TraCI: An Interface for Coupling Road Traffic and Network Simulators

Axel Wegener[1], Michał Piórkowski[2], Maxim Raya[2], Horst Hellbrück[1], Stefan Fischer[1] and Jean-Pierre Hubaux[2]
[1] Institute of Telematics, University of Luebeck, Germany
{wegener, hellbrueck, fischer}@itm.uni-luebeck.de
[2] LCA Lab, EPFL, Lausanne, Switzerland
{michal.piorkowski,maxim.raya,jean-pierre.hubaux}@epfl.ch

Keywords: Vehicular Ad-Hoc Networks (VANETs), Network Simulation, Node Mobility

Abstract

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 asses 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.

In contrast to the existing approaches, i.e., generating mobility traces that are fed to a network simulator as static input files, the online coupling allows the adaptation of drivers' behavior during simulation runtime. This technique is not limited to a special traffic simulator or to a special network simulator. We introduce a general framework for controlling the mobility which is adaptable towards other research areas.

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.

1. INTRODUCTION

VANETs appear to be the most promising applications of mobile ad-hoc networks and will have a strong impact on road traffic efficiency and safety as provisioned by the Intelligent Transportation System (ITS) community [1].

An exhaustive evaluation of VANET applications requires more than a deep understanding of information generation and networking aspects within VANETs. It is just as important to investigate how vehicular applications affect the movement of vehicles, because as a consequence of a more extensive knowledge of the surrounding traffic patterns (as reported by VANET applications) drivers will adapt their driving strategy – ranging from speed changes to route selection. This new driver behavior might cause again a change in traffic patterns. For example, if drivers are warned about hazardous road conditions, they will adjust their speed accordingly. In return, this will also affect other vehicles, including those that are not part of the VANET.

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 evaluate VANET applications in realistic scenarios.

Our contribution is the new open-source architecture for coupling traffic and network simulators that aims to achieve 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].

The paper is organized as follows. In Section 2. we present the related work. In Section 3. we introduce the concept of

 ISBN 1-56555-318-7

mobility primitives that set the ground for mobility commands used in our framework. Next, we describe the proposed system architecture. We devote the Section 4. to the data exchange protocol, which is the main ingredient of our architecture. We present the implementation details in Section 5. and evaluate the performance of TraCI design choices in Section 6. Finally, in Section 7. we conclude the paper.

2. 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-licenced, as well as licensed, tools that are widely used in both the ITS and VANET communities.

3. APPROACH

3.1. Mobility Primitives

Any complex mobility pattern, which is a result of a decision taken by a driver, can be decomposed into a sequence of *mobility primitives* such as ‘change speed’, ‘change lane’, ‘change route’, etc. These mobility primitives are independent from VANET applications. They depend on macroscopic (road network topology, speed limits, etc.), as well as microscopic (current vehicle speed, location, etc.) mobility constraints.

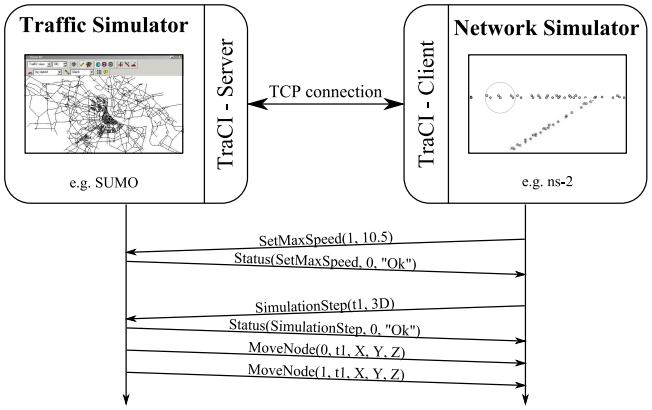


Figure 1. System architecture and an operation example of the command-response exchange between the network and the traffic simulator presented as a timeline diagram. This example shows an exchange of four TraCI messages, two commands: [*SetMaxSpeed*] and [*SimulationStep*] and two responses: [*Status*] and [*Status*, *MoveNode*, *MoveNode*]. It corresponds to a scenario when two vehicles exchange VANET messages and one of them decides to adjust its maximum speed.

