IP Research assignment
University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Design and Analysis of Communication Systems (DACS)

Cooperative Adaptive Cruise Control model study based on traffic & network simulation

Chenxi Lei

January 2011

Supervisors:
Dr.ir. Georgios Karagiannis
M.Sc. Martijn van Eenennaam
M.Sc. Wouter Klein Wolterink

Table of Contents

1 Introduction

In this section an introduction is given to car-to-car communication networks and Adaptive Cruise Control (ACC) and Cooperative-ACC (CACC). Furthermore, a more specific background related to the project is provided, which includes the problem statement and the research questions.

1.1 General Background

Every day, our cars are using more computing technologies, primarily for safety reasons. As one of the pioneers, the UCLA Vehicular Network Lab was established to turn cars into wireless network nodes facing the traffic problems in LA City [Escu08].

Car-to-car communication changes the role of vehicles from mere transportation means to "smart objects". According to [EiSc06]: "Car-to-car communication enables many new services for vehicles and creates numerous opportunities for safety improvements". For example, it can be used to realize driver support and active safety services like collision warning, up-to-date traffic and weather information or active navigation systems. With such benefits, researchers are motivated to study the behaviours of vehicles and vehicular networks. Car-to-car communication networks are also denoted as vehicular networks. Two types of vehicular networks can be distinguished. Vehicle to Vehicle (V2V) and Vehicle to infrastructure (V2I). V2I is related to the communication between vehicles and a fixed communication infrastructure and V2V is related to the communication between vehicles. VANET is representing (1) the communication between vehicles Vehicle to Vehicle (V2V) and (2) the communication between vehicles and road side units (RSUs) when they are using the same ad-hoc wireless technology, such as IEEE 802.11p [IEEE802.11p-2010]. An RSU is a fixed base station that is located along the side of roads. The vehicle module that is supporting the communication of a vehicle with (1) other vehicles, (2) with RSUs and (3) with the infrastructure is denoted as OBU (On Board Unit). Figure 1 shows a scenario, where a car accident occurred in an intersection, and where VANET is used as a V2V communication network to inform vehicles in the neighbourhood about this accident.

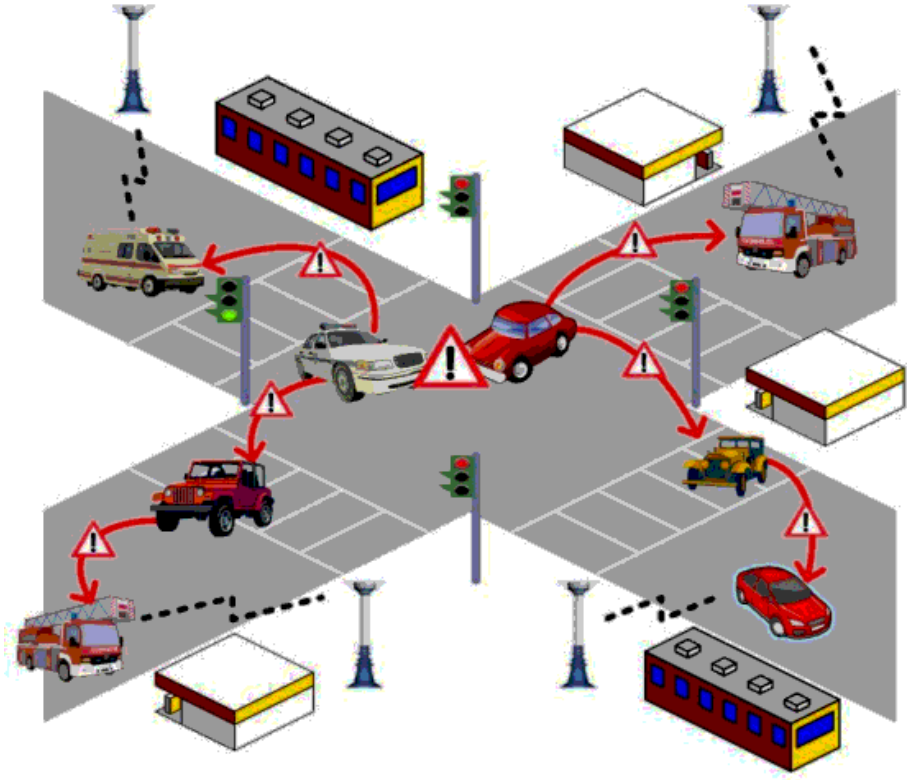


Figure 1: Example of accident in Intersection with VANET, copied from [POSTECH]

VANET turns every participating car into a wireless router or node, allowing cars approximately 100 to 300 meters of each other to connect and, in turn, create a network with a wide range. As cars fall out of the signal range and drop out of the network, other cars can join in, connecting vehicles to one another so that a mobile Internet is created.

1.2 Project-specific Background

The traffic density on the roads of most industrialized countries keeps increasing. This increase in traffic density is also increasing the traffic congestion on the roads, which will have a significant negative effect on travel time, traffic safety air pollution, and energy consumption. A possible solution to this problem is to use the Adaptive Cruise Control (ACC) concept. Initially, ACC was developed to increase user comfort, but research activities have shown that ACC could indeed have a positive impact on traffic safety and efficiency [WiKl07]. By extending the Cruise Control system with a radar sensor, ACC allows a vehicle to maintain a preset speed, as well as to adapt its speed to the speed of its predecessor in order to keep a minimum distance from its predecessor. In order to maintain these conditions a vehicle may accelerate when the preceding vehicle is increasing its speed and it slows down when it is approaching a vehicle that is driving with a lower speed than its own. An enhancement on the ACC concept is the Co-operative ACC (CACC), where the OBU in a vehicle is using a communication medium to communicate with OBUs available in other vehicles or RSUs. The information that is communicated and received by a vehicle is including vehicle dynamics information and general traffic information ahead, such as speed, acceleration, and position of other vehicles. Typically, this communicated information is denoted as CACC traffic information. The CACC traffic information can be used to enhance the performance of the current ACC systems. It is expected that CACC will increase vehicle traffic efficiency and traffic stability [WiKl07], [ArTa03]. CACC can be applied in traffic applications such as co-operative following [ArTa03], or vehicle platooning [Ioan97], [ReMi02].

The most important feature of vehicular networks and in particular, VANETs is the high mobility of nodes. Another parameter that needs to be considered is associated with the location of road side units (RSUs) and other intersection equipment, which has to be determined accurately. Therefore, it is important to study and model the mobility of vehicles, which should be carefully applied when evaluating any suitable network protocol [DjSo08].

With many years of research and design activities in this field, the technology still poses many challenges in the network and wireless transmission part, like efficient message dissemination, network scalability, and information security mechanisms.

In this assignment, more attention will be paid to the impact of the communication medium on the vehicular traffic part.

This assignment is realized in the context of the project "connect&drive" [C&D], which is developing a CACC solution by enhancing the ACC functionality already available in vehicles with wireless communication, coordination and cooperation between vehicles mutually, and vehicles and infrastructure combined. This CACC solution, see [NaVu09], uses radar to measure the distance and relative speed between vehicles. By decreasing the relative distance, a high throughput can be achieved and for the heavy duty vehicles, the drag force can be decreased to lower the fuel consumption. Moreover, CACC is using the vehicular communication network in combination with longitudinal control, allowing for anticipation to emerging shock waves, and minimizing the occurrence and the length of traffic jams. The performance measure that is usually used to quantify the anticipation to shock waves is denoted as traffic flow stability or string stability, see e.g., [PuAr10].

1.3 Problem Statement

As already mentioned, traffic flow stability, or string stability is a performance measure used to quantify the anticipation to emerging traffic shock waves. In the context of vehicle platoon, see e.g., [Ioan97], [ReMi02], string stability is defined as the traffic flow stability measure that is measuring the propagation of a traffic shock wave, caused by a disturbance from one vehicle to other following vehicles in the same platoon. If the magnitude of this shock wave grows as it propagates from the leading to the following vehicles, then the platoon is said to be unstable (or string-instable) [PuAr10]. Guaranteeing the string stability is important. For example, for a platoon of vehicles of the same speed, if there's some incident happened to the leading vehicle like deceleration, with string stability not guaranteed, the following vehicle would decelerate more than necessary (because the shock wave grows as stated above). This will decrease the speed of the involved vehicles and may cause a traffic jam and may lower the traffic throughput. Moreover, an incidental and unnecessary acceleration might cause accidents, i.e., vehicle crashes. Actually, manual driving and cruise control cannot guarantee string stability, see [PuAr10].

Advanced Driver Assistance (ADA) systems are systems that support a driver in his driving tasks. An example of an ADA system that is commercially available is the Adaptive Cruise Control (ACC) system: by extending a 'regular' cruise control system with a radar sensor, the vehicle can maintain a preset speed, but also adapt the speed to a slower predecessor [WiKl07]. However, according to [NaVu09], this ACC system cannot achieve good string stability, either, while by using CACC the string stability can be significantly improved.

Therefore, the main goal of this report is to investigate what will be the impact of an ACC and of a CACC that uses a realistic communication medium, on the string stability performance.

In this report, a combined simulation environment will be considered, This combined model includes: (1) a Simulink environment [Matlab_Simulink], used to simulate the vehicle and CACC behaviour, (2) SUMO [SUMO] traffic environment used to simulate the mobility behaviour of vehicles, (3) the MIXIM / OMNET ++ environment, used to simulate the communication networking behaviour.

1.4 Research Questions

The main research question is: What is the impact of an ACC and of a CACC that uses a realistic communication medium, on the string stability performance?

The research questions that have to be answered by this assignment are:

1) How do ACC and CACC operate?

The research approach used to answer this research question is literature study.

2) Which vehicle model that includes ACC and CACC, traffic mobility model and communication networking model could be used in this assignment?

The research approach used to answer this research question is to investigate, design and implement the combined vehicle model, traffic mobility and communication networking models within the SUMO, Simulink and OMNET++ simulation environments.

3) Which experiments should be performed in order to investigate the impact of an ACC and of a CACC that uses a realistic communication medium, on the string stability performance?

This research question is answered by designing, performing and analyzing the experiments that are needed to quantify and compare the impact of ACC and CACC on the string stability.

4) How the loss of CACC traffic influences the CACC performance from the point of view of string stability?

This research question is answered by designing, performing and analyzing experiments that quantify the impact of the CACC traffic losses, i.e., one or more beacons that are carrying CACC traffic are lost, on the CACC performance from the point of view of string stability.

1.5 Outline of this report

This report is organized as follows. In section 2, the control theory used by the ACC and CACC controllers will be described. This section is partially used to answer the first research question. Section 3 describes the used simulation environments and models used in this assignment. This section is mainly answering the second research question. Section 4 describes the performed experiments and analyses the obtained results. This section is mainly answering the third and the fourth research questions. Finally, Section 5 concludes and provides recommendations for future activities. It is important to note that three Appendices accompany this report, Appendix A, Appendix B and Appendix C. Appendix A is included in this report, while Appendix A and Appendix B are not. The reason of this is that Appendix B and Appendix C include TNO confidential information that cannot be made public.

2 Control theory and string stability

In this section, the control theory related to this assignment is introduced. To achieve string stability, a specific control structure is designed which is based on [NaVu09]. First the concept of string stability is introduced and then the ACC and CACC control structure is briefly described.

2.1 Control Theory

Control theory [Kuma10] is an interdisciplinary branch of engineering and mathematics, which deals with the behaviour of dynamical systems. The desired output of a system is called the reference. When one or more output variables of a system need to follow a certain reference over time, a controller manipulates the inputs to a system to obtain the desired effect on the output of the system.

Cruise Control (CC), see e.g., [WiKl07], can be considered as a control system that automatically controls the speed of a motor vehicle. The system takes over the throttle of the car to maintain a steady speed as set by the driver. It is useful in long drives by reducing driver fatigue and can also be used to avoid unconsciously violating speed limits.

Consider a car's cruise control, which is designed to maintain a constant vehicle speed with the desired or reference speed provided by the driver. Furthermore, consider that the system is the vehicle. The system output is then the vehicle speed, and the control variable is the engine's throttle position which influences engine torque output.

In ACC or CACC controllers, the references are the desired speed and the desired distance between vehicles. The system output and control variable are the same as the ones used for the CC controller.

A primitive way of implementing CC is simply to lock the throttle position when the driver engages cruise control. However, on mountain terrain, the vehicle will slow down going uphill and accelerate going downhill. In fact, any parameter different from what was assumed at design time will translate into a proportional error in the output velocity, including exact mass of the vehicle, wind resistance, and tire pressure. This type of controller is called an open-loop controller because there is no direct connection between the output of the system (the vehicle's speed) and the actual conditions encountered. That is to say, the system does not and cannot compensate for unexpected forces.

The ACC and CACC controllers can be considered as closed-loop control systems: a sensor monitors the output (the vehicle's speed) and feeds the data to a computer which continuously adjusts the control input (the throttle) as necessary to keep the control error to a minimum (that is, to maintain the desired speed). Feedback on how the system is actually performing allows the controller (vehicle's on board computer) to dynamically compensate for disturbances to the system. In our assignment, the disturbance is caused by other traffic’s behaviour, such as preceding vehicle’s suddenly deceleration. An ideal feedback control system should be able to cancel out all errors, effectively mitigating the effects of any forces that might or might not arise during operation and producing a response in the system that perfectly matches the user's wishes. In reality, this might be difficult to be achieved taking measurement errors in the sensors, delays in the controller, and imperfections in the control input into consideration.

2.2 String Stability

The term string stability is often used interchangeably with platoon stability in this area, which means any

nonzero position, speed, and acceleration errors of an individual vehicle in a string do not amplify when they propagate upstream, see e.g., see [PuAr10], [BoIo01]. The movement direction of vehicles, or of a string of vehicles, is considered to be the downstream direction. This means that the upstream direction is the direction from the leading vehicle towards the last following vehicle within the string.

A simple scenario which can be used to explain string stability is showed in Figure 2 and Figure 3.

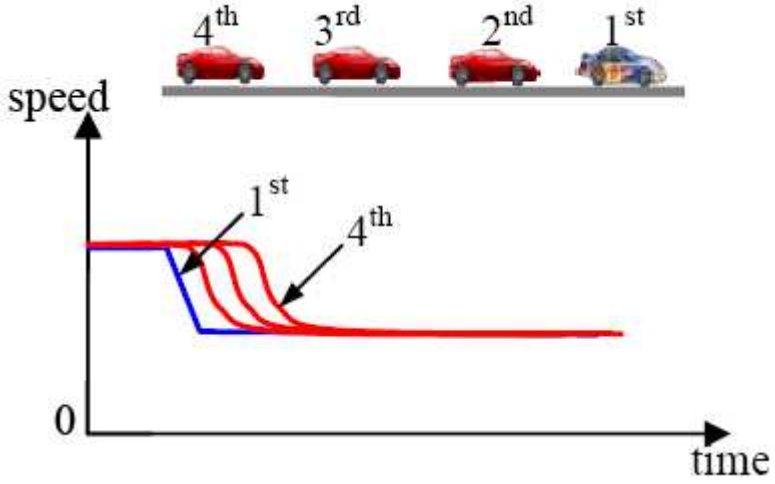


Figure 2: Platoon stability: stable, copied from [PuAr10]

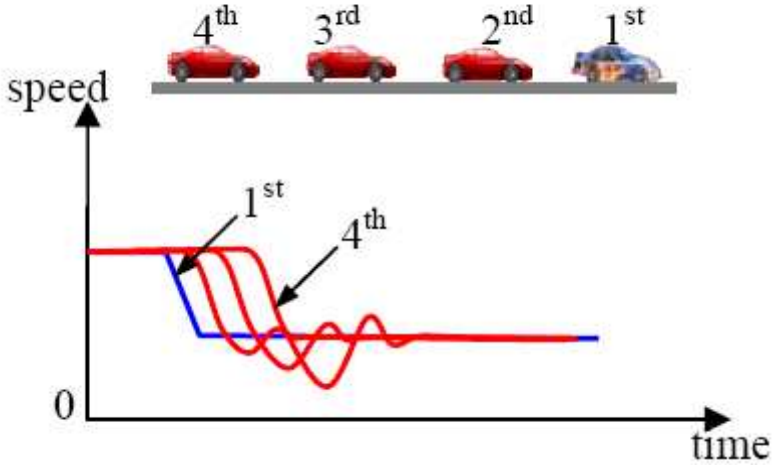


Figure 3: Platoon stability: instable, copied from [PuAr10]

In Figure 2 and Figure 3, a string of four vehicles moving from left to right is shown. The leading vehicle is denoted as 1st while the last vehicle is denoted as 4th. The direction of the moving (from left to right) is called the downstream direction, while the opposite direction which is from the leading vehicle to the last vehicle (from right to left) is called upstream direction. In each of these figures, below the shown string of vehicles, a speed vs. time coordinate graph is shown. As time goes by, the leading vehicle decelerates linearly and we can see different response of the following vehicles in the platoon depending on whether the platoon is string stable or not.

In Figure 2, the situation is shown where the platoon is string stable: the deceleration of the leading vehicle is not amplified through the following vehicles and the deceleration of following vehicles' is smooth without any fluctuation of the speed. While in Figure 3, the platoon is considered of being not string stable (string in-stable): the following vehicles decelerate even more than the leading vehicle. Though finally, the speed of following

vehicles approach to the leading vehicle's speed, their speed fluctuate a lot. Actually, during the period of fluctuation, the distance of neighbouring vehicles' also fluctuate, when collisions are more likely to happen, in other words, safety is worse.

String stability can be guaranteed if the information of the platoon leader and the preceding vehicle is used in the feedback loop, and the information of the platoon leader and the preceding vehicle can be collected by communication. CACC, which is different from ACC with the main feature of communication included, is treated as a solution to achieve a desired distance with string stability.
As we have seen in this section, the string stability can be measured using vehicle‘s speed. However, also other measures can be used for this purpose, such as traffic throughput and acceleration.

2.3 Control Structure

Though, many solutions exist to implement the CACC controller, we will focus on the control structure designed by Naus et al as stated in [NaVu09], due to the fact this structure is developed within the Connect &Drive [C&D] project.

For a string of vehicles, the primary control objective is to follow the preceding vehicle at a desired distance $\mathbf{x_{r,d,i}(t)}$, see Eq. 1.

$$\mathbf{x_{r,d,i}(t)} = \mathbf{r_i} + \mathbf{h_{d,i}\dot{x}_i(t)}, \text{ for } i>=1 \qquad \text{Eq. 1}$$

Where $\mathbf{r_i}$ is the desired distance at standstill, $\mathbf{h_{d,i}}$ is the so-called desired time headway, and $\mathbf{\dot{x}_i(t)}$ is the velocity of vehicle i. The time headway is the time it takes for vehicle i to reach the current position of its preceding vehicle i − 1 when continuing to drive with a constant velocity. The above equation Eq.1, is referred to as the spacing policy dynamics. The available measurement data include the output of the radar, which is used by a standard ACC controller. Furthermore, the acceleration of the preceding vehicle that is used in a feed-forward setting can be provided by using a wireless communication medium. Suppose we have a string of three vehicles, the platoon leader is assumed to follow a given time-varying reference position $\mathbf{x_0(t)}$ and the resulting control setup is depicted in Figure 4.

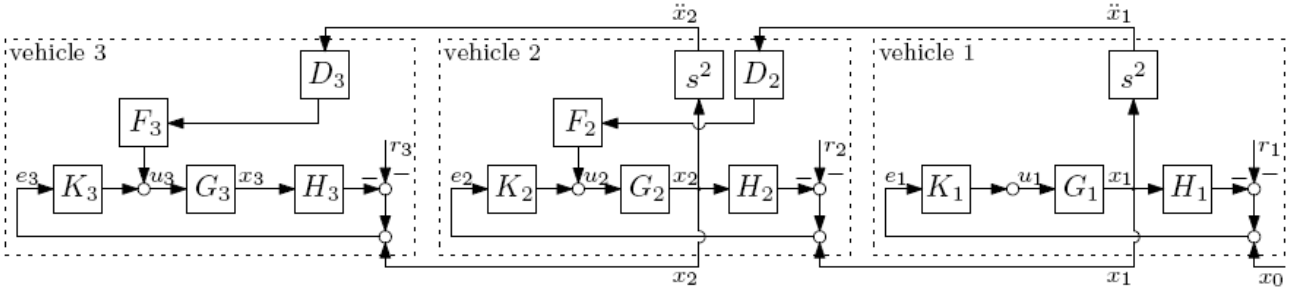


Figure 4: Control structure of a three-vehicle platoon, where G_i represents the dynamics of the i^th vehicle, K_i the corresponding ACC feedback controller, F_i the feedforward controller, D_i the communication delay and H_i = 1+h_(d,i), s the spacing policy dynamics, for i = 1, 2, 3, copied from [NaVu09]

The acceleration of the preceding vehicle is used as a feedforward control signal via a feedforward filter $\mathbf{F_i(s)}$. The design of this feedforward filter is based on a zero-error condition, where the error is defined as in Eq. 2, see also Figure 4.

$$e_i(t) = x_{i-1}(t) - H_i x_i(t), \quad \text{for } i > 1$$
$$= \frac{1 - H_i G_i F_i D_i s^2}{1 + H_i G_i K_i} x_{i-1}(t)$$

Eq. 2

So in order to make the error $e_i(t)$ equal to zero and accounting for the fact that the communication delay cannot be compensated by a causal feedforward filter, yields the optimal design for the feedforward controller filter, see Eq. 3.

$$F_i = \frac{1}{H_i G_i s^2} = \frac{m_i(\tau_i s + 1)}{1 + h_{d,i} s}$$

Eq. 3

Finally, the wireless communication includes delay, i.e., $D_i(s) = e^{-\theta_i s}$, for i > 1. This delay is represented by a constant time delay θ_i, yielding $L(\ddot{x}_{i-1}(t - \theta_i)) = D_i(s) s^2 X_{i-1}(s)$, where $L(\ddot{x}_{i-1}(t - \theta_i))$ represents the Laplace transform of $\ddot{x}_{i-1}(t - \theta_i)$.

Other details of the controller algorithms can be found in [NaVu09].

2.4 Conclusion

In this section, the theoretical information about the ACC and CACC controller is illustrated including the basic control theory, and the important parameters to be investigated in our experiments. Moreover, the definition of string stability is provided and the specific control structure of the CACC controller which indicates the structure of building the CACC controller model stated in Section 3.2.1 is provided. Most of the information presented in this section is based on [NaVu09].

3 Simulation Environment and Models

This section describes the simulation environment and simulation models used in this assignment in order to study the impact of an ACC and of a CACC on the string stability performance.

The used simulation environments and simulation models include: (1) the vehicle behaviour, including the ACC and CACC models, which is implemented using the Simulink environment; (2) the mobility behaviour of vehicles, which is modelled using SUMO traffic environment; (3) the communication networking behaviour that is modelled using the MIXIM / OMNET ++ simulation environment.

3.1 Simulation Environments

This section describes the three simulation environments, the Simulink environment, the SUMO traffic simulation environment and the MIXIM / OMNET++ simulation environment.

3.1.1 Simulink Model Environment

As stated in section 2, in this assignment the vehicles' behaviour is modelled using control theory, with a vehicle being the system, desired distance (decided by time headway) as the reference value, and the velocity of the vehicle as system output. Moreover, the control variable is the engine's throttle position which influences engine torque output.

Since the ACC and CACC controllers we plan to investigate are supplied by TNO, [TNO-safety], in the context of Connect&Drive [C&D] project, in the form of Matlab Simulink model. This Simulink model is the starting point of implementing the ACC and CACC controllers required in this assignment. Below, the Simulink simulation environment is introduced.

Matlab [Matlab] is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation. Using the Matlab product, people can solve technical computing problems faster than with traditional programming languages, such as C, C++, and Fortran.

Simulink is an environment for multidomain simulation and Model-Based Design for dynamic and embedded systems founded on Matlab. It provides an interactive graphical environment and a customizable set of block libraries that let one design, simulate, implement, and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing [Matlab_Simulink].

The Real-Time workshop supplies an interface for the Simulink model to couple with other models. It automatically generates and executes stand-alone C/C++ code for developing and testing algorithms originally implemented in Simulink and Embedded MATLAB code. The resulting code can be used for many real-time and non-real-time applications, including simulation acceleration, rapid prototyping, and hardware-in-the-loop testing. People can tune and monitor the generated code using Simulink blocks and built-in analysis capabilities, or run and interact with the code outside the MATLAB and Simulink environment [Matlab_rtw].

Therefore, in our experiment, we are able to convert the Simulink model composed by Matlab source code to C++ code so that this model can be used by simulators that are using C++ libraries, such as SUMO, see [SUMO].

3.1.2 Traffic Simulator—SUMO

3.1.2.1 Mobility Models

Vehicular mobility models are usually classified as either microscopic or macroscopic. When focusing on a macroscopic point of view, then motion constraints such as roads, streets, crossroads, and traffic lights are considered. Furthermore, in this case also the generation of vehicular traffic such as traffic density, traffic flows, and initial vehicle distributions are defined.

The microscopic approach, instead, focuses on the movement of each individual vehicle and on the vehicle behaviour with respect to others [HaFi07]. In Figure 5, the vehicular mobility models are in advance classified from left to right: macroscopic, microscopic, and sub-microscopic (within the circle: mesoscopic).

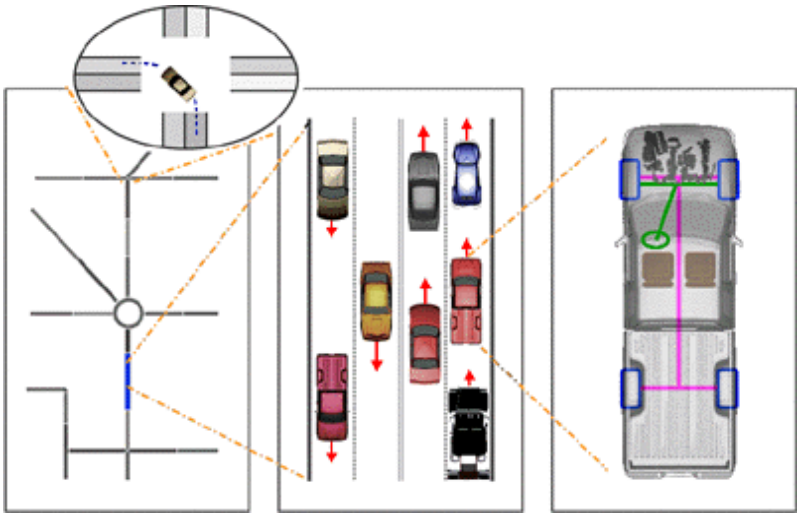


Figure 5: Mobility Models—Macroscopic, Microscopic, Sub-Microscopic from left to right (within the circle: mesoscopic,. copied from [SUMO]

Also according to [HaFi07], several candidates are considered to simulate the VANET related issues but they have clear deficiency: MOVE [KaMo07] could not provide an interaction between the network simulator and mobility model. The method of FDK [FDK] has the limitation that CORSIM [CORSIM] as a traffic simulator has complex calibration and large number of configuration parameters. AutoMesh [VuOg07] is unable to reproduce the non-uniform distribution of positions and speed usually experienced in urban area.
The Simulation of Urban Mobility (SUMO) [SUMO] is an open source, highly portable road traffic simulation package designed to handle large road networks. It is widely used in research community. The decision of using SUMO in combination with MIXIM / OMNET++ is the fact that this combination is often used in the research community and because it has been used within other research activities accomplished in the UT/DACS [UT/DACS] group.

The development of modern vehicular mobility models may be classified in four different classes, see [HaFi07]:

- synthetic Models wrapping all models based on mathematical models;
- traffic Simulators-based Models, where the vehicular mobility models are extracted from a detailed traffic simulator;
- survey-based Models extracting mobility patterns from surveys;
- trace-based Models, which generate mobility patterns from real mobility traces.

Furthermore, according to [HaFi07], synthetic models may be separated in five classes:

- stochastic models wrapping all models containing purely random motions;
- traffic stream models looking at vehicular mobility as hydrodynamic phenomenon;
- Car Following Models, where the behaviour of each driver is modelled according to vehicles ahead;
- Queue Models which models roads as FIFO queues and cars as clients;
- behavioural Models where each movement is determined by behavioural rules imposed by social influences for instance.

The car microscopic movement model in SUMO is a car following model. In SUMO, several car following models have already be implemented can be seen from Table 1 copied from [SUMO]. The SUMO environment is a discrete time simulator, meaning that the location of vehicles moving on a specified road network is calculated, using among others the mobility model, periodically on predefined discrete times. The calculation period can be configured, but a typical value is 10ms.

Table 1: SUMO-implemented car-following models

Element	Short Name	Description
carFollowing-Krauss	SUMOKrauß	The Krauß-model with some modifications which is the default model used in SUMO
carFollowing-KraussOrig1	SKOrig	The original Krauß-model
carFollowing-PWagner2009	PW2009	A model by Peter Wagner, using Todosiev's action points
carFollowing-BKerner	Kerner	A model by Boris Kerner **Caution:** currently under work
carFollowing-IDM	IDM	The Intelligent Driver Model by Martin Treiber **Caution:** Problems with lane changing occur

3.1.2.2 History of SUMO

The development of SUMO started in the year 2000 by the German Aerospace Center, in order to support the traffic research community with a tool into which own algorithms can be implemented and evaluated without the need to regard all the artefacts needed to obtain a complete traffic simulation. Such artefacts are related to the implementation and/or setting up methods for dealing with road networks, demand, and traffic controls [SUMO]. By supplying such an open source microscopic road traffic simulation tool, the German Aerospace Center wanted to (1) make the implemented algorithms more comparable, as a common architecture and model base is used, and (2) gain additional help from other contributors.

SUMO allows high-performance simulations of huge networks with roads consisting of single and multiple lanes, as well as of intra-junction traffic on these roads, either using simple right-of-way rules or traffic lights. Vehicle types are freely configurable with each vehicle following statically assigned routes, dynamically generated routes, or driving according to a configured timetable [SoYa08].

