REAL TIME TRAFFIC SIMULATOR FOR SELF-ADAPTIVE NAVIGATION SYSTEM VALIDATION

Vít Ptošek[(a)], Jiří Ševčík[(b)], Jan Martinovič[(c)], Kateřina Slaninová[(d)], Lukáš Rapant[(e)], Radim Cmar[(f)]

[(a),(b),(c),(d),(e)]IT4Innovations, VŠB - Technical University of Ostrava,
17. listopadu 15/2172, 708 33 Ostrava, Czech Republic
[(f)]Sygic

[(a)]vit.ptosek@vsb.cz, [(b)]jiri.sevcik@vsb.cz, [(c)]jan.martinovic@vsb.cz, [(d)]katerina.slaninova@vsb.cz, [(e)]lukas.rapant@vsb.cz, [(f)]rcmar@sygic.com

ABSTRACT

We have developed an enhanced real time traffic simulator running on High Performance Computing infrastructure for testing an efficiency and usability of a self-adaptive navigation system which implements a traffic flow optimization service coordinated with external client-side navigation applications and heterogeneous traffic data sources collected and fused in an intelligent way. Building blocks of the simulator include a server-side navigation system, Virtual Smart City World, benchmark settings, and a test bed containing industrial Sygic client-side navigation and a simplified simulation of vehicles. The important feature of the simulator is the ability to evaluate the traffic flow control strategy in the Smart City world, both with and without enabled Global View calculation of a traffic network for a given percentage of vehicles connected to the server-side service. The integration of the Sygic navigation to the large-scale traffic simulator allows performing compliance test of real navigation applications to the developed central navigation system.

Keywords: traffic simulator, dynamic routing, HPC, smart city, navigation optimization

1. INTRODUCTION

One of the most significant challenges in a field of dynamic routing algorithms development and testing is to create a stable environment with data sets which it is possible to perform reliable and repeatable experiments on. Client-side navigation implemented as a mobile application by Sygic provides us with floating car data (FCD) which can be well used for our self-adaptive navigation system running on a central and knowledgeable server along with other data sources. This data is crucial for dynamic routing enhancement and the bigger the data is the better and more improved service of higher quality we can offer. However, to validate such dynamic routing, we need to ensure the correctness of the data in the first place. Simulating traffic from a macroscopic perspective turned out to be a convenient way how to repeatedly achieve a controlled traffic situation without massive amounts of data required by microscopic models. Nevertheless, our developed approach is also borrowing some ideas from the microscopic traffic simulation because the basic unit of the simulation is an individual vehicle driving along the chosen route. These vehicles then generate physical quantities like traffic flow or speed for the road network. In general, we consider this mixed approach to be a macroscopic model because these vehicles do not directly interact, and purpose of the model is to monitor the behavior of physical quantities describing the traffic and not behavior of individual vehicles. Our proposals are supported by article written by Harri, Filali and Bonnet (2009). By combining both macro and microscopic approaches we can have the model which does not require much data and is as easily manageable and adaptable as a microscopic simulation.

The traffic simulator represents a move of an imaginary world vehicles based on a real world routing system. The whole simulator and its parts are meant to exploit High Performance Computing (HPC) infrastructure as the process itself is computationally demanding as described by Zehe, Knoll, Cai and Aydt (2015) and in need of many computational resources to work properly in time.

The aim is to reflect current and future traffic situation and not only to find the shortest path while still on the road but also the most efficient one changing accordingly to traffic conditions. Thanks to above mentioned process we are able to give such a result with respect to data, benchmark and visualization. Our another goal was to keep a response time limit for every navigation request irrespective of the number of concurrent requests at all times. We consider 500 ms to be a reasonable threshold.

There are many kinds of existing macroscopic traffic models and simulators. For instance, Abadi, Rajabioun and Ioannou (2015) utilize macroscopic simulation to predict the traffic flow. However, their simulation is closer to the classical macroscopic model because while they simulate individual routes, they utilize only information about traffic flow in conjunction with link-to-link dividing ratio to simulate its changes. The article from Batista, Leclercq and Geroliminis (2017) utilizes the combination of macroscopic and microscopic models to simulate and describe the behaviour of the traffic network. Behaviour of their model is, though, more

