2011

study on vehicular network application and simulation

Xuejia Lu
Iowa State University

Study on vehicular network application and simulation

by

Xuejia Lu

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Wensheng Zhang, Major Professor
Daji Qiao
Ying Cai

Iowa State University

Ames, Iowa

2011

DEDICATION

I would like to dedicate this thesis to my parents, my wife and my daughter without whose encouragement, love, understanding and support I would not have been able to keep working on this research and finish it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

VANET is an emerging mobile ad hoc network paradigm that facilitates vehicle-to-vehicle and vehicle-to-infrastructure communication. The most important application of the VANET is for driving safety. Road condition-awareness is critical for driving safety. Existing VANET-based systems usually assume drivers detect and report safety related road conditions, which however may be untrue because, drivers may not be willing to perform these duties, or such duties may distract drivers and thus make driving even unsafe. Therefore, automatic detection without human intervention is desired. As the first contribution of this thesis work, an automatic road condition detection system has been designed based on the idea of collecting and analysing the footprints of vehicles to infer anomaly. It has also been studied how to utilize inexpensive roadside devices, such as sensors, to facilitate the information collection and analysis, especially in the absence of connectivity between vehicles.

Due to the difficulty of conducting large-scale experiments on real roads, simulation plays an important role in VANET research. To make simulation close to the reality, it is desired to include detailed and realistic simulation of vehicle behaviour under various road conditions, and this is especially needed for studies targeted at driving safety. In the past, however, the simulation of vehicle behaviours are often overly simplified and implemented as a trivial extension of the network simulator. As a second contribution of this thesis work, a detailed and realistic simulator of vehicle behaviour has been developed based on the car-following and lane-changing models.

As the simulation of vehicle behaviour and that of communication behaviour are different tasks, they should be implemented separately for better modularity and meanwhile they should be seamlessly integrable. As another contribution of this thesis work, the online and seamless

integration of vehicle behaviour simulator and network simulator has been studied. Specifically, a set of APIs has been designed and implemented atop the vehicular behaviour simulator to facilitate its integration with network simulator. Being a concrete example, the integration of ns2 and SUMO, an open-source vehicular behaviour simulator, has been implemented, and applied to simulate an electric vehicular network.

CHAPTER 1. Introduction

1.1 Overview

Vehicular ad hoc network (VANET) communication has recently become an increasingly popular research topic in the area of wireless networking as well as the automotive industries. The goal of VANET research is to develop a vehicular communication system to enable quick and cost-efficient distribution of data for the benefit of passengers' safety and comfort. The most important application of the VANET is for driving safety. It is desired that road conditions affecting driving safety (called *problematic road conditions*), e.g., road segments occupied by colliding or malfunctioning cars, thick ice, animal bodies, or of big holes, can be promptly detected, and vehicles approaching these conditions can be warned beforehand. Research on VANETs has been extensive, but automatic detection of problematic road conditions is barely touched. Most of existing VANET-based systems usually assume drivers detect and report safety related road conditions. The reliance on drivers, however, has salient limitations: drivers may not be willing to perform the duties; even they are, engaging them in these duties may distract driving and thus harm safety. Apparently, when a driver meets a problematic road condition, it is more important for her/him to focus on driving than to report it.

To address the problem, an automatic road condition detection system has been designed. The basic idea is that every vehicle records its footprints periodically, and the footprints are aggregated and analysed to infer the occurrence and locations of problematic conditions.

First of all, footprints generated in geographically close locales are highly related and should be efficiently collected for aggregation and analysis. Due to the mobility of vehicles, the footprint information may be scattered quickly if not collected promptly. The simplest approach is requiring vehicles to report their footprints to certain central station, which however may be

inefficient in communication and even infeasible if no central station exists. Alternatively, vehicles may form a dynamic cluster at each area, and at any time a certain vehicle of the cluster (e.g., the cluster head) aggregates and analyses footprints from other members of the cluster. This however may not work if the VANET is disconnected. Note that disconnection happens frequently on rural roads, in late nights, or in bad weather, exactly where and when the information about problematic road conditions is the most useful for driving safety. To deal with this issue, we propose to deploy inexpensive sensors along the roadside as assistance. Under this architecture, roadside sensors can assist vehicles to efficiently aggregate their footprints when the VANET is connected; when the VANET becomes disconnected, roadside sensors themselves can conduct the aggregation and buffer the aggregation results. Furthermore, the inherent resource heterogeneity between vehicular nodes and roadside sensors are fully leveraged in the design: vehicular nodes take the workload whenever possible, while roadside sensors are used only when necessary.

Secondly, collected footprints should be appropriately analysed to infer the occurrence and locations of problematic road conditions efficiently and accurately. For this sake, the potentially large amount of footprints should be analysed to extract the essential information, and the extracted information should be well organized to facilitate further analysis. To deal with this issue, a tree-based aggregation scheme is proposed for efficiently processing footprints, capturing the most updated information about road conditions and meanwhile keeping historical information. In addition, the occurrence and locations of problematic road conditions should be derived from the footprint information. For this purpose, the relation between the footprint information and the problematic road conditions should be discovered. Based on intensive empirical study, we propose a thoughtful scheme for problematic spot inference. The scheme extracts various factors, such as the distribution of vehicle footprints on the road, the lane-switching patterns, the traffic density, and so on, from the collected information, and synthesizes them together using a simple yet useful empirical formula to infer problematic road conditions with low delay and high accuracy.

To evaluate the system, it's ideal to perform an outdoor experiment. Many wireless tech-

nologies have been proposed for reliable traffic information. But setting up the vehicular network and running an outdoor experiment still has a high cost and potential safety issue. For example, in order to effectively evaluate the proposed system, we need a lot of vehicles to participate in a large road network, which may be risky in real life. For this purpose, software simulations can play a vital role in imitating real world scenarios. To make simulation close to the reality, it is desired to include detailed and realistic simulation of vehicle behaviours under various road conditions, and this is especially needed for studies targeted at driving safety. In the past, however, the simulation of vehicle behaviours are often overly simplified without considering any driving behaviour. For example, the mobile capability that ns2 provides just assume random movements for mobile objects. Towards addressing this issue, we develop a detailed and realistic road simulator according to the intelligent driver model (IDM) [21] and the MOBIL model [26]. The simulator can simulate detailed and realistic behaviours of individual vehicles on multi-lane roads with configurable features such as, dynamic traffic flow, speed limits, on-road obstacles, icy surfaces, speeding, lane-switching and so on. Extensive simulations of the proposed schemes have been performed on top of the simulator to tune system parameters and study design trade-off. The simulator can also be used for other VANET-related research.

However, there are some limitations for standalone vehicular simulators. First of all, trace file size is one big limitation. Simulator dumps all vehicles' state data during the whole simulation period to a trace file. When the traffic volume is 0.8 vehicle/s and simulation time is 1,000,000 sec, we could have a trace file of size greater than 1 GB. So it's hard for us to simulate a longer simulation time or to simulate a larger traffic volume. Secondly, vehicles' movements in ns2 are just a *replay* of what happened in vehicular simulator. ns2 read the whole trace file, find out each vehicle's movements, and set the corresponding vehicle to the pre-defined state at each time step. Lastly but not least, it's hard to achieve on-line interaction between vehicular simulator and network simulator. In order to have interactions between these two simulators, a typical approach is using a TCP connection or other inter-process communication methods between these two simulators.

Although there are already some integrated simulators between vehicular simulator and network simulator, each of them has its own pros and cons. We will discuss more in chapter 3. To address the limitation issues of standalone vehicular simulator, a scheme for the on-line and seamless integration of vehicular simulator and network simulator has been designed. Specifically, a set of APIs have been carefully designed and implemented atop the vehicular behaviour simulator to facilitate its integration with network simulator.

1.2 Our Contributions

In this work, we propose an automatic road condition detection system to auto detect and report safety related road conditions. It has also been studied how to take advantage of inexpensive roadside devices, such as sensors, to facilitate the information collection and analysis, especially in the absence of connectivity between vehicles.

To make simulation close to the reality, we develop our own traffic simulator based on car-following model [21] and the lane-changing model [26], both of which characterise realistic driving behaviours. This standalone simulator can be run to generate vehicle trace data, which are then fed into network simulator. We evaluate the proposed automatic road condition detection system based on these vehicle trace data.

In order to have real-time interaction between vehicular simulator and network simulator, a typical approach would be adopting the inter-process communications methods between these two simulators. There are some limitations for such approaches. Instead, we embed vehicular simulator as one component into ns2, combine two different processes into a single one. Moreover, a set of APIs has been designed and implemented atop the vehicular simulator to facilitate its integration with network simulator.

As one application example, we extend this integrated simulator to support the simulation of electrical vehicle, and evaluate one charging scheme.

1.3 Organization of Thesis

In the rest of the thesis, Chapter 2 presents the preliminaries in which we discuss the traffic models that we use in our standalone vehicular simulator. Chapter 3 first presents the related work on road accident auto detection system, and also presents the state of the art on vehicular simulator and its integration with network simulator. Chapter 4 presents the road accident auto detection system, introduces the collection and inference algorithm, and show how we evaluate the system based on our detailed and realistic trace data generator. We also discuss the limitations of standalone vehicular simulator and some implementation issues. Chapter 5 presents our integration approach for vehicular simulator and network simulator, where a set of APIs have been proposed and implemented for vehicular simulator to facilitate the integration. Finally, chapter 6 concludes the work.

CHAPTER 2. Preliminaries

2.1 Car Following Model

In this simulation, we adopt the Intelligent-Driver Model (IDM) to simulate the longitudinal dynamics, i.e., accelerations and braking decelerations of the drivers. The IDM is a car-following model, i.e., the traffic state at a given time is characterized by the positions, velocities, and the lane index of all vehicles. The decision of any driver to accelerate or brake depends on his own velocity and on the front vehicle immediately ahead of him. Lane-changing decisions, however, depend on his neighbours. Specifically, the acceleration dv/dt of a given driver depends on his velocity v, on the distance s to the front vehicle(if when on free road, i.e., no front car, then $s = \infty$), and on the velocity difference Δ_V(positive when approaching),

dv/dt=a[1-$(v/v_0)^{\delta}$-$(s^*/s)^2$],

where

s^*=s_0+(vT+($v\Delta_V/2\sqrt{ab}$)).

Table 2.1 Parameters of IDM Used as Reference Values

Parameter	Typical Value
Desired velocity v_0	120 km/h
Safe time headway T	1.5 s
Maximum acceleration a	1 m/s^2
Comfortable deceleration b	2 m/s^2
Minimum distance s_0	2 m
Jam distance s_1	0 m
Acceleration exponent δ	4

The IDM has following parameters:

- v_0 - Desired velocity when driving on a free road,
- T - Desired safety time headway when following other vehicles,
- a - Acceleration in everyday traffic,
- b - comfortable braking deceleration in everyday traffic,
- s_0 - Minimum bumper-to-bumper distance to the front vehicle, and
- δ - Acceleration exponent.

Every driver-vehicle unit can have its individual parameter set. For example:

- Trucks are characterized by low values of v_0, a and b.
- Careful drivers drive at a high safety time headway T and also low values of v_0, a and b.
- Aggressive drivers are characterized by a low T in connection with high values of v_0, a and b.

So by varying the parameter values, we can have many different driving behaviours in the simulator, which is close to real-life scenarios.

2.2 Lane-changing Model

Lane changes take place if

- The potential new target lane is more attractive, i.e., the incentive criterion is satisfied, and
- The change can be performed safely, i.e., the safety criterion is satisfied.

In the lane change model MOBIL, we base both criteria on the accelerations on the old and the prospective new lanes, as calculated with the longitudinal model, i.e., the IDM in our case.

2.2.1 Equations

- The safety criterion is satisfied if the IDM braking deceleration imposed on the back vehicle B' of the target lane after a possible change does not exceed a certain limit b_{safe}. This means, the safety criterion
  $acc'(B') > - b_{safe}$
  is satisfied.

- To assess the incentive criterion, we weigh the advantage on the target lane, measured by the increased acceleration (or reduced braking deceleration), against the disadvantage imposed to other drivers, again measured by the decrease acceleration or increased braking deceleration for these drivers. Since we tend to be egoistic, we weigh the disadvantage imposed on other drivers with a politeness factor p whose values are typically less than 1, resulting in following incentive criterion:
  $acc'(M') - acc(M) > p\ [\ acc(B) + acc(B') - acc'(B) - acc'(B')\ ] + a_{thr}$

- As above, acc means the actual IDM acceleration while acc' means the acceleration after a possible change. The car labels M and M' mean "Me" before and after a possible change, respectively, while B and B' mean the back vehicle before and after a possible change, respectively.

- The advantage is measured by "me" acceleration difference $acc'(M') - acc(M)$ after the change, compared to the actual situation.

- The combined disadvantage to the new and old back vehicles is given by the sum $[acc(B) + acc(B')]$ of the accelerations of both vehicles before the change, minus the acceleration sum $[acc'(B) + acc'(B')]$ of these vehicles after the change.

- To avoid lane-change manoeuvres triggered by marginal advantages which can lead to frantic lane hopping, an additional lane-changing threshold a_{thr} has been added to the

balance of the above equation.