Since 2002, one popular use of SUMO is the evaluation of vehicle-to-vehicle and vehicle-to-infrastructure communication. Two major third-party projects should be mentioned in this context, the first, TraCI, [SUMO], is an extension of SUMO by the possibility to communicate with external applications, done at the University of Lübeck by Axel Wegener. The second project that should be mentioned in this context is "TraNS" [TraNS]. TraNS is a direct coupling between SUMO and the network simulator ns2 [NS2], which uses TraCI for communication and that was set up by Michal Piorkowski and Maxim Raya at the EPFL Lausanne.

Different than “TraNS”, in this assignment, we replace the ns2 by OMNeT++ and use TraCI for the support of

the bidirectional coupling and communication with SUMO.

3.1.2.3 Simulation Processes

For setting up a simulation for SUMO, first, the road network on which the vehicle traffic is moving on, is needed. This can be either done by a) generating an abstract road network using NETGEN, b) setting up an own description in XML and importing it using the NETCONVERT tool, or by c) importing an existing road network using also the NETCONVERT tool. Second, each vehicle should know its route, which is a list of edges that have to be passed and can be known. This can be accomplished either by: a) describing explicit routes on the road network, b) using predefined routes and activating only a percentage of them only, c) generating random routes, d) importing OD-matrices, or e) importing existing routes. Then, if needed, a) compute the dynamic user assignment, b) calibrate the simulation using given measures. The final step is to perform the simulation.

NETGEN allows building abstract networks. Three types of networks can be built, which are: grid-networks, spider-networks and random-networks. One always has to supply the name of the network to generate and the type of network you want to create. However, by using the NETCONVERT tool, one can build a road network of any topology freely.

NETCONVERT imports digital road networks generated by other sources and at the same time can generate road networks that can be used by other SUMO tools. It assumes at least one parameter - the combination of the name of the file type to import as parameter name and the name of the file to import as parameter value. Of course, a user can specify the output file name and type. In our experiments, we did use the method of setting up an own description in XML and importing it using the NETCONVERT tool to generate a road network. This road network is generated in the form of a grid, where the most-left- and most-bottom node (vehicle) in the grid is identified by the coordinate (0,0). For more details, see Appendix C.

3.1.3 Network Simulator—MiXiM/OMNeT++

Network simulation is commonly used to model computer network configurations long before they are deployed in the real world. In this assignment, the CACC controller needs to receive information that is being disseminated using a VANET. The operation of the VANET is modelled using a network simulator. Network simulators are able to evaluate the performance of network protocols and of the communicated traffic, under dynamic changes of e.g., the traffic conditions, the communication channel conditions.

In most cases, network protocols are analyzed using discrete event simulations. A large number of simulation frameworks are available in this area. Examples of such frameworks are open source tools such as the network simulator ns-2 [NS2], [BrEs00], OMNeT++ [Omnetpp], J-SIM [SoHo06], and JiST /SWANS [BaHa04] and commercial tools like OPNET [OPNET]. The reason of selecting OMNeT++ in this assignment is due to the fact that it is often used in research community and due to the fact that is also being used within the DACS group on some research activities.

NS2 and OMNeT++ are considered as candidates in our assignment to couple with SUMO, but as already mentioned the combination of OMNET++ and SUMO is selected to be used in this assignment. It is important to emphasize that SUMO is a discrete time simulator, while OMNET++ is a discrete event simulator. During the integration process of these two types of simulators, this fact is can be considered as an important challenge that needs to be solved.

3.1.3.1 OMNET++

OMNeT++ (Objective Modular Network Tested in C++), see [Omnetpp], is an extensible and modular component-based C++ simulation library and framework that is running on different operating systems such as

Linux, Mac OS X, other Unix-like systems and Windows. Primarily, OMNET++ is developed for building network simulators. The simulator can be used for traffic modelling of telecommunication networks, protocol modelling, queuing networks modelling, multiprocessors and other distributed hardware systems modelling, hardware architectures validating, evaluating performance aspects of complex software systems and modelling any other systems where the discrete event approaches are suitable.

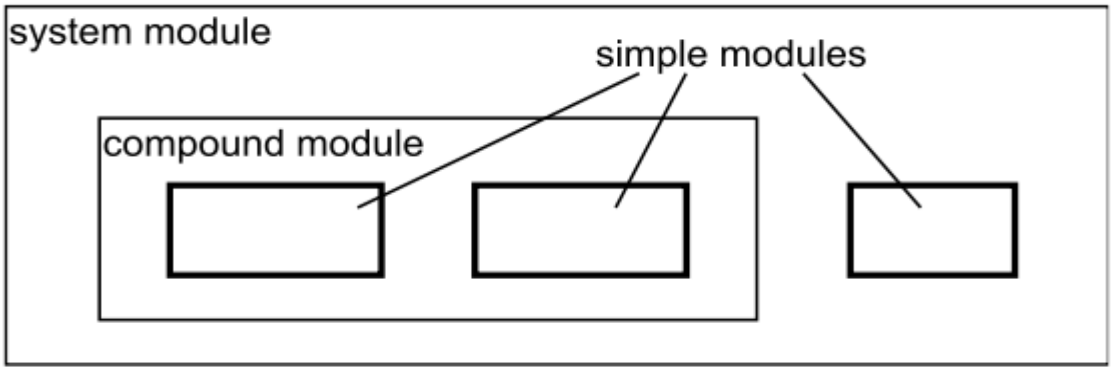


Figure 6: Simple modules, compound module and system module, copied from [Omnetpp_manual]

OMNeT++ provides a component architecture for models. These components programmed in C++ are nested hierarchically and simpler components can assemble to compound components and models using a high-level language—NED (Network Description), see Figure 6. NED lets the user declare simple modules, and connect and assemble them into compound modules. The user can label some compound modules as networks. These compound models are self-contained simulation models. Communication channels can be defined as another component type, whose instances can also be used in compound modules. The NED language has several features which let it scale well. Therefore, it can be used to model large communication topologies [Omnetpp_manual]. These features are:

- **Hierarchical:** The traditional way to deal with complexity is by introducing hierarchies. Any module which would be too complex as a single entity can be broken down into smaller modules, and used as a compound module.
- **Component-Based:** Simple modules and compound modules are inherently reusable, which not only reduces code copying, but more importantly, allows component libraries (like MiXiM) to be reused.
- **Interfaces:** Module and channel interfaces can be used as a placeholder where normally a module or channel type would be used, and the concrete module or channel type is determined at network setup time by a parameter.
- **Inheritance:** Modules and channels can be subclassed.
- **Packages:** The NED language features a Java-like package structure, to reduce the risk of name clashes between different models.
- **Inner types:** Channel types and module types used locally by a compound module can be defined within the compound module, in order to reduce namespace pollution.
- **Metadata annotations:** It is possible to annotate module or channel types, parameters, gates and submodules by adding properties.

Reusability of models makes building certain models flexible. Also, the depth of module nesting is not limited, which allows the user to reflect the logical structure of the actual system in the model structure. Modules communicate with message passing. Messages can contain arbitrarily complex data structures. Modules can send messages either directly to their destination or along a predefined path, through gates and connections.

Modules can have parameters which are used for three main purposes: to customize module behaviour; to create

flexible model topologies (where parameters can specify the number of modules, connection structure etc); and for module communication, as shared variables.

Modules at the lowest level of the module hierarchy are to be provided by the user, and they contain the algorithms in the model. During simulation execution, simple modules appear to run in parallel, since they are implemented as co-routines (sometimes termed lightweight processes). To write simple modules, the user does not need to learn a new programming language, but he/she is assumed to have some knowledge of C++ programming.

Therefore, an OMNeT++ model is combined by simple modules by using the NED language while the simple modules themselves are programmed in C++. The simulation system provides two components: simulation kernel containing the code that manages the simulation and the simulation class library; user interfaces. Graphical, animating user interfaces are highly useful for demonstration, while command-line user interfaces are best for batch execution.

Thus, the way of how OMNeT++ is used is as follows. First, the NED files are compiled into C++ source code, using the NEDC compiler which is part of OMNeT++. Then all C++ sources are compiled and linked with the simulation kernel and a user interface to form a simulation executable.

3.1.3.2 MiXiM

MiXiM (a **MiX**ed s**iM**ulator) is an OMNeT++ modelling framework created for mobile and fixed wireless networks, such as wireless sensor networks, body area networks, ad-hoc networks, vehicular networks, etc. [MiXiM]. MiXiM provides detailed models and protocols, as well as a supporting infrastructure. These can be divided into five groups [KöSw08]:

- **Environment models:** in a simulation, only relevant parts of the real world should be reflected, such as obstacles that hinder wireless communication.
- **Connectivity and mobility:** when nodes move, their influence on other nodes in the network varies. The simulator has to track these changes and provide an adequate graphical representation.
- **Reception and collision:** For wireless simulations, movements of objects and nodes have an influence on the reception of a message. The reception handling is responsible for modelling how a transmitted signal changes on its way to the receivers, taking transmissions of other senders into account.
- **Experiment support:** the experimentation support is necessary to help the researcher to compare the results with an ideal state, help him to find a suitable template for his implementation and support different evaluation methods.
- **Protocol library:** last but not least, a rich protocol library enables researchers to compare their ideas with already implemented ones.

The base framework of MiXiM provides the general functionality needed for almost any wireless modelling. And since every module in OMNeT++ can be replaced, we can easily implement another module using different protocol.

3.2 Integrated Simulation Model

Our Integrated simulation model is constituted by the Simulink-model-based controller, SUMO based models, and MiXiM/OMNeT++ based models. In this part, we'll first give an introduction to these models and then describe how they are combined into the whole integrated simulation model.

3.2.1 Used Simulink Model

The original Simulink model, developed and provided by TNO, simulates a complete system which comprises a platoon of ten vehicles including one leading vehicle and nine following vehicles equipped with both ACC and CACC controllers. Besides the control system, see Figure 7 and Figure 8, each following vehicle has a Wi-Fi (i.e., IEEE 802.11p) [IEEE80211p] interface, ideal radar, an HMI block, a module named "G_a" mimicking the response of the output of the control system and sensors which actually store parameters. "G_a" is part of the module "Vehicle" which also calculates the velocity and position with the generated acceleration.

A vehicle, see Figure 7 and Figure 8, would read data from Wi-Fi (antenna 1), Radar, Sensor, HMI blocks at the beginning of a simulation timestep. These parameters are coupled to the Controller to calculate a reference acceleration "a_ref". During this process, these parameters are also transmitted by Wi-Fi (antenna 2) so that information of this host vehicle can be received by following vehicles. With "a_ref" coupled to the Vehicle block, the acceleration "a", speed (velocity) "v" and position "s" of the vehicle required for the next timestep are calculated and transmited out by antenna . This is done in order to fake the reflection of a radar signal so that its following vehicle can calculate the relative speed and relative distance to this host vehicle. These calculated acceleration, speed and position then are coupled to the Radar and Sensor blocks to be read in the next simulation timestep. Furthermore, inside the Radar block, the preceding vehicle's speed and position can be gotten from antenna 4 so that this host vehicle can calculate its relative speed and position to its preceding vehicle.

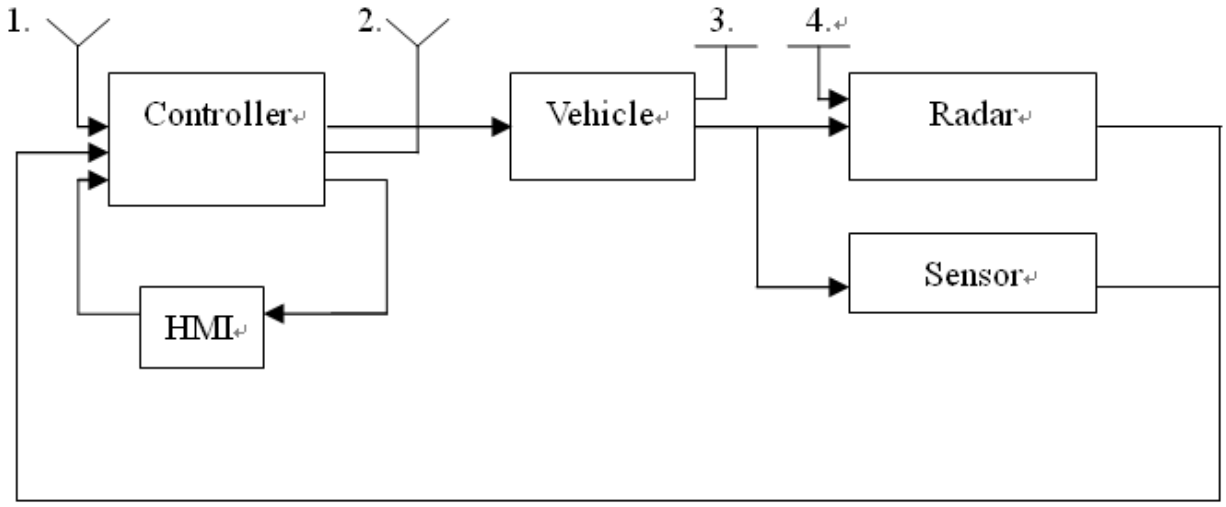


Figure 7: Vehicle's Control System

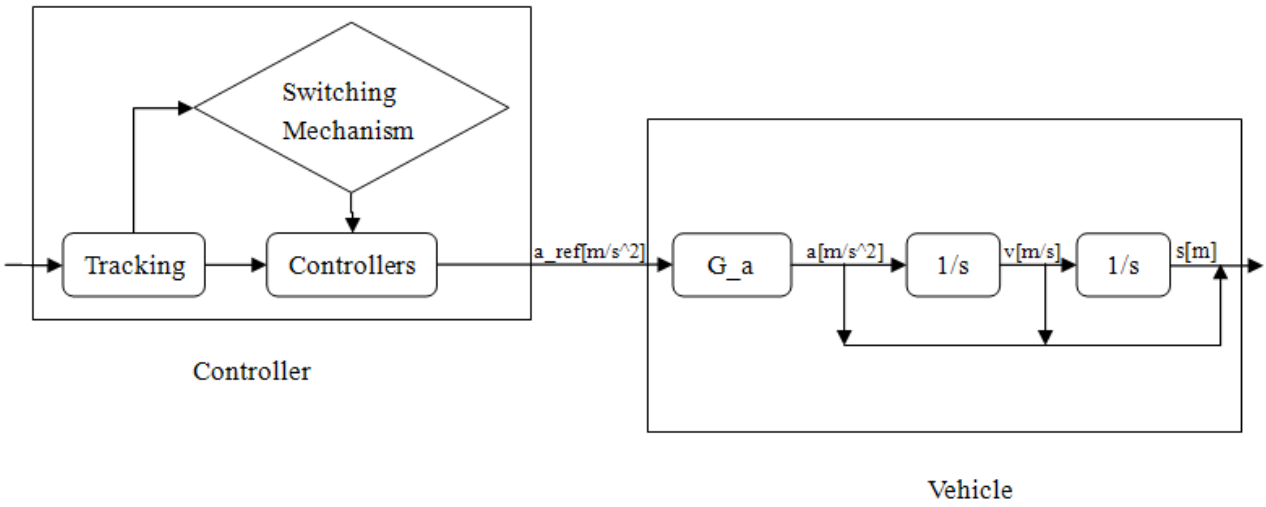


Figure 8*: "Controller" and "Vehicle"blocks*

In Figure 7, Antenna 1 and 2 are the input and output interface of Wi-Fi, while antenna 3 couples the host vehicle's speed and position to the following vehicle to mimic the function of a radar reflection. Antenna 4 is the input of preceding vehicle's speed and position.

Figure 8, shows the modules inside the "Controller" and "Vehicle" blocks. Inside the Controller blocks, the inputs parameters are coupled to several tracking modules: platoon control, host track and target track. The platoon control module selects the to be used time headway and cruise speed values from those specified by an user and from those transmitted by the RSU. The tracking module is consisted of two components: (1) host tracking and (2) target tracking. In the model used in this assignment, the host tracking does not implement any function. The target tracking can be implemented in three modes: direct measurement, single target tracking and multiple tracking. The first one just uses the input of the Controller block as input, while the last two would use Kalman filters to first estimate the received target data for different cases: (1) Wi-Fi data only, (2) Radar data only and (3) full data set. Which mode to be used is specified by the user. Based on the available data and the type of the controller suggested by the user, a controller Switching Mechanism, see Figure 8, actually decides which type of controller will be used. If the controller Switching mechanism is able to select all controller types, CACC, ACC and CC, then the mechanism prefers to first select CACC. If this is not possible, then it selects ACC and if this is not possible then the Switching mechanism selects CC. If the user suggests a specific type of controller and the user requirements can be fulfilled, then the Switching mechanism selects the suggested controller. The selected controller uses inputs to calculate a reference acceleration "a_ref", see Figure 8, which will be coupled to the Vehicle block. First, this reference acceleration "a_ref", will be used as the input of an LTI system named "G_a" which mimics the response of a vehicle and the output value is bounded by the vehicle's maximal acceleration and minimal deceleration. In this process, a reference acceleration is used for the calculation of the acceleration "a". Using two integration blocks, this acceleration "a" can be used to calculate the speed "v" and position "s" by using Eq. 4 and Eq.5, respectively. These calculated values are coupled to the Radar and Sensor modules. Details of this model can be found in Appendix B.

$$velocity=velocity\_lasttime+a*timestep \quad \text{Eq. 4}$$

$$position=position\_lasttime+velocity*timestep \quad \text{Eq. 5}$$

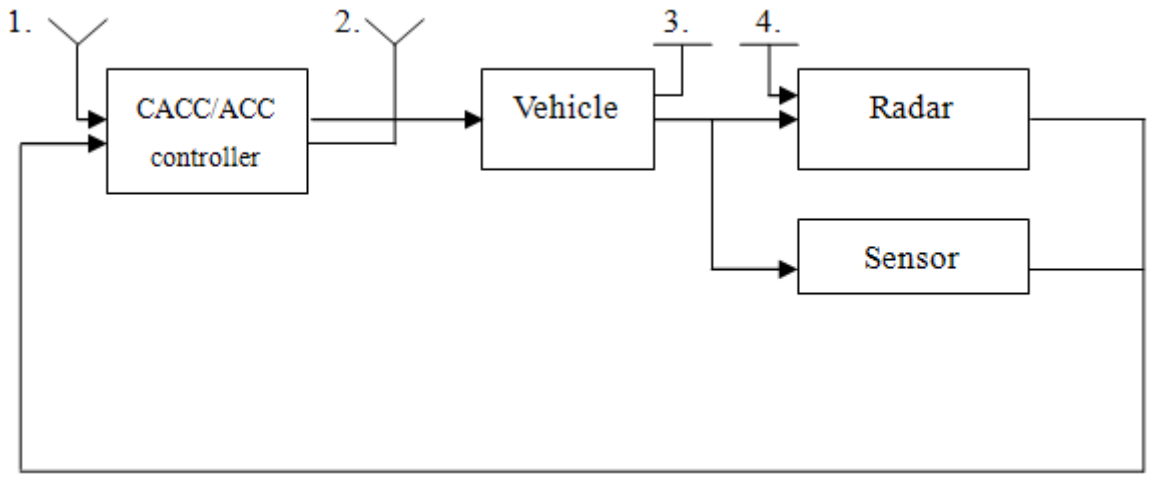


Figure 9: modified vehicle control system

The Simulink model that is used in this assignment and applied in the performed experiments is shown in Figure 9, which is different than the original model developed by TNO and shown in Figure 7. The parameters from Wi-Fi (IEEE 802.11p), radar, etc., located inside the controller system are directly used instead of using Kalman filters. The reason of not using the Kalman filters in this integrated model is related to fact that the Simulink model of the Kalman filters could not be successfully converted into C++ code using the Real-Time Workshop tool of Matlab. Note that the CACC/ACC controller module is used instead of the Controller module shown in Figure 8. The main difference between these controller modules is that the CACC/ACC controller module shown in Figure 9 is not using the Switching mechanism shown in Figure 8, One of the reasons of not using this mechanism is the fact the Simulink model of this Switching mechanism could not successfully be converted into C++ code using the Real-Time Workshop tool of Matlab. Another reason is that we would like to measure the performance of a pure CACC controller and compare it with the performance of a pure ACC controller.

The radar block computes the distance from a vehicle to its preceding vehicle, see Eq. 6, and the relative velocity, see Eq. 7. .

$$\text{Relative velocity} = \text{precedingVehicle\_velocity} - \text{hostVehicle\_velocity} \quad \text{Eq. 6}$$

$$\text{Distance} = \text{precedingVehicle\_position} - \text{hostVehicle\_position} - \text{Vehicle\_length} \quad \text{Eq. 7}$$

The inputs of the CACC controller comprise the host vehicle's acceleration, time headway, cruise speed, relative position and speed to preceding vehicle, and preceding vehicle's acceleration. For the ACC controller the same inputs are used, excluding the preceding vehicle's acceleration. The output of the controller is the reference acceleration "a_f".

Actually, not all of the functions shown in Figure 9 are directly realized by converting the modules into C++ code. This is only realized for the CACC, ACC and "G_a" Simulink modules, where C++ shared libraries are generated for each of these modules. All the other modules shown in Figure 9 are implemented in SUMO using C++ code. Moreover, the function of sensor is implemented by directly reading parameters stored in SUMO. The detailed description of the modified Simulink model is given in Appendix B.

3.2.2 SUMO Model

The SUMO model used in this assignment is quite simple. The road network is just a long straight single-lane road. This is because at this moment the CACC and ACC controllers operate on a single lane, and there's no lane changing mechanism implemented in the model.

Though there are several ways of building a road network, for such a simple scenario, setting up an own road network description in XML and importing it using NETCONVERT can be considered to be an easy task. In order to create a road, which contains sections (i.e., edges) and nodes (i.e. start/end points), three types of XML files are created:

- Node based XML: this XML file describes the coordinates of the start/end points of a road section. This XML file should contain at least three nodes.
- Edge based XML: this file describes the edges (i.e., of each road section), that has to be used between two nodes included in the node based XML file.
- Link type XML: this file describes the properties of the edges included in the edge based XML file. When the link type XML file is not used, then SUMO considers that the used edges are single lanes.

In this assignment we use a single lane road network, that comprises two road sections (i.e., two edges) between three nodes. Therefore, in this assignment we use a node based XML file that contains three nodes. Furthermore, an edge based XML file is created that contains two edges that are interconnecting the three nodes specified in the node based XML file. So by converting the two XML files using the NETCONVERT tool, we constructed a one-single lane road with two edges and three nodes. In this assignment it is considered that 10 vehicles see Appendix B, will be located on one of the generated edges, i.e., first edge. Therefore, the length of the first edge is set to 5000 m. The second edge is not relevant for our experiments and therefore, the length of the second edge is set to 1m.

Vehicles can be distributed on a predefined road network using a fourth type XML file, denoted as route XML file. This route XML file contains the vehicle parameters and the description of the routes that each vehicle can take on the road network. The vehicle parameters can be the car length, the route ID, vehicle ID, maximum speed, and maximum acceleration. In Figure 10 a part of the generated road network is shown, where a platoon of 10 vehicles is located on the first edge. We mark each vehicle with an ID, where the leading vehicle's ID is "veh0", that of the first following vehicle is "veh1" and the last vehicle's ID is "veh9". In the original model, see Appendix B, the car length is 4.46 meters, the desired distance between neighbouring vehicles when standstill is 7.7m, and the time headway 0.7s, see Section 2.3. Moreover, in the original model, see Appendix B, the leading vehicle has an initial velocity of 20m/s and the following vehicles' an initial velocity of 19m/s. Therefore, by using Eq. 1 from Section 2.3: the desired distance between neighbouring vehicles when moving equals: 7.7m+20m/s*0.7s=21.7 m. When the car length is taken into consideration, the desired distance between neighbouring vehicles when moving is equal to: 21.7m+4.46m=26.16m. Therefore, the initial distance used in the route XML file is set equal to the distance when the platoon become stable (i.e., 26.16m), which ensures that the time before the platoon gets stable will not be large.

In SUMO, a following vehicle uses the car-following model to track preceding vehicle. In SUMO there are already several car-following models implemented, see Section 3.1.2. Note that in all experiments performed in this assignment are using the "carFollowing-IDM" model, see [SUMO].

What we want to do is to change the way of calculating vehicle's speed and position resulted from the existing SUMO car-following models. We can specify any car-following model in the route XML file, because it will not change the way of calculating speed and position.

We let the leading vehicle move on a straight single-lane road from left to right. When the platoon gets stable (the speed and relative position of vehicles do not change any more), we would specify the leading vehicle's behaviour including acceleration and deceleration to observe the behaviours' of following vehicles. In the

following experiments accomplished in this assignment, the downstream direction means the direction from the last vehicle to the leading vehicle (from left to right in the model), while the upstream direction means the direction from the leading vehicle to the last vehicle (from right to the left in the model).

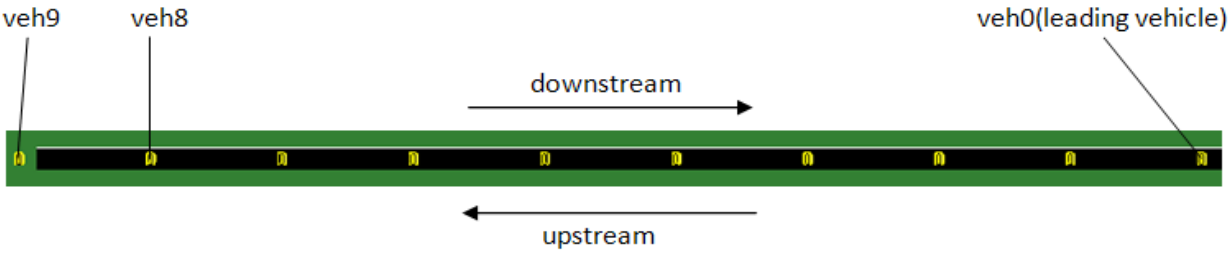


Figure 10: SUMO road network

3.2.3 MiXiM / OMNET++ Model

The used MiXiM/OMNET++ model in this assignment has been developed within the UT/DACS [UT/DACS] group. This model is implementing a cooperative awareness mechanism using beaconing, see [EeKa10]. The beaconing procedure is using a Simple Timer, see EeKa10],which means that a node transmits a beacon when a timer τ expires. Afterwards, this timer is reset. By tuning the value of τ , the beacon sending rate/frequency can be varied.

In this assignment it is considered that each beacon packet needs to carry for each vehicle, the acceleration and vehicle ID. This is because only preceding vehicle's acceleration is necessary to be sent by communication means. After each vehicle receives one beacon, it will decide whether this beacon has been sent by its preceding vehicle, which has a certain know vehicle ID. As has been already mentioned, 10 vehicles were used, meaning that the used vehicle IDs vary from 0 to 9 through the platoon. Therefore, if the receiving beacon's vehicle ID is larger than the receiving vehicle's ID by 1, it is considered that this beacon is sent by the preceding vehicle and is fed to the controller. Otherwise, this beacon will not be used and will be dropped.

The MAC and Physical layers used in the MiXiM/OMNET model are based on the IEEE 802.11p technology, see [IEEE802.11p-2010]. IEEE 802.11p is an approved amendment to the IEEE 802.11 standard to add wireless access in vehicular environments. It is important to note that currently no IEEE802.11p model is included in the MiXIM version 1.2 environment, see [MiXiM]. The model used in this assignment was realized by modifying the currently available IEEE 802.11 MiXiM example, i.e., Mac80211, such that it could operate as an 802.11p model. In particular, just the "Host.ned" module is used, which describe the individual vehicle's communication architecture. The modified MAC layer module and Physical layer source code, plus the higher layer mechanisms' source code are developed in activities accomplished outside the context of this assignment.

In addition to that, in order to take advantage of a model that successfully integrated the SUMO and MiXiM environments, a modified version of MiXiM environment is used, see [MiXiM_sommer]. This modified MiXiM environments is created by Christoph Sommer. In particular, in this assignment the "Highway.ned" module provided in the example "traci_lauchd", see [MiXiM_sommer] , is used.

Below, a short introduction is given of the parameter values used in the IEEE 802.11p model. The carrier frequency of this model is set to 5.87 GHz which is in the frequency allocated by European Commission in August, 2008 for or priority road safety applications and inter-vehicle, infrastructure communications. In this model, the header length in each layer is different from the "Mac80211" example used in [MiXiM]. In particular, the header length associated with the in Physical layer is set to 0bit, the header length associated with the MAC

layer is set to 160bit, the header length associated with the Network layer is set to 3200bit and the header length associated with the Application layer is set to 512bit.

The model transmits the beacons using the IEEE 802.11p broadcast channel. Moreover (1) the MAC queue length is set to 14 frames, and the MAC bit rate is set to 3 Mbps, (2) the RtsCtsThreshold is set to1400, (3) the beacon length is 3200bit with a duration of 0.001175, (4) the burst size is set equal to 3. For other parameters, please refer to the used source code, which can be accessed using the guidelines given in Appendix C.

3.2.4 Complete Simulation Model

The MiXiM/OMNET++ model is used to simulate the wireless communication medium between vehicles. In Figure 11, the communication medium is represented by the Wi-Fi module, which is among others used to disseminate the dynamic traffic parameters of vehicles, such as speed, location, acceleration. The controller used in each vehicle uses these parameters and influence the movement of the cars. Figure 12 gives an abstract view of the integrated/complete simulation model used in this assignment, where (1) the MiXiM model simulates the operation of the wireless communication medium, (2) SUMO simulates the mobility behaviour of a "vehicle", whose traffic mobility-related parameters were supplied by its controller provided by the (3) Simulink Model, which simulates a series of functions including the ACC and CACC controller functions. Note that the Simulink model is integrated into the SUMO simulation environment.

