Realistic mobility simulator for smart traffic systems and applications

Cosmin-Stefan Stoica, Ciprian Dobre, Florin Pop
University Politehnica of Bucharest
Bucharest, Romania
cosmin.stoica@cti.pub.ro, ciprian.dobre@cti.pub.ro, florin.pop@cti.pub.ro

***Abstract*—Cars have become essential elements of modern life. But nowadays the increasing number of cars also leads to problems: pollution, traffic jams, wasted time spent in traffic because of traffic bottlenecks, etc.. Traffic cannot cope anymore with the rate of car usage today. Fortunately, in the last years various Intelligent Transportation Systems (ITS) demonstrate innovative services relating to different modes of transport and traffic management, and enable various users to be better informed and make safer, more coordinated, and 'smarter' use of transport networks. For such systems, a lot of attention has been dedicated to develop realistic models of roads and of the vehicle mobility, by exploiting the extensive literature developed in the field of transportation systems, e.g., models of how cars move along a road. Here, we present the realization of a more-realistic simulation tool, Sim^2Car. The simulator uses as input data real-life traces collected over long periods of time, involving potentially thousands of drivers, to provide a realistic support mobility model for high-level methods and techniques designed to optimize traffic. On top, developers can simulate advanced traffic networking solutions and intelligent transportation applications, under real-world conditions. We present our results for an application designed to optimize the costs involved in using navigators, that actively incorporate solutions borrowed from opportunistic computing and context data to optimize the user's experience.**

***Index Terms*—realistic simulator, traffic simulation, congestion, real-world mobility data, uncertain location data and correction algorithms**

I. Introduction

Traffic congestion is happening in many urban environments. The road infrastructure capacities cannot cope with the rate of increase in the number of cars. This, coupled with traffic incidents, work zones, weather conditions, make traffic congestion a major concern for municipalities and research organizations [1]. Advanced traffic control technologies may lead to more efficient use of existing road network systems, resulting in reduced traffic congestion, delays, emissions, energy consumption and improved safety. The communication capabilities provided by modern wireless technologies and mobile devices offer opportunities for the development of traffic control applications where cars and devices within the road infrastructure collaborate to solve traffic problems. Thus, Intelligent Transportation Systems (ITS) have socio-economic advantages towards reducing traffic congestion, the high number of traffic road accidents, etc.. Coupled with advances brought by modern vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) communication technologies, ITS research can lead to a pletaphora of traffic applications (e.g., collision avoidance, road obstacle warning, safety message dissemination, etc.), traffic information and infotainment services (e.g., games,etc.) [2].

Simulation has a major role in designing and evaluating such solutions. In the last years increase attention has been dedicated to developing realistic models of vehicle mobility (i.e., we previously proposed such a mobility model based on the social behavior of drivers in [3]). However, such models are still unable to acurately reproduce the way drivers behave in real-world cities. Several well-known simulators (ns-2, ns-3, SWANS, OMNET++, OPNET, etc.) are able to take as input trace files, but such files have to be generated by specialized vehicle mobility models (e.g., Vanet-MobiSim, SUMO), which mimic vehicle mobility behaviors. However, in some scenarios, the inderdependencies among vehicle communication, or their mobility patterns, are still insufficiently close to real-world. For example, when a congestion/accident occurs, the vehicle mobility changes due to triggers from an ITS service.

In this paper, we propose to advance the field, by providing realistic simulation for the evaluation of ITS, using real-world mobility traces. The input is represented by real-world mobility data, collected over a time interval from traffic, in different cities (i.e., we previously developed such a traffic collector app for Android, that uses crowd-sensing techniques [2]). The few previous attempts to create realistic traffic models were mostly based on wrong conclusions [4], mostly because of uncertainty in GPS readings. We advance on that, and present the Sim^2Car simulator. On a microscopic level, each car moves according to the mobility trace. On a microscopic level, we use OpenStreetMap (OSM) to build the street graph, and to correct the mobility traces using an algorithm to correct the GPS inaccurate position of each car.

We present experiments with mobility traces of 500 cabs from San Francisco Bay, C.A., and prove the capabilities of the simulator – in this case, Sim^2Car is able to successfully cope with a simulation of realistic taxi movement, covering a total distance of trajectories of approx. 9 million kilometers.

The rest of the paper is structured as follows. Section II discusses related work. Sections III presents the proposed solution, while Section IV describes the implementation details. Section V presents experimental results and analysis the obtained performance. Section VII concludes and presents future work.

Proceedings 28th European Conference on Modelling and Simulation ©ECMS Flaminio Squazzoni, Fabio Baronio, Claudia Archetti, Marco Castellani (Editors)
ISBN: 978-0-9564944-8-1 / ISBN: 978-0-9564944-9-8 (CD)

II. Related Work

Different authors previously attempt to provide realistic conditions for the evaluation in simulation of various traffic and vehicular communication mechanisms. SUMO [5], [6] is one of the mostly cited purely microscopic open-source simulator for traffic, designed to handle large road networks. SUMO integrates a large suite of tools for street network processing and traffic patterns generation. Internally, SUMO uses a graph to model the network of modeled streets within a city. The streets (with one or more lanes) are represented as edges of an oriented graph, with nodes being the intersections [6]. Edges can keep additional information, such as speed restrictions, street category, traffic lights position, and other traffic signs, etc.. The netgenerate tool is capable in SUMO to automate the generation of streets networks (as grid-networks, circular spider networks and random networks). For realistic simulations, SUMO provides netconvert, a tool capable to transform digital world maps (e.g., in OpenStreetMap format) into SUMO-format.