2.2.2 Model Parameters and Typical Values

Table 2.2 Parameters of MOBIL Used as Reference Values

Parameter	Typical Value	Remarks
Politeness factor p	$0 \sim 0.5$	see below for detail
Maximum safe deceleration b_{safe}	4 m/s^2	$<$ maximum deceleration: 9 m/s^2
Threshold a_{thr}	0.2 m/s^2	$<$ lowest acceleration: a of IDM

While other lane-change models typically assume purely egoistic behaviour, i.e., $p = 0$, we can model different behaviours by varying factor, p as follows:

- $p > 1 \Rightarrow$ a very altruistic behaviour.
- p in [0, 0.5] $\Rightarrow$ a realistic behaviour: Advantages of other drivers have a lower priority, but are not neglected. Note that this feature means yielding to aggressive drivers is included into MOBIL.
- $p = 0 \Rightarrow$ a purely selfish behaviour. Note that selfish drivers do not ignore the safety criterion!
- $p < 0 \Rightarrow$ a malicious personality who takes pleasure in thwarting other drivers even at the cost of own disadvantages. This may have some interesting game-theoretic consequences. Of course, even those mischief makers do obey the safety criterion.

A special case is given by $p = 1$ and $a_{thr} = 0$. In this case, lane changing takes place whenever the sum of the accelerations of all affected drivers increases after the change, or equivalently, the overall decelerations are minimized.

CHAPTER 3. State of the Art

3.1 Problematic Road Conditions Automatic Detection

Numerous communication and data management protocols have been proposed for pure VANETs composed of only mobile vehicles. But, if the VANET is disconnected, many of the schemes may not keep effective. To overcome this limitation, roadside units (RSUs) assisted systems have been proposed to facilitate inter-vehicle communication, especially for routing [23, 14], data dissemination [35, 19] and improving driving comfort [34, 20]. However, due to the high cost of RSUs it appears impractical to apply the protocols in highway scenarios. Motivated by this, a prototype of hybrid sensor-vehicular networks has been proposed recently. Based on this concept, some safety-related architectures [12, 33] have been proposed. Comparing with our proposed system, they have some salient limitations. On one hand, they require sensors to be connected with each other, which demands for a large number of sensors, making the system expensive. However, in our system we do not require sensors to be connected. On the other hand, in their proposed architectures vehicular nodes are not fully exploited to relieve the workload of sensors and may cause sensors to have their energy depleted in a short time. This problem is addressed in our system through the heterogeneity-aware design, which makes use of vehicular nodes as much as possible and use sensors only when necessary.

There are a few works dealing with incident detection in VANETs. In [7], the author proposes a system, NOTICE, where sensor belts embedded in the roadway, every mile or so, are used to communicate with vehicles to infer the traffic incident from passing vehicles' trajectory, specifically, the lane-changing information. [31] also proposes using NOTICE as highway infrastructure to support planned evacuations. [8] presents how to use table as data structure to analyze possible event on some lane also by using vehicle's lane-changing information and

it is designed to enhance the existing incidents detection techniques. Comparing with these works, our proposed systems have the following advantages. The most obvious and important merit is that our design considers the *heterogeneity* between vehicles and sensors and distribute more workload to vehicles whenever possible. Besides, our roadside facility is very simple (i.e., sensors). Different from [7] and [8], the communication overhead in our system is much lower. This is because we adopt the tree-based aggregation mechanism to express the vehicles' traces, which can greatly save the storage and also mitigate the data exchange between vehicle and roadside facility. This is of critical importance, especially for sensors. The presence of aggregator in our system can eliminate the communication collision when multiple vehicles attempt to send their collected information to the roadside facility. Our scheme makes use of more information to derive incidents. Beside the lane-changing information, our proposed scheme also consider the different lane-changing trend in various traffic density. This can make our detection more reliable with low false positive. Finally, [8] fails to consider the risk range, which may lead to underestimation of false positive rate.

3.2 Standalone Vehicular Simulators

VanetMobiSim [1] extends the CANU Mobility Simulation Environment (CanuMobiSim [2] which is a flexible framework for user mobility modeling). The simulator focuses on vehicular mobility and features realistic automotive motion models at both macroscopic and microscopic levels. At the macroscopic level, VanetMobiSim imports maps from the US Census Bureau TIGER database, or randomly generates them using Voronoi tesselation. At the microscopic level, VanetMobiSim implements mobility models which provide car-to-car and car-to-infrastructure interactions. According to these models, vehicles regulate their speeds depending on nearby cars, overtake each other and act according to traffic signs in presence of intersections.

SUMO(an open source micro-traffic simulator) [3] is conceived to simulate a traffic road network of the size of a city. [15] points out some features of SUMO: collision free vehicle movement, different vehicle types, multi-lane streets with lane changing, a XML-raw-output

containing information about the state of the net for every time step, and so on. SUMO also adopts the car-following and lane-changing models.

MOVE (MObility model generator for VEhicular networks) [17] uses the SUMO environment for the simulation of roads. The output of MOVE is a mobility trace file that contains information of realistic vehicle movements which can be used by network simulation tools such as ns2 or Qualnet. MOVE provides a set of graphical user interfaces that allows the user to generate realistic simulation scenarios without writing simulation scripts as well as learning about the internal details of the simulator.

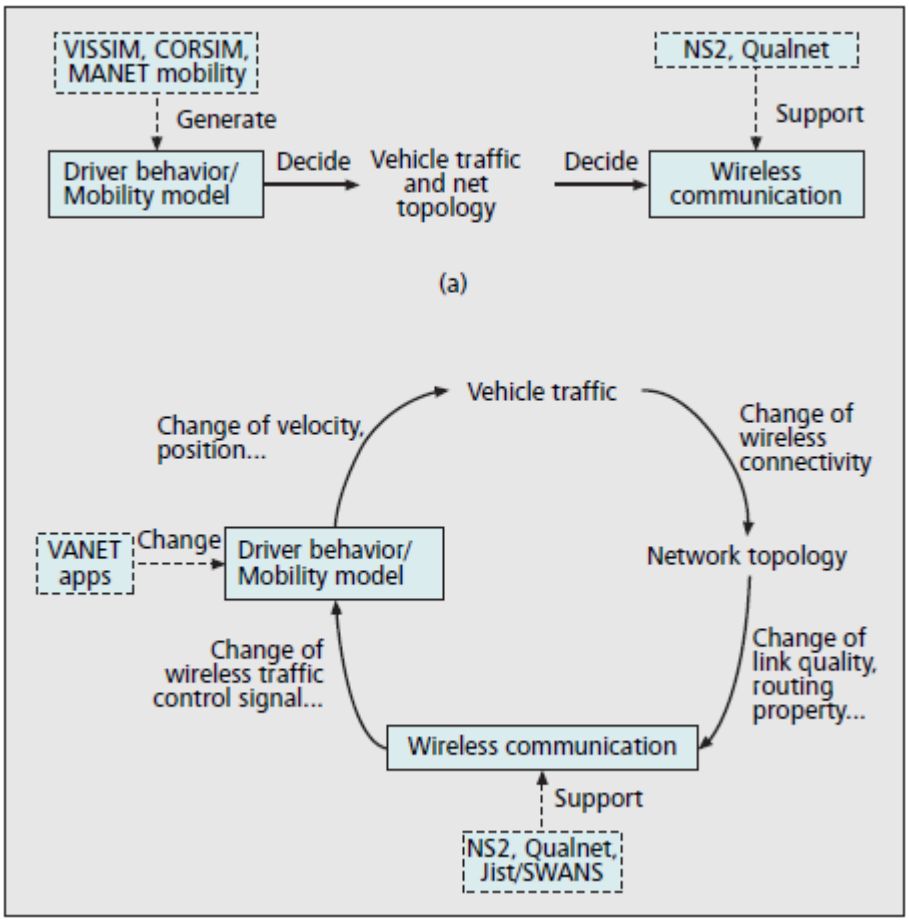
3.3 Integration of Vehicular and Network Simulator

The following table from [10] summarizes the current research on the integration of vehicular simulator and network simulator in the past several years:

Table 3.1 Approaches for Integration of Vehicular Simulator and Network Simulator

Simulation approach	Description	Mobility model
Open-loop, simplistic mobility model	Network simulator with simplistic MANET or macroscopic models	MANET models or macroscopic traffic models
Open-loop, trace driven mobility model	Microscopic simulation such as VISSIM generated vehicle traces fed into network simulator(e.g., NS2, QuaNet)	Microscopic traffic models
Closed-loop, realistic mobility model	Integrating network communication and vehicular traffic simulation, supporting interaction between the two	Microscopic traffic models

The following figure, also taken from [10], shows the information flow of these two VANET simulation approaches: open loop and close loop.



Notes:

(1) In the federated approach, only TraNS is open-source, while VISSIM, CARISMA and CORSIM/QuelNet are all commercial-products.

(2) From [4], we can not see that SWANS is an integration of simulator, just a wireless ad hoc network simulator.

(3) Since MoVES and AutoMesh do not consider the lane-changing behaviour, so we do not survey them.

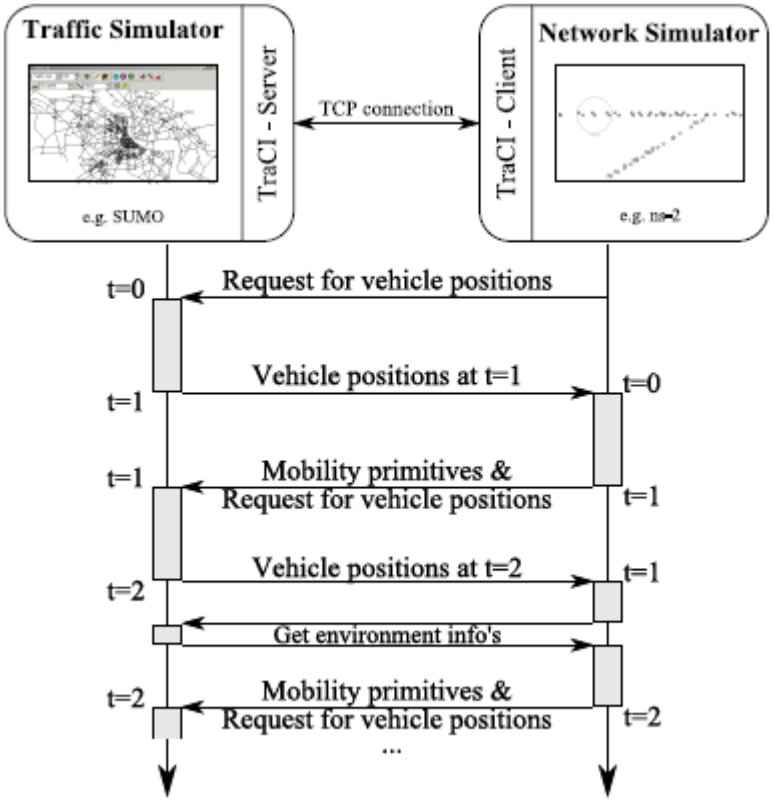
3.3.1 Federated Approach

3.3.1.1 TraNS

[16] is the project of TraNS [5]. The project is open-source. They provide two versions of simulator. One is TraNS (Traffic and Network Simulation Environment) , which is a GUI tool

that integrates traffic and network simulators (SUMO and ns2) to generate realistic simulations of VANETs. TraNS allows the information exchanged in a VANET to influence the vehicle behaviour in the mobility model. For example, when a vehicle broadcasts information reporting an accident, some of the neighbour vehicles may slow down. So, the integrated simulator now is closed-loop, supporting interaction between the two. The other one is TraNSLite, which is a stripped-down version of TraNS suitable for quickly generating realistic mobility traces for ns2 from SUMO.

The vehicular simulator is coupled with the network simulator over a TCP connection by using the Traffic Control Interface (TraCI) [6], as shown from the following figure taken from [30]:



After successful connection, the network simulator acts as a master by sending commands to the road vehicular simulator, that are answered if necessary with the requested data. TraNS has two distinct modes of operation, each addressing a specific need. The first mode, which they term network-centric, as shown in below Figure 3.1 [30], can be used to evaluate VANET

communication protocols that do not influence in real-time the mobility of nodes. One example is user content exchange or distribution (e.g. music or travel information). The second mode, termed application-centric, as shown in below Figure 3.2 [30], can be used to evaluate VANET applications that influence node mobility in real-time, and thus during the traffic simulation runtime. Safety applications (e.g., abrupt breaking, collision avoidance, etc.) are such examples.

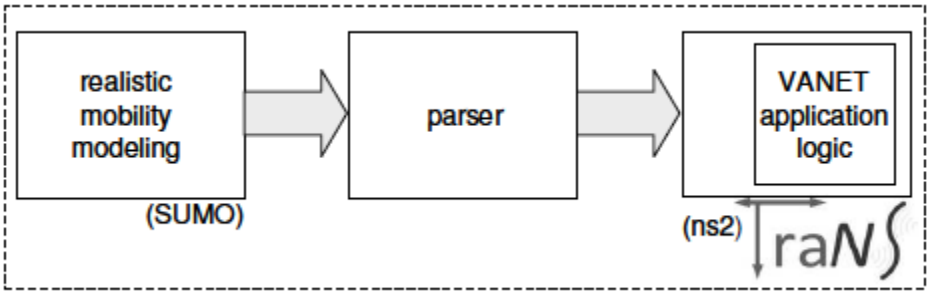


Figure 3.1 Network-centric Simulator

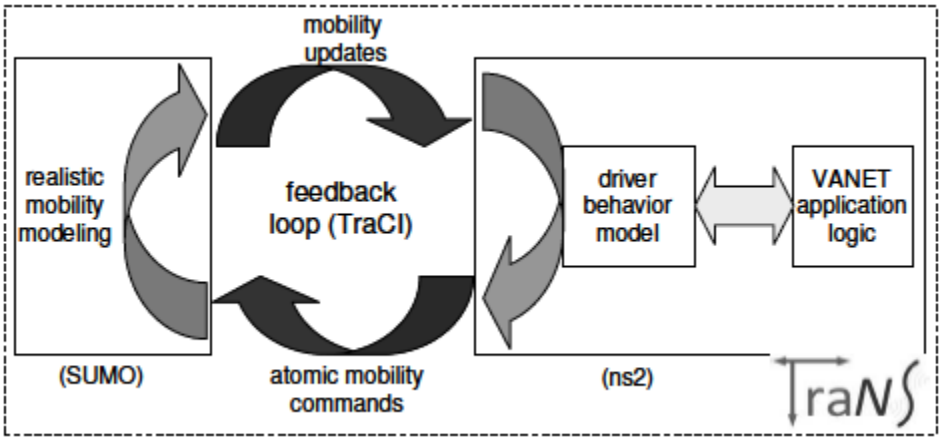


Figure 3.2 Traffic-centric Simulator

As a disadvantage the feedback(via TCP connection) between these two is not as fast as the the NCTUns, which is presented later.