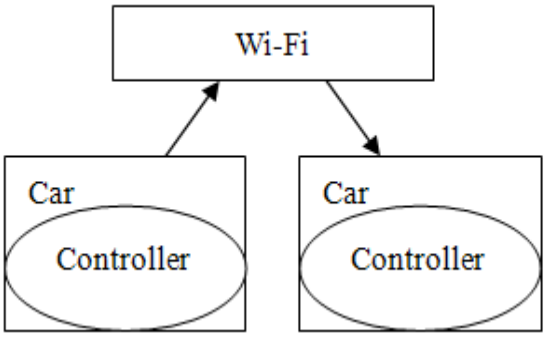


Figure 11: experiment structure in reality

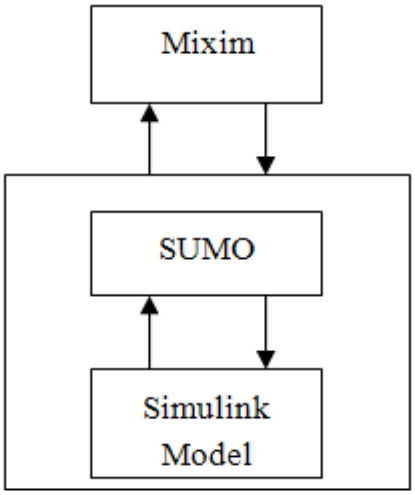


Figure 12: experiment structure in simulation

3.2.4.1 Coupling between SUMO and the Simulink Model

The coupling between the Simulink and SUMO models is used to simulate the mobility behavior of vehicles. Since SUMO is C++ based, it is required to convert the Simulink model into C++ code or available C++ libraries. This conversion is realized by using the Real-Time Workshop tool in Simulink, where the Simulink model is converted into a C++ shared library. After this, the SUMO source code associated with the car following model on how speed and position is modified. The detailed description of this method and the associated Source code can be found (or found via) in Appendix C.

It is important to mention that SUMO is a discrete time simulator; MiXIM is an event based simulator. This means that SUMO will calculate the location of all vehicles during periodic discrete times. We denote these discrete times as timesteps. The duration of each time step used in this assignment is set to 10ms. The converted Simulink model (the C++ libraries) is using parameters received from MiXiM and parameters provided by SUMO, e.g., position, velocity, and acceleration during one timestep. Moreover, the converted Simulink model is generating the host vehicle's position and velocity for the coming timestep. These calculated parameters are used by the SUMO model to move the vehicle on a road network. It is important to note that the information provided by MiXIM is only used when the CACC model is used. In the situation that ACC is used, and then only the parameters provided by SUMO are used by the converted Simulink model for the calculation of the velocity and position for the coming timestep.

In particular, for the situation that the CACC model is used, at the beginning of each simulation step, the SUMO model associated with one vehicle gets the preceding vehicle's acceleration from MiXiM. The SUMO model has already stored for each vehicle, the speed, position, and acceleration generated during the previous timestep. The time headway and desired cruise speed is preconfigured in the SUMO source code.

The preceding vehicle's information can also be fetched directly in SUMO, without using the information communicated by MiXIM. The relative velocity and distance are calculated using Eq. 6 and Eq. 7, respectively. Moreover, the acceleration can be directly fed from the SUMO model, instead of retrieving it via the MiXiM model, see Section 4.2. When pure ACC is used then all controller inputs are retrieved from the SUMO model. All these parameters are passed to the ACC/CACC controller as inputs so that updated speed and position of a vehicle is generated and provided to SUMO, which updates the position of each vehicle.
These operations are repeated during each timestep.

3.2.4.2 Bidirectional Coupling between OMNeT++/MiXiM and SUMO

In this assignment the method described in [SoYa08], [SoGe11] is used for bidirectional coupling between OMNeT++/MiXiM (similar to OMNeT++/INET in [SoYa08], [SoGe11]) and SUMO. As already mentioned OMNeT++ is an event-based simulator, being able to handle mobility by scheduling node movements at regular intervals. This suits the approach followed by SUMO, which is a discrete time simulator.

Figure 13 shows the control modules and used state machines that are integrated with OMNeT++ and SUMO, see [SoGe11]. Using these state machines it can be seen that any commands arriving in-between timesteps can be buffered to guarantee synchronous execution at defined intervals, see Figure 14. In particular, OMNeT++ would then send at each timestep all buffered commands to SUMO. At the same time trigger the corresponding timestep of the road traffic simulation. Subsequently, when the road traffic simulation timestep is completed, SUMO sends a series of commands and the position of all the instantiated vehicles back to OMNET++ module. By receiving this information, the OMNET++ module can react to this mobility trace by changing the status of the vehicles involved in the trace. This could mean that new vehicles can be introduced, vehicles that reached their destinations can be removed and vehicles can be moved according to their road traffic simulation counterpart. When all the received commands are processed and all the vehicles are moved according to the mobility trace, then OMNET++ advance the simulation until the next scheduled timestep. In this way the vehicles are allowed to react to the altered environmental conditions.

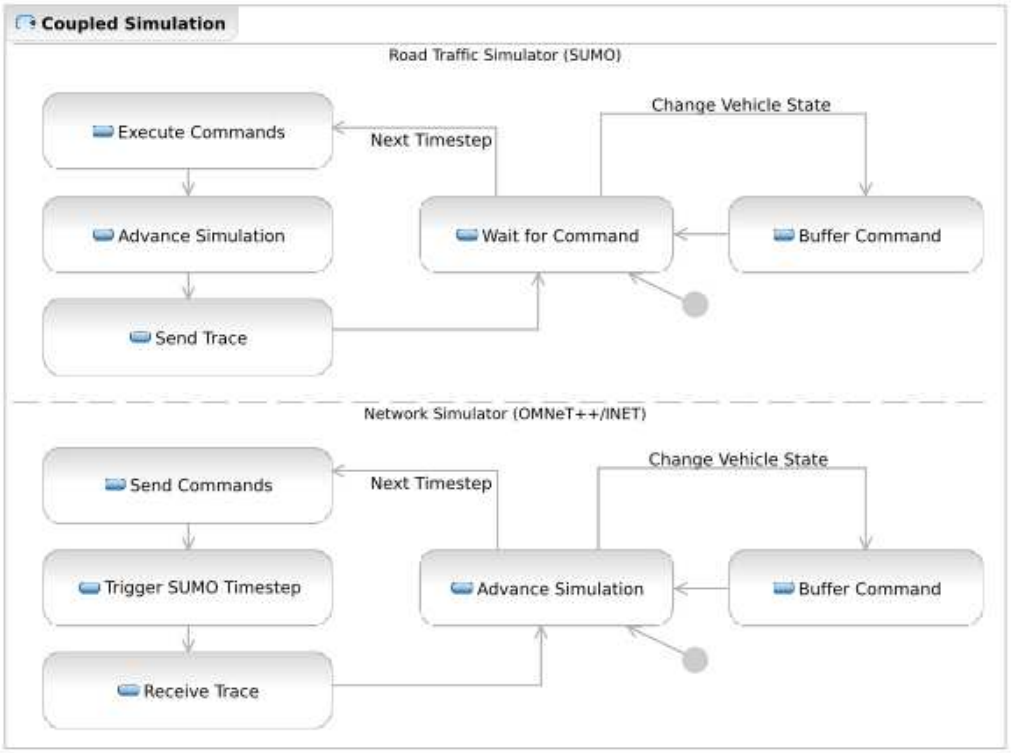


Figure 13: Overview of the coupled simulation framework. State machines of road traffic and network simulator communication modules, copied from [SoGe11]

Figure 14 shows the sequence diagram of messages exchanged between network and road traffic simulator communication modules. By using a simple request/response protocol, the road traffic in SUMO can be influenced by OMNeT++ in different ways. It is important to see that timesteps are generated to advance the simulation in SUMO. The two alternating phases show that the OMNET++ and SUMO modules are bidirectionally coupled to each other. In the first phase, OMNET++ commands are sent to SUMO, while in the second phase the execution of these commands in SUMO is triggered by OMNET ++ and the resulting mobility trace generated by SUMO is sent to OMNET++. In this way, both simulators are tightly coupled. Furthermore, it can be seen that SUMO is only able to perform a simulation step after all events within a time step have been processed in the OMNET++ network simulation. It is important to see that the network simulator advances the road traffic microsimulation only at fixed intervals, meaning that the granularity of these intervals needs to be sufficiently fine-grained to obtain realistic results. This can be realized since the SUMO road traffic microsimulatrion can be processed much faster compared to the simulation of OMNET++ wireless networks.

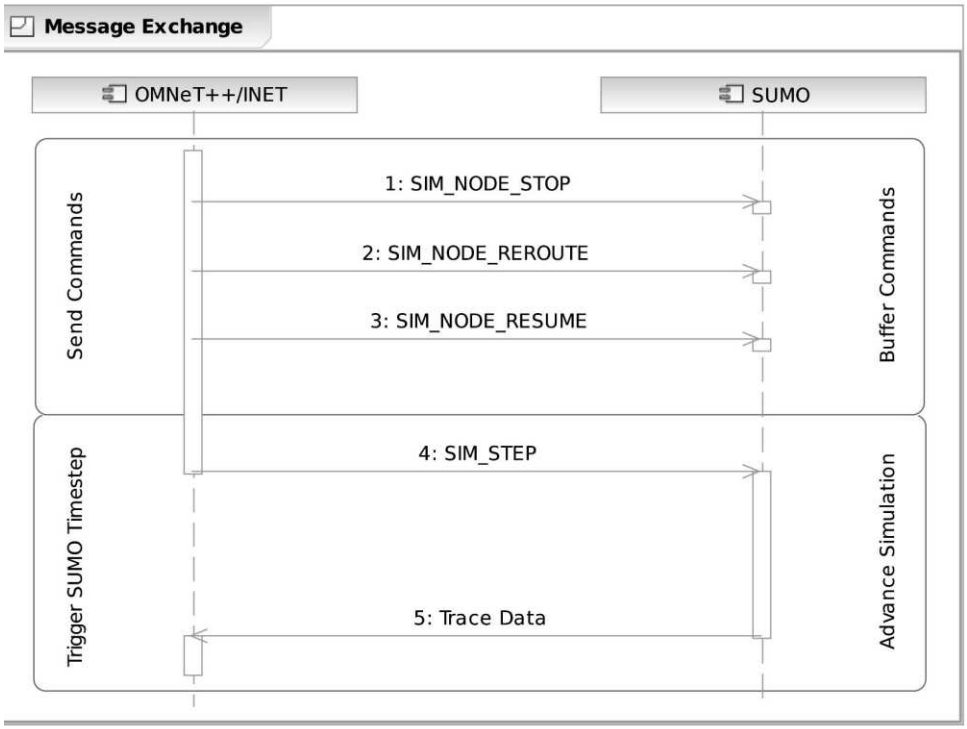


Figure 14: Sequence diagram of messages exchanged between network and road traffic simulator communication modules. Command execution is delayed until the next road traffic simulation timestep is triggered, copied from [SoGe11]

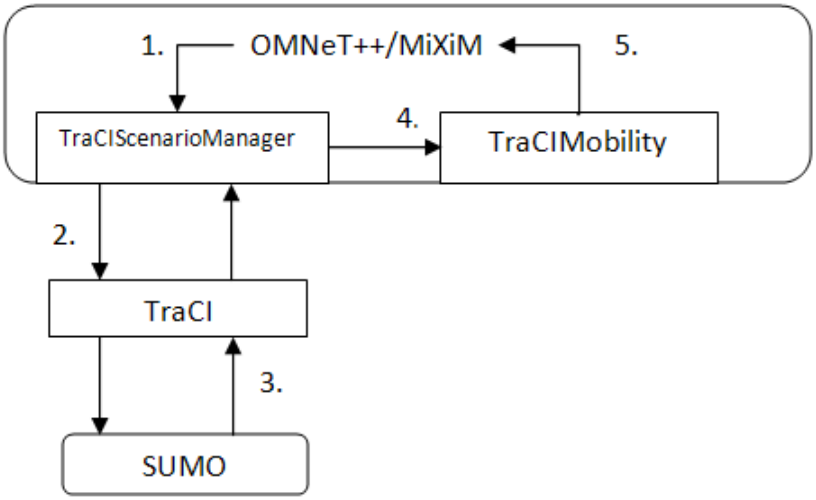


Figure 15: coupling between OMNeT++/MiXiM and SUMO

Figure 15 shows also the coupling between OMNET++/MiXiM and SUMO. The functionality blocks shown in Figure 15 are:

- **TraCI**: is a traffic control interface, which is a TCP based client/server architecture and it provides access to run a road traffic simulation. During simulation runs, both SUMO and OMNeT++ use their communication modules to exchange commands by using TCP connections. As a TraCI client, OMNeT++/MiXiM uses TraCIScenarioManager to communicate with the TraCI server—SUMO.

- **TraCIMobility**: is a OMNeT++/MiXiM function that is able to move corresponding communication nodes. The communication between interacting modules in OMNeT++ is accomplished by exchanging internal messages. The mobility of communication nodes in MiXiM is realized by TraCIMobility function once vehicles in traffic simulation environment update their position and speed.

- **TraCIScenarioManager:** TraCIScenarioManager connects OMNeT++ to a TraCI server (SUMO) running road traffic simulations. It sets up and controls the simulation experiments, moving nodes with the help of a TraCIMobility module. It makes the initialization of the stages in the simulation as well as controls the connection to the TraCI server (SUMO). The communication between OMNeT++/MiXiM and SUMO is based on exchanging TraCI messages. The TraCIScenarioManager as being the "Manager" can ask SUMO for all the parameters such as vehicle speed, position and to execute all the commands such as creating an object. This can however be accomplished only if these parameters and commands are defined in the "TraCIConstants.h" header file. "TraCIConstants.h" is a header file that defines all parameters allowed to be transmitted between SUMO and OMNET++/MIXiM), e..g., acceleration, position, velocity). Moreover, this header file contains all the allowed program commands that can be executed and can use on these parameters, e.g., "get", "set". The function "queryTraCI" specifies exactly which parameters (and commands) will be exchanged between SUMO and MiXiM. In this assignment the exchanged parameters are acceleration, position, and velocity and time headway. The "queryTraCI" command is send in a message that is buffered within the "TraCIBuffer" module until the beginning of each time step. During the simulation, SUMO would execute as "queryTraCI", executing commands and sending back information through TraCI back to OMNeT++/MiXiM.

At the beginning of each timestep, OMNeT++ would send buffered commands to SUMO by using the TraCIScenarioManager (step 1&2 in Figure 15). SUMO uses this information as described in the previous subsection. Once the received information is used, then SUMO sends a trace to MiXiM (step 3 in Figure 15), which includes the traffic information such as position, speed and acceleration of vehicles. In MiXiM, the communication nodes can also move discretely according to the positions received from the SUMO trace. This movement of the communication nodes is implemented by the MiXiM mobility module—TraCIMobility (step 4 in Figure 15). Then the communication nodes' state of each vehicle is changed followed by the procedure of transmitting a beacon (step 5 in Figure 15). Note that the received information is buffered before the start of next simulation timestep.

3.3 Conclusion

In this section, first the original model environment—Simulink is introduced. Then the traffic simulation environment—SUMO and the network simulation environment—OMNeT++/MiXiM are described. The SUMO and OMNeT++/MiXiM models are used in the experiments described in Section 4.

4 Experiments, Results and Analysis

This section describes the accomplished experiments used to investigate the impact of ACC and CACC on the string stability performance.

4.1 Experiment Setup

The main goal of this assignment is to investigate the impact of an ACC and of a CACC on the string stability performance. In order to achieve this goal a set of experiments are performed that evaluate the ACC and CACC performance, using as performance measure the string stability, see Section 4.3. String stability, see Section 3.2.2, is important and should be guaranteed to stabilize the movement of the platoon. By observing the speed and acceleration of following vehicles it can be investigated whether the disturbance of the leading vehicle is amplified upstream through the platoon.

As described in Section 3.1.1 the ACC and CACC controllers are available in SUMO in the form of shared C++ libraries, which are generated by converting the Simulink models of these controllers into C++ code. Due to the fact that small modifications have been brought to these converted C++ shared libraries, a set of experiments is performed to verify whether the modified and integrated model is satisfactorily equivalent with the original Simulink model provided by TNO. This set of experiments is described in Section 4.2.

The topology that is used in all experiments is shown in Figure 10 and explained in Section 3.2.2. A platoon of ten vehicles is placed in a straight single lane road that has a length of 5000 meters. The parameters used for the ACC and CACC controllers, vehicle IDs, starting vehicle positions and departure speed are the same as the ones described in Section 3.2.2, which are also used by the original Simulink model provided by TNO.

These parameters are specified in the road based network XML file and route XML file, as stated in Section 3.2.2. Similar to the value used by the original Simulink model provided by TNO, the default time headway is specified to be 0.7s and the default desired cruise speed is specified to be 50m/s, see also Appendix B and Appendix C. Furthermore, the upper limit of the vehicle's acceleration is specified to be $2m/s^2$ and the minimal deceleration is specified to be $-9m/s^2$, which are also implemented in source code. These parameters apply to all experiments accomplished in this assignment. Note however, that in Section 4.3, various values for the acceleration and time headway are used.

4.2 Simulink Model Verification and ACC vs. CACC performance when using an ideal communication medium

4.2.1 Experiment Description

4.2.1.1 **Experiment goal, topology, measures and parameters:**

The first goal of this set of experiments is to compare the combined SUMO-Simlink modified model with the original Simulink model provided by TNO and described in Section 3.2.1.

The second goal of this set of experiments is to compare the CACC and ACC performance, under ideal communication medium circumstances. This means that in this set of experiments it is considered that all

information that needs to be received by the ACC and CACC controllers in each vehicle is indeed received with a probability of 100%. In other words, it is considered that the communication medium between vehicles is an ideal communication line, with no delays and no losses.

This set of experiments is achieved by using the same topology, the same values for the used static and variable parameters for both models. In particular, the dynamicity of the leading vehicle is changed and the velocity and acceleration of the following vehicles is observed. If the observed velocity and acceleration for the following vehicles, are the same (under certain bounds) for both models then it can be assumed that these two models are satisfactorily equivalent. It is important to emphasize that the results associated with the original Simulink model provided by TNO are obtained using the Simulink monitoring facilities, while the results associated with the modified model, are obtained using the SUMO monitoring facilities. They are plotted using the GNUplot tool.

In this set of experiments two scenarios have been observed. In the first situation the leading vehicle decelerates, while in the second situation the leading vehicle accelerates. Note that for all experiments the value of the timestep is set to 10 ms.

Decelerating scenario: In the decelerating situation it is assumed that the leading vehicle initially moves with a speed (velocity) of 20m/s. For the original Simulink model it is considered that at time t=500timestep, the leading vehicle starts to decelerate with an acceleration of -9m/s^2, until the leading vehicle reaches the speed (velocity) of 15 m/s. This is acceleration is kept constant for another 54 timesteps. For the last timestep (t=555timestep), before the leading vehicle reaches the speed (velocity) of 15m/s, the acceleration of the leading vehicle is set to -5m/s^2. The acceleration of the leading vehicle for t: t<500timesteps and t > 555timesteps is set to 0. For the modified model it is considered that the leading vehicle starts to decelerate with an acceleration of -9m/s^2 at t=8000 timestep. This acceleration is kept constant for another 54 timesteps until the leading vehicle reaches a speed (velocity) of 15 m/s. This is accomplished in order to give the possibility for the movement of all vehicles in the platoon to become stable and to move with a speed (velocity) of 20m/s. Similarly, to the original model, for the last timestep, before the leading vehicle reaches the speed (velocity) of 15m/s, the acceleration of the leading vehicle is set to -5m/s^2. Note that the acceleration of the leading vehicle for t: 8000timesteps > t > 8055timesteps is set to 0.

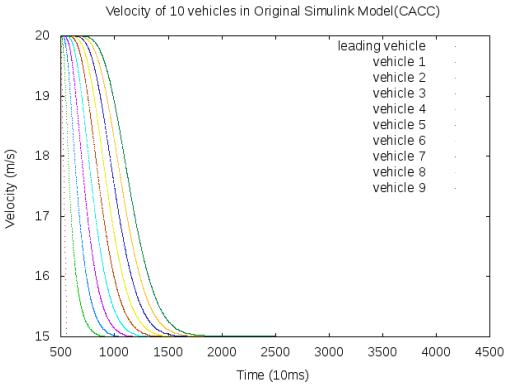
Accelerating scenario: For completeness of the comparison between ACC and CACC, we also accomplished the experiments for the situation that the leading vehicle is accelerating, instead of decelerating. In this scenario the initial speed (velocity) of the leading vehicle is 20m/s that accelerates with an acceleration 2m/s^2 until the speed (velocity) of the leading vehicle reaches the value of 25 m/s. This acceleration is kept constant for another 249timesteps.

For the original model, provided by TNO, at t=500timestep, the leading vehicle starts to accelerate with an acceleration of 2m/s^2 until it reaches the speed (velocity) of 25m/s. Note that the acceleration of the leading vehicle for t: t<500timesteps and t > 749timesteps is set to 0. For the modified model, the leading vehicle starts to accelerate with the same acceleration as in the original model at t= 8000timestep. Similar to the deceleration scenario, this is accomplished in order to give the possibility for the movement of all vehicles in the platoon to become stable and all vehicles move with a speed (velocity) of 20m/s. Note that the acceleration of the leading vehicle for t: 8000timesteps > t > 8249 timesteps is set to 0.

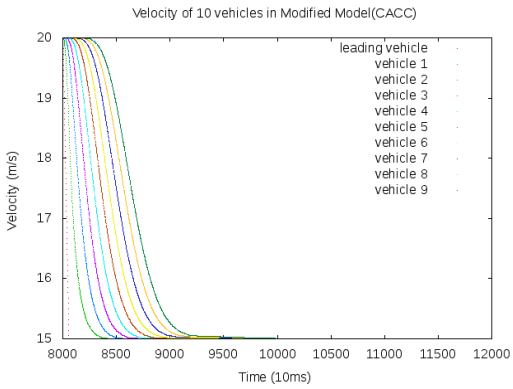
4.2.2 Experiment Result &Analys

4.2.2.1 Decelerating scenario

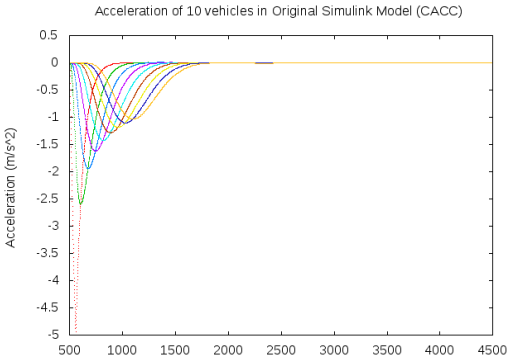
The CACC and ACC results associated with the decelerating scenario can be seen in Figure 16 and Figure 17, respectively**.**



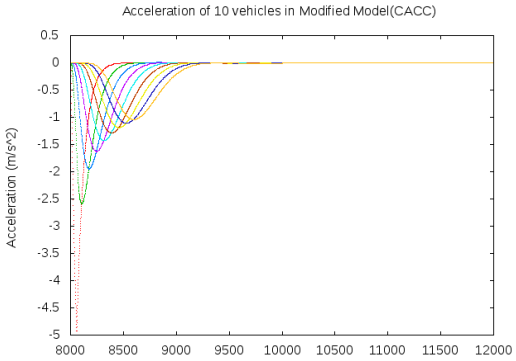
(a) velocity of 10 vehicles in Original Simulink Model (CACC)



(b) velocity of 10 vehicles Modified Model (CACC)

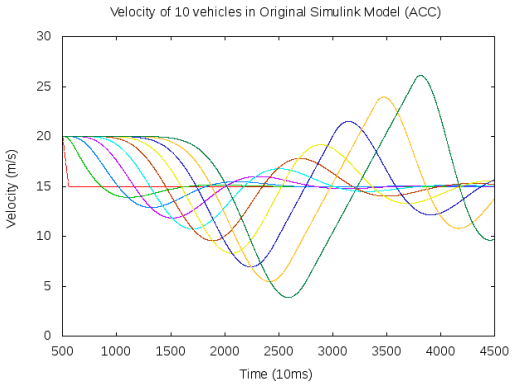


(c) acceleration of 9 following vehicles in Original Simulink Model (CACC)

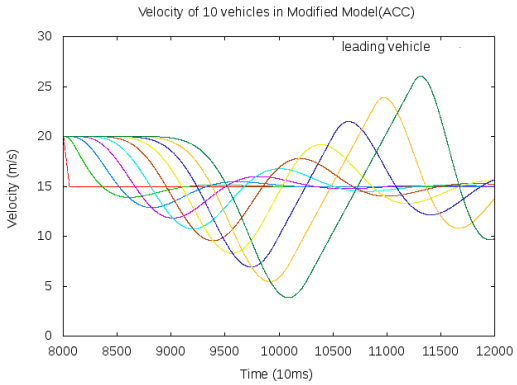


(d) acceleration of 9following vehicles in Modified Model (CACC)

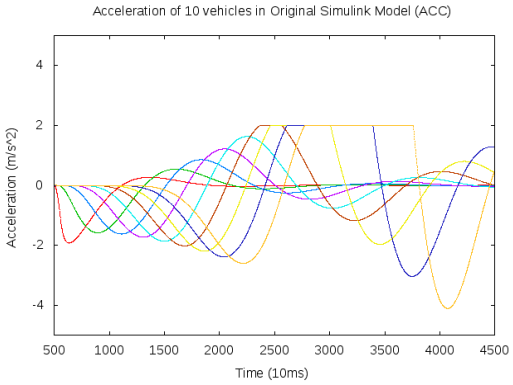
Figure 16: velocity and acceleration of 10 vehicles in Original Simulink Model and modified model for CACC (decelerating scenario)



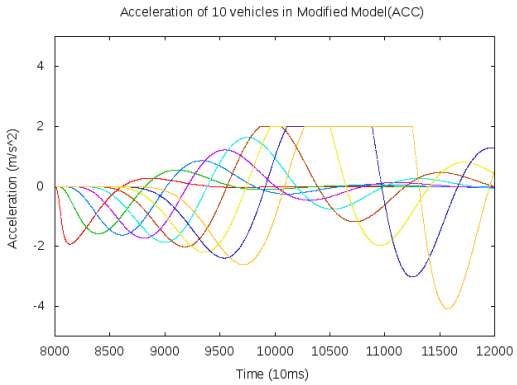
(a) velocity of 10 vehicles in Original Simulink Model (ACC)



(b) velocity of 10 vehicles in Modified Model (ACC)



(c) acceleration of 9 following vehicles in Original Simulink Model (ACC)



(d) acceleration of 9 following vehicles in Modified Model (ACC)

Figure 17: velocity and acceleration of 10 vehicles in Original Simulink Model and modified model for ACC(decelerating scenario)

The result showed in parts (a) and (c) of Figure 16 and Figure 17 are obtained from the experiments performed using the original Simulink model, while the result showed in parts (b) and (d) of Figure 16 and Figure 17 are obtained from the experiments performed using the combined SUMO – Simulink modified model. The parts (a) and (b) of Figure 16 and Figure 17, show the curve velocities of the used vehicles, which are drawn from left to right, staring from the velocity of the leading vehicle towards the last one, i.e., veh9. The parts (c) and (d) of Figure 16 and Figure 17, are representing the acceleration of the 9 vehicles following the leading vehicle. Starting from left to right, the first curve is associated with the first following vehicle, while the last curve is associated with the last following vehicle, i.e., veh9. The vehicle acceleration of the leading vehicle is not shown, but is realized in the way as in Section 4.2.1.1. From Figure 16 and Figure 17, it can be seen that for the string stability from the point of view of velocity and for both CACC and ACC controllers there are no significant differences between the original Simulink model and the modified model.

For CACC, see Figure 16, the maximum difference of velocity between the original Simulink model and the modified model is 0.02m/s and the maximum difference of acceleration between these two models is 0.03m/s^2.
For ACC; see Figure 17, the maximum difference of velocity between the original Simulink model and the modified model is 0.1m/s and the maximum difference of acceleration between these two models is 0.08m/s^2.,

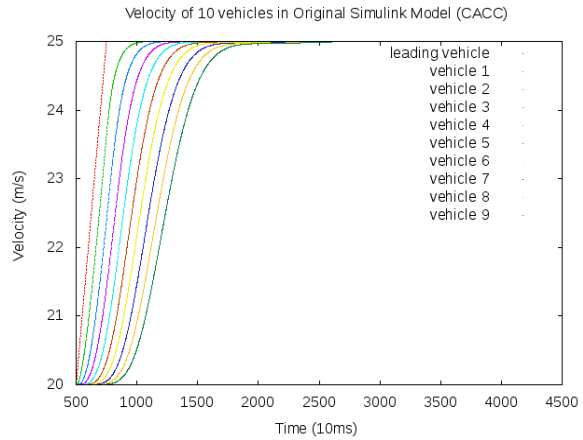
The reasons of observing these (relative small) differences can be the following:

- Kalman filters are not used because of converting problem

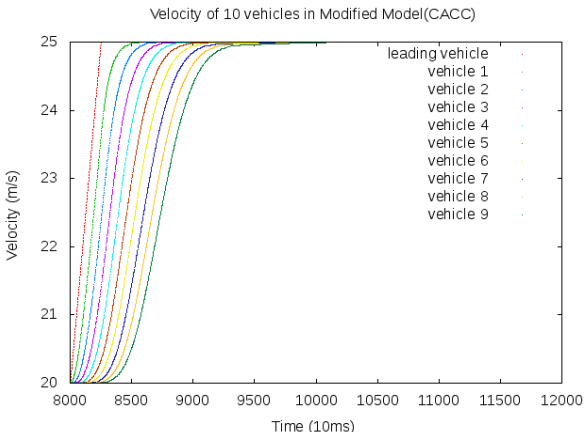
- Complexity of the original model is much larger, which might introduce larger processing delays.

4.2.2.2 Accelerating scenario

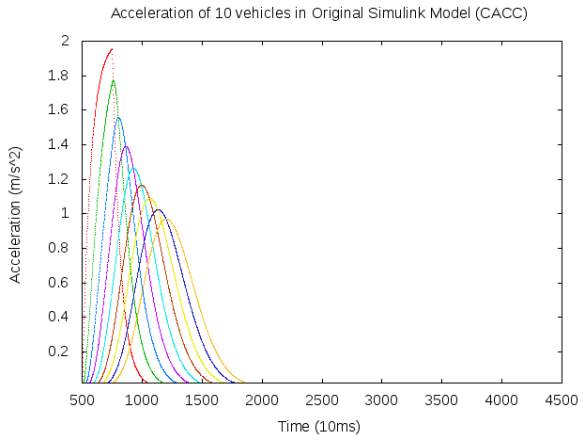
The CACC and ACC results associated with the accelerating scenario can be seen in Figure 18 and Figure 19, respectively.



