Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

NCTUns tool for wireless vehicular communication network researches

S.Y. Wang [*], C.L. Chou

Department of Computer Science, National Chiao Tung University, 1001 Ta-Hsueh Road, HsinChu 300, Taiwan

ARTICLE INFO

Article history:
Received 6 August 2008
Received in revised form 2 February 2009
Accepted 14 April 2009
Available online 3 May 2009

Keywords:
Intelligent transportation systems (ITS)
Network simulator
Traffic simulator

ABSTRACT

Several goals such as improving road safety and increasing transport efficiency are being pursued in intelligent transportation systems (ITS). Wireless vehicular communication is one technology to achieve these goals. Conducting vehicular experiments on the roads is an approach to studying the effectiveness of wireless vehicular communication. However, such an approach is costly, hard-to-control (repeat), dangerous, and infeasible when many vehicles and people are involved in the field trial. In contrast, the simulation approach does not have these problems. It is a very useful approach and complements the field trial approach. This paper presents NCTUns, an open source integrated simulation platform, for wireless vehicular communication network researches. This tool tightly integrates network and traffic simulations and provides a fast feedback loop between them. Therefore, a simulated vehicle can quickly change its driving behavior such as moving speed and direction when it receives a message from the wireless vehicular communication network. This capability is required by several novel ITS applications such as active collision avoidance systems. In this paper, we present the design, implementation, validation, and performance of this tool.

1. Introduction

Intelligent transportation systems (ITS) has become an attractive research field for many years. Among many technologies proposed for ITS, wireless vehicular communication, covering vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-person (V2P) communication, aims to increase road safety and transport efficiency and provide ubiquitous wireless connectivity to the Internet. With the assistances from these different means of communication, drivers and pedestrians can quickly obtain useful and/or emergent traffic information on the roads at a low cost. For this reason, wireless vehicular communication has become a very important technology for intelligent transportation systems.

The IEEE 802.11p [1] and IEEE 1609 draft standards [2–5] have been proposed as a networking technology for vehicular environments. ITS applications based on this new technology must be thoroughly tested and evaluated before being deployed on the roads. This means that many experiments under different parameters settings, configurations, scenarios, and conditions must be performed to verify the feasibility and effectiveness of an application and the used networking technology in the real-life environment. According to the results obtained from experiments, the designs of the application and the used networking technology might need to be revised many times before acceptable performances can be achieved.

A field trial of wireless vehicular communication usually involves a large number of vehicles and people (drivers and computer operators) for generating meaningful results. Conducting such field trials is very costly because many vehicles need to be rented (or purchased), many communication equipments need to be purchased, and many experimenters need to be employed for conducting the field trials. Sometimes, during a field trial with a specifically-designed high-speed scenario, the

[*] Corresponding author. Tel.: +886 3 5131550; fax: +886 3 5724176.
E-mail address: shieyuan@csie.nctu.edu.tw (S.Y. Wang).

1569-190X/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2009.04.008

experimenters may even have to face potential dangers such as collisions with vehicles or pedestrians. Besides, it is very difficult to accurately control and repeat a field trial on the roads, which is bad for debugging the problems and improving the performances of a new protocol or application. Given these problems, it is highly desirable to use software simulation to perform indoor function testing and performance evaluations prior to conducting field trials.

To study wireless vehicular communication networks, a simulator must be able to simulate both communication/network protocols and microscopic vehicular movements. The two requirements are not new and such capabilities are already provided by existing network simulators or traffic simulators, respectively. Regarding network simulators, they are usually used to test the functions and evaluate the performances of network protocols and applications under various network conditions. One can use them to test how his/her protocols (e.g., routing protocols, medium access control protocols, or transport protocols) and applications (e.g., HTTP, FTP, or VoIP) would perform under various network conditions. On the other hand, traffic simulators are usually used to simulate drivers' driving behavior (e.g., car following, lane changing, overtaking, etc.) on different kinds of road networks (e.g., freeways, urban areas, etc.). One usually uses them in the research areas of transportation engineering, such as transportation planning and traffic engineering.

Mostly, a network simulator is dedicated only to the studies of network protocols and applications and a traffic simulator is only dedicated to the studies of transportation engineering. To study advanced ITS applications, however, a simulation platform must be able to simulate both network and traffic simultaneously and provide a fast feedback loop between them. For example, some intelligent transportation systems aim to add information and communication technologies into transport infrastructures and vehicles to improve safety and reduce vehicle wear, transportation time, and fuel consumption. For these applications, the driving behavior of a vehicle may need to be changed after receiving a message from the wireless vehicular communication network. To study these applications, a simulation platform must tightly integrate both network and traffic simulations and provide a fast feedback loop between them.

In this paper, we present an integrated simulation platform, called NCTUns, for wireless vehicular communication networks research. NCTUns 1.0 [6] was originally developed as a network simulator with unique network simulation capabilities. Later on, NCTUns 5.0 [7] 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. With these capabilities, NCTUns now is a useful simulation platform for wireless vehicular communication network researches.

The rest of the paper is organized as follows. In Section 2, we survey related work that combines the capabilities from both a network simulator and a traffic simulator. In Section 3, we present the design and implementation of NCTUns, including platform architecture, supported road types, vehicle movement controls, application program interfaces, and network protocol simulations. In Section 4, we validate the mobility control of vehicles on NCTUns with mathematical models based on Newton's laws of motion. In Section 5, the scalability performances of NCTUns are evaluated. In Section 6, we present three usage examples of NCTUns. In Section 7, we discuss ongoing work for the next release of NCTUns. Finally, we conclude the paper in Section 8.

2. Related work

As stated above, a simulator suitable for conducting wireless vehicular communication network researches should have the capabilities supported by both a traffic simulator and a network simulator. An intuitive method is to write a middleware to loosely couple a traffic simulator with a network simulator to provide the required capabilities. In this paper, such a method is called the "federated approach." This approach has the advantage that one need not spend time and effort on developing both a new network simulator and a new traffic simulator. However, because the two simulators are loosely coupled and may need to be run on different machines over a network due to different operating system platform requirements, the simulation performance of this approach may be quite low due to excessive message transfers (e.g., for fine-grain simulation time synchronization) between the two simulators. Another problem with this approach is that if any of the two simulators is a commercial software without opening its source code for a researcher to modify, it will be quite difficult (if not totally impossible) to provide a fast feedback loop between their program code executions.

On the other hand, another method, called the "integrated approach," tries to add network simulation functions into an existing traffic simulator, or add microscopic vehicle movement simulation functions into an existing network simulator, or to develop a new simulator from scratch with all of these capabilities. This approach has the advantages that (1) one need not develop both a network simulator and a traffic simulator from scratch, saving much time and effort; (2) the program code of the traffic and network simulation subsystems are tightly integrated as a single program. With this approach, the feedback loop between the two subsystems exists naturally and is very efficient in providing feedback to the other subsystem.

Regarding the federated approach, there are several existing network simulators (e.g., [8–11]) and traffic simulators (e.g., [12–16]) from which one can choose to combine. Some of them are commercial products while some of them are free and/or open source software. In Table 1, we list four existing federated traffic/network simulator combinations: CORSIM/QualNet [17], VISSIM/ns2 [18], CARISMA/ns2 [19], and SUMO/ns2 (TraNS) [20]. Among the used network and traffic simulators, only SUMO and ns2 are free open source software. This means that only the SUMO/ns2 (TraNS) combination is totally open source and allows researchers to freely modify for their researches.

Table 1
Microscopic mobility and other features of traffic/network simulators.