3.3.2 Integrated Approach

3.3.2.1 NCTUns

NCTUns [29] (National Chiao Tung University Network Simulator) is based on Harvard simulator proposed by S.Y. Wang in 2002. NCTUns is purely written in C++ with a powerful GUI support. The latest version is 5.0. NCTUns incorporates traffic simulation (e.g., road network construction and microscopic vehicle mobility models) with its existing network simulation, tightly integrates them together, and provides a fast feedback loop between them.

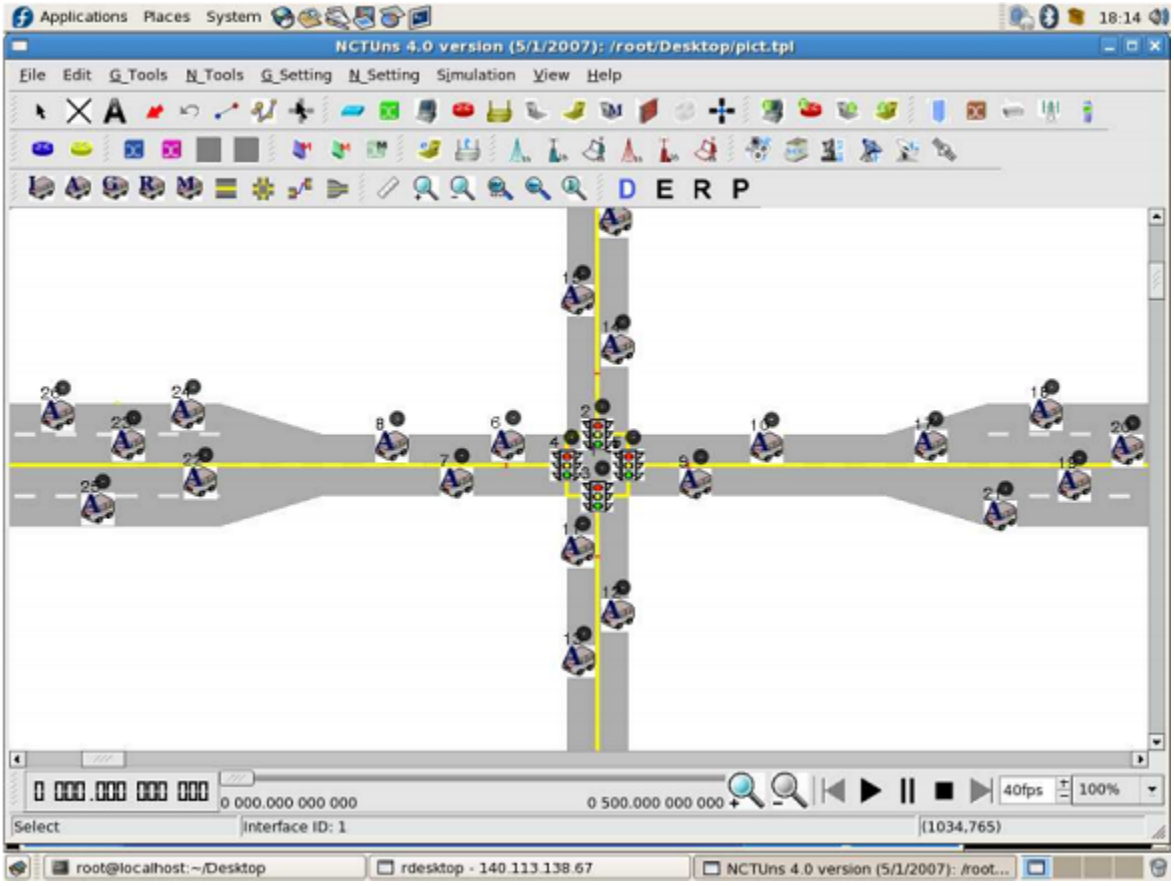


Figure 3.3 A Snapshot of The GUI of NCTUns

As a disadvantage, the mobility component is highly integrated with the network simulator. This makes it hard to utilize realistic road traffic simulators, for example, those developed within the Intelligent Transportation Systems (ITS) community. In other words, it can not utilize any other traffic simulator and network simulator(ex. ns2).

3.3.3 VANET Research Based on Vehicular Simulator of Realistic Model

[13] illustrates that a realistic mobility model and a variety of target environments are essential to ensure that VANET applications meet their performance criteria when deployed. Although trace-based evaluations provide accurate mobility in simulation, they argue for an integrated vehicular traffic and wireless network simulator when VANET applications are expected to influence the mobility of participating vehicles.

[27] and [28] use realistic mobility models to study the performance of routing protocols, analyse the performance of AODV and GPSR and show the performance degradation when directly applying these protocols from MANET to VANET. In [22], the adopted simulator models the behaviour of people living in an area, reproducing their movement (using vehicles) within a workday. They use a 24 hour car trace file to feed into ns2 as mobility movements. As we will discuss in Chapter 4, there is a limitation on how long they are able to simulate.

3.4 Summary

Unlike ns2 dominating the network simulation, there is no such popular and well-adopted simulator in the VANET research. The most popular approach is to let vehicular simulator generate trace file, and then the movement trace is fed into network simulator. Although it seems that the approach to effectively evaluate the safety applications in VANET is using the integrated simulator, only [11] presents how to model safety application in such simulator. We think the challenge is due to the bi-directional interaction between network simulator and vehicular simulator. There is a tradeoff: if both are self-developed, the interaction will be better; but the extension to different traffic models and network simulator will be limited. We need to have a set of well defined interfaces for vehicular simulator if the vehicular simulator expects to be integrated with ns2.

CHAPTER 4. Automatic Detection of Problematic Road Conditions

4.1 Introduction

The motivation of this work is that most of existing VANET safety research assumes drivers detect problematic road conditions, and focuses on subsequent dissemination of the detection. However, the reliance on drivers has salient limitations. For example, drivers may not be willing to perform the duties. Even they are, engaging them in these duties may distract driving and thus harm safety. So, we propose to rely on roadside sensors and vehicle communication to solve this challenge.

4.2 System Overview

The system is composed of vehicular nodes and sensor nodes. Each vehicular node is a communication and computation device such as laptop, PDA and so on. It has two communication interfaces: a long-range WiFi interface for communicating with other vehicular nodes and a short-range Zigbee interface for communicating with roadside sensors. Each vehicular node is also equipped with GPS component to be aware of its position at any time. Due to the mobility of vehicles, vehicular nodes can be connected or disconnected from time to time.

Each wireless sensor is a low-power, low-cost wireless communication and computation device such as Crossbow MicaZ, Telos mote and so on. Sensor nodes are deployed statically on road sides. Each sensor has a short-range Zigbee interface for communicating with vehicular nodes when necessary.They have constrained computation, communication and storage capacity, and are powered by batteries. All road-side sensors can be sparsely deployed along the road and thereby may be disconnected. Note that, we do not assume the existence of more powerful roadside stations because deploying sensors is cheaper. However, the design presented in this

paper is compatible with powerful roadside stations if they exist.

The road can be one-way or two-way, and sensors can be deployed on either side of the road. For simplicity, this paper assumes the road to be one-way and sensors to be deployed on only one side of the road; however, the proposed design can be extended for the two-way road scenario with slight changes.

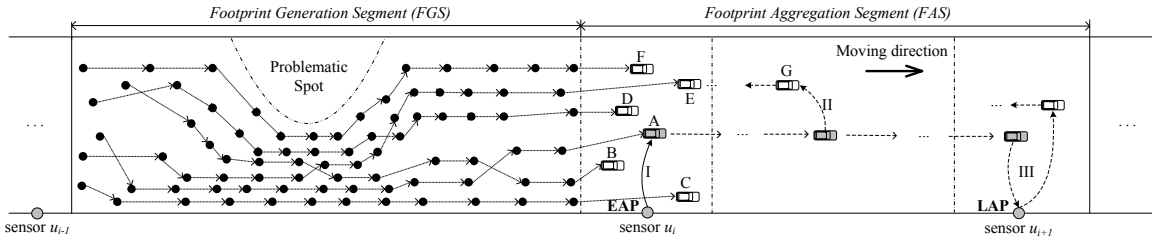


Figure 4.1 System Overview

4.3 Algorithm

Each vehicle records its own footprints periodically and waits to aggregate its footprints with others. Roadside sensors serve as *access points* to facilitate the aggregation. Let sensors $u_0, u_1, \cdots, u_i, u_{i+1}, \cdots$ be the sequence of sensors deployed on one side of a certain highway (e.g., I-35 South) along the driving direction. The highway is divided into segments where the i^{th} segment starts from the point where sensor u_i is deployed and ends at the point where sensor u_{i+1} is deployed. Sensor u_i is responsible for initiating the aggregation of footprints generated during the $(i-1)^{th}$ segment and thus is called the *entering access point (EAP)* for the segment; sensor u_{i+1} is responsible for aggregating and buffering the aggregation results for the footprints generated during the $(i-1)^{th}$ segment and thus is called the *leaving access point (LAP)* for the segment. Hence, sensor u_i is both the EAP of the $(i-1)^{th}$ segment (if $i \geq 1$) and the LAP of the $(i-2)^{th}$ segment (if $i \geq 2$).

Without loss of generality, the following description addresses only how the footprints generated in the $(i-1)^{th}$ segment are aggregated through collaborations among vehicles, sensor u_i (i.e., the EAP of the $(i-1)^{th}$ segment) and sensor u_{i+1} (i.e., the LAP of the $(i-1)^{th}$

segment). To ease presentation (See Fig. 4.1), the $(i-1)^{th}$ segment is called the *footprint generation segment (FGS)* and the i^{th} segment (i.e., the segment bounded by sensor u_i and u_{i+1}) is called the *footprint aggregation segment (FAS)*. The footprint aggregation scheme includes following components:

4.3.1 Collection of Footprints

Each EAP periodically broadcasts a *start of aggregation (SoA)* message within its communication range. The vehicles hearing the SoA message collaboratively elect an *aggregator* to collect the footprints from all of them, and then analyse the footprints. We avoid the simple approach of requiring all vehicles to report their footprints to EAP, because this may incur high communication cost for sensors and shorten its lifetime.

4.3.2 Analysis and Aggregation of Footprints

Footprints generated by each vehicle are stored as a list of pairs: $\{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$. By analysing the footprints it has collected, an aggregator vehicle can infer a set of potential problematic spots called *P-Spots* and aggregate P-Spots into a tree called *PS-tree*.

4.3.2.1 Analyzing Footprints into P-Spots

To facilitate analysis, the footprint generation segment (FGS) is divided into a $M \times N$ grid, where M is the number of lanes and N is equal to the length of the FGS divided by a predefined constant ΔX. The index (x, y) ($x \in \{1, \cdots, N\}$ and $Y \in \{1, \cdots, M\}$) of each cell in the grid represents the approximate position on the road. Fig. 4.2 shows a grid representation of footprints of six vehicles which are collected through the procedure illustrated by Fig. 4.1. The filled cells represent the areas visited by vehicles. The blank or shaded ones represent the areas that have not been visited by any vehicle, and are identified as P-Spots.

Each P-Spot is uniquely denoted as $S_{t|j}$, where t is the unique time stamp standing for the time the collection is conducted (note: it is generated by the EAP) and integer j distinguishes this P-Spot from other P-Spots identified in the same collection. $S_{t|j}$ is represented as $S_{t|j} =$

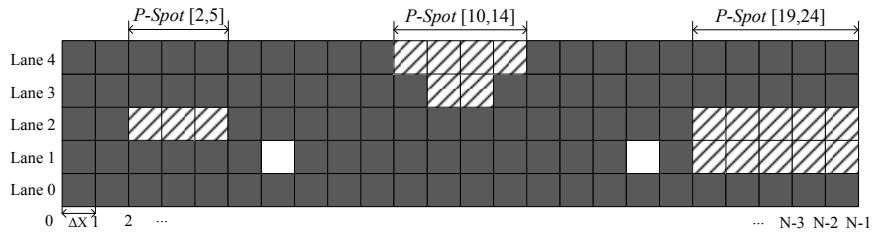


Figure 4.2 Road Segment Grid

$([s_x, e_x], [s_y, e_y], D)$, where $[s_x, e_x]$ stands for the bound of the P-Spot in the X-dimension (i.e., along the moving direction), $[s_y, e_y]$ stands for the bound of the P-Spot in the Y-dimension (i.e., perpendicular to the moving direction), and D stands for the associated information the content of which will be detailed when presenting the method for inferring actual P-Spots. *For simplicity, P-Spots can be represented by the bound in the X-dimension in our paper*, as the example shown in Fig. 4.2. Note that scattered small spots (e.g., $[6, 7]$ and $[17, 18]$ on Lane 1 in Fig. 4.2) are most possibly not actual problematic spots but caused by random movement of vehicles. To exclude these spots, the length of a P-Spot should be greater than a certain minimum length (L_{min}) and its width is at least one lane. Any spot that has a length less than the minimum length (e.g., 10 meters) is ignored.

