International Journal of Advance Engineering and Research Development (IJAERD)
Volume 1,Issue 6,June 2014, e-ISSN: 2348 - 4470 , print-ISSN:2348-6406

Implementation of TRACI and Induction loop in Vehicular Ad-hoc Network

Rahul N. Vaza[1],Amit B. Parmar[2], Trupti M. Kodinariya[3]

[1]*PG Student, AITS,Rajkot, rahulatmiya.rahul9@gmail.com*
[2]*PG Student,AITS,Rajkot, er.amitparmar@gmail.com*
[3]*Asst. Prof., AITS,Rajkot, tmkodinariya@aits.edu.in*

ABSTRACT—Vehicular Ad-Hoc Network (VANET) is surging in popularity, in which vehicles constitute the mobile nodes in the network. Due to the prohibitive cost of deploying and implementing such a system in real world, most research in VANET relies on simulations for evaluation. A key component for VANET simulations is a realistic vehicular mobility model that ensures that conclusions drawn from simulation experiments will carry through to real deployments. In this work, we introduce a tool MOVE that allows users to rapidly generate realistic mobility models for VANET simulations. MOVE is built on top of an open source micro-traffic simulator SUMO. The output of MOVE is a realistic mobility model and can be immediately used by popular network simulators such as ns-2 and qualnet.TraCI is the short term for "Traffic Control Interface". Giving the access to a running road traffic simulation, it allows to retrieve values of simulated objects and to manipulate their behavior "on-line" Vehicular Ad-Hoc Networks (VANETs) enable communication among vehicles as well as between vehicles and roadside infrastructures. Currently available software tools for VANET research still lack the ability to assess the usability of vehicular applications. In this article, we present Traffic Control Interface (TraCI) a technique for interlinking road traffic and network simulators. It permits us to control the behavior of vehicles during simulation runtime, and consequently to better understand the influence of VANET applications on traffic patterns. We describe the basic concept, design decisions and the message format of this open-source architecture. Additionally, we provide implementations for non-commercial traffic and network simulators namely SUMO and ns2, respectively. This coupling enables for the first time systematic evaluations of VANET applications in realistic settings.

Keywords--Vehicular Ad-Hoc Networks (VANETs), Network simulation, Node Mobility, Traffic Control Interface(Traci)

I. INTRODUCTION

A Vehicular Ad-Hoc Network or VANET is a technology that uses moving cars as nodes in a network to create a mobile network. VANET turns every participating car into a wireless router or node, allowing cars approximately 100 to 300 meters of each other to connect and, in turn, create a network with a wide range. As cars fall out of the signal range and drop out of the network, other cars can join in, connecting vehicles to one another so that a mobile Internet is created. It is estimated that the first systems that will integrate this technology are police and fire vehicles to communicate with each other for safety purposes.

Several tools are available for network simulations: e.g. ns2 [2], JiST/SWANS [3] or Shawn [4]. But none of them qualifies as a VANET simulator (as is the case with ns2 or GloMoSim for ad-hoc networks), mainly because none of them allows to evaluate how the networking applications influence mobility. They are mostly concerned about assessing the performance of routing, forwarding, MAC layer protocols, etc. of VANETs under realistic but unmodifiable mobility scenarios. The common approach to integrate mobility is to either rely on stand-alone road traffic simulators, such as SUMO [5], Vissim [6], MatSim [7], TRANSIMS [8], or to use collected mobility traces specific to some geographical region [9] [10]. Tools with such functionalities can be found in [11–13].

We address the problem of application-centric mobility- oriented evaluation of VANETs by proposing TraCI: Traffic Control Interface. Specifically, we suggest an open-source architecture that couples two simulators: a road traffic and a network simulator. In such a realistic simulation setup, mobility patterns are not pre-established as fixed trace files. Instead, the traffic and network simulators are connected in real time by TraCI thus enabling the control of mobility attributes of each simulated vehicle. Thus, the movement of each vehicle can be influenced by the VANET application running inside the network simulator. We claim that this approach will allow to fully evaluating VANET applications in realistic scenarios.

Our contribution is the new open-source architecture for coupling traffic and network simulators that aims toachieve two goals:
1) To create a generic API for controlling a traffic simulator so that a road traffic simulator can be coupled with a network simulator or any other simulator that needs to control the road traffic.
2) To set a ground for a framework for implementing VANET applications that can influence vehicle movement during runtime [13].

