$lane_dist);
        my $initial_y = $center[1] + sin($theta) * ($inner_edge_dist +
$lane_dist);
        if($direction == 1){
            $next_x = $center[0] + cos($theta + $theta_const) *
($inner_edge_dist + $lane_dist);
            $next_y = $center[1] + sin($theta + $theta_const) *
($inner_edge_dist + $lane_dist);
            $theta += $theta_const;
        }else{
            # traffic going in the opposite direction
            $next_x = $center[0] + cos($theta - $theta_const) *
($inner_edge_dist + $lane_dist);
            $next_y = $center[1] + sin($theta - $theta_const) *
($inner_edge_dist + $lane_dist);
            $theta -= $theta_const;
        }
 #       printf("PHASE %d \(%4.5f\, %4.5f\) \(%4.5f\, %4.5f\) %3.3f
%3.3f\n",
 #             $j, $initial_x, $initial_y, $next_x, $next_y,
$min_velocity, $max_velocity);

# Definition of the nodes within the circular road
# using the "printf" to state the position of the nodes and the radius
of the circle
  #$node_(1) set X_ 1331.178204310675
  #$node_(1) set Y_ 1897.0444947001733
  #$node_(1) set Z_ 0.0
  #$node_(1) radius 230.81019207872924
```

```
  printf("\$node_(%d) set X_ %4.5f\n",  $j, $next_x) ;
  printf("\$node_(%d) set Y_ %4.5f\n",  $j, $next_y);
  printf("\$node_(%d) set Z_ 0.0\n",  $j);
  printf("\$node_(%d) radius   230.81019207872924\n",  $j) ;


    }
}
```

2.1.1 Another way of creating the map (ready to be used by specifying the parameters of the map topology)

```
#@Ghada
#this file is to generate initial random location and transmission
Range for the mobile nodes

set val(nn)               [lindex $argv 0] ;#number of nodes
set val(x)                [lindex $argv 1]
set val(y)                [lindex $argv 2]
set val(outfile)          [lindex $argv 3] ;#output file name

if { $argc != 4 } {
        puts "The initial_topology.tcl script requires four parameters
to be inputed."
        puts "1.number of node >0"
      puts "2. X coordinate"
      puts "3. Y coordinate"
        puts "4. output file name"
      puts "Please try again."


} else {
    set topo [open $val(outfile) w] ;#open the file for writing
    set rng_ [new RNG]
    $rng_ seed 0
    for {set i 0} {$i<$val(nn)} {incr i} {

      puts $topo "\$node_($i) set X_ [$rng_ uniform 0.0 $val(x)]"
        puts $topo "\$node_($i) set Y_ [$rng_ uniform 0.0 $val(y)]"
        puts $topo "\$node_($i) set Z_ 0.0" ;#third dimension is not
used
        puts $topo "\$node_($i) radius [$rng_ uniform 200.0 250.0]"
        ;#transmission range default is 250m in ns2, you may use this
if only you want heterogeneous transmission ranges.

    }
   close $topo
}
```

2.2 map.txt

```
$node_(0) set X_ 797.63441
$node_(0) set Y_ 537.59997
$node_(0) set Z_ 0.0
$node_(0) radius   230.81019207872924
$node_(1) set X_ 790.57495
$node_(1) set Y_ 574.60697
$node_(1) set Z_ 0.0
$node_(1) radius   230.81019207872924
$node_(2) set X_ 778.93295
$node_(2) set Y_ 610.43737
$node_(2) set Z_ 0.0
$node_(2) radius   230.81019207872924
$node_(3) set X_ 762.89200
$node_(3) set Y_ 644.52610
$node_(3) set Z_ 0.0
$node_(3) radius   230.81019207872924
$node_(4) set X_ 742.70510
$node_(4) set Y_ 676.33558
$node_(4) set Z_ 0.0
$node_(4) radius   230.81019207872924
$node_(5) set X_ 718.69059
$node_(5) set Y_ 705.36413
$node_(5) set Z_ 0.0
$node_(5) radius   230.81019207872924
$node_(6) set X_ 691.22720
$node_(6) set Y_ 731.15397
$node_(6) set Z_ 0.0
$node_(6) radius   230.81019207872924
$node_(7) set X_ 660.74804
$node_(7) set Y_ 753.29838
$node_(7) set Z_ 0.0
$node_(7) radius   230.81019207872924
$node_(8) set X_ 627.73379
$node_(8) set Y_ 771.44812
$node_(8) set Z_ 0.0
$node_(8) radius   230.81019207872924
$node_(9) set X_ 592.70510
$node_(9) set Y_ 785.31695
$node_(9) set Z_ 0.0
$node_(9) radius   230.81019207872924
$node_(10) set X_ 556.21439
$node_(10) set Y_ 794.68618
$node_(10) set Z_ 0.0
$node_(10) radius   230.81019207872924
$node_(11) set X_ 518.83716
$node_(11) set Y_ 799.40802
$node_(11) set Z_ 0.0
$node_(11) radius   230.81019207872924
$node_(12) set X_ 481.16284
```