4.3.2.2 Tree-based Aggregation of P-Spots

Keeping a set of P-Spots is inefficient for further analysis or storage. A tree-based algorithm for aggregating the P-Spots is hence proposed based on the following idea: Information about P-Spots identified in different collections is aggregated together and organized into a tree, where each leaf node represents the most updated information of a P-Spot, while the nodes from the leaf to the root record historical information related to the P-Spot. As aggregation continues, the scope of each P-Spot is adjusted.

The proposed tree-based aggregation algorithm is described as follows. At the beginning, the aggregator vehicle initializes a PS-tree with only a root node denoted as $T_0 = ([0, R_x], [0, R_y], D)$, where R_x and R_y are the length and the width of FGS, respectively. From

the footprints reported by all vehicular nodes, including the aggregator itself, the aggregator gets a set of P-Spots using the afore-described footprint analysis method (Section 4.3.2.1). Then, each P-Spot in the set, denoted as S, is aggregated into the current PS-tree one by one with the following algorithm.

Case I: If S has no overlap with any leaf node in the current PS-tree, S is added as a leaf node of the root.

This case indicates that a new P-Spot is detected. Hence, the new P-Spot is recorded for further investigation.

Case II: For each leaf node T such that S and T overlap and S is not fully covered by T (i.e., $S \cap T \neq \emptyset \wedge S \not\subseteq T$), two leaf nodes are added: a new P-Spot which is the intersection of S and T is added as a leaf node of T, and meanwhile S itself is added as a leaf node of the root. See Fig. 4.3.

This case indicates one of three possibilities. The first possibility is, there is an actual problematic spot covered by both S and T. Therefore, a more precise estimation of the problematic spot can be inferred, which is the intersection of S and T. Hence, a new P-Spot which is the intersection of S and T is added as a leaf node of T. The second possibility is, the actual problematic spot covered by the old P-Spot T has changed, and it is now covered by S. In this case, S is independent of T, and hence S should be added as a leaf node of the root. The third possibility is that there is no actual problematic spot within the scopes of S and T; that is, S and T are blanks due to random movement of vehicles. Since the first two possibilities are likely to occur, we conservatively assume both of them to be true to avoid missing detection.

Case III: For each leaf node T such that S is the same as or fully covered by T, S is added a leaf node of T. See Fig. 4.3.

This case also indicates three possibilities. The first possibility is that there is a problematic spot covered by both S and T. Since S is covered by T, S is a more accurate estimation of the problematic spot. Hence, S is added as a leaf node of T. The second possibility is that there is no actual problematic spot covered by either S or T; that is, S and T are blanks caused

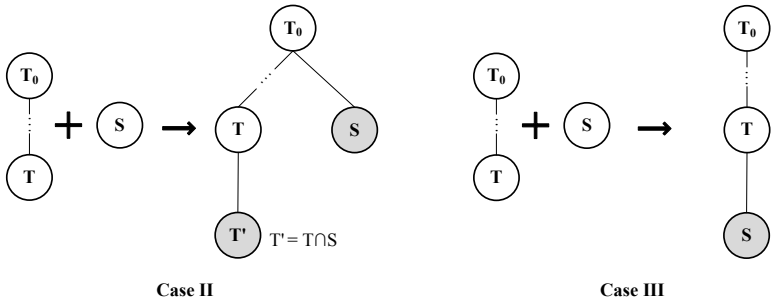


Figure 4.3 Tree Operations in Case II and III

by random movement of vehicles. The third possibility is that a problematic spot is covered by T but not in S. According to the footprint analysis depicted before, that problematic spot must be covered some other P-Spot, which will be considered in the corresponding aggregation. Since the first possibility is likely to happen, we conservatively assume it to be true to avoid missing detection.

Case IV: If a leaf node T of the tree has no overlap with any P-Spots that have been aggregated, it indicates that P-Spot T has been passed by some vehicles recently. Hence, T is no longer a P-Spot and is marked as a *dead* leaf node. Dead nodes and nodes whose descendent leaf nodes are all dead are removed.

Fig. 4.4 shows an example of tree aggregation during two rounds of collection. For simplicity, we only consider the range of P-Spots in X-dimension (i.e., ignoring their $[s_y, e_y]$ and D fields). Thus, a P-Spot can be represented as $[s_x, e_x]$. Fig. 4.4 (a) lists the P-Spot sets found in each of the two rounds of collection. Fig. 4.4 (b) is the initial status of the tree. According to Case I of the above algorithm, P-Spots in S_1 are directly appended to be leaf nodes of root T_0 as illustrated in Fig. 4.4 (c). Fig. 4.4 (d), (e) and (f) show the aggregation procedures for the second round of collection. Fig. 4.4 (d) illustrates the operations on leaf nodes while Fig. 4.4 (e) illustrates the operations on the root. Specifically, for $S_{2|1} = [50, 70]$, it overlaps with $T_{1|2}$ which falls into Case II. According to the algorithm, the intersection of $S_{2|1}$ and $T_{1|2}$, i.e., $T_{2|1} = [60, 70]$, is appended as a new leaf node of $T_{1|2}$, and meanwhile, $S_{2|1} = [50, 70]$ itself appended as a child of the root and is renamed as $T_{2|4}$. For $S_{2|2} = [80, 100]$, it is fully covered

by $T_{1|2}$, which falls into Case III. According to the algorithm, it is appended as a leaf node of $T_{1|2}$. For $S_{2|3} = [110, 160]$, it also falls into Case II. Therefore, the intersection of itself and $T_{1|3}$ is added as a leaf node of $T_{1|3}$, and meanwhile $S_{2|3}$ itself is added as a leaf node (i.e., $T_{2|5}$) of the root. Further note that, leaf node $T_{1|1}$ has no overlap with any P-Spots identified in the second round of aggregation. Hence, it is removed from the tree according to Case IV. Fig. 4.4 (f) illustrates the final result of the two rounds of aggregation.

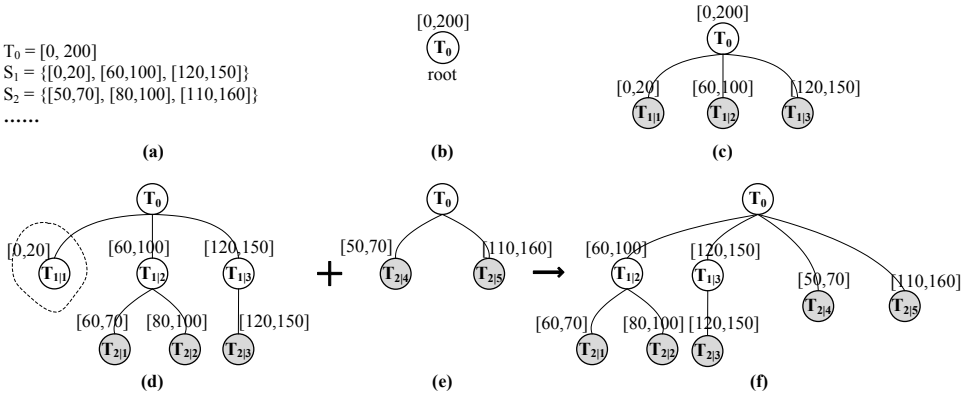


Figure 4.4 Example of Tree Construction

4.3.3 Accumulative Aggregation of Footprints

As presented above, footprints generated at the FGS are aggregated by aggregator vehicles round by round, where each round is initiated by the EAP. The footprints aggregated at different rounds are correlated and should be aggregated together for inferring actual problematic spots. We call such aggregation *accumulative aggregation*. The following presents how the accumulative aggregation is performed in an efficient and heterogeneity-aware manner. The key idea is that the accumulative aggregation should be conducted among vehicles as much as possible: when the VANET is disconnected, it is performed by roadside sensors (i.e., LAPs); when the connectivity of the VANET is restored, the duty should be handed back to the VANET again.

4.3.3.1 Accumulative Aggregation within the VANET

After aggregating footprints reported by its neighbors, each aggregator vehicle v will find an opportunity to transfer the aggregated result (i.e., its PS-tree produced by the aggregation algorithm described above) to a follow-up vehicle w. Ideally, w is also an aggregator that is elected later than v, and thus the two PS-trees aggregated in two different rounds can be further aggregated at w. Even if w is not an aggregator, it can serve a relay to transfer the PS-tree of v to a later aggregator. If each aggregator can successfully transfer its PS-tree to a follow-up vehicle, the trees will keep being aggregated accumulatively within the VANET until the actual problematic spots are identified using the method to be elaborated in Section 4.3.4.

To find a follow-up vehicle to transfer its data, each aggregator vehicle v periodically advertises its availability of data and its current location. Upon receiving the message, a follow-up vehicle will reply immediately if it is also an aggregator. Otherwise, the follow-up vehicle will reply after a random backoff of which the length is *inversely* proportional to its distance from v, if it does not hear any reply during the backoff. Vehicle v transfers its data to the node which replies the earliest.

The basic idea of aggregation of two PS-trees is as follows: Each tree is associated with a unique time stamp issued by the EAP, and the time stamp is increased as time elapses. Therefore, the two trees to be aggregated are of different ages. To aggregate the trees, the one with older age is treated as the current tree, while the one with younger age is broken into individual leaf nodes, from which P-Spots can be restored and classified into sets of P-Spots. Then, these sets of P-Spots are aggregated into the current tree according to the algorithm described in Section 4.3.2.2.

4.3.3.2 Accumulative Aggregation at the LAP

It is likely that an aggregator vehicle v cannot find a follow-up vehicle to transfer its PS-tree. This happens when v is disconnected from its follow-up vehicles. If this issue is not addressed appropriately, the PS-tree may get lost after vehicle v moves away. Hence, the LAP is designed to serve as the backup site to buffer and aggregate PS-trees that cannot be

continuously stored and aggregated in the VANET. The procedure is as follows.

The LAP periodically broadcasts a *notification of leaving* message to notify vehicles passing by that they are to leave the footprint aggregation segment (FAS). If a node that stores a PS-tree but fails in transferring it to a follow-up vehicle has heard the message, it will send its tree to the LAP (See arrow III in Fig. 4.1 for example), where LAP will temporarily store the aggregation result. In this case, if the LAP already has stored some aggregation result (i.e., a PS-tree), it will aggregate that result with the one received using the method described in Section 4.3.3.1.

4.3.3.3 Transferring the Aggregation Duty from the LAP to the VANET

To save the resources of roadside sensors, it is desired that the duty of accumulative footprint aggregation is transferred back to the VANET whenever the VANET is able to perform it. However, if the handoff is performed too aggressively, there may be no benefit. For example, if the VANET is just partially connected or the VANET becomes disconnected soon after the aggregation duty is transferred to it, the duty may have to be transferred to the roadside sensor again, resulting in bigger resource waste. To address this problem, the following scheme is designed to factor in the current connectivity of the VANET in duty transferring.

When a vehicle v passes the LAP, it will try to relay the PS-tree stored in the LAP back to VANETs. This process will be launched if the following heuristics metric, called *relay condition*, is satisfied:

$$\widehat{T} = \frac{d}{\widehat{v}} e^{\alpha \widehat{\rho}} \geq \theta \tag{4.1}$$

where d is the distance from vehicle v to the furthest backward vehicle in its current connected partition (as illustrated in Fig. 4.5), which can be acquired by broadcasting a short query message backward. $\widehat{v}$ and $\widehat{\rho}$ are the average speed (m/s) and traffic volume [1] (veh/s) respectively. θ is the threshold. $\widehat{T}$ represent the expected time duration that the relayed data can be kept in VANETs. If $\widehat{T}$ reaches θ, the leaving regular node will fetch the tree from LAP and forward it back to the further vehicle within the FAS.

[1] Traffic volume(veh/s) = Traffic density(veh/m) $\times$ Speed(m/s)

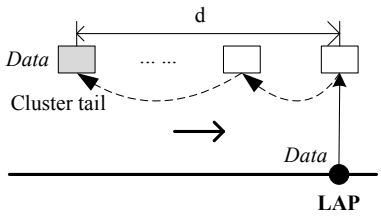


Figure 4.5 Relay Aggregated Data to VANET

Two traffic statistics $\widehat{v}$ and $\widehat{\rho}$ can be acquired in two ways. One way is from preloaded digital maps available on the vehicular nodes, which may provide traffic statistics such as traffic density and vehicle speed on roads at different time of the day. This is based on the fact that the traffic density and average speed on roads are usually stable at the same time of different days. The other way is to use the roadside sensors to gather the information. Specifically, when the aggregators collect the footprints of various vehicles, they can easily estimate the average speed and number of vehicles reporting the footprints. This information can be sent to the sensors later when the short-range channel is available. Based on the information, the sensor can dynamically update those statistics.

As shown in Fig. 4.5, $d/\widehat{v}$ gives an estimation of how long the data transferred back to the VANET can be kept in VANET if there is no follow-up vehicle. Nevertheless, it is possible that some follow-up vehicle may appear in the vicinity of the tail of the current connected partition, which is obviously related to current traffic volume. Thus, we add an *exponential* [2] factor $\exp(\alpha\widehat{\rho})$ to adjust our time estimation. α reflects the degree that traffic volume affects $\widehat{T}$. In addition, $d/\widehat{v}$ is necessary since it is possible that even in some dense traffic scenario the leaving vehicle may still have no neighbor. In this case, it is of no benefit for the vehicle to transfer data from the LAP.