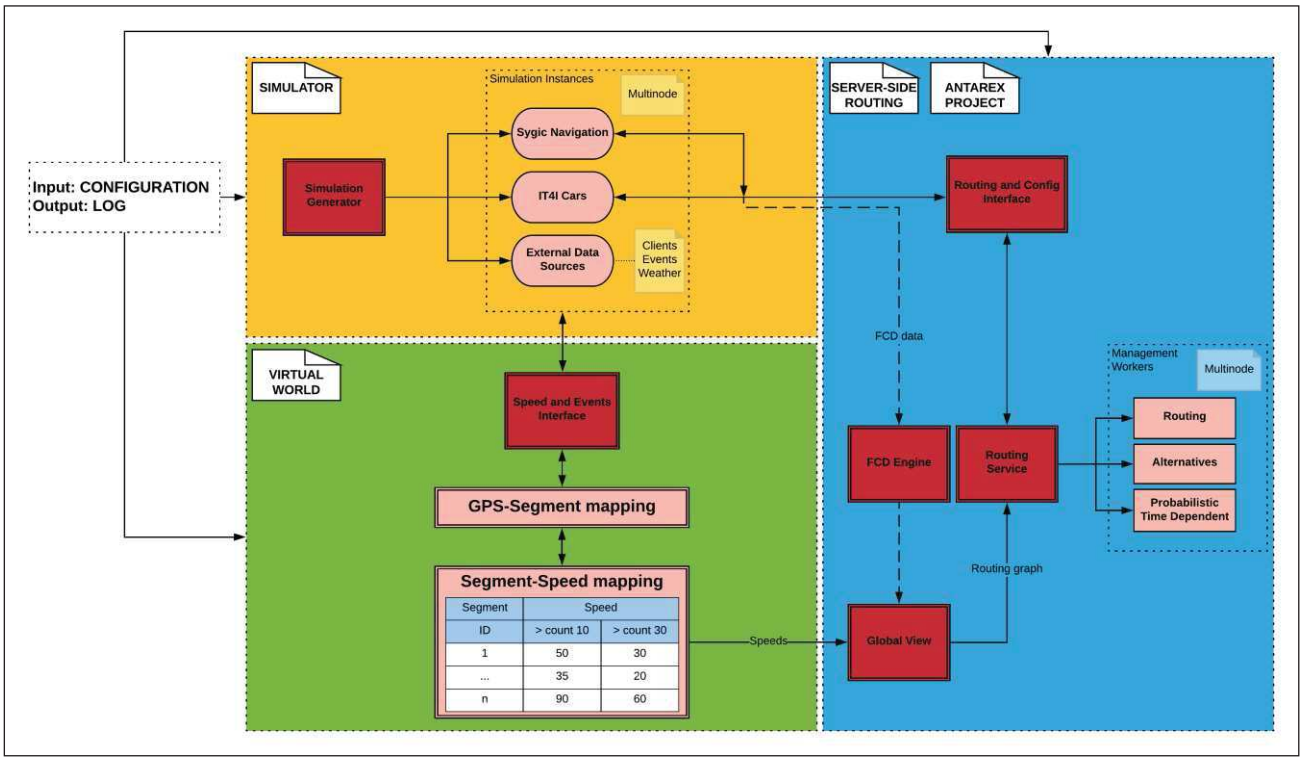


Figure 1: Simulation design

influenced by the macroscopic part of the simulation than the behaviour of individual vehicles influenced by changing the traffic conditions during the simulation. Zhang, Wolshon and Dixit (2015) try to integrate two macroscopic models (Cell Transmission Model (CTM) with the Macroscopic Fundamental Diagram (MFD)) for urban networks in their article. While CTM is a very useful tool for macroscopic modelling, its data requirements are approaching the microscopic model.
There is also a number of both free and paid traffic simulators like VISSIM (Fellendorf and Vortisch, 2010), SUMO (Behrisch, Bieker, Erdmann and Krajzewicz, 2011), TRANSIMS (Smith, Beckman, Anson, Nagel and Williams, 1995) or Matsim (Horni, Nagel and Axhausen, 2016). Generally, these products usually require much more data for their proper functioning because they are mainly based on microscopic approaches. Also, their potential application on an HPC infrastructure is questionable because of their existing implementation limits. Therefore, for our application, creating our own traffic simulator is more preferable.

2. TRAFFIC SIMULATOR STRUCTURE

The traffic simulator consists of several mutually cooperating parts that can be imagined as modules. The proposed design of the simulation model is divided into three main groups pictured and coloured in Figure 1:

1. Simulator – yellow,
2. Virtual World – green,
3. Server-side Routing – blue.

These groups use interfaces to communicate together and those are further explained in Section 3. They also have in common a general configuration file as an input used across all the parts (described in Section 4.1) and a united logging environment for the entire output.

2.1. Simulator

The Simulator part is responsible for initializing and running simulation modules that contribute to the Virtual World and thus affect the Global View. The initialization strongly depends on simulation inputs that can be generated prior to the simulation optionally. Every simulation module subsists up to hundreds of simulated instances running in threads and to exploit HPC infrastructure better, every module can simultaneously run on numerous cluster computing nodes in parallel. Simulation instances are also able to directly obtain itineraries from the Routing service and actual road speed from the Virtual World.

Every simulation experiment we launch must be accompanied by its description. This helps us to reproduce experiments, tell them apart knowing immediately what the aim of the experiment was supposed to be, and mainly to keep them deterministic. Because simulation consists of different modules, each one of them has its own setting file representing simulation instances. Each entry of a concrete setting file then represents a single instance of a given simulation module giving us total number of simulation instances per module.
In the case of vehicle module description, we distinguish between Sygic and IT4I cars and especially keep the

track of their individual travelling intensions. These are recognized by origins and destinations, start times, frequency of logging, requests for a current segment speed from the Virtual Smart City World (Section 2.2), and routing request to the Server-side Navigation System (Section 2.3).

Even if we do not touch the vehicle setting, we can still adjust another simulation results seriously by changing the initial state of the Virtual World describing road network conditions and a traffic status. Every single segment is defined by a set of thresholds representing number of concurrent vehicles and their matching speeds. That way we are able to decide how many vehicles on the same way would imply a traffic jam, or when they should slow down or speed up.

2.1.1. Generator

As we need to modify the whole simulation run for various experiments repeatedly, we often need to adjust specified simulation settings. This task can be done manually by a user or automatically by a setting generator which creates setting files if those do not exist. The generator uses the general configuration file and runs in two modes – lightweight one for testing purposes or another one considering simulation aspects.
With both modes, the segment setting resolves how the initial state of the Virtual World would be generated and the vehicle setting determines the initial state of every single instance of a vehicle.

For example, if we choose generating vehicle setting for the testing mode, we need to declare origin-destination positions and those will be used for all the vehicles, basically creating desired number of vehicle instance copies. On the other hand, if we want simulated vehicles to vary, we can use our ordered routing graph to generate a different route for each vehicle in a deterministic manner as the ordering by a unique identifier stays the same. It is also possible to postpone start time of a vehicle and its round trip individually. Vehicles may use the Global View more or less often or not at all depending on the configuration file.

We can also set a length of a vehicle, which will take effect in generating more realistic routing segment speed relations. Due to the fact that each routing segment has its known length, we can calculate how many vehicles would fit on such segment. This gives us an idea of concurrent vehicle thresholds. Connecting this information with the known maximal allowed speed of a specific segment extracted from OpenStreetMap (Ptošek and Slaninová, 2018) road classification (highway, motorway, tertiary, etc.) we are able to set corresponding speeds for these thresholds. We are aware of a model like this being considerably simplified, but sufficient enough for our deterministic testing purposes.