To better understand the concept of mobility primitives, let us consider the Road Condition Warning application that belongs to the class of Vehicle-to-Vehicle Decentralized Environmental Notification applications as proposed by the Car2Car Communication Consortium (C2C-CC) [21]. A vehicle that detects an incident immediately starts broadcasting a specific warning message to nearby vehicles; this message includes the type of warning, its location and occurrence time. Each vehicle that receives such a warning may adjust its movement accordingly. For example, while approaching the scene, a vehicle may first slow down, then change the lane and finally speed up.

In order to identify all mobility primitives specific to VANETs, we studied the set of vehicular applications proposed by C2C-CC. In Table 1, we present a taxonomy of VANET applications, i.e., use cases and the associated mobility primitives. The complex mobility patterns of a driver can be represented by a certain sequence of mobility primitives proposed here. We rely on these mobility primitives to specify the set of *atomic mobility commands* used by the network simulator to control movements of vehicles. The detailed definition of these commands is presented in Section 4.

3.2. System Architecture

TraCI avoids the creation of mobility traces prior to the ad-hoc network simulation. Both simulators run concurrently, whereby the ad-hoc network simulator controls the road traffic simulator. Therefore, we extend the road traffic simula-

Table 1. VANET application taxonomy proposed by the Car-to-Car Communication Consortium [21] and the resulting mobility primitives.

Application Type	Use Cases	Mobility Primitives				
		Change speed	Stop	Change lane	Change route	Change target
V2V Cooperative Awareness	Traffic Congestion Warning	+		+	+	+
	Forward Collision Warning	+	+	+		
	Cruise Control	+	+			
	Intersection Collision Warning	+	+			
V2V Unicast Exchange	Pre-Crash Sensing	+	+	+		
	Approaching Emergency Vehicle	+	+	+		
	Merging Assistance	+	+	+		
V2V Decentralized Environmental Notification	Road Condition Warning	+		+	+	
	Low Bridge Warning	+				
	Freezing Bridge/ Road Warning	+				
	Work Zone Warning	+		+	+	
	Fog Zone Warning	+				
	Post Crash Warning	+	+	+	+	
	Incident Detection	+	+	+	+	+
I2V Communication	Curve Speed Warning	+				
	Speed Limit Warning	+				
	Stop Light Assistance	+	+			

tor by a *TraCI-Server* and the network simulator by a *TraCI-Client*. These two components communicate over a TCP connection. The general system architecture is depicted in Figure 1

Once the TCP connection is established, the network simulator controls the traffic simulator via the data exchange protocol, which enables movement changes for each simulated vehicle. Thus, it is possible to instantaneously adjust the movement of individual vehicles due to information generated within the VANET.

Data exchange between the simulators is controlled by the network simulator that sends commands as requests to the road traffic simulator. Upon reception of the commands, the traffic simulator performs actions that result in one or more responses to each command.

To ensure time synchronization between both simulators, the network simulator sends periodically, every simulation step, e.g. 1[s], a command to the road traffic simulator that contains the actual simulation time plus one simulation step (cf. Section 4.2.). The road traffic simulator performs the next simulation step and sends the resulting vehicle positions back to the network simulator. Those vehicle positions are converted into linear movements by the network simulator, so that all vehicles reach their positions at a time instant as specified by the traffic simulator (cf. Section 4.4.).As a small drawback of this method, the actual execution of the atomic mobility commands is delayed by one simulation step, since the road traffic simulator is always one simulation step ahead. However, given that the simulation time step is usualy very short - for traffic efficiency applications approximately 1[s] and for safety applications - 0.1[s], this kind of drawback is negligible.

When the simulation ends either in the traffic or the network simulator, it closes the TCP connection. This causes the other simulator to stop too.

4. PROTOCOL DETAILS

As described in the previous section, 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 (cf. Figure 2) to bundle, respectively, a set of *commands* or *responses*.

