A TWO-LEVEL EVENT BROKERING ARCHITECTURE FOR INFORMATION DISSEMINATION IN VEHICULAR NETWORKS

By

Tina Devkota

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Computer Science

May 2009

Nashville, Tennessee

Approved:

Dr. Aniruddha Gokhale

Dr. Yi Cui

DEDICATION

I dedicate this thesis to my parents, Kamala and Lok Nath Devkota.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Aniruddha Gokhale, for providing valuable guidance during my Master's research. I am grateful to my thesis committee member Dr. Yi Cui for his insightful comments. I appreciate all the help provided to me by my fellow graduate students at Vanderbilt University.

Last but not the least, I would like to acknowledge the support and encouragement provided by my family during my Master's studies.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Intelligent Transportation Systems (ITS) combine sensing, computing, network and communication technologies to provide services for better road experience. Vehicles equipped with short-range radio sense, collect and disseminate information to other interested vehicles forming an ad hoc network of Vehicles (VANETs). Vehicles can also communicate with RSU and internet nodes that form a wired backbone infrastructure. A plethora of applications have emerged in this domain: safety related applications like accident warning, cooperative collision detection, traffic management applications like congestion control, better route detection, fun driving applications like gaming, chatting or multimedia streaming, travelers information applications like pre-trip planner, finding parking spot and many more. The benefits of such applications are immense, improved efficiency, reduction in accidents, injuries and fatality rates, better response system, reduced congestion, reduced pollution, reduced costs, reduced fuel consumption and many more.

Technology-wise ITS applications are very demanding. They need real-time, scalable and reliable delivery of information. Many ITS applications have very strict time constraint, like a driver won't be interested in any traffic congestion warning received after taking an exit or any collision warning after he himself visualizes the situation. Scalability becomes critical because there can be very different technologies (wireless-wireline, GPS, TCP, UDP etc) and many vehicles involved in road. Reliable dissemination as well as priority based service is required for many ITS applications to become useful. For example, accident notification information needs to be disseminated to security people as fast as possible.

Apart from high demand of the application level parameters, ITS applications

run on VANETs which have inherent challenges due to short-range unreliable wireless communication and high mobility of vehicles. These networks show very dynamic behavior; some nodes are continuously connecting to the system while other nodes continuously get disconnected. Connectivity becomes sparse due to fluctuating channel and mobile node. Frequent connections/disconnections, scarcity of bandwidth and power, and frequent changes in location are the major challenges in mobile network [2]. Consider a VANET using short-range wireless communication technologies like DSRC that has a maximum coverage of 1km. A vehicle with speed of 65 mph can remain connected only for .573 minute. Also, [5] taking into account contention from other cars, there may not be enough bandwidth to allow each user to download emails, songs as well as other multimedia rich web-sites in the short time they are connected to the RSU. Thus, ITS applications need to ensure timeliness, scalability, reliability in an environment of mobile nodes without consistent connectivity.

We argue that distributed publish-subscribe middleware support is a must for ITS applications. The traditional client-server model that requires ubiquitous connection cannot scale well in VANETs environment. Publish/Subscribe (pub-sub) architecture is best-suited in mobile scenario because it supports multicast, anonymous and asynchronous communication paradigm decoupled in space and time. In a pub-sub environment, information providers (publishers) publish data and information receivers indicate their interest in a particular kind of data by subscribing to that data. Whenever there is match between publication and subscription, communication takes place. The communication is anonymous meaning that communication partners are not required to identify each other. The publisher does not identify who the intended receivers are. The subscribers are implicitly selected based on their subscriptions. Pub-sub is also inherently asynchronous because publisher does not have to wait for an acknowledgment from subscriber before moving on. Pub-sub resembles multicast because it allows a publisher to send the same event to many subscribers with only

one publish operation. Also publisher need not be around for the publication to be valid. Each data can be assigned a time value or a space value which denotes it validity. The system is therefore able to quickly adapt in a dynamic environment where publishers-subscribers continuously join and leave.

I.1 Our Approach

In order to deal with the chanllenges in ITS domain, our protocol uses an event brokering architecture at two levels, RSU level and Vehicles level. Whenever a vehicle requests for data with an RSU, it evaluates the speed of the vehicle and the number of vehicles it will simultaneously serve and accordingly divides the data into different chunks and distributes different chunks to various RSUs in the path of the vehicle so that when the vehicle gets to the next RSU, automatically the missed part will be delivered to the vehicle. Data requested by vehicles moving together in a particular direction will typically be structured and highly correlated, such as traffic situation on a sequence of roads from a starting point to a destination [1]. So instead of giving each vehicle an individual copy of the information, dissemination can be brokered in such a way that each car in a cluster of closely spaced cars gets different pieces of the data and they can communicate among themselves to find the missing pieces. Thus after some time all the cars in the cluster will have the same data. Such peer-to-peer network of vehicles can be reliable as cars moving with a constant speed towards a particular direction tend to form a more stable network. Peer-to-peer networks [16] offer promising paradigms for developing efficient distributed systems and applications.

I.2 Thesis Organization

The thesis is organized in the following manner. In Chapter II we describe the related work in the field of information dissemination in VANeTs. Chapter III we

introduce the Two-Level Event Brokering Architecture. Chapter IV we give a brief overview of the simulation tool used. Chapter V and VI we discuss the experiment performed and results observed. Chapter VII describes the future enhancement and concludes the thesis.

CHAPTER II

RELATED WORK

A lot of researches have been focused in information dissemination in VANETs. Some protocols describe dissemination without infrastructure support. Rover forms on-demand tree of vehicles to route the information. GVGrid

However, they don't consider the connectivity time of a vehicle with road-side unit or size of the data that can be transmitted.

CHAPTER III

TWO LEVEL EVENT BROKERING ARCHITECTURE FOR INFORMATION DISSEMINATION IN VANETS

III.1 Background

Vehicular communications can be broadly categorized into 1) Infrastructure based communication where RSUs are set up at regular intervals along the road which are responsible for information dissemination to vehicles, 2) pure vehicle-to-vehicle communication where vehicles "gossip" among themselves to disseminate information, and 3) hybrid approach which combines the infrastructure with "gossiping" vehicles to disseminate information. The cost of deployment can be a major issue in the first category because larger number of RSUs will be required to cover every possible area on road. In the second category, if the vehicle density is too sparse, information dissemination may not be possible. The hybrid approach, however, can provide real-time global information and the deployment cost can be reduced as the physical movement of vehicles can be exploited in areas not covered by the infrastructure.

We leverage the vehicle-to-vehicle, vehicle-to-road-side-unit and road-side-unit-to- road-side-unit communication to design a two-level event-brokering architecture to disseminate information reliably, scalably and at real-time.