In even more simplified testing case, we have omitted any of this information and let the speed table to be generated with the same thresholds and speed values for all the segments.

2.1.2. Sygic Navigation System

To enhance the Global View's data from the Virtual World by results coming from the real world, it is possible to use data from the Sygic mobile application as an additional input for FCD Engine producing an output.

2.1.3. IT4I Cars Simulation

The IT4I car (also producing FCD) is a basis of the simulation as every instance can interact with the Virtual World and the FCD Engine, both contributing to the Global View. HPC environment enables us to launch thousands of virtual cars simultaneously. This is very important as it allows us to get better and faster results in the case of urgent need of alternative routes to avoid the creation of higher traffic load on some roads.

The logic behind the simulation of a moving IT4I car is explained in Section 4.2.

2.1.4. External Data Sources

External Data Sources (EDS) are a part of simulation instances and speak for any (third party) modules, such as clients for weather, traffic events, or yet another navigations, which are able to use our exposed interfaces (Section 3). We generally expect fewer EDS in comparison with instances of simulated moving vehicles. Unlike vehicles, EDS do not need to directly interact via routing interfaces.

2.2. Virtual Smart City World

As our simulation experiments rely mainly on dynamic routing, it is crucial to keep track of actual road network situation once it is generated (explained in Section 2.1.1) and initialized. The update of the Virtual World is based on many requests of running simulation instances that can indirectly (via interfaces) start to modify number of concurrent vehicles on segments and even segment speeds alone in the case of traffic events. How often is the Virtual World updated depends on individual setting of every simulation instance.
In return, all vehicles are moving along their itinerary according to the speed they retrieve from the Virtual World. In response to the updated Virtual World, they can actually get a different itinerary, because the Virtual World plays a part in the Server-side routing graph update as it is more or less reflected in the Global View.

Although the Virtual World can be updated in milliseconds, the update period of the Global View is set from the configuration. After that period, a snapshot of actual Virtual World changes is transformed into HDF5 (Folk, Heber, Koziol, Pourmal and Robinson, 2011) format and stored in a real time. This means that vehicle's speed on a certain segment can change immediately, but its itinerary would change no sooner than after the Global View update even though the vehicle is forced to request new route more often.

Updating the routing graph for speed weight of segments gives us a testing and validation tool for our dynamic routing environment as shown in Sections 5.1 and 5.2.

2.2.1. GPS-Segment Mapping

Simulation instances can use GPS coordinates to determine their positions, however our routing graph is derived from geometries representing edges (segments in our Virtual World). We have decided to use library called SpatiaLite (Casagrande, Cavallini, Frigeri, Furieri, Marchesini and Neteler, 2014) that extends SQLite for spatial queries and helps us with translation between segments and coordinates.

2.2.2. Segment-Speed Mapping

To update our Virtual World correctly, we need to know a current position in the case of an event (Section 3.2.2) and both previous and current position of every moving instance, like vehicle (Section 2.1.3). After mapping these positions to segments (if there is a need), it is possible to change speed value based on thresholds (shown in Table 1).

The following simplified pseudocode describes obtaining speed value based on an instance of a vehicle and its visited segments. If the vehicle moves between two segments, the counts of concurrent vehicles on a given segments should change. Actual speed value of a current segment is returned even if the segment has not changed from the previous one, because meanwhile another vehicles could have entered or left from this segment and therefore affected thresholds. In spite of the multimode and multithread simulation process, it is inevitable to use locks.

```
Structure segmentSpeedTable = InitializeSpeedTable();

UpdateVirtualWorld(previousSegment, currentSegment){
Speed speed;

Lock(segmentSpeedTable);

If(previousSegment <> currentSegment){
      segmentSpeedTable[previousSegment]-=1;//decrease
      segmentSpeedTable[currentSegment]+=1;//increase
}

//get count of concurrent vehicles for actual segment
var count = segmentSpeedTable[currentSegment];
//get speed matching actual segment's threshold
speed = segmentSpeedTable[currentSegment][count];

Unlock(segmentSpeedTable);

Return speed;
}
```

Algorithm 1: Virtual World Update Pseudocode

Because a behavior of every simulated vehicle at a certain point of a time is highly dependent on its actual segment and speed, we came up with a bound system based on a relation of the two.

Let speed value (sv) expressed in km h^{-1} for a segment (S) with ID (n) be set to number (v) in the case of count of concurrent vehicles (ccv) being greater or equal to number (k). The expression can be written in the following form

$$S_n = \begin{cases} ccv \geq k_1, & sv = v_1 \\ ccv \geq k_2, & sv = v_2 \\ ccv \geq k_3, & sv = v_3 \end{cases}$$

The examples of a segment-speed relations based on thresholds (bounds) are explained in Table 1 and expressed as

S_n=<ccv_min_n, ccv_max_n):sv_n and therefore for ID 1 S_1=<0, 3):90; <3, 10):50; <10, +∞):1 and therefore S_1=0:90, 3:50, 10:1 to only use inclusive bounds

Table 1: Segment Speeds

Segment ID	Speed for Bounded Intervals of Concurrent Vehicles		
	Threshold 1	Threshold 2	Threshold 3
1	<0,3):90	<3,10):50	<10, +∞):1
2	<0,3):50	<3,10):30	<10, +∞):1
3	<0,10):130	<10,30):90	<30, +∞):1

The default threshold count is three (as used for a demonstration), but can be changed dynamically. The only rule is that regardless the number of bounds, they should always cover the interval of <0, +∞). Then we are able to check the actual speed for the currently highest applicable bound. In the case of nine vehicles, that would be 50 km h^{-1} from the second threshold on S_1, but 30 km h^{-1} for the same number of vehicles and threshold on S_2, and 130 km h^{-1} on S_3 from the first threshold. This is applied and shown in the Algorithm 1 where *segmentSpeedTable[S_n][ccv]* is referring to a formally expressed $sv_{n,bound}$=S_n:ccv giving the actual speed for the specific segment bound in respect of a current number of vehicles on the specific segment, therefore S_1:9=$50_{1,2}$ and analogically S_2:9=$30_{2,2}$ and S_3:9=$130_{3,1}$.

