University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Distributed Traffic Simulation

State of the Art and Future Research

Tomáš Potužák

Supervisor: Pavel Herout

Technical Report No. DCSE/TR-2008-02
March, 2007

Distribution: public

Abstract

The road traffic density on highways and especially in cities is permanently increasing. An important tool, which can be used for analysis and controlling of traffic networks, is the computer simulation. However, in order to be performed in suitable time, a detailed simulation of a large traffic network (e.g. whole city) requires a great amount of computational power. A way how to obtain sufficient power for the simulation is to adapt it for distributed computing environment.

In this work, the common issues of distributed traffic simulation are discussed. Because on of the main bottlenecks of every distributed application is the communication among its particular processes, we focus mainly on inter-process communication. Besides the description of commonly used approaches, we present also several modifications to the communication protocol, which cause significant reduction of inter-process communication. The resulting communication protocol has been tested on the Java Urban Traffic Simulator (JUTS), which is being developed at our department. However, in final stage, our communication protocol will be applicable for general time-stepped simulations. The results of the tests, which have been so far performed, are also presented in this work.

Keywords: distributed traffic simulation, inter-process communication, process synchronization, simulation decomposition, communication reduction, JUTS, urban traffic

Contents

1 Introduction

The road traffic density on highways and especially in cities is permanently increasing. An important tool for the managing of the road traffic is the computer simulation. It helps to analyze existing traffic networks and to improve their performance. It is also possible to predict the behavior of the traffic by a traffic lane closure or by a traffic accident. Simulation of road traffic is also useful throughout the designing of new traffic structures, where they help to predict the future behavior of the new structures and their integration into the current traffic network.

In order to model the real traffic situations as accurate as possible, the simulation must be very detailed. Moreover, in many cases, multiple executions of the simulation are necessary to guarantee the required fidelity of statistical results. However, although the computing power of the computers is increasing day by day, it is still not possible to perform a detailed simulation of a large traffic network on a single-processor computer in suitable time. A way how to speedup the simulation is to adapt it for distributed computing environment. In that case, the combined power of more single-processor computers connected via a computer network is utilized. In that case, the simulation is divided into number of processes, which are then performed on particular computers (nodes) of the distributed computer.

One of the main bottlenecks of any distributed application is the communication among its particular processes. The communication is necessary for synchronization of the simulation and for exchange of necessary data between the particular processes. Because this inter-process communication is relatively slow, it influences negatively the resulting performance of the distributed simulation. Hence, it is desirable to reduce it to the necessary minimum.

In this work, the basic issues of distributed traffic simulation and commonly used solutions of these issues are described. Although all problems are discussed quite closely, we focus mainly on inter-process communication, because the design of an efficient communication protocol is the main aim of our work. Several adjustments, which cause significant reduction of inter-process communication, has been already invented and tested. The tests have been performed on the Java Urban Traffic Simulator (JUTS), which is being developed at our department. However, the resulting communication protocol will be applicable for general distributed time-stepped simulations. The results of the tests performed so far are also presented in this work. Moreover, several other modifications to the communication protocol have been proposed. They are also described in this work.

In chapter 2, the basics of the general simulation are briefly described. The specifics of the simulation of the road traffic along with the description of the JUTS system are introduced in chapter 3. Chapter 4 describes the main issues of the general distributed simulation. In chapter 5, the specifics of the distributed traffic simulation are described in detail. Chapter 6 is focused on reduction of inter-process communication and the achieved results are presented in chapter 7. Chapter 8 gives a brief survey of the possible modifications to the communication protocol, which are the main aim of our future research. The work is concluded in chapter 9.

2 Simulation Basics

First, we mention the basic principles of a computer simulation. A computer simulation is some sort of computation that models the behavior of a real or hypothetical system over time. The simulation is performed on a model of the system. The model must contain all features, which are important for objective of examination, but there are always some simplifications compared to the original system. The simplifications are necessary, because we have only limited pool of resources (like time, computer power etc.) to commit the simulation. On the other hand, the simulations must provide "the same" results as the original system (with acceptable statistical variation), so, they can be used to make some useful conjectures regarding the model, which are also applicable to the system itself [1].

The computer simulations can be classified from several points of view. The most important divisions are according to the purpose of the simulation and according to the way the simulation time is elapsed.

2.1 Purpose of the Simulation

According to the purpose of the simulation, the simulations can be classified as *analytical ones* and *virtual environments*.

2.1.1 Analytical Simulation

The analytical simulations are used to model existing or designed system from the real world. Their goal is to analyze the behavior of the system at current conditions or predict its behavior at changed conditions. In case of existing systems, the results can be used to improve the performance of the system. If some new system is designed, the simulation is a cheap way to examine whether the system would perform required functions. So, it can be decided whether the further development should continue or not.

Analytical simulations usually attempt to obtain concrete and precise statistical data from system they are modelling. In some cases, many executions of the simulation are needed to guarantee required fidelity of the statistical results. Thus, analytical simulations often execute as fast as possible. They also often include limited or no interaction with user during execution of the simulation program. So, the users can merely analyze statistical results after the execution is completed. In some cases, the progress of the simulation can be visualized, but the user still plays the role of external observer [2]. In this work, we will deal only with analytical simulations. Hence, they will be simply referred as *simulations*.

2.1.2 Virtual Environments

Besides the analytical simulations, there exist the virtual environments. The virtual environments are simulations, which attempt to create computer-generated worlds, in which human and artificial participants are embedded. The users (humans) directly influence the simulation during its execution. Unlike the analytical simulations, the time must advance approximately at the same rate as the real time in order to ensure that the virtual environment seems to be realistic for human participants. The most common use of these environments is training (e.g. training of military and civil pilots) and fun (e.g. world wide internet games) [2]. We will not consider this type of simulation further.

2.2 Time-Flow Mechanism

The latter classification of the simulations determines, in which way the simulation time is advanced (so-called *time-flow mechanism*). The simulation can be either *continuous* or *discrete*.

In a continuous simulation, its state is changing continuously during its execution. The behavior is often described by set of differential equations, so the simulation state can be computed for any single unit of time. A typical example can be the weather forecasting model [2].

In a discrete simulation, its state changes only at discrete points in simulation time. There are two most common types of discrete simulations – the *time-stepped* and *event-driven* simulations.

2.2.1 Time-Stepped Time-Flow Mechanism

In a time-stepped simulation, the simulation time between the beginning and the end of the execution is subdivided as a sequence of equal-sized time steps. The simulation time is then elapsed from one small interval (e.g. one second) to another. In every step, new values of every state variable (which determine the state of the simulation) are recomputed, although some of them may not be changed. Events in the simulation, which occurs in one time step are often consider as simultaneous and assumed not to have effect on each other [2].

2.2.2 Event-Driven Time-Flow Mechanism

Unlike a time-stepped simulation, an event-driven simulation does not compute new value of every state variable in each time step. The simulation advances in time by interpreting events from the evens list. An event incorporates some actions (determining which state variables should be changed and how) and a time-stamp (determining when the actions should happen). So, the simulation time elapses from one time stamp to another [2].

3 Traffic Simulation

Now, as we discussed the issues of the general simulation, we can focus on the specifics of the simulation of the road traffic.

3.1 Level of Detail

The simulations of the road traffic can be classified according to several aspects. However, the most common division can be performed according to the level of detail (level of vehicles' representation). The traffic simulations can be divided as:

- Macroscopic simulation
- Mesoscopic simulation
- Microscopic simulation

All three types of traffic simulation are briefly described in following sections.

3.1.1 Macroscopic Simulation

The macroscopic simulations deal only with aggregate traffic flows across the streets. The flow is described for example by the mean speed and concentration, no individual vehicles are considered. These models are the oldest and simplest ones [3]. They are widely used and exist in many modifications [4]. Because of their simplicity, they are least computation-consuming. Hence, in most cases, a single-processor computer nowadays is able to provide sufficient power to perform macroscopic simulation in suitable time [5].

3.1.2 Mesoscopic Simulation

The mesoscopic simulations add some characteristics of individual vehicles to the model, but the simulation is computed for groups of vehicles traveling along similar paths. The examples of this approach are gas kinetic models [6] and queuing networks models [7].

3.1.3 Microscopic Simulation

In the microscopic simulation, every individual vehicle is modelled as a single simulation object. These simulated vehicles then drive along the streets, change traffic lanes and their driving directions and interact with each other. So, each simulated vehicle has its own immediate position, speed, acceleration, and so on. The most important models in this area are the car following model [8] and cellular automaton model [9]. A branch of this type of traffic simulation is so-called *nanoscopic* simulation, which in addition involves individual behavior of the drivers. For more details, see [10].

Because the microscopic simulations model the real traffic at high level of detail, they are more useful than macro- or mesoscopic simulations for estimation of detailed traffic characteristics such as delays or queues lengths. For the same reason, the microscopic simulations are also much more computation- and time-consuming. The simulation of large areas (e.g. entire cities) on a single-processor computer often requires unbearable amount of time. Distributed execution is a way how to achieve suitable performance of large-scale microscopic traffic simulations. Possible approaches to this are discussed further in the text.

3.2 Time-Flow Mechanism of Traffic Simulations

Regardless to the level of detail, every type of traffic simulation must use a mechanism for advancing of simulation time. The time-flow mechanisms were discussed in section 2.2. In the field of traffic simulation, the discrete time-stepped model is most common. The simulation time is divided into equal-sized time intervals (time steps), typically one second long. In every time step, the positions of the vehicles and other state variables of a microscopic simulation or the attributes of traffic flow of a macroscopic simulation are computed. After all necessary computations are complete, the simulation proceeds with next time step and so on.

The time-stepped approach is very convenient for the microscopic traffic simulation where every single vehicle is considered. In every time step, every vehicle has known position, speed, acceleration, and so on. The vehicles can interact among each other in every single part of the traffic network, not only at crossroads, but in traffic lanes as well. For example a faster vehicle must slow down if there is a slower vehicle in the way. These interactions are important for the simulation and there can be hundreds of them in every time step. With event-driven approach, there would be a large amount of events that would simulate the vehicle interactions. This would lead to very ineffective execution of the simulation. Consequently, the time-stepped approach is used in most microscopic traffic simulations, for example [11], [12], and [13].

When the simulation is focused on the crossroads and the lane interactions of the vehicles are neglected, the event-driven approach can be easily used. For example, every crossroad would be simulated as single logical process. The arrivals and departures of the vehicles would be represented as events. Every departure event processed in one crossroad would cause an arrival event in a neighbouring crossroad. Although event-driven approach is convenient in this case, the time-stepped approach can also be used, as shown in [14]. The event-driven approach is used in [15].

3.3 Traffic Model in the JUTS System

Now, as we discussed the issues of the general traffic simulation, we can proceed with description of the single-processor version of the JUTS system. The JUTS system is a discrete time-stepped simulator of urban traffic at microscopic level of detail.

Throughout the development, the communication protocol for distributed computing environment, which is described in further chapters of this work, is tested on the JUTS system. However, in final stage, the communication protocol will be applicable for general distributed time-stepped simulation.

3.3.1 Movement of the Vehicles

As has been said, the traffic model in the JUTS system is the microscopic one. So, every single vehicle moving in the road is modelled as an object, which has its own immediate position, direction, speed, and acceleration. For the vehicle movement, the basics of Nagel-Schreckenberger's cellular automata model are used [16]. Because this model was originally constructed only for highways and freeways, several modification were necessary to meet urban traffic simulation requirements. The resulting model will be now briefly described, more precise description can be found in [17], [18], and [19].

The simulated vehicles are moving in the model of real traffic network, which is described in following section. The elements of traffic network, in which the vehicles are moving, are internally divided into equal-sized traffic cells. Each cell can be either empty or occupied by a single vehicle. The length of the cell is set to 2.5 meters. This division is smooth enough for urban traffic conditions. It also enables to distinguish the types of the vehicles. For more realistic simulation, there are six types of vehicles, which differ in the length. These types represent all common known vehicles (see Table 1).

Table 1: Vehicle types and their lengths

Vehicle type	Length [cells]	Length [m]
Motorcycle	1	2.5
Passenger vehicle	2	5.0
Van	3	7.5
Minibus	4	10.0
Bus and truck	5	12.5
Lorry	6	15.0

As can be seen in Table 1, a vehicle can occupy from one to six cells. The whole simulation is discrete. So, in each time step, every vehicle moves from one group of cells to another. It is not possible for the vehicles to be placed somewhere between the cells. Hence, the speed of the vehicle movement must be also discrete and represented in cells per time step (cpts). The speed has to be changed according to the actual road situation in the immediate surrounding of the vehicle. For this purpose, there are four rules for speed modification:

1. Acceleration
2. Deceleration
3. Randomization
4. Move

The first rule says that the vehicles want to accelerate to the maximum speed. The second rule represents the necessary slow down if there is a road-block in the front of the vehicle (e.g. slower vehicle). The third rule represents random adjustment to the final speed, which simulates the natural speed fluctuations caused by a human factor and road conditions. The last rule only shifts the vehicle according to the final value of speed calculated following the first three rules [17]. The final movement of two vehicles of different lengths and velocities is depicted in Fig. 1.

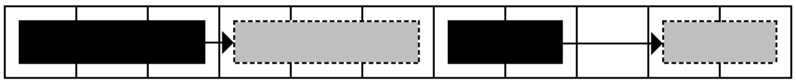

Fig. 1: Movement of two different vehicles

There is one problem with the vehicle movement, which is not obvious in the one-lane scenario depicted in Fig. 1. Sometimes, the trajectory of a vehicle longer than one cell is more complicated than a straight line. In that case, it may not be sure, in which cells the tail pieces of the vehicle should be put. This problem is solved by the *head leading algorithm*. Its main idea is that the tail pieces of the vehicle are shifted over the same way that the head piece was shifted before. The algorithm is described in [17], [18], and [19] in detail.

3.3.2 Traffic Network Structure

As we discussed the movement of the vehicles in the JUTS system, we can continue with the description of the traffic network. There are four types of traffic network elements:

- Road
- Crossroad and roundabout
- Parking
- Generator and terminator

The basic element is the road, on which the most of the simulation is performed. Each road is composed of one or more unidirectional traffic lanes. As has been said in previous section, the traffic lanes are internally divided into traffic cells, in which the simulated vehicles are moving [17].

Other important elements are the crossroads and the roundabouts. These elements interconnect the particular roads and thereby form a complex traffic network. The movement of the vehicles in these elements is quite complicated, but it is based on similar principles like the road movement.

The parking only represents amount of parking places in corresponding parts of the traffic network. The last two important objects are the generator and the terminator. These objects are located at the edges of the simulated area. Their task is to maintain the interaction of the simulated area with its surroundings. The generators create vehicles ingoing to simulated area from surroundings according to the specified stochastic distribution and insert them into the ingoing traffic lanes. The terminators then remove the vehicles that are leaving the simulated part of traffic network from the outgoing traffic lanes [17]. The example of a small traffic network is depicted in Fig. 2.

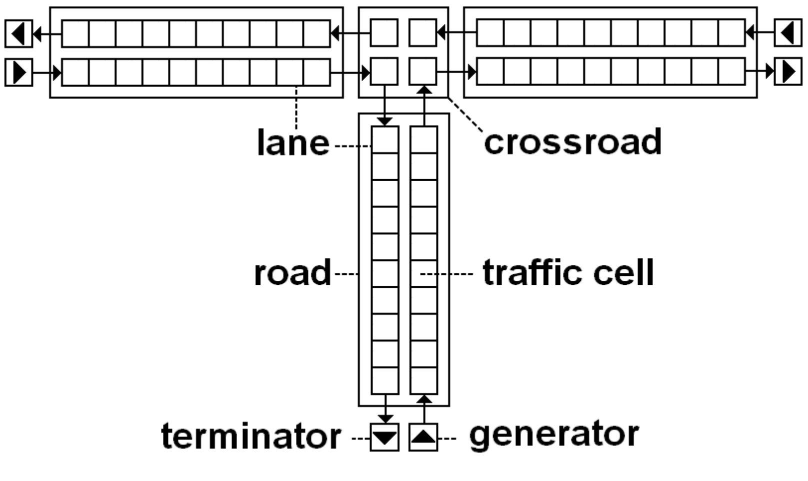

Fig. 2: Example of a small traffic network

3.3.3 Hybrid Traffic Simulation

As mentioned before, the JUTS system is a microscopic traffic simulator. However, a hybrid traffic model, which represents different areas of traffic network in a different level of detail, is presently under construction. This allows us to simulate larger areas at macroscopic level of detail and only small specific areas of interest at microscopic level [20].