III.2 Publish Subscribe

Publish-subscribe paradigm is one in which the endpoint nodes communicate with each other by sending (publishing) data and receiving (subscribing) data anonymously. Usually the only property a publisher needs in order to communicate with a subscriber is the name and definition of the data. The publisher does not need any information about the subscribers, and vice versa. The subscription is check against

all publication and dispatched to all the interested users by the pub-sub infrastructure. The subscription language may be group-based, topic-based or content-based. Content based subscription is very flexible than the other two but implementation becomes a challenge in itself. In this design we talk about topic based subscription. Each event is tagged with a short "subject" (or "topic") that describes its content. The subject is either an arbitrary string or a string taken from an agreed-upon domain. The subscriber defines its subscription in terms of this subject line. Event Brokering System

An Event Brokering System consists of Event sources, Event sinks and Event Broker. Event sources are those that generate events, like a publisher publishing events. Event Sinks are those entities that subscribe to certain events. Event broker keeps track of the active subscription and publication, and whenever there is a match, it forwards the data to the corresponding subscriber or Event Sink. In case of mobile subscribers where subscribers disconnect after subscribing at one location, Event brokers keep data in catch and forward it them to the place where the subscribes show up the next time they connect.

III.3 Overview of the System

The system consists of RSU placed at certain intervals in Road. Each RSU is connected to the internet and through internet to an event broker. The RSU acts as both publisher as well as a subscriber. All RSUs are uniquely named and they listen for events published for them with their RSUname as topic name. For example, RSU1 subscribes to all the events with topic_name RSU1. Each RSU calculates the amount of data that it can send and if it cannot transmit all the data requested by the vehicles, it publishes events about the vehicles, data requested by those vehicles and the amount of data yet to transmit. Supposing RSU1 has 2 vehicles moving towards RSU2 asking for multimedia_data, and it can only transmit 50% of the requested

data, RSU1 transmits only 50% data and sends a message to RSU2 indicating the requested data and its amount that needs to be transmitted, so that when the vehicles reaches RSU2 it automatically get the required portion.

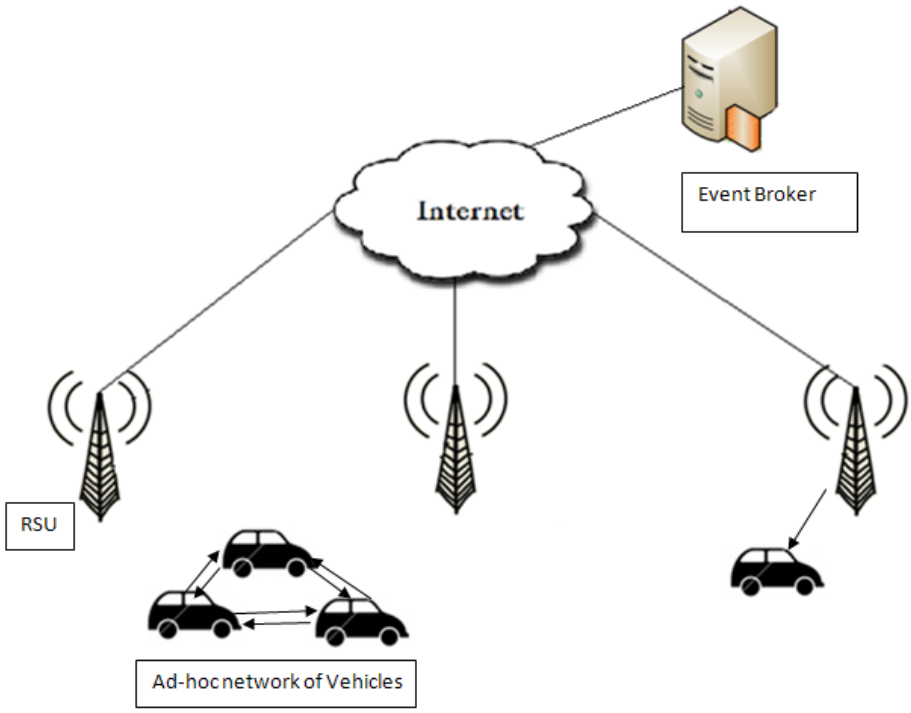

Figure III.1: System Overview

III.4 Description of the Protocol

Vehicles are assumed to be equipped with GPS device and a digital map of the city it is travelling. This digital map will contain the location information of all RSUs. Each RSU is uniquely numbered. Vehicles act as mobile subscribers. Whenever it wants some data it requests it through subscription. Each DataRequest_Message contains the interested_data, speed, next_RSU along the destination travelling, current position, VehicleId which is a unique identifier for each vehicle (VID) and, optionally, QoS parameters of the subscription (time_to_live, etc). If a vehicle doesn't need any

data, it sends a Ping_Message to the RSU containing all the above fields but data requested. Ping_Message is a way of letting an RSU know of a vehicle's presence even when it doesn't need data.

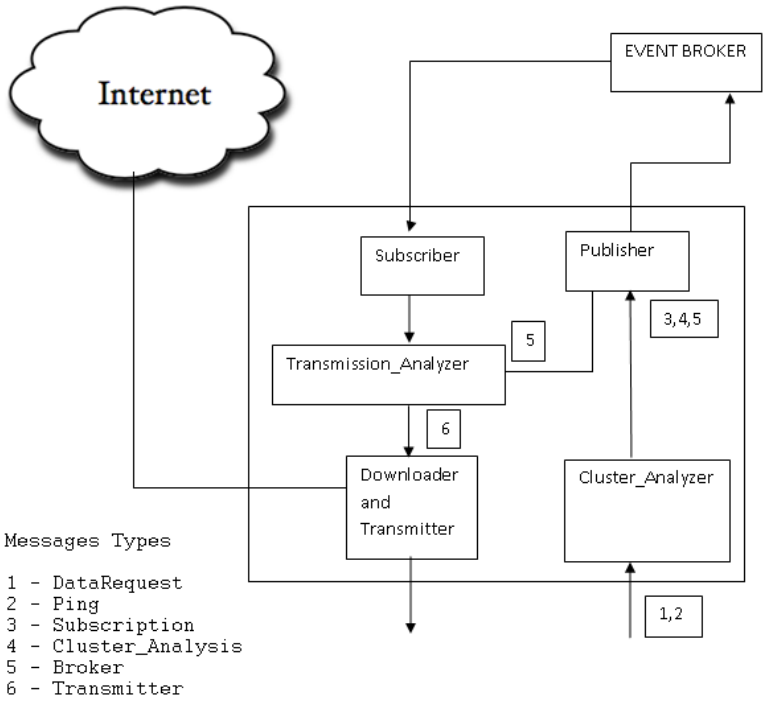

Figure III.2: Road Side Unit

RSU contains the modules described in fig. Let us describe each module in detail.