The more vehicles, the higher bound and the lesser speed. It is important to mention, that the highest bound tends to be set at least to 1 km h^{-1} to prevent vehicles belonging to affected segment to get stuck there forever as they would be unable to leave with a zero speed. But as vehicles are slowly leaving a segment with speed of 1 km h^{-1}, the segment can eventually get to lower bound and thus higher speed. In the case of S_1, the speed would increase back to 50 km h^{-1} and 90 km h^{-1} afterwards.

2.3. Server-side Navigation System

The server-side routing optimized by ANTAREX tools – project use case II, Silvano, C. et al. (2016) is the core of the Smart City navigation that is designed to handle a significant number of routing requests and processes them in parallel within low-level computing workers running on a heterogeneous HPC cluster. This self-adaptive navigation system is largely used within

simulation instances and covers every part related to the routing. It can be enriched by the Global View of the traffic network.

2.3.1. Routing Service

The service is based on a management system providing scheduling and allocation of computing resources for routing workers as well as their communication with service clients. These workers can run in multinode mode in order to reach sufficient request throughput.

At this moment the routing service supports several routing algorithms on a custom-generated routing graph. The routing algorithm option is part of a simulation configuration. By default, to find the shortest path our implementation of Dijkstra routing algorithm is used. We have chosen this algorithm for all of our simulation testing experiments to omit any need of a heuristic function which served our performance purposes well.

2.3.2. Global View

As every routing algorithm needs a network to find a path between nodes, we advance our routing graph edges with additional information and metrics to balance it accordingly. Changing weights of edges (costs) gives us an opportunity of the dynamic routing. The Global View can be calculated on the basis of the Virtual World data and the FCD Engine output.

2.3.3. FCD Engine

If we wanted to connect the Virtual World with the real one, we could connect another module into the scheme. Our FCD Engine manages to process data from Sygic navigations outside the simulation and offers a real time traffic monitoring that could serve for the initial Global View state instead of generated Virtual World.

Processing FCD from Sygic navigation inside the simulation could help us comprehend what minimal percentage of FCD coverage (in comparison with IT4I cars) is needed to reliably monitor the traffic situation in the case of collaborative routing usage.

3. TRAFFIC SIMULATION INTERFACES

Interfaces serve to connect component modules described in Section 2 together as well as they help external parts to communicate with the server-side navigation services. They are based on HTTP and TCP protocols and are meant to serve two fundamental roles – simulation run and validations of aforehand mentioned services.

3.1. Server-side Navigation Interface

This interface is crucial for all the simulations, but has zero dependency on the simulation itself, because routing services can be used for a real world real-time navigations outside the simulation process as well.

3.2. Simulation Interface

The following interfaces were specially built for the simulation purposes and are highly dependent on the simulation itself.

3.2.1. Configuration

Since we can generate configurations for EDS instances described in Section 2.1.4, this interface can be used to deliver particular configurations before the simulation starts. For example, if Sygic navigation application wanted to use our pre-generated scenarios, they could ask for every single instance's setting from the application. This guarantees simulation consistency as we are in charge of their origin and destination positions, number of instances, request limits and many more. This approach takes the advantage of avoiding hard-coded simulation scenarios and being able to change them relatively easily and quickly.

3.2.2. Events

By events we generally mean road closures, accidents or lane restrictions formed by their geozones and optional speeds that affect the Global View directly. Events can

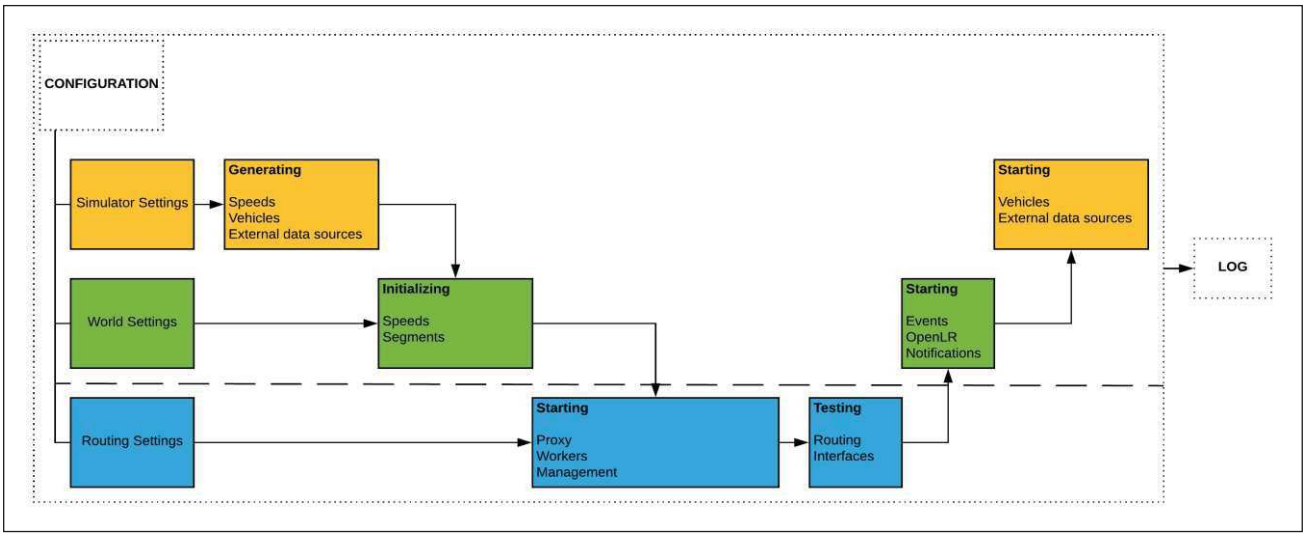


Figure 2: Simulation run

be raised interactively from within a map as shown in Figure 10 or by planned event instances based on a simulation setting.