II. RELATED WORK

In recent years new simulation tools for VANETs have been developed; these tools can be classified into three different approaches. The first approach uses real maps to generate random waypoint mobility traces [14, 15] or more realistic traces [10]. The second approach uses integrated road traffic and network simulators, i.e., a Network simulator with embedded realistic mobility component, such as [16–18]. The third approach couples a commercial traffic simulator with a network simulator [19, 20].

Note that only the second and the third approach have the potential to evaluate VANETs at application-centric level under realistic scenarios. However, the major shortcoming of the existing tools is that the information exchanged between vehicles cannot influence their whereabouts. The only exception in the class of integrated simulators is a framework proposed by Wang et al. [18]. It allows for controlling vehicle movements by using an intelligent driving behavior module, coded in the agent logic that simulates the vehicle. Our solution is more generic, because it is based on a different architecture, which allows us to use various road traffic as well as network simulators. In the class of the coupled traffic and network simulators there are two exceptions, [19] [20]. They achieve necessary real-time coupling between two simulators, VISSIM or CARISMA (for road traffic) and ns2 (for communication). But it remains unclear what features are supported to control the traffic simulation. Furthermore, both VISSIM and CARISMA are commercial products that are not publicly available. Our open-source system architecture allows researchers to couple non-licensed, as well as licensed, tools that are widely used in both the ITS and VANET communities.

III. ROAD TOPOLOGY GENERATION

Step 1 Create nodes: Each node has to be assigned a unique ID. User also has the option to traffic lights for a node. This option can be used when nodes can be thought of as junctions in a system. The X and Y co-ordinates of the system has to be defined for mapping the node in a real world environment. Save this file as <filename>.nod.xml.

Step 2 Crete roads: Each node defined must be connected through a edge called as a Road. Hence From node and To node has to be defined. The type of the road eg. Fast lane or Express can be defined as per user need. No. lanes are the number of lanes that exist in each edge. The maximum speed of all the vehicles in the edge can be defined. The priority and the length of the edge can also be defined. Save this file as <filename>.edg.xml.

Step 3 Configuration: This is used map the node file and the edge file. The output file name should be <filename>.net.xml. Save this configuration map as <filename>.netc.cfg

Step 4 Create Map: The system that is just configured must be cross checked for inconsistancies. The Create map does this facility. The configuration file file: <file>.netc.cfg must be ocated for the same. Click on the OK button to check for inconsistencies.

Step 5 Flow (optional): Flow provides the option to define the number of vehicles for each of the edges defined in the <filename>.edg.xml file. Save the file as <name>.flow.xml (ex_FLOW.flow.xml). You can also open and edit an existing .flow.xml file.

Step 6 TURN (Optional): Select "Turn" from MOVE main menu. This is when you to specify a turning ratio for each junction. You must also create flow definitions as previously explained for the vehicle movements. In the editor, you must group percentages together so they sum up as 1.0(100%).

Step 7 Create Vehicle: Select "Create Vehicle" from MOVE main menu. This will simply create a number of vehicles at the start of simulation. For our example, we use junction turning ratios function. Select your previously created map file (e.g. <filename>.net.xml). Specify your output file location and name it as <name>.rou.xml (<filename>.rou.xml). Set the beginning and end time of simulation. Finally click OK. The rou.xml file will be automatically generated.

Step 8 Simulation setup: After the map and movement is complete, you will need to specify the configurations of the simulation. Select "Configuration" at the bottom on MOVE main menu. Specify the <filename>.net.xml and <filename>.rou.xml location and specify the beginning and end time of simulation. If you want to create the trace file, do not forget to check the checkbox and specify your trace output name e.g. <filename>.move.trace. Save this file as <name>.sumo.cfg.

Step 9 Visualization: Select "Visualization" to see the actual movements on vehicle. Select the <filename>.sumo.cfg saved in the above Configuration option. For the example chosen the following is the simulated result. One can notice there is a Signal that is designed in the center of the map. This takes care of any congestion that might occur due to heavy traffic load simulated. Delay (ms) can be increased to make sure the vehicles being simulated are working as per design.