[2]It is shown in the existing work [32] that traffic volume is exponential to inter-vehicle arrival time.

4.3.4 Inference of Actual Problematic Spots

Each time when a round of aggregation has completed, the aggregator, which is either a vehicle or the LAP, runs an algorithm to infer the occurrence and locations of actual problematic road conditions. The inference is conducted based on information stored in the PS-tree. Particularly, whether a potential problematic spot (P-Spot) is an actual problematic spot is inferred by considering the following factors:

- *How long the P-Spot has been identified.* A spot that has no vehicle passing is very likely to be problematic. However, the information collected may not be complete, and it is possible to falsely identify a normal spot that has not been used by any vehicle only during a short period time as problematic. To avoid such false detection, it is necessary to factor in how long the P-Spot has been regarded as problematic.

- *Lane-switch pattern.* If there is a problematic spot, it is very likely to observe vehicles switch lanes before approaching the spot. Hence, the lane-switching pattern is highly useful for inference.

- *Traffic density.* Traffic density plays an important role in considering the above two factors. On one hand, when the traffic density is low, a spot not passed by any vehicle is a weaker indicator of problematic spot than when the traffic density is high. On the other hand, lane-switching before a spot is a stronger indicator of problematic spot when the traffic density is low than when it is high.

Taking these factors into account, the proposed algorithm computes a carefully-designed weight function. If the value of the weight function exceeds a certain threshold, a problematic road condition is inferred to have occurred and its location is estimated. In the following, what information is stored in the tree and how to make use of the information for inference are presented in more detail.

4.3.4.1 Information in PS-tree

As mentioned above, each node on the PS-tree specifies a potential problematic spot (P-Spot). For each branch from the root to a leaf node, the leaf node specifies the most specific PS-Spot, while the P-Spot specified by a non-leaf node fully covers the P-Spots specified by its descendants. In addition to the scope of P-Spot, extra information related to the P-Spot is also stored in each node. The information stored in each node is extracted from the footprints collected in one round and has the following format:

$$D = \langle id, t, n, L \rangle,$$

where id is the unique id (i.e., $< vehicle\_id, index >$) assigned to each P-Spot. t is the timestamp issued by the EAP and represents the time when the footprints are collected. n is the total number of vehicles reporting their footprints during the round of collection. L records the lane-switching pattern of all the reporting vehicles and has the following format:

$$\{(l_1^0, l_1^1), (l_2^0, l_2^1), \cdots, (l_m^0, l_m^1)\},$$

where m is the number of vehicles that have switched lane when approaching the P-Spot specified by the node, and for each $i \in \{1, \cdots, n\}$, l_i^0 is the original lane where the i^{th} vehicle stays while l_i^1 is the lane where the i^{th} vehicle switches to. Fig. 4.6 shows an example, where vehicles V_1 switches from lane 3 to lane 4 when it approaches the P-Spot and V_2 switches from lane 3 to 2. However, vehicles V_3 and V_4 do not switch the lane. Thus, $L = \{(3, 4), (3, 2)\}$.

id and t are used when two tree are aggregated together. n and L are used to infer the problematic road conditions.

4.3.4.2 Synthesizing Factors to Infer Problematic Road Conditions

To exploit the information stored in the PS-tree to infer the occurrence and locations of problematic spots, for each P-Spot specified by a leaf node, the following weight function is evaluated:

$$W = \sum_{k=1}^{R} \left(\exp\left(-\frac{\lambda}{\frac{n'_k}{w}} \right) + \exp\left(-\frac{1}{\frac{n_k - n'_k}{w'}} \right) \right) \quad , 0 < \lambda < 1 \tag{4.2}$$

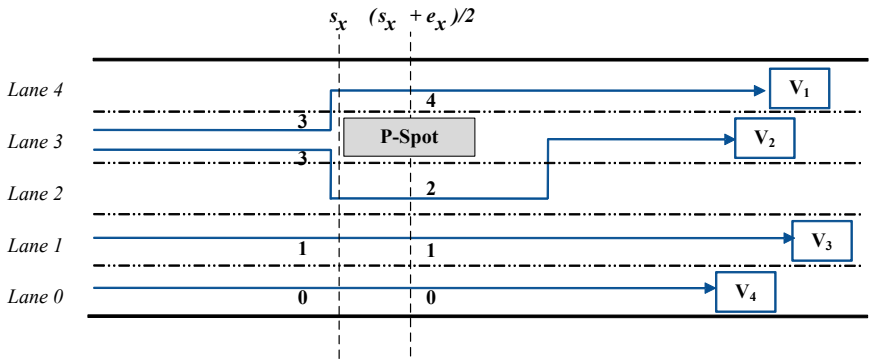


Figure 4.6 Lane-switching Information

where R is the total number of nodes on the tree from the leaf node to the root. w is the width of the P-Spot specified in the leaf node, and $w' = M - w$ where M is the total number of lanes on the road. For each node k on the path, n_k is the total number of vehicles that are recorded, and n'_k is the number of vehicles that have recorded to switch lanes to bypass the P-Spot. λ is a *predefined* system parameter. For the example shown in Fig. 4.6, $w = 1$, $w' = 4$, $n_k = 4$, and $n'_k = 2$. If $W \geq \delta$, where δ is another system parameter, the P-Spot specified by the leaf node is considered as an actual problematic spot. The value of δ is tuned via simulations reported in Section 4.4.

The design is based on the following considerations:

Factoring in how long the P-Spot has been identified. As discussed before, how long the P-Spot specified by a leaf node has been identified is an important factor. This is indicated by the number of nodes on the path from the leaf node to the root since each such node corresponds to a round of footprint collection that has identified the P-Spot. Hence, Eq. (4.2) assigns a weight to each of these nodes to present the belief that the P-Spot is an actual problematic spot based on the information stored in the node, and then sum these beliefs up to form the final belief.

Factoring in lane-switching pattern. The weight for each round of collection is composed of two parts. The first part collects evidence for the existence of problematic spot from vehicles that have switched lanes while the second part collects evidence from those that have not.

In the first part, n'_k/w is the number of lane-switching vehicles on the lanes with P-Spot, which indicates the average chance of lane-switching on these lanes. The larger is the chance the stronger indicator of the existence of problematic spot. In the second part, $(n_k - n'_k)/w'$ indicates the average chance of keeping on the lanes without P-Spot. Also, the larger is the chance the stronger indicator of the existence of problematic spots. Among these two chances, the first chance is more directly related to the existence of problematic spot than the second one; hence, a larger weight is assigned to the first part, which is implemented by using coefficient $\lambda \in (0, 1)$ in the function.

Factoring in the impact of traffic density. Drivers have a higher tendency to switch lanes to find better routes when the traffic density is high than when it is low. To the contrary, they may tend to keep the same lane when the traffic density is low since no other vehicle is moving in front, which can be also observed from our developed road simulator. Hence, if a driver switches lane before approaching a P-Spot when the traffic density is low, this is a stronger indicator that the P-Spot is true than when this happens when the traffic density is high. This trend has been reflected in the designed function. Specifically, as the number of vehicles collected in each round increases, the average weight value assigned to each vehicle decreases. Thus, the lane-switching vehicle contributes larger weight in sparse traffic than it does in dense traffic.

Finally, we define that $\exp(-\lambda\frac{w}{n'_k}) = 0$ and $\exp(-\frac{w'}{n_k - n'_k}) = 0$ when $n'_k = 0$ and $n_k - n'_k = 0$ respectively. Obviously, no weight should be given when no vehicle appears.

4.3.5 Propagation of Detection Results

Once a problematic spot is inferred by either sensor or vehicle, the warning message can be propagated backwards via VANETs by using a certain communication protocol, such as [25, 18, 9]. Under our system model, such protocol can be extended by taking advantage of the presence of the roadside sensor. Specifically, the sensor node can store the warning messages when the connection is not available and resume the propagation when the connection becomes available later.

4.4 Performance Evaluation

To evaluate the proposed detection system in the realistic environment, we have conducted comprehensive simulations based on ns2, which has been extended to support multi-channel system model, and our developed vehicular simulator. Different traffic scenarios, e.g., various traffic density and various problems on the road such as roads of big holes, icy surface and dead animals, are simulated. The vehicular simulator outputs traffic trace data in files, which are used as input to the ns2-based simulator.

4.4.1 Vehicular Simulator

The vehicular simulator dumps all trace data into a trace file, then we have a parser in ns2 to parse these files to extract the state of each vehicle at any sample point, i.e., x coordinate, y coordinate and velocity.

4.4.1.1 Software Requirements

We specify a road topology by supplying a configuration file to specify how the road should be constructed. Here is an example:

```
HighwaySection:
0.(0,50) (2000,50) 1 22 {1,1,1,1,1} {} {3,970,60,false} {4,970,60,false}
1.(2000,50) (4000,50) 1 22 {0,0,0,0,0} {} {}
---
ConnectLanes:
0(0)-1(0)
0(1)-1(1)
0(2)-1(2)
0(3)-1(3)
0(4)-1(4)
---
UpstreamToDownstream:
```

1 0

———

We have to specify highway sections in the Highway Section section. For example, there are two sections in this topology. Each section is specified in the format of "StartPosition EndPostion IsSectionAbleToTravelThrough SpeedLimit SetOfLanes OptionalObstacles".

Particularly, {3,970,60,false} means there is an obstacle on Lane 3, starting at 970m from the beginning, Length=60m, and the obstacle can't be passed through. This is a P-Spot that we should be able to infer by our system.

Then we connect lanes between two adjacent sections. For example, "0(0) - 1(0)" means lane 0 of section 0 is connected to lane 1 of section 1.

And finally, we specify the sequence of sections from upstream to downstream.

Three types of vehicles are specified in this simulator:

Table 4.1 Three Types of Vehicles in Simulator

Vehicle Type	Characteristics	Weight
Ambulance/Police	highest priority and others having to yield	0.1%
Truck	long length, low acceleration	9.9%
Normal Cars	typical values	90%

Note that normal cars are not set to a predefined parameter values; instead, they are associated with different and random values in the given range of a parameter.

Input: The flow rate of high way traffic is varied from a v/h to b v/h (assume a=200, b=2000). We adopt the IDM and MOBIL model for the car model and lane-change model, respectively. These different traffic units have different parameters according to their characteristic.

Output: All vehicles' states are deployed into a well-defined format of trace file. We also provide the capability to query a specific car by vehicle's id. Here is an example of trace file:

```
222222      5.49    10.00     3.00
  0         5.49    39.84    29.84
```

1	5.49	69.68	29.84
2	5.49	99.53	29.84
3	9.49	129.08	29.84
4	16.19	158.15	29.84
5	25.49	186.50	29.84
6	37.31	213.89	29.84
7	51.55	240.10	29.84
8	68.11	264.92	29.84
9	86.84	288.14	29.84
10	107.58	309.57	29.84
11	130.18	329.05	29.84

...

4.4.1.2 Software Modules

There are following software modules:

- **Vehicle** - The vehicle module contains the IDM driver model implementation and the MOBIL implementation.
- **Road** - A road module knows its collection of road segments, along with information needed to draw their locations.
- **RoadSegment** - A road segment module knows its collection of lane segments, along with information needed to draw their locations. A road segment also has a speed limit, and an ID number.
- **Lane** - A lane module is a container for all vehicles moving on that lane and is responsible to move vehicles around.

4.4.2 Performance Metrics

Two categories of performance metrics are measured. On one hand, the system performance is measured by event detection *delay* and *false positive rate* (FPR). The optimal values of system parameters δ and θ are sought for various traffic scenarios. On the other hand, resource consumption on roadside sensors is measured in terms of the *energy* consumed in communication and the *space* used for data storage.

Note that miss-detection rate is not considered since miss-detected problems are always found to be problems with short duration ($< 30sec$) in the simulations.

4.4.3 Experimental Settings

The following table lists fixed parameters.

Table 4.2 Experimental Settings

Simulation time	1,000,000 sec
Road type	One way
Lane	5
Average speed	30m/s
Sensor broadcast period	2 sec
Footprint sampling period	1 sec
Range of ZigBee interface	100m
Range of WiFi interface	250m
Risk range	50m

To evaluate the proposed system in different traffic scenarios, two traffic settings, namely, the road length and the traffic volume (density) are varied. In the simulation, the footprint generation segment (FGS) and the footprint aggregation segment (FAS) have the equal length. Unless otherwise specified, we set the length of FGS and FAS to 2000m, i.e., a roadside sensor is deployed every 2000 meters. The traffic volume varies from $0.05veh/s$ to $0.8veh/s$, covering a wide range of traffic dynamics on highway.

To simulate wireless communication in realistic environment, each moving vehicle randomly launches communication session using its long range interface in addition to the communication required by the automatic detection system. The purpose is to introduce the background

interference, which may occur in reality. Specifically, each vehicle launches a communication session every certain time interval, which follows an *Exponential* distribution with the mean of 5 seconds. In each session, the vehicle sends out a certain number of packets, following a *Poisson* distribution with the mean of 10. The data size of each packet is uniformly selected from 0 to 2312 bytes, a typical range of data size used in IEEE 802.11.

To optimize the performance of the designed system, the best values for system parameters should be found. Two essentially important parameters in the system are the detection threshold δ and the relay threshold θ. Other two parameters are λ and α. Through experiments, $\lambda = 0.06$ and $\alpha = 10$ have been found to result in a reasonable granularity on detection performance when varying δ and θ. Thus, λ is set to 0.06 and α is set to 10.