In the macro-JUTS model, there are also the same elements of traffic network as in the micro-JUTS model (road, crossroad, generator, and terminator), but their inner structure is adapted for the needs of macroscopic simulation. For interconnection of both micro and macro models, there is a special object called *convertor*, which performs conversion of traffic flows (from micro to macro and vice versa). For more information about the JUTS hybrid traffic model, see [20] and [21].

4 Distributed Simulation

As has been said before, although the computing power of the computers is still increasing, there are still simulations, which are too computation consuming to be performed on a single-processor computer in suitable time (e.g. a detailed simulation of large traffic network). A way how to obtain sufficient power for the simulation is to adapt it for multiprocessor hardware. In that case, each processor computes part of the simulation parallel to other processors. Each processor performs the computations of the assigned part of the simulation locally. It communicates with other processors only if it needs information from another part of the simulation.

4.1 Performance of the Distributed Simulation

The main reason for utilization of multiprocessor hardware is the increase of the simulation speed. If the adaptation of the simulation for the multiprocessor computer shall be meaningful, the enhancement of the simulation speed must be significant. There are several parameters, which describe the performance of the general parallel or distributed algorithms. They can be also used for evaluation of the multiprocessor simulation speed. The main two parameters are briefly described in following sections.

4.1.1 The Speedup of the Simulation

The *speedup* expresses the rate between the speed of simulation on the single-processor computer and the speed of simulation on the multiprocessor computer by utilization of p processors. It can be calculated as:

$$S = \frac{T_s}{T_p}, \qquad (1)$$

where S is the speedup, T_s is the computation time on the single-processor computer and the T_p is the computation time on the multiprocessor computer [22].

4.1.2 The Efficiency of the Simulation

The *efficiency* (η) is the rate between the speedup and the number of utilized processors [22]. It can be expressed as:

$$\eta = \frac{S}{p}, \qquad (2)$$

where η is the efficiency, S is the speedup and p is the number of processors utilized for computations. For example, if the simulation execution on the single-processor computer lasts 100 seconds and on the four-processor computer 50 seconds, the speedup is 2.0 and the efficiency is 0.5.

4.2 Multiprocessor Hardware

There are several types of multiprocessor computers. However, the most common classification can be done according to the physical area occupied by the computer as:

- Parallel computers
- Distributed computers

Both types are briefly described in following sections. More precise description of the multiprocessor hardware can be found for example in [2] and [23].

4.2.1 Parallel Computers

Parallel computers consist of multiple processors and other components, which are incorporated in one working unit. They are spatial compact (usually in one cabinet) and mostly homogenous (CPU of one type) [24]. Because the physical distances between processors are short, the communication latency is relatively low (less than 100 microseconds) [2].

According to the main memory distribution, the parallel computers can be classified as *shared-memory computers* and *distributed memory computers*. The shared memory allows using of simpler communication protocols among the processors. All processors can read and write values to the shared variables. It is only necessary to synchronize the access to these variables. The main disadvantage of the shared memory is that it reduces the possible maximum number of processors. Therefore, parallel computers with distributed memory are constructed when mass parallelism is needed (hundreds or thousands of processors). In this type of computers, the only possible way to maintain communication among all processors is the message passing [2].

In our work, we focus on the communication protocol for the distributed applications. Hence, the parallel computers will not be considered further.

4.2.2 Distributed Computers

Distributed computers originate from connection of several single-processor computers by communication links and cover a much broader area. They can be spread in a single building or university corpus or in larger area like city, state, or even whole world. The particulars computers in the distributed network are usually heterogeneous stand-alone machines with their own memory and I/O devices. These stand-alone machines can be ordinary workstations (e.g. in university labs) interconnected by means of ordinary Ethernet. In that case, no additional hardware costs are needed. The particular computers of the distributed computer are referred as *nodes*.

The main disadvantage of the distributed computer is relatively high communications latency. Because of the physical distances between particular computers the communication latency can be from hundreds of microseconds up to seconds. For the same reason, there can be no shared memory. So, like the parallel computer with distributed memory, the only manner of communication between particular computers is the message passing [2].

In this work, we will consider only the distributed computers and therefore only the distributed simulation. There are two main issues of the distributed simulation – *the decomposition* and *the synchronization* of the simulation. Both are described in following sections.

4.3 Decomposition

The decomposition determines how the simulation shall be divided into processes, which will be then preformed on particular nodes of the distributed computer. The appropriate decomposition is vital for resulting performance of the simulation. Right decision is highly application-dependent, but most common decompositions are:

- Task parallelization
- Spatial decomposition
- Temporal decomposition

All three approaches to the decomposition are briefly described in following sections.

4.3.1 Task Parallelization

By utilization of the task parallelization, the whole simulation program is decomposed into several modules. Each module is then performed on different node of the distributed computer. This approach is quite straightforward. However, the simulation speed is limited by the slowest module. So, it is necessary for all modules to consume similar amount of computing power and time. If one module consumes major part of computing power, this approach to the decomposition will not help much [14].

4.3.2 Spatial Decomposition

The spatial decomposition means that all modules of the simulation program are distributed among all nodes. So, the whole simulation program runs as a process on each node of the distributed computer. Each process then simulates assigned part of the simulation. For example, in traffic simulation of a city, every simulation process would run on a node and would simulate one part of city traffic network with its vehicles. This approach is straightforward if the original simulated system can be "easily" divided into parts according to space [14].

Like the task parallelization, it is necessary for all simulation processes to consume similar amount of computing power in order to run at "the same" speed. The computation within one simulation process can be performed locally, only when some event from a process affects another simulation process, the communication is needed. In order to achieve good performance of the resulting simulation, the communication among the processes of the simulation should be minimal. Hence, if it is possible, the simulation should be divided in way that most of the computations can be performed within every process itself and the events affecting other simulation processes should be relatively rare.

4.3.3 Temporal Decomposition

In some cases, where no spatial decomposition can be found, the temporal decomposition can be used. By this approach, the simulation is divided into (equal-sized) time intervals. Every computer then performs entire simulation, but only for one time interval.

The main problem of this approach is to ensure that the states of the simulation computed at the time interval boundaries match. This seems to be quite difficult, since how can be computed initial state of some interval without known final state of previous interval? In some applications, there are points in the simulation time, where the state is known or can be easily computed. So, the only thing to do is to divide the simulation in these points. When such points in time cannot be found, another approach can be used. Every node of the distributed computer computing some time interval of the simulation "guesses" the initial state of its interval and performs the simulation based on this guess. After all intervals are computed, there can be mismatches between the final and initial states of the neighbouring intervals. In that case, the computed final state of the first interval is used as the initial state of the second interval and the simulation of the second interval is recomputed. Then the correct final state of the second interval can be used as the initial state of the third interval and so on. By the recomputation of each interval, if there is match between the current computed state and previous computed state, the recomputation process can be stopped, because the remainder of the computation will be identical to the previous computation. The approach of temporal decomposition is limited to a handful of applications, such as queuing networks or Petri nets [2].

4.4 Synchronization

As we discussed the possible approaches to the simulation decomposition, we can proceed with the latter problem of the distributed simulation, which is its synchronization. Although it was said that it is convenient for all simulation processes distributed among particular nodes of the distributed computer to run at approximately same speed, their exact speed cannot be guaranteed neither predicted. Because the simulation processes need communicate among each other, the differences in their speeds can cause serious errors in the whole simulation.

For example, we consider an event-driven simulation where every simulation process advances its time autonomously. It is possible for one simulation process to receive message from another process with an event, which contains time-stamp less than the process' current simulation time. This event should be processed in the "past" and it could affect already processed events.

Another example can be seen in a time-stepped simulation. If every process advances its time autonomously, the messages from one process can arrive to another process in incorrect time step (past or future) and the whole simulation can produce incorrect results.

These violations of time causality, which cannot happen in a nonparallel simulation, are referred as *causality errors* [2]. In order to avoid these errors, every distributed simulation program must have some synchronization mechanism, which guarantees that each simulation process advances its time with respect to all other processes and the whole simulation produces the same results as a nonparallel execution of the simulation.

The problem of synchronization is the key issue of the distributed simulation. There exist a lot of synchronization mechanisms, which differ in efficiency and applicability. The selection of an appropriate synchronization mechanism closely depends on the time-flow mechanism used in the simulation.

4.4.1 Synchronization of Event-Driven Simulations

The synchronization mechanisms for event-driven simulations can be classified according to the approach to the causality errors as:

- conservative synchronization mechanisms
- optimistic synchronization mechanisms

Conservative mechanisms strictly avoid causality errors. Every simulation process can advance its time (process next event from event list) only if it is “safe”. This means that no event with less time stamp than the first event in the event list of the simulation process will arrive from another simulation process. Hence, the key problem the conservative mechanism must solve is to determine, whether it is safe to process an event or not.

The approach to achieve this can be either *synchronous* or *asynchronous*. In an asynchronous approach, there is no global mechanism, which could control the advancement of time in the particular simulation processes. The simulation processes only informs each other about the minimal time stamp of the next event they will send. This is the main idea of an asynchronous synchronization mechanism called *null-message protocol*. For further information see [2].

Synchronous conservative mechanism uses the synchronization construct of *barrier* known from general parallel programming (for details see [22]). The computation of every simulation process consists of two phases. In first phase, the simulation processes identify the events, which can be safely processed. In second phase, these events are processed. After each phase, the simulation processes are synchronized at the barrier. The synchronization means that no simulation process can proceed with next phase, unless all processes have finished the current phase.

Unlike conservative approach to the synchronization, with the optimistic synchronization mechanism, the causality errors are possible, but a mechanism is provided to enable detecting of these errors and repairing them. The first and still most well-known optimistic mechanism is called *Time Warp*. The basic idea is quite simple. For every message sent in the simulation, a so-called anti-message is generated. When some message causes a causality error in a process, this message must be cancelled in order to repair the error. This could be a problem, because the cancelled message could already cause some other messages to be generated. However, the cancellation of the message can be done easily by sending the anti-message to the same process. The messages caused by cancelled message can be also cancelled by sending corresponding anti-messages to correct processes. It should be noted that there is no global mechanism, which control the time advancement of particular processes. The processes run autonomously until a causality error is detected. It should be also noted that the cancellation of messages is quite computation-consuming. So, the causality error must be rare in order to achieve good performance of the resulting simulation.

The use of conservative and optimistic synchronization remains highly application-dependent. The conservative protocols offer simple simulation executive, but the possible parallelization is application-dependent. On the other hand, the optimistic protocols are more robust and less application-dependent, but they require complex simulation executive [2].

4.4.2 Synchronization of Time-Stepped Simulations

In the field of time-stepped simulations, there is one synchronization mechanism, which is used most often – the synchronous conservative mechanism called *master-slave approach*.

The mechanism is similar to the synchronous conservative mechanism for event-driven simulations (see previous section). There is one master process and number of slave processes. The master process controls the time advancement of the slave processes and the slave processes performs the computations of the assigned parts of the simulation. More precisely, the master process provides the barrier, on which the slave processes are synchronized at the end of every time step. Hence, all slave processes perform at the same moment the same time step and no causality errors are possible.

5 Distributed Traffic Simulation

Now, as we discussed the issues of the traffic simulation and general distributed simulation, we can proceed with the description of the distributed traffic simulation. The issues of the general distributed traffic simulation will be discussed in this chapter, as well as the concrete solutions used in the distributed version of the JUTS system.

5.1 Decomposition of Traffic Simulation

The decomposition of a general simulation was discussed in section 4.2. In this section, we will discuss the usability of the particular decomposition types in the field of distributed traffic simulation.

5.1.1 Task Parallelization of Traffic Simulation

Task parallelization is hardly suitable for the traffic simulation, because the modules of the simulation program often consume unequal amount of computing time. The major part of the computing power is consumed by the vehicle movement module. The requirements of other modules like map depiction or statistical results collection are negligible, so, the speedup of the simulation would be minimal.

However, a sort of task parallelization can also be uses in some special cases. The example can be seen in [15], where the whole simulation incorporates heterogeneous modules. The traffic simulation module models urban street traffic flow on microscopic level, second module is a man-in-the-loop driving simulator and third module visualize entire simulation. Every module is performed on different computer, so, it can be assumed as a task-parallelized simulation.

5.1.2 Spatial Decomposition of Traffic Simulation

The most common way how to decompose a traffic simulation is the spatial (or domain) decomposition. The whole simulated traffic network is divided into number of sub-networks and every simulation process then performs the simulation of one assigned traffic sub-network. The movement of the vehicles within every sub-network is computed locally. The communication is needed only when a vehicle has to pass to the neighbouring sub-network. The main problem is that it is relatively difficult to divide the network into sub-networks, which would be approximately equally computation-consuming. This is necessary, because the slowest sub-network determines the resulting speed of entire simulation. There are several solutions of this problem, which will be discussed next.

The easiest solution is to divide the traffic network into equal-sized pieces. This approach is used for example in ParamGrid [12]. The whole traffic network is divided into a grid, arranged in rows and columns, of smaller rectangular geographic areas. The whole traffic network can be then watched on a grid of monitors. The main disadvantage is that the number of vehicles in every geographic area can be very different, because of various road densities and also various traffic densities in the roads. The latter problem is that large amount of roads can be affected by the cut, because the number of roads that are crossing the boundaries of the sub-networks is not considered during network division. However, this approach is suitable if the road density is more or less uniform or for testing purposes.

There are some more sophisticated methods of traffic network division. One is used in TRANSIMS [11]. In this simulator, the whole traffic network is divided into sub-networks of similar size. The size is measured as accumulated length of the roads associated with the sub-network. By the division of traffic network, the number of divided roads and the number of sub-networks' neighbours are minimized. For this purpose, the graph partitioning methods are used (e.g. orthogonal recursive bi-section).

An interesting approach can be seen in the implementation of the traffic simulator *vsim* [25]. In this case, the traffic network is divided according to the number of vehicles passing along the particular traffic lanes. In *vsim*, the numbers of vehicles are collected after one simulation run. Similar method can be also used in the distributed version of the JUTS system. For testing purposes, we can exploit the real traffic intensity data measured at all crossroads with traffic lights of Pilsen city, which are at our disposal [26]. From this data, we can make quite accurate estimation of traffic densities along particular traffic lanes. These densities can be used for uniform network division. However, for the preliminary test, the division of the traffic network into equal-sized pieces is utilized.

Another question, which needs to be solved by the division of traffic network, is where the traffic lanes of the network should be divided. The cut can be performed either in the middle of the traffic lanes, or between the lanes and the crossroads. Both of these possibilities are commonly used, the first one for example in TRANSIMS and the second one in *vsim*. In the distributed version of the JUTS system, the first approach will be used, because an immediate surrounding of a crossroad is more interesting from a traffic analysis viewpoint and it should not be divided between two or even more sub-networks.

5.1.3 Temporal Decomposition of Traffic Simulation

Although the spatial decomposition is most common in the field of traffic simulation, there are also several attempts to use the temporal decomposition. An example can be seen in [27]. The main advantage of this approach is possible utilization of massive parallelism, because there is no need for communication among processes during the parallel computation. The main disadvantage is that, after the computation of all temporal intervals, the states on the boundaries of the intervals may not match, especially in traffic simulation, which exhibits very complex states. Hence, after the computations of all intervals are complete, the so-called fix-up computations are needed to match the states on interval boundaries.