Traffic/network simulator	Approach	Microscopic mobility		Radio obstacles	Visualization tool
		Car following	Lane changing		
CORSIM/QualNet [17]	Federated	Yes	Yes	No	Yes
VISSIM/ns2 [18]	Federated	Yes	Yes	No	Yes
CARISMA/ns2 [19]	Federated	Yes	No	Yes	Yes
SUMO/ns2 (TraNS) [20]	Federated	Yes	No	No	Yes
NCTUns	Integrated	Yes	Yes	Yes	Yes
SWANS [21]	Integrated	Yes	Yes	No	No
MoVES [22]	Integrated	Yes	No	No	Yes
AutoMesh [23]	Integrated	Yes	No	Yes	Yes
VANET [24]	Integrated	Yes	Yes	No	Yes

The conceptual architecture of a federated traffic/network simulator is shown in Fig. 1. This approach employs a middleware to interconnect a traffic simulator and a network simulator. The middleware provides bidirectional links, usually realized by TCP connections, between the two simulators. Because the traffic simulator is responsible for simulating road networks and vehicle mobility, the latest position of every vehicle needs to be sent from the traffic simulator to the network simulator during simulation. The position update is performed periodically or by the request from the network simulator. Conversely, in the network simulator, if a vehicle receives a message from the wireless vehicular communication network and this message makes it decide to change its driving behavior (e.g., to change the current route to avoid congested or dangerous area or to stop immediately to avoid a forthcoming collision, etc.), the network simulator needs to send a request to the traffic simulator asking for such a change. Issuing such a request usually is the result of analyzing the information contained in the received message by the application program running on the vehicle.

The architectural advantage of the federated approach is that in theory it can combine any traffic simulator with any network simulator. However, in practice it is difficult to do so (if not totally impossible). Two independent simulators may have different designs, implementations, and definitions. For example, the used coordinate systems and the representations of continuous vehicular movement may be different. A transformation mechanism for accommodating these differences must exist in the middleware at the cost of performance degradation during simulation. Besides, a commercial software normally does not release its source code but only exports some pre-defined application program interfaces (API) for external software programs to use. Since the development of new ITS applications advances so quickly, the pre-defined API's may not meet the demands of new ITS applications.

Regarding the integrated approach, Table 1 lists five integrated traffic/network simulators: NCTUns, SWANS [21], MoVES [22], AutoMesh [23], and VANET [24]. Three different methods are possible for constructing a simulator using the integrated approach and they are shown in Fig. 2. In Fig. 2a, communication model and network protocol simulation capabilities are added into an existing traffic simulator. Because there are various communication models and network protocols used in the real life and they can be very complicated (e.g., IEEE 802.16(e) WiMAX), this method requires a huge amount of time

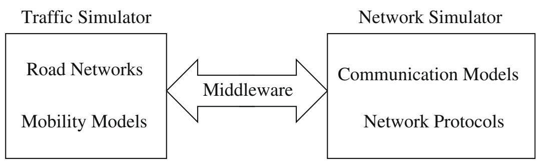

Fig. 1. The conceptual architecture of a federated traffic/network simulator.

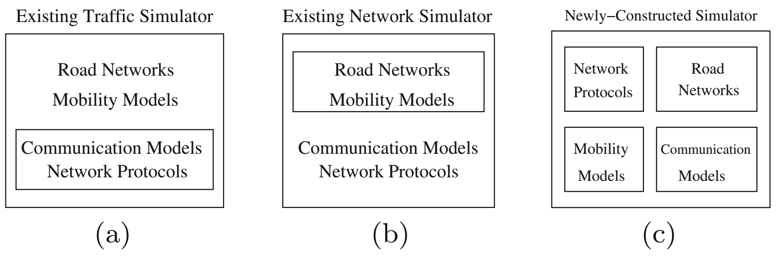

Fig. 2. Three different methods for constructing an integrated traffic/network simulator: (a) adding communication models and network protocols into a traffic simulator; (b) adding road networks and mobility models into a network simulator (e.g., NCTUns and SWANS); and (c) develop all required components to form a new simulator (e.g., MoVES, AutoMesh, and VANET).

and effort. Therefore, our survey found no integrated simulator adopting this method. In contrast, in Fig. 2b, an existing network simulator is extended to include the capabilities of road network simulation and vehicle mobility models. This method is more feasible and incurs less cost because a network simulator already has the capability to simulate mobile node movement (e.g., the commonly used random waypoint mobility model). Based on this capability, it just needs to simulate road networks and support more realistic mobile node mobility models generated from a microscopic traffic simulator. Relatively, this task is easier to accomplish than simulating various complicated communication models and network protocols. Therefore, NCTUns and SWANS adopt this method to take advantage of their existing network simulation capabilities. Yet another method is to develop all required components from scratch to construct a new simulator, which is represented by Fig. 2c. MoVES, AutoMesh, and VANET adopt this method. Conceivably, this method will require a very huge amount of time and effort to develop a complete simulator from scratch. Another problem with this method is that since the newly developed simulation code may have not gone through extensive uses and testing, their simulation results may need more validations before being fully trusted.

NCTUns, unlike SWANS, which is a Java based simulator, is a C++ based simulator running on Linux Fedora platform. On top of its supports for various communication models and network protocols, road networks and many realistic microscopic vehicle mobility models have been implemented into NCTUns. This platform will be presented in detail in Section 3.

Table 1 also compares the microscopic mobility and other important features of each simulator. These features include car following, lane changing, radio obstacles, and the visualization tool. The car-following capability is supported by all simulators because without it vehicles will move randomly instead of following their previous vehicles on paved roads. This will generate inconvincible simulation results as the resulting vehicular moving paths do not reflect the paths in the real life.

To have the lane-changing capability, a platform must support multi-lane roads and more intelligent vehicular driving behaviors in the used vehicle mobility models. Table 1 shows that some platforms do not have this capability. Lacking this capability will limit the uses of the simulator to only a few scenarios. For example, without this capability, if a vehicle breaks down on a multi-lane road, the vehicles behind it can only stop because no lane-changing operation can be used. However, in the real life, they can change lanes to avoid this broken vehicle. In addition, lacking the lane-changing capability means that no overtaking will occur during simulation. This means that the topology of the vehicular ad hoc network (VANET) formed in such a case will not change much during simulation. Since this does not reflect the situations in the real life, the simulation results on such a simplified and unrealistic VANET may be misleading.

Regarding radio obstacles, they can be used to totally block the transmission of wireless signal or reduce the power of wireless signal. Radio obstacles can significantly affect the performance of network protocols on wireless networks. In some research topics, radio obstacles are required and used to build a specific wireless network environment. For a network simulator, supporting radio obstacles is an important capability. Table 1 shows that only NCTUns, CARISMA/ns2, and AutoMesh have this capability.

The last capability compared is the support of a visualization tool. It is a GUI program displaying the road networks and vehicular movement during simulation or after a simulation is finished. This tool provides visual observations of a simulated network. Moreover, some visualization tools also provide the capabilities of interacting with the simulated network during simulation, such as dynamically changing some system parameters of the simulated network or dynamically controlling vehicular mobility. A visualization tool is very useful in specifying, controlling, and observing the simulated network. Table 1 shows that every platform has a visualization tool except SWANS.

3. NCTUns integrated simulation platform

In this section, we present the major components of NCTUns and their relationships. The operational procedures in NCTUns are also presented to explain how NCTUns simulates wireless vehicular communication networks. Fig. 3 depicts the architecture of NCTUns. NCTUns includes GUI (graphical user interface), SE (simulation engine), CA (car agent), and SA (signal agent), which will be explained below.

3.1. GUI (graphical user interface)

The GUI provides five major functions to help users easily generate the configuration files required by a simulation case. These files will be read by other components at the beginning of a simulation. We leave the explanations of them to later subsections. In this subsection, we present these functions and their corresponding output files.