1) Cluster_Analyzer

Cluster_Analyzer performs analysis on the data received and passes in appropriate messages to Publisher. It gets the DataRequest_Message and Ping_Message from vehicles. It takes into account three things: speed, current_position and next_RSU to predict the cluster_size at the next RSU. For example, if 3 vehicles are moving with 40mph towards RSU2 and 2 vehicles are moving towards RSU3 with speed 50mph, this module will predict 2 clusters towards different RSUs. Similarly if 3 vehicles are moving with 40 mph and 2 with 60mph towards RSU3, then (depending upon the

distance between RSU1 and RSU3) this module will predict two different clusters at RSU3. It forwards its prediction towards publisher as Cluster_Analysis_Message. Also the DataRequest_Message is changed as Subscription_Message which contains only data_requested and VID and is passed towards the Publisher. Cluster_Analyzer checks to see if there are multiple requests for the same data from the vehicles in the same cluster. i.e. If vehicle1 and vehicle2, both have the same next_RSU field and requests for Traffic_congestion data, then only one Subscription_Message is generated.

2) Publisher

The task of publisher is to publish different messages passed to it. It publishes Cluster_Analysis_Message with topic name as next_RSU. Subscription_Message or Broker_Message are published with VID as topic name. We will talk about Broker_Message later. All the publications are pushed towards the Event Broker and it checks its subscription list to see if there is a match. If there is a match have subscribed, it pushes the respective messages towards them. For example, if RSU1 predicts one cluster moving towards RSU2 and another one towards RSU3, it publishes two Cluster_Analysis_Messages, one with topic name RSU2 and the other with RSU3. If vehicle 1,2, 3 have subscribed to some data, that is published as topic 1, 2 and 3.

3) Subscriber

The Subscriber module in RSU subscribes to all the events published with its name as topic. As soon as it gets Cluster_Analysis_Message, it subscribes to all the VID of the incoming vehicles, so that their Subscription_Message can be forwarded to this RSU. Whenever there is a match in publicationsubscription, the respective data gets pushed towards the subscriber who, in turn, forwards it to the Transmission_Analyzer module. 4) Transmission_Analyzer The work of this module is to analyze whether the requested data can be transmitted by the RSU to the respective vehicles or not. From the Cluster_Analysis_Messages it extracts the Average_Speed of all approaching

vehicles. It also predicts the Estimated_Number_of_Clusters that the RSU need to be serving simultaneously. This can be calculated by using the speed of the approaching clusters. Then, it makes an estimate as to how much duration will each cluster be connected to the RSU.

Connection_Time, t = (Average_Speed / Coverage_Distance) + (Average_Speed / (Cluster_size/2))

Coverage_Distance depends on the wireless technology used. Then, bandwidth that can be separated for each cluster is evaluated.

Bandwidth, b = Achievable B/w / Estimated_Number_of_Clusters

The Total_Data that can be transferred in any time window can be calculated using Connection_Time and Bandwidth,

Total_Data = Connection_Time * Bandwidth = t * b = X bytes

After deriving the number of bytes that can be at most transmitted to the whole cluster, data will be brokered among various RSUs along the path the cluster is travelling. Also, our protocol differentiates between low and high priority data. High priority data or First Priority data, P1 is that data which the vehicle cannot wait for the next RSU. This category of data is urgent, like some accident notification or traffic congestion along the road segment. Next is lower priority data or Second Priority data, P2, which doesn't have critical deadline like infotainment or multimedia streaming data. So for all the requested data, RSU assigns priority. For simplicity we assume only two priorities, however it can be extended to any number of priorities.

i.e. The data requested is either Priority 1 or Priority 2 or Dreq = P1 + P2. Depending upon the size of data requested the following cases may occur.

Case I: When the amount of data that can be transferred is greater than requested data;

$X \geq Dreq$

This is not a problem at all. The whole data can be transferred to the cluster easily.

Case II:

When the amount of data that can be transferred is greater than priority 1 data requested but less than the total data requested, i.e.

$P1 < X < Dreq$ or, $P1 < X < P1 + P2$

In this case, divide the second priority data into two parts,

$P2 = P2' + P2$"

Then P1+ P2' (=X) is transmitted to the cluster and P2" is forwarded to next RSU in the direction of travel of the vehicle.

Case III:

When the amount of data that can be transferred is less than the requested priority 1 data. We generally don't expect this case to arise. However in such a case, P2 is forwarded to the next RSU in the direction of travel. Priority 1 data is divided into two chunks,

$P1 = P1' + P1$"

One chunk P1' is transmitted to the cluster and the next chunk P1" is send along with the next cluster, such that when some members of the latter cluster meet some members of the former, they can forward the packets. We do not elaborate on this since it is outside the scope of the thesis.

Now after deciding what to transmit (Transmitter_Message) and what not to (Broker_Message), it will pass this information to other modules. Broker_Messages are generated for the data that cannot be transmitted. This message is similar to the Subscription_Message but has two additional fields, portion_indicator which indicates the portion that is already transmitted and next_RSU field which is left empty. Supposing only 50% of the data requested by vehicle1 can be transmitted, the portion_indicator field will be .5. This message is passed to the Publisher which publishes it under the

topic VID. The Transmitter_Message contains information about data that is to be transmitted to vehicles. There could be cases where many vehicles in that cluster subscribed to the same data. In such case, only one Transmitter_Message is generated. Next, Transmitter_Message is forwarded to the Downloader_and_Transmitter module.

5) Downloader_and_Transmitter

This module is connected to the internet so it can download virtually any data in real-time. The Transmitter_Message contains information about what and where to download and transmit to which vehicle and when. Accordingly, this module downloads the data and transmits it to the incoming vehicular cluster. While disseminating to a cluster, the data is divided into many chunks and each chunk is numbered. Our protocol ensures that same copy of data is never transmitted. This is because vehicles can communicate among themselves and download the pieces they don't have. This is done by forming a ad-hoc network of nearby vehicles. A mobile peer-to-peer protocol like car-torrent [13] can be used to disseminate information in this ad-hoc network of vehicles. The benefit of this network is that if any vehicle didn't get what it subscribed for, it can check to see if anyone else in the cluster has the same piece. Whenever vehicles come in communication range with one another, they send-in peer messages subscribing the data they need and publishing the data they have. Thus after finite time, all the vehicles will have all the data they want without the need for the RSU to transmit the data requested by many vehicles more than once.