MOVE [7], [8] extends SUMO with advanced usability capabilities. The tool automates the generation of simulation scenarios, and adds new features, such as the capability to generate traces from Tiger database. However, its current implementation does not offer support for traffic parameters (as compared to the standard SUMO generated traces, it does not include features as lane change or obstacle mobility model), and cannot easily support large simulations (with many cars).

VNSim [9], [10] is a microscopic simulator capable to generate synthetic, social-driven, mobility, with a network model on top. Unfortunately, even if the synthetic mobility model in VNSim can be generated starting from TIGET maps, it still lacks the realism needed to correctly evaluate ITS mechanisms and techniques.

Analyzing previous ITS simulators, they suffer from the problem of huge memory consumption in case of intensive simulation scenarios. We analyzed the possibility to use previous simulators with mobility traces with more than a thousand cars (a realistic assumption, considering real-world conditions, where most likely more than a thousand car will use the application under test), and they all fail to scale. For example, trying to avoid this memory problem, SUMO (the only other simulator capable to handle this) deals with large traces by simplifying the imported map according to its own data format. SUMO provides the tool to import data from online maps (i.e., an OpenStreetMap format), but this operation is highly memory consuming (in our experiments, we were able to cope with scenarios of maximum 100 cars, on a powerful workstation). The problem is that when netconvert transforms an OSM map into a SUMO network file, the size of the generated file becomes considerably larger than the original one. Also, another problem with netconvert is the low level of details which are kept in the resulting SUMO network, with incomplete information about streets, number of lanes on a street, etc. [11].

Currently, almost all tracking devices use GPS technologies, so the data representation in GPS coordinates (WGS84) is a requirement for being a general simulator. SUMO and MATSim use Cartesians coordinates. The direct utilization of mobility traces collected by some GPS receivers in a raw format is impossible, because they firstly need to be processed and converted to the demanding coordinates system of these simulators. When SUMO is connected to the real sensors it manifests an overhead determined by the conversion of coordinates having delays that can affect the traffic flow. MOVE being an extension of SUMO inherits all his problems.

Another problem with analyzed traffic simulators is the fact that their implementations are single threaded, and this radically increases the time of simulation in case of intensive traffic experiments. The authors of [12] confirm this aspect, with an experiment with 10.000 vehicles simulated on the streets infrastructure of Coimbra, Portugal, which lasted 27.000 seconds, an unrealistic amount of time, more than the actual simulated time.

III. Architecture

Sim^2Car is a simulator designed to provide new research capabilities for next-generation traffic applications. It offers a generic tool for vehicular traffic evaluation: its underlying mobility and networking models are easily extendable; it supports large-scale simulations involving potentially thousands of cars; it offers a realistic mobility model that incorporates real-world data captured in traffic (currently it provides out-of-the-box capabilities for practitioners to evaluate their ITS solutions considering realistic scenarios involving real-world taxis in San Francisco or Beijing); it offers tools to correct uncertainty of location information related to flaws in GPS data sensing and conversion mechanisms to various coordinate systems. The realistic property of its mobility models is boasted by the use of real GIS datasets, freely provided by the OpenStreetMap community. The geographical information has enough details about an urban environment in order to be a good support to generate realistic mobility traces and to simulate complex traffic scenarios. Therefore, developers could construct simulations where cars can sense their environments and take decisions based on the graph of streets declared by the open-source community and already made available within the OpenStreetMap project [13].

Sim^2Car includes a tool called TracesTool. It can correct raw data collected by traffic infrastructure sensors (as the data publicly available in real-world data traces – see [14] for a collection of such public data) and make it compatible with the OpenStreetMap digital map format. TracesTool is also capable to generate mobility traces that conform to the real street network provided by OSM, and respect the real traffic flow from GPS datasets.

Sim^2Car's architecture is designed to easily integrate and simulate a wide range of traffic scenarios (see Figure 1). Around the simulator core, TracesTool ensures several functions: building street network graph, correction of the raw mobility datasets collects by GPS sensors like [14], [15], [16],

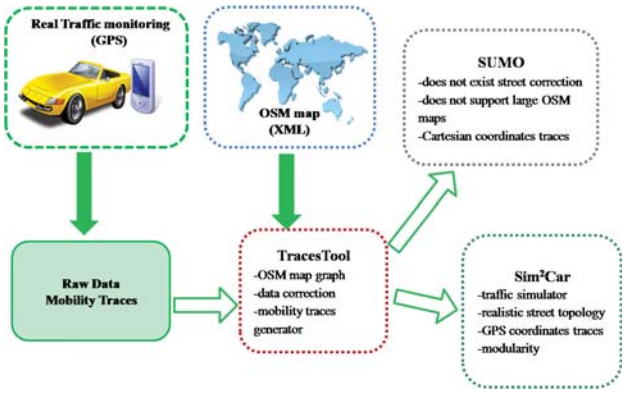

Figure 1. The proposed architecture

generating new mobility traces from corrected traces under the restrictions of OSM graph.