3.1.1. Road network construction

The GUI provides users with an environment in which they can easily construct their desired road network. For example, road construction and connection can be completed in just a few mouse operations. The GUI supports different types of roads, including single-lane roads, multi-lane roads, crossroads, T-shape roads, and lane-merging roads. In Fig. 4, a screenshot of the GUI is shown with different types of roads. A T-shape road is formed from a crossroad with one of the four branches being closed. Four traffic lights are automatically placed at the four corners of a crossroad. In addition, at the place where the central divider of each branch joins the crossroad, one roadside unit equipped with an 802.11(a/b/g)-based radio

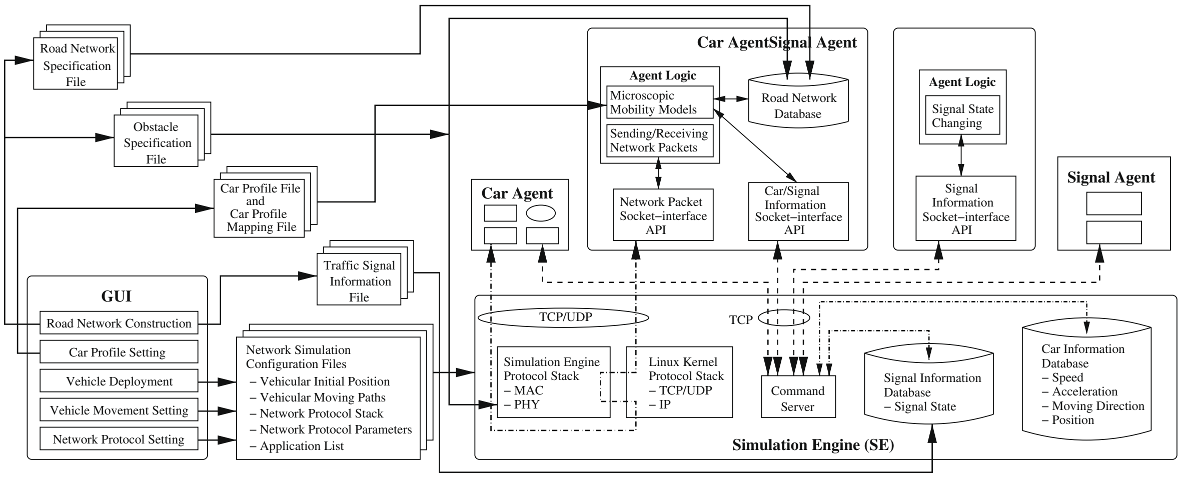
Road Network Specification File
Obstacle Specification File
Car Profile File and Car Profile Mapping File
Traffic Signal Information File
GUI
Road Network Construction
Car Profile Setting
Vehicle Deployment
Vehicle Movement Setting
Network Protocol Setting
Network Simulation Configuration Files
– Vehicular Initial Position
– Vehicular Moving Paths
– Network Protocol Stack
– Network Protocol Parameters
– Application List
Car AgentSignal Agent
Agent Logic
Microscopic Mobility Models
Sending/Receiving Network Packets
Road Network Database
Network Packet Socket–interface API
Car/Signal Information Socket–interface API
Car Agent
Agent Logic
Signal State Changing
Signal Information Socket–interface API
Signal Agent
TCP/UDP
TCP
Simulation Engine Protocol Stack
– MAC
– PHY
Linux Kernel Protocol Stack
– TCP/UDP
– IP
Command Server
Signal Information Database
– Signal State
Car Information Database
– Speed
– Acceleration
– Moving Direction
– Position
Simulation Engine (SE)

Fig. 3. The architecture of NCTUns.

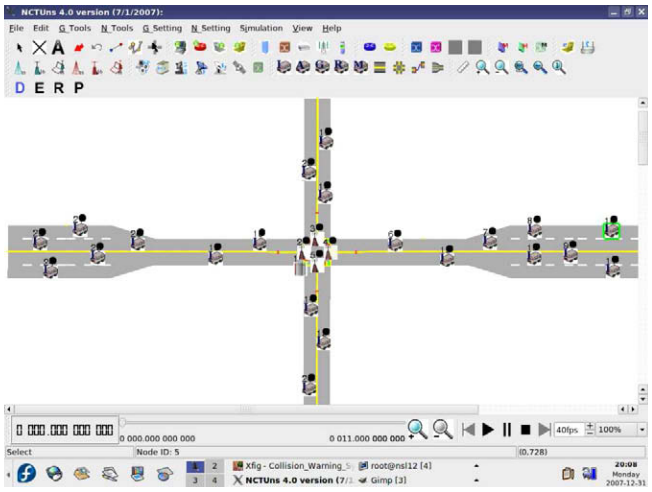
NCTUns 4.0 version (7/1/2007):
File Edit G_Tools N_Tools G_Setting N_Setting Simulation View Help
D E R P
0 000.000 000 000
0 011.000 000 000
40fps
100%
Select
Node ID: 5
(0,728)

Fig. 4. A screenshot of the GUI of NCTUns.

is automatically placed. This arrangement is useful for studying advanced ITS applications such as a collision warning system that needs roadside units to be deployed at the crossroad.

After a desired road network is constructed, the GUI will automatically generate two configuration files before a simulation starts. One of them is the traffic signal information file and the other is the road network specification file. A traffic signal can be one of several types such as a traffic light, a stop sign, or a speed-limit sign. The traffic signal information file contains the coordinates of all traffic signals, the facing direction of each traffic signal, the type of each traffic signal, and some type-specific data such as the initial (color) state for a traffic light and the group identifier assigned to the four traffic lights surrounding a crossroad. For the set of four traffic lights surrounding a crossroad, a signal-agent application will be run up to control the state changes of these traffic lights during simulation. The information about when and on which nodes to launch these applications is exported into the application-launching configuration file.

As for the road network specification file, it contains the information of all road blocks, including the four coordinates of a road block's four corners, the moving direction for vehicles on a road block, which road blocks are linked to form a lane, and which lanes are joined together to form a multi-lane road. Take the road network shown in Fig. 5 as an example. In this square road network, road block 1, 3, 5, and 7 are linked to form a lane, road block 2, 4, 6, and 8 are linked to form another lane, and these two lanes are joined together to form a square road.

In addition to manually construct a road network, NCTUns can import a real-world map to automatically construct a road network. This capability can save much time required to manually construct a large road network. Besides, since the constructed road network corresponds to a real road network, the moving pattern of vehicles on these constructed roads will be more realistic. Fig. 6 shows that NCTUns imports the map of the southern part of Taipei to construct its corresponding road network.

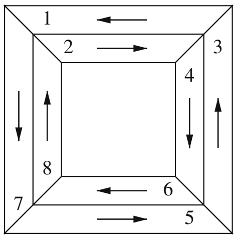

Fig. 5. An example of road network representation.

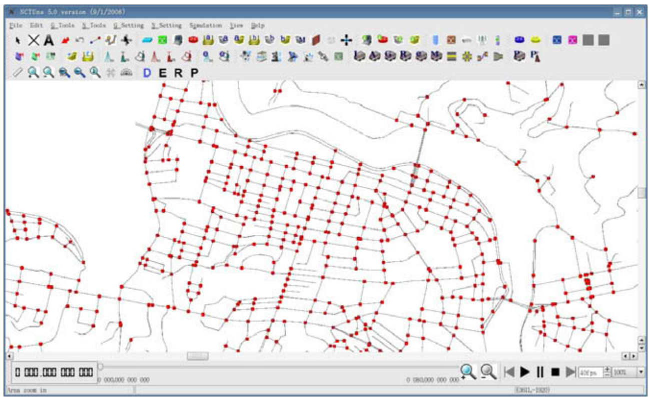

Fig. 6. NCTUns can import a real-world map (Taipei) to automatically construct the road network.

On the constructed roads, the GUI supports the view/radio obstacle, which is a rectangular object. A user can arbitrarily stretch or shrink this rectangular object to change its length. A building with an arbitrary shape (e.g., a triangular building) can be built by using multiple obstacles to form the desired shape. An obstacle can block a vehicle driver's view, and/or totally block a wireless signal or just reduce the power of a wireless signal. These capabilities are very useful for making a testing environment for advanced ITS researches. The information of all obstacles, such as the coordinates and the specified signal power reduction, is exported by the GUI into the obstacle specification file.

3.1.2. Car profile setting

To support different moving characteristics of vehicles, the GUI allows users to specify different car profiles. At most five car profiles are supported in the current release and each of them is stored in a separate car profile file. In each car profile, a vehicle's maximum (allowable) speed, maximum acceleration, and maximum deceleration are specified. A user can specify the percentage of vehicles that will use a particular car profile during simulation. The GUI will automatically map each car to a specific car profile to match the percentage distribution. This mapping information is exported into the car profile mapping file, which will be read by CA's at the beginning of a simulation. The CA is responsible for controlling the moving behavior of its associated vehicle during simulation.