Unfortunately, the fix-up process is efficient only if the matching states are found quickly. In the worst case scenario, the matching states are not found at all, and all intervals must be recomputed. In that case, the parallel simulation degrades to the sequential simulation. As it is shown in [27], the states on the boundaries of temporal intervals match quiet quickly for low traffic density. However, for higher traffic densities, the convergence of the states is not sufficient. A solution of this problem, which is used in [27], is the approximate state matching. This method enables to increase the performance of the simulation by cost of an introduced error. The states on the interval boundaries do not have to match exactly, it is sufficient, if they are "close enough". These similar states are found much faster than the identical states, but some error is introduced into the results.

As it is obvious from previous paragraph, the temporal decomposition of the traffic simulation is indeed possible, but is at least problematic. We will not consider it further.

5.2 Inter-Process Communication

The inter-process communication is one of the main bottlenecks of every distributed application. In the distributed computing environment, the only means of communication is the message passing, which is relatively slow and has high communication latency (depending on mutual distances of particular nodes of the distributed computer) [2].

5.2.1 Inter-Process Communication Requirements

To achieve good performance and sufficient speed-up of the distributed simulation, the communication among its particular processes should be minimal. If we consider the most common spatial decomposition of the traffic network, the inter-process communication is necessary for synchronization and for transfer of vehicles between the neighbouring traffic sub-networks. The synchronization is discussed in section 5.3, the transfer of vehicles between the sub-networks is described here.

In order to minimize the communication necessary for transfer of vehicles, the number of traffic lanes affected by the network division should be minimal. This should be considered already during the decomposition of the traffic network (see section 5.1). However, even if the traffic network is well divided and number of traffic lanes affected by the cut is small, there are still vehicles, which need to be transferred from a sub-network to another.

5.2.2 Implementation in the JUTS System

In the distributed version of the JUTS system, the communication links are maintained between the neighbouring traffic sub-networks because of the vehicle transfer. The vehicles are transferred in the form of messages from the source sub-network to the target sub-network.

Besides the transfer of vehicles, it must be also possible to notify a neighbouring sub-network that a traffic lane is congested and cannot receive any vehicles. For this purpose, the *lane-block message* is introduced [28]. Compared to the vehicles, these messages travel in opposite direction. If a lane-block message is received, the corresponding traffic lane becomes temporally blocked and no vehicles are permitted to pass to the neighbouring sub-network. If the lane-block message is received again, the blocked traffic lane becomes disposable again. For both vehicle and lane-block transfers, the modified terminators and generators are used (see Fig. 3).

In general, the neighbouring traffic sub-networks are interconnected by set of traffic lanes with various travel directions. So, in a time-step, multiple vehicles and lane-blocks can be sent from a sub-network to one of its neighbours. It would be an inefficient solution to send each vehicle or lane-block as a separate message because of the communication overhead. Hence, all vehicles and lane-blocks determined for one neighbouring sub-network are stored in a buffer. At the end of every time step, the buffer is checked. If it is not empty, the content of the buffer is sent as one message to the corresponding neighbouring sub-network. There is one buffer for each neighbouring sub-network. So, the maximum number of messages sent by a traffic sub-network in a time step corresponds to the number of its neighbours [29].

The transfer of vehicles and lane blocks is depicted in Fig. 3. There are two neighbouring traffic sub-networks interconnected by two traffic lanes with opposite travel directions (the lane 2 conducts from the sub-network 1 to sub-network 2 and the lane 1 conducts from the sub-network 2 to sub-network 1). Every end of each lane is equipped by special generator or terminator according to the travel direction in the lane.

The lane 1 in the sub-network 1 is congested, so the lane-block is stored to the buffer (a) by the special generator (G). In the lane 2 in the sub-network 1, a vehicle is passing to the sub-network 2. So, it is removed from the lane (b) by the special terminator (T) and the vehicle's description is stored to the buffer (c). At the end of the time step, the content of the buffer is packed into a message and sent to the sub-network 2 (d). In the sub-network 2, the message is received (e). The lane-block is forwarded to the corresponding terminator and the lane is blocked (f). The vehicle's description is also forwarded to the corresponding generator (g). The generator then creates the vehicle according to the received description and inserts it in the lane (h).

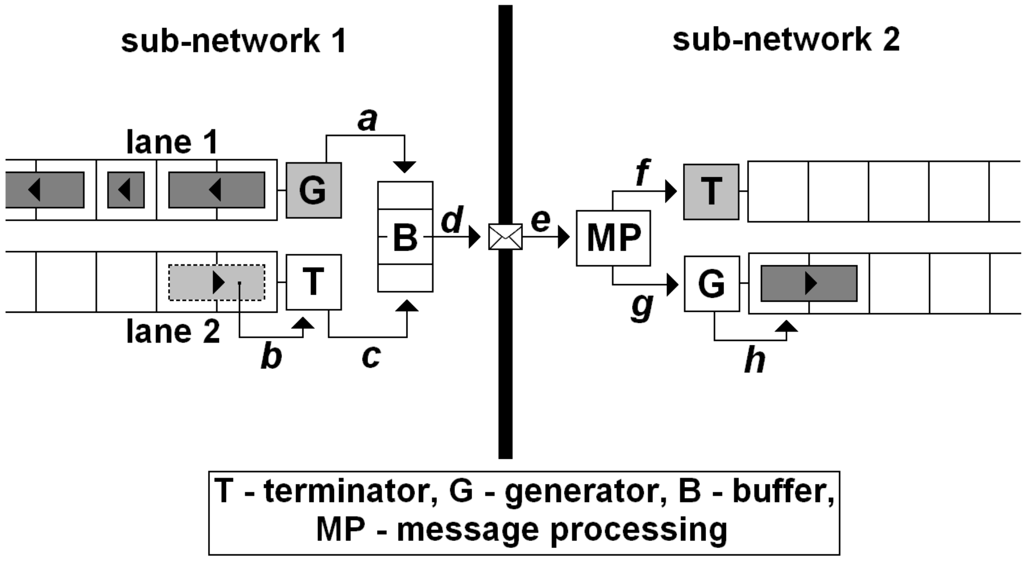

Fig. 3: Transfer of vehicles and lane blocks between two traffic sub-networks

The described solution of the communication between the traffic sub-networks is consistent with traffic network structure of the JUTS system. There is one problem however, which is caused by the various lengths of the simulated vehicles. Because a vehicle can be up to six cells long, it is theoretically possible for it to be simultaneously on two traffic sub-networks (see Fig. 4). However, from the implementation point of view, it is impractical to split a vehicle into two pieces in one time step and link these pieces together in next time step.

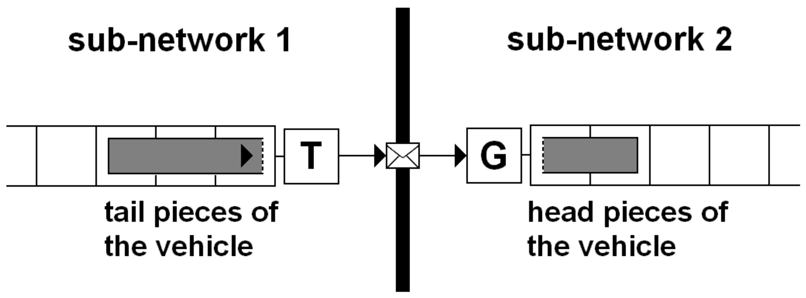

Fig. 4: A long vehicle passes between two neighbouring sub-networks

This problem can be solved easily. The terminator waits until the whole vehicle disappears from the road in the sub-network 1. Then it stores the vehicle description in the buffer together with the information about the vehicle shift. The shift value represents the distance in cells, which the head piece of vehicle would travel if the road will not be divided. After the vehicle description reaches the corresponding generator in the sub-network 2, the vehicle is generated and then immediately moved forward according to the shift value sent in the message.

It should be also noted that the maximal number of the vehicles, which can be transferred from the sub-network 1 to the sub-network 2 is not limited by the number of the traffic lanes conducting from the sub-network 1 to the sub-network 2. For example, if two vehicles in one traffic lane are traveling at high speed tight one after another, they are able to pass the boundary between the sub-network 1 and 2 in the same time step. In that case, both vehicles must be transferred to the sub-network 2 (see Fig. 5). Hence, the capacity of the buffer for every particular sub-network must be several times greater than the number of lanes conducting to the sub-network. Also, the massage must be able to incorporate more than one vehicle from one particular traffic lane.

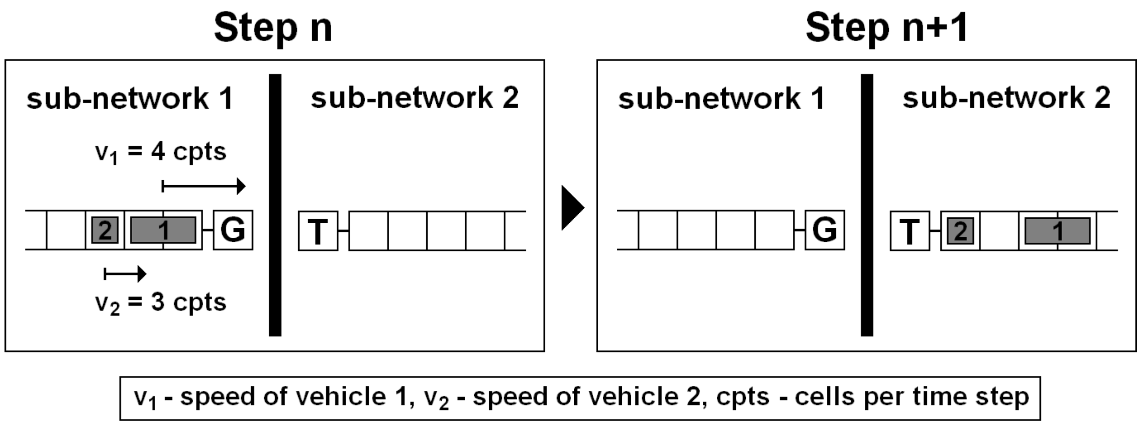

Fig. 5: Two vehicles pass between neighbouring sub-networks in one time step

5.3 Synchronization of Traffic Simulation

As has been said before, the synchronization is necessary for correct mutual running of the simulation processes. So, the simulation can advance in time without any causality errors. Also, the distributed simulation must give the same results as any sequential execution of the simulation [2].

5.3.1 Master-Slave Approach in the JUTS System

As has been said in section 4.4.2, the most often used synchronization mechanism for general time-stepped simulation is the master-slave approach. It is also widely used in the field of distributed traffic simulations and it is also utilized in the distributed version of the JUTS system.

By this approach, there is one master process and number of slave processes. Each process performs the simulation of one assigned traffic sub-network locally. After the slave process finishes the computation of one time step, it sends the vehicles and lane-blocks to its neighbouring slave processes, if needed. Then it sends a notification message to the master process and waits. After the master process receives notifications from all slave processes, it broadcasts the permission to continue with next time step to all slave processes. So, the synchronization requires two messages per time step for every slave process [29]. The whole communication scheme is depicted in Fig. 6.

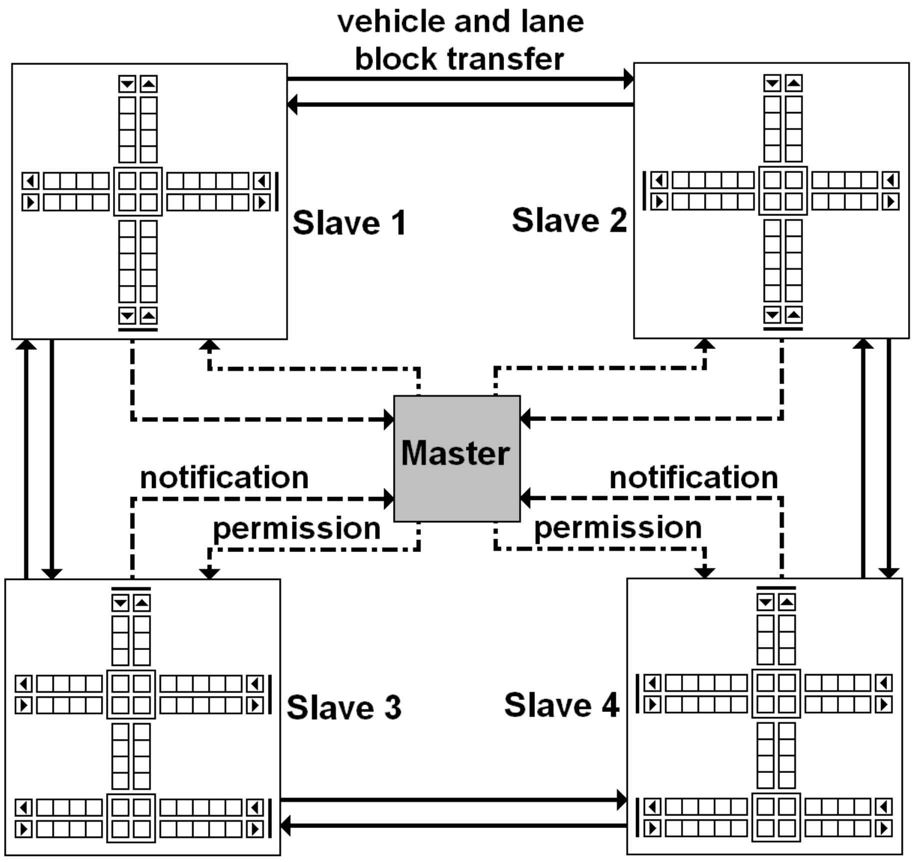

Fig. 6: The master-slave approach in the distributed version of the JUTS system

5.3.2 Commonly Used Synchronization Mechanisms

Most of the distributed traffic simulation uses an implementation of the master-slave approach. For example, the TRANSIMS [11] and *vsim* [25] simulators utilize an implementation, which is very similar to that one used in the distributed version of the JUTS system (see previous section). Because this type of synchronization requires two messages (a notification and a permission message) per time step for every slave process, the total number of messages is equal to *2p* if there are *p* slave processes. So, the number of messages increases linearly with increasing number of slave processes.

The master-slave approach represents the centralized synchronization mechanism. It is also possible to use a distributed synchronization method where no central master process exists. This approach is used for example in ParamGrid [12]. In this case, each working process broadcasts a message with its identification number and the time stamp of current finished step in every time step to all other processes. Each process then waits until it receives messages with the same time stamp from all other processes.

This synchronization mechanism has the advantage of no central point of the simulation. However, the number of messages necessary for synchronization per one time step can be expressed as follows:

$$M_s = p \cdot (p-1) = p^2 - p, \tag{3}$$

where the M_s is the total number of messages per one time step and p is the number of processes. As it is obvious from the equation (3), the number of messages depends quadratically on the number of processes. Already for four processes, this approach requires more messages than the master-slave approach. Hence, we can consider the master-slave approach as the more efficient solution than the distributed synchronization method.

5.4 Middleware for Inter-Process Communication

As arise from the previous chapter, the inter-process communication in the simulation is needed for the synchronization and for the transfer of vehicles from one traffic sub-network to another. The communication protocol can be implemented in many ways. It is possible to implement one's own communication protocol. However, the most common approach is to utilize an existing *middleware*, which is a layer that provides services for inter-process communication. The main advantage of this approach is the strict separation of the communication and the business logic of the simulation.

The majority of the distributed traffic simulations use any common middleware. The middleware commonly used in distributed traffic simulations are for example the CORBA (Common Object Request Broker Architecture) [12] or the HLA (High Level Architecture) [15].

In the distributed version of the JUTS system, we also want to use an appropriate solution of inter-process communication. Because the JUTS system is written in the Java programming language, the selected middleware must be applicable for programs written in this language. The possible middleware are briefly discussed in following sections. More precise description can be found in [28].