3.2.3. Speed

Because our objective is not to develop a multi-agent system where simulation instances interact with each other, this interface was introduced to map position of a vehicle onto our routing graph returning a segment speed relation in response to the Virtual World. This way, each vehicle is assured to know how fast it can move without knowing about other vehicles it shares the road with and vice versa. It also delegates a vehicle's understanding of our routing road network topology and corresponding threshold speed limits to centralized Virtual World.

Speed interface works both ways, it returns value based on a current road type and number of concurrent vehicles and updates the Global View metrics at the same time to avoid cumulating, as can be seen in Figure 11.

4. TRAFFIC SIMULATION PROCESS

The whole simulation process is based on a communication between the connected modules described in Section 2. After preparing essential environment (endpoints, datasets, services, notifications, etc.) and generating simulation settings, simulation instances are initialized, and vehicles start to move. The simulation runs until the last vehicle reaches its destination.

Figure 2 shows the model described in Figure 1 from the runtime execution perspective.

4.1. Execution

The execution is done automatically and completely in the HPC environment and is handled by one main script. This master script is linked to all the modules' initial settings, configurations and auxiliary scripts as well as their separate process runtime logging. With this design we can easily launch self-contained experiments repeatedly to reproduce the results or to see how they change in time with different properties.

4.2. Vehicle moving behaviour

The moving strategy of a vehicle object is based on its itinerary and a speed determined by the road network condition at the time, e.g., free flow speed, heavy traffic, traffic jam or even a closure-causing obstacle. The itinerary may vary as the Global View keeps updating during a simulation. The speed may differ from the same reason, but based on the Virtual World.

The vehicle (repeatedly) obtains its route itinerary from the server-side navigation system as seen in Figure 3, and its (in this case constant) speed from the Virtual World. The blue line represents a current itinerary and the purple line stands for already visited segments from either current or past itineraries (they can differ based on the Global View). The most recent position of the vehicle is represented by the blue dot, whereas the purple dots express tracks of history positions of the same vehicle.

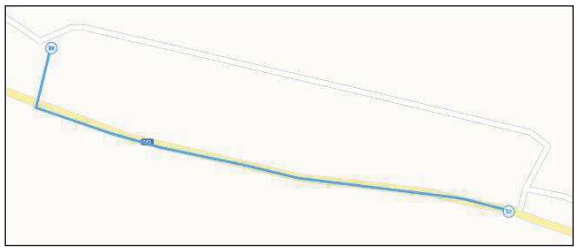


Figure 3: Vehicle route start

Unlike the speed from the Virtual World, the itinerary is a subject to change with the Global View enabled only (described in Section 2.3.2). It is important to always be aware of the current road segment the vehicle is on because its travelled distance depends on the actual speed of the current segment for a given vehicle count. Since there is only one vehicle situated on the segment, as pictured, the speed does not change and neither do the distances between its positions.

The vehicle does not leave its current segment till the total distance travelled by the vehicle is shorter than the distance from the start of the route to the end of a current segment as pictured in Figures 4 and 5.

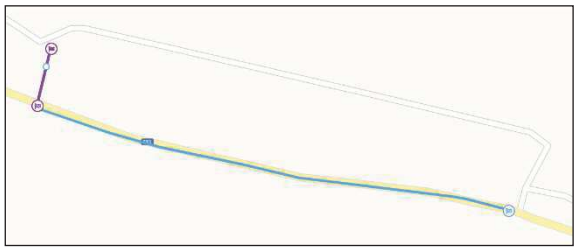


Figure 4: Vehicle route move on a segment

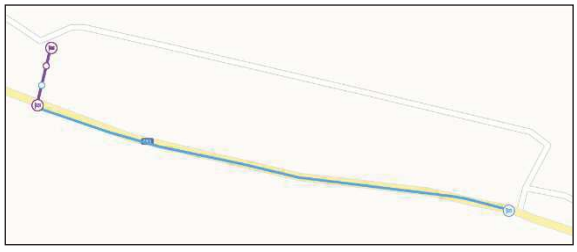


Figure 5: Vehicle route move on the same segment

While the vehicle is travelling along the road segment, its speed may be affected by a volume of concurrent vehicles on the same segment at the same time. As the vehicle instance continues, it may move onto a following road segment (as seen in Figure 6) from its (updated) itinerary (that can vary from the previous one) or not – regarding the actual segment speed and remaining segment distance.

This motion process continues until a vehicle reaches its destination regardless an alternative route (no following segment – graph edge exists) as is depicted in Figure 7.