3.1.3. Vehicle deployment

The GUI allows a user to automatically deploy a specified number of vehicles on the road network at a specified average separation distance between two neighboring vehicles on the same lane. A user can also select which kinds of wireless radios should be equipped with a vehicle. The initial position of each vehicle is exported by the GUI into the node-movement-scenario configuration file.

3.1.4. Vehicle movement setting

NCTUns supports two approaches for controlling the movement of vehicles. The first one is the pre-specified approach and the second one is the autopilot approach. In the pre-specified approach, a user employs the GUI to specify the moving path and speed of each vehicle before a simulation starts. The GUI will export these information into the node-movement-scenario configuration file, which will be read by the SE at the beginning of a simulation. During simulation, each vehicle will move along its pre-specified moving path at the pre-specified moving speed(s) on a road network. The moving paths and speeds of vehicles can also be imported from a file, which may be generated by any program such as a traffic simulator, to provide extensibility.

In contrast, in the autopilot approach, a user need not specify each vehicle's exact moving path/speed but instead only need to specify the car profile used for each vehicle. During simulation, a CA will automatically control the moving behavior of its associated vehicle based on its assigned car profile. According to the dynamic control issued from the CA, each vehicle will dynamically change its moving direction and speed during simulation. In the autopilot approach, the GUI will automatically assign a car-agent application to each vehicle to control its movement during simulation. This information is exported by the GUI into the application-launching configuration file.

3.1.5. Network protocol setting

In NCTUns, the simulation of different kinds of wireless radios is realized by simulating different network protocol stacks. A vehicle equipped with a wireless radio is associated with that radio's corresponding protocol stack. Each layer of a protocol stack is implemented as a protocol module in NCTUns. A protocol stack can be viewed as a series of protocol modules linked together. The GUI allows users to easily select/replace protocol modules, such as mobile ad hoc routing protocol modules, and set the parameter values associated with each module. The information about the used protocol stacks and module parameter values is exported by the GUI into the protocol-module-specification configuration file.

In addition to the aforementioned functions, the GUI can play animations of packet transmission and vehicle movement, either during simulation or after simulation. This visual display of simulation results greatly helps a user check the correctness of their network protocol designs and vehicle movement behavior.

3.2. SE (simulation engine)

When a simulation starts, the SE reads in the signal information file, the node-movement-scenario configuration file, the protocol-module-specification configuration file, the obstacle specification file, and the application-launching configuration file. The SE also needs to read in other configuration files. However, because they are not directly related to the settings for a wireless vehicular communication network simulation, they are not explained here.

The signal information file is read by the SE for building the signal information database. As stated before, the many attributes of traffic lights are recorded in this database. The node-movement-scenario configuration file is read by the SE for setting each vehicle's initial position. In the case of using the pre-specified vehicular movement control, the file is also used to schedule the moving path/speed changing events, which will be triggered during simulation. During simulation, the positions of all vehicles will be updated and maintained in the car information database. The protocol-module-specification configuration file is read by the SE for building each vehicle's network protocol stack by linking a series of protocol modules, and for initializing the parameters associated with each protocol module. In addition, the SE reads the obstacle specification file to simulate radio obstacles, which can block or reduce wireless signal power during simulation. When reading the application-launching configuration file, the SE schedules the application-launching events, which will be triggered during simulation. When an application-launching event is triggered, the SE forks a CA process or an SA process, depending on which application is specified. Like other applications forked by the SE, a CA process or an SA process can be forked at any time during simulation and then be killed at any time before the end of a simulation. The GUI allows a user to freely specify the start/end time of an application. When a CA or an SA process is created, its corresponding node identifier used in the SE is recorded into a kernel data structure through a NCTUns-specific system call. This information can be retrieved from the kernel through another NCTUns-specific system call during simulation.

The SE sets up a TCP-based command server (which is a function periodically invoked in the SE process) to receive the commands issued from a CA or an SA. According to the type of a command, the command server may store/retrieve data into/from the signal information database or the car information database. The operations between the command server and the CA or SA will be described later.

The network protocol stacks simulated in NCTUns include the Linux kernel protocol stack, including TCP/IP and UDP/IP, and the user-level SE protocol stack, including the MAC- and PHY-layer protocols. Fig. 3 shows that different CA's exchange Internet packets with each other over TCP or UDP connections. These TCP/UDP connections are set up by the CA's using the standard POSIX socket-interface API's. To give a more detailed description about how an Internet packet passes through the simulated protocol stack, we use the example shown in Fig. 7 for illustration. In the example, two vehicles move on the roads and they exchange Internet packets with each other. To simulate this case, two CA's are run up to control these two vehicles (one for each vehicle) and the SE is run up to simulate the MAC and PHY layers of the protocol stack. Suppose that the left CA sends a packet to the right CA, the detailed packet delivery process is described below.

(i) The agent logic of the left CA uses the standard POSIX socket-interface API's (e.g., sendto( ), write( ), etc.) to write a segment of data into the socket send buffer in the kernel.
(ii) The data segment will first reach the TCP/UDP layer (which is defined as the transport layer in the OSI model). After being encapsulated with a TCP/UDP header, this TCP/UDP packet is then passed to the IP layer (which is defined as the network layer).
(iii) The TCP/UDP packet will be encapsulated again with a IP header and then be written into a tunnel interface.
(iv) Later on, the user-level SE will retrieve the IP packet from the tunnel interface.
(v) The media access control (MAC) protocol (which is defined as the datalink layer in the OSI model) and the physical (PHY) protocol (which is defined as the physical layer) are simulated in the SE. The fetched IP packet will be encapsulated again with a MAC header and then sent from the sending PHY to the receiving PHY under the control of the MAC protocol.

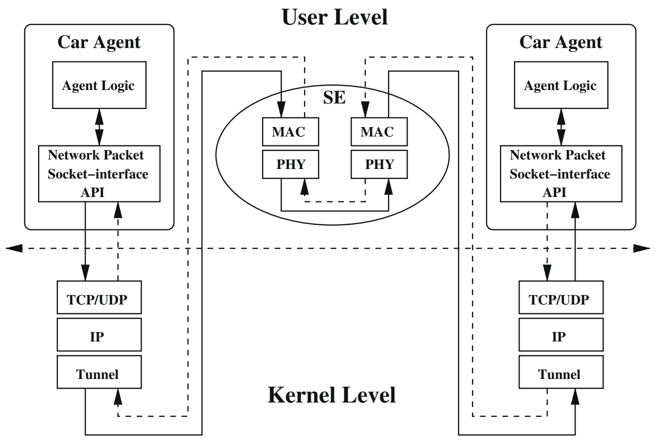

Fig. 7. The architecture of network protocol simulation.

(vi) The MAC header of the MAC packet will be stripped off when the packet arrives at the receiving MAC. The SE then writes the packet into another tunnel interface in the kernel, which is associated with the right vehicle.
(vii) The kernel then delivers the IP packet from the tunnel interface to the IP layer. Although this packet is received from the (pseudo) tunnel interface, the kernel processes it in exactly the same way as it processes a packet received from a (real) network interface. The IP header of the IP packet is then stripped off at the IP layer, and then the packet is passed up to the TCP/UDP layer.
(viii) At the TCP/UDP layer, the TCP/UDP header of the TCP/UDP packet is stripped off and the remaining data segment is then stored into the socket receive buffer.
(ix) Finally, the agent logic of the right CA uses the standard POSIX socket-interface API's (e.g., recvfrom(), read(), etc.) to read the data segment from the socket receive buffer.

With the capability to use the real-life TCP/UDP/IP protocol stack in the Linux kernel, NCTUns generates realistic TCP/UDP/IP protocol stack simulation results for wireless vehicular communication networks. In addition, because a CA is an independent user-level application program using the standard POSIX system calls to get services from the operating system, it can be easily and quickly deployed in the real world without any modification once its functions and performance have been verified in simulated environments. This will save much time and effort.

3.3. CA (car agent)

When a CA is forked by the SE, it reads the road network specification file to build its own road network database. This database will be queried by the agent logic during simulation. In addition, it reads the car profile mapping file to know which car profile is assigned to it and then reads the corresponding car profile file. As stated before, different car profiles determine different moving characteristics such as maximum speed, maximum acceleration, and maximum deceleration. Finally, a CA reads the obstacle specification file to construct visual obstacles for the driver. A CA is an independent process that communicates with the command server located within the SE through its car/signal information socket-interface API's, which are TCP-based. Besides, as stated before, a CA exchanges Internet packets with other CA's through its network packet socket-interface API's, which are TCP- or UDP-based over the simulated network.