4.4.4 Evaluation Results

4.4.4.1 Impact of Threshold δ on Performance

Fig. 4.8 shows the impacts of the threshold δ on the delay and the false positive rate (FPR) of detection. Since we found that δ does not affect the energy consumption of roadside sensors in the simulations, the corresponding result is not shown here. According to the figure, as δ increases the detection delay increases linearly while FPR decreases exponentially. This is because more footprints are collected and analyzed to infer the problem as δ increases, which results in longer delay but higher accuracy. As $\delta = 10$ delivers both acceptable delay and FPR when the traffic volume is high ($0.8veh/s$), moderate ($0.4veh/s$) or low ($0.1veh/s$), δ is fixed at 10 in the following simulations.

4.4.4.2 Impact of Threshold θ on Performance

According to the figure, the strategy of transferring data from LAP to the VANET has no noticeable effect on the performance when the traffic volume is not low, because in this case the connection of VANETs is stable and, therefore, the aggregation data are rarely handed to LAP. When the traffic volume is low, as illustrated by the figure, the detection delay increases as θ increases while the energy consumption at roadside sensors decreases. This is because larger

θ decreases the frequency of data exchange between vehicles and the LAP, which leads to less energy consumption at roadside sensors and meanwhile, may separate correlated information at two places (i.e., the LAP and the VANET) and thus increases the detection delay. Since the impact of θ on energy consumption is more significant than that on delay, it is fixed at 500 in the following simulations to achieve good energy efficiency.

4.4.4.3 Energy and Storage Costs at Roadside Sensors

Fig. 4.10 illustrates the communication and storage costs at each roadside sensor as the traffic volume varies. The first figure shows the date exchange rate (byte/s) between each sensor and vehicles. It decreases rapidly as the traffic volume grows since more and more workload is migrated from the sensor to the vehicular nodes. This trend is also reflected in the second figure, which shows the percentage of problematic conditions detected finally by the vehicular node and by the sensor, respectively. As the traffic volume increases, most of the detection is conducted by the vehicular nodes. The third figure shows the average and maximum size of storage that each roadside sensor has used for detection purpose. The storage size is shown to be affordable by current generation of sensors (i.e., it is less than 500 bytes), and it decreases as traffic volume gets higher, because the number of P-Spots found in each collection becomes smaller as more footprints can be collected in each collection.

The results of this set of simulations have verified that the proposed system can adaptively distribute workload among vehicular nodes and roadside sensors in a heterogeneity-aware manner.

4.5 Limitation of Standalone Vehicular Simulator

A standalone simulator has the following limitations:

- Big trace file size. Because we dump all vehicles' state during the whole simulation period to a trace file. when the traffic volume is 0.8 veh/s and simulation time is 1,000,000 sec, we have a trace file of size more that 1 GB. So it's hard for us to simulate for long time or for large traffic volume.

- Vehicles' movements in ns2 being simply *replay*. ns2 reads the whole trace file, finds out each vehicle's movements during the simulation period, and sets the *vehicle* in the ns2 to the pre-known state at each time step.
- No on-line interaction between vehicular simulator and network simulator.

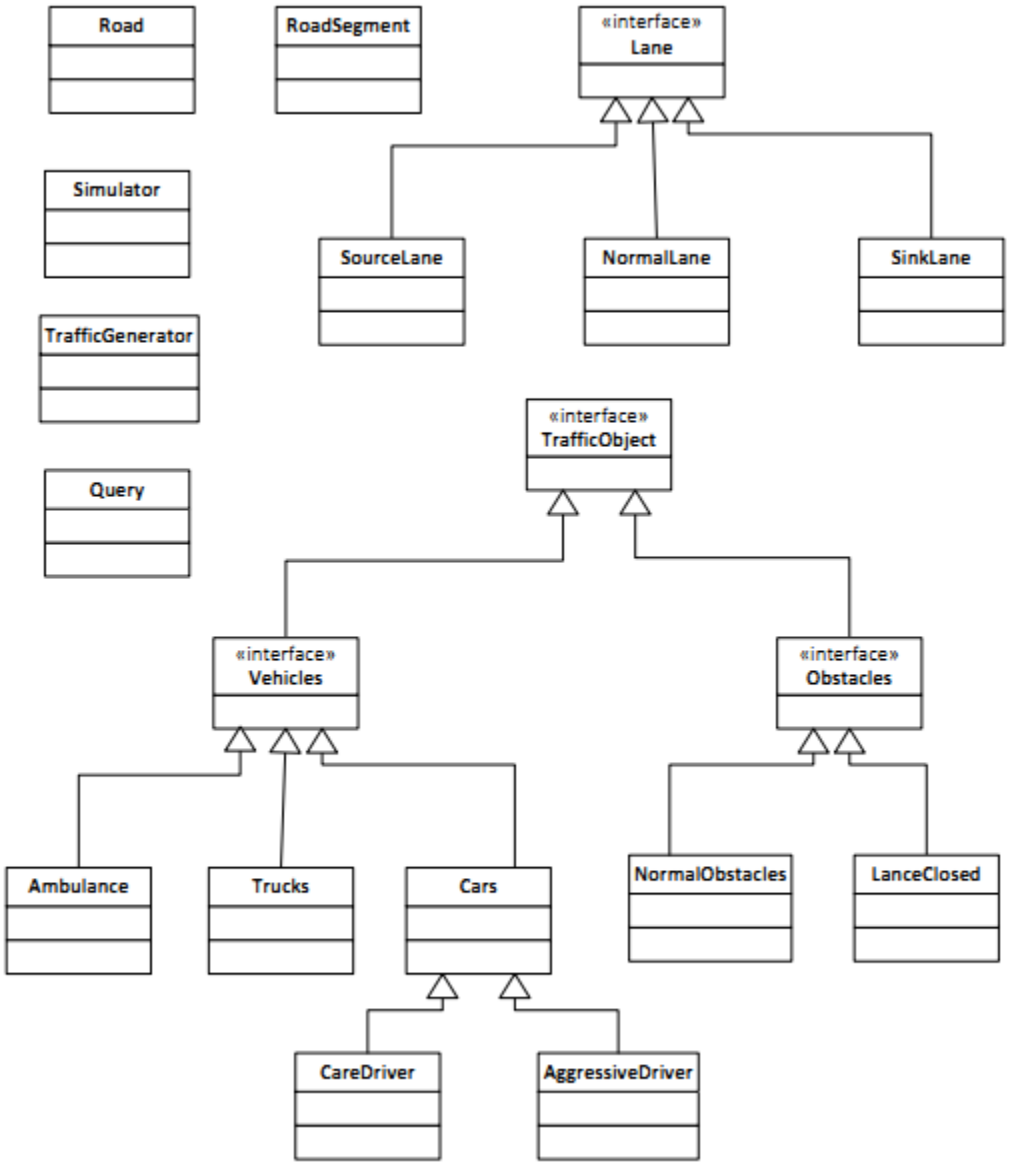


Figure 4.7 Architecture of Traffic Simulator System

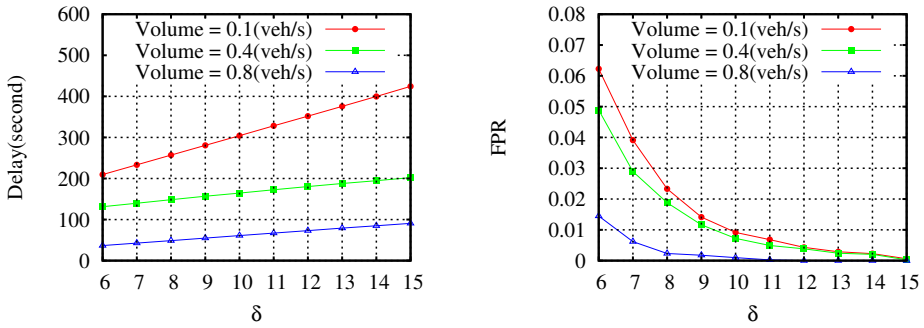


Figure 4.8 Impacts of δ on Detection Performance

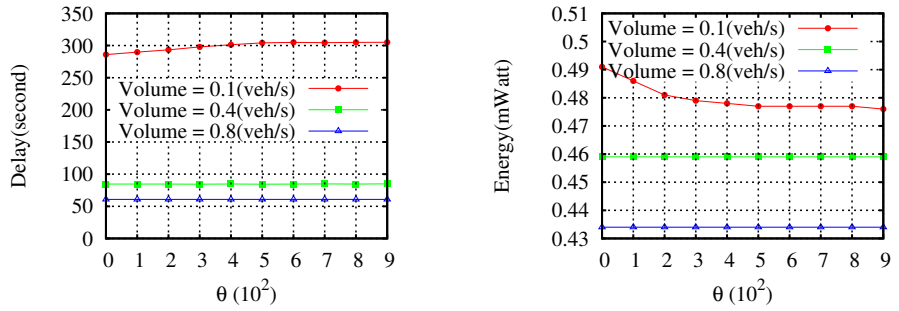


Figure 4.9 Impacts of θ on Detection Performance

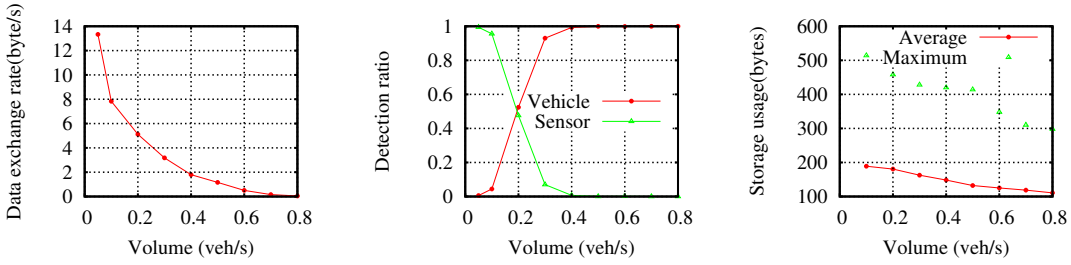


Figure 4.10 Communication and Storage Costs at Each Roadside Sensor

CHAPTER 5. Integration of Vehicular Simulator and ns2

The drawback of standalone vehicular simulator is that we have to generate a static trace file before we can run ns2 VANET simulation. In order to get trace data on-line, we need an integration between vehicular simulator and ns2. In the work, we integrate SUMO, an open-source vehicular simulator, with ns2.

5.1 SUMO Overview

To run a simulation in SUMO, input files are needed. All SUMO input files are the XML format, so they are both human and machine readable. The following describes which files are needed for a simulation and how the information can be obtained and delivered to SUMO.

5.1.1 Road Network

To set up the road network, a network file is needed. This file contains junctions, links, lanes, connections and traffic lights. As this is a very complex dataset, network files are not meant to be build by the user. Instead, a tool called *netconvert* is provided to the network file from a number of input files. The most important input files are as follows:

1. Node file: This file contains information on junctions, i.e., their positions in the grid and their type. For example:

   <nodes>
   <node id="beg" x="0" y="-200" type="priority"/>
   <node id="end" x="0" y="200" type="priority"/>
   <node id="absEnd" x="0" y="400" type="priority"/>
   </nodes>

2. Edge file: In this file all the data on likes and lanes is stored: position (either as coordinates or as a pair of nodes), number of lanes, maximum speed, priority for unregulated junctions, length and a set of coordinates to describe their shape. Furthermore the vehicle types which are allowed or disallowed on specific lanes are stored in this file. E.g.
   <edges>
   <edge id="middle" fromnode="beg" tonode="end" nolanes="1" speed="36"/>
   <edge id="end" fromnode="end" tonode="absEnd" nolanes="1" speed="36"/>
   </edges>

3. Connection file: In here specific information on allowed turning manoeuvres is stored, i.e. which link connects to which link or on a more detailed level which lane connects to which lane. This file is optional. If not supplied, system will connect lanes between adjacent road segments in a default way.

All this information constitutes the bare road network. These data can be fetched from the data model and then transformed into the necessary XML format. Then the netconvert is called with these files as input to generate the network file. For example:
netconvert –xml-node-files=hello.nod.xml –xml-edge-files=hello.edg.xml –output-file=hello.net.xml

Other information regarding the road network is not a part of the network file itself but it is stored in so-called additional files. Additional files contain information about:

1. Traffic light programs: a representation of the light changing cycles on particular traffic lights.

2. Variable speed signs: These allow adapting the maximum speed on a lane. This could be used to simulate road blocking scenarios. When setting the maximum velocity to 0, this blocks the lane because vehicles in the model strictly adhere to the speed limits.

3. Bus stops: The position and length of bus stops is also defined in additional files.

5.1.2 Traffic Demand

Traffic demand and route data is defined together with vehicle type data in a so called routing file. This file contains vehicle type data, traffic routes and traffic demand. As this is also a very complex dataset network files are not meant to be build by the user. Instead provides a tool called netconvert which generates the network file from a number of input files. The input files are:

1. Road network file: this file is generated by some input files as discussed in 4.2.1.1.

2. Flow file: Traffic flow contains info like traffic volume from some starting point in a specified period. E.g.

   <flowdefs>

   <flow id="0" from="middle" begin="0" end="1000" no="100"/>

   </flowdefs>

3. Turns: Describe the distribution of vehicles moving from one edge to another.

   <turns>

   <interval begin="0" end="10000">

   <fromedge id="middle">

   <toedge id="end" probability="1"/>

   </fromedge>

   </interval>

   </turns>

   Then JTRROUTER is called to generate a routing file.

   Jtrrouter –net-file=tri.net.xml –flow-definition=tri.flows.xml –turn-definition=tri.turns.xml

   –output-file=tri.rou.xml

5.2 Integration of SUMO and ns2

5.2.1 Approach for Integration

Since SUMO and ns2 are two different processes, so the communication between these two falls into the category of Inter-Process Communication.

Let's review these methods first to see their limitations.