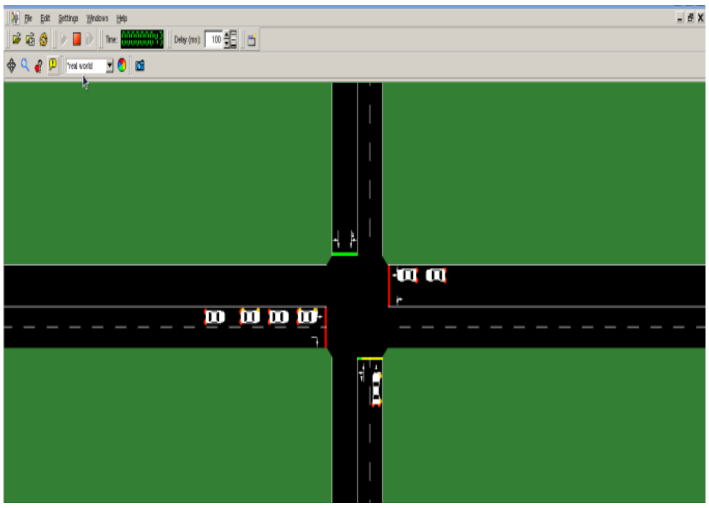

Fig.1. Road Topology

IV. TRACI INTERFACE

SUMO does not support algorithm running in it. In order to run our algorithm in SUMO, we needed to create a way to control the vehicles in SUMO during a simulation. This was accomplished via TraCI, an extension that provides a channel for communication with SUMO. With TraCI, SUMO acts as a server and listens for commands through a port. The library of commands in TraCI is extensive and includes control of each vehicle, traffic light, road, and almost every other variable in the simulation. When SUMO is called with the option to use TraCI, SUMO starts up, loads the scenario, and then waits for a command. Variables can be changed and then a command can be sent with how many seconds to run the simulation for before stopping and waiting for another command. For the simulations in this research, the actual connection is made through a python script, called Simulation Controller.

TraCI is the short term for "**Tra**ffic **C**ontrol **I**nterface". Giving the access to a running road traffic simulation, it allows to retrieve values of simulated objects and to manipulate their behavior "on-line".
TraCI uses a TCP based client/server architecture to provide access to SUMO. Thereby, SUMO acts as server that is started with additional command-line options: **--remote-port** ***<INT>*** where ***<INT>*** is the port SUMO will listen on for incoming connections. When started with the **--remote-port** ***<INT>*** option, SUMO only prepares the simulation and waits for an external application that takes over the control. Please note, that the **--end** ***<TIME>*** option is ignored when SUMO runs as a TraCI server, SUMO runs until the client demands a simulation end.

V. PROTOCOL DETAILS

A TCP connection between the mobility generator and network simulator is used for the exchange of data. Both the network and the road traffic simulator use TraCI messages to bundle, respectively, a set of commands or responses.The TraCI message, as depicted in consists of a small header that contains the overall message length including the message header, followed by a variable number of commands. Each command starts with its length and identifier, coded as an unsigned byte sequence. By definition, commands with identifiers are commands from the network simulator to the mobility generator. The respective responses have an identifier.

These identifiers are given in the header of each command's section. The content of each command depends on the particular command type.

The commands and corresponding responses can be split into three functional groups. The first group controls the simulation run:

1)Simulation Step is called periodically by the network simulator to let the traffic simulator perform the simulation up to the next time period. In addition to a Status response, a Move Node response for each equipped vehicle is expected as the answer.

2) **Status** is sent as a reply to every request command. It contains a status flag and a descriptive text.

3)Move Node contains movement information for one vehicle. It serves as a response to the Simulation Step command to transfer the mobility into the network simulator.

Commands from the second group are used to affect the behaviour of a single vehicle.

1) **Set Maximum Speed** limits vehicle's speed or removes such a limit (change speed primitive).
2) **Stop Node** causes a vehicle to stop and wait for a period of time at a certain position as soon as the vehicle gets there (stop primitive).
3) **Change Lane** fixes a vehicle to a specific lane for a certain amount of time.
4) **Change Route** causes the traffic simulator to reroute an individual vehicle by setting a particular travel time for a road segment.
5) **Change Target** changes the destination of a vehicle.

The environmental commands that give access to the simulation scenario form the third group. The simulation scenario is maintained mainly by the traffic simulator, as it holds the road map, points of interest, building footprints, traffic lights, public transportation and so on. Depending on the network simulation's focus, a subset of this information is required within the network simulation as well. We currently support the following set of environmental commands.

1) **Scenario** is used for getting scenario parameters, mainly at simulation setup time. These are x- and y- dimensions as well as the expected number of network nodes.
2) **Position Conversion** converts between different types of positions, e.g., to map a Cartesian coordinate to an appropriate position on the road network.
3) **Driving distance** calculates the path and estimated time to get from one position to another based on the under-lying road network.