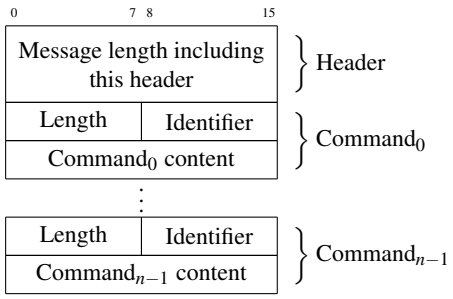


Figure 2. TraCI message format: TraCI messages are composed of a small header and a variable number of commands.

The TraCI message, as depicted in Figure 2, 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 $\in [0, 127]$ are commands from the network simulator to the mobility generator. The respective responses have an identifier $\in [128, 255]$. 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:

- **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.
- **Status** is sent as a reply to every request command. It contains a status flag and a descriptive text.
- **Move Node** contains a 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 represent the atomic mobility primitives identified for the use cases in Section 3. They are used to affect the behavior of a single vehicle.

- **Set Maximum Speed** limits vehicle's speed or removes such a limit (*change speed* primitive).
- **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).
- **Change Lane** fixes a vehicle to a specific lane for a certain amount of time.
- **Change Route** causes the traffic simulator to reroute an individual vehicle by setting a particular travel time for a road segment.
- **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.

- **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.
- **Position Conversion** converts between different types of positions, e.g., to map a Cartesian coordinate to an appropriate position on the road network.
- **Driving distance** calculates the path and estimated time to get from one position to another based on the underlying road network.

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

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. Due to space limits, we refer the reader to [22] for environmental commands and their full references.

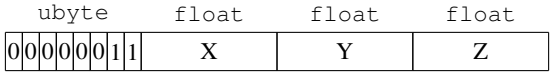
4.1. Data Types

To save the computation time, our protocol does not use structural information within its messages as XML does. Instead, 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 datatype 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, a `ubyte` identifier is put in front of each position (later on referred to as *PositionType*).

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



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

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	Pos	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.

4.2. Command *Simulation Step* (Id 0x01)

double	ubyte
TargetTime	PositionType

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 as described in Section 4.3. and a collection of *Move Node* responses – one for each active node – as stated in Section 4.4. The returned node positions are either 3D or road map representations according to the requested *PositionType* (cf. also Section 4.1.). All these responses are bundled as one TraCI message.

4.3. 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.

4.4. Response *Move Node* (Id 0x80)

integer	double	position
NodeId	TargetTime	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 *TargetTime*. The representation of the *Position* is interpreted according to the requested *PositionType* of the *Simulation Step* command.

4.5. Command *Set Maximum Speed* (Id 0x11)

integer	float
NodeId	MaxSpeed

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*.

4.6. Command *Stop Node* (Id 0x12)

integer	position	float	double
NodeId	StopPosition	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 allows to 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 *StopPosition* 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 (*WaitTime*), by setting its maximum speed to zero, before it goes ahead on its route.

4.7. Command *Change Lane* (Id 0x13)

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.

4.8. Command *Change Route* (Id 0x30)

integer	string	double
NodeId	RoadId	TravelTime

The command *Change Route* allows a vehicle identified by *NodeId* to react to certain traffic situations by adapting its route. Therefore, an individual (estimated) *TravelTime* 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.

4.9. Command *Change Target* (Id 0x31)

integer	String
NodeId	RoadId

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.

In Figure 1, we present a simple example of TraCI message exchange that corresponds to the simulation scenario, where two vehicles communicate over VANET and one of them decides to adjust its speed.

5. IMPLEMENTATION

As our main goal is to offer an open-source tool that is modular, and thus easily extensible, we have chosen two open-source simulators to implement the TraCI framework for VANETs. We have selected the Simulation of Urban MObility simulator (SUMO) [5] as the road traffic simulator and the Network Simulator (ns2) [2] as the wireless ad-hoc network simulator. Both simulators are well established in the research community.