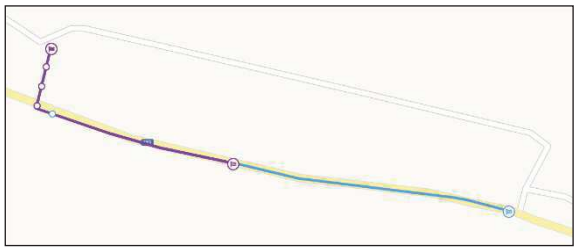


Figure 6: Vehicle route extended by following segment

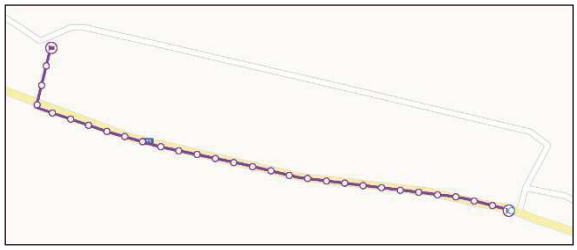


Figure 7: Vehicle route finished

Moving a certain instance of a vehicle on segments of a routing graph is relied upon an invariant that can be described in do-while block by the pseudocode in Algorithm 2.

```
Time time = GetTimeNow();
Position actualPosition = GetGpsFromSettings();

MOVE:
do {
    WaitForNextMove(waitingPeriod);

    Route route = GetRoute(vehicleId); //Figure 3
    Speed speed = GetSpeed (vehicleId, actualPosition);
    Distance distance = speed * (GetTimeNow() - lastTime);

    vehicleDistance += distance;
    while (vehicleDistance >= routeDistance) {
        If (routeItineraryNotEmpty) {
            segment = DequeueItinerary();
            routeDistance += segmentLength; //Figure 6
        } else {
            Finished = true; //Figure 7
            break;
        }
    }

    //get ratio value between 0 (start) and 1 (end) inclusive
    segmentRatio =
    1 – (routeDistance – vehicleDistance) / segmentLength;

    actualPosition = GetGpsFromSegmentRatio(); //Figure 4
    Log(route, speed, distance,  actualPosition);

    lastTime = GetTimeNow();
} while (NotFinished);

If(RoundTrip){
    Finished = false;
    vehicleDistance = routeDistance = 0;
    Swap(GetGpsFromSettings(), actualPosition);
    GoTo MOVE;
}
```

Algorithm 2: Real time Position Change Pseudocode

4.3. Routing adaptation

The adaptation is based on a cost of a given route and a chosen routing algorithm which calculates a total cost. The total cost of the route is a summary of costs of all the segments belonging to a given route – itinerary cost. This means that few segments of a route can outweigh the rest and vice versa. When the cost changes significantly, it may happen that from the point of the Global View, another route becomes more convenient than the existing one (as is illustrated in Figures 8, 9 and 10).
The weight of an edge of a routing graph between two nodes can be in our case:

1. Static – distance, which is not subject to change,
2. Dynamic – transition time based on a speed being calculated just-in-time from the Global View according to a current traffic situation.

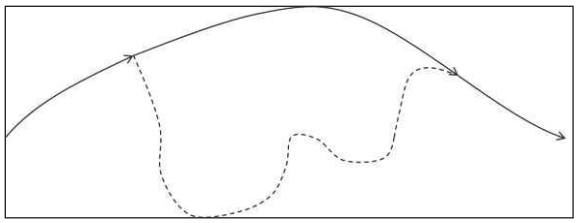


Figure 8: Original route based on distance

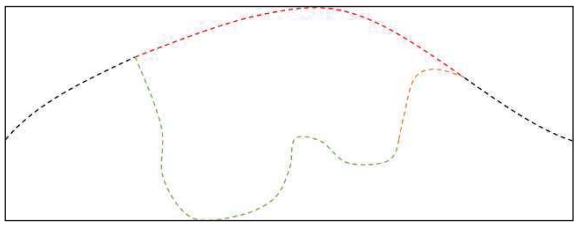


Figure 9: Segment speed evaluation

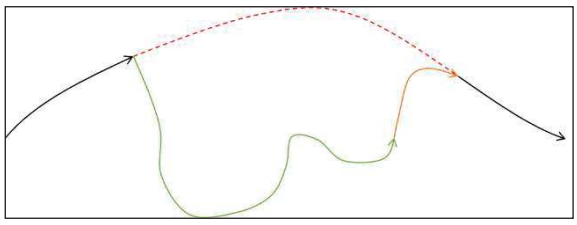


Figure 10: Alternative route

5. SIMULATION SCENARIOS

The strategy and use cases of a simulation are created by an initial setting. It is also possible to use a map visualization to interact with the still running simulation itself. We have picked two use cases for a demonstration – roadblocks in Vienna, Austria city centre and a wilful cumulative traffic jam in Ostrava city, Czech Republic.

5.1. Forcing optimization with blockages

Figure 11 represents a scenario where we generated multiple vehicle instances and assigned them unique pair of nodes belonging to a routing graph, such that the nodes meet a criterion of having at least one crucial edge

between them. In a real world, the nodes represent origin and destination points and a set of edges creates a routing path. From a traffic experience, bridges are examples of critical edges, so we aimed at destinations across a river. The bridge areas being a part of their way were then flagged as traffic jams gradually in such manner so that their way was no longer considered as the optimal one. Affecting an actual road situation with the accidents shows vehicles rerouting eventually to avoid a rush.

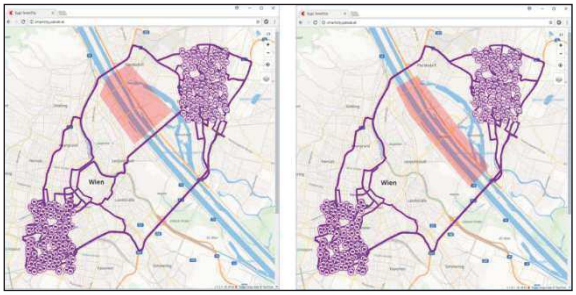


Figure 11: Static traffic jam

5.2. Forcing optimization with Global View

Our aim is to be able to dynamically cope with traffic jams accordingly – in our case to change the route of a vehicle based on a community contributing to the traffic situation as a whole. This means the final route is both individually the most beneficial as well as efficient from a global point of view.

Figure 12 demonstrates numerous vehicle instances driving from and to the very same start and end points. In contrast with Figure 11 we decided not to influence the road network with any blockages and had vehicles to form the traffic jam themselves as they are moving.

The route optimal for a single vehicle is shown on the left side and was used for all the vehicles, which leads to an overall slowdown caused by a traffic jam. To mitigate the slow-down and compare our results, we also run the same simulation, but with the Global View enabled at this time. As can be seen on the right side, there are several routes pictured apart from the original one. Distributing the vehicles into diverse routes helped the road network to balance and the last vehicle to arrive significantly sooner via the more fluent way as described in Section 6.