5.4.1 TCP/UDP Sockets

The utilization of sockets is a low-level way, how to maintain communication among processes of a distributed simulation. Sockets are a standard abstraction of computer network connections and are implemented in many programming languages including Java.

There are two types of services – the connectionless UDP (User Datagram Protocol) and the connection-oriented TCP (Transmission Control Protocol). In the UDP, there is no concept of stream or connection between the two communicating computers. The data are sent in datagrams (UDP packets) and there is no guarantee of their delivery. The datagrams can arrive out of order, get lost, or be duplicated. The UDP protocol is unreliable and therefore there is no reason to use it in the JUTS system. We will not consider it further.

The TCP is a protocol, which provides reliable communication among computers. The TCP creates connections and uses them for exchanging of data streams. The protocol guarantees reliable transfer of data from sender to receiver. The data packets also arrive in order they were send [30]. Because the TCP protocol is reliable, it could be used in distributed version of JUTS system.

Because the TCP sockets support only the transfer of raw data, there is no overhead caused by additional features like object serialization. Therefore the main advantage of the sockets is the speed of communication. The main disadvantage is that the protocol for encoding and decoding of transferred data must be implemented in application level along with business logic of the simulation. This approach can be error-prone. From this point of view, the TCP sockets cannot be considered as a middleware, since it does not guarantee the separation of the communication from the rest of the simulation. On the other hand, because the TCP is a low-level and therefore very fast solution of inter-process communication, it is considered as a good solution for the distributed version of the JUTS system.

5.4.2 RMI

The RMI (Remote Method Invocation) is an object-oriented version of RPC (Remote Procedure Call) incorporated in the Java Core API. The RMI uses the object-oriented paradigm in the distributed environment. By its utilization, it is possible to invoke a method on an object that resides in different JVM (Java Virtual Machine). Moreover, the invocation of a remote method is very similar to the invocation of a local method. All common mechanisms known from local computing (e.g. propagation of exceptions, garbage collection) are provided.

In the RMI communication, there are two processes – the *client*, which makes the invocation of a remote method, and the *server*, which executes the method and returns the result. The RMI protocol uses a local surrogate object (stub), which is the client's local representative for the remote object. The client makes invocation on the local stub, which is responsible for carrying out the method invocation on the remote object [30].

The RMI uses the TCP sockets for data transfer. Hence, by every method invocation, the parameters and method's name must be encoded into stream of bytes on the client side, transferred via the TCP sockets, and than decoded on the server side. The result is transferred from the server to the client in similar way.

So, the RMI must implement a mechanism for object serialization and also for distributed garbage collection and distributed propagation of exceptions. These additional features bring an overhead into the communication. Hence, the RMI is slower than the TCP sockets. On the other hand, no additional features (encoding and decoding of transferred data) must be implemented in the business logic of the simulation. So, the inter-process communication is quite comfortable and transparent. Hence, the RMI seems to be even better solution for the JUTS system than the TCP sockets if the speed difference between the RMI and the TCP sockets were negligible.

However, the performances of both RMI and TCP have been tested and compared (see section 7.1). The results show that the communication using TCP is from 3.5 to 29 times faster than the communication using RMI. On the other hand, the RMI hides the communication details and therefore is less error-prone than the TCP. A way how to exploit the advantages of both solutions is a compromise. The RMI is used throughout the development of the distributed version of the JUTS system. After the debugging of the simulation program, slower RMI will be replaced by the faster TCP [28].

The comparison of communication speeds of the RMI and TCP sockets can be found in section 7.1.

5.4.3 CORBA

The CORBA (Common Object Request Broker Architecture) is an alternative to the RMI. It is a standard language- and platform-independent architecture for distributed object systems. Using CORBA, objects written in different programming languages can interoperate each other [30].

The functionality of CORBA is similar to RMI. It also uses the stubs for local representation of remote objects. However, because the cooperating objects can be written in different languages, the CORBA executive is far more complex than the RMI executive. Hence the CORBA is slower and less transparent than the RMI. So, there is no reason for using it in the JUTS system [28].

5.4.4 HLA

The last solution, which will be mentioned, is the HLA (High Level Architecture). The HLA was originally developed for interconnection of various man-in-loop simulators into one distributed virtual environment, which could be used for training of military personnel. More information can be found in [31]. Since then, the HLA has been widely used as middleware in various simulations both in military and commercial sectors.

Similar to CORBA, the HLA executive enables interconnection of heterogeneous simulations. It is even possible to connect a real component into the simulation. However, because of this commonness, the HLA executive is very complex and therefore slower than RMI. Because there is no need for special abilities of the HLA in the distributed version of the JUTS system, there is no reason to use it [28].

6 Reduction of Inter-Process Communication

As has been said before, the inter-process communication in the distributed version of the JUTS system is necessary for transfer of vehicles and lane blocks and for synchronization of particular simulation processes. The concrete implementation of the inter-process communication in the JUTS system is described in sections 5.2 and 5.3. In order to improve the resulting performance of the distributed simulation, we will now focus on reduction of the inter-process communication.

6.1 Number of Messages Sent per One Step

First, we will calculate the maximum number of messages, which can be sent per one time step. We consider that, for synchronization of the simulation, the master-slave approach is used. So, there is one master process and *p* slave processes. Every particular slave process is connected to the master process in order to enable transfer of synchronization messages. Moreover, the communication links are also maintained between the neighbouring slave processes in order to enable the transfer of vehicles and lane blocks. Under these circumstances, the maximum number of messages sent per one time step can be expressed as follows:

$$N = N_s + N_t = 2p + \sum_{i=1}^{p} n_i \ , \tag{4}$$

where N_s is the number of messages necessary for synchronization, N_t is the maximum number of messages necessary for vehicle and lane block transfer, and n_i is the number of neighbours of the i*th* slave process. For the synchronization, two messages are necessary per every time step (one notification and one permission message). The maximum number of messages sent by a slave process to another slave process is equal to the number of its neighbours. This value depends on division of the traffic network. In a real case, a slave process has approximately from four to six neighbours [12], [14].

Throughout the development of the distributed version of the JUTS systems, two enhancements of the communication protocol were proposed and implemented. Both are focused on reduction of the communication necessary for the vehicle transfer. These enhancements are described in following sections in detail. The possible approaches to the reduction of the communication necessary for the synchronization of the particular simulation processes are mentioned in section 8.4 and 8.5.

6.2 Traffic Flow Characteristics Transfer

For high traffic densities, the number of messages sent by a slave process to its neighbours reaches the maximum value in almost every time step. This means the intensive communication among the slave processes. However, this communication can be considerably reduced.

6.2.1 Main Idea of the Traffic Flow Characteristics Transfer

For the reduction of inter-process communication, we can utilize the main idea of the so-called *dead reckoning*. This technique is used in the HLA [31] and its ancestors. As has been said, the HLA is a distributed simulation executive, which enables the interconnection of various man-in-loop simulators in one distributed virtual environment. The dead reckoning is used for projection of the other simulation's participants to the field of view of each participant. To reduce the communication necessary for update of all participants' positions, the dead reckoning is used. It interpolates the trajectories of all participants in a participant's field of view according to their last known position. Only if the interpolated movement is too different from the real one, the synchronization message with current position is sent [32].

A similar approach is used in the distributed version of the JUTS system for the reduction of vehicle passing between the sub-networks. The transfer of vehicles is still provided by the special terminators and generators (see section 5.2.2). Instead of sending the particular vehicles, it is possible to send only the characteristics of traffic flow (e.g. vehicle density, mean speed). These parameters are calculated by the terminator according to the passing vehicles, and then sent to the corresponding generator instead of vehicles. The generator can then create new vehicles according to the received characteristics [32].

The main advantage of this approach is that, besides the beginning of the simulation, the characteristics must be sent only if they change. If the characteristics are more or less constant, no inter-process communication is needed. The generator can create the vehicles according to last received values. Only if the difference between the actual traffic flow and the last sent characteristics reaches certain threshold, the message with new values is sent from the terminator to the corresponding generator [32].

There are also some disadvantages. The traffic flow in front of the terminator (terminated traffic flow) and the traffic flow behind the corresponding generator (generated traffic flow) have indeed the same characteristics, but they are not identical. If a vehicle is removed from the road by the terminator, is does not need to be immediately generated by the corresponding generator [32].

There is also a little inaccuracy if the traffic flow characteristics changes less than the defined threshold. In that case, the characteristics messages will not be sent and the generator will produce the vehicles according to the old values [32].

Moreover, the characteristics are calculated from the number of last terminated vehicles, in order to moderate the immediate random fluctuations in the traffic flow. Consequently, a change of the terminated traffic flow does not take effect in the generated traffic flow immediately, but after a short period of time, designated as a *delay* [29]. However, the mentioned problems can be minimized by appropriate settings of the threshold (see section 7.2).

6.2.2 Implementation of the Characteristics Transfer

The transfer of characteristics is very similar to the transfer of vehicles described in section 5.2.2. All characteristics determined for each neighbouring sub-network are stored in the buffer. At the end of a time step, the content of the buffer is sent as one message. It should be noted that it is still necessary to use lane blocks as an indication, that a traffic lane is congested and cannot receive any vehicles. The lane blocks are also stored in the corresponding buffers, similar to the transfer of vehicles [29].

The inner structure of the characteristics message is quite different from the vehicle message. The traffic flow in the JUTS system can be described by three parameters – the temporal density of the vehicles (how many vehicles passed through the terminator per time step), the mean speed of the vehicles, and the array with distribution of vehicle lengths. The mean length would be inadequate, because it does not describe the real length of the vehicles. For example, if there are two vehicles with lengths 1 and 6, the mean length is 3.5, which is far from the real lengths of the vehicles. Hence, the characteristics message incorporates a variable for vehicle density (lambda), a variable for mean speed and an array for vehicle lengths. So, a characteristics message is a bit longer than a vehicle message, which incorporates only three integer variables [29].

The vehicle lengths and the mean speed are calculated according to last N_v vehicles, which passed through the terminator. The vehicle density is calculated according to the number of vehicles passed through the terminator within N_s last steps. As had been said, the characteristics should be sent to the corresponding generator if the difference between the last sent and actual characteristics reaches certain threshold. Because the three described parameters are independent each other, it is desirable to have separate threshold for each of them. The thresholds can be designated as T_d, T_l, and T_s for the vehicle density, lengths, and speed, respectively. So, the characteristics message is sent when at least one of the thresholds is reached. The correct setting of the thresholds and the N_v and N_s constants is a compromise between the number of sent characteristics C and the difference between the terminated and generated traffic flows. The optimal values of the thresholds and constants have been determined by set of tests, which are described in section 7.2.

The performance of the characteristics transfer has been intensively tested and compared to the vehicle transfer. The test and results are described in section 7.3.

6.3 Adaptive Vehicle Density Transfer

The communication protocol, which uses the transfer of traffic flow characteristics instead of transfer of particular vehicles, can be improved in order to achieve additional reduction of the inter-process communication.

As has been said, there are three parameters, which describe the traffic flow (see previous section). For each parameter, there is a separate threshold. If one of the thresholds is reached, the characteristics message with all three parameters is sent. Throughout the testing of the characteristics transfer (see section 7.3), it was determined that the majority of messages is sent because of the vehicle density. It is the most fluctuating parameter and also the most important one, because it determines the distribution of the vehicle in the traffic lanes.

The fluctuations of the latter two characteristics – the mean speed of the vehicles and lengths of the vehicles – are less frequent. Moreover, the majority of changes to these two characteristics are transferred when the transfer of vehicle density is needed. Consequently, the number of messages sent only because of the speed or lengths of the vehicles is minimal. For this reasons, we will focus on the reduction of the vehicle density transfer.

6.3.1 Main Idea of the Adaptive Vehicle Density Transfer

In the real traffic network of a city, a large number of crossroads is equipped by traffic lights. These traffic lights cause periodical changes of the vehicle density in the traffic lanes outgoing from the crossroad. If such traffic lane conducts to a neighbouring sub-network, the periodical behavior of the vehicle density can be learned by the traffic terminator at the end of the lane and the corresponding generator. After the behavior of the vehicle density is learned from a traffic lights cycle, it can be predicted in the next cycle. The generator can then generate the vehicles according to the predicted vehicle density. In that case, the communication is necessary only for corrections of the prediction. These corrections are necessary because of the random fluctuations in the traffic flow [33].

The described approach corresponds to the time series prediction problem. However, there are several specific conditions. First of all, the learning process must be fast. In an ideal case, the generator should be able to predict the vehicle density after one cycle of traffic lights. Nevertheless, the learning process must be in progress throughout the whole simulation run, because the behavior of the vehicle density can change in time (e.g. because of rush hours). Also, the generator must be able to learn and predict the vehicle density behavior only with limited communication with the terminator. The last requirement is necessary, in order to achieve communication savings [33].

6.3.2 Utilization of an Artificial Neural Network

For the time series prediction, the artificial neural networks are often used [34]. The artificial neural networks are the class of mathematical algorithms, which are originally based on biological networks found in living organisms [35]. Unlike the ordinary computer calculations, the neural networks require no formal definition of the algorithm. The required functionality of the network is gained by the process of learning.

There are many types of neural networks. For the time series prediction, the multi-layer perceptron networks are often used [36]. A perceptron network consists of several interconnected neurons (perceptrons), which are organized in layers. Each neuron incorporates m weighted inputs. These inputs are summed and then transformed by the threshold function into a single output. The network is learning by modifying the weights of particular neurons' inputs. The neural network, which is able to predict future values of the time series, is depicted in Fig. 7.

As can be seen in Fig. 7, the prediction (x_{t+1}) is made according to ($n+1$) previous values of the time series ($x_t - x_{t-n}$) [37]. However, this is a problem for our prediction of the vehicle density. By the utilization of a perceptron network, the ($n+1$) past values of vehicle density are needed for prediction of the next value. Hence, in every time step, a value of the vehicle density must be transferred between the terminator and the generator. Obviously, this approach would lead to no communication savings.

Theoretically, this problem can be evaded, because the predicted values of the network can be used also as inputs for next prediction. However, in that case, the inaccuracy of the prediction increases considerably in every time step. Thus, large number of corrections is needed and the communication is not reduced at all [33].

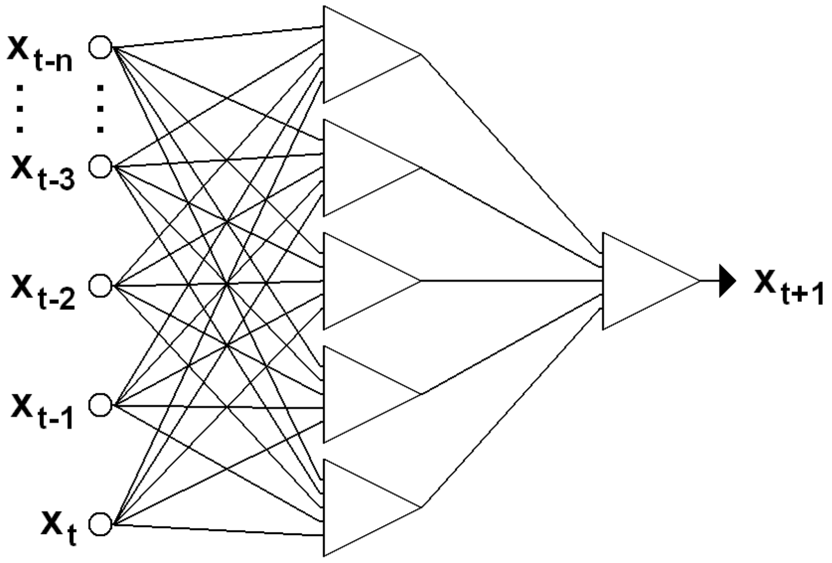

Fig. 7: An example of the perceptron network for time series prediction

Moreover, the perceptron network must be learned first, before it could make any prediction. The learning process lies in the submission of the training input values to the network and checking of its output. If the output is too different from the required one, the weights are adjusted, in order to decrease the error [35]. This procedure must be done before the utilization of the network for prediction. There are also methods for on-line training of the networks when the network is learning continually [36]. Still, the learning process requires many samples of input values and corresponding required output values, which are unavailable for the generator. Hence, the utilization of an artificial neural network is not suitable for our prediction of the vehicle density.