A CA is the mobility controller of its associated vehicle simulated in the SE. The agent logic in a CA is the decision maker that determines when to make an action. After setting up the connection with the command server located within the SE, the agent logic first retrieves the node identifier of the associated vehicle through a NCTUns-specific system call. With the node identifier, the agent logic informs the command server to enable its associated vehicle in the SE. In addition, the agent logic queries for its initial position from the command server. The command server retrieves the queried position from the car information database and then sends it back to the agent logic. Before moving a vehicle, the agent logic sets the current speed and current acceleration of the vehicle to zero.

During simulation, the agent logic periodically updates/accesses the car and signal information databases through the car/signal information socket-interface API's. For example, the agent logic may store/retrieve the current moving direction, current speed, current acceleration, current position of a vehicle and retrieve the state of a traffic light that is the nearest in front of its associated vehicle. The command server provides not only update/access services but also data-analyzing services, such as the one used in the traffic-lights-state retrieving case. In this case, the command server first retrieves the current position of a specific vehicle indicated by the agent logic from the car information database and all traffic lights' positions from the signal information database. Because the command server does not have the road network information, the nearest traffic light in front of a vehicle is obtained by doing some mathematical calculations on the information provided by the agent logic. After the nearest traffic light is identified, its state is sent back to the agent logic. With this information, the agent logic can control the vehicle to drive across a crossroad if the traffic light is green or stop it at a crossroad if the traffic light is red.

Moreover, the agent logic can retrieve the position of the nearest vehicle that is in front of a vehicle and within the visual area of the driver from the command server. Because the command server considers only vehicular positions and not obstacle objects when finding the nearest vehicle, the agent logic has to analyze the retrieved position again with consideration of obstacle objects. If the straight line between a vehicle's position and the nearest vehicle's position crosses any obstacle's rectangular area, then from the vehicle driver's viewpoint this nearest vehicle should not be seen by the driver. This view-blocking setting is useful for simulating a crossroad where tall buildings are located at the four corners of the crossroad, which block drivers' views when they are about to make turns.

Other commands are also provided for the agent logic to collect comprehensive information to make driving decisions. For example, the agent logic obtains the direction of a road ahead of the vehicle so that the vehicle can move in the correct direction on the road. Another example is that the agent logic obtains the information of neighboring lanes so that the vehicle can safely change lanes and/or overtake other vehicles. Yet another example is that the agent logic obtains the information of the crossroad ahead of the vehicle so that the vehicle can make a turn smoothly. On top of the default autopilot intelligence, a user can easily add more intelligence into the agent logic of a CA. A user can also easily replace the default autopilot intelligence with more advanced autopilot intelligence.

3.4. SA (signal agent)

The SA is forked by the SE with its group identifier. As presented before, the four traffic lights surrounding a crossroad are grouped together by assigning all of them a unique group identifier. An SA with a given group identifier is responsible for controlling the state changes of the four traffic lights with the same group identifier. Using the group identifier as an index, the agent logic of an SA uses the signal information socket-interface API's to retrieve the type-specific data of a traffic signal, such as the initial state of a traffic light, from the command server located in the SE. The command server retrieves the queried data from the signal information database and sends it back to the agent logic. During simulation, the agent logic periodically exchanges the states of the two pairs of the traffic lights surrounding a crossroad. It uses the signal information socket-interface API's to update each traffic light's state in the signal information database.

4. Validation of simulation results

The results of a simulator should be validated with either mathematical modeling results (when the simulated system is simple enough to be modeled) or real-life data (when the simulated system is too complicated to be modeled) before it can be trusted. NCTUns integrates network simulations with traffic simulations. Because NCTUns directly uses real-life network protocol stacks and application programs to generate realistic network simulation results, here we focus only on the validations of its traffic simulation results.

Traffic simulations consist of two components – road network simulation and vehicle mobility simulation. As presented before, NCTUns now can import a real-life map to automatically construct its corresponding road network. Therefore, validation of road network simulation now is not a concern and in the following we focus only on validation of vehicle mobility simulation.

As presented before, a CA is the mobility controller of its associated car simulated in the SE. The agent logic in a CA communicates with the SE frequently during simulation to update its information in the information databases located in the SE or retrieve a vehicle's or a signal's current information from these databases. For example, to perform the car-following operation, a CA needs to constantly retrieve the current information (including location, speed, etc.) of the vehicle ahead of it and feed the information into its internal logic unit to quickly determine the speed and acceleration/deceleration that it should use next. Ideally, each CA should update its location at a very high frequency (say, every 1 ms) so that when another vehicle retrieves its current information from the SE, the retrieved information will be very accurate. However, to reduce the communication overhead between the CA's and the SE (this is especially important when a very large number of vehicles are simulated), currently in NCTUns a CA/SA is set to update its information into the information databases every 200 ms. This design improves scalability but at the cost of minor inaccuracy in the retrieved information about a vehicle. When simulating a case, a user can select an appropriate update frequency to suit his/her needs.

In this section, we test the CA mobility control with respect to two fundamental driving behaviors: reaction to traffic light signal and car following. In each test, a vehicle needs to accelerate/decelerate or maintain its speed during the simulation. The correctness of the vehicle's mobility can be validated against the results derived from Newton's laws of motion, which are presented below:

$$\begin{cases} v_2 = v_1 + a(t_2 - t_1) \\ s = v_1(t_2 - t_1) + \frac{1}{2}a(t_2 - t_1)^2 \end{cases}$$

where v_1 is a vehicle's speed at time t_1, v_2 is the vehicle's speed at time t_2, a is the vehicle's acceleration/deceleration, and s is the vehicle's displacement from t_1 to t_2. By the formulas, we first derive the changes of vehicle's position and velocity over time. Then, we perform the simulation and log the same information over time. Finally, we compare these two results to observe if the theoretical driving behavior is correctly simulated by NCTUns. If the validation shows that NCTUns correctly simulates a vehicle's acceleration/deceleration behavior, since any complicated moving behavior is just a mix of acceleration and deceleration, one can be sured that if one implements a driving behavior model correctly in the agent logic of a CA, the vehicle controlled by the CA will correctly exhibit the driving behavior during simulation. In the following, we present the scenario and comparison results for each test.

4.1. Reaction to traffic light signal

In the first test, we would like to observe the CA's reaction to a traffic light signal. Fig. 8 illustrates the scenario used in this test. At the beginning, the tested car is stopped and a traffic light is placed in front of the car. The distance between the car and the traffic light is 200 m. Next, the car starts moving with a fixed acceleration of 1 m/s^2. When the car reaches its maximum speed of 30 m/s, it keeps this speed for a while. Later on, when the car sees the traffic light signal turn red and the distance between itself and the traffic light is less than 30m, it starts slowing down with a fixed deceleration of 2m/s^2 until it stops in front of the stop line at the traffic light.

Fig. 9 shows the changes of the tested car's velocity over time. The x-axis is the elapsed time while the y-axis is the tested car's velocity. One curve in this figure is derived from the formulas and the other is obtained from the simulation. One sees that the controlled mobility curve (obtained from simulation) is very consistent with the theoretical mobility curve (derived from the formulas). This consistency confirms that a CA can correctly control its associated car according to a specified reaction to a traffic light signal.

4.2. Car following

In the second test, we would like to observe a CA's car-following behavior. Fig. 10 illustrates the scenario used in this test.

(i) Fig. 10a: At the beginning, the rear car starts moving from 0 m/s and the front car keeps moving at 20 m/s. The initial distance between the rear car and the front car is 30 m. The rear car's acceleration is set to 1 m/s^2 and its maximum speed is set to 30 m/s.