III.5 Distribution

The idea of having a centralized event broker doesn't scale well when the number of vehicles and RSU increases. Also centralized event broker means central point of failure and also the server can get too at loaded at time. This problem is solved by having an Event broker distributed on the network. Each EB can serve only few RSUs

and then be connected to other EBs. The connection between EBs is determined by the layout of the road. So whenever an EB cannot find a matching subscription from its list, it broadcasts or multicasts this message to other EBs, which can in-turn check to see if match can be obtained. Instead of having a flat relation between the EBs, hierarchical relation can also be used so as to divide the responsibility and take the load off from a single EB.

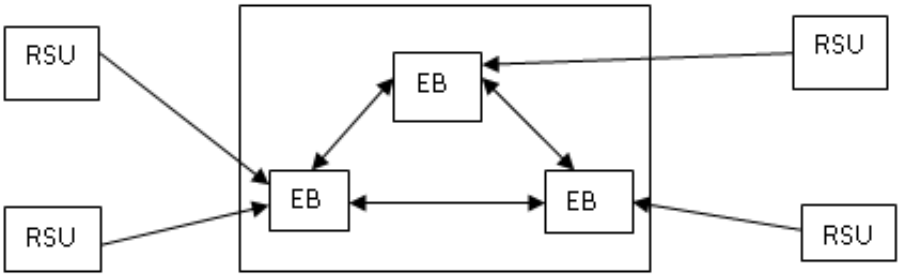

Figure III.3: Distribution of Event Broker

III.6 Replication

Replication can be used in a publish/subscribe system to increase its availability and reliability when faced with server failures or network partitions. In a replicated publish/subscribe system, a userś subscription is monitored by multiple Event Brokers independently. In particular, in figure we assume that two EBs, EB1 and EB2, simultaneously monitor the subscriptions for each user. This results in increased responsibility for each EB and also requires some level of understanding among the EB to make sure no duplicate copies are sent.

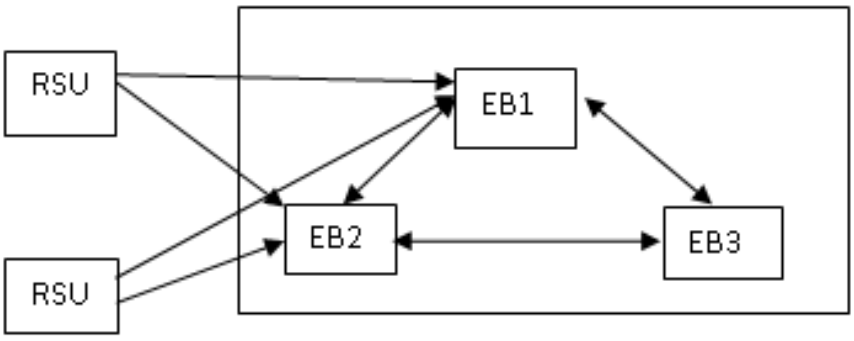

Figure III.4: Replication of Event Broker

CHAPTER IV

THE DISCRETE EVENT SIMULATION SYSTEM OMNET++

Simulation was done using OMNET++ and INET Framework. This chapter presents the features of the OMNET++ simulator, INET framework and applications used for simulation.

IV.1 Introduction

Objective Modular Network Testbed in C++ (OMNeT++) is an object-oriented modular discrete event network simulator which can be used for traffic modeling of telecommunication networks, protocols, queuing networks, multi-processors and other distributed hardware systems as well as validating hardware architectures or evaluating performance aspects of complex software systems.

An OMNeT++ module consists of hierarchically nested modules which communicate by passing messages to one another. The nesting of submodules is not limited, and thus model structure can reflect the logical structure of the actual system. The lowest level of the module hierarchy contains the algorithms implemented in C++ in an event-driven or process-style way. Model structure is described in OMNeT++'s NED language. Gates are the input and output interfaces of modules through which messages are sent in or out. Each connection (also called link) is created within a single level of the module hierarchy: within a compound module, one can connect the corresponding gates of two submodules, or a gate of one submodule and a gate of the compound module. Connections can be assigned three parameters propagation delay, bit error rate and data rate which facilitate the modeling of communication networks, but can be useful in other models too.

IV.2 INET Framework

The INET Framework builds upon OMNeT++, and uses the same concept: modules communicating by message passing. It contains IPv4, IPv6, TCP, UDP protocol implementations, several application models and supports wireless and mobile simulations as well. Support for mobility and wireless communication has been derived from the Mobility Framework [8].

Some common INET modules are as follows:

1)Channel Control

This module takes care of establishing communication channels between nodes that are within communication distance and tearing down the connections when they move out of range [8]. This information is used by the radio interfaces of nodes at transmissions. For mobile hosts, the Mobility Controller module recalculates its position regularly and updates the graphical representation of its host and communicates the current location information to the Channel Control.

2)FlatNetworkConfigurator

This module configures IP addresses and routing tables for a "flat" network, "flat" meaning that all hosts and routers will have the same network address and will only differ in the host part. The module runs at the beginning of the simulation and it assigns IP addresses to cars and rsus. It then discovers the topology of the network (using OMNeT++'s cTopology class), and calculate shortest paths, and finally, adds routes which correspond to the shortest paths to the routing tables. The configurator picks all modules of types listed in the moduleTypes parameter and their connections, and builds a graph from it. Then it runs Dijstra's shortest path algorithm on it, and configures all modules which are IP nodes, that is, not listed in the nonIPModuleTypes parameter.

3)NotificationBoard

This module is used by other modules to notify each other about "events" such

as routing table changes, interface status changes (up/down), interface configuration changes, wireless handovers, changes in the state of the wireless channel, mobile node position changes, etc. Modules can "subscribe" to categories of changes (e.g. "routing table changed" or "radio channel became empty"). When such a change occurs, the corresponding module will let NotificationBoard know, and it will disseminate this information to all interested modules.

4)InterfaceTable

This module contains the table of network interfaces (eth0, wlan0, etc) in the host. The interfaces are registered dynamically in the table. This module does not send or receive messages but implements methods to simply read and update the interface table.

5)RoutingTable

This module contains the IP (v4) routing table, and heavily relies on InterfaceTable for its operation. It implements functions for querying, adding, deleting routes, and finding the best matching route for a given destination IP address.

6)Mobile Hosts

The mobile has follows ISO/OSI architecture. It is composed of Network Layer, Transport Layer and Application Layer sub-modules.

In the INET Framework, when an upper-layer protocol wants to send a data packet over a lower-layer protocol, the upper-layer module just sends the message object representing the packet to the lower-layer module, which will in turn encapsulate it and send it. The reverse process takes place when a lower layer protocol receives a packet and sends it up after decapsulation.