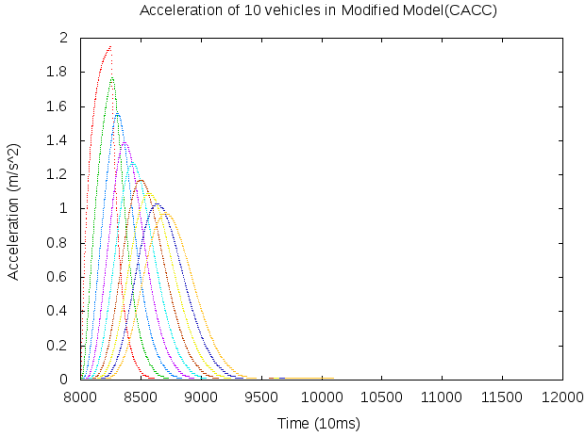
(a) velocity of 10 vehicles in Original Simulink Model (CACC)



(b) velocity of 10 vehicles in Modified Model (CACC)

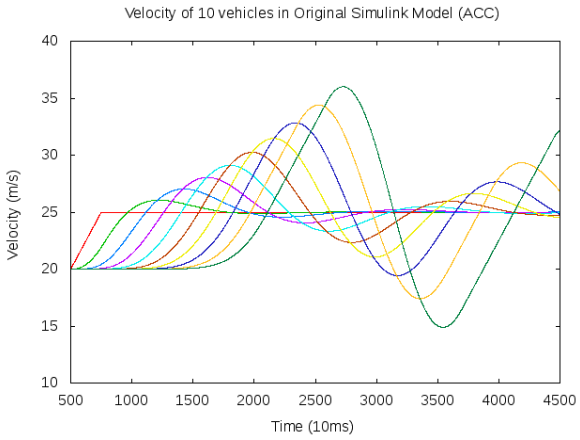


(c) acceleration of 9 following vehicles in Original Simulink Model (CACC)

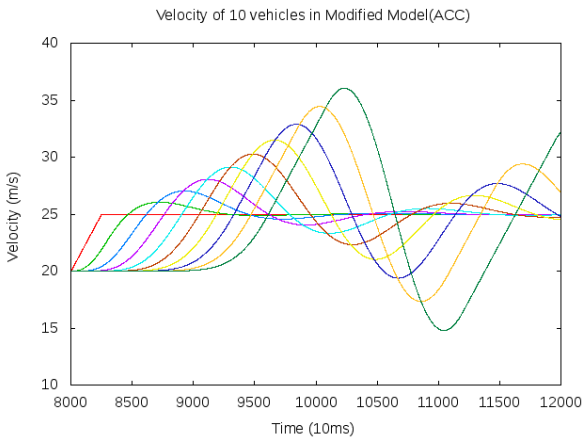


(d) velocity of 10 vehicles in Modified Model (CACC)

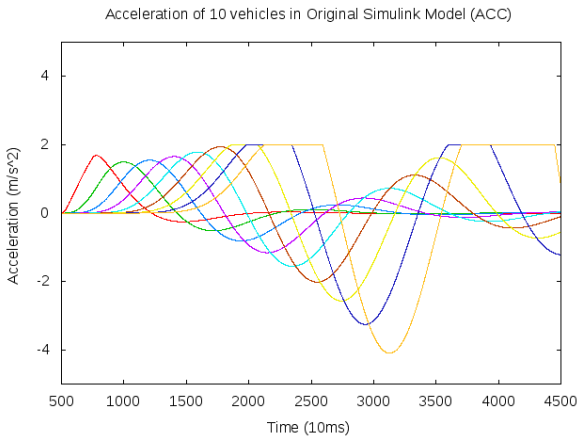
Figure 18: velocity and acceleration of 10 vehicles in Original Simulink Model and modified model for CACC (accelerating scenario)



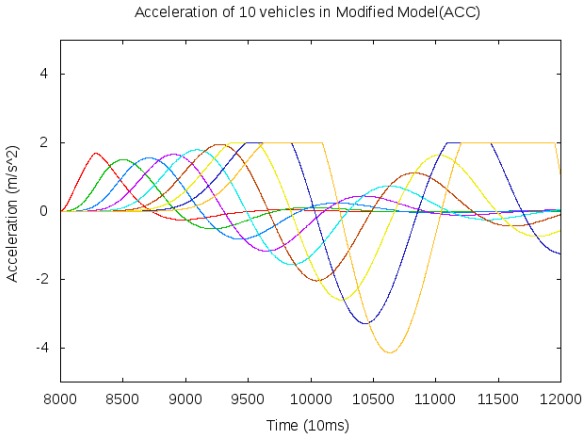
(a) velocity of 10 vehicles in Original Simulink Model (ACC)



(b) velocity of 10 vehicles in Modified Model (ACC)



(c) acceleration of 9 following vehicles in Original Simulink Model (ACC)



(d) acceleration of 9 following vehicles in Modified Model (ACC)

Figure 19: velocity and acceleration of 10 vehicles in Original Simulink Model and modified model for ACC(accelerating scenario)

Parts (a) and (b) of Figure 18 and Figure 19 , show the vehicle velocity. Starting from left to right, the first curve is related to the velocity of the leading vehicle and the last curve represents the velocity of the last following vehicle, i.e., veh9. Parts (c) and (d) of Figure 18 and Figure 19 show the vehicle acceleration. Starting from left to right, the first curve represents the acceleration of the first following vehicle and the last curve represents the acceleration of the last following vehicle, i.e. veh9. The vehicle acceleration of the leading vehicle is not shown, but is realized in the way specified in Section 4.2.1.1.

From Figure 18 and Figure 19, it can be seen that for the string stability from the point of view of velocity and for both CACC and ACC controllers there are no significant differences between the original Simulink model and the modified model.

For CACC, see Figure 18, the maximum difference of velocity between the original Simulink model and the modified model is 0.01m/s and the maximum difference of acceleration between these two models is 0.01m/s^2.

For ACC; see Figure 19, the maximum difference of velocity between the original Simulink model and the modified model is 0.09m/s and the maximum difference of acceleration between these two models is 0.07m/s$^{2.}$, For CACC, in case of deceleration (first scenario), following vehicles decelerate smoothly as the leading vehicle decelerates, but not decelerate more than the leading vehicle and in the case of acceleration, they don't accelerate more the leading vehicle does. From parts (c) and (d) of Figure 18, it can be seen that the maximum absolute value of acceleration of following vehicles are smaller than the leading vehicle and decreases in upstream direction.

Similar, to the decelerating situation, the reasons of observing these (relative small) differences can be the following:

- Kalman filters are not used because of converting problem
- Complexity of the original model is much larger, which might introduce larger processing delays.

In general it can be concluded that:

- For CACC:
    - in case of the deceleration scenario, the following vehicles decelerate smoothly when the leading vehicle decelerates, but do not decelerate more than the leading vehicle does.
    - In case of the acceleration scenario, the following vehicles do not accelerate more than the leading vehicle does. In particular, the maximum values of acceleration of the following vehicles are smaller than the acceleration of the leading vehicle and their values are decreasing in the upstream direction.

- For ACC:

    - for both decelerating and accelerating scenarios, the following vehicles will decelerate or accelerate more than the leading vehicle does, until their velocities become stable (do not fluctuate anymore).
    - In case of the accelerating scenario, the maximum absolute value of acceleration of following vehicles can be larger than the leading vehicle and increases in upstream direction except the first following vehicle. From parts (c) and (d) of Figure 19, it can be seen that the maximum absolute value of acceleration for the last following vehicle is limited by the maximum acceleration of 2m/s^2. Different from the case of CACC where acceleration of following vehicle go back to zero smoothly, here acceleration for the case of ACC would fluctuate around the 0 for a while before finally approaching to zero.

By comparing the obtained results it can be observed that for both accelerating and decelerating scenarios, the disturbance on the velocity and acceleration caused by the leading vehicle is not being amplified through the platoon upstream when the CACC controller is used. This conclusion does not hold for the situation that the ACC controller is applied.

Since the differences between the original Simulink model and the combined SUMO – Simulink modified model are quite small, we can assume that the behaviour of the two models, from the point of view of string stability, are satisfactorily equivalent. Therefore, the following set of experiments will only use the combined SUMO – Simulink modified model.

4.3 Evaluating the impact of wireless communication medium on CACC string stability performance

4.3.1 Experiment Description

4.3.1.1 Experiment goal, topology, measures and parameters

The main goal of this set of experiments is to observe and analyse the CACC string stability performance, assuming that the wireless communication medium is a realistic IEEE 802.11p wireless medium. In particular, the performance of the CACC controller is observed, when the beacon sending rate, the packet loss probability of beacons and the time headway between vehicles are varied.
In this set of experiments the same topology and parameters are used as the ones described in Section 4.2.2 for the first set of experiments. The main differences are related to the use of the OMNET++/MiXiM simulation model, described in Section 3.2.3, as the wireless communication medium that interconnects the 10 vehicles. Moreover, only the CACC controller is used, that is incorporated in the combined SUMO -Simlink modified model. Note that the new combined SUMO-Simulink –OMNET++/MiXiM model is denoted in this section (and the following sections as "modified model".
In this set of experiments the preceding vehicle's velocity and acceleration is received by a host vehicle via the OMNET++/MiXiM model as described in Section 3.2, instead of receiving it directly via SUMO. So, it is not guaranteed that a host vehicle receives the velocity and acceleration of the preceding vehicle at each SUMO simulation timestep.

The different packet loss ratios are realized in the following way. A module used to compute the packet loss ratio is used and located at the point where the received information by a vehicle needs to be propagated to the CACC controller. This module is used to drop the received beacons by using a loss probability with uniform distribution. The module generates a random value between 0 and 1 with uniform probability every time a beacon received. If a packet loss ratio of a% is needed, then this module compares the random generated value with the preconfigured packet loss ratio a%. If the random value is larger than a%, then the received beacon is kept and propagated towards the CACC controller. If this generated random value is equal or smaller than a% then the beacon is dropped.

The different beacon sending rates (R) are realized by using different beacon sending intervals (T), where R=1/T, For example, in order to compute a beacon sending rate for a vehicle equal to 10Hz, the vehicle should send a beacon every 100ms (i.e., 10 SUMO simulation timesteps) in the MiXiM model.

The different used time headways are: 2s, 1.5s, 1s, 0.9s, 0.8s, 0.7s, 0.6s, and 0.5s.

In order to provide statistically accurate results we calculate the 90% confidence intervals, see e.g., [Jain91], of the obtained experimental results. Every experiment was run ten times (using different random seeds for each run). The 90% confidence intervals of the obtained results are discussed in Appendix A of this report.
In these experiments we are not observing the string stability for all 9 following vehicles, but we are only observing the string stability for the last following vehicle (veh9). The last following vehicle is chosen for this purpose, due to the fact that when the platoon is not string stable then the disturbance on a leading vehicle would be amplified through the platoon in the upstream direction, and the last following vehicle would experience the most significant disturbance effect.

Similar to Section 4.2.2, two types of scenarios are used, (1) the decelerating scenario, where the acceleration of the leading vehicle decelerates from a velocity of 20m/s to a velocity of 15 m/s, and the (2) accelerating scenario, where the leading vehicle is accelerating from 20m/s to a velocity of 25m/s.

Furthermore, for each type of scenario (i.e., decelerating or accelerating) we performed two sets of experiments.

In the first set of experiments the string stability, i.e., velocity and acceleration, is observed when the time headway is set constant to 0.7 s and varied the packet loss ratio and the beacon sending rate. The value of 0.7 s is chosen, since this value has been specified in the original Simulink parameters file specified by TNO. The chosen values of packet loss ratio are 10%, 20%, 30%, 40%, 50% and that of beacon sending rates are 25Hz, 20Hz, 15Hz, 10Hz, 5Hz.

In the second set of experiments the string stability, i.e., velocity and acceleration, is observed when the packet loss rate and beacon sending rate are set constant and the time headway is varied, from 0.5s to 2s. In particular, the beacon sending rate is set to be 15Hz and the packet loss ratio is set to be 20%, since this combination provides a good reference to observe the influence of time headway on string stability..

4.3.2 Experiment Result &Analysis

This section describes the experiment results and their analysis that are accomplished in order to observe the impact of a wireless communication medium on the CACC string stability performance.
Two scenarios are used, i.e., (decelerating and accelerating), and for each of these scenarios two sets of experiments are performed. In the first set of experiments the packet loss ratio and the beacon sending rate are varied, while the time headway is kept constant. In the second set of experiments the time headway is varied, while the packet loss ration and beacon sending rate are kept constant.

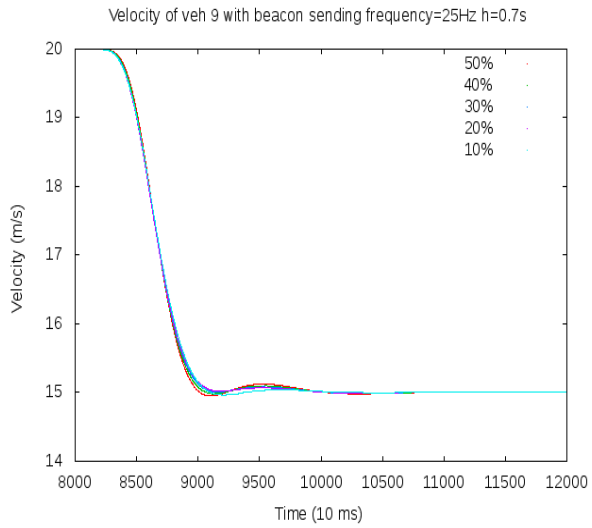
4.3.2.1 Decelerating scenario

This section describes the two sets of experiments that have been accomplished in the context of the decelerating scenario.

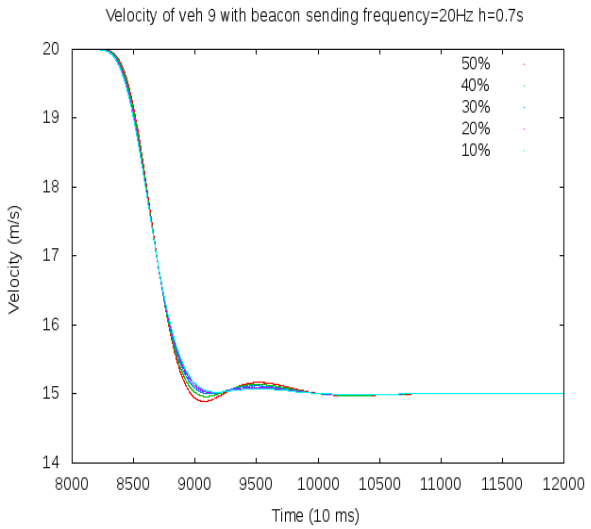
4.3.2.1.1 Varying packet loss ratio and beacon sending rate

In this set if experiments the time headway is set to be 0.7s and the packet loss ratio and beacon sending rates are varied In this part, with a constant time headway of 0.7s, The chosen values of packet loss ratio are 10%, 20%, 30%, 40%, 50% and that of beacon sending rate are 25Hz, 20Hz, 15Hz, 10Hz, 5Hz.

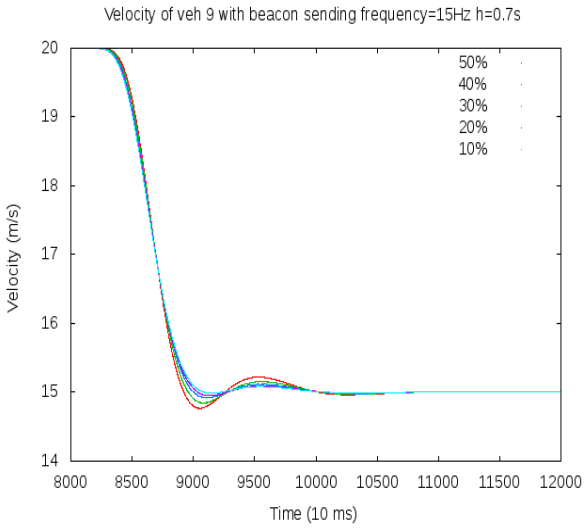
In particular, Figure 20 and Figure 21 show the curves associated with velocity and the acceleration, respectively, of the last following vehicle, i.e., veh9. Starting from left to right, the first curve is associated with the packet loss (PL) ratio of 10%, while the last curve is associated with the packet loss (PL) ratio of 50%. Note that for the captions used in this section, “h” denotes time headway and PL denotes packet loss ratio.



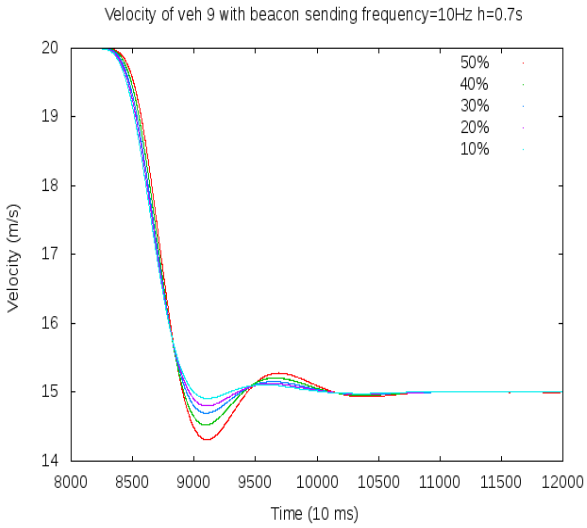
(a) velocity of veh 9 with beacon sending frequency=25Hz, h=0.7s



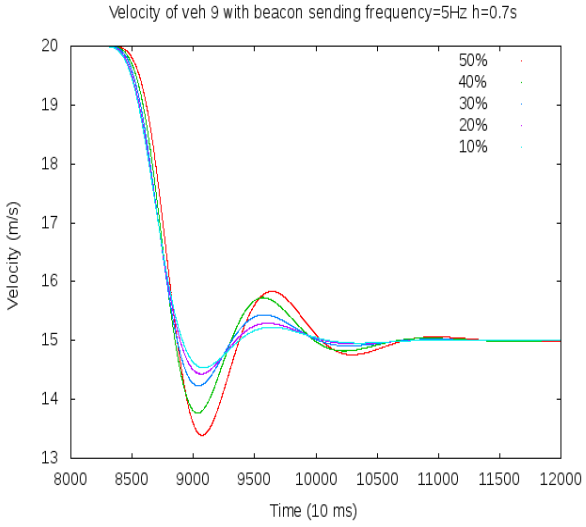
(b) velocity of veh 9 with beacon sending frequency=20Hz, h=0.7s



(c) velocity of veh 9 with beacon sending frequency=15Hz, h=0.7s



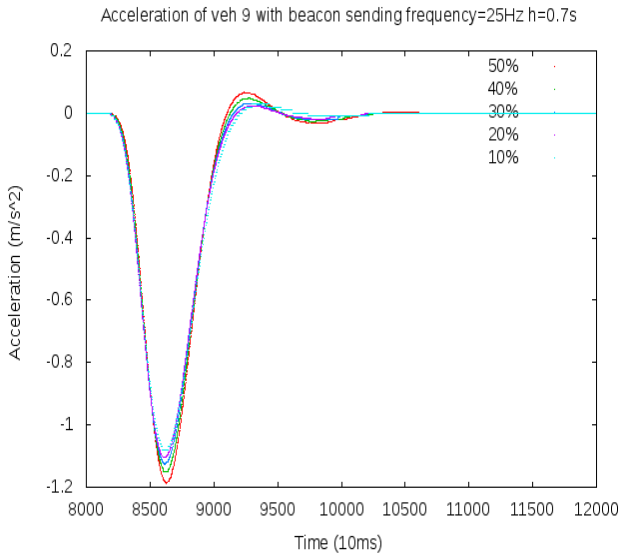
(d) velocity of veh 9 with beacon sending frequency=10Hz, h=0.7s



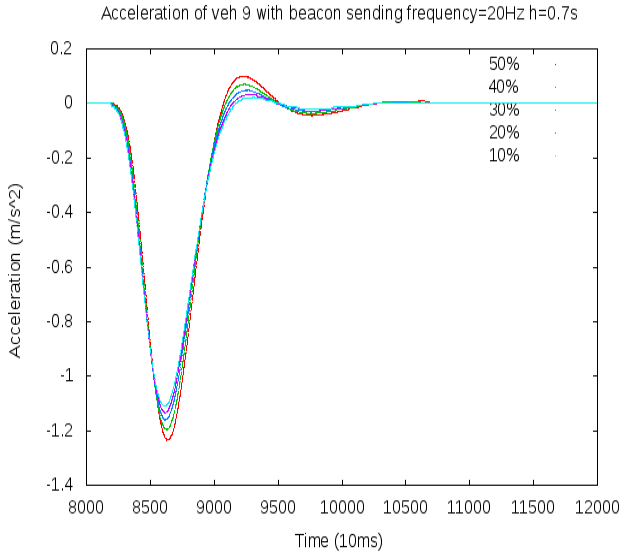
(e) velocity of veh 9 with beacon sending frequency=5Hz, h=0.7s

Figure 20: velocity of veh 9 in modified model for CACC with MiXiM (decelerating scenario)

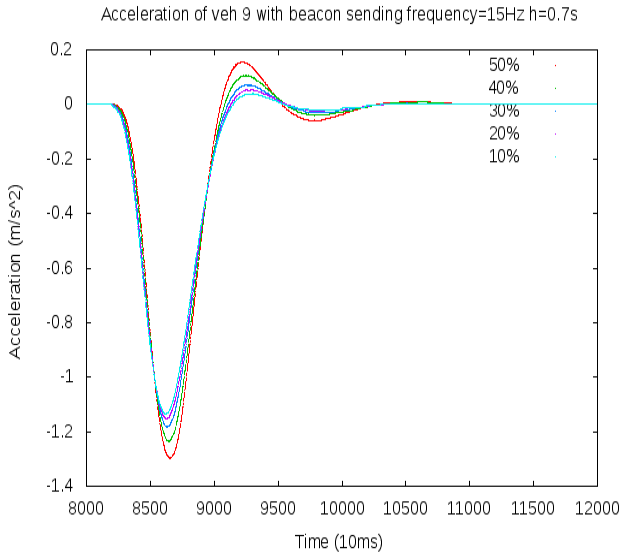
And the results of acceleration can be seen from Figure 21:



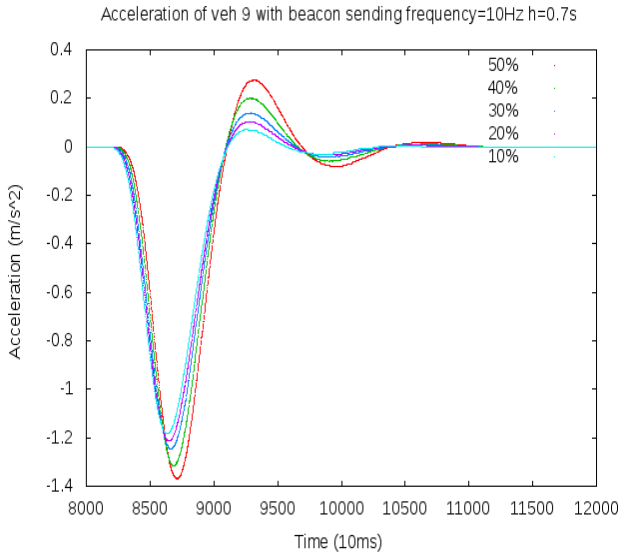
(a) acceleration of veh 9 with beacon sending frequency=25Hz, h=0.7s



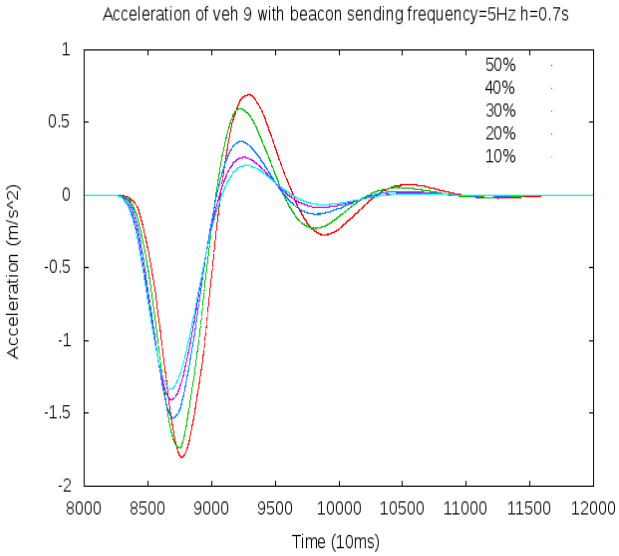
(b) acceleration of veh 9 with beacon sending frequency=20Hz, h=0.7s



(c) acceleration of veh 9 with beacon sending frequency=15Hz, h=0.7s



(d) acceleration of veh 9 with beacon sending frequency=10Hz, h=0.7s



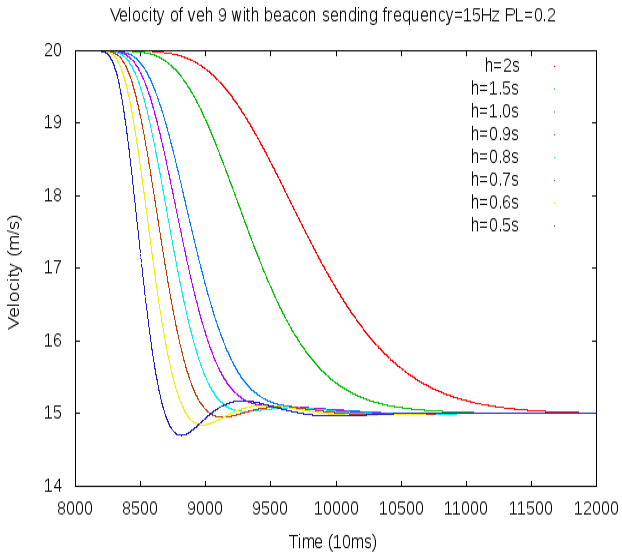
(e) acceleration of veh 9 with beacon sending frequency=5Hz, h=0.7s

Figure 21: acceleration of veh 9 in modified model for CACC with MiXiM (decelerating scenario)

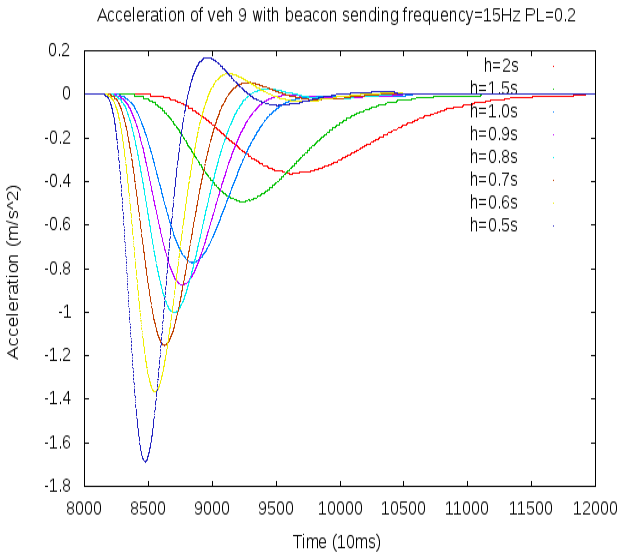
From each part (a) until (e) of Figure 20 it can be seen that for a constant value of beacon sending rate and time headway (0.7s), as the packet loss ratio increases, the velocity fluctuations of veh9 are increasing, which means that the disturbance of the leading vehicle is amplified more through the platoon upstream. Furthermore, for a constant value of packet loss ratio and time headway (0.7s), as the beacon sending rate decreases, the velocity fluctuations of veh9 are increasing. Thus for a given value of time headway, as the packet loss ratio is increased or the beacon sending rate is decreased, the platoon becomes to be, from the point of view of velocity, more string-instable.

From each part (a) until (e) of Figure 21, it can also be seen that for a constant value of beacon sending rate and time headway (0.7s), as the packet loss ratio increases the acceleration fluctuations of veh9 are also increasing. Furthermore, the absolute value of the maximum acceleration and minimum deceleration gets larger as the packet loss ratio increases. Furthermore, for a constant value of packet loss ratio and time headway (0.7s), as the beacon sending rate decreases, the acceleration fluctuations of veh9 are increasing. Moreover, in this case the absolute value of the maximum acceleration and minimum deceleration become to be larger. Thus, also for acceleration, it can be concluded that for a given value of time headway, as the packet loss ratio is increased or the beacon sending rate is decreased, the platoon becomes to be, from the point of view of acceleration, more string-instable.

Therefore, it can be concluded that for a constant time headway value, as packet loss ratio increases and/or beacon sending rate decreases, the platoon becomes to be more string-instable, which means that the disturbances of a leading vehicle are being amplified. The cause of this effect that occurs when the packet loss ratio is increased and/or when the beacon sending rate decreased, can be related to the fact that the CACC controller is not always using up to date acceleration values. Note that during one timestep, when the CACC controller does not receive an acceleration value in time, then it uses an acceleration value that was used by this controller during the previous timestep. The causes of not receiving the acceleration value in time can be due to one or more lost beacons, or due the fact that beacons are not sent (and received) frequently enough. In other words, when the difference between the used acceleration value of the preceding vehicle and up-to-date acceleration value of the preceding vehicle increases, then the platoon becomes to be more string-instable.



(a) velocity of veh 9 with beacon sending frequency=15Hz, PL=0.2



(b) acceleration of veh 9 with beacon sending frequency=15Hz, PL=0.2

Figure 22: velocity and acceleration of veh 9 in modified model for CACC with MiXiM with different time headway (decelerating scenario)

4.3.2.1.2 Varying the time headway

In this set of experiments the packet loss ratio and beacon sending rates are kept constant and the time headway is varied. In particular, the packet loss ratio is set to 20% (PL=0.2), and the beacon sending frequency (rate) is set to 15Hz. The time headway is varied from 0.5s to 2s.