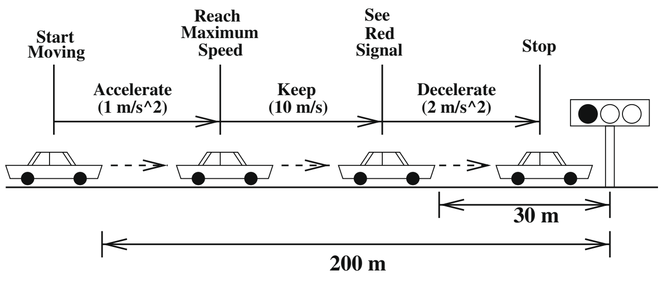

Fig. 8. The moving scenario used in the reaction to traffic light signal test.

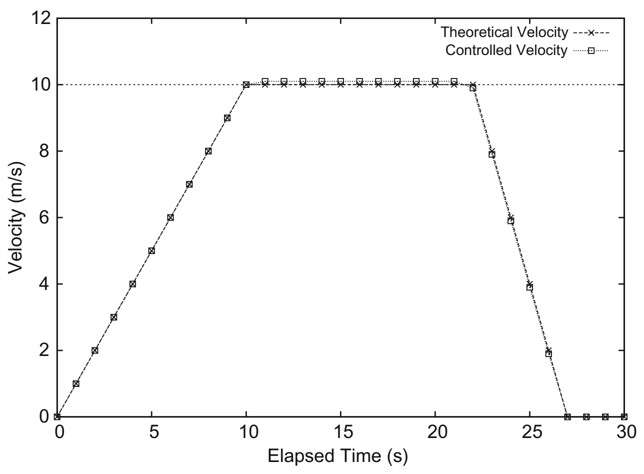

Fig. 9. The changes of the tested car's velocity over time.

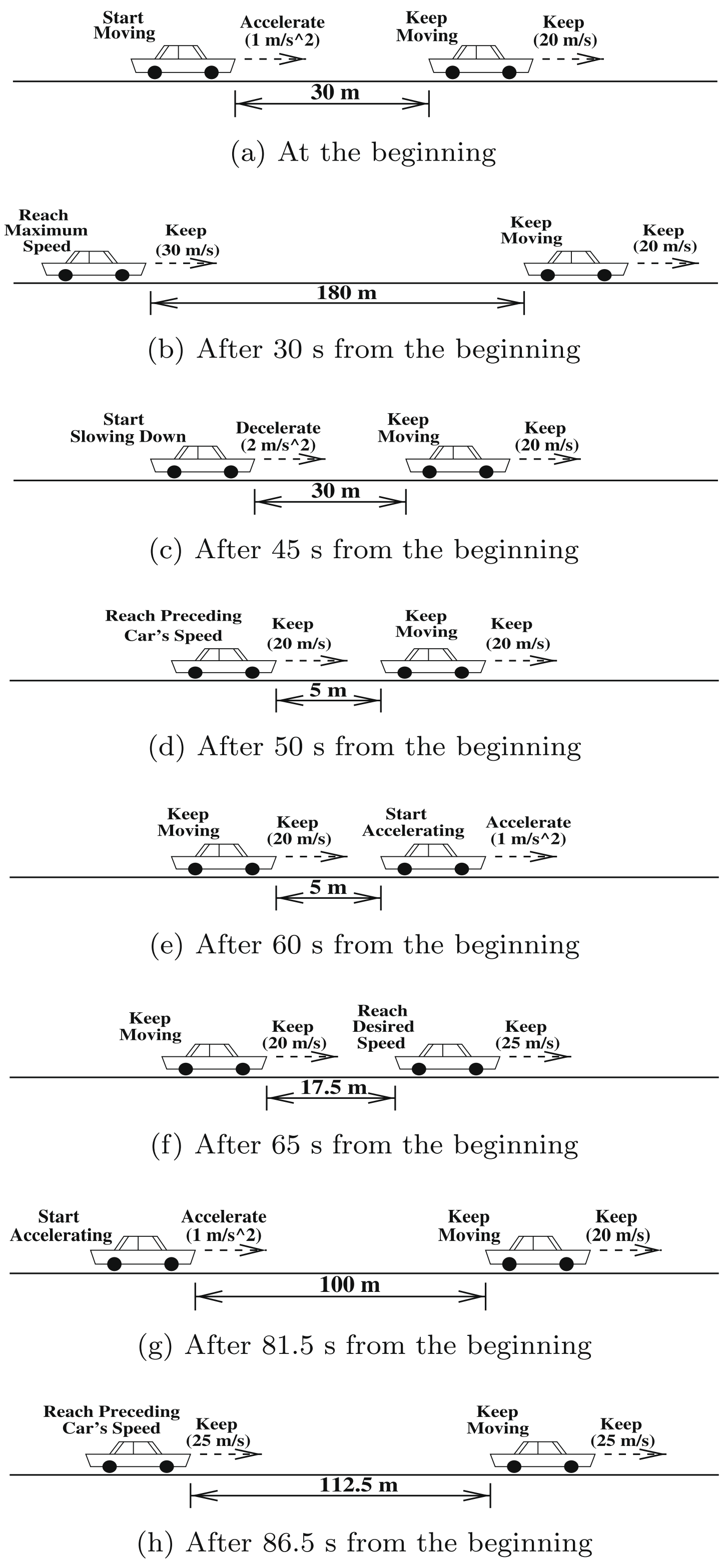

Fig. 10. The moving scenario used in the car-following test.

(ii) Fig. 10b: After 30 s from the beginning, the rear car reaches its maximum speed (i.e., 30 m/s) and stays at that speed. At this time, the distance between the two cars is 180 m. Because the rear car's speed is higher than the front car's speed, the rear car will gradually approach the front car when time moves on.

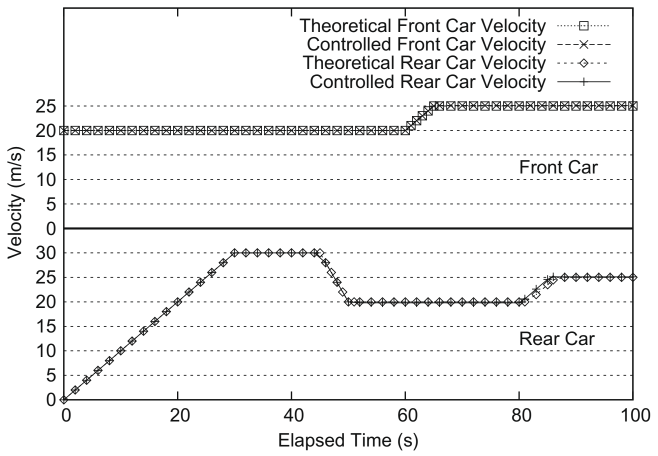

Fig. 11. The changes of the tested cars' velocities over time.

(iii) Fig. 10c: After 45 s from the beginning, the rear car finds that the distance between itself and the front car is only 30 m and thus decides to start slowing down to avoid colliding with the front car. Its deceleration is set to 2 m/s^2 and its desired speed is set to the front car's current speed (i.e., 20 m/s).

(iv) Fig. 10d: After 50 s from the beginning, the rear car reaches its desired speed (i.e., 20 m/s) and keeps that speed. At this time, the distance between the rear car and the front car is 5 m.

(v) Fig. 10e: After 60 s from the beginning, the front car starts speeding up. Its desired speed is set to 25 m/s and its acceleration is set to 1 m/s^2. The current distance between the rear car and the front car is still 5 m. This speed-up will gradually increase the distance between the two cars when time moves on.

(vi) Fig. 10f: After 65 s from the beginning, the front car reaches its desired speed (i.e., 25 m/s) and keeps that speed. At this time, the distance between the two car is 17.5 m.

(vii) Fig. 10g: After 81.5 s from the beginning, the rear car finds that the distance between itself and the front car is 100 m and decides to start speeding up. Its desired speed is set to the front car's current speed (i.e., 25 m/s) and its acceleration is set to 1 m/s^2.

(viii) Fig. 10h: Finally, after 86.7 s from the beginning, the rear car reaches its desired speed (i.e., 25 m/s) and stays at that speed. At this time, the distance between the two cars is 112.5 m.