Due to the wide range of these commands, we will wait for users' real needs and deliver the lacking features whenrequired.

We describe the single commands and responses of the first two groups in the following subsections, taking their semantics and syntax into account. Therefore, the used data types are introduced beforehand.

A. Data Types

To save the computation time, our protocol does not use structural information within its messages as XML does. In-stead, all messages are composed of a stream of elementary data types. These elementary data types are byte, integer, float, double and string.

All numeric data types are interpreted as signed, only for the data type byte exists an unsigned counterpart.

In addition, we introduce the composed data type position that is based on the aforementioned elementary data types. We use two different types of position representations: Cartesian 3D positions and positions on a road map. The former format is necessary, since in the vast majority of the network simulators the Cartesian coordinates are used to represent node's position. The latter format can be used for example at the application layer (implemented also in the network simulator), where information about map location can be required to perform application-specific actions. To specify which representation is used, aubyte identifier is put in front of each position (later on referred to as Position Type).

*1) **3D Position** In most cases, network simulators work with 3D coordinates to describe node positions.*

Ubyte								*float*	*float*	*float*
0	0	0	0	0	0	1	1	X	Y	Z

The ubyte identifier for 3D positions is 0x03. It is followed by three float variables describing the Cartesian X, Y and Z coordinates.

*2) **Road Map Position** To relieve the network simulator from converting Cartesian coordinates to positions on a road map, vehicles' positions can be reported directly as road map positions.*

ubyte								*String*	*float*	*ubyte*
0	0	0	0	0	1	0	0	Roadid Position	Laneid	

Road map positions have an identifier of 0x04.RoadId identifies an edge, i.e., a road segment on the road map graph; Pos describes the node's position in longitudinal direction in the range of (0; Length RoadId). LaneId contains the driving lane on the road segment. Those are numbered sequentially starting with zero from the rightmost lane. Note that lanes in opposite direction are identified by a different RoadId. In future work, more representations may be added, e.g. NMEA-0183 standardized GPS-positions or geographic coordinates using latitude and longitude.

B. Command Simulation Step (Id0x01)

double	*ubyte*
Target time	Position type

This command is sent repetitively by the network simulator at each simulation step (e.g., every second) to retrieve actual node positions and to make sure that the simulation times are synchronized between both simulators. It triggers the traffic simulator to simulate the next simulation step up to TargetTime that is the actual simulation time of the network simulator plus one simulation step. After simulating, the traffic simulator replies with a Status response and a collection of Move Node responses – one for each active node. The returned node positions are either 3D or road map representations according to the requested PositionType. All these responses are bundled as one TraCI message.

C. Response Status

ubyte	*String*
Result	Description

This response acknowledges each command; it includes a Result and a Description. In contrast to all other commands and responses, the identifier of the Status response is not fixed, but refers to the identifier of the respective command that is acknowledged. The Result value is set to 0x0 in case of success or to 0xFF if the requested command failed. If the requested command is not implemented in the traffic simulator, a status of 0x01 is returned. Additionally, a Description text must be added.

It is up to the network simulator to decide if the simulation needs to be stopped after receiving an error or not implemented status.

D. Response Move Node(Id0x80)

integer	*double*	*position*
NodeidTargetTime	Position	

The network simulator receives Move Node responses as replies to the command Simulation Step. Each active, i.e, moving vehicle, which is identified by its nodeId, results in one Move Node response. These must be converted into linear movements by the network simulator to ensure that each node reaches its Position at Target Time. The representation of the Position is interpreted according to the requested Position Type of the Simulation Step command.

E. Command Set Maximum Speed (Id0x11)

integer	*float*
NodeidMaxspeed	

This command causes the vehicle identified by NodeId to limit its speed to an individual maximum of MaxSpeed. The traffic simulator is responsible for slowing down the vehicle in a proper way according to its mobility model. To remove an individual speed of a vehicle, this command is called with a negative value for MaxSpeed

F. CommandStop Node(Id0x12)

integer	*position*	*float*	*double*
NodeidStopposition		Radius	WaitTime

Using the Set Maximum Speed command, a vehicle can be stopped immediately by setting its speed to zero. To stop a vehicle sometime in the future at a predetermined position, the Stop Node command can be used. It allowsto set a position where a node stops for an amount of time, whenever it gets there.Regarding the mobility model, stopping a vehicle requires some time. The traffic simulator is responsible for following its models and for influencing a node in a proper way, so that it is able to stop at the desired position.The Stop Node command refers to the vehicle identified by the field NodeId. Its Stop Position can be given in any position representation. As most simulators are time discrete, it is typically not feasible to hit a position exactly. Thus, a circular stoppage area is defined by a Radius around the stop position. Once the node stops, it waits for a period of time (Wait Time), by setting its maximum speed to zero, before it goes ahead on its route.