The simulator is designed to support simulation scenarios with overlarge street networks. It can generate new synthetic mobility models, starting for example from the original mobility trace. This allows, for example, to modify a simulated vehicular traffic scenario according to different conditions: congestion rate, level of pollution, restricted areas, etc.. Thus, Sim^2Car is a simulator easily configurable and extendable, for next-generation ITS applications. Figure 1 shows the proposed architecture, together with a possible execution flow. The simulator also offers the possibility to interoperate with SUMO, through resulted mobility traces from TracesTool, modifying them in the SUMO format.

A. TracesTool

TracesTool is a Java-based application that helps users convert the mobility traces into Sim^2Car's data format, and deal with uncertainty in GPS sensing – the raw data collected by GPS receivers from cars contains a lot of errors (generally, GPS sensing is known to have a deviation +/-50-100 meters [17] from the real position). TracesTool implements a correction algorithm (described below), to corrects the GPS data. To maintain the realistic characteristic of the original vehicular data, all operations conform to the map graph obtained from OSM map.

The new resulted traces respect the real streets topology, but they do not have an uniform time resolution (due to the periods when GPS receiver does not transmit car position to data collector center). Thus, an improvement of the quality of simulation is also presented below – we use a flow-centric correction algorithm to supply intermediary trace points when necessary.

B. Sim^2Car

Sim^2Car is next constructed as a discrete time simulator – simulation clock advances with a fixed time unit. The simulator provides capabilities to simulate realistic mobility models, as it combines GIS data from an OSM graph with GPS mobility datasets. The generated traffic closely follows the real-world original mobility conditions.

Figure 2 shows the modular architecture of simulator, and how its components interact. The *Simulation Engine* is the core of the simulator. Its role is to analyze vehicles involved in simulation, and execute their actions for different time moments (the internal clock is advanced with a fixed time resolution).

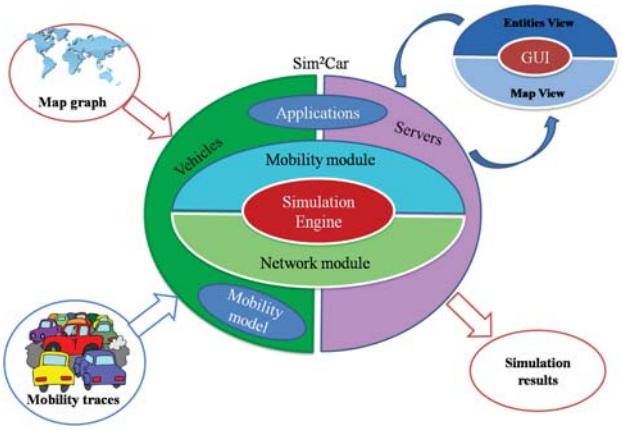

Figure 2. Sim^2Car Architecture

The *Vehicle* entities, next, model the car behavior. Every car is characterized by a mobility module (in charge with the update of the traffic route), a network module (dealing with communication capabilities, an application layer (implements an application logic, under simulation) and optionally the graphical view.

The *Server* entities model static elements, generally positioned with the street infrastructure, at fixed locations. For example, we can model intelligent traffic lights, Induction Detection Loops, various servers. Each server is characterized by network module (communication capabilities), an application layer (the application logic) and optionally a graphical view.

The *mobility module* updates the position of each vehicle according to the input traces. It follows the traffic rules and positions cars on passing streets. Also, the communication between simulated nodes is sustained by the *network module*, where communication is realized via messages exchange. The network module offers a generic interface which can be particularized to different technologies (i.e., Wireless, Bluetooth, cell-based communication). For example, Sim^2Car provides a *Wireless network* module, that integrates functions such as: discovering of peers and/or servers, sending and receiving messages (all used by the application layer).

The *application layer* simulates the application running on car's computer (or on the driver's smartphone). Various ITS and traffic applications can be simulated at this layer. Such an application is detailed later on in Section VI.

IV. Implementation

A. TracesTool

TracesTool, written in Java, supports operations for building streets graph, raw datasets corrections and traces generation. It works with several entities: Node, Way, Location.

A point from an OSM map is represented as a *Node* object. It stores details about a POI (Point of Interest), such as the GPS coordinates (latitude and longitude), and the node identifier.

The *Way* object is next used to describe a street parsed from an OpenStreetMap file. It contains the nodes (Node objects) forming the street. We also keep information such as: the neighboring streets which intersects the current one (sorted by junction nodes identifier), street identifier, physical boundaries, and type (one or two directions traffic allowed).

Location describes a point in the mobility trace. The object keeps attributes for a trace point (collected in the real-world trace): geolocation (latitude and longitude), timestamp of collection, information about the vehicle (e.g., for the 500 traces of cabs from San Francisco [14], every record in the trace shows also if the taxi is occupied or not).