6.3.3 Traffic Lights Cycle Recording

To find a suitable solution of prediction of the vehicle density, we will focus on the traffic lights cycle, which is causing the periodical behavior of the vehicle density. In many cases, the traffic lights cycle is static. This means that the cycle length and also the length of the particular green and red periods are constant. A more advanced alternative is the dynamic cycle with constant length. This means that the cycle length is constant, but the particular green and red periods can change according to the number of passing vehicles. These two types of cycles are used in traffic lights of Pilsen city, whose traffic network is used for the tests of the JUTS system. The most general type of traffic lights cycle is the dynamic one with varying cycle length [38]. Presently, this type of traffic lights cycle is not considered in the simulation [33].

If we consider only the constant lengths of the traffic lights cycles, the behavior of the vehicle density can be easily recorded as an array of values with the length corresponding to the length of the cycle. This record of the vehicle density behavior can be then used for prediction of the vehicle density behavior in the next cycle [33].

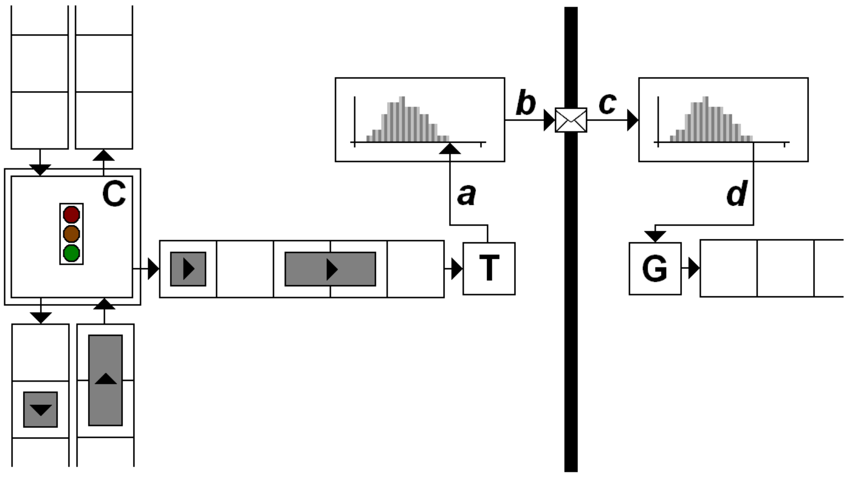

Fig. 8: The recording of the vehicle density by the terminator

Consider now the situation depicted in Fig. 8. There is a crossroad with traffic lights (C), which are affecting the behavior of vehicle density in the terminator (T). At the beginning of the simulation, the terminator determines the length of traffic lights cycle from the crossroad and sends it to the generator (G). Throughout the simulation, the terminator records the current vehicle density into an array (a) and compares it with the corresponding density from the last cycle. If the difference reaches certain threshold, the terminator packs the whole array using RLE (Run Length Encoding) compression and sends (b) it to the generator. The generator unpacks the array (c) and uses it for prediction of the vehicle density (d). The algorithm is described by pseudo-code in Fig. 9 and Fig. 10.

```
cycleLength = determineCycleLength();
sentToGenerator(cycleLength);
densities = new double[cycleLength];
lastSentDensity = 0.0;
actualDensity = 0.0
step = 0;

while (simulationInProgress()) {
 index = step % cycleLength;
 actualDensity = getActualDensity();
 d = diff(actualDensity, lastSentDensity);

 if (d > DENSITY_THRESHOLD) {
  d = diff(densities[index], actualDensity);

  if (d > ADAPTIVE_THRESHOLD || isRecorded(index) == false) {
   densities[index] = actualDensity;
   packed = packRLE(densities);
   sendToGenerator(packed);
  }
  lastSentDensity = actualDensity;
 }
 densities[index] = lastSentDensity;
 step++;
}
```

Fig. 9: The pseudo-code for the terminator

In the terminator, there are two thresholds. The `DESITY_THRESHOLD` is the threshold of the difference between the last sent and current vehicle density. By the characteristics transfer (see section 6.2), if this threshold was reached, the vehicle density would be sent to the generator. By the adaptive vehicle density transfer, the array with recorded vehicle densities is sent only if the second condition is also satisfied. This occurs if the difference between the current density and the density recorded in the last cycle reaches the `ADAPTIVE_THRESHOLD` or the cycle has not been recorded yet. The second threshold is set only to 15 %, because it increases the difference between the terminated and generated traffic flow considerably [33].

```
cycleLength = acceptFromTerminator();
densities = new double[cycleLength];
lastDensity = 0.0;
actualDensity = 0.0;
step = 0;
while (simulationInProgress()) {
 index = step % cycleLength;
 if (incomingMessage()) {
  packed = acceptFromTerminator();
  densities = unpackRLE(packed);
  lastDensity = densities[index];
 }
 actualDensity = lastDensity * decrease()
  + densities[i] * (1 - decrease());
 generateVehicles(actualDensity);
 step++;
}
```

Fig. 10: The pseudo-code for the generator

In the generator, the received array with vehicle densities is being used for prediction of the current vehicle density. Consider now the situation that the cycle length is 60 steps and the array is received in 40th step of the cycle. In that case, the first 40 densities are recorded from the current cycle and the remaining 20 densities are recorded from the previous cycle. The last known density of the current cycle is that one from the 40th step. For this reason, if the prediction for the 45th step shall be computed, it is calculated as the weighted average from the density in 40th step (recorded in current cycle) and the density from 45th step (recorded in previous cycle). The weights are represented by function `decrease()` (see Fig. 10). Its value decreases with increasing distance from the last known density of current cycle (density from 40th step in our example). After several steps, its value reaches zero, and the prediction is calculated only from the array of densities [33].

6.3.4 RLE Compression of the Recorded Vehicle Densities

By utilization of the characteristics transfer (see section 6.2), one value of vehicle density is transferred in every message. By utilization of the adaptive vehicle density transfer, the whole array of values is transferred in the message. So, the message is considerably longer. To minimize the message length, the array is compressed using the RLE compression [33].

Because the values of the vehicle density are real, the values must be transformed to the integer first. The vehicle density is calculated from ten last steps (see section 7.2). So, the density multiplied by ten give us the number of vehicles passed in ten last steps (i.e. small integer number). These values can be stored in a byte array, because the range of a byte is sufficient. In the resulting byte array, there are many repeating values, because the vehicle density is not changing in every time step. Hence, the RLE compression can reduce the length of the array considerably [39].

7 Tests and Results

The proposed enhancements of the communication protocol (see previous chapter) have been intensively tested. In this chapter, the performed tests and their results are presented.

In the first section, the comparison of the RMI and TCP speeds is discussed. The second section describes the tuning of the characteristics transfer parameters. The third section is focused on the performance of the characteristics transfer. In the fourth section, the performance of the transfer of characteristics messages and adaptive vehicle density messages is compared. Finally, in the fifth section, the performance of the adaptive vehicle density transfer in comparison with characteristics and vehicle transfer is discussed.

All described tests were performed on a cluster called Hydra, which is available at our department. The cluster incorporates one control and ten working nodes. The control node includes two processors Intel Xeon 2.8 GHz, 2 GB of RAM and 80 GB of hard disk space. Each working node then includes one processor Intel Xeon 3.2 GHz, 2 GB of RAM and 80 GB of hard disk space. All nodes are interconnected by 1 Gb Ethernet.

7.1 Comparison of the RMI and the TCP performances

The first set of tests was focused on the speed comparison of two most appropriate middleware for the distributed version of the JUTS system. As had been said in section 5.4, the RMI and the TCP sockets seem to be a good solution for the inter-process communication in the distributed version of the JUTS system. The use of the RMI makes the simulation program more transparent, but it is slower than the TCP sockets.

In the JUTS system, there are three types of objects that will be transferred via the communication links – the vehicles, the lane blocks, and the synchronization messages. The object of vehicle incorporates three integer parameters and the object of lane block incorporates only one boolean parameter (see Fig. 11). The synchronization message can be empty. So, it can be implemented as the transfer of a single character in the TCP protocol and as a single invocation of a method without arguments in the RMI protocol [28].

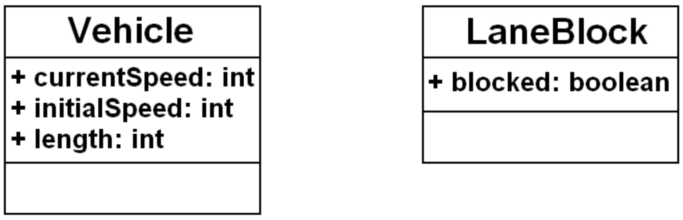

Fig. 11: The UML diagrams of the transferred objects

Of course, as has been said in section 5.2.2, all vehicles and lane blocks determined for one particular neighbouring sub-network are sent as one message. However, for the testing of the speed difference between the RMI and the TCP protocols, this is not essential.

For testing purposes, two small distributed applications in Java 1.6 were written, one with utilization of the TCP sockets and second with utilization of the RMI. Both applications were transferring all three types of objects (i.e. vehicles, lane blocks, and synchronization). The results can be seen in Table 2. All numbers are arithmetic means calculated from ten attempts.

Table 2: Comparison of the speed of the RMI and the TCP protocols

		Computation time [ms]			
	Object type \ object count	10^2	10^3	10^4	10^5
RMI	Vehicle	75	473	3081	28253
	Lane block	77	454	3022	27894
	Synchronization	47	299	2384	20556
TCP	Vehicle	21	60	210	1572
	Lane block	22	59	207	1530
	Synchronization	5	28	119	708

As can be seen in the Table 2, the communication using TCP sockets is much faster then the communication using RMI. The TCP is 3.5 times faster than the RMI in the best case scenario and even 29 times faster in the worst case scenario. On the other hand, the RMI hides the communication details and therefore is less error-prone than the TCP. A way how to exploit the advantages of both solutions is a compromise. The RMI can be used throughout the development of the distributed simulation. After the debugging of the simulation program, the slower RMI can be replaced by the faster TCP [28].

7.2 Thresholds Setting of the Characteristics Transfer

After we discussed the appropriate middleware for the distributed version of the JUTS system, we can proceed with the setting of the characteristics transfer protocol. As has been said in section 6.2.2, there are three thresholds and two constants, which need to be set.

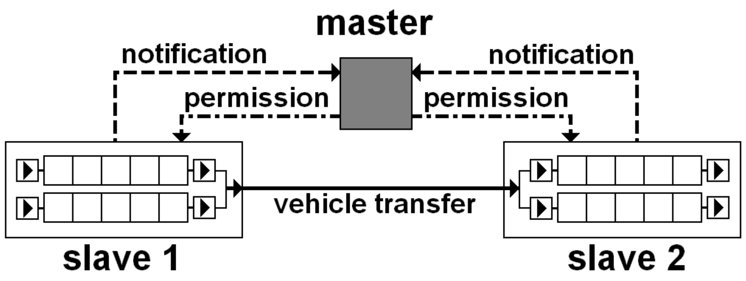

Fig. 12: The division of traffic network for tests of characteristics transfer

For all tests of the characteristics transfer, two slave processes and one master process were used. The traffic network was composed of number of parallel traffic lanes, which are conducting from slave 1 to slave 2 (see Fig. 12). There are no crossroads or runabouts in the traffic network, but the traffic network incorporates traffic lights for some tests. This simplification of a real traffic network allows us to concentrate on the performance of the communication protocol.

7.2.1 Settings of the Vehicle Density Transfer

The most important parameter is the vehicle density, because it determines the distribution of the vehicles in the traffic lanes. To optimize the transfer of vehicle density, the threshold T_d and the number of steps N_s must be determined. The threshold T_d is the difference between the actual and the last sent vehicle density. It is expressed in percents and can be calculated according to the equation:

$$T_d = \frac{|\lambda_l - \lambda_a|}{\max(\lambda_l, \lambda_a)} \cdot 100, \tag{5}$$

where λ_l is the last sent vehicle density and λ_a is the actual density.

The performance of the vehicle density transfer is determined by three values – the difference between the terminated and generated vehicles count (Δd), the delay (Δt) described in section 6.2.1, and the total number of sent characteristics C. All these values should be minimal. If we consider them as equally important, the simple product of them will give us the coefficient K_d. According to the coefficient K_d, the particular settings of the threshold T_d and the number of steps N_s can be compared.

For the tests, the traffic network with four traffic lanes was used. Two lanes were equipped by the traffic lights, which causes the periodical increase and decrease of the vehicle density. These periodical changes of the traffic flows allow us to determine the delay. To maximize the correctness of the results, the transfer of vehicles speed and lengths was suppressed [29]. The results of the tests can be seen in Table 3.

Table 3: Settings of the vehicle density transfer

N_s [steps]	T_d [%]	Δd [%]	Δt [steps]	C	K_d
5	10	9.39	3.48	165	5392
5	20	5.47	3.65	103	2056
5	30	4.10	3.22	82	1083
5	40	3.70	3.67	67	910
5	50	7.63	5.19	52	2059
10	10	8.79	3.42	163	4900
10	20	6.30	3.38	106	2257
10	30	4.27	3.35	84	1202
10	**40**	**3.19**	**3.40**	**68**	**738**
10	50	7.21	4.70	51	1728
15	10	10.26	3.19	166	5433
15	20	5.27	3.54	104	1940
15	30	4.53	3.34	83	1256
15	40	3.75	3.25	64	780
15	50	7.05	5.25	51	1886

Each line of the table corresponds to one setting of the threshold T_d and the number of steps N_s. For each setting, ten simulation runs have been performed and the particular results have been averaged. In every simulation run, the Δd and the C values were averaged from all four lanes. The *Δt* value was averaged from two lanes with traffic lights. All simulation runs were 500 time steps long.

The ranges for the threshold T_d and the number of steps N_s have been selected according to the preliminary tests. For lower values of N_s or T_d, the number of sent characteristics *C* increases considerably. For higher values of N_s or T_d, the delay (*Δt*) and the difference between the generated and the terminated vehicles count (*Δd*) increase considerably. According to the coefficient K_d, the optimal value of the threshold T_d is 40 % and the vehicle density should be calculated from last 10 time steps.

7.2.2 Settings of the Vehicle Lengths Transfer

The settings of the transfer of vehicle lengths have been determined in similar manner. The optimal value of the threshold T_l and the number of vehicles N_v, according which the vehicle lengths are calculated, have been searched. The threshold T_l is the difference between the actual and the last sent vehicle lengths. It is expressed in percents and can be calculated according to the equation:

$$T_l = \frac{\sum_{i=1}^{6} \frac{|l_{li} - l_{ai}|}{\max(l_{li}, l_{ai})}}{6} \cdot 100 \tag{6}$$

where l_{li} is the last sent portion of the vehicles with length of *i* and the l_{ai} is the actual portion of the vehicles with length of *i*. The coefficient K_l is in this case calculated as product of the delay *Δt* and the number of sent characteristics *C*.

For the tests, the traffic network was composed of one traffic lane with uniform traffic flow and random lengths of the vehicles. For five time intervals, the length of vehicles was set to one, in order to determine the delay *Δt*. As can be seen in the Table 4, the optimal value of the threshold T_l is 50 % and the vehicle lengths should be calculated from last 10 vehicles.