7) TCP

It is the TCP protocol implementation. A TCPSegment is represented by the class TCPSegment. For communication between client applications and TCP, the

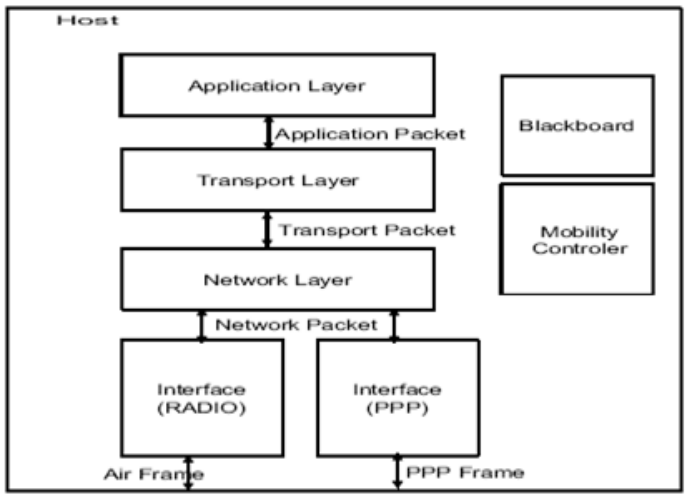

Figure IV.1: Mobile Host

TcpCommandCode and TcpStatusInd enums are used as message kinds, and TCP-Command and its subclasses are used as control info. TCP sends notifications to the application whenever there's a significant change in the state of the connection: established, remote TCP closed, closed, timed out, connection refused, connection reset, etc. These notifications are also cMessages with message kind TCP_I_xxx (TCP_I_ESTABLISHED, etc.) and TCPCommand as control info. One TCP module can serve several application modules, and several connections per application. The kth application connects to TCP's from_appl[k] and to_appl[k] ports. When talking to applications, a connection is identified by the (application port index, connId) pair, where connId is assigned by the application in the OPEN call. The TCPSocket C++ class is provided to simplify managing TCP connections from applications. TCPSocket handles the job of assembling and sending command messages (OPEN, CLOSE, etc) to TCP, and it also simplifies the task of dealing with packets and notification messages coming from TCP. The TCP model relies on sending and receiving IPControlInfo objects attached to TCP segment objects as control info.

8) UDP

This module is UDP protocol implementation. The UDP protocol header is represented by the class UDPPacket.The module can be connected to several applications. For sending an UDP packet, the application should attach an UDPControlInfo object to the payload, and send it to UDP. UDP will also attach an UDPControlInfo object to any payload message in sends up to the application. For receiving UDP packets, the connected applications should first "bind" to the given UDP port. This can be done by sending an arbitrary message with message kind UDP_C_BIND and an UDPControlInfo attached with srcPort filled in. If there is only one app which doesn't bind to any port, it will receive all packets. The UDP model relies on sending and receiving IPControlInfo/IPv6ControlInfo objects attached to UDPPacket objects as control info .

9) Network This module is a compound and represents the Network layer of an IP node. It has Interfaces to transport layer: TCP, UDP, echo/ping, RSVP modules.

10)Radio

It implements the physical layer for 802.11 models. It consists of several submodules oriented on the ISO/OSI stack like the MAC layer, Decider layer and SNREval. While SNREval module is responsible for recording the signal and noise power at different points in time (at each transmitted signal), the decider module uses that information along with the applied modulation scheme to calculate the bit error rate or to mark erroneus bits if needed.

These modules provided by INET Framework can be freely combined to form hosts and other network devices with the NED language (no C++ code and no recompilation required).

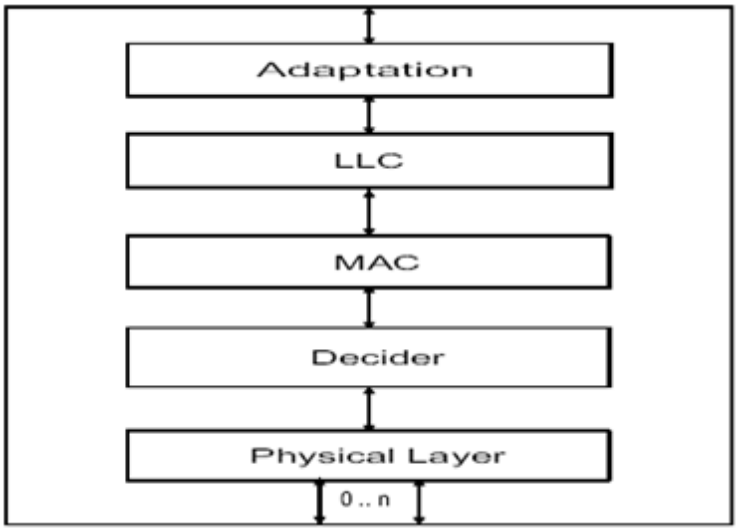

Figure IV.2: Radio

CHAPTER V

SIMULATION USING OMNET++

This chapter describes the simulation setting used. It lists all the modules used in the thesis.

V.1 Simulation Settings

The wireless radio interface used was 802.11b with a maximum range of 250m. All the broadcasts occur at the same channel frequency. 802.11b was selected because it is similar to the new 802.11p standard (Wireless Access for the Vehicular Environment or WAVE) and because it is already widely used in simulation and in real life. The datarate used is 2Mbps. The following modules are used.

1) RSU (Road-Side-Unit)

RSU is a compound module consisting of 802.11 radio interface in adhoc mode for wireless connection and any number of PPP interfaces and Ethernet interfaces for wire-line connection. It also contains NotificationBoard, Mobility and InterfaceTable and RoutingTable modules. A TCP application runs on it that establishes one connection with a Car and transfers specified number of data in 1KB chunks. Also, it has a PPP connection with another RSU. A UDP application is responsible for transferring data from the RSU to another one through the PPP connection.

2) Car It is a compound module consisting of 802.11 radio interface in adhoc mode. It also contains NotificationBoard, Mobility and InterfaceTable and RoutingTable modules. It has a TCP application that accepts data from RSU and a UDP application which can transfer data to other cars around it.