We decided to represent the oriented graph of streets as a dictionary of *Way* objets, containing the list of identifiers for all the intersected roads. To increase the searching speed on large maps, we introduced additional levels of indexation, by divided the map surface in squared areas (an edge can be experimentally adjusted according to streets density in the map, and the searching speed increases because the point localization on map becomes a detection of the row and column in the grid).

The algorithm for building the street graph includes two phases:

1) *Parsing Phase* – Initially, the input data of the algorithm is the XML file of a map exported from OpenStreetMap. This raw data is parsed and structured in a format easy to be used in the next step. The resulting file contains all the ways from the map, and for each road we keep the node identifiers and associated tags.

2) *Building Phase* – The previously file is further processed, to detect relations between streets and skip useless OSM elements (i.e., bike paths, pedestrian ways, buildings, and so on). After this step, the algorithm generates several files. A first file contains all vehicular ways. For each street, we store the way identifier, type and all the nodes with associated GPS coordinates. Another file keeps, for each road, the list of junction nodes with correspondent neighbor streets. For this, the algorithm forms a grid on the map, where the edge of a cell is adjusted by the user according to the desired streets density of the map surface. Every cell will further contain the streets which cross through the perimeter delimited by its boundaries.

For the *correction algorithm*, presented below, the Traces-Tool prepares the input using parsed functions adapted to the different GPS raw datasets. The algorithm works correctly with collections of *Location* objects and the *map graph*. To deal with the uncertainty in GPS readings, most GPS correction methods [18] rely on the projection of the GPS reading to the surrounding streets. Our algorithm consider the mobility flow – for each car, each GPS point representing its trajectory are, again, projected on the nearest streets. From the set of candidate corrective points, we select one which conforms to the previous trajectory (i.e., the point that does not disconnect the flow of points on the streets). In this algorithm, the distance between two GPS coordinates is computing using the haversine formula [13].

```
foreach point in traceData{
/*crtAreaStreets - contains all the streets from the square
      where the point is located on the map
projection(point, street ) - return the projection of the
     point on the street
Delta - error range experimentally set between 10 and 100
     meters.*/
crtAreaStreets = getAroundStreets(point);
foreach street in crtAreaStreets{
  dist = dist(point, projection(point, street));
  if( dist < Delta ){
    add(candidates, point, street.id);
  }
}
/* Remove all points with an inacceptable GPS error (no
     candidates)*/
foreach point in traceData{
  if( candidates[point] == [] ){
    remove point from candidates;
  }
}
/* convergence_criteria experimentally established ( the
     remained unresolved nodes ) */
do{
  /* Keep the original traffic flow. Correct the points
       which are situated on the wrong way of a street.*/
  foreach (crt_point,next_point) in traceData{
    next_point_candidates = candidate[next_point];
    foreach street in next_point_candidates{
      if( angle( direction(crt_point, next_point),
           street_direction ) > 90 ){
        remove candidates[next_point][street.id];
      }
    }
  }
  /* Removing too far candidates */
  foreach point in traceData{
    foreach street in candidates[point] {
     if( dist(point, projection(point, street))> 20) {
       remove street from candidates[point];
     }
    }
  }
  /*Remove all neighbors that are not situated on the same
       street or two jointed streets.*/
  foreach street1 in candidate[crt_point]{
    next_point_candidates = candidate[next_point];
    foreach street2 in next_point_candidates{
      /* junction(street1, street2) tests if exists
           junction between street1 and street2
      if( street1.id != street2.id && !junction(street1,
           street2) ){
        remove candidates[next_point][street2.id];
      }
    }
  }
  Analyse a subset of points from trace data and
  if 0.80 of them are on the same street put the
  remained points on the same street.
} while( !convergence_criteria );
}
```

Next, we deal with errors caused by the lack of data. In the input mobility trace, it often happens that nodes (cars) failed to send GPS data with a constant rate all the time to the sink server. Thus, our correction algorithm is not always capable to rebuild the original traffic flow all the times (we actually experimented data gaps in the traffic flow in the generated mobility data). To solve this, we next developed another algorithm to deal with generating synthetical mobility points for the missing gaps, such that to have a continuous flow of mobility for each car. This newly generated data trace has to respect in all details the original car itinerary, and the streets network.

Firstly, we determine the minimum global time over all traces. After that, we process the corrected traces in several phases:

Phase 1 – The algorithm first aligns the start time for each trace (data belonging to each car) to a global minimum time.

Phase 2 – The algorithm next analyses the time interval between consecutive points. According to the resulted time gap, the algorithm treats the traffic flow using different methods. These methods try to keep the realistic aspect of the trace conforming to the original vehicular movement. The effort is additionally sustained by the usage of streets topology, since the algorithm fills the gap between points with new entries which are generated according to the trace context: the locations in the roads graph of the analyzed points and the length of time gap.

Method 1 – If the time interval is equal to the fixed resolution, or higher than the superior limit, the current point is kept in the resulted trace. The situation when the time gap exceeds superior limit is met, for example, when the GPS receiver was stopped for a while. In this case, we keep the point in the resulted trace. When the trace is simulated, Sim^2Car will disable the car drawing on the graphical user interface because of this time gap. The car's disappearance from the map is called "teleportation", and is connected to the situation when the GPS does not have signal in places such as tunnels or underground parking.