In particular, parts (a) and (b) of Figure 22 show the curves associated with velocity and acceleration, respectively, of the last following vehicle, i.e., veh9. Starting from left to right, the first curve is associated with the time headway of 0.5s, while the last curve is associated with the time headway of 2s.

Note that for the captions used in this section, "h" denotes time headway and PL denotes packet loss ratio.

From Figure 22, it can be seen that when the packet loss ratio and beacon sending rate are kept constant, as the time headway increases the platoon becomes to be more string-stable. The velocity of the last vehicle can decelerate with less fluctuations and its acceleration becomes to be more stable, see also [NaVu09].
Furthermore, with larger time headways, the relative distance between vehicles is larger and when a disturbance occurs on a leading vehicle, the following vehicles do not react as sharp as when small time headways are used. However this will decrease the road throughput and capacity. Therefore, finding the smallest time headway to guarantee string stability and keeping the road capacity high can be considered as an important challenge.

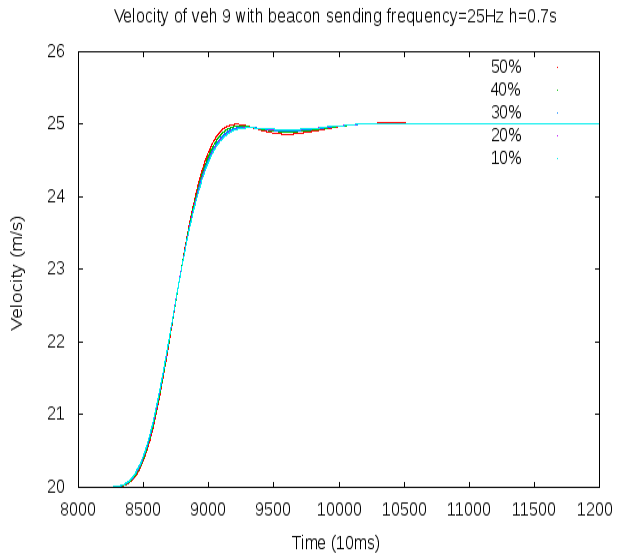
4.3.2.2 Accelerating scenario

This section describes the two sets of experiments that have been accomplished in the context of the accelerating scenario.

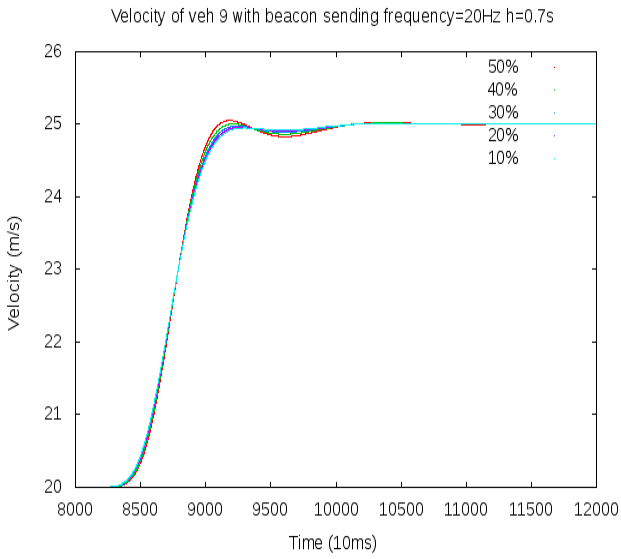
4.3.2.2.1 Varying packet loss ratio and beacon sending rate

The way of how the time headway, packet loss ratio and beacon sending rate are chosen is the same as in the experiments described in Section 4.3.2.1.1.

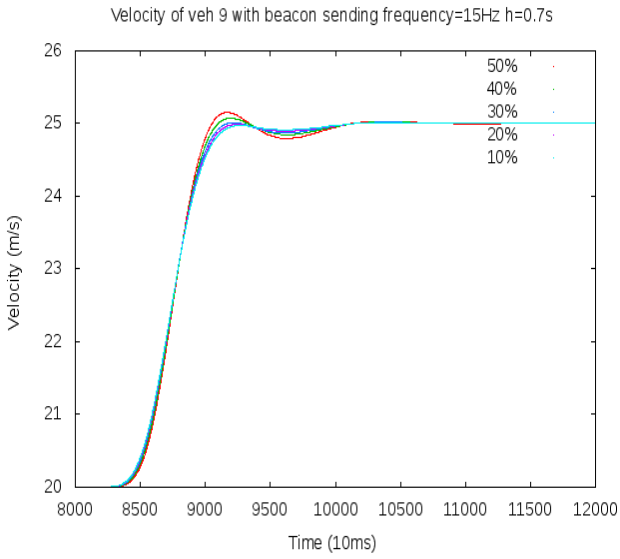
Figure 23 and Figure 24 show the curves associated with velocity and the acceleration, respectively, of the last following vehicle, i.e., veh9. Starting from left to right, the first curve is associated with the packet loss (PL) ratio of 10%, while the last curve is associated with the packet loss (PL) ratio of 50%.



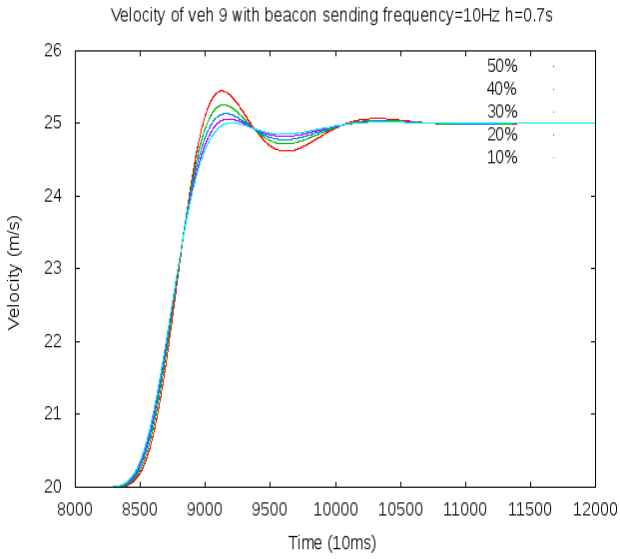
(a) velocity of veh 9 with beacon sending frequency=25Hz, h=0.7s



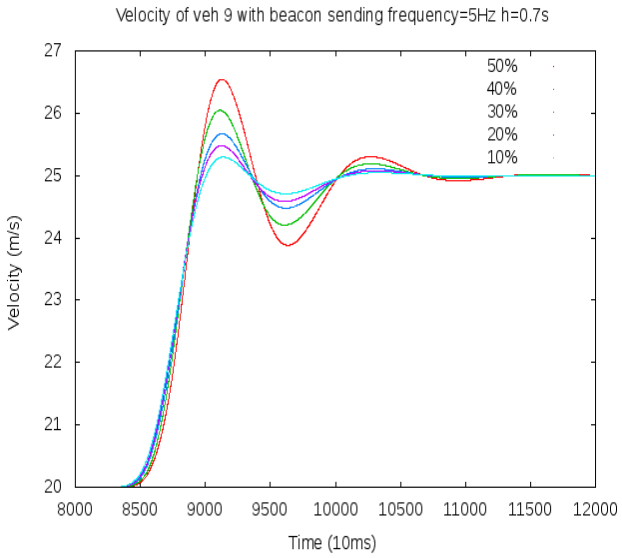
(b) velocity of veh 9 with beacon sending frequency=20Hz, h=0.7s



(c) velocity of veh 9 with beacon sending frequency=15Hz, h=0.7s

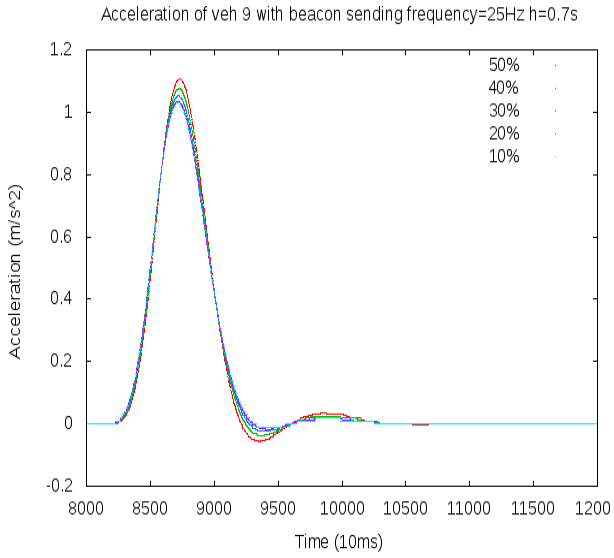


(d) velocity of veh 9 with beacon sending frequency=10Hz, h=0.7s

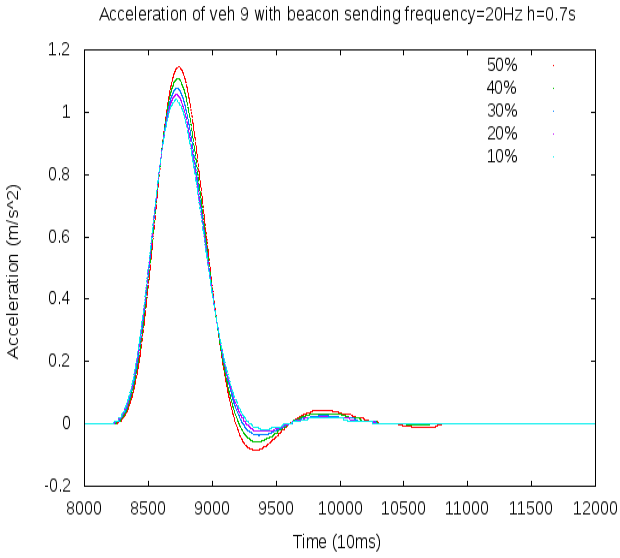


(e) velocity of veh 9 with beacon sending frequency=5Hz, h=0.7s

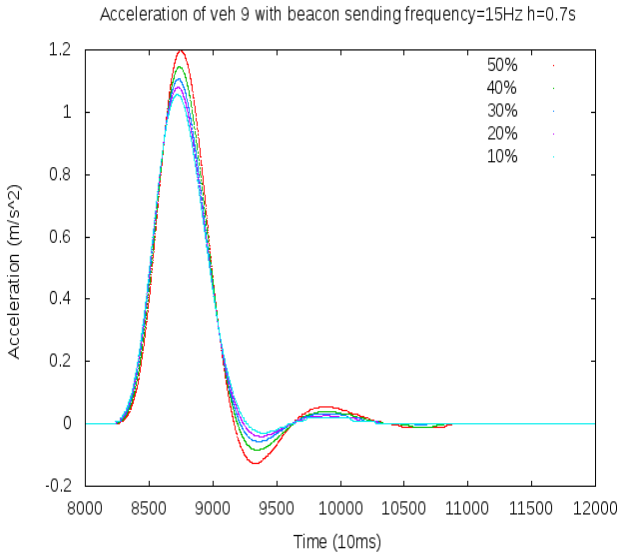
Figure 23: velocity of veh 9 in modified model for CACC with MiXiM (accelerating scenario)



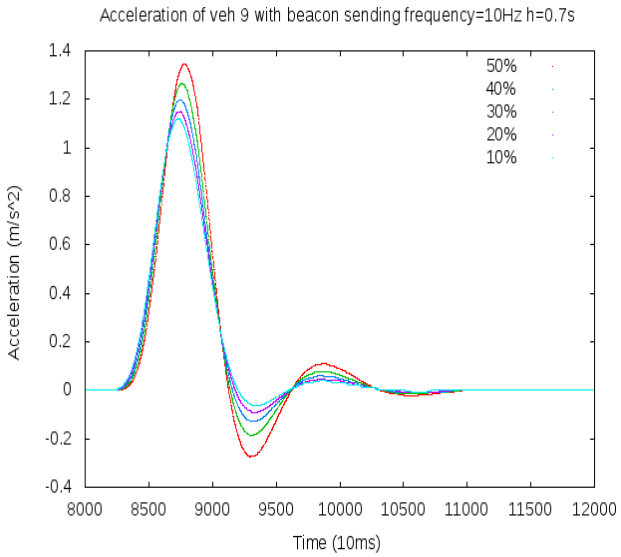
(a) acceleration of veh 9 with beacon sending frequency=25Hz, h=0.7s



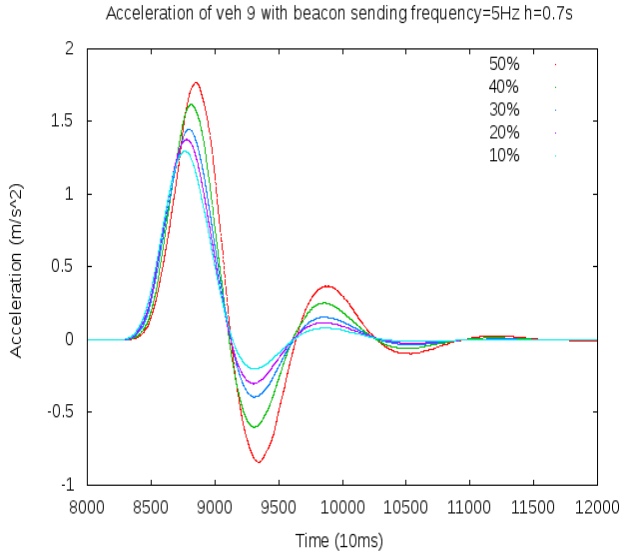
(b) acceleration of veh 9 with beacon sending frequency=20Hz, h=0.7s



(c) acceleration of veh 9 with beacon sending frequency=15Hz, h=0.7s



(d) acceleration of veh 9 with beacon sending frequency=10Hz, h=0.7s



(e) acceleration of veh 9 with beacon sending frequency=5Hz, h=0.7s

Figure 24: acceleration of veh 9 in modified model for CACC with MiXiM (accelerating scenario)

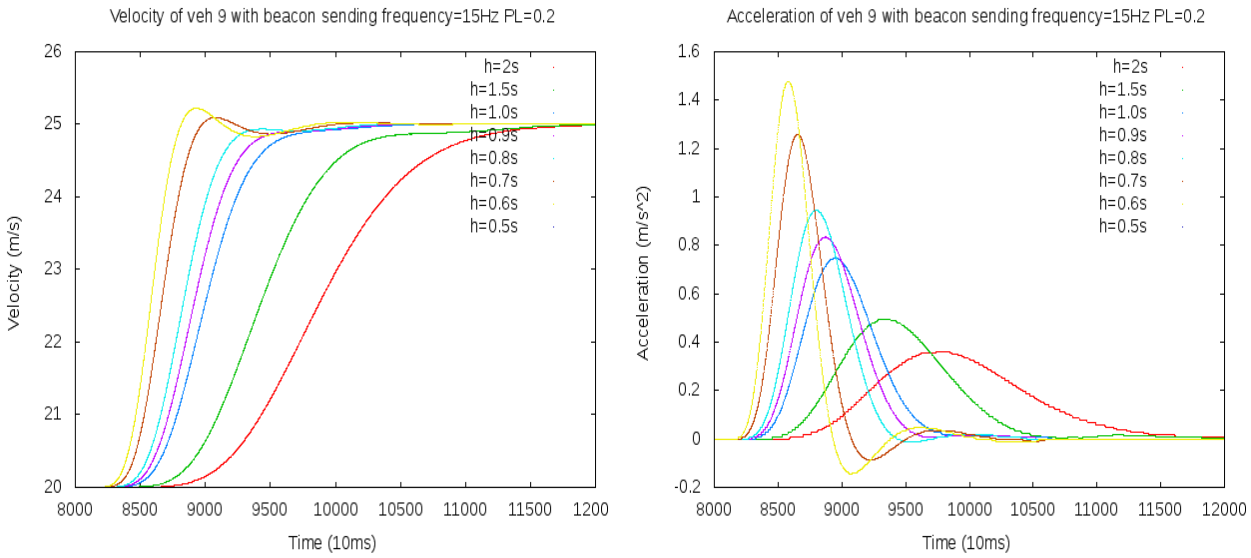
Similar conclusions are derived as the ones derived in Section 4.3.2.1.1. In particular, for a constant time headway value, as packet loss ratio increases and/or beacon sending rate decreases, the platoon becomes to be more string-instable, which means that the disturbances of a leading vehicle are being amplified.

4.3.2.2.2 Varying time headway

The way of how the time headway, packet loss ratio and beacon sending rate are chosen is the same as in the experiments described in Section 4.3.2.1.2.

Parts (a) and (b) of Figure 25 show the curves associated with velocity and acceleration, respectively, of the last following vehicle, i.e., veh9. Starting from left to right, the first curve is associated with the time headway of 0.5s, while the last curve is associated with the time headway of 2s.

Similar conclusions are derived as the ones derived in Section 4.3.2.1.2. In particular, when the packet loss ration and beacon sending rate are kept constant, as the time headway increases the platoon becomes to be more string-stable.



(a) velocity of veh 9 with beacon sending frequency=15Hz, PL=0.2

(b) acceleration of veh 9 with beacon sending frequency=15Hz, PL=0.2

Figure 25: velocity and acceleration of veh 9 in modified model for CACC with MiXiM with different time headway (accelerating scenario)

4.4 Combining CACC and ACC

Experiment Goal: Just as stated above, when the input acceleration is not updated in time, the string stability performance of the vehicle will decrease. In worst cases also accidents (vehicle crashes) might happen, because the CACC controller will base its decision on outdated acceleration information.

The goal of this experiment is to study whether the CACC and ACC controllers can be combined in such a way that the ACC controller gets the control responsibility of the vehicle in situations where the CACC controller is not anymore able to successfully provide this responsibility. In particular, this experiment studies whether a CACC – ACC controller Switching mechanism is able to switch from CACC to ACC in situation that the received acceleration information is too much outdated, and back from ACC to CACC when this situation is corrected. Note that due to time constraints, only some preliminary experiments have been accomplished. Therefore, this section includes only some preliminary conclusions associated with these experiments, without showing the obtained results.

4.4.1 Experiment parameters and analysis

In this set of preliminary experiments, the same topology as described in Section 4.3.1 is used.

Regarding the CACC-ACC controller switching mechanism, it is important to emphasize that it is not operating in the same way as the CACC-ACC controller switching mechanisms provided by TNO. The main reason of this is related to the fact that the CACC-ACC controller switching mechanism and the Kalman filters available in the original Simulink model provided by TNO could not be converted into C++ shared libraries. Moreover, the CACC-ACC controller switching mechanism used in the original Simulink model does not mention how to calculate the necessary inputs for the switching mechanism by using real-time parameters measured from the field.

The original Simulink model provided by TNO supports a requirement with respect to preceding vehicle(s) timing: information, which should not be older than 200 ms. If this timing information is older than 200 ms, then the host vehicle will not use the CACC controller, but will instead switch back to the ACC controller. For this reason the CACC controller requires an effective beaconing rate of at least 5 Hz in order to be active.

This experiment studies what happens if the above mentioned timing requirement is dropped, i.e., what happens when the CACC controller keeps using the received preceding vehicle(s) timing: information, even when this information is older than the required 200ms.

In particular, in this experiment we have set the time headway to 0.7s, the packet loss ratio to 0.5, and beaconing rate to 1Hz. Furthermore, the CACC-ACC controller switching mechanism used in this set of experiments is triggered based on the value of the measured time headway value. In particular, two thresholds for time headway are specified. If the time headway is below 99.85% of the required time headway, the vehicle chooses to use the ACC controller. When the time headway is equal to 100% of the required time headway or higher, then the controller switching mechanism returns back to the CACC controller mode.

Two sets of experiments are accomplished. In both sets of experiments a decelerating scenario is used, where the values of the velocity and acceleration are set in an identical way as the experiments described in Section 4.3.2.1.

In the first set of experiments, the proposed CACC-ACC controller switching mechanism described above is not used. In this set of experiments 10 simulation runs, (using the same parameters, but for each run using different random seeds) are performed. During all these simulation runs, the leading vehicle is decelerating in the same way as described in the experiments described in Section 4.3.2.1. During this set of experiments it has been observed during 4 simulation runs that due to the deceleration of the leading vehicle a number of vehicles accidents (vehicle crashes) occur.

In the second set of experiments, the proposed CACC-ACC controller switching mechanism described above is used. All the other parameters and number of simulation runs are the same as the ones used for the first set of experiments. During this set of experiments it has been observed during 2 simulation runs (instead of the 4 runs)

that due to the deceleration of the leading vehicle a number of vehicles accidents (vehicle crashes) occur.

Furthermore, using a CACC-ACC controller switching mechanism could help solving this problem. However, for the studied CACC-ACC controller switching mechanism in these experiments, it seems to help solving the string instability, but for the chosen parameter values for time headway, packet loss ratio and beacon sending rate this controller switching mechanism is not operating satisfactorily. More and extensive experiments are needed to optimize the operation of such a CACC-ACC controller switching mechanism.

In addition to the time headway a CACC-ACC controller switching mechanism can also be triggered by other parameters. For example, the beacon receiving rate could be used for this purpose. The receiving vehicle can set a beacon receiving rate threshold and maintain a counter that counts the number of beacons received every second. If this measured beacon receiving rate is lower than the set threshold then the controller switching mechanism could select the ACC mode. Another beacon receiving rate could be used to trigger to switch from the ACC controller mode to the CACC mode.

Another solution that could be used to optimize the operation of such a CACC-ACC controller switching mechanism is to trigger the switching operation based on a combination of parameters, e.g., time headway and beacon receiving rate. Moreover, similar to the original Simulink model provided by TNO, Kalman filters and the freshness of the received preceding's vehicle timing information could be included and used by such switching mechanisms. In addition to the switching operation between CACC-ACC controllers such a controller switching mechanism could increase/decrease the desired time headway depending on wireless communication conditions.

From these preliminary experiments it can be observed that when the beaconing rate is too low and the packet loss is too high then a CACC controller cannot guarantee string-stability and is not able to prevent accidents (vehicle crashes) from occurring. Note that the original Simulink model provided by TNO, uses the value of 200 ms to define whether the received preceding's vehicle timing information is either fresh or outdated. If this timing information is older than 200 ms, then the host vehicle will not use the CACC controller, but will instead switch back to the ACC controller. For this reason the CACC controller requires an effective beaconing rate of at least 5 Hz in order to be active. The above experiments justify that the beaconing rate should not be lower that a certain value. More extensive experiments are needed to find the optimum beaconing rate value applied under certain packet loss percentages.

More work and experiments are needed to develop and evaluate such a CACC-ACC controller switching mechanism.

4.5 Conclusion

In this section, the accomplished experiments are described and analyzed. It can be concluded that the original Simulink model provided by TNO and the modified model developed in this assignment and implemented as an integrated simulation model are from the point of view of string stability, reasonably equivalent on how they operate. The main differences between these models are that (1) another CACC-ACC controller switching mechanism is used in the modified model, (2) the Kalman filters used in the original Simulink model are not used in the modified model used in this assignment. Another conclusion that has been derived is that when a CACC controller is used instead of an ACC controller then the string stability performance is significantly increased.

Furthermore, based on the experiments where only the integrated modified model (built on SUMO and OMNeT++/MiXiM) was used, the following conclusions are derived. When the time headway value is kept constant, as packet loss ratio increases and/or beacon sending rate decreases, the platoon becomes to be more string-instable, which means that the disturbances of a leading vehicle are being amplified. Moreover, when the

packet loss ration and beacon sending rate are kept constant, as the time headway increases the platoon becomes to be more string-stable. Furthermore, it can also be concluded that the use of only a CACC controller can be dangerous in some situations, which could become a cause of vehicle crashes. Therefore, a combination of a CACC and ACC controller operation is required. This can only be accomplished by using a CACC-ACC controller switching mechanism that would operate in such a way to guarantee that vehicle crashes caused by the imperfections of the wireless communication link are avoided.

5. Conclusions and Future Work

5.1 Conclusions

In this assignment the impact of ACC and CACC that uses a realistic communication medium on the string stability performance has been investigated. This has been realized by answering the research questions that are listed in Section 1.4. In particular, the needed control theory used by the ACC and CACC controllers has been briefly described in Section 2. Subsequently, the used simulation environments and models used in this assignment have been described in Section 3. A simulation model has been realized that integrates different simulation models that were originally implemented in the SUMO, Simulink and OMNET++/MiXiM simulation environments. In particular, an original Simulink model of the ACC and CACC controllers that has been provided by TNO has been successfully converted into a shared C++ library so that it could be simulated within the traffic simulator SUMO. Moreover, using an existing IEEE 802.11p communication model provided by UT/DACS, implemented in the OMNET++/MiXiM simulation environment it was possible to build the integrated simulation model and to test the performance of the CACC controller. After realizing this integrated simulation model, several sets of simulation experiments were performed and analyzed based on this simulation model, see Section 4. In particular, the first set of experiments compares the original Simulink model and the modified and integrated simulation model. In another set of experiments the ACC and CACC string stability performance was compared assuming that the CACC controller was able to receive information using an ideal communication link, i.e., no packet losses and no delays. Subsequent sets of experiments analyzed the CACC string stability performance, considering that the CACC was using an IEEE 802.11p communication medium. In the last set of experiments a combined ACC and CACC controller model has been investigated and some preliminary conclusions were derived.

The main conclusions derived within this assignment are:

- The original Simulink model provided by TNO and the modified model developed in this assignment and implemented as an integrated simulation model are from the point of view of string stability, reasonably equivalent on how they operate. The main differences between these models are that (1) another CACC-ACC controller switching mechanism is used in the modified model, (2) the Kalman filters used in the original Simulink model are not used in the modified model used in this assignment.
- When a CACC controller is used instead of an ACC controller than the string stability performance is significantly increased.

Based on the experiments where only the integrated modified model (built on SUMO and OMNeT++/MiXiM) was used, the following conclusions are derived:

- When the time headway value is kept constant, as packet loss ratio increases and/or beacon sending rate decreases, the platoon becomes to be more string-instable, which means that the disturbances of a leading vehicle are being amplified.
- When the packet loss ration and beacon sending rate are kept constant, as the time headway increases the platoon becomes to be more string-stable.
- The use of only a CACC controller can be dangerous in some situations, which could become a cause of vehicle crashes. Therefore, a combination of a CACC and ACC controller operation is required. This can only be accomplished by using a CACC-ACC controller switching mechanism that would operate in such a way to guarantee that vehicle crashes caused by the imperfections of the wireless communication link are avoided.

5.2 Future Work

Based on the conclusions derived in this assignment several recommendations for future activities have been identified see below:

- More work and extensive experiments are needed to develop and evaluate an optimal CACC-ACC controller switching mechanism that will guarantee that the imperfections of a communication medium will not become a cause for vehicle crashes.
- Some features that were available in the CACC/ACC Simulink model provided by TNO, including Kalman filters, are not implemented in this assignment. It is recommended to implement these features in the modified model developed in this assignment and investigate whether the conclusions derived in this assignment regarding the CACC string stability performance still hold.
- In this assignment the packet loss ratio has been emulated at each receiving vehicle, by either dropping or accepting a received beacon depending on a predefined packet loss probability. It is recommended to use IEEE 802.11p background traffic in such a way that the packet loss ratio and beacon delays are obtained by varying the network conditions (via this background traffic).
- Improving the communication part should be able to increase the beacon receiving ratio. The improved beacon generating schemes stated in [EeKa10] which aimed at decreasing beacon collisions to improve beacon throughput should be tested in future. Of course, other improvement in the network protocols can be tested.
- At this moment, the CACC/ACC controllers just support the case of a single lane road. In the future it will be needed to investigate also network topologies where multilane roads and other complex traffic conditions are used.
- More complex communication infrastructures could be used, e.g., including Road Side Units, for the dissemination of the beacons.

Acknowledgements

We would like to thank Jeroen Ploeg (from TNO) for providing the Simulink models of the ACC and C-ACC controllers and for helping realizing this assignment. Moreover, we would like to thank Geert Heijenk for his valuable comments on this assignment.

References

[ArTa03] B. Van Arem, C. Tampere, and K. Malone, “Modelling traffic flows with intelligent cars and intelligent roads,” in Proc. of IEEE IV, June 2003, pp. 456–461.

[BaHa04] R. Barr, Z. J. Haas, and R. van Renesse, “JiST: Embedding Simulation Time into a Virtual Machine”, In Proc. of EuroSim Congress on Modelling and Simulation, 2004.

[BoIo01] Arnab Bose and Petros Ioanno , “Analysis of traffic flow with mixed manual and intelligent cruise control vehicles: theory and experiments”, California PATH Res. Rep. UCB-ITS-PRR-2001-13, 2001.

[BrEs00] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, “Advances in network simulation”, IEEE Computer, 33(5):59–67, May 2000.

[C&D] Project Connect&Drive, Dutch NL Agency/HTAS (High Tech Automotive Systems), Project no. HTASD08002, to be found via (URLs visited in January 2011): http://www.htas.nl/?pid=111, http://www.ctit.utwente.nl/research/projects/national/senter/connect-drive.doc/

[CORSIM] CORSIM: Microscopic Traffi c Simulation Model (visited in January 2011) http://www-mctrans.ce.ufl.edu/featured/TSIS/Version5/corsim.htm

[DjSo08] D. Djenouri, W. Soualhi and E. Nekk, “VANET's Mobility Models and Overtaking: An Overview”, In Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on, pages 1–6, April 2008.

[EeKa10] Martijn van Eenennaam, Georgios Karagiannis and Geert Heijenk, “Towards Scalable Beaconing in VANETs”, in Fourth ERCIM workshop on eMobility, Luleå Sweden. Luleå, Sweden: Luleå University of Technology, Luleå, Sweden, May 2010, pp. 103–108.

[EiSc06] S. Eichler, C. Schroth and J. Eberspächer, “Car-to-Car Communication”, In Proc. of the VDE-Kongress - Innovations for Europe, October 2006.

[Escu0] B. Escurado, “The Theory of Vehicular Ad-Hoc Network”, 2008. (visited in January 2011) http://techviewz.org/2008_02_01_archive.html

[HaFi07] Jérôme Härri, Fethi Filali and Christian Bonnet, “Mobility Models for Vehicular Ad Hoc Networks: A Survey and Taxonomy”, Technical Report RR-06-168, Institut Eurecom, January 2007

[IEEE802.11p-2010] IEEE Computer Society. IEEE P802.11p "IEEE Standard for Information technology--Telecommunications and information exchange between systems--Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments", June 2010.