3) BrokerTCPSessionApp This is a single-connection TCP application. It opens a connection, sends the given number of bytes, and closes. The total number of bytes

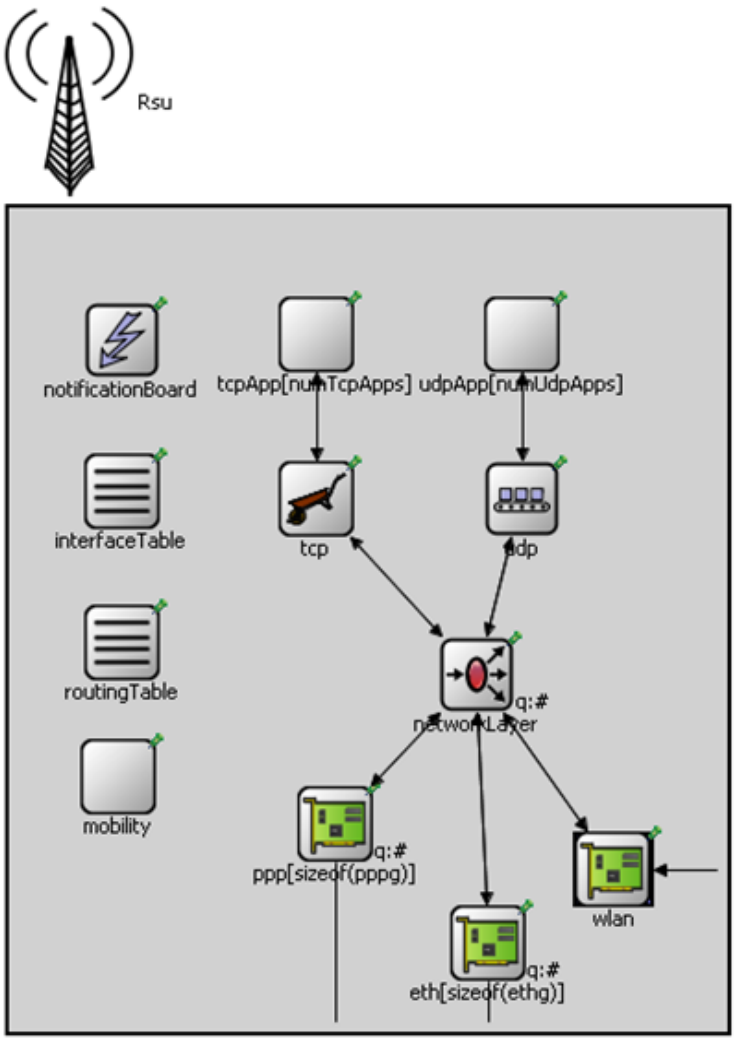

Figure V.1: Road Side Unit

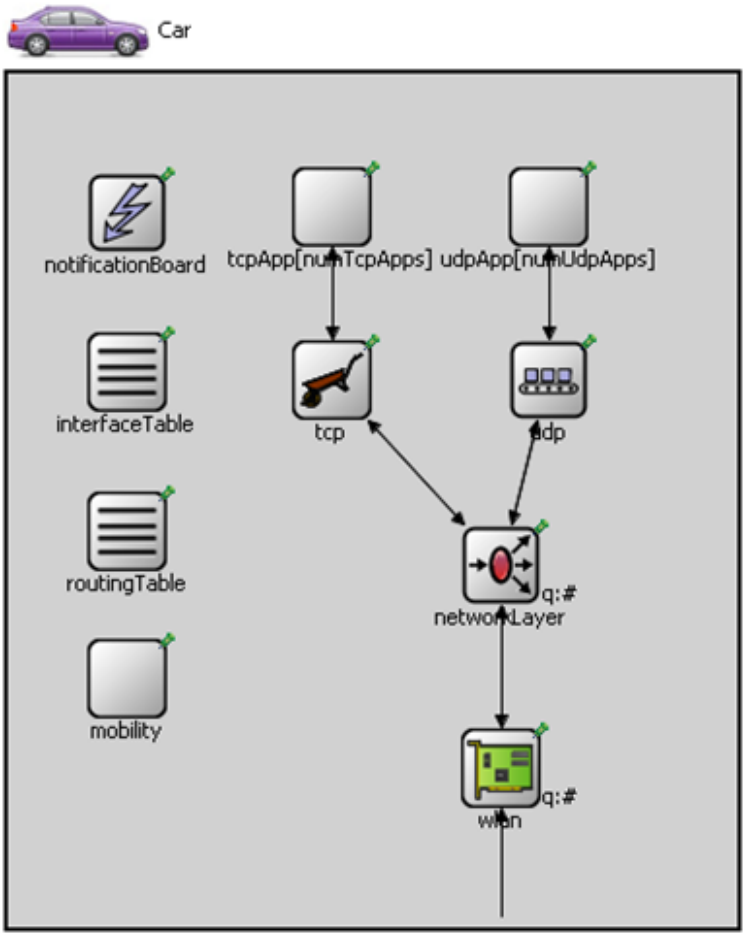

Figure V.2: Car

to send is determined by the cluster_size and cluster_speed. Each time it sends a packet of 1024 bytes.This is employed in RSU for sending data to Cars.

4) UDPMyBrokerApp This UDP application sends UDP packets to the given IP address at the given interval. It will wait for tOpen time before starting to send and closes connection at tClose time. This is employed in RSU for sending data to another RSU.

5) TCPSink This application accepts any number of incoming TCP connections and discards whatever arrives on them but keeps track of total data received. It is employed in cars and it receives data from any RSU.

CHAPTER VI

EXPERIMENTAL EVALUATION

VI.1 Experiment I

The objective of this experiment is to establish a relation between number of cars passing by an RSU and the amount of data a car can potentially receive at different speed. A car with linear mobility passes by a stationary RSU with constant speed. As soon as the RSU senses car in communication range, it establishes a TCP connection and passes data to the car in 1KB chunks. The number of bytes received is noted. Next, speed of the car is varied from 15mps to 36 mps and amount of data received by a car in average is noted. In each case of multiple cars all cars run at the same speed close to one another as if they had been moving in a cluster. The experimental data is averaged between 10 runs.