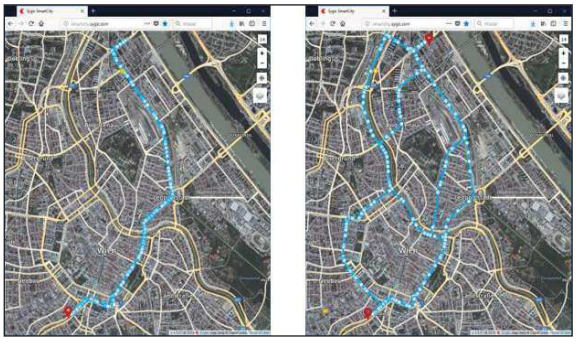


Figure 12: Dynamic traffic jam handling

6. RESULTS

We have chosen three different origin locations and a common destination (route lengths 11.5, 6.5 and 5.9 kilometres) such that they share final parts of their routes as captured in Figure 13 that represents every route without applying updates from the Global View. In this case, only one route exists for every origin-destination.

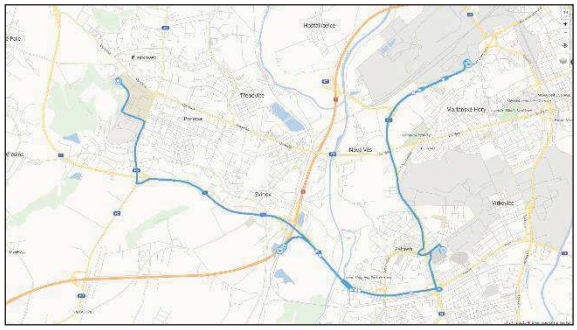


Figure 13: Static routes during a simulation

We have applied lightweight mode for generating testing Virtual World (described in Sections 2.1.1 and 2.2) and launched 200 vehicles for each starting position, 600 in total. Then, we have run the simulation with both Global View disabled and enabled several times to verify that our results match our settings repeatedly in both cases. Figures 14 and 15 show an obvious navigation adaptation where, unlike in Figure 13, vehicles started using by degrees more than three original routes obtained for three origin-destination pairs depending on another vehicle instances already sharing the same route at the time.

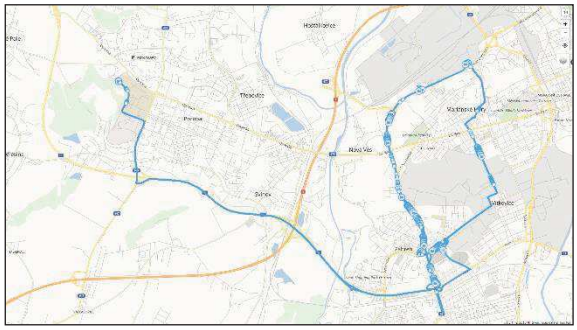


Figure 14: Alternative routes during a simulation

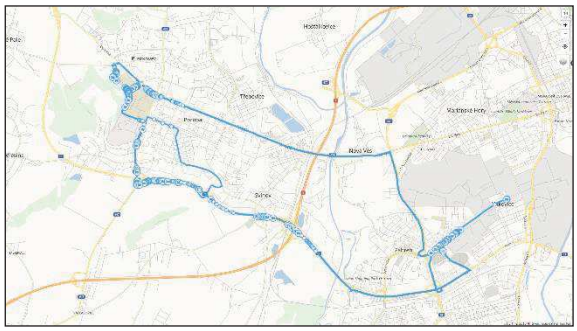


Figure 15: Alternative routes during a simulation II

Our first step was to prove our model to be deterministic in the meaning that the same simulation gives us very similar results regularly with little to no deviation caused by occasional request/response delays. When this has been achieved, we started with measuring the impact of the Global View usage being assured that the difference would not come from nondeterministic behaviour. These results based on the Global View only are showed below in Table 2, the better values are highlighted in green.

Table 2: Driving Times in minutes and seconds

Global View	600 Vehicles (3x200)			
	Average	Median	Minimum	Maximum
Off	17:26	14:28	04:36	39:49
On	14:13	14:28	04:27	30:52

We were able to observe ≈22.5% speed-up in the case of the maximal duration of a vehicle run with the Global View enabled. In our testing simulation scenario, this duration represents the very last car arriving to its destination.

Our final results helped us to prove that from a globally collaborative perspective, utilization of the Global View based self-adaptive routing was timewise more efficient than in the event of a static one causing traffic jams.

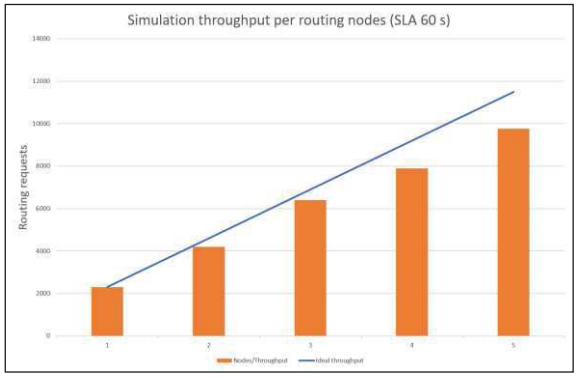
7. CONCLUSION AND FUTURE WORK

We have presented the real time traffic simulator developed for running on HPC infrastructure for testing an efficiency and usability of the self-adaptive navigation system. Our first proposal was to meet a response time limit for every navigation request under 500 ms, which we are able to achieve at the moment. During our development and after the testing phase, we have discovered that the proposal can be different for our simulation case.

We came up with the fact that during a simulation, a vehicle does not need to receive a response for an updated route in a strictly short time requiring more computing resources, especially when its amount for the simulation with thousands of cars could be very highly computationally demanding and not necessary for our case. Also, as mentioned in Section 2.2, the routing graph changes much less often than the Virtual World that vehicles contribute to; therefore, routing services give the same results during a short interval. And lastly, we have learnt that a vehicle is able to drive even without an updated route for some time as there is usually no need to obtain a completely new route every half a minute.