```
$node_(12) set Y_ 799.40802
$node_(12) set Z_ 0.0
$node_(12) radius   230.81019207872924
$node_(13) set X_ 443.78561
$node_(13) set Y_ 794.68618
$node_(13) set Z_ 0.0
$node_(13) radius   230.81019207872924
$node_(14) set X_ 407.29490
$node_(14) set Y_ 785.31695
$node_(14) set Z_ 0.0
$node_(14) radius   230.81019207872924
$node_(15) set X_ 372.26621
$node_(15) set Y_ 771.44812
$node_(15) set Z_ 0.0
$node_(15) radius   230.81019207872924
$node_(16) set X_ 339.25196
$node_(16) set Y_ 753.29838
$node_(16) set Z_ 0.0
$node_(16) radius   230.81019207872924
$node_(17) set X_ 308.77280
$node_(17) set Y_ 731.15397
$node_(17) set Z_ 0.0
$node_(17) radius   230.81019207872924
$node_(18) set X_ 281.30941
$node_(18) set Y_ 705.36413
$node_(18) set Z_ 0.0
$node_(18) radius   230.81019207872924
$node_(19) set X_ 257.29490
$node_(19) set Y_ 676.33558
$node_(19) set Z_ 0.0
$node_(19) radius   230.81019207872924
$node_(20) set X_ 237.10800
$node_(20) set Y_ 644.52610
$node_(20) set Z_ 0.0
$node_(20) radius   230.81019207872924
$node_(21) set X_ 221.06705
$node_(21) set Y_ 610.43737
$node_(21) set Z_ 0.0
$node_(21) radius   230.81019207872924
$node_(22) set X_ 209.42505
$node_(22) set Y_ 574.60697
$node_(22) set Z_ 0.0
$node_(22) radius   230.81019207872924
$node_(23) set X_ 202.36559
$node_(23) set Y_ 537.59997
$node_(23) set Z_ 0.0
$node_(23) radius   230.81019207872924
$node_(24) set X_ 200.00000
$node_(24) set Y_ 500.00000
$node_(24) set Z_ 0.0
$node_(24) radius   230.81019207872924
$node_(25) set X_ 202.36559
```

```
$node_(25) set Y_ 462.40003
$node_(25) set Z_ 0.0
$node_(25) radius   230.81019207872924
$node_(26) set X_ 209.42505
$node_(26) set Y_ 425.39303
$node_(26) set Z_ 0.0
$node_(26) radius   230.81019207872924
$node_(27) set X_ 221.06705
$node_(27) set Y_ 389.56263
$node_(27) set Z_ 0.0
$node_(27) radius   230.81019207872924
$node_(28) set X_ 237.10800
$node_(28) set Y_ 355.47390
$node_(28) set Z_ 0.0
$node_(28) radius   230.81019207872924
$node_(29) set X_ 257.29490
$node_(29) set Y_ 323.66442
$node_(29) set Z_ 0.0
$node_(29) radius   230.81019207872924
$node_(30) set X_ 281.30941
$node_(30) set Y_ 294.63587
$node_(30) set Z_ 0.0
$node_(30) radius   230.81019207872924
$node_(31) set X_ 308.77280
$node_(31) set Y_ 268.84603
$node_(31) set Z_ 0.0
$node_(31) radius   230.81019207872924
$node_(32) set X_ 339.25196
$node_(32) set Y_ 246.70162
$node_(32) set Z_ 0.0
$node_(32) radius   230.81019207872924
$node_(33) set X_ 372.26621
$node_(33) set Y_ 228.55188
$node_(33) set Z_ 0.0
$node_(33) radius   230.81019207872924
$node_(34) set X_ 407.29490
$node_(34) set Y_ 214.68305
$node_(34) set Z_ 0.0
$node_(34) radius   230.81019207872924
$node_(35) set X_ 443.78561
$node_(35) set Y_ 205.31382
$node_(35) set Z_ 0.0
$node_(35) radius   230.81019207872924
$node_(36) set X_ 481.16284
$node_(36) set Y_ 200.59198
$node_(36) set Z_ 0.0
$node_(36) radius   230.81019207872924
$node_(37) set X_ 518.83716
$node_(37) set Y_ 200.59198
$node_(37) set Z_ 0.0
$node_(37) radius   230.81019207872924
$node_(38) set X_ 556.21439
```