Table 4: Settings of the vehicle lengths transfer

N_v [vehicles]	T_l [%]	Δt [steps]	C	K_l
5	10	3.51	331	1162
5	20	3.84	257	987
5	30	5.31	153	812
5	40	5.20	124	645
5	50	23.14	82	1897
10	10	5.59	320	1789
10	20	7.63	179	1366
10	30	8.77	111	973
10	40	9.21	79	728
10	**50**	**9.39**	**57**	**535**
15	10	8.12	252	2046
15	20	10.53	143	1506
15	30	11.98	96	1150
15	40	13.32	62	826
15	50	14.81	44	652

7.2.3 Settings of the Vehicle Speed Transfer

The settings of the transfer of vehicles speed have been determined in the same way as the transfer of vehicles lengths. The optimal value of the threshold T_s and the number of vehicles N_v have been searched. The threshold T_s is the difference between the actual and the last sent vehicle speed. It is expressed in percents and can be calculated according to the equation:

$$T_s = \frac{|s_l - s_a|}{\max(s_l, s_a)} \cdot 100 \tag{7}$$

where s_l is the last sent mean speed of the vehicles and s_a is the actual mean speed of the vehicles. The coefficient K_s is in this case calculated as product of delay Δt and the number of sent characteristics C.

Again, the traffic network was composed of one traffic lane. There were performed two sets of tests. For the determination of the delay, a uniform traffic flow with gradually increasing vehicle speed was used. For the determination of the number of sent characteristics C, a real traffic flow with randomly altered vehicles speed was used. The summary of both sets of test can be seen in Table 5. The optimal value of the threshold is 10 % and the vehicle speed should be calculated from last 10 vehicles.

Table 5: Settings of the vehicle speed transfer

N_v [vehicles]	T_s [%]	Δt [steps]	C	K_s
5	10	2.76	74	204
5	20	7.19	55	395
5	30	12.71	42	534
10	**10**	**3.42**	**54**	**185**
10	20	9.83	47	462
10	30	15.37	39	599
15	10	6.83	59	403
15	20	14.00	48	672
15	30	31.57	36	1136

7.3 Performance of the Characteristics Transfer

After we determined the optimal thresholds for the characteristics transfer, we can compare the performances of the vehicle transfer and the characteristics transfer. For this purpose, two sets of tests were performed. In the first set of tests, the performances of the vehicle and characteristics transfers were tested on traffic networks with various number of traffic lanes. In the latter set of tests, the influence of the vehicle density on the vehicle or characteristics transfer was investigated.

7.3.1 Various Number of Traffic Lanes in the Traffic Network

In the first set of tests, the performance of both vehicle and characteristics transfers were tested on a traffic network with 1, 2, 4, and 8 traffic lanes. The traffic densities in the lanes were realistic (i.e. from 0.05 to 0.5 vehicles per time step) [26]. The results can be seen in Table 6. For each count of traffic lanes (L), both vehicles (V) and characteristics (C) transfer (TT) were tested. The observed parameters were the number of sent characteristics or vehicles (C/V), the total number of sent messages (M), the total amount of transferred data (D), and the total time of computation (T). The messages needed for transfer of the lane blocks (see section 5.2.2) are not considered in the tests, because their counts are approximately the same for both vehicle or characteristics transfers [29].

Table 6: Vehicle and characteristics transfer comparison for various numbers of lanes

L	TT	C/V	M	D [B]	T [ms]
1	V	294	275	170257	1180
	C	173	173	117886	1065
2	V	563	480	299943	1390
	C	298	276	191308	1199
4	V	1244	751	487518	1583
	C	638	490	351783	1474
8	V	2456	936	652689	1708
	C	1334	767	590110	1695

The communication savings caused by the use of characteristics transfer are then summarized in Table 7. The subscripts of particular savings (S) correspond to abbreviations of the parameters in Table 6.

Table 7: Communication savings summary

L	$S_{C/V}$ [%]	S_M [%]	S_D [%]	S_T [%]
1	41.16	37.09	30.76	9.75
2	47.07	42.50	36.22	13.74
4	48.71	34.75	27.84	7.39
8	45.68	18.06	9.59	0.76

As can be seen in Table 7, the number of sent characteristics is on average by 46 % smaller than the number of sent vehicles. However, the total messages count is only reduced by 33 % on average.

This “inconsistence” can be explained. As has been said in section 5.2.2, by the use of the vehicle transfer, all vehicles (and lane blocks) determined for one particular sub-network are transferred as one message. Also, in one traffic lane, more than one vehicle can be transferred to the neighbouring sub-network in one time step. This can happen if two or more vehicles in one traffic lane are traveling at high speed tight one after another. Then, they are able to pass the boundary between the sub-networks in the same time step. In that case, all these vehicles must be transferred to the neighbouring sub-network (see section 5.2.2). The probability of this phenomenon is inconsiderable. For example, in the one lane scenario of our tests, 294 vehicles have been transferred only via 275 messages. However, the described situation cannot occur by the use of the characteristics transfer. In one traffic lane, only one characteristic can be transferred to the neighbouring sub-network in one time step. This fact is also obvious in the one lane scenario of our tests – 173 characteristics have been transferred via 173 messages. For these reasons, the savings of messages count is less than the difference between the number of sent characteristics and vehicles.

Another observation is that the savings of sent messages have downtrend for the increasing number of traffic lanes. That is natural, because with the increasing number of traffic lanes, the probability, that no characteristics transfers are needed in a time step, sinks considerably. Still, even for large number of traffic lanes (eight and more), the amount of transferred data is reduced, because the transferred messages are shorter. The amount of characteristics in the messages by the characteristics transfer is approximately by 46 % smaller than the amount vehicles in the messages by the vehicle transfer regardless to the number of traffic lanes.

7.3.2 Various Vehicle Densities in the Traffic Lanes

The latter set of tests was focused on the influence of the vehicle density in a traffic lane on the resulting performance of the characteristics or vehicle transfer. There was only one lane in the traffic network, but the tests were performed for various vehicle densities (from 0.05 to 0.5 vehicles per time step). The results can be depicted as the dependence of the sent messages count on the vehicle density (see Fig. 13).

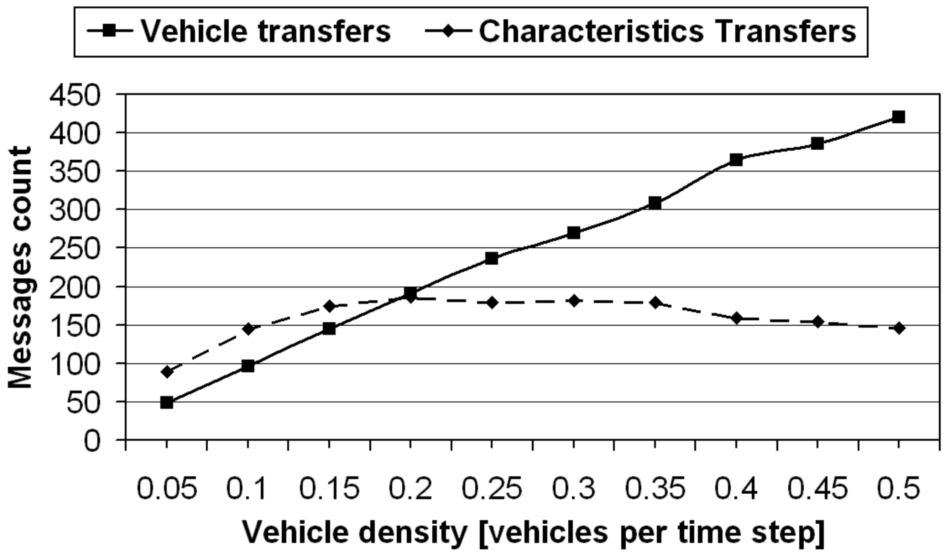

Fig. 13: Dependence of the sent messages count on the vehicle density in the traffic lane

As can be seen in Fig. 13, the characteristics transfer is less efficient than the vehicle transfer for low vehicle densities (0.2 vehicles per time step and lower). On the contrary, the characteristics transfer is more efficient for high vehicle densities (0.25 vehicles per time step and higher).

7.4 Transfer Time of Adaptive Vehicle Density Message

Now, as we finished the testing of the characteristics transfer, we can proceed with the testing of the latter enhancement of the communication protocol – the adaptive vehicle density transfer (AVDT). The first set of tests is focused on comparison of the characteristics message transfer time and the AVDT message transfer time. These tests are necessary in order to determine, whether the length of the transferred message influences the transfer time of the message significantly. The AVDT messages are longer than the characteristics messages, but we assume that the adaptive vehicle density transfer requires less messages than the characteristics transfer (see section 7.5). So, if the difference between the transfer times of longer AVDT and shorter characteristics message is negligible, then the adaptive vehicle density transfer should be more efficient than the characteristics transfer.

The tests must be performed separately from the traffic simulation in order to eliminate the influence of other simulation computations. For this reason, one hundred uncompressed arrays with densities were recorded during several simulation runs. These arrays were used as testing data. The time necessary for the RLE compression and decompression of the array is also included in the test [33].

As it was mentioned in section 5.2.2, all characteristics and lane blocks from one traffic sub-network to another are sent as one message in a time step. So, one message determined for a sub-network can contain several characteristics or AVDT information. Hence, the test was performed with messages containing one, three and five characteristics or AVDT information. In every test, 10000 messages were sent. The results are summarized in Table 8. All values are averaged from ten attempts.

Table 8: Comparison of the characteristics and adaptive vehicle density transfer times

Characteristics or ADVT count	Characteristics time [ms]	ADVT time [ms]
1	5299	5327
3	5386	5534
5	5454	5560

As can be seen in the Table 8, the differences between the transfer times of both types of messages are negligible (less than 2 % on average). So, the adaptive vehicle density transfer will be more efficient than the characteristics transfer if the saving of the messages count will be greater than 2 %.

The main reason for so little transfer time difference is the communication overhead. Besides the useful data (characteristics or AVDT information), certain amount of administrative data is transferred as well. These administrative data are utilized by the middleware, which is used for inter-process communication. In the RMI (which has been used by the tests), these administrative data are used for example for the remote propagation of the exceptions or for the remote garbage collection. In the TCP, these administrative data are used for maintaining of the socket connection. If the useful data are short (i.e. several bytes), the substantial part of the message is occupied by the administrative data. So, if the length of the short useful data increases for example two times, the increase of the total message length will be considerably smaller (depending on the amount of administrative data), as well as the increase of message transfer time.

By the test, it was also determined that the average length of the compressed array is only 13 elements. Because the length of an uncompressed array was 60 elements, the average compression ratio is 4.6.

7.5 Performance of the AVDT

Now, as we determined, that the difference of transfer times is negligible, we can compare the performances of the vehicle transfer, the characteristics transfer, and the adaptive vehicle density transfer. For this purpose, two sets of tests were performed. In the first set of tests, the performances of all three transfers were tested on the traffic lane with various traffic lights cycles. The latter set of tests was focused on the influence of the vehicle density in the traffic lane on the performances of the particular transfers.

Both set of tests were performed on the traffic network divided into two sub-networks. There was only one lane in the traffic network, which was conducting from the sub-network 1 to the sub-network 2. The vehicle density in the traffic lane was affected by nearby traffic lights. The whole situation is depicted in Fig. 14.

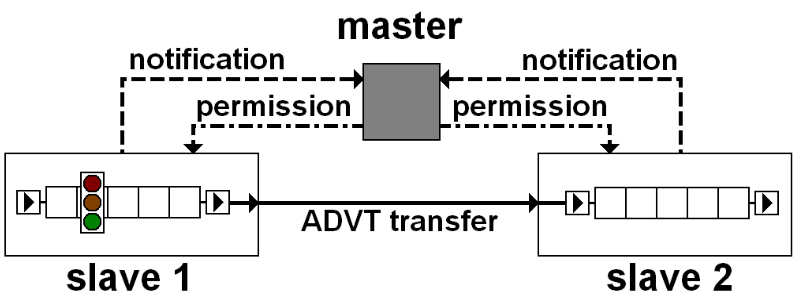

Fig. 14: The division of traffic network for testing of AVDT

7.5.1 Various Traffic Lights Cycles

First, the performances of all three transfers were tested on the traffic network depicted in Fig. 14. The vehicle density in the lane was affected by the traffic lights. Two tests were performed with various lengths of the traffic lights cycle. For the third test, the traffic lights were switched off. The results for each transfer type (TT) – the vehicle (V), characteristics (C), and adaptive vehicle density (A) transfer – are summarized in Table 9. The preset variables were the cycle length (CL) and green period length (GL). The measured variables were the difference between the terminated and generated vehicles count (Δd), the messages count (M), and the total time of the simulation run (T). All values in the table were averaged from ten simulation runs, each run was 500 steps long.

Table 9: Performances of the tested transfer types

TT	CL [s]	GL [s]	Δd [%]	M	T [ms]
V	60	20	0.00	72	759
C	60	20	4.91	55	590
A	60	20	6.89	35	571
V	120	50	0.00	79	806
C	120	50	5.37	48	568
A	120	50	7.84	32	552
V	N/A	N/A	0.00	147	949
C	N/A	N/A	2.73	68	799
A	N/A	N/A	6.41	56	647

As can be seen in Table 9, the adaptive vehicle transfer requires on average by 35 % less messages than the characteristics transfer and by 55 % less messages than the vehicle transfer for lanes with traffic lights. Even for lanes without traffic lights, the adaptive vehicle density transfer requires less messages than the characteristics transfer (by 18 % on average). However, the difference between the generated and terminated traffic flow increased by 3 % in comparison with characteristics transfer [33].

7.5.2 Various Vehicle Density in the Traffic Lane

In the latter set of tests, the influence of the vehicle density on the transfers was investigated. There was one traffic lane with traffic lights (cycle length 60 seconds, 20 seconds green period) and the test was performed for various vehicle densities in the lane (from 0.05 to 0.5 vehicles per time step). The results are depicted in Fig. 15.

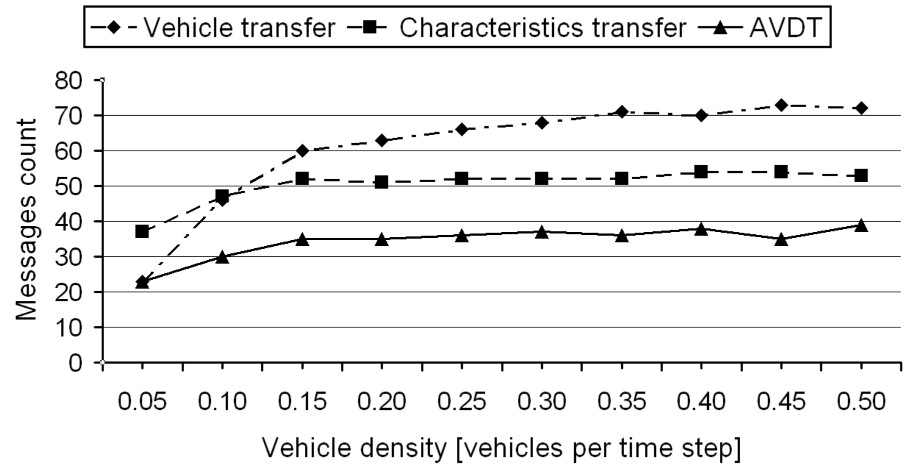

Fig. 15: Dependence of the sent messages count on the vehicle density in the traffic lane

As can be seen in the Fig. 15, the adaptive vehicle density transfer (AVDT) gives the best results for all vehicle densities in the traffic lane. Therefore, it is utilizable for any vehicle density and not only for higher vehicle densities like the characteristics transfer is (see section 7.3.2).

8 Future Work

In previous chapters, we presented the enhancements of the communication protocol for the distributed version of the JUTS system. Our current experiments have been performed only on the JUTS system in order to verify the basic theories of our research. Our future work will be focused on further development of proposed ideas and their generalization for general distributed time-stepped simulations. As a use case of the proposed communication protocol, the concrete implementation for the JUTS system will be used.