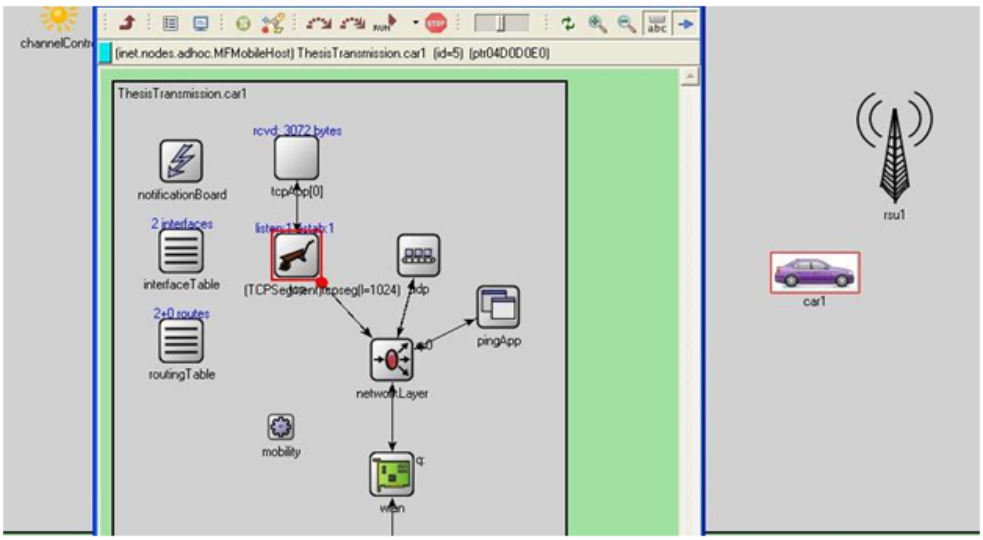

Figure VI.1: Experiment 1

VI.2 Results

The data received by a single car with a speed of 33.5 miles per hour is the maximum. However, it can be seen from the graph that as the number of cars increases the average data received by a car decreases drastically. Similarly drastic reduction in data received by a car can be seen when the speed of the car is increased. The results follow intuition. As the speed of car increases, the time it remains connected to the RSU decreases and data that can be transferred is seriously lessened. Also as the number of cars increases, they contend for the bandwidth of the RSU and that affects the amount of data the RSU can transfer to each car.

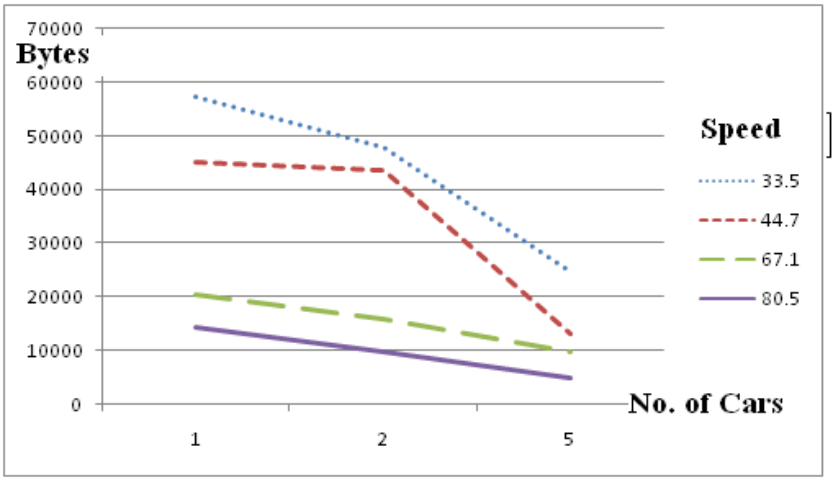

Figure VI.2: Data at different cluster density and speed

There are various things that can be observed from the above graph.

It is intuitive that when there are many cars, the amount of data received by each car is less. However, it is interesting to notice that there isn't much gain in performance even when the speed is reduced. 5 cars running at a speed of 44.7 or 67.1 doesn't have a lot of difference in terms of bytes received. This implies that reducing speed is not a solution to transfer more data for larger cluster size.

The amount of data received by a single car is never 5 times as much as the

average amount of data received by a cluster of 5 cars. That means, the total data received by a cluster of cars is much more than that a single car receives. Let us assume all the cars had been interested in the same information. This is a genuine assumption as vehicle moving together have similar (or may be same) short-term (or long term) interests or destination. The graph above clearly depicts that when each car is passed an individual copy of the data, very less data can be received by each. Instead, if the whole information is divided into optimal chunks and different chunks are transmitted to the cluster of cars, greater amount of data can be transmitted. In other words, 5 cars running at a speed of 67.1 can get as much information as a single car running at speed of say 39 (somewhere between 33.5 and 44.7). This greatly motivates the brokering of information in peer-to-peer network of cars.

Let us analyze "best performance" points in the graph. By best performance we mean highest data transfer by keeping speed as high as possible. For the case of 5 cars, the speed of 67.1 gives best performance since decreasing the speed doesn't give much benefit. This speed is normal speed limit at highway scenarios. Next best point for the case of 2 cars is towards 44.7 mph which is again a typical lower level speed limit at highway scenario. Lastly, it doesn't look inappropriate to conclude that the best case for a single car is to drive past the RSU as slow as possible. Let us now analyze the "worst performance" points in the graph. For 1 car case, high speed is bad. For 2 cars case, there exists a cut-off speed between 44.7 to 67.1 mph beyond which the speed affects the performance drastically. For 5 cars case, the lower speed is not much of performance gain. This motivates the design of RSU-less or lesser number of RSU based design when there is a dense slowly moving traffic, typical scenario of a city.

VI.3 Experiment II

This experiment is performed to study data dissemination amount from many RSUs for varying speed. To keep the study simple we show the case of one car and two cars using 1, 2 and 3 RSU placed at a distance of 1000m apart.

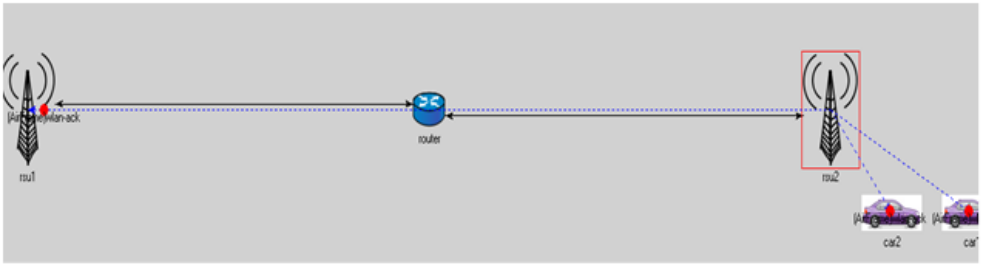

Figure VI.3: Experiment 2

VI.4 Results

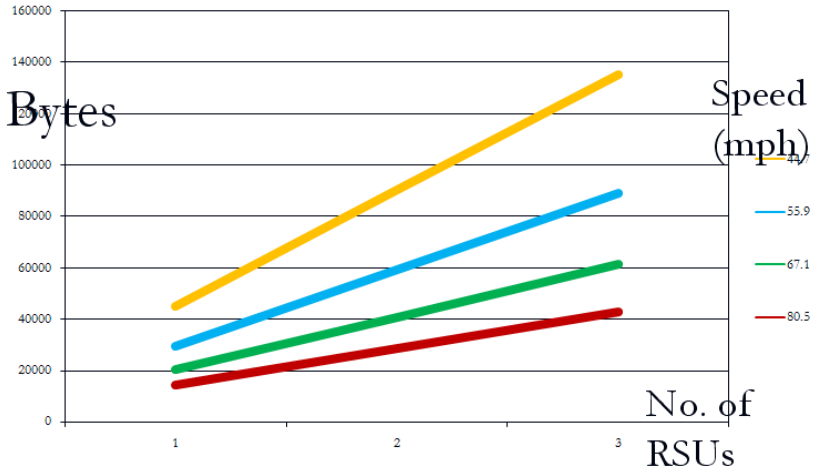

Figure VI.4: Results with cluster density 1

The positive slopes in all the graphs show that data received by each car increases when we use many of RSU along its way. The graph has positive slope at all the