Fig. 11 shows the changes of the front car's and the rear car's velocities over time. The *x*-axis is the elapsed time while the *y*-axis shows the tested cars' velocities. For each car, both its theoretical velocity curve and its controlled velocity curve are shown. In order not to mingle the four curves together, we plotted the two curves of the front car at the top of the figure and plotted the two curves of the rear car at the bottom of the figure.

From this figure, one sees that (1) for each car, its theoretical velocity curve and controlled velocity curve are very consistent, (2) for the front car, its velocity changes totally match with the moving scenario specified for it, and (3) for the rear car, its car-following behavior totally matches with what it should perform. These results together show that a CA can correctly control its associated car to perform a specified car-following behavior.

In summary, the validation results presented in this section show that NCTUns correctly simulates a vehicle's acceleration/deceleration movements. This means that when one correctly implements a specific driving behavior model in the agent logic of a CA, the vehicle controlled by the CA will correctly exhibit the specified driving behavior.

5. Performance evaluations

In this section, we evaluate the simulation performance of NCTUns with respect to the elapsed time and the physical memory usage of each run-time component, including the SE, the CA, and the SA. Two important system parameters are studied in this paper: the number of road blocks and the number of vehicles deployed in a simulated wireless vehicular communication network. The simulation machine used in our evaluations is a desktop computer equipped with a P4 2.53 GHz CPU and 1 GB RAM. The total time to be simulated for each simulation case is set to 500 s.

The topology of the road network is a 6×6 grid network, as shown in Fig. 12. The edge length of each grid is 1 km. Thus, the simulated field covers an area of 36 km^2. The edge of a grid is a road that is formed by four lanes. Each lane is in turn formed by a single or multiple road block(s). In the grid road network, a crossroad is positioned at each intersection. A T-shape or a L-shape road is simulated by closing the unused branch(es) of a crossroad. No roadside unit is deployed at a closed branch.

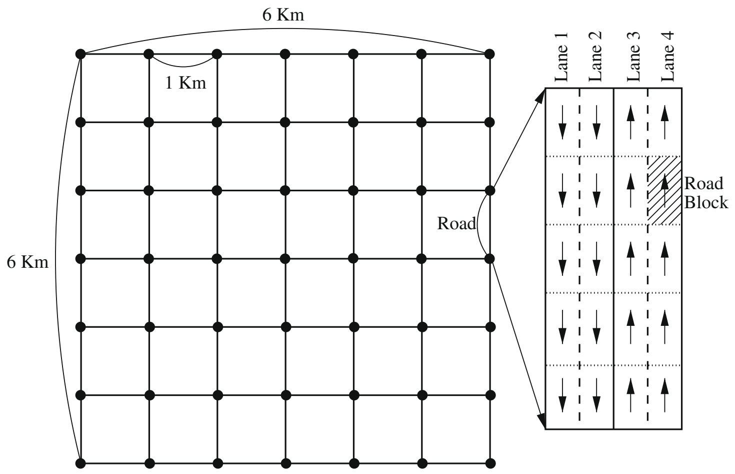

Fig. 12. The topology of the grid road network used for performance evaluation.

The car profile settings and distribution shown in Table 2 are applied to all simulation cases. Currently, we use the maximum (allowable) speed of a vehicle as the desired maximum speed for its driver. The maximum speed settings and distribution reflect the normal driving speeds (from 40 km/h to 80 km/h) in a urban area, where the road network is similar to the used grid topology. The range of the used maximum accelerations reflects the characteristic of a normal vehicle that can accelerate from 0 km/h to its maximum speed in about 10 s. Finally, the range of the used maximum decelerations reflects the characteristic of a normal vehicle that can decelerate from its maximum speed to 0 km/h in about 3 s.

Regarding the network communication scenario, the CA running on each vehicle is programmed to broadcast a 1084-byte UDP packet (1056 bytes for the data payload, 20 bytes for the IP header, and 8 bytes for the UDP header) once per second to the vehicles located within its wireless transmission range. A simplified wireless PHY-layer module is used, which uses 250 and 550 m as the wireless transmission range and interference range, respectively. The used wireless MAC-layer module is based on 802.11(b) standard. No routing protocol is adopted because all packet transmissions are based on broadcast.

5.1. Number of road blocks

In the first evaluation, in total 200 vehicles are deployed in each of the five simulation cases. Without changing the topology shown in Fig. 12, we vary the number of road blocks on each lane from 1 to 5 in the five cases by purposely using different sizes of road blocks. Therefore, we deploy 385, 721, 1,057, 1,393, and 1,729 road blocks in these cases, respectively. The total number of road blocks deployed in a case can be calculated by the formula: [Number of crossroad + (number of roads $\times$ number of lanes on each road $\times$ number of road blocks on each lane)].

Because the size of the area of the road network is kept the same in all cases, the density of vehicles on the road network is the same in all cases. This keeps the simulation overhead of broadcasting UDP packets about the same in all cases. Increasing the number of road blocks will increase the size of the road network database. This may increase the memory space usage of each CA as it needs to store every road block information into its road network database. This may also increase the query time of a CA when it searches for the information of a road block. For example, every time when a vehicle reaches the end of a road block, it needs to get the information of the new road block ahead of it. Thus, when the number of road blocks increases, we expect to see increased physical memory usage for a CA and increased time for running a simulation (i.e., the elapsed time).

The results shown in Table 3 confirm the above conjectures. One sees that the elapsed time increases slightly as the number of road blocks increases. In addition, as expected, the physical memory usage of a CA increases slightly as the number of road blocks increases. The slight performance change between two adjacent cases indicates that the number of road block has little impact on the simulation performance of NCTUns.

Table 2
Car profile settings and distribution.

Profile number	1	2	3	4	5
Percentage	10%	35%	35%	15%	5%
Max speed (m/s)	11	14	17	19	22
Max acceleration (m/s^2)	1.1	1.4	1.7	1.9	2.2
Max deceleration (m/s^2)	3.67	4.67	5.67	6.33	7.33

Table 3
Elapsed time and physical memory usage in each case with different number of road blocks.

Simulation settings					
Number of vehicles	200				
Number of crossroads	49				
Number of roads	84				
Number of lanes on each road	4				
Number of road blocks on each lane	1	2	3	4	5
Total number of road blocks	385	721	1,057	1,393	1,729
Simulation results					
Elapsed time (min)	17.3	20.1	20.6	21.3	21.9
Physical memory usage of the SE (MB)	15.22	15.85	15.87	15.85	15.85
Physical memory usage of a CA (MB)	2.14	2.20	2.27	2.34	2.75
Physical memory usage of an SA (MB)	1.08	1.08	1.08	1.08	1.08

Table 4
Elapsed time and physical memory usage in each case with different number of vehicles.

Simulation settings							
Number of road blocks	1,729						
Number of vehicles	250	350	450	550	650	750	850
Simulation results							
Elapsed time (min)	55.8	108.7	173.2	244.6	341.9	456.1	525.3
Ratio of simulated time to elapsed time	1:7	1:13	1:21	1:29	1:41	1:55	1:63
Physical memory usage of the SE (MB)	29	37	46	57	70	85	110
Physical memory usage of a CA (MB)	2.75	2.75	2.75	2.75	2.75	2.75	2.75
Physical memory usage of an SA (MB)	1.08	1.08	1.08	1.08	1.08	1.08	1.08

5.2. Number of vehicles

In the second evaluation, in total 1729 road blocks are deployed in each of the five simulation cases. We deploy 250, 350, 450, 550, 650, 750, and 850 vehicles in these cases, respectively.

Because the total number of road blocks is the same in all cases, the overhead of the road network database in terms of the access time and the physical memory consumption by a CA is the same in all cases. Increasing the number of vehicles will increase the vehicle density on the road network. Therefore, the SE needs to spend more time on broadcasting (and receiving) more UDP packets. Also, the SE needs to consume more memory space for storing these UDP packets in its program during simulation. Thus, it is expected to see increased time for running the simulation and increased physical memory usage for the SE.

The results shown in Table 4 confirm the above conjectures. One sees that the elapsed time and the physical memory usage of the SE increase with the number of vehicles deployed on the road network. In addition, the case with 850 vehicles requires about 525 min to complete the 500-s simulation. The ratio of simulated time to elapsed time is about 1:63. In other words, in this case, the advance of the simulated virtual time is 63 times slower than that of real-time.