Method 2 – In this case the value of time difference is between the value of fixed resolution and the superior limit. If the two analyzed points are on the same street, the algorithm adds to the trace new points, computed using their equidistantly place on the line (determined by the two points and their projection on the road).

Method 3 – During this stage, the algorithm treats two points which are on the intersected streets as having similar time issue. Firstly, the junction point is determined. Next, the number of needed points to fill the gap is computed similarly to the method described in *Method 2*. The new points are equally distributed before and after the cross point.

Method 4 – This method deals the situation where the gap between points is too large, so there are several streets between the two consecutive points. For this case, we apply Dijkstra's algorithm to find the optimal path between them. Before using this algorithm, a subgraph that contains all the streets on a range of a given number of streets is formed around the first point. If the algorithm finds a path between the two points, the algorithm next places new points on every street of the path, with an uniform spatial step. For the Dijkstra's algorithm we use as weight the length of the street.

The traces generator offers a generic format for the resulted traces. One record from the output trace contains the following information about the vehicle at a certain moment: latitude, longitude, timestamp, street identifier, and other details. Adjusting with a format convert, a trace generated in the above way can be used in SUMO having the advantage that the trace points will be placed on the right streets and the disadvantage that SUMO will probably not be able to load the graph of a large map like the one of San Francisco city in the RAM of a personal computer.

B. Sim^2Car Implementation

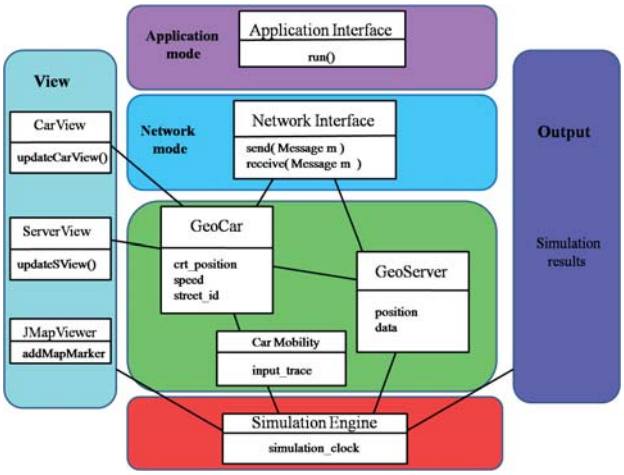

Figure 3. UML diagram of Sim^2Car.

Sim^2Car is developed as a modular simulator. Figure 3 presents an UML diagram of Sim^2Car, showing the main components. At the core is the *simulation engine*. On top, the behavior of the nodes (vehicles, servers) is modeled by the *Entity* class. Depending on the simulated scenario, different types of entities can exist, such as cars and servers.

The mobility is management by the *CarMobility* class, in charge with advancing the current position for each car entity. As an optimization, the class does not load an entire car's trace into memory. Instead, it only reads the car's current position and the next position from the trace file.

Next, the *network module* is a member of the Entity class, because any car and server can have one or more network interfaces. This module is easily extensible, as it can be adjusted to different technologies: 3G/4G/WiMAX, Wireless or Bluetooth, IEEE802.11p, etc. In the current simulator we implemented a Wi-Fi module, which offers through its API methods for discovering simulation nodes and provides communication primitives (using message exchanges). A message is modeled by the *Message* class. In general, the network layer is used at the application layer to communicate to other nodes in proximity, having the same application installed (running in the simulation).

The *Application* interface models the logic at the application layer. By implementing this interface, anyone can add inside the simulation new ITS applications and services: to monitor traffic flow, congestion, routing, etc.. In Section VI we present an example of such an application.

On top, the Graphical User Interface is implemented through a *View* class. The class operates three main areas: map area, control area and logging area. For the slippery map, for example, we use the JMapViewer [19] library, provided by OpenStreetMap community. JMapViewer is a useful tool for rendering an OSM map using online resources or downloaded

tiles. Being an open-source project, it was easily adjustable to our needs, and comes already equipped with a vast collection of GUI features. For example, the GUI currently provides integrated zoom controls, the possibility to add markers and polygons on the represented surface. The control area can be used by the users to control the simulation flow. The logging area shows different information about actions performed by the simulated nodes.

V. Experiments

A. OSM Graph Building Algorithm

For evaluating our corrective algorithms for the mobility data traces, we choose five maps with different sizes from Metro Extracts site [20]. The site provides parts of the OpenStreetMap database for the majority of world cities. We first converted the OSM maps into SUMO network format using netconvert. Next, we converted the same maps into OSM graph format, using our algorithm. A comparison between the resulting graphs and indexation table (in terms of file size, and memory needs to load the data), is presented in Tables I (for SUMO) and II (for our algorithm). The measurement unit is MB, and the acronyms are: *XFS* - XML File Size, *LSNFS* - Large SUMO Network File Size, *RSNFS* - Reduced SUMO Network File Size, *AFS* - Areas File Size, *SFS* -Streets File Size, and *AdFS* - Adjacency File Size.