G. Command Change Lane (Id0x13)

integer	*byte*	*float*
Nodeid	Lane	Time

Using the command Change Lane, a vehicle identified by NodeId can be forced to move to another Lane for an amount of Time. After the given Time, the constraint is removed. If the road segment changes while a vehicle drives on a fixed lane, it gets fixed to the corresponding lane on the succeeding road segment. If this does not exist, the constraint is removed, which allows the vehicle to use all lanes again.

H. Command Change Route (Id0x30)

integer	*string*	*double*
NodeidRoadidTravelTime		

The command Change Route allows a vehicle identified by NodeId to react to certain traffic situations by adapting its route. Therefore, an individual (estimated) Travel Time for an appointed road segment (RoadId) is set. Under the normal operation, an estimated travel time for all road segments is used according to their length and allowed speed. This accords to the mode of operation of today's route guidance systems. The travel times set by this command are only visible to the given vehicle and a new route is calculated before the simulation continues.By setting a negative travel time, a typical travel time for that road segment is restored. To mark a road as blocked, a near infinite travel time should be used.

I. Command Change Target (Id0x31)

integer	string
NodeidRoadid	

The Change Target command is used to change the destination of a vehicle (NodeId). It needs the road segment (RoadId) that is the new target. Before the traffic simulator continues its simulation, a route to the modified destination is calculated.

VI. CONCLUSION

We have presentedTraCI, the interface for controlling road traffic simulators via a flexible and extendable request-reply protocol. This kind of interlinking of two simulators is required, for example, to perform realistic evaluation of VANET applications as specified in the TraNS framework. We have derivedbasic control commands for the road traffic simulation andhave implemented interfaces for SUMO and ns2 - two open source simulators developed by different research communities.

We have evaluated the performance of TraCI and havedemonstrated that it is even faster to couple a network simulator with the road traffic simulator instead of writing tracefiles first and run the network simulation in the next step, evenif we do not need a control loop during execution.Using the TraCI interface one can even evaluate complexVANET scenarios, such as accidents (e.g. by stopping certain vehicles at a certain time instant) or simulating hazardousroad conditions (e.g., by modifying the speed of vehicleswhich have directly 'witnessed' such conditions).

REFERENCES

[1] IEEE Intelligent Transportation Systems Society. http://www.ieee.org/its.
[2] Rahul N. Vaza,Amit B. Parmar,Trupti M.Kodinariya,"Implementing Current Traffic Signal Control Scenario in VANET Using SUMO", International Journal of Advance Engineering and Research Development (IJAERD)ETCEE-2014 Issue, March 2014, e-ISSN: 2348 -4470 , print-ISSN:2348-6406
[3] Kevin Fall and Kannan Varadhan. The ns manual, 1989–2001.
[4] JiST/SWANS: Java in Simulation Time/Scalable Wireless Ad hoc Network Simulator. http://jist.ece.cornell.edu
[5] Sandor P. Fekete, Alexander Kr¨ oller, Stefan Fischer, and Dennis Pfisterer. Shawn: The fast, highly customizable sensor network simulator. InProc. of INSS '07, 2007.
[6] Daniel Krajzewicz, Michael Bonert, and Peter Wagner. The Open Source Traffic Simulation Package SUMO. In RoboCup 2006 Infrastructure Simulation Competition, Bremen, Germany, 2006.
[7] VISSIM: microscopic, behavior-based multi-purpose traffic simulation program. http://www.ptvamerica.com/vissim.html.
[8] MATsim: Multi-Agent Transport Simulation Toolkit.http://www.matsim.org.
[9] TRANSIMS: Transportation Analysis Simulation Sys-tem. http://www.ccs.lanl.gov/transims/index.shtml.
[10] Richard M. Fujimoto, Randall Guensler, Michael P. Hunter, Hao Wu, Mahesh Palekar, Jaesup Lee, and JoonhoKo. CRAWDAD data set gat-ech/vehicular (v. 2006-03-15), March 2006.http://crawdad.cs.dartmouth.edu/gatech/vehicular.
[11] Axel Wegener, Michał Pi´ orkowski, Maxim Raya,HorstHellbr¨ uck , Stefan Fischerand Jean-Pierre Hubaux, TraCI: An Interface for Coupling Road Traffic and Network Simulators. In 11th communications and networking simulation symposium Pages 155-163, ISBN:1-56555-318-7