5.2.1.1 File

Using this approach, SUMO writes the network state at a given point to the file, then ns2 is informed that the write is already done, ns2 proceeds to read info from file. This approach involves at least two IOs, first IO is that SUMO write all info to a file, then ns2 read these info back to memory. If we have a large road network, this approach could incur performance penalty. Moreover, there is one overhead of communicating the state of file, whether it's read or write done.

5.2.1.2 Signal

A signal is a limited form of inter-process communication used in Unix, Unix-like, and other POSIX-compliant operating systems. Essentially it is an asynchronous notification sent to a process in order to notify it of an event that occurred. In this example, SUMO send a signal to ns2, which has previously registered a signal handler, that routine is executed. We can't use this method independently to communicate the vehicles' movements between SUMO and ns2, since signal is just a notification.

5.2.1.3 Socket

Most interprocess communication uses the client server model. These terms refer to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server, typically to make a request for information. We can design ns2 to be the server while SUMO to the client. ns2 make a request for all vehicles' next movements on every simulation step.

These are the standard routines for establishing SUMO and ns2 communications.

On SUMO side:

1. Create a socket with the socket() system call

2. Connect the socket to the address of ns2 using the connect() system call

3. Send and receive data. There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

On ns2 side:

1. Create a socket with the socket() system call

2. Bind the socket to an address using the bind() system call.

3. Listen for connections with the listen() system call

4. Accept a connection with the accept() system call.

5. Send and receive data

As we can see here, there are a lot of work involving establishing communications. For every traffic simulator that wants to be integrated with ns2, they have to have establishing routine ready. While integrating, we have to test the connections between client side(SUMO) and server side(ns2).

5.2.1.4 Shared Memory

Shared Memory is an efficient means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access. Shared memory is another method of interprocess communication (IPC) whereby 2 or more processes share a single chunk of memory to communicate. There is overhead to manage the shared memory by both SUMO and ns2, moreover, there is limitation on the share memory size, the on-line documentation says the limit is 4 $MBytes$.

5.2.1.5 Message Queue

Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the

same time. Messages placed onto the queue are stored until the recipient retrieves them. This method does not fit into the need of real time communications between SUMO and ns2.

5.2.2 Design and Implementation

Fig. 5.1 shows the design of our approach for integration. Instead of running these two simulators independently as two independent process, we propose a design to integrate traffic simulator as a component into the process of network simulator. This approach brings us the following benefits:

1. removing the need for inter-process communication,
2. reducing the memory usage by two processes, and
3. easier and more efficient to pass data around since both reside in a single address space.

Fig. 5.2 shows an example of integration between SUMO and ns2.

5.2.2.1 Initialize SUMO from ns2

We need supply a list of arguments to the SUMO program:

```
int main(int argc, char **argv)
```

After the initialization works, SUMO enters the simulation loop and logs every vehicle's movements into a file till the end.

As shown on the right side of Fig. 5.2, we start SUMO from the inside of ns2, right after ns2 starts.

In order to implement this change, we change the entry point from

```
int main(int argc, char **argv)
```

to

```
void sumo_main::initSystem(int argc, char **argv)
```

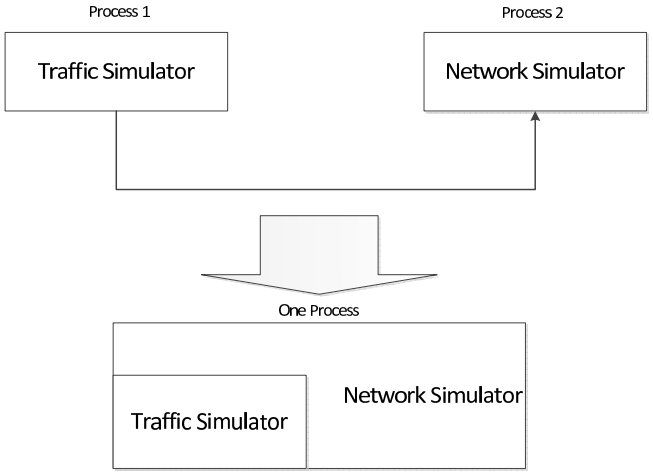


Figure 5.1 Design of Integration

To minimize the changes to SUMO, we keep arguments being passed in unchanged. This is important, as traffic simulator can provide almost the same entry point for both standalone system and integrated system.

The other thing that we need to do is that SUMO should enter a pause state after initialization, as shown in Fig. 5.3. The existing kick-off run command for SUMO is within the *main* method:

```
// simulate
ret = net->simulate(begin, end);
```

We just remove this one statement then SUMO enter a pause state.

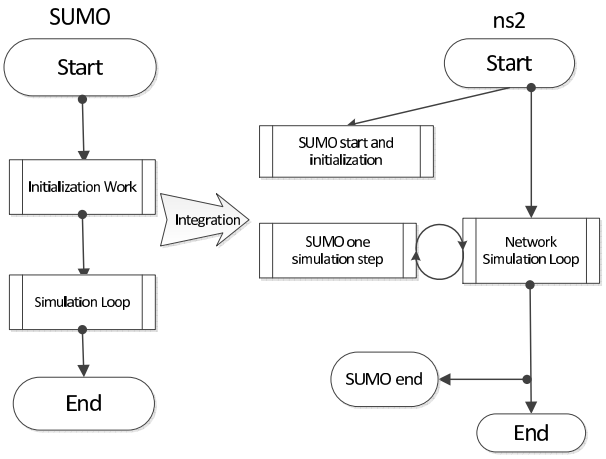


Figure 5.2 Example of Integration Between SUMO and ns2

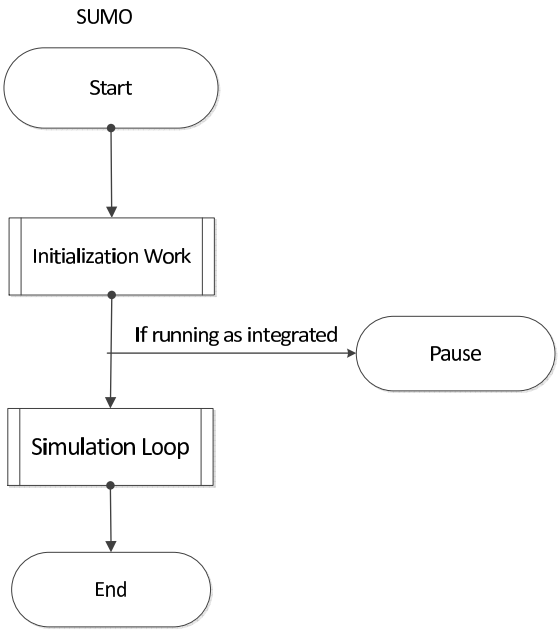


Figure 5.3 SUMO Enter into Pause State

Here is an example for initialize system:

// init sumo traffic simulation system

char *str[] = “sumo”, “-c”, “/home/xuejialu/ns-allinone-2.34/ns-2.34/traffic/hello.sumo.cfg”;

sumo.initSystem(3, str);

5.2.2.2 One Step At a Time

SUMO needs to run one step of simulation at a time, because we don't want to replay all of them during the simulation period, instead, we only want to replay the last step in ns2. There is a simulation loop in SUMO that keep simulating each step until the end:

```
int MSNet::simulate(SUMOTime start,
 SUMOTime stop) {
    // the simulation loop
        ...
    do {
        ...
        simulationStep();
                ...
    } while (!quitMessage);

    // exit simulation loop
    closeSimulation(start);
    return 0;
}
```

As shown in Fig. 5.4, we remove the loop and only simulate one step, and also need to return the simulating results back to ns2.

```
vector<string> MSNet::GetNextStepInfo(SUMOTime start, SUMOTime stop)
{
        ...
```

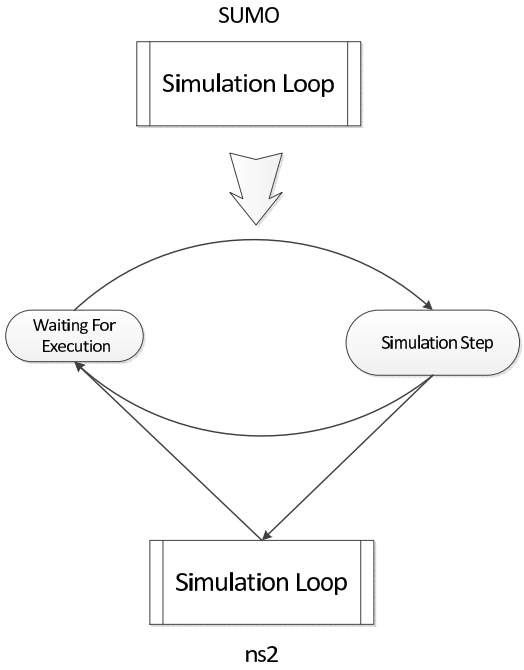


Figure 5.4 Breakdown Loop to Steps

```
        result = simulationStep();
        ...
        return result;
}
```

And this internal function is wrapped to one function that we provide to outsiders:

```
vector<string> sumo_main::ExecuteNextStep()
{
    vector<string> stateInfo;
    stateInfo = net->GetNextStepInfo(begin, end);
    MapStringID2IntID(stateInfo);
```

```
    return stateInfo;
}
```

For every step, ns2 has to instruct SUMO to simulate one step and get all vehicles state, i.e. position and velocity, and then move corresponding mobile nodes in ns2 to its correspond state, so on so forth. The return result is a list of string, each string has the format: *id x y velocity*, we have a parser in ns2 to parse all these results for each vehicles.

Since we don't have a loop now, we need to tell caller whether there is more steps of simulation ahead, so we provide one function in SUMO to return whether there is more steps to come:

```
// Test whether we still has more steps to run
bool sumo_main::HasMoreSteps()
{
        ...
}
```

5.2.2.3 Return Results

SUMO will not output data to a file, instead, return a string representing the traffic network state to ns2, so ns2 can extract all vehicles' state.

SUMO log every simulation step into a XML, we need to modify SUMO to return the results instead to ns2.

To save space, we ignore much details in the following code:

```
void MSXMLRawOut::write(...)
{
        ...
        a loop to writeEdge();
        ...
}
```

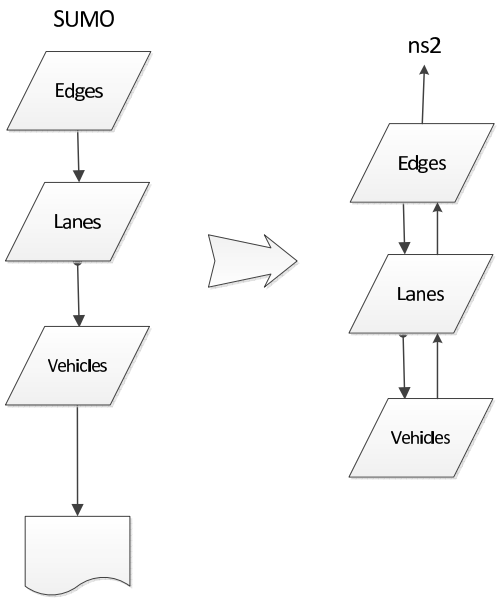


Figure 5.5 Returns Result to ns2

```
void MSXMLRawOut::writeEdge(...)
{
        ...
        a loop to writeLane();
        ...
}

void MSXMLRawOut::writeLane(...)
{
        ...
```

```
        a loop to writeVehicle();
        ...
}

void MSXMLRawOut::writeVehicle(...)
{
        //real work here
}
```

As shown in Fig. 5.5,we remove the logging to file, but instead we add a parameter of a reference to a vector of strings to every above functions, then we call the MSXMLRawOut::write function with an empty vector of string and return this vector after function return:

```
vector<string> stateInfo;
MSXMLRawOut::write(...stateInfo...);
...
return stateInfo;
```

5.2.2.4 Close System

This is a simple wrapper on the internal function to close the simulation:

```
// Close the system
void sumo_main::closeSystem()
{
        ...
        net->closeSimulation();
        ...
}
```

After all works are done, we call this function to clean up SUMO system.

5.2.3 Implementation Issues

1. Different vehicle id format. In ns2, we represent each mobile object, i.e, each vehicle as a positive integer. While in SUMO, since they have the concept of flow, e.g, on a two way west-east-direction road, one flow is traffic from west to east, while the other flow is traffic from east to west. So, they represent each vehicle id as $flowid\_vehicleid$, e.g, 0_1 represents id=1 in flow 0, so on so forth.

   To make this difference transparent to ns2, we transform the vehicle id into the id format in ns2. We assume each flow contains at most 1000 vehicles at a single moment, and before returning results back to external caller, we transform base on this formulation: $flowid * 1000 + vid$, e.g 0_1 map to 1, 1_1 map to 1001.

2. SUMO writes every vehicle's state into a XML file, we change to return all vehicles' state at a given step to ns2:

```
// Execue next step, and return the timestep's info
vector<string> sumo_main::ExecuteNextStep()
{
        ...
}
```

   Each vehicle's state is represented as a string in this format: "$vid\ x\ y\ v$", where vid is the vehicle id, x is the vehicle's x position, y is the vehicle's y position, and v is the vehicle's velocity. ns2 will use these state info to update the corresponding mobile object's state.

3. ns2 invokes ExecuteNextStep on SUMO to get next step's state for all vehicles. There are two approach that ns2 knows when will be the end of simulation. One approach is that ExecuteNextStep returns *null* string, the other approach is that ns2 explicitly ask SUMO whether there is more steps to come before invoke ExecuteNextStep. We adopt the second approach because it's easier and more efficient to implement in SUMO.