During our tests, we noticed several debatable situations such as: warnings, unknown OSM XML tags, problems with physical memory. Besides, netconvert demands to edit the original OSM file with the JOSM application (otherwise it does not make differences between pedestrian ways and vehicular street, lanes), and this requires an extra effort for the users.

The map editing on personal computers is not possible all the times for complex maps (e.g., *San Francisco*, involving 500 taxis), because the JOSM tool is not able to load the entire map into memory, or if it succeeds, the experience is quite annoying for the users due to the lack of application interactivity.

After the editing phase, all maps were accepted by netconvert as input, but results were unsatisfactory because the output file size is almost double than original file size. The OSM maps use GPS coordinates for internal elements, while SUMO uses Cartesians coordinates system for data representation (keeping additional metadata for in its internal orientated graph). Therefore, the conversion between both coordinates systems can increase the resulted file size.

We used the same tests for our OSM graph building algorithm. The *Areas File* represents the file which keeps the indexation table for the entire map. The *Streets File* contains all streets from OSM map with their nodes. The *Adjacency File* contains for each street from OSM map lists of neighbors and every list is identified by junction point identifier.

During the experiments with our corrective algorithm, we did not encounter problems. Even for the *Paris* map, the algorithm managed well to handle the load into memory. As the results in Table II show, we manage to utilize an acceptable quantity of the memory, and still offer to the user the possibility to use streets graph in Sim^2Car without leading to memory problems.

Table I
The results of SUMO tests

Map	XFS	LSNFS	RSNFS	RAM Used
San Francisco	255	629	500	$\geq$ 1930
Shanghai	195	509	383	$\geq$ 1723
Paris	3660	OofM	OofM	-
Bucharest	0.876	1.75	1.32	21
Brussels	544	801	456	$\geq$1954

Table II
The tests results for OSM graph building algorithm

Map	XFS	AFS	SFS	AdFS	RAM Used
San Francisco	255	0,982	21,4	9,28	137
Shanghai	195	5,879	27,428	11,019	178
Paris	3660	3,08	58,3	30,03	240
Bucharest	0.876	0,003	0,055	0,033	8
Brussels	544	1.31	24	12.6	175

VI. Use Case Scenario: Tiles Application

To evaluate Sim^2Car's capabilities, we develop on top an application (called *Tile Application*) designed to optimize the storing of location-aware data in a medium composed of fixed points (workstations), wireless access points and smartphone applications (as in typical smart city scenarios). This application, demonstrated in [21], uses context information (location, neighbors) and techniques specific for opportunistic computing to provide an efficient management of the available data for satellite navigation systems based on high resolution raster maps.

We simulated different scenarios using mobility datasets collected in San Francisco [14] and Beijing [15], [16]. The territory of San Francisco, California covers an area of about $121km^2$ [22], whilst Beijing encompasses an area of about 16.807,8 km^2 [23]. For the *Tile Application* we used the following parameters: each Client (car's computer, or smartphone) has a local storage capable to hold up to 5 MB (average space occupied by modern apps). This data is represented by tiles (geographical data needed for the representation of the navigation system). Each tile covers a geographic area of 0,4 km^2, for San Francisco, and 6 km^2, for Beijing. The user can download tiles from neighbors (if available) directly over wireless close-range communication (which actually saves transfer costs). Otherwise, if a tile is not available, the user has to download a tile from a central repository. Each tile has a resolution of 256256 pixels. We actually used real map tiles downloaded from the OSM site, and considered that each contact between simulated entities (car-to-car, car-server) last sufficient to download the necessary tiles. For the communication between cars, we simulated the use of 802.11p technology (with a range of approx. 300 meters, and maximum data throughput somewhere between 6 and 9 Mbit/s) [24].

We present below the results for two scenarios. First, we simulated the movement of all 500 cabs in the San Francisco datasets, over the period of 12 days (from 17th May 2008 to 29th May 2008). In this case Sim^2Car models both cars and servers (the *Tile Application* logic was implemented also for the central repository of tiles, aka the server). We defined 10 servers (wireless routers capable to provide tiles), uniformly spread over the area of San Francisco (we used the main junctions actually). The entire simulation lasted 19.020 seconds (approximately 5 hours and 30 minutes).

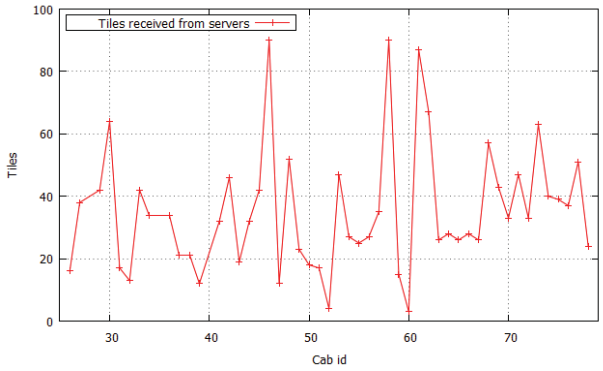

Figure 4. The number of tiles receive by peers from servers (San Francisco).