[FDK] The Federated Simulations Development Kit (FDK) (visited in January 2011) http://www-static.cc.gatech.edu/computing/pads/fdk.html

[Ioan97] P. E. Ioannou, Automated Highway Systems, Plennum Press New York, ISBN: 0-306-45469-6, 1997.

[Jain91] R. Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling," Wiley- Interscience, New York, NY, April 1991, ISBN:0471503361.

[KaMo07] F. Karnadi, Z. Mo and K.-C. Lan, "Rapid Generation of Realistic Mobility Models for VANET", in Proc. of the IEEE Wireless Communication and Networking Conference (WCNC'07), March 2007.

[KöSw08] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. K.Haneveld, T. Parker, O.Visser, H. S. Lichte, and S. Valentin, "Simulating wireless and mobile networks in OMNeT++: The MiXiM vision. In Proc. Intl. Workshop on OMNeT++ (co-located with SIMUTools '08), Mar. 2008

[Kuma10] Sisil Kumarawadu, "Control systems : theory and implementation", Alpha Science International, ISBN: 978-1-8426-5605-1, 2010.

[Matlab] Matlab introduction in the mathworks official website (visited in January 2011) http://www.mathworks.com/products/matlab/description1.html

[Matlab_Simulink] Simulink introduction in mathworks official website (visited in January 2011) http://www.mathworks.com/products/simulink/description1.html

[Matlab_rtw] Real-Time Workshop in mathworks official website (visited in January 2011) http://www.mathworks.com/products/rtw/

[MiXiM] MiXiM introduction in the sourceforge website (visited in January 2011) http://mixim.sourceforge.net/

[MiXiM_sommer] modified MiXiM by Christoph Sommer (visited in January 2011) https://github.com/sommer/mixim-sommer.git

[NaVu09] G. Naus , R. Vugts , J. Ploeg , R. Vd Molengraft and M. Steinbuch, "Towards On-the-road Implementation of Cooperative Adaptive Cruise Control", Proceedings of the 16th World Congress & Exhibition on Intelligent Transport Systems and

Services (ITS World 2009), Stockholm, Sweden, September 21–25, 2009.

[NS2] NS2 (Network Simulator – 2) official website, visited on 10th of February 21011, http://www.isi.edu/nsnam/ns/

[Omnetpp] OMNeT++ official website (visited in January 2011) http://www.omnetpp.org

[OPNET] OPNET official website (visited in January 2011) http://www.opnet.com/solutions/network_rd/modeler.html

[Omnetpp_manual] OMNeT++ User Manual (visited in January 2011) http://www.omnetpp.org/doc/omnetpp41/manual/usman.html

[POSTECH] website of Mobile networking lab of POSTECH (visited in January 2011) http://monet.postech.ac.kr/research.html

[PuAr10] Rattaphol Pueboobpaphan and Bart van Arem, "Understanding the relation between driver and vehicle characteristics and platoon and traffic flow stability for design and assessment of cooperative adaptive cruise control", Transportation Research Record, 2010(Accepted).

[ReMi02] D. Reichard, M. Miglietta, L. Moretti, P. Morsink, and W. Schultz, "Cartalk2000: safe and comfortable driving based upon inter-vehicle communication," in IV, 2002, pp. 545–550.

[SoYa08] Christoph Sommer, Zheng Yao, Reinhard German and Falko Dressle, "On the Need for Bidirectional Coupling of Road Traffic Microsimulation and Network Simulation", 9th ACM Int'l. Symp. Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2008): 1st ACM Int'l. Wksp. Mobility Models for Networking Research (MobilityModels'08), Hong Kong, China: ACM, May 2008.

[SoHo06] A. Sobeih, J. C. Hou, L.-C. Kung, N. Li, H. Zhang, W.-P. Chen, H.-Y. Tyan, and H. Lim. J-Sim: a simulation and emulation environment for wireless sensor networks. IEEE Wireless Communications, 13(4):104–119, 2006.

[SoGe11] C. Sommer, R. German, F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis", IEEE Transactions on Mobile Computing, Vol. 10, No. 1, January 2011.

[SUMO] SUMO introduction in the sourceforge website (visited in January 2011) http://sourceforge.net/apps/mediawiki/sumo/

[TNO-safety] TNO Defence, Safety and Security, (visted in February 2011) www.tno.nl

[TraNS] TraNS official website (visited in January 2011)
http://lca.epfl.ch/projects/trans

[WiKl07] I.R. Wilmink, G.A. Klunder, and B. van Arem, “Traffic flow effects of integrated full-range speed assistance (IRSA)”, Intelligent Vehicles Symposium, 2007 IEEE, pages 1204–1210, 13-15 June 2007.

[Wiki_cruise_control] article of “cruise control” on Wikipedia
(visited in January 2011)
http://en.wikipedia.org/wiki/Cruise_control

[Wiki_80211p] article of “IEEE802.11p” in Wikipedia (visited in January 2011)
http://en.wikipedia.org/wiki/IEEE_802.11p

[UT/DACS] UT-DACS (Design and Analysis of Communication Systems) official web site (visited in February 2011)
http://www.utwente.nl/ewi/dacs/

[Veins] Veins official website (visited in January 2011)
http://veins.car2x.org

[VuOg07] Rama Vuyyuru and Kentaro Oguchi, ”Vehicle-to-Vehicle Ad Hoc Communication Protocol Evaluation using Simulation Framework”, in Proc. of the 4th IEEE/IFIP Wireless On demand Networks and Services, pp. 100-106, Austria 2007.

Appendix A: Confidence intervals associated with Section 4.3 experiments

This appendix describes the two-sided 90% confidence intervals calculated for the average results of the experiments provided in Section 4.3. In order to calculate these confidence intervals, 10 simulation runs (where each of them uses a different random seed) have been accomplished for each experiment described in Section 4.3. This means that for each point in each curve presented in the figures given in Section 4.3, 10 samples are found, where their average value is used to calculate the two-sided 90% confidence interval.

According to [Jain91], since the collected number of samples is lower than 30, in order to find the two-sided 90% confidence intervals, the t-student distribution is used for the calculation of these confidence intervals. According to [Jain91], the 100(1-α)% confidence interval is given by:

$$\left(\bar{x} - \frac{t_{\left[1-\frac{\alpha}{2};n-1\right]}s}{\sqrt{n}}, \bar{x} + \frac{t_{\left[1-\frac{\alpha}{2};n-1\right]}s}{\sqrt{n}}\right)$$

Here, $t_{\left[1-\frac{\alpha}{2};n-1\right]}$ is the (1-α/2)-quantile of a t-variate with n-1 degrees of freedom, where n represents the number of samples. These quantiles are listed in Table A.4 in [Jain91]. With n equal to 10 samples, n-1=9, and two-sided 90% confidence interval, we found $t_{[0.95;9]}$=1.833.

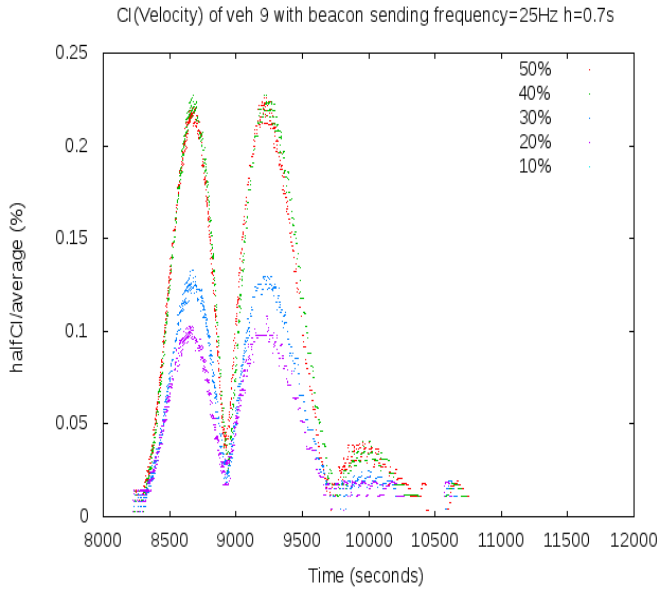
In order to be able to plot these confidence intervals in such a way that also the relation to their associated average value is shown, we decided to calculate and plot the ratios between each confidence interval and its associated average value. This is accomplished instead of just plotting the confidence interval and its average value in the same figure, which will make such a figure severely unclear.

For the velocity related experiments, we calculated the ratio of half of the CI range and its associated average value.

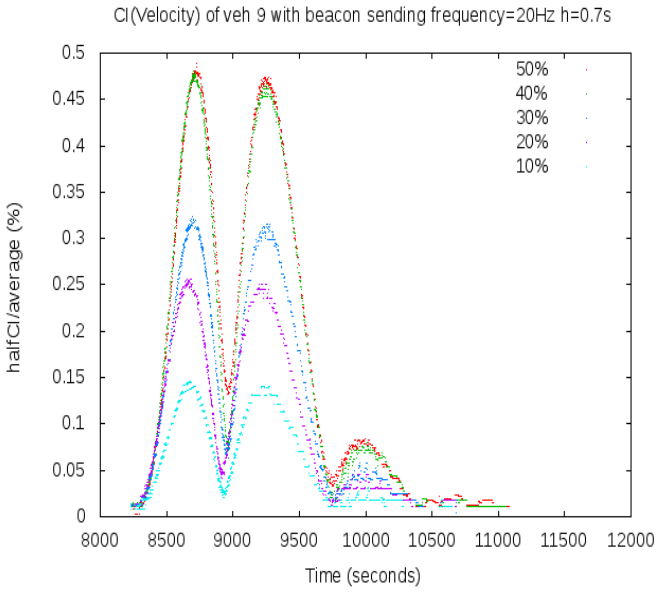
For the acceleration related experiments, it was not possible to calculate this ratio, since some average values equal to zero. Therefore, we just calculated and plotted only the absolute values of confidence interval.

For the decelerating scenario, see Section 4.3.2.1, the confidence intervals corresponding to Figure 20 are showed in Figure 26 and Table 2. The confidence intervals corresponding to Figure 21 are showed in Figure 27 and Table 3. Furthermore, the confidence intervals corresponding to Figure 22 are showed in Figure 28, Table 4 and Table 5.

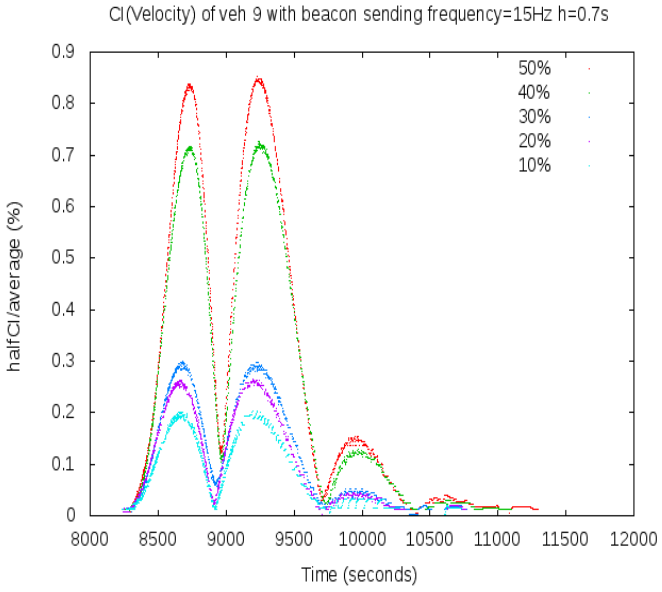
Note that in the below figures and tables, “h” denotes the time headway; “BSF” denotes the beacon sending frequency (rate); “PL” denotes the packet loss ratio and “CI” is denotes a confidence interval, which means the value of $\left(2*\frac{t_{\left[1-\frac{\alpha}{2};n-1\right]}s}{\sqrt{n}}\right)$.



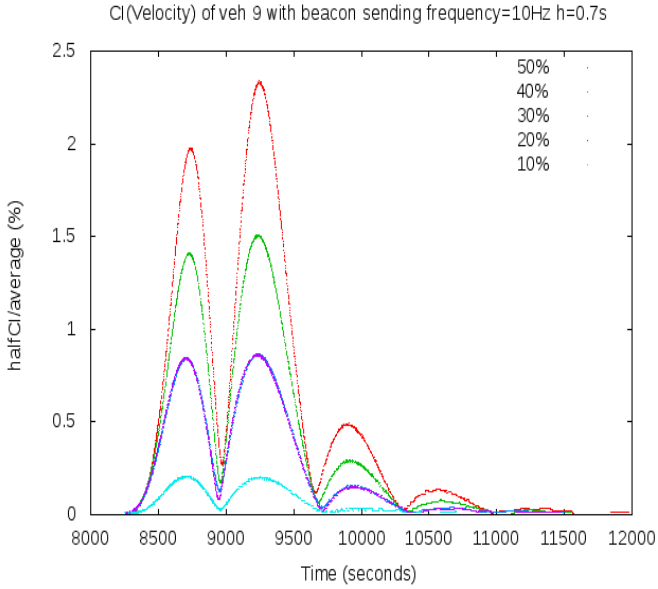
(a) CI(Velocity) of veh9 with beacon sending frequency=25Hz h=0.7s



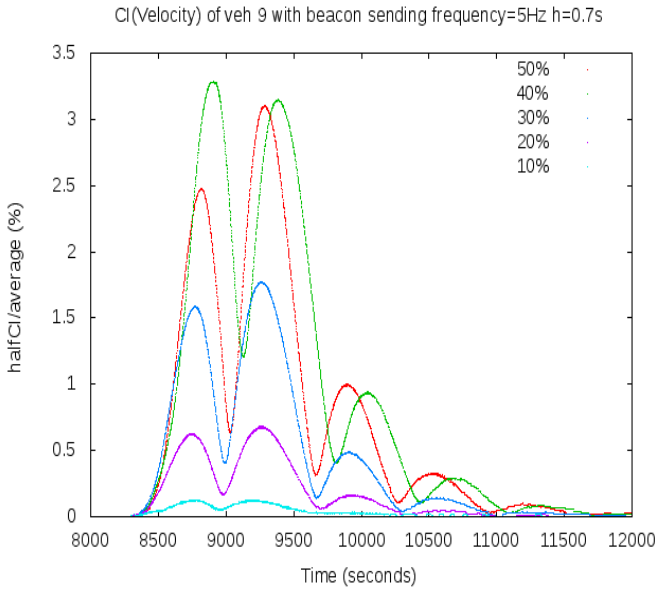
(b) CI(Velocity) of veh9 with beacon sending frequency=20Hz h=0.7s



(c) CI(Velocity) of veh9 with beacon sending frequency=15Hz h=0.7s



(d) CI(Velocity) of veh9 with beacon sending frequency=10Hz h=0.7s

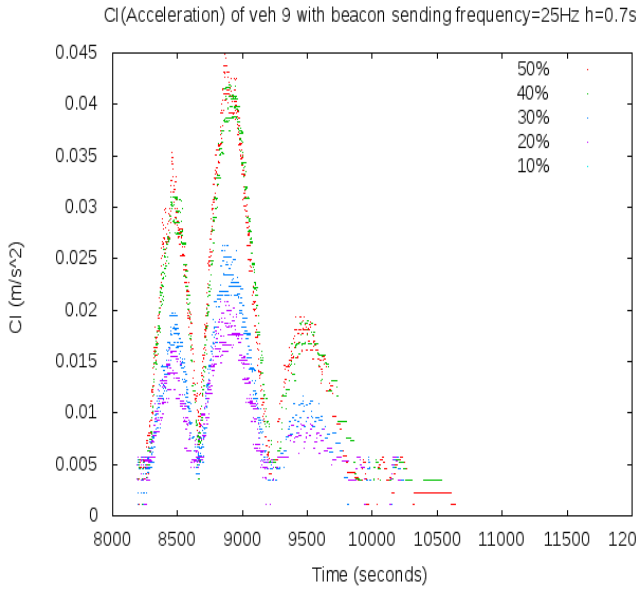


(e) CI(Velocity) of veh9 with beacon sending frequency=5Hz h=0.7s

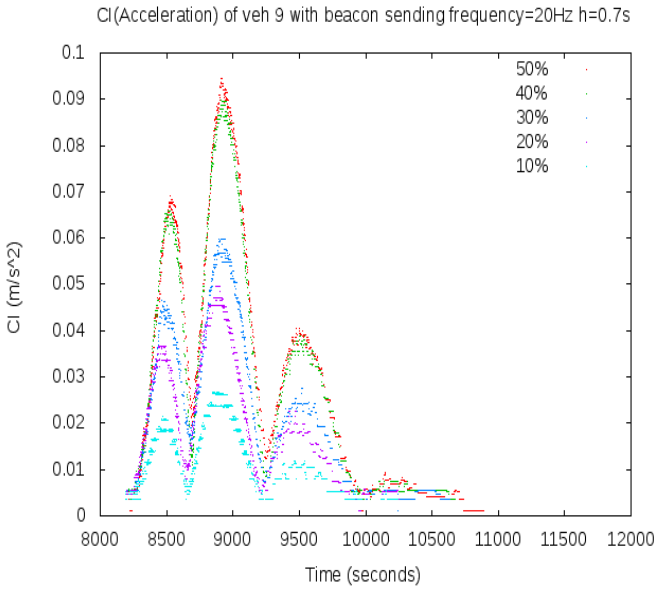
Figure 26: confidence interval corresponding to (a), (b), (c), (d), (e) of Figure 20

Table 2: half of maximum CI/average of veh 9 on velocity in decelerating scenario (%), h=0.7s

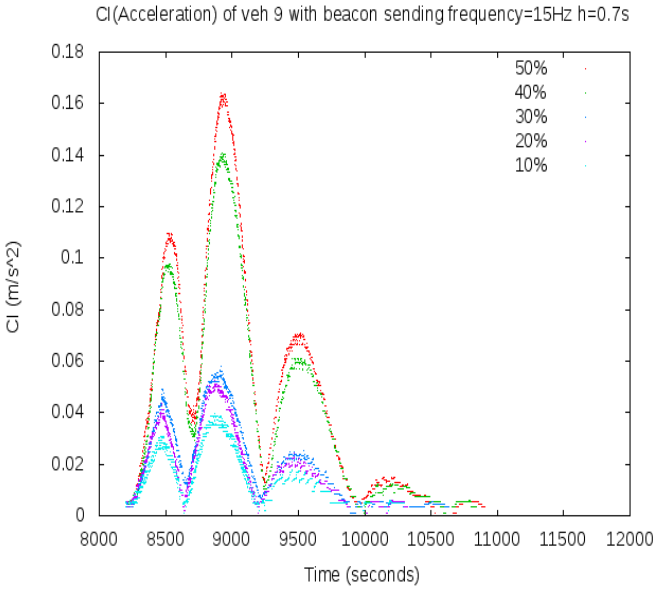
PL / BSF (Hz)	50%	40%	30%	20%	10%
25	0.2261	0.2273	0.133	0.1078	0
20	0.4891	0.4805	0.3237	0.2564	0.1454
15	0.8523	0.7275	0.2996	0.2659	0.2027
10	2.3387	1.5087	0.8689	0.8674	0.2099
5	3.1052	3.284	1.7768	0.6812	0.1286



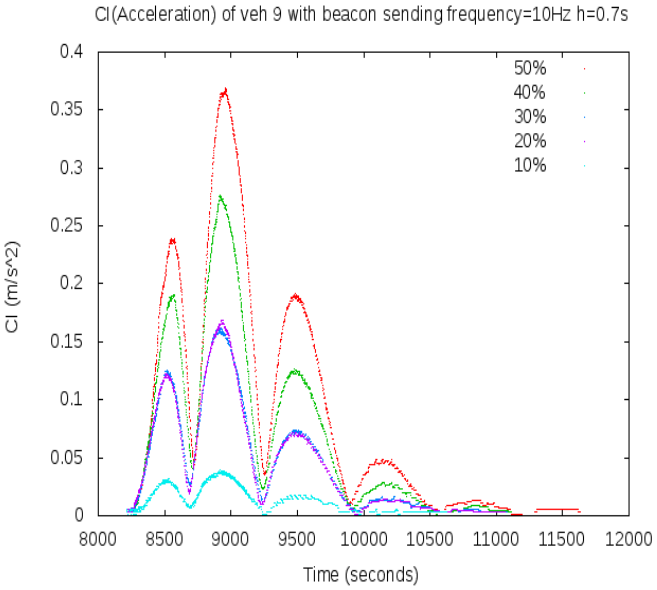
(a) CI(Acceleration) of veh9 with beacon sending frequency=25Hz h=0.7s



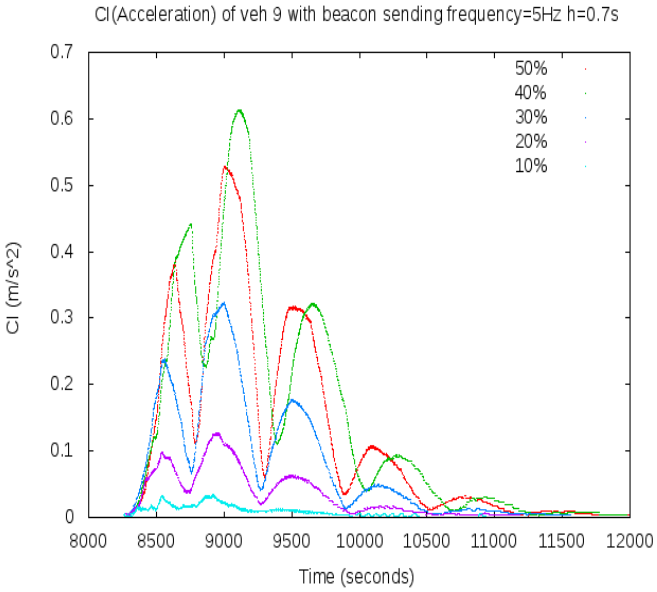
(b) CI(Acceleration) of veh9 with beacon sending frequency=20Hz h=0.7s



(c) CI(Velocity) of veh9 with beacon sending frequency=15Hz h=0.7s



(d) CI(Acceleration) of veh9 with beacon sending frequency=10Hz h=0.7s

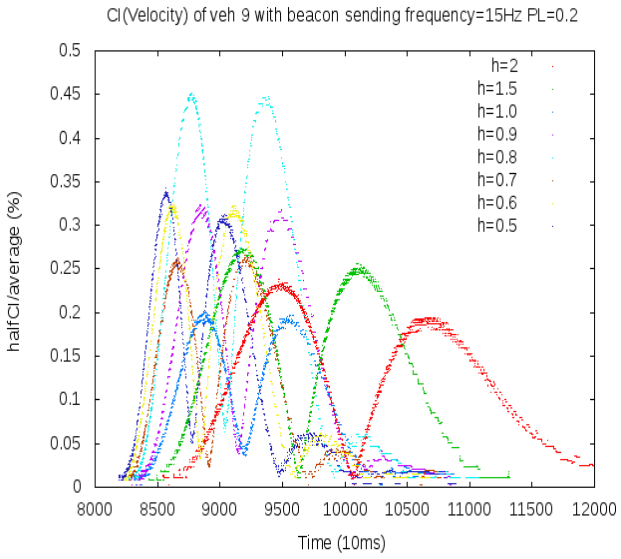


(e) CI(Acceleration) of veh9 with beacon sending frequency=5Hz h=0.7s

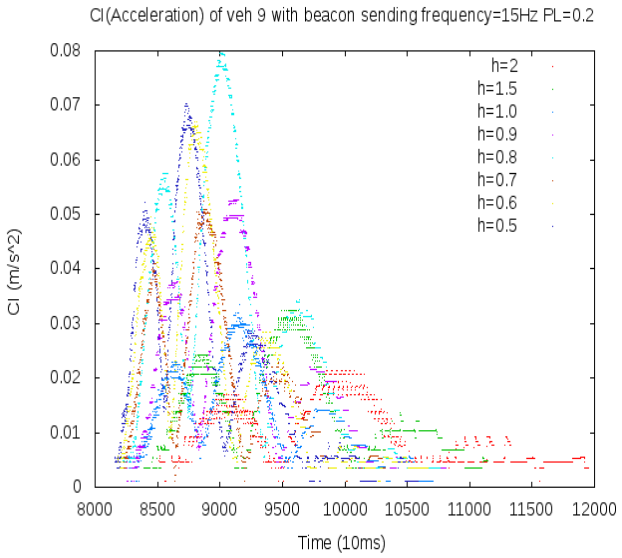
Figure 27: confidence interval corresponding to (a), (b), (c), (d), (e) of Figure 21

Table 3: maximum CI of veh 9 on acceleration in decelerating scenario (m/s^2), h= 0.7s

PL / BSF (Hz)	50%	40%	30%	20%	10%
25	0.0449	0.0419	0.0263	0.0208	0
20	0.0946	0.0897	0.0597	0.0497	0.0265
15	0.1641	0.1407	0.0581	0.0524	0.0399
10	0.3693	0.2771	0.1616	0.1685	0.0395
5	0.5293	0.614	0.3234	0.1274	0.0342



(a) CI(Velocity) of veh9 with beacon sending frequency=15Hz PL=0.2



(b) CI(Acceleration) of veh9 with beacon sending frequency=15Hz PL=0.2

Figure 28: confidence interval corresponding to (a), (b) of Figure 22

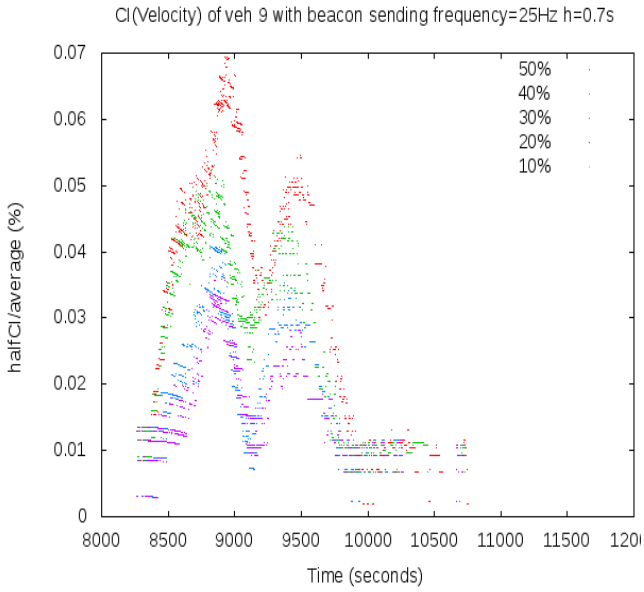
Table 4: half of maximum CI/average of veh 9 on velocity in decelerating scenario (%), BSF=15Hz, PL=20%

h (s)	2	1.5	1	0.9	0.8	0.7	0.6	0.5
	0.2383	0.2735	0.2005	0.3242	0.4523	0.2659	0.3266	0.3436

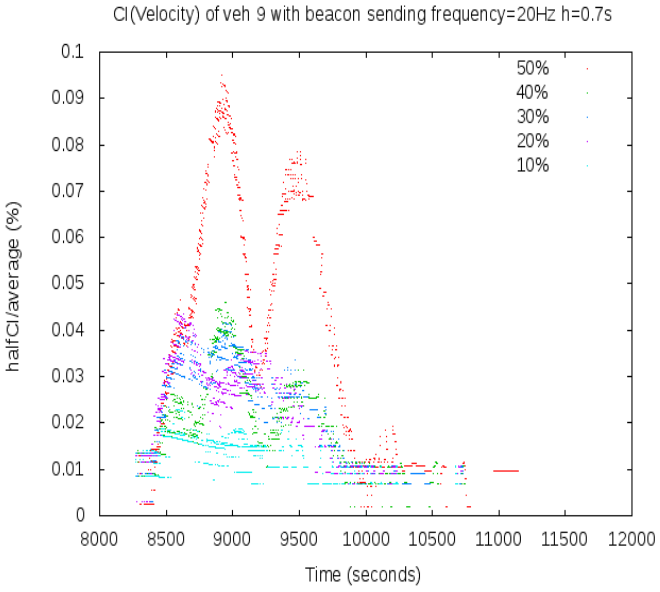
Table 5: maximum CI of veh 9 on acceleration in decelerating scenario (m/s^2), BSF=15Hz, PL=20%

h (s)	2	1.5	1	0.9	0.8	0.7	0.6	0.5
	0.0214	0.0323	0.0334	0.0527	0.08	0.0524	0.067	0.0703

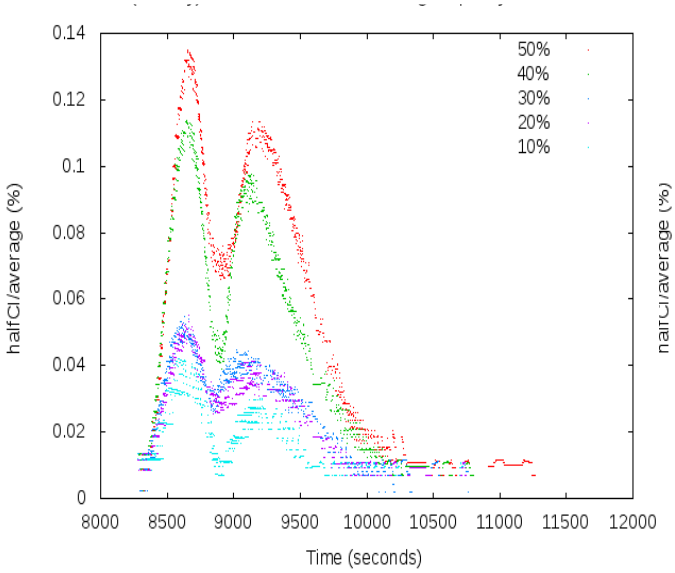
For the accelerating scenario, see Section 4.3.2.2, the confidence intervals corresponding to Figure 23 are showed in Figure 29 and Table 6. The confidence intervals corresponding to Figure 24 are showed in Figure 30 and Table 7. Furthermore, the confidence intervals corresponding to Figure 25 are showed in Figure 31, and Table 8 and Table 9.