5.3 A Set of APIs for Vehicular Simulator

5.3.1 Basic Set of APIs

Here are a basic set of APIs that we propose to facilitate other traffic simulators to be integrated with ns2.

initSystem(int argc, char **argv);

This is an easy API, just change the main entry point to this API, and do not kick off a simulation loop within the main method, i.e, just pause the system and waiting for next instruction.

vector<string> ExecuteNextStep();

Decommission the simulation loop to one step at a time. Instead of writing results to a log file or to the screen, return all vehicles' state as a vector of strings, where each string contains a vehicle's state in the format of "*id x y velocity*".

bool HasMoreSteps();

Return a bool value to the caller to indicate whether we still have more steps to simulate.

void closeSystem();

This is a simple API to clean up the system, release any resources.

5.3.2 Advanced Set of APIs

To support more advanced simulation, e.g, to set a vehicle's next movement from network simulator, we proposed some more advanced APIs:

bool SetMovement(string vehicleState);

Vehicular simulator should be able to accept instructions for a specified vehicle's next movement, e.g, in a system, the vehicle should be informed that there is some obstacles in the front of road, and suggested a new movement for vehicle. The format is a string of vehicle's movement: "*id x y velocity*". Traffic simulator returns a bool value to indicate whether the specified vehicle is going to act as specified.

VehicleObject* GetVehicle(int vid);

Vehicular simulator should be able to return a pointer to the vehicle object by specifying a vehicle's id *vid*. In some scenarios, network simulator needs to know more details of a vehicle object in order to get more info out of a vehicle, in this way, the vehicle in network simulator can do some real work based on more detail info from its corresponding vehicle part in traffic simulator.

5.4 Application Example

Coexistence of electric vehicles(EV) with the emerging smart grids has been recently an attractive research topic. Recently, a number of studies have been undertaken regarding EV charging. In [24], a scheme is proposed to minimize the waiting time for EV charging in a large scale road network. In order to support simulation of proposed scheme in EV charging network, we extend the integrated simulator to include the ability of simulating charging station and EV. To minimize the input to simulator, we reuse the current input file for edge of road network to specify where the station is. The following is an example:

<edges>

<edge id="leftBegToStationBegEdge" fromnode="leftBeg" tonode="stationBeg" nolanes="2" speed="30"/>

<!–station edge 1–>

<edge id="stationEdge" fromnode="stationBeg" tonode="stationEnd" nolanes="3" speed="30"/>

<edge id="betweenStationEdge" fromnode="stationEnd" tonode="stationBeg2" nolanes="2" speed="30"/>

<!–station edge 2–>

<edge id="stationEdge2" fromnode="stationBeg2" tonode="stationEnd2" nolanes="3" speed="30"/>

<edge id="betweenStationEdge2" fromnode="stationEnd2" tonode="stationBeg3" nolanes="2" speed="30"/>

<!–station edge 3–>
<edge id="stationEdge3" fromnode="stationBeg3" tonode="stationEnd3" nolanes="3" speed="30"/>
<edge id="stationEndToRightEndEdge" fromnode="stationEnd3" tonode="rightEnd" nolanes="2"
speed="30"/>
</edges>

As we can see here, we prepend "stationEdge" to the edge that have charging station, simulator will base on these info to determine whether a vehicle has reached station, here is implementation:

```
bool MSVehicle::amIReachStation()
{
    // enter next lane && it's station edge
    if( myState.myPos > myLane->getLength()
    && myLane->getEdge().getFollower(0)>getID().find("stationEdge")
    != std::string::npos)
    {
        return true;
    }

    return false;
}
```

As a prototype, we only implement the simple charging scheme in this stage. First, the vehicle computes whether to be able to drive to next station without charging, Second, if vehicle needs to charge, then vehicle computes the charging time. The battery pack will be fully charged if vehicle cannot reach destination, otherwise, will only be partially charged to be able to reach destination. Here is partial implementation:

```
/* Compute charing time */
double MSVehicle::chargingTime(double dis_to_next_station)
```

```
{
        ... // ignore here
        //Cannot reach the destination if fully charged
        if(energy_dest >= capacity)
                timeToCharge = (capacity - energy)/(16.8*1000);
        //Can reach the destination if partially charged
        else
                timeToCharge = (energy_dest - energy)/(16.8*1000);

    return timeToCharge;
}
```

CHAPTER 6. Conclusion and Future Work

6.1 Summary of Thesis

In this thesis, we studied the automatic detection of problematic road conditions, the construction of a vehicular simulator, and integration of vehicular simulator with network simulator(i.e., ns2).

The main contributions of this dissertation are summarized as follows.

1. We designed an automatic detection system for problematic road conditions. Most of existing works rely on driver's involvement to help detect problematic road conditions. We designed a heterogeneity-aware system that takes advantage of inexpensive sensors for its communication and storage ability.

2. We designed and implemented a standalone vehicular simulator based on realistic car-following model (IDM) and lane-switching model (MOBIL). In order to evaluate our proposed detection system in a more realistic environment, we designed a simulator that takes into account realistic driving behaviours. We designed the simulator to deliver high performance, since some VANET research requires a lot of vehicle data in order to effectively evaluate the proposed system.

3. We designed and implemented an integrated vehicular simulator and network simulator. Unlike the federate approach which relies on TCP communications between two simulators, we adopt the approach of tight integration of the vehicular simulator to network simulator. Specifically, we simplified the SUMO vehicular simulator, stripped unnecessary components, and let ns2 treat this SUMO traffic simulator as a component in the same process. We not only reduced the overhead of TCP communications, but

also improve the overall performance and remove the limitations of standalone vehicular simulator.

4. We proposed and implement a set of APIs for vehicular simulators that want to be integrated with a network simulator. We proposed two sets of APIs. A basic set of APIs is for the basic integration of vehicular simulator and network simulator. While an advanced set of APIs enables deep integration of the simulators. We presented a concrete example of integration between SUMO and ns2.

5. We extended the integrated simulator to support the simulation of electric vehicles. With the growing research interest on this area, this integrated simulator will play an important role in the VANET research.

6.2 Future Work

Supporting bi-directional communications between vehicular simulator and network simulator is an important and challenged work. There are some VANET researches that require sending commands to vehicles, e.g., to reduce speed or to change lane. Vehicles in our integrated simulator only react to the surrounding vehicles, so we would like to add the ability to the system that allows external input to vehicles. On the API level, the vehicular simulator should be able to expose interfaces to allow commands to be passed on to a specific vehicle.

We have an ongoing research to address the issue of minimizing total waiting time of all electric vehicles in a road network. One charging scheme requires vehicle to determine whether to charge or not when reaching a station. Vehicle receives charging stations' information via network communication. The network communication part has to be simulated in ns2, while the vehicular behaviour has to be simulated in SUMO. So, we need to make decisions of charging in ns2, while inform the decision to the corresponding vehicle in SUMO. Moreover, we should be able to switch from different charging schemes in the simulation. This will help us to easily simulate different charging schemes in different road network.

Simulating of large scale of road network is another interesting research direction. Since

there are a lot of potential interactions between vehicles on different road segments, how to apply parallel processing to simulator is very interesting and challenged.

We also plan to publish this ns2-adapted-component into ns2 community, so that the VANET research community will be benefited from realistic vehicular simulating capabilities to evaluate their research in a realistic environment.

ACKNOWLEDGEMENTS

I would like to thank Dr. Wensheng Zhang for his consistent guidance, patience and help throughout this research and the writing of this thesis. His deep insights and valuable directions inspired me to keep on exploring this research and pursuing a higher standard. Without his continuous encouragement, guidance and help, I would not have completed this work.

I would also like to thank my committee members Dr. Ying Cai and Dr. Daji Qiao for their efforts, valuable advices and contributions to this work.

Deep thanks to my friends Hua Qin and Yanfei Wang at the Department of Computer Science for their warm help and guidance so that I could finish this work in time. Last but not least, I express my hearty appreciation to my beloved parents and wife for their ceaseless love, concern and understanding.

BIBLIOGRAPHY

[1] `http://vanet.eurecom.fr`.

[2] `http://canu.informatik.uni-stuttgart.de/mobisim/`.

[3] `http://sumo.sourceforge.net/`.

[4] `http://jist.ece.cornell.edu`.

[5] `http://trans.epfl.ch/`.

[6] M. Raya H. Hellbruck S. Fischer A. Wegener, M. Piorkowski and J.-P. Hubaux. Traci: An interface for coupling road traffic and network simulators. In *CNS08*, Ottawa, Canada, April 2008.

[7] M. Abuelela, S. Olariu, and M. Weigle. Notice: An architecture for notification of traffic incidents. In *Proceedings of IEEE 67th Vehicular Technology Conference*, 2008.

[8] M. Abuelela, S. Olariu, and Gongjun Yan. Enhancing Automatic Incident Detection Techniques Through Vehicle To Infrastructure Communication. In *11th International IEEE Conference on Intelligent Transportation Systems*, 2008.

[9] A. Agarwal, D. Starobinski, and T. D.C. Little. Exploiting Downstream Mobility to Achieve Fast Upstream Message Propagation in Vehicular Ad Hoc Networks. In *Mobile Networking for Vehicular Environments (MOVE)*, 2007.

[10] H. Du D. Ghosal C. Chuah B. Liu, B. Khorashadi and M. Zhang. Vgsim: An integrated networking and microscopic vehicular mobility simulation platform. In *IEEE Communications Magazine*, May 2009.

[11] H. Du D. Ghosal C. Chuah B. Liu, B. Khorashadi and M. Zhang. Vgsim: An integrated networking and microscopic vehicular mobility simulation platform. In *IEEE Communications Magazine*, May 2009.

[12] J. Bohli, A. Hessler, O. Ugus, and D. Westhoff. A secure and resilient wsn roadside architecture for intelligent transport systems. In *WiSec'08*, 2008.

[13] D. Choffnes and F. Bustamante. An integrated mobility and traffic model for vehicular wireless networks. In *ACM VANET 05*, September 2005.

[14] Y. Ding, C. Wang, and L. Xiao. A static-node assisted adaptive routing protocol in vehicular networks. In *VANET'07*, 2007.

[15] C. Gorgorin et al. An integrated vehicular and network simulator for vehicular ad-hoc networks. In *Proc. 20th Euro. Simulation Modeling Conf.*, Oct 2006.

[16] M. Piorkowski et al. Joint traffic and network simulator for vanets. In *MICS 2006*, Zurich, Switzerland, Oct 2006.

[17] K.-C. Lan F. Karnadi, Z. Mo. Rapid generation of realistic mobility model for vanet. In *WCNC07*, March 2007.

[18] F. Farnoud and S. Valaee. Reliable Broadcast of Safety Messages in Vehicular Ad Hoc Networks. In *INFOCOM '09*, 2009.

[19] N. Frangiadakis, D. Camara, and F. Filali. Virtual Access Points for Vehicular Networks. In *Mobilware'08*, 2008.

[20] R. Lu, X. Lin, H. Zhu, and X. (Sherman) Shen. SPARK: A New VANET-based Smart Parking Scheme for Large Parking Lots. In *INFOCOM '09*, 2009.

[21] A. Hennecke M. Trieber and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. In *Phys. Rev. E 62*.

[22] V. Naumov and TR. Gross. Connectivity-aware routing (car) in vehicular ad hoc networks. In *INFOCOM'07*, 2007.

[23] Y. Peng, Z. Abichar, and J. M. Chang. Roadside-Aided Routing (RAR) in Vehicular Networks. In *ICC '06*, 2006.

[24] Hua Qin and Wensheng Zhang. Charging scheduling with minimal waiting in a network of electric vehicles and charging stations. In *ACM VANET*, Las Vegas, US, September 2011.

[25] O. Tonguz, N. Wisitpongphan, F. Bai, P. Mudalige, and V. Sadekar. Broadcasting in VANET. In *INFOCOM '08 proceedings*, 2008.

[26] M. Trieber and D. Helbing. Realistische microsimulation von strassenverkehr mit einem einfachen modell. 2002.

[27] R. Baumann V. Naumov and T. Gross. An evaluation of inter-vehicle ad hoc networks based on realistic vehicular traces. In *MobiHoc06*, Florence, Italy, May 2006.

[28] F. Xie W. Wang and M. Chatterjee. An integrated study on mobility models and scalable routing protocols in vanets. In *INFOCOM'08*, 2008.

[29] S.Y. Wang and C.L. Chou. Nctuns simulator for wireless vehicular ad hoc network research. In *Ad Hoc Networks: New Research*, 2008.

[30] Hellbruck H. Wewetzer-C. Wegener, A. and A. Lubke. Vanet simulation environment with feedback loop and its application to traffic light assistance. In *GLOBECOM Workshops*, 2008.

[31] M. Weigle and S. Olariu. Intelligent Highway Infrastructure for Planned Evacuations. In *Performance, Computing, and Communications Conference*, 2007.

[32] N. Wisitpongphan, F. Bai, P. Mudalige, V. Sadekar, and O. Tonguz. Routing in Sparse Vehicular Ad Hoc Wireless Networks. *IEEE journal on Selected Areas in Communications*, 2007.

[33] K. Xing, M. Ding, X. Cheng, and S. Rotenstreich. Safety Warning Based on Highway Sensor Networks. In *Wireless Communications and Networking Conference*, 2005.

[34] Y. Zhang, J. Zhao, and G. Cao. On Scheduling Vehicle-Roadside Data Access. In *VANET'07*, 2007.

[35] J. Zhao, Y. Zhang, and G. Cao. Data Pouring and Buffering on the Road: A New Data Dissemination Paradigm for Vehicular Ad Hoc Networks. In *IEEE Transactions on Vehicular Technology*, 2007.