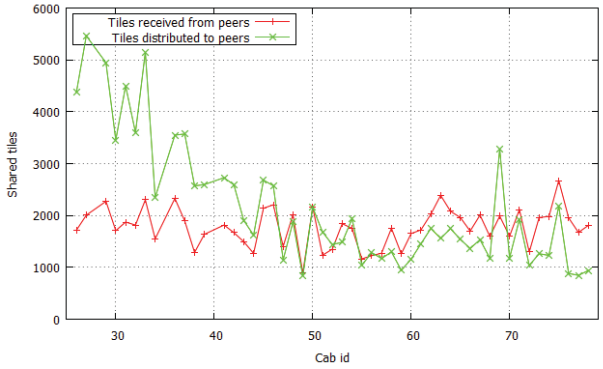

Figure 5. The number of tiles exchange between peers (San Francisco).

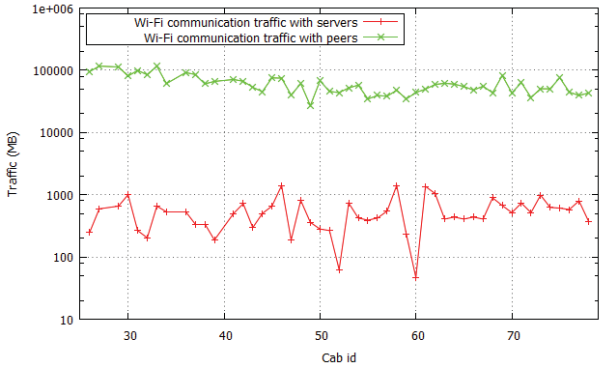

Figure 6. The network traffic over Wi-Fi (San Francisco).

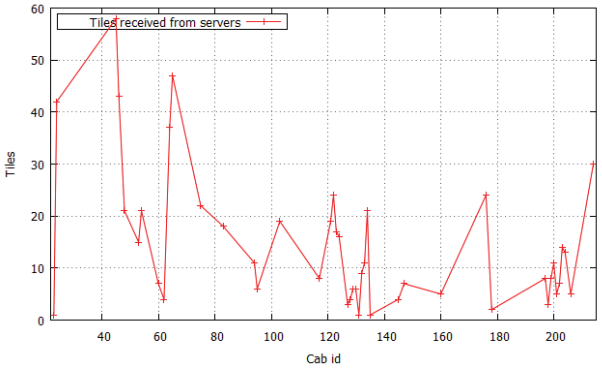

Figure 7. The number of tiles exchange between peers (Beijing).

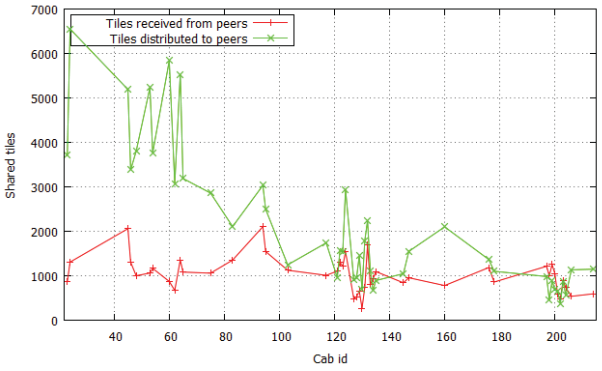

Figure 8. The number of tiles exchange between peers (Beijing).

We randomly selected 50 cabs from all simulated cabs, and plotted the data collected in the simulation. Figure 4 shows the number of the tiles received by peers from servers, and Figure 5 presents the number of tiles provided and the ones obtained to/from peers. The results show that the application can actually lead to a reduction in traffic, which is especially valuable in crowded urban environments. In this case, data can be better disseminated between peers, reducing the expenses of the drivers and the network traffic to central servers. As the results show, peers seldomly established connections to servers to require tiles.

Figure 6 shows the Wi-Fi traffic realized in peer-to-peer and client-to-server communication. In this example, peers send to other peer/server the ID of the required tile (approx. 8 Bytes). A sent/received tile has the cost of 20 KBytes. As seen, peers generally take the needed tiles from the other peers – which is the main reason why peer-to-peer network traffic over Wi-Fi reaches a GB order of magnitude.

In the second set of experiments, we simulated the movement of 1000 cabs from Beijing datasets, during one week (from 2nd February 2008, to 10th February 2008). We added 55 servers, again uniformly spread over the area of Beijing (in its main junctions). The simulation lasted 38.200 seconds (approximately 10 hours and 30 minutes). We also used the same costs for network traffic over Wi-Fi as in the previous set of experiments.

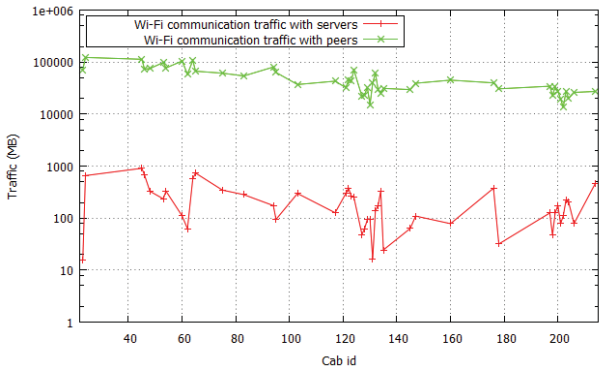

Figure 9. The network traffic over Wi-Fi (Beijing).