SUMO is an open-source, portable, microscopic road traffic simulator, designed to handle large road networks. It is also a multimodal simulator, i.e., it is capable of processing different types of traffic (buses, cars, etc.). As in most of the available traffic simulators the mobility model in SUMO is based on a car following model, especially the model proposed by Krauss [23].

Ns2 is the most widely used network simulator in the mobile ad-hoc network community. It provides extensions with more accurate and up-to-date models [24], including a wireless interface adjusted according to the IEEE 802.11p draft (the envisioned technology for VANETs) and probabilistic radio wave propagation.

Although we focus our implementation effort on the aforementioned simulators, we would like to stress that our framework is neither limited to a special road traffic simulator nor to a special network simulator. We demonstrate this by providing an additional TraCI-Client implementation for the algorithmic centric ad-hoc network simulator Shawn [4]. Furthermore, a TraCI-Client implementation for JiST/SWANS [3] is currently conducted. We would like to refer the reader to the TraCI wiki [22] for more implementation details and the latest version of the protocol. The actual download of SUMO, patches for ns2 as well as other simulation tools that provide TraCI are also available.

5.1. Simulation Run

As in our approach the traffic simulator acts as a server, it must be started first. SUMO provides two additional command-line parameters used for starting it in TraCI-Server mode.

- `--port <int>` Sets the server port on which SUMO listens for an incoming connection. If set, the simulation is triggered by using the *Simulation Step* command.
- `--penetration <float>` Defines the ratio of vehicles (between 0 and 1) that are equipped with a radio interface. This value defaults to 1.

In the case of ns2 as network simulator, the TraCI-Client is set up inside the TCL script that also configures the network scenario. The following code instantiates a TraCI-Client object that is responsible for the connection to a TraCI-Server. Using `set-remoteHost` and `set-remotePort`, the connection to the TraCI-Server is specified. Afterwards, the time interval for calls of the *Simulation Step* command is set and the `startSimStepHandler` starts the periodic *Simulation Step* commands. The TCP connection is established automatically.

```
set traciC [new TraCIClient]
```

```
$traciC set-remoteHost localhost
$traciC set-remotePort 8888
$traciC set-timeInterval 1.0
$traciC startSimStepHandler
```

Whenever a node is instantiated, it must be attached to the TraCI-Client using the following line of code:

```
$traciC add-node $node($i)
```

To start the VANET application on a network node at the moment the corresponding vehicle starts to move in the traffic simulation, the corresponding ns2 agent must be registered using `start-on-move`. To disable a network application when the vehicle leaves the simulation `stop-on-halt` is used accordingly.

```
$traciC start-on-move $node($i) $agent($i)
$traciC stop-on-halt $node($i) $agent($i)
```

A node's behavior can be influenced using TCL as well as C++. The following TCL code sends a *Set Maximum Speed* command to the traffic simulator, so that `$node($i)` slows down to zero:

```
$traciC command-setMaximumSpeed $node($i) 0
```

To make the same call from within C++ use the following line of code:

```
TraCIClient::instance()
    ->commandSetMaximumSpeed(node, 0);
```

For the full reference, we refer the reader to [22].

6. EVALUATION

We show that our TraCI approach can compete with the traditional way of using trace files and give thereafter an application example, where the TraCI approach allows to dynamically reroute vehicles.

At first, we compare three different kinds of mobility sources in a ns2 network simulation:

- **Static.** Nodes do not move during simulation.
- **Traces.** Mobility is generated prior to the network simulation. Thereby SUMO generates an output file that is converted to a ns2 trace file by using traceExporter.
- **TraCI.** Mobility is generated on-the-fly by SUMO that is controlled via TraCI by ns2.

To benchmark only the influence of mobility sources, no network packets are exchanged between the wireless network nodes inside the ns2 simulation.

The road traffic scenario is the same for all simulations. 1500 vehicles drive on a circular highway segment of 78.5 km length (25 km diameter) for a duration of 2 h. At the time when the VANET technology will be on the market, only a subset of operating vehicles will be equipped with it. Hence the penetration ratio of this technology will be far less than 100 %. We can simulate this using the proposed architecture by reporting only a portion of the 1500 vehicles to ns2. We measured the simulation runtime of the three approaches with a varying number of equipped vehicles. The evaluation was performed on a 3.4 GHz Intel Pentium D workstation with 2 GB of RAM.