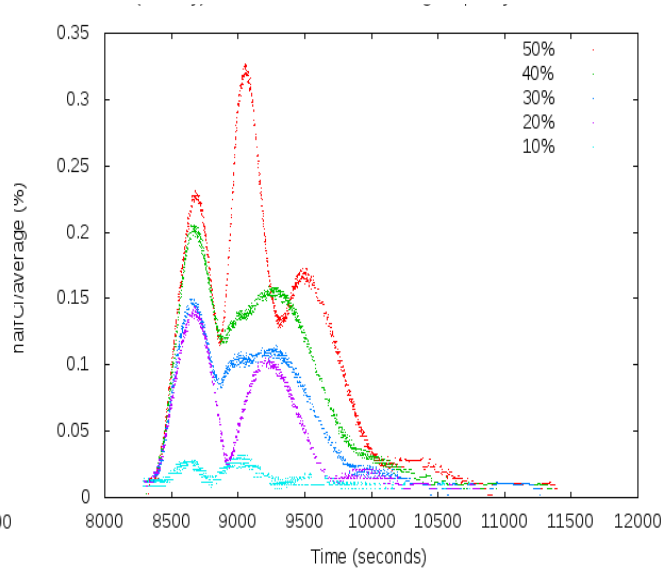
(a) CI(Velocity) of veh9 with beacon sending frequency=25Hz h=0.7s



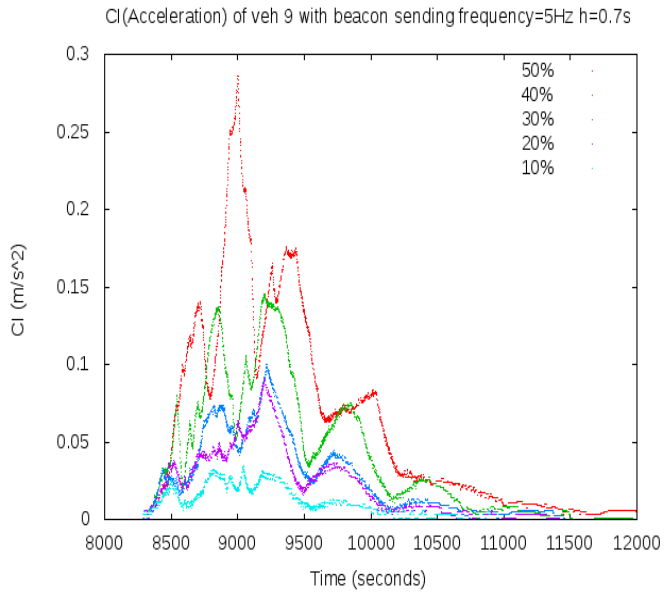
(b) CI(Velocity) of veh9 with beacon sending frequency=20Hz h=0.7s



(c) CI(Velocity) of veh9 with beacon sending frequency=15Hz h=0.7s



(d) CI(Velocity) of veh9 with beacon sending frequency=10Hz h=0.7s

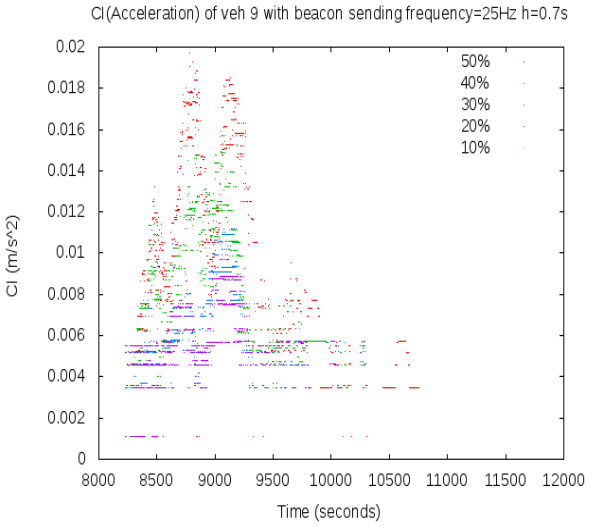


(e) CI(Velocity) of veh9 with beacon sending frequency=5Hz h=0.7s

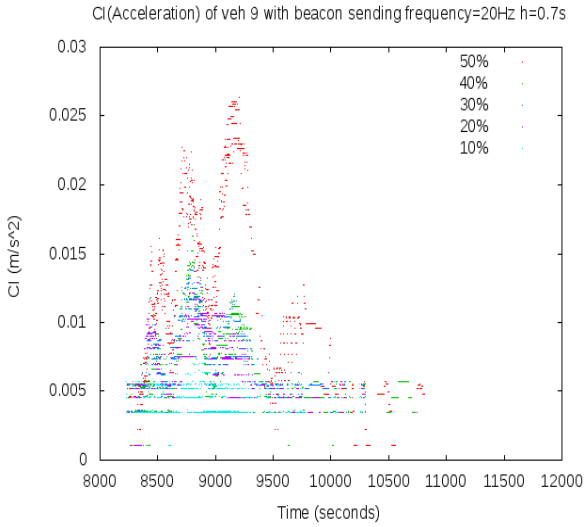
Figure 29: confidence interval corresponding to (a), (b), (c), (d), (e) of Figure 23

Table 6: half of maximum CI/average of veh 9 on velocity in accelerating scenario (%), h= 0.7s

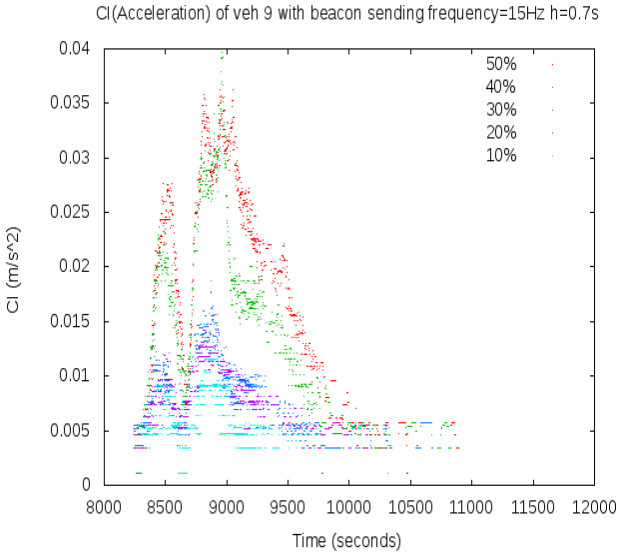
BSF (Hz) \ PL	50%	40%	30%	20%	10%
25	0.0694	0.0531	0.0422	0.0361	0
20	0.0949	0.0461	0.0416	0.0435	0.0229
15	0.135	0.1138	0.0548	0.055	0.0421
10	0.3267	0.2049	0.1497	0.1441	0.0321
5	0.6398	0.4346	0.2918	0.2563	0.1108



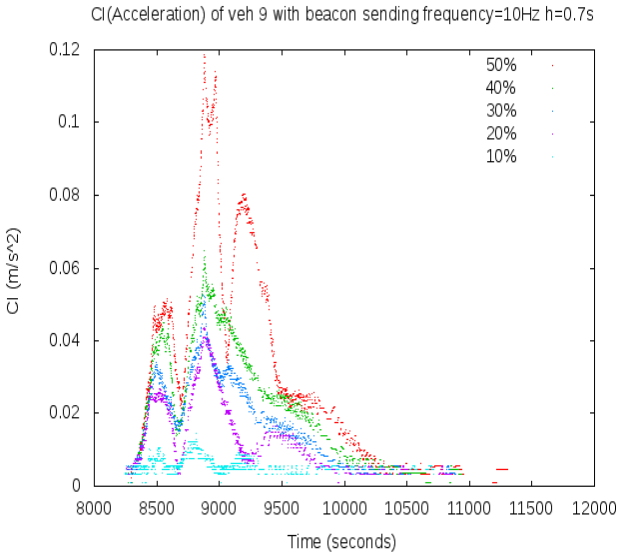
(a) CI(Acceleration) of veh9 with beacon sending frequency=25Hz h=0.7s



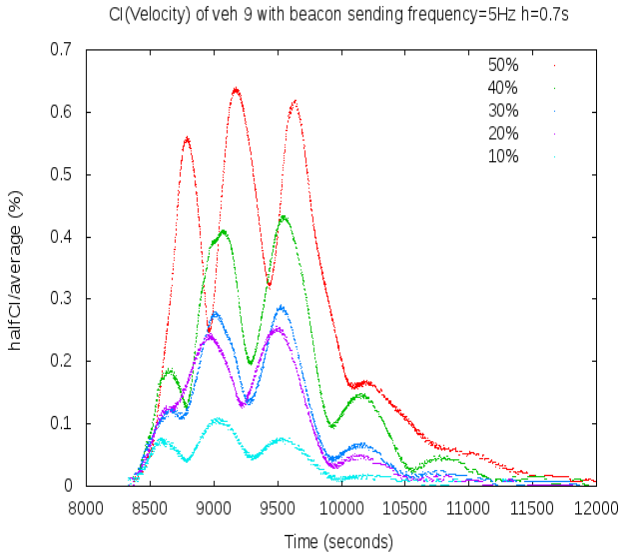
(b) CI(Acceleration) of veh9 with beacon sending frequency=20Hz h=0.7s



(c) CI(Acceleration) of veh9 with beacon sending frequency=15Hz h=0.7s



(d) CI(Acceleration) of veh9 with beacon sending frequency=10Hz h=0.7s

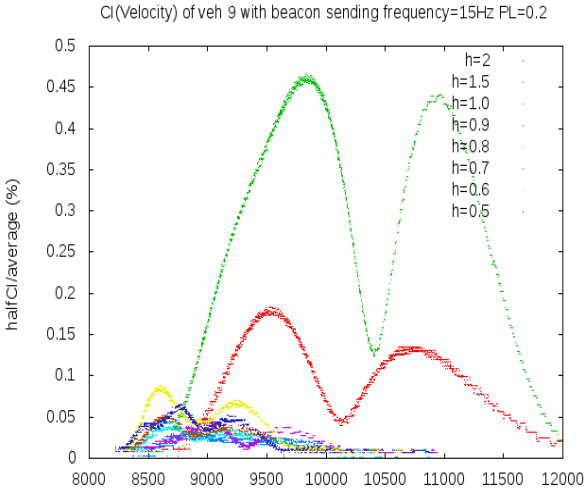


(e)CI(Acceleration) of veh9 with beacon sending frequency=5Hz h=0.7s

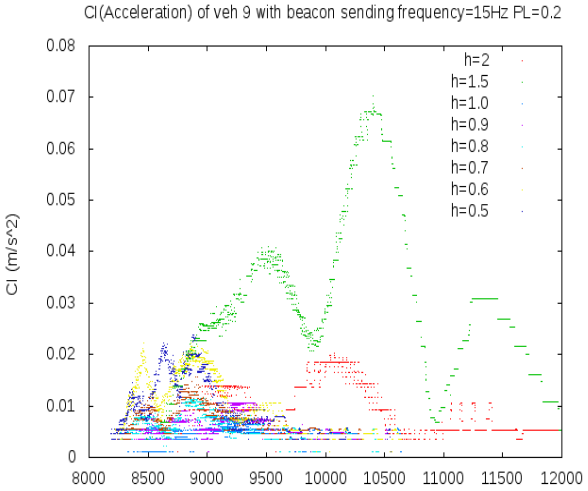
Figure 30: confidence interval corresponding to (a), (b), (c), (d), (e) of Figure 24

Table 7: maximum CI of veh 9 on acceleration in accelerating scenario (m/s^2), h=0.7s

PL / BSF (Hz)	50%	40%	30%	20%	10%
25	0.0197	0.0159	0.0125	0.0099	0
20	0.0263	0.0175	0.014	0.0132	0.0075
15	0.0363	0.0397	0.0188	0.0157	0.0123
10	0.1187	0.0647	0.0526	0.0436	0.0146
5	0.2861	0.1455	0.0997	0.0906	0.0344



(a) CI(Velocity) of veh9 with beacon sending frequency=15Hz PL=0.2



(b) CI(Velocity) of veh9 with beacon sending frequency=15Hz PL=0.2

Figure 31: confidence interval corresponding to (a), (b) of Figure 25

Table 8: half of maximum CI/average of veh 9 on velocity in accelerating scenario (%), BSF=15Hz, PL=20%

h (s)	2	1.5	1	0.9	0.8	0.7	0.6	0.5
	0.1833	0.4663	0.0331	0.0484	0.0455	0.055	0.0887	0.0658

Table 9: maximum CI of veh 9 on acceleration in accelerating scenario (m/s^2), BSF=15Hz, PL=20%

h (s)	2	1.5	1	0.9	0.8	0.7	0.6	0.5
	0.0203	0.0703	0.0093	0.0106	0.0123	0.0157	0.0223	0.0239

Conclusions:

This appendix shows that the statistical accuracy of the results associated with the experiments performed in Section 4.3 is good enough, since the ratio of CI and its average value is in most cases below 1%. Only in one case this ration is equal to 3.1%.

Appendix B: The Simulink Model

Appendix C: Guidelines for realizing experiments on SUMO-Simulink-OMNET++/MiXiM combined model

Appendix C: Guidelines for realizing experiments on SUMO-Simulink-OMNET++/MiXiM combined model

This appendix describes the guidelines for realizing the experiments given in the main report and are accomplished using the SUMO-Simulink-OMNET++/MiXiM combined model. This guideline is split into two parts: controller library generated part (procedures of how to generate a shared library by existent Simulink model) and bidirectional coupling part (procedure to couple SUMO and OMNeT++/MiXiM).

Note that:

- We will try to make it as clear as we can, however, if there are any questions, please contact Chenxi Lei (current email address: c.lei@student.utwente.nl)
- Also, here we will just explain the procedures exactly as they were done during the experiments, so it is possible that there are alternative ways to realize some sub procedures, such as the way of calling a shared library.

1. Controller library generated part:

Thanks to the Real-Time Workshop tool, we can generate the model built into the Simulink environment into C and/or C++ source code.

1.1 Background

A Linux version of Matlab is used to convert the original Simulink model into C++ source code. In particular, the Real-Time Workshop tool is used for this conversion. The converted C++ code is structured in C++ shared libraries that can be used by the Linux-based simulator—SUMO. A Linux version of Matlab is used, since it is difficult to build a shared library in the Linux-based simulator SUMO, using the source code generated from Real-Time Workshop tool embedded in a Windows version of Matlab.

1.2 Software specification

Because the latest version of the controller (original Simulink model) provided by TNO can only be used without any bug in Matlab 2010b and the current Linux OpenSUSE (version 10.3) distribution installed on the UT/DACS lab computers does not support the Matlab 2010b version, we had to use other operating systems or a higher version of a Linux kernel. Using the support of the UT/EWI ICT helpdesk we installed on a computer the operating system: **Ubuntu 10.04**. Using this operating system, we could install and use the **linux version of Matlab 2010b** (which can be found at http://www.mathworks.com/support/sysreq/current_release/linux.html).

1.3 C++ Code generation

First, one has to open Simulink model to be used (see Section 2 of Appendix B). Then in order to convert this Simulink model into C++ code by using the Real-Time Workshop tool, one has to configure the Real-Time Workshop parameters accordingly. One can get access to the configuration parameters by selecting “tools->Real-Time Workshop-> Options” . Figure 1 gives a screenshot of the dialog window used during the Real-Time Workshop configuration.

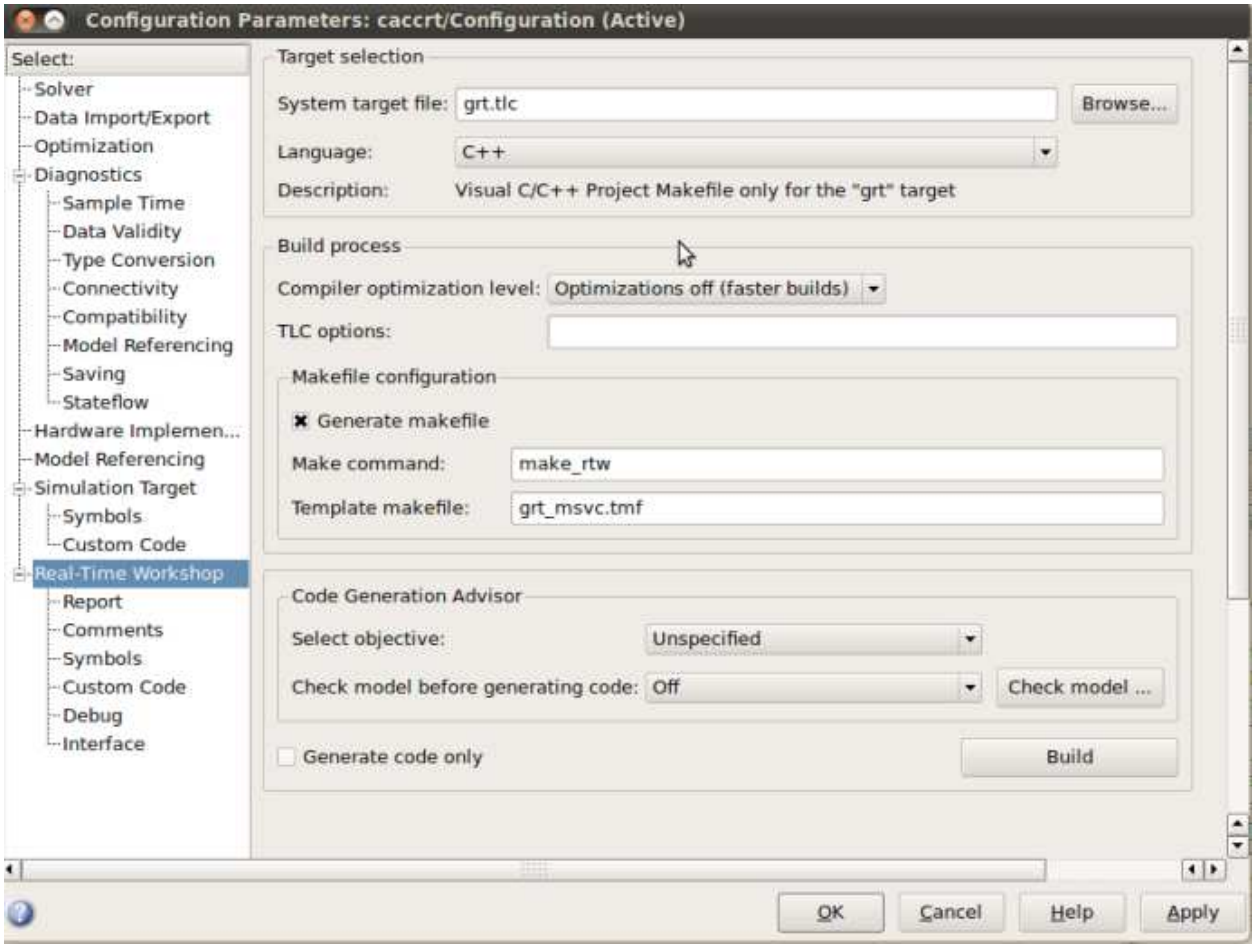


Figure 1: Real-Time Workshop Configuration

1.3.1 Real-*Time Workshop configuration: "system target file"*

In the "system target file" option in Figure 1 you can select which kind of coder you would like to use to generate the code by clicking the "Browse" button. Since we have to use the code in Linux environment (here we used Ubuntu 10.04), available options for compiling in Linux are

- -ert.tlc (Real-Time Workshop Embedded Coder)
- -grt.tlc (Generic Real-Time Target)

Moreover, because we do not have the license of "ert.tlc", the only choice for us is to use "grt.tlc".

1.3.2 Real-*Time Workshop configuration: "Language"*

The "Language" (see Figure 1) of the generated code is chosen as "C++", because the simulator (SUMO) we are going to apply is using (in calls) C++ shared libraries.

For other parameters in the tab "Real-Time Workshop", see Figure 1, we just use the default values. Of course, one can customize other parameters such as "compiler optimization level", see Figure 1, in order to make the compilation faster.

1.3.3 Real-Time Workshop—Solve configuration

In addition to the "Language", also the parameters associated with the "Solver" have to be configured, see Figure 2.

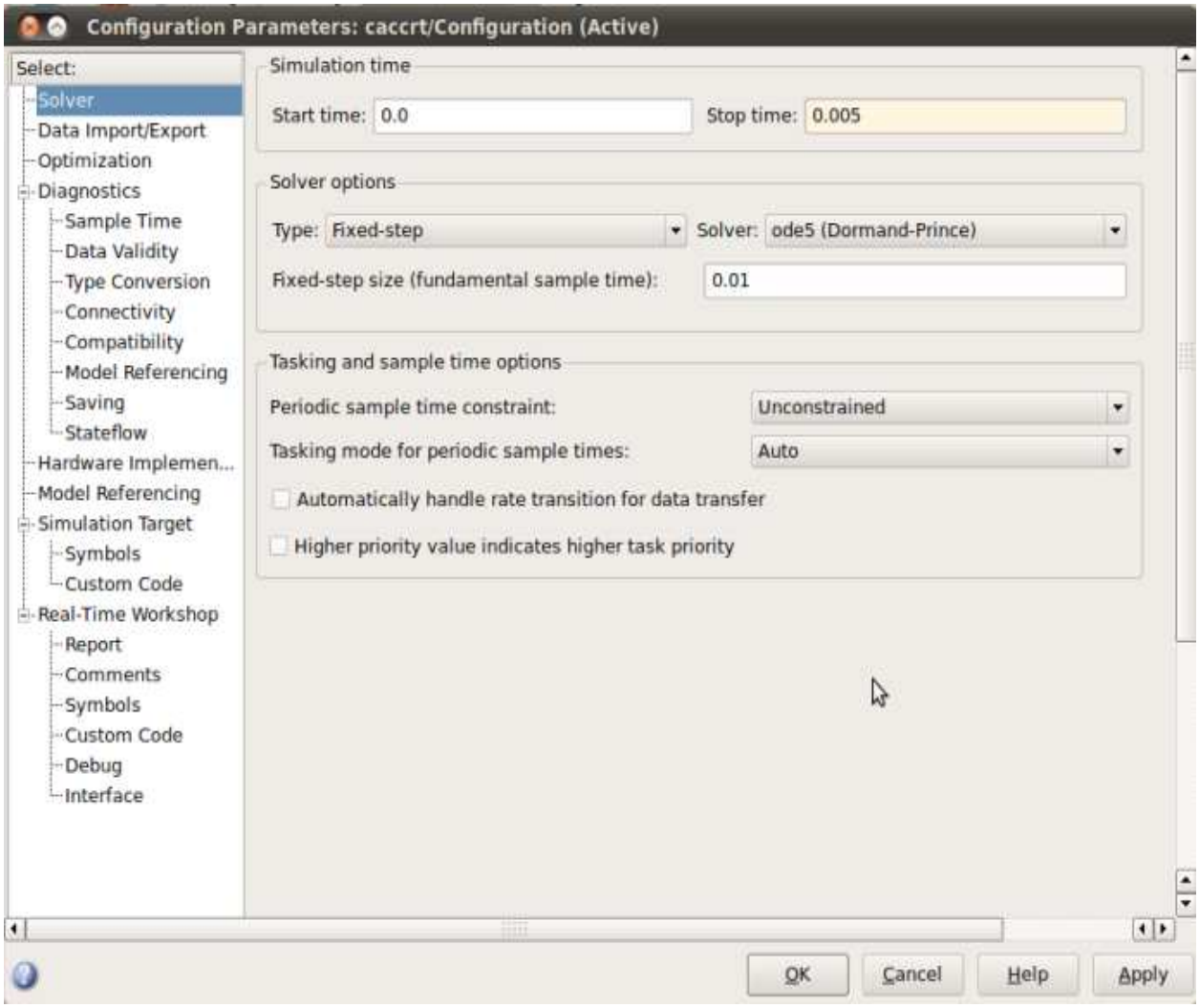


Figure 2: Solver Configuration

In the "Solver" tab (see Figure 2), "type" is chosen to be "Fixed-step" and "solver" is chosen to be "ode5".

Since all sample times in the model must be an integer multiple of the fixed-step size" (specified by Real-Time Workshop) and the sample time (sampling timestep) of the model is 0.01s, we just set the same value for the "fixed-step size". We set the stop time to 0.005s, since this stop time must be smaller than one sample timestep ("0.01"s). So every time we call the C++ shared library (built using the generated code), it will have the same effect as when the Simulink model would execute one run.

If you specify the stop time equal or higher than "0.01"s, then every time the C++ shared library is called, it will have the effect as when the Simulink model executes more than two runs. This will mean that for CACC, at the beginning of each run, the C++ shared library would use the same value for preceding vehicle's acceleration (one input of the controller). Thus every time the C++ shared library is called, the value for preceding vehicle's acceleration for all the runs after the first run will be out-dated.

In our experiments, we would like to evaluate the performance of CACC controller used in ideal communication circumstances, where the controller can always get "fresh" and up to date input every time it runs. So in each SUMO timestep, we specify that the CACC controller runs once by calling the C++ shared library once. In this way we can timely assign the fresh (and up to date) value of preceding vehicle's acceleration to the CACC controller. Therefore, we have to specify the value of stop time less than one sampling timestep. For other parameters, we just use the default values.

1.3.4 Other Considerations

After that, one can convert the whole Simulink model by selecting "tools->Real-Time Workshop-> Build Model". It is important to note that we encountered several problems during the process of converting the original Simulink model provided by TNO, into the C++ source code.

In the original Simulink model provided by TNO, two stateflow (charts) are used, which could not be converted. One stateflow (chart) is used in the "fault detection, isolation&recovery" module in the RT system, the other is used in the "from HMI" module, see Appendix B. The following error message was generated during the conversion process: "To build RTW with Stateflow blocks requires a valid Stateflow Coder license". It's still not clear whether a real license is needed or it might only be a bug. A similar problem occurred when we tried to convert the Kalman filters.

Therefore, we have just taken the pure ACC and CACC controllers from the RT control system and the "G_a" module that is used to revise the value generated by the ACC and CACC controllers from the "vehicle" module. Due to the converting problems with the Kalman filters, we were not able to use the "single target tracking" or "multiple target tracking" functions incorporated in the "target-tracking" block. This is because these functions are using the Kalman filters. The only "target-tracking" function that we could use is the "direct measurement" function. All details about the original Simulink model can be found in Section 1, Appendix B, and the modified Simulink model used in the experiments performed in this assignment can be found in Section 2, Appendix B.

Finally, we were able to generate the C++ code of the modified Simulink model, which does not include fault detection and host tracking functions. Note, before one converts the model, the parameters of the model can be loaded by running the parameters file. The original parameter file is named "caccrt_p.m" which was used together with the original Simulink model named "caccrt.mdl". The modified parameter file is named "testpartofcacc_p.m" used for modified Simulink models for controllers which supplied by this appendix. Note that the CACC controller models' names start with "trysome" while the ACC controller models' names start with "accwithout".

1.4 Code Modification and Share library generation

Though the C++ code could be successfully generated by Real-Time Workshop tool using parts of the original Simulink model provided by TNO, this C++ code cannot be applied within the SUMO environment directly without modifications.

1.4.1 short introduction of the generated code

As example, we use the Simulink CACC controller model that is included in a file denoted as "trysome". So a folder containing the generated code would be created with the name "trysome_grt_rtw". Figure 3 is a screenshot of the files contained in this folder.

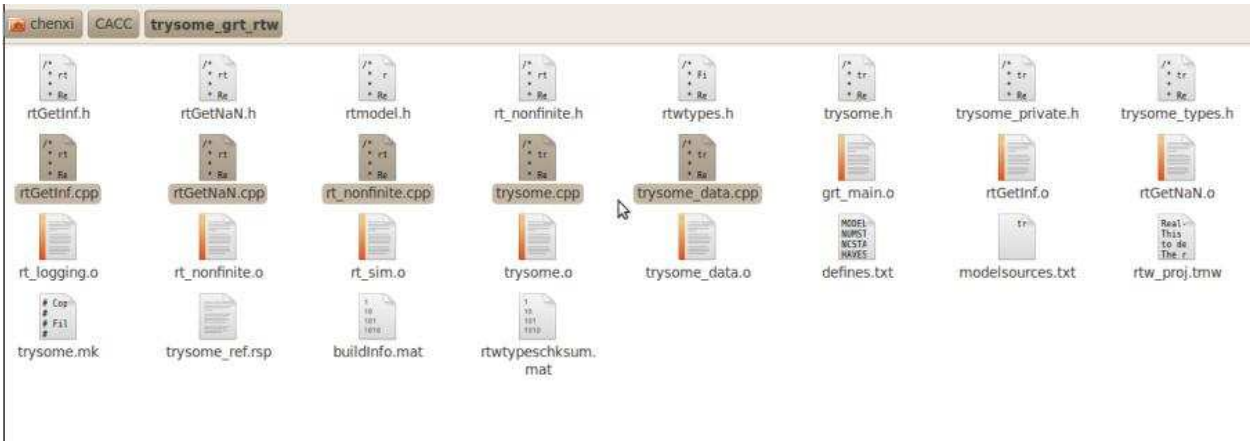


Figure 3: folder of generated code

As seen from Figure 3, the folder of the generated code comprises five sources files (.cpp) while the most important two of them are "trysome.cpp" (whose content is related to Simulink model's algorithm), "trysome_data.cpp" (comprising the values for parameters of the model inherited from the file "testpartofcacc_p.m".

Other files are the corresponding header files of the above source files, the corresponding objective files and the most important "make" file that is named " trysome.mk" (at lower left of Figure 3). The " trysome.mk" file can be used to compile the modified code and to generate an executable file in the parent folder named " trysome" (here is the folder "CACC").

1.4.2 Modifying procedures

Commonly, to change values for the parameters in the model one can just modify the corresponding values in "trysome.cpp". For the purpose of coupling the source file to our traffic simulator (SUMO), we have to modify the "trysome.cpp" in the following way:

1) Because these machine-generated code are hard to read by human-beings, in order to know which functions inside "trysome.cpp" would be called and their sequence to be called, we just simply add one sentence to every function defined in "trysome.cpp" so that once some function is called, it will print out the corresponding line. Figure 4 shows such an example.

```
extern "C" void rt_ODECreateIntegrationData(RTWSolverInfo *si)
{ cout<<"extern c void rt_ODECreateIntegrationData is called"<<endl;
  UNUSED_PARAMETER(si);
  return;
}                                        /* do nothing */
```

Figure 4: example for adding sentence

In Figure 4, once the function "rt_ODECreateIntegrationData" is called, the sentence "extern c void rt_ODECreateIntegrationData is called" will be displayed in the "terminal" window of Ubuntu. Then one can run the "trysome_mk" in the command line to compile the code and then also run the executable file named "trysome" available in the parent folder as stated in 1.4.1. After that, the exact functions called during one run of the model will be displayed in the "terminal" in time sequence.