More concretely, we will continue with the development of an efficient communication protocol for general distributed traffic simulation. So far, we have dealt only with the reduction of the inter-process communication necessary for the vehicle transfer. In order to achieve additional communication savings, three adjustments of the communication protocol are proposed – the enhanced characteristics transfer (see section 8.1), the hybrid vehicle-characteristics transfer (see section 8.2), and the centralized vehicle or characteristics transfer (see section 8.3).

However, the communication necessary for the vehicle transfer cannot be entirely eliminated, because the vehicles must be able to pass between the particular sub-networks. Hence, we will utilize also another way how to improve the performance of the communication protocol, which is the reduction of the communication necessary for the synchronization. This approach is briefly discussed in sections 8.4 and 8.5.

8.1 Enhanced Characteristics Transfer

The utilization of the characteristics transfer described in section 6.1 reduces the amount of transferred data in comparison with the vehicle transfer. As has been said in section 6.2.2, all vehicles or characteristics determined for one particular neighbouring sub-network in a time step are sent as one message. So, there is at the most one message sent to one particular neighbouring sub-network per one time step.

As has been said in section 7.3.1, the number of sent characteristics is on average by 46 % smaller than the number of sent vehicles. This value is independent on the number of the traffic lanes conducting between the neighbouring traffic sub-networks. However, the savings of the sent messages count have downtrend for the increasing number of the traffic lanes. This is natural. The savings of the sent messages count occur when no characteristics transfer to the neighbouring traffic sub-network is needed in some time steps. However, for the large number of traffic lanes conducting to the neighbouring sub-network (eight and more), the number of steps, in which no characteristics transfer is needed, sinks considerably. Consequently, also the savings of the sent messages count sinks considerably.

On the other hand, the reduction of the sent messages count is important because of the communication overhead (see section 7.4). The less the messages count is, the smaller is also the accumulate communication overhead. So, it is desirable to achieve the considerable savings of the sent messages count also for the large number of traffic lanes conducting between the neighbouring traffic sub-networks. For this reason, we introduce the enhanced characteristics transfer.

8.1.1 Main Idea of the Enhanced Characteristics Transfer

By the standard characteristics transfer, the characteristics are sent only for the traffic lanes, in which the difference between the actual and the last sent characteristics reaches the threshold (see section 6.1). The need for the characteristics transfer occurs independently in particular traffic lanes. So, very often, each traffic lane needs the characteristics transfer in different time step. Consequently, the messages are sent in almost every time step and they contain only one characteristics transfer.

The basic idea of the enhanced characteristics transfer is to synchronize the characteristics transfer of the particular traffic lanes. This means that the characteristics will be sent for all traffic lanes conducting between the sub-networks, not only for the lanes, in which the difference reaches the threshold.

More precisely, if one traffic lane needs the characteristics transfer, the characteristics transfers of all remaining lanes will be sent in the same message as well. Thus, we will ensure that, in all lanes conducting between the sub-networks, the difference between the actual and the last sent characteristics is reset to zero. In that case, it is very unlikely that a characteristics transfer will be needed in next several time steps. Hence, after the time step, in which the message with all characteristics transfers is sent, there will be several time steps, in which no characteristics transfer will be needed and no message will be sent. So, the considerable savings of the sent messages count will be achieved regardless to the number of traffic lanes conducting between the neighbouring traffic sub-networks.

The described enhanced characteristics transfer will be implemented and intensively tested in order to verify its applicability.

8.2 Hybrid Vehicle-Characteristics Transfer

As has been said in section 7.3.2, the characteristics and vehicle transfers are efficient for high and low vehicle densities, respectively.

8.2.1 Main Idea of the Hybrid Transfer

In order to exploit advantages of both approaches, we can combine them into one hybrid vehicle-characteristics transfer. For traffic lanes with low vehicle densities, the traffic transfer will be used and for traffic lanes with high traffic densities, the characteristics transfer will be used [29].

Of course, the traffic density in a traffic lane can alter in time. So, it must be possible to switch between both types of transfer according to the actual traffic density in the lane. However, because our main goal is the reduction of inter-process communication, the switching between transfer types must not require additional messages.

8.2.2 Implementation of Hybrid Transfer

As has been said in section 5.2.2, all vehicles or characteristics and lane blocks determined for one neighbouring traffic sub-network are stored in the buffer. At the end of every time step, the content of the buffer is sent to target sub-network as one message. So, the message is essentially a container for vehicles or characteristics and lane blocks. For the switching between the transfer types, we can exploit the fact that the message can contain the characteristics and vehicles simultaneously. In that case, it is only necessary to implement a module for the terminator, which will decide whether use the vehicle or characteristics transfer. The decision will be based on the vehicle density in the particular traffic lane.

So, the terminator will put either the vehicle(s) information or the characteristics information into the buffer. At the end of the time step, the content of the buffer will be packed into a message and sent to the target traffic sub-network. In the target sub-network, the message will be unpacked and the vehicle(s) information or characteristics information will be forwarded to the corresponding generator. The generator will check whether a vehicle or characteristics information arrived. If vehicle information arrives, the generator will create the vehicles according to the received information and inserts it into the traffic lane. If characteristics information arrives, the generator will update the values, according which it is generating the vehicles.

8.3 Centralized Vehicle/Characteristics Transfer

For additional savings of inter-process communication, we can exploit the synchronization messages. We assume that the standard master-slave approach is used for the synchronization of the simulation. In that case, *2p* messages, where *p* is the number of slave processes, are necessary for synchronization in every time step, as stated in equation (4) (see page 29). Besides the synchronization, the slave processes communicate among themselves in order to enable the transfer of vehicles and lane blocks. This communication means additional message load per each time step.

8.3.1 Main Idea of the Centralized Transfer

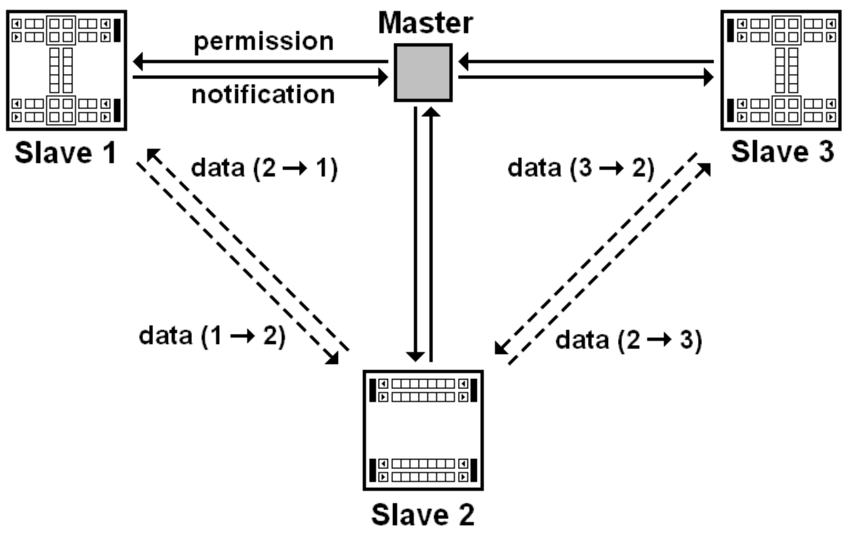

Fig. 16: Schema of standard master-slave approach

The standard schema of the master-slave approach is depicted in Fig. 16. There are bidirectional connections between the master and the particular slaves and bidirectional connections between the neighbouring slave processes. The total number of messages sent per one time step is summarized in equation (4) (see page 29). Because this number can be relatively high (for the situation in Fig. 16, the maximum number of messages per one time step is six), it is desirable to reduce it.

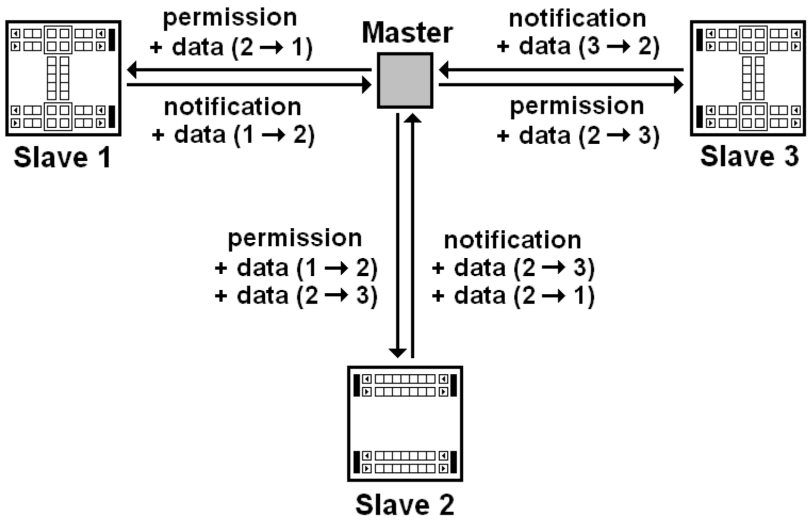

Fig. 17: Schema of centralized master-slave approach

This can be achieved by slight modification to the master-slave approach. The number of synchronization messages cannot be reduced easily. However, these messages can be used for reduction of communication among the slave processes. Instead of maintaining connections directly among the slave processes, we can utilize the connections between the master and the slave processes. The messages sent from a slave process to its neighbouring slave processes can be then “pasted” to the synchronization messages. By this approach, the master process takes the role of message router and forwards the incoming messages from the slaves pasted to the synchronization messages to the target slave processes [40]. The schema of this centralized version of master-slave approach is depicted in Fig. 17.

8.3.2 Implementation of the Centralized Transfer

In every time step, there are performed following actions:

- Every slave process performs local computation of current time step. Throughout the computations, the vehicles or characteristics and lane blocks determined for all neighbouring slave processes are stored in one buffer.
- After the computations are finished, the slave process packs the content of the buffer, inserts it into the notification message and sends this complete message to the master process.
- The master process receives the notification and stores the incorporated vehicles, characteristics, and lane blocks into its own buffer.
- After the master process receives notifications from all slave processes, it packs the corresponding vehicles, characteristics, and lane blocks to the particular permission messages and sends these messages to the corresponding slave processes.

It should be noted that the master process must reorganize the content of received and sent messages. Every message, which the master process receives, contains vehicles or characteristics and lane blocks sent from one source sub-network to all neighbours of this sub-network. However, every message, which the master process sends, must contain all vehicles or characteristics and lane-blocks from all sub-networks determined for one particular sub-network. For this reason, the contents of received messages are stored in a "global" buffer and all particular vehicles, characteristics, and lane blocks must incorporate the destination information. In that case, the master process will be able to find all vehicles, characteristics, and lane blocks determined for one particular sub-network in the buffer and send them along with the permission message.

Because there are no connections between the slaves and the entire communication is transmitted via the synchronization messages, the total number of messages is reduced to *2p* per one time step. Of course, the synchronization messages are longer, because they incorporate useful data from the slave processes. However, the transmission of less number of longer messages is more efficient than transmission of larger number of shorter messages (see section 7.4). Utilization of this approach also means larger load of the master process, but this is not a problem since the master process is idle throughout the local computations of every time step.

8.4 Semi-Asynchronous Synchronization

All adjustments of the communication protocol for distributed traffic simulation described in previous sections were based on reduction of the communication necessary for the transfer of vehicles, characteristics, and lane blocks. Another way how to achieve communication savings is the reduction of the communication necessary for the synchronization.

8.4.1 Main Idea of the Semi-Asynchronous Approach

As has been said in section 4.4, the synchronization of the simulation is necessary in order to avoid or at least handle the causality errors. In the distributed simulation of road traffic, the only causality error, which can occur, is the arrival of a vehicle or lane block in incorrect time step (past of future). So, the synchronization is necessary because of the transfers of the vehicles and lane blocks between the particular sub-networks. If we want to reduce the communication necessary for the synchronization, it is desirable to summarize the basic features distributed version of the JUTS system.

In the JUTS system, the roads are composed of unidirectional traffic lanes (see section 3.3.2). The vehicles can only move forward or stand and are allowed to pass only between the traffic lanes with the same direction. So, in the road, a vehicle moving in a traffic lane cannot influence the movement of the vehicles in the neighbouring lane with opposite travel direction. Hence, the traffic in a particular traffic lane is influenced only by the vehicles moving in the lane. This approach is common also in other traffic simulations [41]. Moreover, all simulated vehicles are limited by maximal speed, which is 6 cells per time step (54 km/h) in the JUTS system [17].

In section 5.1.2, it is said that the traffic network is divided between the crossroads, in the middle of the roads (see Fig. 18). This approach can be slightly modified. Because the traffic lanes with opposite direction do not influence each other, the division of the network can be performed also between the crossroad and its outgoing traffic lanes (see Fig. 19).

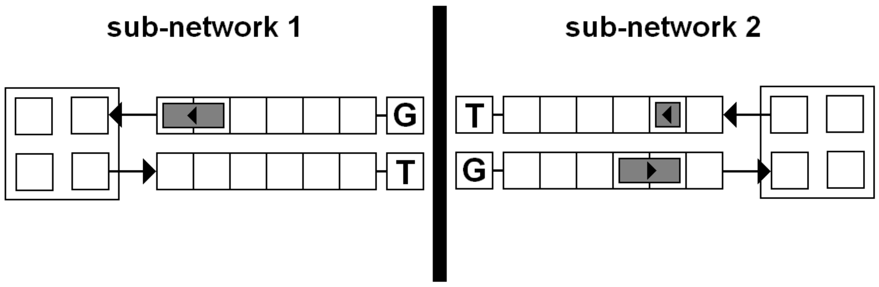

Fig. 18: The standard division of the traffic network in the JUTS system

The described features of the JUTS system, which are also common for other traffic simulators, can be utilized for significant reduction of the communication necessary for the synchronization. The basic idea is not to send the vehicles in every time step, but only once every several steps. The time period between two successive transfers of vehicles is designated as *long step*. We utilize the fact that the movement of the vehicles in the single traffic lane is affected only by the vehicles themselves. Hence, all vehicles, which shall be transferred to a neighbouring traffic sub-network throughout the long step, are stored in the buffer. After the long step period is elapsed, the content of the buffer is packed into a message and sent to the corresponding sub-network. In the target sub-network, the message is received, unpacked and the contained vehicles are forwarded to the corresponding traffic lanes. There, the vehicles are immediately inserted into the traffic lanes and shifted forward according to the distance, which they would travel in the time period of long step.

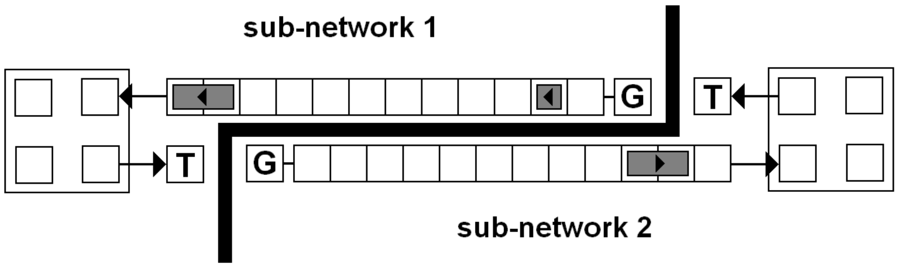

Fig. 19: The modified division of the traffic network in the JUTS system

Because the synchronization is necessary only because of the transfer of the vehicles between the sub-networks, the synchronization would be also necessary only once per long step by the vehicle transfer. The length of the long step depends on the minimal distance of two neighbouring crossroads, between which the traffic network is divided. By the division according to Fig. 19, this distance is maximized, because we utilize the length of entire traffic lanes between the neighbouring crossroads. The length of long step is expressed in time steps and can be calculated as:

$$T_{ls} = \frac{d_{\min}}{v_{\max}}, \tag{8}$$

where T_{ls} is the length of the long step, d_{min} is the minimal distance between the crossroads on the boundary of two sub-networks, and v_{max} is the maximal speed of the vehicles. For example, if the minimal distance between the crossroads on the boundary of two sub-networks is 150 meters (i.e. 60 cells) and the maximal speed of the vehicles is 6 cells per time step, the length of the long step is 10 steps.