```
$node_(38) set Y_ 205.31382
$node_(38) set Z_ 0.0
$node_(38) radius   230.81019207872924
$node_(39) set X_ 592.70510
$node_(39) set Y_ 214.68305
$node_(39) set Z_ 0.0
$node_(39) radius   230.81019207872924
$node_(40) set X_ 627.73379
$node_(40) set Y_ 228.55188
$node_(40) set Z_ 0.0
$node_(40) radius   230.81019207872924
$node_(41) set X_ 660.74804
$node_(41) set Y_ 246.70162
$node_(41) set Z_ 0.0
$node_(41) radius   230.81019207872924
$node_(42) set X_ 691.22720
$node_(42) set Y_ 268.84603
$node_(42) set Z_ 0.0
$node_(42) radius   230.81019207872924
$node_(43) set X_ 718.69059
$node_(43) set Y_ 294.63587
$node_(43) set Z_ 0.0
$node_(43) radius   230.81019207872924
$node_(44) set X_ 742.70510
$node_(44) set Y_ 323.66442
$node_(44) set Z_ 0.0
$node_(44) radius   230.81019207872924
$node_(45) set X_ 762.89200
$node_(45) set Y_ 355.47390
$node_(45) set Z_ 0.0
$node_(45) radius   230.81019207872924
$node_(46) set X_ 778.93295
$node_(46) set Y_ 389.56263
$node_(46) set Z_ 0.0
$node_(46) radius   230.81019207872924
$node_(47) set X_ 790.57495
$node_(47) set Y_ 425.39303
$node_(47) set Z_ 0.0
$node_(47) radius   230.81019207872924
$node_(48) set X_ 797.63441
$node_(48) set Y_ 462.40003
$node_(48) set Z_ 0.0
$node_(48) radius   230.81019207872924
$node_(49) set X_ 800.00000
$node_(49) set Y_ 500.00000
$node_(49) set Z_ 0.0
$node_(49) radius   230.81019207872924
$node_(0) set X_ 802.59498
$node_(0) set Y_ 538.22664
$node_(0) set Z_ 0.0

. . . .
```

APPENDIX 3 – Network Traffic

3.1 traffic-w-jitter.pl

```
#! /usr/bin/perl
#
#  traffic-w-jitter.pl generates broadcast traffic for an
#  ns2 simulation.  The script randomly selects a broadcast
#  time for each node in the simulation.  For each time
#  interval during which a node boardcasts, the transmit
#  time is varied by +/- the amount of jitter.  The
#  variables start and end are the start time and end time
#  of  the simulation.  traffic_rate is the number of
#  times that a nodes transmits during one second.
#
#  Nathan Balon
#  University of Michigan - Dearborn
#


use strict;
use warnings;

sub setValuesFromArgs($);
sub getCommentBlock();

# the default values used to generate traffic

our $jitter = 0.1;              # the amount of jitter
our $nodes = 50;                # number of node
our $traffic_rate = 20;  # number of times a node transmits per second
our $seed = 1;                  # seed for the random number generator
our $start = 1;                 # start time of the simulation
our $end = 61;                  # end time of the simulation
our $file_name = 'traffic400.tcl'; # file to save traffic to

my $sim_length = 0;             # length of the simulation
my @trans_time = ();            # the transmit time


# read the command line arguments
for(my $argnum = 0; $argnum <= $#ARGV; $argnum++){
    # display help
    if($ARGV[$argnum] eq "-h"  || $ARGV[$argnum] eq "--help" ){
        printHelp();
        exit;

 # set the values used to generate traffic from the cammand line args
    }elsif($argnum + 1 <= $#ARGV){
```

```
        setValuesFromArgs($argnum);
        $argnum++;
    }
}

# determine the length of the simulation
if($end > $start){
    $sim_length = $end - $start;
}else{
    die "The end time of the simulation must be " .
         "greater than the start time of the simulation\n";
}

srand $seed;

#open the file and write the comment block to the file
open(FILE, ">$file_name");
print FILE getCommentBlock;

# the transmission rate to send messages
my $trans_rate  = sprintf("%.2f",  1/$traffic_rate);


# Select the initial time a node will
# transmit at based on the transmission rate.
for(my $i = 0; $i < $nodes; $i++){
     $trans_time[$i] = rand($trans_rate);
}