[12] Valery Naumov, Rainer Baumann, and Thomas Gross, An Evaluation of Inter-Vehicle Ad Hoc Networks Based on Realistic Vehicular Traces. InProc. ofMobiHoc ’06, pages 108–119, 2006.
[13] Rapid Generation of Realistic Simulation for VANET, http://www.csie.ncku.edu.tw/klan/move/index.htm.
[14] TraceExporter: Exports SUMO dumps as ns2 traces, http://sumo.sourceforge.net/wiki/index.php/traceExporter
[15] TraNS: Joint Traffic and Network Simulator -Application-centric framework for evaluation of VANETs. http://trans.epfl.ch.
[16] Amit Kumar Saha and David B. Johnson. Modeling Mobility for Vehicular Ad Hoc Networks. InProc. Of VANET ’04 (Poster abstract), pages 91–92, 2004.
[17] D. R. Choffnes and F. E. Bustamante. An Integrated Mobility and Traffic Model for Vehicular Wireless Net-works. InProc. of VANET ’05, pages 69–78, 2005.
[18] J. H¨ arri, F. Filali, C. Bonnet, and Marco Fiore. Vanet-MobiSim: Generating Realistic Mobility Patterns for VANETs. InProc. of ANET ’06 (Poster abstract), pages 96–97, 2006.
[19] Rahul Mangharam, Daniel S. Weller, Daniel D. Stan-cil, RagunathanRajkumar, and Jayendra S. Parikh. GrooveSim: A Topography-Accurate Simulator for Ge-ographicRouting in Vehicular Networks. InProc. Of VANET ’05, pages 59–68, 2005.
[20] S.Y. Wang, C.L. Chou, Y.H. Chiu, Y.S. Tseng, M.S. Hsu, Y.W. Cheng, W.L. Liu, and T.W. Ho. NCTUns 4.0: An Integrated Simulation Platform for Vehicular Traffic, Communication, and Network Researches. InProc. Of WiVec’07, 2007.
[21] Stephan Eichler, BenediktOstermaier, ChritophSchroth, and TimoKosch. Simulation of Car-to-Car Messaging: Analyzing the Impact on Road Traffic. In Proc. of MASCOTS ’05, pages 507–510, September 2005.
[22] Christian Lochert, Murat Caliskan, Bj¨ ornScheuer-mann, Andreas Barthels, Alfonso Cervantes, and Martin Mauve. Multiple Simulator Interlinking Environment for IVC. InProc. of VANET ’05 (Poster abstract), pages 87–88, 2005.
[23] Roberto Baldessari, Bert B¨ odekker, Matthias Dee-gener, Andreas Festag, Walter Franz, C. Christo-pherKellum, imoKosch, Andras Kovacs, Massimil-ianoLenardi, Cornelius Menig, TimoPeichl, Matthias R¨ ockl, Dieter Seeberger, Markus Straßberger, HannesStratil, Hans-J¨ org V¨ ogel, Benjamin Weyl, and Wenhui Zhang. Car-2-Car Communication Consortium - Man-ifesto (Version 1.1), August 2007. http://www.car-2-car.org/index.php?id=570.
[24] Wiki for the Traffic Control Interface. http://sumo.sf.net/wiki/index.php/TraCI.
[25] Stefan Krauß. Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics. PhD thesis, Universit¨ at zuK¨oln, April 1998.
[26] Marc Torrent-Moreno. Inter-Vehicle Communications: Achieving Safety in a Distributed Wireless Environment: Challenges, Systems and Protocols. PhD thesis, Univer-sit¨ atsverlag Karlsruhe, 2007.
[27] Axel Wegener, Horst Hellbr¨ uck, Stefan Fischer, Chris-tiane Schmidt, and S´ andorFekete. AutoCast: An Adap-tive Data Dissemination Protocol for Traffic Informa-tion Systems. In Proc. of VTC ’07, Baltimore, USA, October 2007.