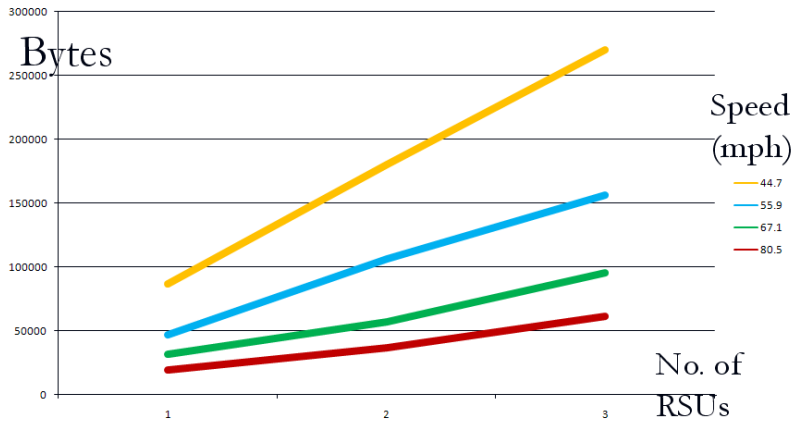

Figure VI.5: Results with cluster density 2

speed. This also shows that employing many RSU on the way has the benefit of delivering more information to a cluster but deployment cost can be very high.

We plan to draw insights from this experiment and estimate largest distance between two RSUs so that it is be reliable enough to be used for deliverying real-time data. This experiment clearly shows that the amount of data disseminated is directly propotional to the cluster density. So utilizing the average cluster density on a road, and the time a protocol like car-torrent take to disseminate information in that cluster_density, we can derive largest distance that best satifies the time requirement of ITS application.

CHAPTER VII

FUTURE ENHANCEMENT AND CONCLUSION

Our event-brokering architecture can be used to provide different context-aware services to vehicles on road. Context-aware software systems specifically have the ability to seamlessly adapt their behavior according to the context on which the system executes. Context Aware middleware allows applications to easily acquire context info, reason new contexts using different logics and adapt themselves to changing context. These kinds of services can be built using the event brokering architecture. A driver doesn't have to re-subscribe to any event at different locations. According to the route he will be taking, his subscription can be passed to all the RSUs on his way and he can get updated at every corner of the road.

WiMAX is a recent wireless broadband standard that has promised high bandwidth over long-range transmission. Mobile WiMaX has a target service range upto 50km (31 miles) with shared data rates of upto 70Mbps and a peak of up to 268 Mbps. It is optimized for dynamic mobile radio channels and supports handoffs and roaming at vehicular speeds of upto 75mph. We plan to test our protocol using WiMAX.

A deployment cost analysis of the system proposed above results in more expensive hardware for the RSU. This is because of the complexity and analysis capability added up in the RSU. We plan to evaluate an alternative solution where the expensive computation is pushed towards the event broker. The cost benefit of such system is an interesting matter of study for us. Content Based Subscription is more flexible than the topic based subscription that we have implemented in the thesis. With content based subscription, filtering will be done not only at the subject level but also at the content of the message. We plan to include content based subscription in our system.

REFERENCES

[1] J. Rybicki, B. Scheuermann, W. Kiess, C. Lochert, P. Fallahi, M. Mauve. Challenge: Peers on Wheels - A Road to New Traffic Information Systems. *MobiCom 2007: Proceedings of the 13th Annual International Conference on Mobile Computing and Networking*, pp. 215–221, Montreal, Quebec, Canada, September 2007.

[2] Yongqiang Hange and Hector Garcia-Molina. Publish/Subscribe in a Mobile Environment. *2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE 2001)* , May 20, 2001, Santa Barbara, California.

[3] A. Nandan, S. Das, G. Pau, M. Gerla, M. Y.Sanadidi. Co-operative downloading in vehicular ad-hoc wireless networks. *Wireless On-demand Network Systems and Services, 2005. WONS 2005. Second Annual Conference on* , vol., no., pp. 32-41, 19-21 Jan. 2005.

[4] M. Kihl, M. Sichitiu, T. Ekeroth, M. Rozenberg. Reliable Geographical Multicast Routing in Vehicular Ad-hoc Networks. *Proc. of the International Conference on Wireless/Wired Internet Communications (WWIC 2007)*, (Coimbra, Portugal), May 2007.

[5] W. Sun, H. Yamaguchi, K. Yukimasa, S. Kusumoto. GVGrid: A QoS Routing Protocol for Vehicular Ad Hoc Networks. *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, vol., no., pp.130-139, 19-21 June 23006.

[6] B. Petit, M. Ammar, R. Fujimoto. Protocols for roadside-to-roadside data relaying over vehicular networks. *Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE* , vol.1, no., pp.294-299, 3-6 April 2006.

[7] I. Leontiadis, C. Mascolo. GeOpps: Opportunistic Geographical Routing for Vehicular Networks *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a* , vol., no., pp.1-6, 18-21 June 2007.

[8] I. Leontiadis. Publish/subscribe notification middleware for vehicular networks. *MDS '07: Proceedings of the 4th on Middleware doctoral symposium*, pages 1-6, New York, NY, USA, 2007. ACM

[9] H. Wu, R. Fujimoto, R. Guensler, M. Hunter. MDDV: a mobility-centric data dissemination algorithm for vehicular networks. *In VANET '04: Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks,* pages 47-56, New York, NY, USA, 2004. ACM Press.

[10] A. Skordylis, N. Trigoni. Delay-bounded routing in vehicular ad-hoc networks. *MobiHoc '08: Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, pages 341-350, Hong Kong, Hong Kong, China, 2008. ACM Press.

[11] www.omnet.org

[12] W. Drytkiewicz, S. Sroka, V. Handziski, A. Kopke, H. Karl. A Mobility Framework for OMNeT++.