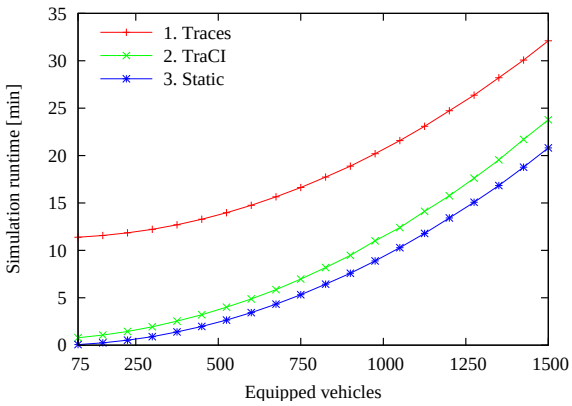


Figure 3. Runtime comparison between traditional sequential execution of SUMO, traceExporter and ns2 (Traces) versus interactive run of SUMO and ns2 coupled by TraCI. A ns2 simulation with static nodes serves as reference (Static).

The results are shown in Figure 3, whereby each line depicts one of the aforementioned approaches.

The 'Traces' curve shows the runtime as a sum of consecutive executions of SUMO, traceExporter and ns2. SUMO needs about 78 s to generate a netstate-dump file containing all the vehicle's movements. This is fed into traceExporter that converts it to a ns2 trace files with varying rates of equipped vehicles, resulting in different numbers of nodes that get into the ns2 simulation (about 10 min). Only the runtime of ns2 depends on the number of nodes and reaches from about 4 s for 75 nodes to about 21 min for 1500 nodes.

When using the 'TraCI' approach, SUMO and ns2 run interactively and are connected through TraCI. We have to sum up the particular runtimes of both simulators to obtain the simulation runtime, as depicted by the 'TraCI' curve in Figure 3. SUMO, together with its TraCI-Server, needs between 38 s and 47 s depending on the number of equipped vehicles that are reported to ns2 (between 75 and 1500). Ns2 together with its TraCI-Client consumes between 9 s (75 nodes) and 23 min (1500 nodes).

In the 'static' case, we have no mobility at all. This simulations serves as a reference showing the simulation time needed by the ns2 core itself.

The above results show that our TraCI approach competes well with the traditional approach of using trace files. In comparison to the runtime of ns2 in static case, it is obvious that

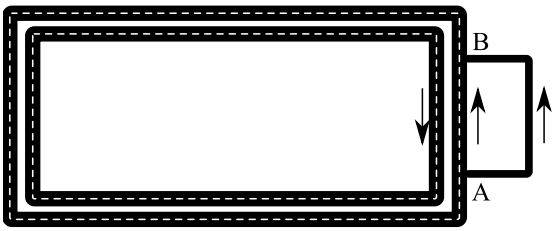


Figure 4. Road traffic scenario used for the exemplary application. It is composed of a circular two-way road, whereby a detour exists between *A* and *B*.

introducing mobility does not significantly extend ns2's runtime at all. We even benefit from using TraCI, because no netstate-dumps need to be written to a hard drive and converted afterwards.

After demonstrating the efficiency of the TraCI approach, we build a simple application that dynamically reroutes vehicles, according to local traffic conditions. Therefore, we modify the aforementioned road traffic scenario as depicted in Figure 4. We add two opposite lanes for two-way traffic to allow for easier packet exchange in the VANET. An additional road is connected to the main road between *A* and *B*. It is twice as long (15 km) as the segment of main road (7.5 km); thus it is normally unused, but can serve as detour. To achieve the same traffic density, as in the previous test, the overall number of vehicles is increased with the road size to 3000.

The VANET application – that is implemented in ns2 – measures on each equipped vehicle the needed time to travel from *A* to *B*. After passing that road segment, the data dissemination protocol AutoCast [25] is used to inform oncoming vehicles about the measured travel time. When TraCI is active, the VANET application on vehicles approaching *A* inform SUMO by the *Change Route* command about the expected travel times. Thereupon, SUMO reroutes those vehicles to the detour.