Thus, our proposal has changed to satisfy a condition where every vehicle gets a response with a service-level agreement (SLA) much higher than the previous request limit time. As pictured in Graph 1, for the selected SLA 60 seconds and for 5,000 simultaneous vehicles, we need to allocate approximately 43 cores, which corresponds to 3 compute nodes from the Anselm cluster (2x8 Intel Sandy Bridge cores @2.4 GHz, 64 GB RAM per node) to reach that SLA.

From the self-adaptive routing system perspective, we have achieved promising navigation time improvement in the form of global traveling time speed-up.



Graph 1: Simulation throughput – SLA 60 s

Because our simulation process is data-driven, we plan to scale up our simulations and extend various models with EDS, better speed profiles and a related data fusion.

In the future, we would like to continue our work with the second proposal and experiment with SLA levels sufficiency. Another goal is to examine the minimal percentage coverage of the external navigation (Figure 16) vehicles needed for reliable traffic monitoring in respect of simulated vehicle instances yet managed.

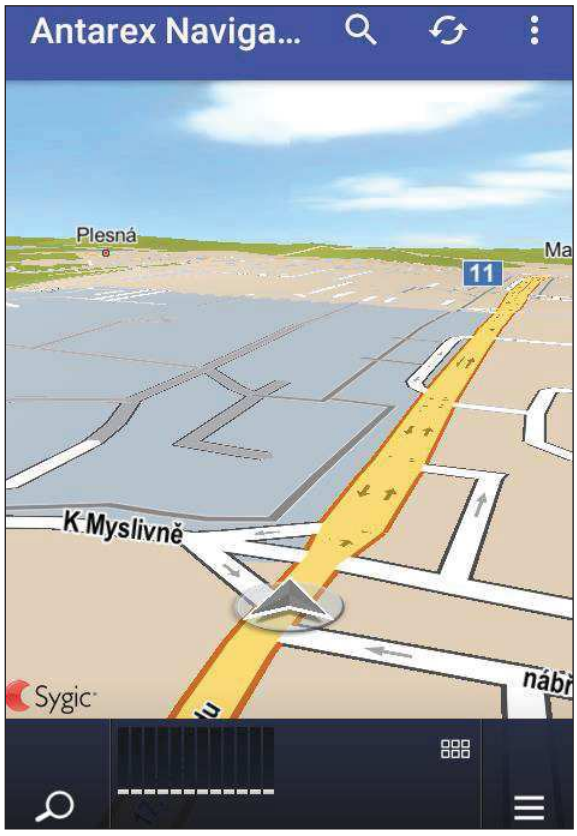


Figure 16: ANTAREX mobile navigation by Sygic

ACKNOWLEDGEMENT

This work has been partially funded by ANTAREX, a project supported by the EU H2020 FET-HPC program under grant 671623, by The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II) project 'IT4Innovations excellence in science - LQ1602', as well as by the SGC grant No. SP2018/173 "Dynamic systems problems and their implementation on HPC", VŠB - Technical University of Ostrava, Czech Republic and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project ”IT4Innovations National Supercomputing Center – LM2015070”.

REFERENCES

Harri, J., Filali, F. and C. Bonnet, 2009, Mobility models for vehicular ad hoc networks: a survey and taxonomy, Communications Surveys & Tutorials, Volume 11, Part 4, Pages 19-41

Zehe, D., Knoll, A., Cai, W. and Aydt H., 2015, Simulation Modelling Practice and Theory, Volume 58, Part 2, Pages 157-171

Abadi, A., Rajabioun, T. and Ioannou P. A., 2015, Traffic Flow Prediction for Road Transportation Networks with Limited Traffic Data, Transactions on Intelligent Transportation Systems, Volume 16, Part 2, Pages 653-662

Batista, S., Leclercq, L. and Geroliminis N., 2017, Trip lengths and the macroscopic traffic simulation: an interface between the microscopic and macroscopic networks. hEART2017 - 6th symposium of the European Association for Research in Transportation, HAIFA, France. hEART2016 - 6th symposium arranged by European Association for Research in Transportation, Page 5

Zhang, Z., Wolshon, B. and Dixit V.V., 2015, Integration of a cell transmission model and macroscopic fundamental diagram: Network aggregation for dynamic traffic models, Transportation Research Part C: Emerging Technologies, Volume 55, Pages 298-309

Fellendorf, M. and Vortisch P., 2010, Microscopic Traffic Flow Simulator VISSIM, Fundamentals of Traffic Simulation. International Series in Operations Research & Management Science, vol 145. Springer, New York

Behrisch, M., Bieker, L., Erdmann, J. and Krajzewicz, D., 2011, SUMO – Simulation of Urban MObility An Overview, SIMUL 2011, The Third International Conference on Advances in System Simulation, pp. 63-68

Smith, L., Beckman, R., Anson, D., Nagel, K. and Williams, M., 1995, TRANSIMS: TRansportation ANalysis and SIMulation System, 5. National transportation planning methods applications conference

Horni, A., Nagel, K. and Axhausen, K.W., 2016, Introducing MATSim, The Multi-Agent Transport Simulation MATSim, Ubiquity Press

Ptošek, V. and Slaninová K., 2018, Multinode Approach for Map Data Processing, 5th International Doctoral Symposium on Applied Computation and Security Systems (ACSS)

Folk, M., Heber, G., Koziol, Q., Pourmal, E. and Robinson, D., 2011, An overview of the HDF5 technology suite and its applications, Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, p.36-47, March 25-25, Uppsala, Sweden

Casagrande, L., Cavallini, P., Frigeri, A., Furieri, A., Marchesini, I. and Neteler, M. G., 2014, GIS Open Source: GRASS GIS, Quantum GIS and SpatiaLite

Silvano, C. et al., 2016, Autotuning and adaptivity approach for energy efficient Exascale HPC systems: The ANTAREX approach, 2016, Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, pp. 708-713