Figures 7, 8, and 9) show the obtained results The *Tiles Application* keeps again the traffic over Wi-Fi in similar proportions as in the previous experiments. However, in this case we witness a decrease in the client-to-server traffic over Wi-Fi, which is due to the high density of peers involved in traffic.

VII. Conclusion

In this paper, we presented Sim^2Car, a generic tool for vehicular traffic evaluation which supports large-scale simulations involving potentially thousands of cars. Currently, our simulator provides a realistic mobility model that incorporates real-world traces (San Francisco and Beijing datasets). For the input data, we constructed our tools to correct raw GPS datasets according to streets graph, using a mapping algorithm.

We evaluated the simulator's capabilities under an application, *Tiles Application* that we previously demonstrated in [21]. Under extensive experiments in Sim^2Car, we demonstrate the capability of the simulator to handle a large number of nodes – we successfully simulated large traffic scenarios, involving 500 cabs in San Francisco, and 1000 cabs in Beijing. Sim^2Car manages very well such kind of experiments, being capable to handle thousands of cars and large OSM maps, covering hundreds of km^2.

Acknowledgment

The research is supported by CyberWater grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012. The work was also supported by the project "SideSTEP - Scheduling Methods for Dynamic Distributed Systems: a self-* approach", PN-II-CT-RO-FR-2012-1-0084.

References

[1] C. Systematics, *Traffic congestion and reliability: Trends and advanced strategies for congestion mitigation*. Federal Highway Administration, 2005, vol. 6.

[2] C. Fratila, C. Dobre, F. Pop, and V. Cristea, "A transportation control system for urban environments," in *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on*. IEEE, 2012, pp. 117–124.

[3] A. Gainaru, C. Dobre, and V. Cristea, "A realistic mobility model based on social networks for the simulation of vanets," in *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*. IEEE, 2009, pp. 1–5.

[4] S. Uppoor, O. Trullols-Cruces, M. Fiore, and J. Barcelo-Ordinas, "Generation and analysis of a large-scale urban vehicular mobility dataset," 2013.

[5] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo–simulation of urban mobility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3 and 4, pp. 128–138, 2012.

[6] (2014, Feb.) Simulation of urban mobility. [Online]. Available: http://sumo.sourceforge.net

[7] (2014, Feb.) Move project homepage. [Online]. Available: http://lens.csie.ncku.edu.tw

[8] A. K. Banik, "Routing protocol with prediction based mobility model in vehicular ad hoc network (vanet)," 2010.

[9] V. Cristea, V. Gradinescu, C. Gorgorin, R. Diaconescu, and L. Iftode, "Simulation of vanet applications," *Automotive Informatics and Communicative Systems*, 2009.

[10] (2014, Feb.) Vnsim homepage. [Online]. Available: http://cipsm.hpc.pub.ro/vanet/vnsim.html

[11] A. K. Mario Krumnow, "Simulation of vanet applications," *Real-time simulations based on live detector data Experiences of using SUMO in a Traffic Management System*, 2013.

[12] D. C. S. Jos Capela Dias, Pedro Henriques Abreu, "Simulation of vanet applications," *Preparing Data for Urban Traffic Simulation using SUMO*, 2013.

[13] C. Vetter, "Fast and exact mobile navigation with openstreetmap data, diploma thesis," 2010.

[14] (2014, Feb.) Crawdad community, san francisco bay traces download section. [Online]. Available: http://crawdad.cs.dartmouth.edu/meta.php?name=epfl/mobility

[15] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: driving directions based on taxi trajectories," in *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*. ACM, 2010, pp. 99–108.

[16] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 316–324.

[17] C. D. Dragoi Vlad Alexandru, "Smart traffic, master thesis," 2012.

[18] J. E. Robbins, "Gps correction methods, apparatus and signals," Sep. 28 2004, uS Patent 6,799,116.

[19] (2014, Feb.) Jmapviewer project homepage. [Online]. Available: http://wiki.openstreetmap.org/wiki/JMapViewer

[20] (2014, Feb.) Metro extracts home. [Online]. Available: http://metro.teczno.com

[21] D. Urda, C. Dobre, and F. Pop, "Storing location-aware data in mobile distributed systems," in *Parallel and Distributed Computing (ISPDC), 2013 IEEE 12th International Symposium on*. IEEE, 2013, pp. 135–142.

[22] (2014, Feb.) Us census bureau, state and county quick-facts, san francisco (city), california. [Online]. Available: http://quickfacts.census.gov/qfd/states/06/0667000.html

[23] (2014, Feb.) Beijing: Population and density by district and county. [Online]. Available: http://www.demographia.com/db-beijing-ward.htm

[24] (2014) Mit technology review, the internet of cars is approaching a crossroads. [Online]. Available: http://www.technologyreview.com/news/515966/the-internet-of-cars-is-approaching-a-crossroads