During the simulation, an accident on the main road blocks one of the two lanes. This leads to a traffic jam, increasing the travel time between *A* and *B*.

Figure 5 shows on the y-axis the resulting travel times throughout the simulation (x-axis) that are needed to pass the road segment between *A* and *B* taking into account the main road as well as the detour. Travel times in Figure 5 are averaged over 15 min of simulation time. The accident happens after 60 min, followed by a traffic jam throughout the rest of the simulation. The curve named 'Fixed traces' shows the scenario without TraCI; the mean travel time raises from about 5 min up to 40 min, since the detour could not be used. The second curve 'TraCI with 30 % equipped vehicles' show the same scenario where 30 % of the vehicles participate in the VANET and use the detour to drive round the traffic jam; so that the mean travel time for all vehicles is limited

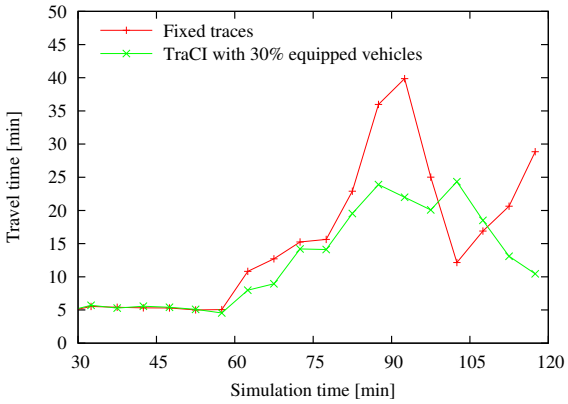


Figure 5. Mean time for travelling from *A* to *B* (cf. Figure 4). By using TraCI, vehicles could dynamically use the detour to decrease the travel time.

to 25 min.

Even if this exemplary VANET application is not optimized in any way, it shows the potential application domain TraCI clears the way for.

7. CONCLUSIONS

We have presented **TraCI**, 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 [13]. Note that the approach is not limited to ad-hoc network simulations as the proposed mobility interface is generic to control the road traffic generator. Based on the taxonomy of VANET applications proposed by C2C-CC, we have derived basic control commands for the road traffic simulation and have implemented interfaces for SUMO and ns2 - two open-source simulators developed by different research communities.

We have evaluated the performance of TraCI and have demonstrated that it is even faster to couple a network simulator with the road traffic simulator instead of writing trace files first and run the network simulation in the next step, even if we do not need a control loop during execution.

Using the TraCI interface one can even evaluate complex VANET scenarios, such as accidents (e.g. by stopping certain vehicles at a certain time instant) or simulating hazardous road conditions (e.g., by modifying the speed of vehicles which have directly 'witnessed' such conditions).

We currently use TraCI to implement and evaluate various VANET applications proposed by C2C-CC [21]. In future work, we will extend TraCI to support a larger diversity of network and road traffic simulators.

REFERENCES

[1] IEEE Intelligent Transportation Systems Society. http://www.ieee.org/its.

[2] Kevin Fall and Kannan Varadhan. The ns manual, 1989–2001.

[3] JiST/SWANS: Java in Simulation Time/Scalable Wireless Ad hoc Network Simulator. http://jist.ece.cornell.edu.

[4] Sandor P. Fekete, Alexander Kröller, Stefan Fischer, and Dennis Pfisterer. Shawn: The fast, highly customizable sensor network simulator. In *Proc. of INSS '07*, 2007.

[5] Daniel Krajzewicz, Michael Bonert, and Peter Wagner. The Open Source Traffic Simulation Package SUMO. In *RoboCup 2006 Infrastructure Simulation Competition*, Bremen, Germany, 2006.

[6] VISSIM: microscopic, behavior-based multi-purpose traffic simulation program. http://www.ptvamerica.com/vissim.html.

[7] MATsim: Multi-Agent Transport Simulation Toolkit. http://www.matsim.org.