It should be noted that the semi-asynchronous long step method has nothing to do with the super-stepping presented in [42]. The basic idea of the super-stepping is to reduce the communication necessary for synchronization in time periods, when only little events occurs. The length of the super-step is variable and must be calculated by the particular simulation processes. This brings additional overhead to the simulation. Moreover, the messages with events are transferred among the simulation processes throughout the super-steps. For more information, see [42]. On the other hand, the semi-asynchronous long step method presented in this section utilizes the equal-sized long steps, whose length is determined only once at the beginning of the simulation based on the traffic network division. Moreover, there is no communication among the particular processes in the time period of the long step. The only communication is performed at the boundaries of particular long steps.

8.4.2 Implementation of the Semi-Asynchronous Approach

As has been said, the synchronization is necessary only by the transfer of vehicles once every long step. So, the synchronization messages are sent in the same time as the messages with vehicles. Hence, the utilization of the master-slave protocol with centralized vehicle transfer is most appropriate (see section 8.3). In this case, the vehicles are transferred by utilization of the synchronization messages. Moreover, the only thing, which must be modified, is the insertion of the received vehicles in the traffic lane, because the communication protocol is already designed for transfer of more vehicles per time step in one traffic lane (see section 5.2.2).

8.4.3 Advantages and Disadvantages

The most important advantage is the communication savings. If we use the master-slave protocol with centralized vehicle transfer and the length of the long step would be 10 steps, the communication is reduced to $2p$ messages per long step and $2p/10$ messages per time step (p is the number of slave processes). This means the 90 % savings of the message count. Of course, the messages will be considerably longer, which means the increasing of the transfer time of every message, but not dramatic (see section 7.4).

Another advantage of the long step method is that, unlike the characteristics or adaptive vehicle density transfers, the terminated and generated traffic flows are identical. If a vehicle is terminated in the source traffic sub-network and transferred to the target sub-network, it will be generated and shifted into the correct position. So, after the long step period, the transferred vehicle is in the position, in which it would be if the traffic lane were not divided. In characteristics or adaptive vehicle density transfers, the terminated and generated traffic flows have indeed the same traffic characteristics, but they are not identical. If a vehicle is terminated in the source sub-network, it does not have to be immediately generated in the target sub-network (see section 6.2.1). From this point of view, the characteristics or adaptive vehicle density transfer is comparable to the lossy data compression and the long step method to the lossless data compression.

The main disadvantage of the long step method is the difficult handling of the lane blocks. So far, we assumed that the traffic lanes in the target sub-network, in which the transferred vehicles are inserted after the long step period, are empty. If there is a vehicle queue, the incoming vehicles need not to fit into the lane. So, the modified lane-block messages are needed, which will inform the source traffic sub-network about the available space in the traffic lane. Still, because of the length of the long step period, the handling of the lane-block messages can be problematic. The extensive test will be performed to determine the usability of the semi-asynchronous long step method.

8.5 Optimistic Synchronization

Another way, which we want to explore, is the optimistic synchronization of our distributed time-stepped traffic simulation. As has been said in section 4.4.1, with the optimistic approach, the causality errors are possible, but a mechanism is provided for detection and reparation of these errors. So far, this approach is used in the event-driven simulations only.

8.5.1 Usability in the Distributed Time-Stepped Simulation

Generally, the mechanism for detection and reparation of the causality errors is application dependent and brings significant overhead to the simulation [2]. Thus, the optimistic approach is utilizable only if the causality errors are rare.

There are two types of causality errors, which can occur in the distributed version of the JUTS system – the late arrival of the vehicles and the early arrival of the vehicles. First, we will discuss the late arrival of the vehicles. We can assume that the traffic network is not divided in the middle of traffic lane, but between the crossroads and their outgoing lanes (see Fig. 19). In that case, if the delay of the vehicles is less than the length of long step expressed in equation (8), the vehicles can be inserted to the traffic lane and shifted forward according to their delay. The problem occurs, if the delay of the vehicles is larger than the length of the long step. In that case, the delayed vehicles could have influenced the traffic on nearby crossroad, but they did not, because of the delay. In that case, an error is introduced in the simulation.

The early arrival of the vehicles is less problematic than the late arrival of the vehicles. The vehicles that arrive earlier can be stored in buffer and inserted to the corresponding traffic lanes in appropriate time-step. From this point of view, the time period of the early arrival is irrelevant if the buffer has sufficient capacity.

As arise from previous paragraphs, a complex mechanism for the recovery from a causality error will be necessary only for the extremely late arrival of the vehicles. In other cases, the errors can be handled by simple modification of the insertion of the vehicles into the traffic lanes. So, it seems that the optimistic approach might be applicable for the distributed version of the JUTS system and also for similar time-stepped traffic simulators.

8.5.2 Communication Savings

As has been said in previous section, the optimistic approach might be applicable for the distributed time-stepped traffic simulation. However, the elimination of the synchronization messages is not sufficient saving of the inter-process communication if the vehicles are transfer whenever they arrive to the boundary between two sub-networks. In order to reduce the communication necessary for the vehicle transfer, the aggregation technique similar to that one used in the semi-asynchronous long step method will be used (see section 8.4). So, the vehicles from multiple steps determined for one particular sub-network will be aggregated into one message. In that case, the overall inter-process communication can be reduced considerably.

Although the implementation of an efficient optimistic protocol for distributed time-stepped traffic simulation seems to be feasible from previous paragraphs, all depends on the mechanism for the reparation of the causality errors caused by the extremely late arrivals of the vehicles. After an efficient mechanism will be invented, we can proceed with the optimistic protocol implementation and testing. Only the intensive testing of the resulting simulation can verify the applicability of the optimistic approach for the distributed traffic simulation.

9 Conclusion

In this work, the distributed simulation of the road traffic was discussed. We described main aspects of the general computer simulation and the specific features of the computer simulation in the field of road traffic. Then, we proceeded with the description of the main issues of the general distributed simulation – the decomposition of the simulation into particular process and the synchronization of these processes. We believe that this part of the work gives a brief survey of the main issues of the distributed simulation and the most common approaches to the solving of these issues.

In latter part of the work, the issues of the distributed simulation in the field of the road traffic are discussed. The decomposition of the simulation, the inter-process communication, and the synchronization of the processes were briefly described. Besides the existing solutions to the issues of the distributed traffic simulation, which are commonly used in various traffic simulators, we described also the approaches, which are used in the distributed version of the JUTS system. This simulator of urban road traffic is used for our current and future experiments.

In another part of the work, we focus on the reduction of the inter-process communication in the distributed traffic simulation. Currently, two adjustments of the communication protocol were proposed, implemented, and tested – the characteristics transfer and the adaptive vehicle density transfer. Both adjustments were described in detail. There were intensive tests performed in order to determine the performance of both adjustments. The results of these tests are also presented in this work.

The characteristics transfer is based on transfer of traffic flow characteristics instead of transfer of particular vehicles. It reduces the communication by 33 % on average and is efficient mainly for high traffic densities.

The adaptive vehicle density transfer also utilizes the transfer of traffic flow characteristics. Moreover, the learning and prediction of the periodical behavior of the vehicle density is utilized for the additional savings of the inter-process communication. This periodical behavior is typical for the traffic lanes equipped by traffic lights. For these lanes, the adaptive vehicle density transfer reduces the communication by 35 % in comparison with characteristics transfer and by 55 % in comparison with the vehicle transfer.

In the last part of this work, our future research topics are briefly discussed. Our final goal is the development of an efficient communication and synchronization protocol for the distributed traffic simulation, which would be applicable also for general distributed time-stepped simulations. In first stage, we will work on other adjustments of the communication protocol, which will cause the additional savings of the inter-process communication necessary for the vehicle transfer. We plan to use the enhanced characteristics transfer, hybrid vehicle/characteristics transfer, and the centralized vehicle transfer for this purpose. Then, we want to focus on the synchronization of the simulation with the aim of the reduction of the synchronization messages count. For this purpose, the possibilities of the semi-asynchronous approach and the optimistic approach to the synchronization will be explored. The results of this research will be summarized in the doctoral thesis.

References

[1] J. A. Hamilton, U. W. Pooch, and D. A. Nash: *Distributed Simulation*, CRC Press, New York, 1996

[2] R. M. Fujimoto: *Parallel and Distributed Simulation Systems*, John Wiley & Sons, New York, 2000

[3] M. H. Lighill and G. B. Whitman: "On kinematic waves II: A theory of traffic flow on long crowed roads", *Proceedings of the Royal Society of London*, s. A, 229, London, 1955, pp. 317-345

[4] C. Braban-Ledoux: *METACOR – A Macroscopic Modelling Tool for Corridor Application to the Stockholm Test Site*, Final Report CRT2000:05, Center for Traffic Engineering & Traffic Simulation, Kungl Tekniska Högskolan, Valhallavägen, 2000

[5] T. Potuzak: "Current Trends in Distributed Traffic Simulation", *Proceedings of 41th Spring International Conference MOSIS '07 – Modelling and Simulation of Systems*, Roznov pod Radhostem, 2007, pp. 34-41

[6] T. Nagatani: "Gas Kinetic Approach to Two-Dimensional Traffic Flow", *J. Phys Soc Jap*, 65(10), 1996, pp. 3150-3152

[7] L. Nizzard: *Combining Microscopic and Mesoscopic Traffic Simulators*, Raport de stage d'option scientifique Ecole Polytechnique, Paris, 2002

[8] P. Wagner and I. Lubashevsky: "Empirical Basis for Car Following Theory Development", *eprint arXiv:cond-mat/0311192v1*, 2004

[9] M. Schreckenberg, L. Neubert, and J. Wahle: "Simulation of Traffic in Large Road Networks", *Future Generation Computer Systems*, 17, 2001, pp. 649-657

[10] D. Ni: "2DSIM: A Prototype of Nanoscopic Traffic Simulation", *Proceedings of Intelligent Vehicles Symposium*, 2003, pp. 47-52

[11] K. Nagel and M. Rickert: "Parallel Implementation of the TRANSIMS Micro-Simulation", *Parallel Computing*, 2001, pp. 1611-1639

[12] R. Klefstad, Y. Zhang, M. Lai, R. Jayakrishnan, and R. Lavanya: "A Scalable, Synchronized, and Distributed Framework for Large-Scale Microscopic Traffic Simulation", *The 8th International IEEE Conference on Intelligent Transportation Systems*, Vienna, 2005

[13] P. T. R. Wang, W. P. Niedringhaus: "Distributed/Parallel Traffic Simulation for IVHS Application", *Proceedings of the 25th Winter Conference on Simulation*, Los Angeles, 1993, pp. 1225-1230

[14] N. Cetin, A. Burri, K. Nagel: "A Large-Scale Agent-Based Traffic Microsimulation Based on Queue Model", *Proceedings of 3rd Swiss Transport Research Conference*, Monte Verita, 2003

[15] U. Klein, T. Schulze, S. Strassburger, H. Menzler: "Distributed Traffic Simulation Based on the High Level Architecture", *Proceedings of Simulation Interoperability Workshop*, Orlando, 1998

[16] K. Nagel and M. Schreckenberger: "A Cellular Automaton Model for Freeway Traffic", *Journal de Physique I*, 2, 1992, pp. 2221-2229

[17] D. Hartman, "Leading Head Algorithm for Urban Traffic Model", *Proceedings of the 16th International European Simulation Symposium ESS*, Budapest, 2004, pp. 297-302

[18] D. Hartman and P. Herout: "Implementation of Head Leading Algorithm in Simulation", *Proceedings of the 8th International Conference on Computer Modelling and Simulation UKSim*, Oxford, 2005, pp. 46-51

[19] D. Hartman and P. Herout: "Head Leading Algorithm and GIS Data Analysis in Simulation of Traffic in Pilsen", *International Journal of Simulation Systems, Science, and Technology*, Nottingham Trend University, 2005, pp. 10-17

[20] D. Hartman: "Testing of JUTS and Construction of Hybrid Traffic Simulation Model", *Proceedings of 20t European Conference on Modelling and Simulation (ECMS)*, Bonn, 2006, pp. 214-219

[21] D. Hartman and P. Herout: "Construction of a Hybrid Traffic Model Based on JUTS Cellular Model", *Proceedings of the 6th EUROSIM Congress on Modelling and Simulation*, Ljubljana, 2007

[22] K. Jezek, P. Matejovic, and S. Racek: *Parallel Architectures and Programs*, University of West Bohemia Publishing, Pilsen, 1999 (in Czech)

[23] A. L. Lastovetsky: *Parallel Computing on Heterogeneous Networks*, Wiley-Interscience, New York, 2003

[24] G. Tel: *Introduction to Distributed Algorithms*, Cambridge University Press, Cambridge, 2001

[25] P. G. Gonnet: *A Queue-Based Distributed Traffic Micro-Simulation*, 2001

[26] T. Potuzak: *System for Processing of Urban Traffic Intensity Data*, Master Thesis, University of West Bohemia, Pilsen, 2006

[27] T. Kiesling and J. Lüthi: "Towards Time-Parallel Road Traffic Simulation", *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*, Monterey, 2005, pp. 7-15

[28] T. Potuzak: "Selection of Middleware for Distributed Traffic Simulation", *The 1st Young Researches Conference on Applied Sciences – Conference Proceedings Book*, Pilsen, 2007, pp. 16-21

[29] T. Potuzak: “Distributed Traffic Simulation and the Reduction of Inter-Process Communication Using Traffic Flow Characteristics Transfer”, *EUROSIM/UKSim 2008*, to be published

[30] W. Grosso: *Java RMI*, O’Reilly, New York, 2001

[31] F. Kuhl, R. Weatherly, and J. Dahman: *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Prentice Hall, 1999

[32] T. Potuzak and P. Herout: “Use of Distributed Traffic Simulation in the JUTS Project”, *Proceedings of EUROCON 2007*, Warsaw, 2007, pp. 2250-2255

[33] T. Potuzak: “Adaptive Transfer of Vehicle Density in the Distributed Traffic Simulation”, *HSI’08*, submitted for publication

[34] C. L. Giles, S. Lawrence, and A. C. Tsoi: “Noisy Time Series Prediction using a Recurrent Neural Network and Grammatical Interference”, *Machine Learning*, 44, Number 1/2, 2001, pp. 161-183

[35] J. M. Zurada: *Introduction to Artificial Neural Systems*, West Publishing Company, New York, 1992

[36] M. Rosen-Zvi, I. Kanter, and W. Kinzel: “Time Series Prediction by Feedforward Neural Networks – Is It Difficult?”, *J. Phys. A: Math. Gen.*, 36, 2003, pp. 4543-4550

[37] R. J. Frank, N. Davey, and S. P. Hunt: “Time Series Prediction and Neural Networks”, *Journal of Intelligent and Robotic Systems*, 31, 2001, pp. 91 – 103

[38] Ministry of Transport of Czech Republic: *Lights Signal Systems Design for Road Traffic Control – Technical Specification*, CVD Brno, Brno, 1996 (in Czech)

[39] D. Solomon: *Data Compression: The Complete Reference*, Springer, New York, 2004

[40] T. Potuzak: “Distributed Simulation and its Utilization in Modelling of Digital Circuits”, *Computer Architectures and Diagnostic*, Pilsen, 2007, pp. 117-122

[41] J. Barcelo, J. F. Ferrer, D. Garcia, M. Florian, and E. Le Saux: “The Parallelization of AIMSUN2 Microscopic Simulator for ITS Applications”, *Proceedings of 3rd World Congress on Intelligent Transportation Systems*, Orlando, 1996

[42] S.C. Tay, G. S. H. Tan, and K. Shenoy: “Piggy-Backed Time-Stepped Simulation with ‘Super-Stepping’”, *Proceedings of the 2003 Winter Simulation Conference*, Miami, 2003, pp. 1077-1085