6. Usage examples

NCTUns supports the IEEE 802.11p/1609 communication technology proposed for wireless access in vehicular environments (WAVE) and the IEEE 802.16e communication standard proposed for mobile WiMAX environments. With NCTUns, we have conducted several researches about wireless vehicular systems and applications. In the following, we briefly describe them as usage examples of NCTUns.

In [25], we designed a vehicle collision warning system that employs vehicle-to-infrastructure communication technologies. This system is deployed at an intersection during simulation to reduce the chance of collisions. In [26], we evaluated and improved TCP/UDP performances of IEEE 802.11p/1609 networks, and in [27] we improved the channel utilization of IEEE 802.11p/1609 networks. These researches are all conducted on NCTUns and have been published in the literature.

7. Ongoing work

In this section, we discuss the designs of the current release of NCTUns that can be further improved in the next release.

–*Traffic signal*
Currently, the only supported type of traffic signal is traffic light. Other different types of traffic signals, such as the stop sign and speed-limit sign, will be provided in the next release. This extension allows more types of road networks to be constructed in NCTUns.

–Vehicle mobility model

With more complex road networks being supported, the microscopic vehicle mobility model has to be improved to support new road network environments, such as a freeway with toll stations. For example, when approaching a toll station on a freeway and finding that there is a long waiting line ahead, some drivers not only slow down their vehicles but also change to an adjacent lane with a shorter waiting line. It is clear that a more intelligent microscopic vehicle mobility model is required to support these behaviors.

8. Conclusion

In this paper, we present NCTUns, an open source tool that tightly integrates communication/network simulation with road/traffic simulation. We compare its architecture with those of other simulators to explain why it can uniquely provide a fast feedback loop between network simulation and traffic simulation. This capability enables a user to use NCTUns to study novel ITS applications in which a vehicle needs to change its moving behavior immediately after receiving a message from a neighboring vehicle or from the infrastructure network.

Besides, we present the functions provided by NCTUns that can facilitate conducting researches on this platform. These functions include manual/automatic road network construction, car profile setting, automatic vehicle deployment, pre-specified/autopilot vehicle movement settings, network protocol settings, and so on.

In this paper, we use two cases to validate the traffic simulation of NCTUns and the validation results show that NCTUns correctly simulates a vehicle's acceleration/deceleration on the roads. This means that when one correctly implements a specific driving behavior model in the agent logic of a CA, the vehicle controlled by the CA will correctly exhibit the specified behavior during simulation. We also evaluate the simulation speed and the physical memory usage of NCTUns under different simulation parameter settings. Our evaluation results show that increasing the number of deployed roads affects the simulation speed minimally. However, as expected, increasing the number of deployed vehicles decreases the simulation speed.

Due to the unique capabilities provided by NCTUns and its complete supports for advanced ITS researches, NCTUns is now widely used in the research community. Several researches conducted on NCTUns have been published in the literature and there is an active user group in its forum discussing how to use it to perform various advanced ITS researches.

References

[1] IEEE 802.11p/D3.0: Draft standard for information technology – telecommunications and information exchange between systems – local and metropolitan are networks – specific requirements – Part 11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Standards Activities Department, July 2007.
[2] IEEE 1609.1, Trial-use standard for wireless accesses in vehicular environments (WAVE) – resource manager, IEEE Vehicular Technology Society, October 2006.
[3] IEEE 1609.2, Trial-use standard for wireless accesses in vehicular environments (WAVE) – security services for applications and management messages, IEEE Vehicular Technology Society, October 2006.
[4] IEEE 1609.3, Trial-use standard for wireless accesses in vehicular environments (WAVE) – networking services, IEEE Vehicular Technology Society, October 2006.
[5] IEEE 1609.4, Trial-use standard for wireless accesses in vehicular environments (WAVE) – Multi-channel operation, IEEE Vehicular Technology Society, October 2006.
[6] S.Y. Wang, C.L. Chou, C.H. Huang, C.C. Hwang, Z.M. Yang, C.C. Chiou, C.C. Lin, The design and implementation of the NCTUns 1.0 network simulator, Computer Networks 42 (2) (2003) 175–197. June.
[7] NCTUns 5.0, Network Simulator and Emulator. <http://NSL.csie.nctu.edu.tw/nctuns.html>.
[8] The Network Simulator – ns-2. <http://www.isi.edu/nsnam/ns>.
[9] The QualNet Software. <http://www.scalable-networks.com/>.
[10] The OPNET Modeler. <http://www.opnet.com/>.
[11] R. Barr, Java in Simulation Time/Scalable Wireless Ad Hoc Network Simulator. <http://jist.ece.cornell.edu>.
[12] The ptv Simulation – VISSIM. <http://www.english.ptv.de/cgi-bin/traffic/traf_vissim.pl>.
[13] The TransModeler Traffic Simulator. <http://www.caliper.com/transmodeler/>.
[14] The SUMO Traffic Simulation Package. <http://sumo.sourceforge.net/index.shtml>.
[15] A Real-Time Freeway Traffic Simulator – FreeSim. <http://www.freewaysimulator.com>.
[16] A Microscopic Traffic Simulation Model – CORSIM. <http://www-mctrans.ce.ufl.edu/featured/TSIS/Version5/corsim.htm>.
[17] H. Wu, J. Lee, M. Hunter, R.M. Fujimoto, R.L. Guensler, J. Ko, Simulated vehicle-to-vehicle message propagation efficiency on Atlanta's I-75 corridor, in: Transportation Research Board Conference Proceedings, Washington DC, 2005.
[18] Multiple Simulator Interlinking Environment (MSIE) for C2CC in VANETs. <http://www.cn.uni-duesseldorf.de/projects/MSIE>.
[19] C. Schroth, F. Dotzer, T. Kosch, B. Ostermaier, M. Strassberger, Simulating the traffic effects of vehicle-to-vehicle messaging systems, in: Proceedings of the Fifth International Conference on ITS Telecommunications, Brest, France, 2005.
[20] The TraNS (Traffic and Network Simulation Environment). <http://wiki.epfl.ch/trans>.
[21] B. Khorashadi, A. Chen, D. Ghosal, C.N. Chuah, M. Zhang, Impact of transmission power on the performance of UDP in vehicular ad hoc networks, in: IEEE International Conference on Communications, ICC 2007.
[22] L. Bononi, M. Di Felice, M. Bertini, E. Croci, Parallel and distributed simulation of wireless vehicular ad hoc networks, in: Proceedings of the ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM), Torresmolinos, Spain, 2006.
[23] R. Vuyyuru, K. Oguchi, Vehicle-to-vehicle ad hoc communication protocol evaluation using simulation framework, in: Proceedings of the Fourth IEEE/IFIP Wireless on demand Networks and Services, Austria, 2007, pp. 100–106.
[24] C. Gorgorin, V. Gradinescu, R. Diaconescu, V. Cristea, L. Ifode, An integrated vehicular and network simulator for vehicular ad-hoc networks, in: Proceedings of the European Simulation and Modelling Conference (ESM), Bonn, Germany, May 2006.

[25] S.Y. Wang, Y.W. Cheng, C.C. Lin, W.J. Hong, T.W. He, A vehicle collision warning system employing vehicle-to-infrastructure communications, in: IEEE WCNC 2008 (Wireless Communications and Networking Conference 2008), Las Vegas, USA, March 31–April 3, 2008.
[26] S.Y. Wang, H.L. Chao, K.C. Liu, T.W. He, C.C. Lin, C.L. Chou, Evaluating and improving the TCP/UDP performances of IEEE 802.11(p)/1609 networks, in: IEEE ISCC (IEEE Symposium on Computers and Communications 2008), Marrakech, Morocco, July 6–9, 2008.
[27] S.Y. Wang, C.L. Chou, K.C. Liu, T.W. Ho, W.J. Hung, C.F. Huang, M.S. Hsu, H.Y. Chen, C.C. Lin, Improving the channel utilization of IEEE 802.11p/1609 networks, in: IEEE WCNC (Wireless Communications and Networking Conference 2009), Budapest, Hungary, April 5–8, 2009.