[8] TRANSIMS: Transportation Analysis Simulation System. http://www.ccs.lanl.gov/transims/index.shtml.

[9] Richard M. Fujimoto, Randall Guensler, Michael P. Hunter, Hao Wu, Mahesh Palekar, Jaesup Lee, and Joonho Ko. CRAWDAD data set gatech/vehicular (v. 2006-03-15), March 2006. http://crawdad.cs.dartmouth.edu/gatech/vehicular.

[10] Valery Naumov, Rainer Baumann, and Thomas Gross. An Evaluation of Inter-Vehicle Ad Hoc Networks Based on Realistic Vehicular Traces. In *Proc. of MobiHoc '06*, pages 108–119, 2006.

[11] Rapid Generation of Realistic Simulation for VANET. http://www.csie.ncku.edu.tw/ klan/move/index.htm.

[12] TraceExporter: Exports SUMO dumps as ns2 traces. http://sumo.sourceforge.net/wiki/index.php/traceExporter.

[13] TraNS: Joint Traffic and Network Simulator - Application-centric framework for evaluation of VANETs. http://trans.epfl.ch.

[14] Amit Kumar Saha and David B. Johnson. Modeling Mobility for Vehicular Ad Hoc Networks. In *Proc. of VANET '04 (Poster abstract)*, pages 91–92, 2004.

[15] D. R. Choffnes and F. E. Bustamante. An Integrated Mobility and Traffic Model for Vehicular Wireless Networks. In *Proc. of VANET '05*, pages 69–78, 2005.

[16] J. Härri, F. Filali, C. Bonnet, and Marco Fiore. VanetMobiSim: Generating Realistic Mobility Patterns for VANETs. In *Proc. of VANET '06 (Poster abstract)*, pages 96–97, 2006.

[17] Rahul Mangharam, Daniel S. Weller, Daniel D. Stancil, Ragunathan Rajkumar, and Jayendra S. Parikh. GrooveSim: A Topography-Accurate Simulator for Geographic Routing in Vehicular Networks. In *Proc. of VANET '05*, pages 59–68, 2005.

[18] 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. In *Proc. of WiVec'07*, 2007.

[19] Stephan Eichler, Benedikt Ostermaier, Chritoph Schroth, and Timo Kosch. Simulation of Car-to-Car Messaging: Analyzing the Impact on Road Traffic. In *Proc. of MASCOTS '05*, pages 507–510, September 2005.

[20] Christian Lochert, Murat Caliskan, Björn Scheuermann, Andreas Barthels, Alfonso Cervantes, and Martin Mauve. Multiple Simulator Interlinking Environment for IVC. In *Proc. of VANET '05 (Poster abstract)*, pages 87–88, 2005.

[21] Roberto Baldessari, Bert Bödekker, Matthias Deegener, Andreas Festag, Walter Franz, C. Christopher Kellum, Timo Kosch, Andras Kovacs, Massimiliano Lenardi, Cornelius Menig, Timo Peichl, Matthias Röckl, Dieter Seeberger, Markus Straßberger, Hannes Stratil, Hans-Jörg Vögel, Benjamin Weyl, and Wenhui Zhang. Car-2-Car Communication Consortium - Manifesto (Version 1.1), August 2007. http://www.car-2-car.org/index.php?id=570.

[22] Wiki for the **Tra**ffic **C**ontrol **I**nterface. http://sumo.sf.net/wiki/index.php/TraCI.

[23] Stefan Krauß. *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics*. PhD thesis, Universität zu Köln, April 1998.

[24] Marc Torrent-Moreno. *Inter-Vehicle Communications: Achieving Safety in a Distributed Wireless Environment: Challenges, Systems and Protocols*. PhD thesis, Universitätsverlag Karlsruhe, 2007.

[25] Axel Wegener, Horst Hellbrück, Stefan Fischer, Christiane Schmidt, and Sándor Fekete. AutoCast: An Adaptive Data Dissemination Protocol for Traffic Information Systems. In *Proc. of VTC '07*, Baltimore, USA, October 2007.