# the number of time each node sends a broadcast
my $num_intervals = $sim_length * $traffic_rate;

# the time interval to transmit during
my $transmit_interval = 0;

# Schedule the broadcast transmissions for the simulation.
for(my $trans_int = 0; $trans_int < $num_intervals; $trans_int++){
    # for each node in the simulation schedule a broadcast
    for(my $node = 0; $node < $nodes; $node++){
        my $trans_time = 0;  # the time the broadcast is sent
        if($trans_int == 0){
            # if first transmission use the
            # randomly selected transmit time
            $trans_time = $start + $trans_time[$node];
        }else{
            # get the random amount of jitter for the broadcast
            my $jit = rand($jitter * $trans_rate);
            # if random number is > 0.5 add the jitter or else
            # subtract the jitter
            if(rand(1) > 0.5){
                $trans_time = $start + $trans_time[$node]
                                     + $jit + $transmit_interval;
```

```
            }else{
                $trans_time = $start + $trans_time[$node]
                                - $jit + $transmit_interval;
            }
        }
        print FILE "\$ns at $trans_time \"\$bc_agent($node)
send_message\"\n";
    }
    $transmit_interval += $trans_rate;
}

close(FILE);

#####################################################################
#
#   functions
#
#####################################################################

# display help containing the command line arguments to the user
sub printHelp {
    my $space = " " x (length($0) + 6) ;
    print "usage $0 [--nodes \"number of nodes\"]\n" .
          "$space [--jitter \"amount of jitter\"]\n" .
          "$space [--traffic-rate \"rate of traffic per second\"]\n" .
          "$space [--seed \"seed for random number generator\"]\n".
          "$space [--start \"start time\"]\n" .
          "$space [--end \"end time\"]\n".
          "$space [--file-name \"name of the file to write to\"]\n";
}

# Return a comment block for the tcl traffic file
sub getCommentBlock() {
    # return the comment block to add to the tcl program.
    return
"#=====================================================================\n" .
         "#\n" .
         "#  Broadcast Traffic genrated by generate_traffic tool.\n" .
         "#  Each node within the network randomly selects the time a
time to\n" .
         "#  generate broadcast message during each broadcast
interval.\n" .
         "#\n" .
         "#  Created by Nathan Balon, University of Michigan -
Dearborn\n" .
         "#\n" .
         "#  Values modified by Mohab Aly, Ecole Polytechnique de\n" .
         "#  Montreal - Montreal\n" .
         "#\n" .
         "#  Values modified to fit 4 different simulations 50,100,200
and 400\n" .
         "#  with higher transmission rate\n" .
```

```
          "#  start time : $start, end time : $end\n" .
          "#  transmission (broadcast) rate: $traffic_rate\n" .
          "#\n" .

"#=============================================================\n\n";
}

# Set the parameters used by the program from the
# command line arguments.
sub setValuesFromArgs($) {
    my $argnum = shift;
    my $arg = $ARGV[$argnum + 1];
    if($ARGV[$argnum] eq "--jitter"){
         $jitter = $arg;
    }elsif($ARGV[$argnum] eq "--nodes"){
         $nodes = $arg;
    }elsif($ARGV[$argnum] eq "--traffic-rate"){
         $traffic_rate = $arg;
    }elsif($ARGV[$argnum] eq "--seed"){
         $seed = $arg;
    }elsif($ARGV[$argnum] eq "--start"){
         $start = $arg;
    }elsif($ARGV[$argnum] eq "--end"){
         $end = $arg;
    }elsif($ARGV[$argnum] eq "--file-name"){
         $file_name = $arg;
    }else{
        die "invalid command line argument";
    }
}
```

APPENDIX 4 – Performance Metrics

4.1 throughput.awk

```
# This script is intended to calculate the throughput of the
# proposed gateway.

BEGIN {
                node =1;
                time1 = 0.0;
                time2 = 0.0;
                num_packet=0;
                bytes_counter=0;
}
{
                time2 = $2;
                if (time2 - time1 > 0.05) {
                     thru = bytes_counter / (time2-time1);
                     thru /= 1000000;
                     printf("%f %f\n", time2, thru) > "throughput";
                     time1 = $2;
                }

                if ($1=="r") {
                                bytes_counter += $6;
                                num_packet++;
                }
}
END {
print("Done");
}
```

4.2 latency.awk

```
# This script is intended to calculate the latency of the
# proposed gateway.

BEGIN {
                time1 = 0.0;
                time2 = 0.0;
}
{
                time2 = $2;
                if ($1=="r") {

                     printf("%f %f\n", time2, time1) > "latency";
                     time1 += $2;
                }
}


END {
                print("Done");
}
```