2) by doing the first step, we know that four functions are necessarily to be called and their names are:

void trysome_initialize(bool firstTime);

extern "C" void MdlInitialize(void);

extern "C" void MdlOutputs(int tid,float data[],float *q1);

extern "C" void MdlUpdate(int tid,float data[],float *q1);

The names of the above four functions are exactly as those in the machine-generated code while the parameters and data types of above four functions have already been changed to make sure they can be called by the SUMO traffic simulator. This is because the original data types are only defined in Matlab.

The previous two functions are used to initialize the controller, and MdlOutputs is the function which performs the control algorithm. Every time the controller is called, the MdlUpdate is used to update information of the controller. The array "data[]" is used to assign values to the inputs of the controller and q1 is used to fetch the output of the controller (the acceleration). In function MdlOutputs the input parameters are named in the form of trysome_U.In1, trysome_U.In2,... and the outputs are named in the form of trysome_Y.Out1, trysome_Y.Out2.

The sequence numbers of inputs and outputs are exactly the same as those used in the Simulink model. In other words, trysome_U.In1, trysome_U.In2,...and trysome_U.In7 exactly correspond to In1, In2, …In7, associated with the modified Simulink model described in section 2 of Appendix B. Furthermore, trysome_Y.Out1 and trysome_Y.Out2 correspond to Out1 (reference acceleration after the "G_a" block" and Out2 (reference acceleration before the "G_a" block), see Section 2 of Appendix B. Therefore, we assigned parameters as shown in Figure 5 and fetched the reference acceleration by "G_a" as shown in Figure 6:

```
/* Model output function */
void trysome_output(int tid,float data2[], float *q1,float *q2)
{
  //cout<<"trysome_output is called"<<endl;
  trysome_U.In1=data2[0];//leading position
  trysome_U.In2=data2[1];//leading speed
  trysome_U.In3=data2[2];//host position
  trysome_U.In4=data2[3];//cruise speed
  trysome_U.In5=data2[4];//time headway
  trysome_U.In6=data2[5];//preceding vehicle's acceleration
  trysome_U.In7=data2[6];//host velocity
```

Figure 5: inputs assignment

```
*q1=trysome_Y.Out1;//reference acceleration after G_a
*q2=trysome_Y.Out2;//reference acceleration before G_a
```

Figure 6: outputs fetched

In Figure 5, one can see an array named "data2" that is used to assign values to the inputs of controller. Here fetching the value of the acceleration before "G_a" (the second output: trysome_Y.Out1) is done only for test purpose.

3) Subsequently, in the source file "trysome.cpp", parameters of the above four functions should be modified (to make these parameters and data type the same as those in the source file) in the function declarations in the corresponding header file "trysome.h".

Furthermore, also the data types in "trysome.h" are needed to be modified from those specific for Matlab to general data types. In particular, the data type "int_T" is changed into "int", the data type "bool_T" is changed to "bool" and the data type "real_T" is changed to "float" or "double".

4) After modifying the code, a shared library is necessary to be created to make use of this code. Here only the target files (all the files with the suffix ".o") in the folder shown in Figure 3 are needed. The shared library can be put anywhere if only it can be linked to. Below the commands we used to generate the shared library are shown:

```
>>make -f trysome.mk //compile the generated code
>>g++ -g -shared -Wl,-soname,libtrysome.so.0 \
  -o libtrysome.so.0.0 trysome.o grt_main.o rtGetInf.o rtGetNaN.o rt_logging.o rt_nonfinite.o
  rt_sim.o trysome_data.o –lc //using target file to create shared library
>>/sbin/ldconfig -n .
>>ln -sf libtrysome.so.0 libtrysome.so //link different names of the shared library
```

For more details about creating a shared library, you can refer to the following URL:
http://www.faqs.org/docs/Linux-HOWTO/Program-Library-HOWTO.html

5) Since we have 9 following vehicles and two kinds of controllers (CACC and ACC), we need to create 18 shared libraries for each possible combination.

Therefore, the above procedures have to be repeated for 18 times. In addition to this note that the names for the four functions (stated in the second procedure of this section) should be different in different shared libraries. This is needed in order to prevent confusion in SUMO when calling the same function belonging to different libraries. These names can be modified manually after the code is generated for each combination.

An easier way to do that is modify these names in /matlab/rtw/c/grt/grt_main.c before generating the code with Real-Time Workshop. As shown in Figure 7, we modified the function name "MdlOutputs" to "MdlOutputs3" and "MdlUpdate" to "MdlUpdates3" manually in /matlab/rtw/c/grt/grt_main.c so that in the generated code, the corresponding function names in source file "trysome3.cpp" (the name of the different Simulink model should also be different. Moreover, for a different Simulink model, e.g., "trysome3", the names of these functions would be "MdlUpdates3" and "MdlOutputs3". In this way, one can assign different names for same functions used in different libraries.

```
extern void MdlOutputs3(int_T tid);
extern void MdlUpdate3(int_T tid);
```

Figure 7: example of modifying function name

6) In order to use this shared libraries, we have to link SUMO to them, which is accomplished by specifying these libraries' address in: "sumo/src/Makefile".

Because we move all the generated shared library to the address "sumo/src/microsim", we specify these libraries' address in two different places of the file "sumo/src/Makefile" as shown in Figure 8:

```
sumo_DEPENDENCIES = ./netload/libnetload.a ./microsim/libmicrosim.a \
        ./microsim/cfmodels/libmicrosimcfmodels.a \
        ./microsim/devices/libmicrosimdevs.a \
        ./microsim/output/libmicrosimoutput.a \
        ./microsim/MSMoveReminder.o \
        ./microsim/trigger/libmicrosimtrigger.a \
        ./microsim/actions/libmsactions.a \
        ./microsim/traffic_lights/libmicrosimtls.a $(MESO_LIBS) \
        ./utils/geom/libgeom.a ./utils/shapes/libshapes.a \
        ./microsim/libtrysome.so  \
        ./microsim/libtrysome1.so  \
        ./microsim/libtrysome2.so  \
        ./microsim/libtrysome3.so  \
        ./microsim/libtrysome4.so  \
        ./microsim/libtrysome5.so  \
        ./microsim/libtrysome6.so  \
        ./microsim/libtrysome7.so  \
        ./microsim/libtrysome8.so  \
        ./microsim/libaccwithout.so \
        ./microsim/libaccwithout1.so \
        ./microsim/libaccwithout2.so \
        ./microsim/libaccwithout3.so \
        ./microsim/libaccwithout4.so \
        ./microsim/libaccwithout5.so \
        ./microsim/libaccwithout6.so \
        ./microsim/libaccwithout7.so \
        ./microsim/libaccwithout8.so \
```

```
am__DEPENDENCIES_3 = ./netload/libnetload.a ./microsim/libmicrosim.a \
        ./microsim/cfmodels/libmicrosimcfmodels.a \
        ./microsim/devices/libmicrosimdevs.a \
        ./microsim/output/libmicrosimoutput.a \
        ./microsim/MSMoveReminder.o \
        ./microsim/trigger/libmicrosimtrigger.a \
        ./microsim/actions/libmsactions.a \
        ./microsim/traffic_lights/libmicrosimtls.a $(MESO_LIBS) \
        ./utils/geom/libgeom.a ./utils/shapes/libshapes.a \
        ./microsim/libtrysome.so  \
        ./microsim/libtrysome1.so  \
        ./microsim/libtrysome2.so  \
        ./microsim/libtrysome3.so  \
        ./microsim/libtrysome4.so  \
        ./microsim/libtrysome5.so  \
        ./microsim/libtrysome6.so  \
        ./microsim/libtrysome7.so  \
        ./microsim/libtrysome8.so  \
        ./microsim/libaccwithout.so \
        ./microsim/libaccwithout1.so \
        ./microsim/libaccwithout2.so \
        ./microsim/libaccwithout3.so \
        ./microsim/libaccwithout4.so \
        ./microsim/libaccwithout5.so \
        ./microsim/libaccwithout6.so \
        ./microsim/libaccwithout7.so \
        ./microsim/libaccwithout8.so \
```

Figure 8: libraries address specification

In Figure 8, we can see the generated libraries with the name “libtrysome.so”, “libtrysome1.so”,…, “libtrysome8.so” (for CACC) and “libaccwithout.so”, “libaccwithout1.so”,.., “libaccwithout8.so” (for ACC) that are used as dependencies by the SUMO environment. By running the file “sumo/src/Makefile” (“make” command under this directory), SUMO will be linked to these libraries.

So far, shared libraries of the controller have been created and can be used by the SUMO environment.

Note: our way of just calling these four functions won't make the executable file “trysome” work (with segmentation fault) after modification, but the controller works well if it is just used in the form of a shared library

1.5 Using the shared libraries in SUMO

The SUMO installation can be found via the following URL:
http://sourceforge.net/apps/mediawiki/sumo/index.php?title=LinuxBuild

The version we used is the subversion of SUMO-0.12, please refer to “subversion checkout” at above URL.

In SUMO, the essential source file used to calculate the vehicle speed (by car-following models, see section 3.1.2 of the report) is “MSVehicle.cpp” under the directory: /sumo/src/microsim/. Figure 9 is a section we wrote in this file to call the shared library for a specific vehicle.

```
if (myfakelead==0)//it's the beginning timestep of the simulation
{trysome_initialize(1);
MdlInitialize3();//initialize CACC library for this vehicle
}

datap[0]=(*pred).getPositionOnLane();
datap[1]=(*pred).myState.mySpeed;
datap[2]=myState.myPos;
datap[3]=v_c;
datap[4]=h;
datap[5]=predacc;
datap[6]=myState.mySpeed;//assign values to a array to pass
if (myfakelead>0)//After the beginning timestep of the simulation
{MdlUpdate3(0,datap,&q1,&q2);}//update the input
MdlOutputs3(0,datap,&q1,&q2);//calculate the reference acceleration

if (q1>2)
{q1=2;}
else if (q1<-9)
{q1=-9;}//limit the acceleration with max value 2 and minimu value -9
myState.mySpeed=myState.mySpeed+q1*0.01;//calculate the speed by reference acceleration
myState.myPos=myState.myPos+myState.mySpeed*0.01;//calculate the position
```

Figure 9: calling shared library in SUMO

As seen in Figure 9, at the beginning of the simulation, we initialize the shared CACC library and then we use an array “datap” to store the values for the input parameters of the controller.

These parameters stored in sequence are: preceding vehicle's position, speed, host vehicle's position, cruise speed, time headway, preceding vehicle's acceleration and host vehicle's speed.

Then this array would be used by the function "MdlOutputs3" to calculate the reference acceleration. However, the inputs have to be updated by calling "MdlUpdate3" before "MdlOutputs3" is being called at each time step after the first timestep. Then the reference acceleration fetched by "q1" (revised by "G_a") should be limited by the maximum value 2m/s^2 and minimum value of -9m/s^2. Finally this acceleration can be used to calculate the speed and position.

In this way, we successfully couple the controller model and SUMO. For other details, please refer to the source code.

2. Bidirectional coupling between MiXiM and SUMO part:

In this part, procedures to bidirectional couple the traffic simulator SUMO and the network simulator OMNeT++/MiXiM are explained. A similar bidirectional coupling example can be found at the following URL:
http://veins.car2x.org/tutorial/, where Christoph Sommer gives an example of coupling SUMO and OMNeT++/INeT.

2.1 Software Specification

In MiXiM, which is modified by Christoph Sommer, an example named "traci_launchd" is given. In this example, vehicles in SUMO are using the OMNeT++/MiXiM to communicate by exchanging messages and move from a starting point to a destination point. This make it possible for us to implement our SUMO model (see Section 3.2.2 in the main report) and the network model (provided by UT/DACS) by modifying such example. The exact software environments that we used:

- OMNeT++ 4.1
- DACS-MiXiM.1.2 (supplied by UT/DACS) packet
- MiXiM-sommer (git://github.com/sommer/mixim-sommer.git) packet
- SUMO-0.12 subversion (stated in section 1.5)

For installation of OMNeT++4.1, please refer to:
www.omnetpp.org/doc/omnetpp41/InstallGuide.pdf
For installation of MiXiM packets, please refer to:
http://veins.car2x.org/tutorial/
For installation of SUMO, please refer to:
http://sourceforge.net/apps/mediawiki/sumo/index.php?title=LinuxBuild

The following sections describe how to implement the traffic and network models used in this assignment.

2.1 Building the traffic model

As stated in Section 3.2.2 of the report, our traffic model is quite simple, which is different from Christoph Sommer's model in "traci_launched" (after installation, its directory is "omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd"). So we have to replace the corresponding "net.net.xml" and"routes.rou.xml" files in the folder of "omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd". Here the "net.net.xml" file specifies the road network, and the "routes.rou.xml" file specifies the route of vehicle.

The "net.net.xml" file is created by two xml files: "hello.nod.xml" and "hello.edg.xml" which can be seen in Figure 10 and Figure 11, respectively.

```
<nodes>
    <node id="1" x="0.0" y="0.0" />
    <node id="2" x="+5000.0" y="0.0" />
    <node id="3" x="+5001.0" y="0.0" />

</nodes>
```

Figure 10: hello.nod.xml

```
<edges>
    <edge fromnode="1" id="r1" tonode="2" />
    <edge fromnode="2" id="r2" tonode="3" />

</edges>
```

Figure 11: hello.edg.xml

Since in SUMO, a road network with only one section is not allowed, we defined in "hello.nod.xml" three nodes in a horizontal line as seen in Figure 10 with the coordinates (0.0, 0.0), (5000.0, 0.0) and (5001.0, 0.0). In Figure 11, we show that we used two edges to connect these three nodes. In order to build such a "net.net.xml" file, the following command should be executed under the directory "sumo/bin/". Moreover, both of the "hello.nod.xml" file and "hello.edg.xml" file should be put under this directory, because "netconvert" tool is installed here by default.

>>./netconvert --xml-node-files=hello.nod.xml --xml-edge-files=hello.edg.xml --output-
>>file=net.net.xml

Furthermore, the "net.net.xml" file (see source code) is also generated here and should be moved to "omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd" and replace the "net.net.xml" created by Christoph Sommer.

Furthermore, the "routes.rou.xml" used for our experiment can be seen in Figure 12, which is also moved to "omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd" folder to replace the "routes..rou.xml" created by Christoph Sommer.

```
<routes>
    <vtype accel="2.5" decel="10" id="vl" length="4.46" maxspeed="100.0" sigma="0.0" />
    <vtype id="vf" length="4.46" maxspeed="50.0" sigma="0.0">
      <carFollowing-IDM accel="2" decel="9" sigma="0.0"/>
    </vtype>
<route id="route0" edges="r1 r2"/>
    <vehicle depart="1" id="veh0" route="route0" type="vl" departpos="235.44" departspeed="20" />
    <vehicle depart="1" id="veh1" route="route0" type="vf" departpos="209.28" departspeed="19" />
    <vehicle depart="1" id="veh2" route="route0" type="vf" departpos="183.12" departspeed="19" />
    <vehicle depart="1" id="veh3" route="route0" type="vf" departpos="156.96" departspeed="19" />
    <vehicle depart="1" id="veh4" route="route0" type="vf" departpos="130.80" departspeed="19" />
    <vehicle depart="1" id="veh5" route="route0" type="vf" departpos="104.64" departspeed="19" />
    <vehicle depart="1" id="veh6" route="route0" type="vf" departpos="78.48" departspeed="19" />
    <vehicle depart="1" id="veh7" route="route0" type="vf" departpos="52.32" departspeed="19" />
    <vehicle depart="1" id="veh8" route="route0" type="vf" departpos="26.16" departspeed="19" />
    <vehicle depart="1" id="veh9" route="route0" type="vf" departpos="0.00" departspeed="19" />
</routes>
```

Figure 12: routes.rou.xml

As Figure 12 shows, we define vehicle's length, maximum speed sigma (human' influence index) and even car-following model. However, since the reference acceleration calculation is calculated in our model using the controller library we generated in Section 1, the following SUMO parameters: "accel", "decel", "maxspeed", "sigma" and "carFollowing-IDM" will not be used in our SUMO model. Nevertheless, the SUMO parameters used for each vehicle on departure position ("departpos") and departure speed ("departspeed") are still applied in our SUMO model.

So far, we are able to provide the traffic model for our experiment.

Note: speed of the 9 following vehicles in Figure 12 is in charge of the shared libraries generated in Section 1, while the behaviour of the leading vehicle is also specified also in /sumo/src/microsim/MSVehicle.cpp, which can be seen in Figure 13 and Figure 14:

```
if (mytakelead>=8000&&mytakelead<8250)
{
myState.mySpeed=myState.mySpeed+0.02;
myState.myPos=myState.myPos+0.01*myState.mySpeed;
```

Figure 13: accelerating scenario

```
if (myfakelead>=8000&&myfakelead<8055)
{myState.mySpeed=myState.mySpeed-0.09;
myState.myPos=myState.myPos+0.01*myState.mySpeed;
myTarget=0;
myacclcx=-9;
myfakelead++;
}

else if (myfakelead==8055)
{
myState.mySpeed=myState.mySpeed-0.05;
myState.myPos=myState.myPos+0.01*myState.mySpeed;
myTarget=0;
myacclcx=-5;
myfakelead++;
}
```

Figure 14: decelerating scenario

Figure 13 and Figure 14 describe the exact leading vehicle's behaviour in accelerating scenario and decelerating scenario given in Section 4.2 of the main report.

In Figure 13, the leading vehicle accelerates with the acceleration 2m/s^2 from timestep=8000 to timestep=8250 and its speed accelerates from 20m/s to 25m/s while in Figure 14, the leading vehicle decelerate with the acceleration -9m/s^2 from timestep=8000 to timestep=8054 and with the acceleration -5m/s^2 at timestep=8055, so its speed decelerates from 20m/s to 15m/s. During other time, the acceleration of the leading vehicle is set to zero.

So far, the traffic model has been built completely.

2.2 Implementing the Network model

To install the network model supplied by UT/DACS, we modified the "car.ned" file and substituted the application layer, network layer, Mac layer, and physical layer modules supplied by UT/DACS. The "car.ned" file can be found under the folder: "omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd". The modules contained in the "car.ned" file can be seen in Figure 15:

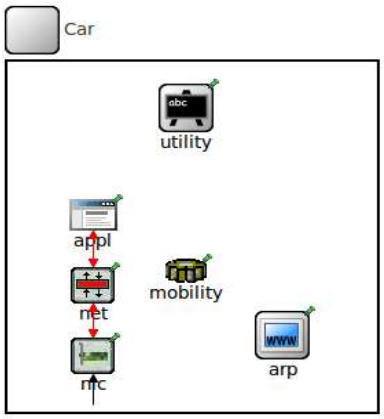


Figure 15: car.ned

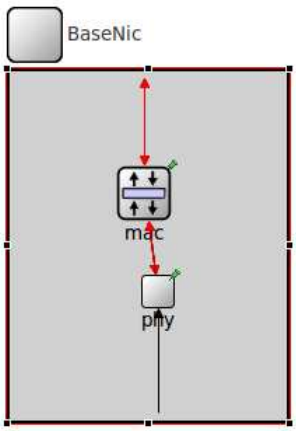


Figure 16: nic

What we changed are the “appl”, “net”, “nic” (comprising “mac” and “phy” modules, see Figure 16) modules in Figure 15. All the other modules “car.ned” file are left unchanged. This part comprised two procedures:

First, we put the necessary files to specific directories.

For the physical layer, we had to move the following files from the directory “omnetpp-4.1/samples/DACS_MiXiM-1.2/modules/phy/” to the directory “omnetpp-4.1/samples/mixim-sommer/modules/phy/”.

“Decider80211p.cc”
“Decider80211p.h”
“PhyLayerp.cc”
“PhyLayerp.h”
“ PhyLayerp.ned”

For Mac layer, we have to move the following files from the directory “omnetpp-4.1/samples/DACS_MiXiM-1.2/modules/mac/” to the directory “omnetpp-4.1/samples/mixim-sommer/modules/mac/”.

“Mac80211p.cc”
“Mac80211p.h”
“Mac80211p.ned’

Since the modules corresponding to physical layer and Mac layer (“mac” and “phy” in Error! Reference source not found.) are comprised in “nic” module as seen in Error! Reference source not found., we also have to move the file “Nic80211p.ned” from the directory “omnetpp-4.1/samples/DACS_MiXiM-1.2/modules/nic/” to the directory “omnetpp-4.1/samples/mixim-sommer/modules/nic/”.

For the network layer, we have to move the following files from the directory “omnetpp-4.1/samples/DACS_MiXiM-1.2/modules/netw/” to the directory “omnetpp-4.1/samples/mixim-sommer/modules/netw/”.

“BeaconNetwLayer.cc”
“BeaconNetwLayer.h”
“BeaconNetwLayer.ned”
(Note that the following six files are not use in our experiment but might be used in future):
“JitterBeaconNetwLayer.cc”
“JitterBeaconNetwLayer.h”
“JitterBeaconNetwLayer.ned”

“ReactiveBeaconNetwlayer.cc”
“ReactiveBeaconNetwlayer.h”
“ReactiveBeaconNetwlayer.ned”

For the application layer, we have to move the following files from the directory “omnetpp-4.1/samples/DACS_MiXiM-1.2/modules/application/” to the directory “/omnetpp-4.1/samples/mixim-sommer/modules/application/”.

“BeaconApplLayer.cc”
“BeaconApplLayer.h”
“BeaconApplLayer.ned”

Second, after moving the files as stated in first procedure, we have to specify the directories of the “BeaconNetwLayer.ned”, “BeaconApplLayer.ned”, “Nic80211p.ned” as shown in Figure 17 to the beginning of the “car.ned” source file.

```
package org.mixim.examples.traci_launchd;

import org.mixim.modules.mobility.traci.TraCIMobility;
import org.mixim.base.modules.*;
import org.mixim.modules.application.BeaconApplLayer;
import org.mixim.modules.netw.BeaconNetwLayer;
import org.mixim.modules.nic.Nic80211p;
```

Figure 17: beginning part of "car.ned" source file

The first line in Figure 17 gives the directory of “traci_launchd”; the second line is empty; the third line gives the directory of “mobility” module in Figure 15; the forth line gives the directories of “utility” module and “arp” module in Figure 15; the other lines (in shadow) give the directories of modules supplied by DACS (the directories of “BeaconNetwLayer.ned”, “BeaconApplLayer.ned”, “Nic80211p.ned”—corresponding to the “net”, “appl”, “nic” modules in Figure 15).

In addition to the above, we have to specify the directories of the “PhyLayerp.ned” and “Mac80211p.ned” as shown in the beginning of the “Nic80211p.ned” source file.

```
package org.mixim.examples.traci_launchd;

import org.mixim.modules.phy.PhyLayerp;
import org.mixim.modules.mac.Mac80211p;
//
```

Figure 18: beginning part of "nic.ned" source file

Again the first line in Figure 18 gives the directory of “traci_launchd”; the second line is empty; the other lines (in shadow) give the directories of modules supplied by DACS (the directories of

the “PhyLayerp.ned” and “Mac80211p.ned”—corresponding to the “phy”, “mac” modules in Figure 16).

Using the other files contained in the following directory we successfully could built our network model:

omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd

2.3 Bidirectional coupling of the traffic model and network model

Since our whole model is based on the existent “traci_launchd” model created by Christopher Sommer, we can already run the simulation after building the traffic model and network model as stated in section 2.1 and 2.2 of this Appendix. However, so far, the CACC controller cannot work, because we have not implemented what parameters to be communicated using the OMNeT++/MiXiM model. The only parameter to be transmitted/received using the wireless channel for CACC in our experiment is the preceding vehicle’s acceleration. So here we just specify vehicle’s acceleration as the only parameters transmitted by the OMNeT++/MiXiM model.

2.3.1 Parameter definition

In order to transmit vehicle’s acceleration between OMNeT++/MiXiM and SUMO, we have to define these parameters in two files with the same name “TraCIConstants.h” and same content. Their directories are “omnetpp-4.1/samples/mixim-sommer/modules/mobility/traci/” and “sumo/src/traci-server”. The same lines showed in Figure 19 should be added to the two files.

```
// acceleration of preceding vehicle
#define VAR_PREDACC 0x94
```

Figure 19: parameter definition

Here, the parameter “VAR_PREDACC”, shown in Figure 19 should be hex number, which is not used by other parameters defined in those two files. Note that VAR_PREDACC represents the acceleration of preceding vehicle. This name is just used in exchanging parameters between OMNeT++/MiXiM and SUMO environments. In the network model source file and the traffic model source file, a different name may be used.

2.3.2 OMNeT++/MiXiM modification

In the OMNeT++/MiXiM, we have to modify three main files. These are:

omnetpp-4.1/samples/mixim-sommer/modules/mobility/traci/TraCIMobility.h
omnetpp-4.1/samples/mixim-sommer/modules/mobility/traci/TraCIScenarioManager.cc
omnetpp-4.1/samples/mixim-sommer/modules/application/BeaconApplLayer.cc

1) In “TraCIScenarioManager.cc”, we defined a function named “commandSetPredacc” shown in Figure 20:

```
//command set predacc
void TraCIScenarioManager::commandSetPredacc(std::string objectId, double acclcx) {
    double wocao=acclcx;
    uint8_t variableId =VAR_PREDACC;
    uint8_t variableType =TYPE_DOUBLE;

    if (objectId[3]>=48&&objectId[3]<=57)
    {queryTraCI(CMD_SET_VEHICLE_VARIABLE, TraCIBuffer() << variableId << objectId << variableType << wocao);
    }
}
```

Figure 20: "commandSetPredacc" definition

In Figure 20, we can see that this function uses the “objectId” (vehicle’s id: from “veh0”, “veh1”…to “veh9”) and “acclcx” (preceding vehicle’s acceleration) as parameters. Moreover, we use the “variableId” (VAR_PREDACC as defined in section 2.3.1) and “objectId” as the parameters of “queryTraCI” to pass the command “CMD_SET_VEHICLE_VARIABLE” to SUMO through TraCI.

2) In order to call this function by the application layer of the network model, we can define another function in “TraCIMobility.h” to call “commandSetPredacc” as shown in Figure 21:

```
//command to set preceding vehicle's acceleration
void commandSetPredacc(std::string objectId,float acclcx1) {
    getManager()->commandSetPredacc(objectId, acclcx1);

}
```

Figure 21: "commandSetPredacc" definition in "TraCIMobility.h"

In Figure 21, only the name of parameters are different from those shown in Figure 20, and we use “getManager()” to call the “commandSetPredacc” defined in “TraCIScenarioManager.cc”.

3) In the “BeaconApplLayer.cc”, we modify the function of “handleLowerMsg”. The modified part can be seen in Figure 22:

```
float lcxp=uniform(0,1);//define a random value between 0 and 1
if (lcxp<(1-fakepl))//fake packet loss
{
if(myApplAddr()==m->getSrcAddr()+1)//message received from a vehicle has id larger than that of host vehicle by1
{EV<<"vehicle "<<m->getSrcAddr()<<" is vehicle "<< myApplAddr()<<"'s predecessor!"<<endl;
//bcounter++;//for test purpose
//EV<<"the bcounter is "<<bcounter<<endl;
std::string nodeId="vehx";
nodeId[3]=myApplAddr()+48;//specify the host vehicle's id
myMobility->commandSetPredacc(nodeId,m->getAcceleration());//pass preceding vehicle's acceleration
```

Figure 22: modified part of "handleLowerMsg" in "BeaconApplLayer.cc"

As shown in Figure 22, we first generate a random value between 0 and 1 so that when the packet loss is specified to “fakepl”, the application layer has a probability of (1-“fakepl”) to deal with the received message. It would first check whether this message comes from a preceding vehicle. If it is, then the message’s source address should be

larger than the application layer's address by 1 or we can say that the id of the vehicle sending the message should be larger than the host vehicle's id by 1. If this message comes from a preceding vehicle, this application layer translates its address to corresponding vehicle's (e.g. translate "1" to "veh1"). The last line calls the function defined in "TraCIMobility.h" –"commandSetPredacc" so that the acceleration of preceding vehicle stored in the message ("m->getAcceleration( )") can be passed to SUMO.

2.3.2 SUMO modification

In SUMO, two main files have to be modified. These are:

sumo/src/traci-server/TraCIServerAPI_Vehicle.cpp
sumo/src/microsim/MSVehicle.h

1) In "TraCIServerAPI_Vehicle.cpp", first we have to add "VAR_PREDACC" (see Section 2.3.1 of this Appendix) to the list of parameters that can be accepted by SUMO, as shown in Figure 23:

```
&&variable!=VAR_ACCEL&&\
&&variable!=VAR_TAU
&&variable!=VAR_SPEED&&\
&&variable!=VAR_PREDACC
```

Figure 23: adding "VAR_PREDACC" to those parameters accepted by SUMO

As shown (in the shadowed part) in Figure 23, "VAR_PREDACC" is put in the list of parameters which SUMO can deal with. Once SUMO received a message to set value for the preceding vehicle's acceleration (sent by the function "queryTraCI" in Figure 20), it would execute the lines shown in Figure 24:

```
case VAR_PREDACC:
    if (valueDataType!=TYPE_DOUBLE) {
        server.writeStatusCmd(CMD_SET_VEHICLE_VARIABLE, RTYPE_ERR, "Setting predacceleration requires a double.", outputStorage);
        return false;
    }
    v->setPredacc(inputStorage.readDouble());

    break;
```

Figure 24: setting "VAR_PREDACC" value in "TraCIServerAPI_Vehicle.cpp"

In Figure 24, we can see that if the type of the value for "VAR_PREDACC" is "TYPE_DOUBLE", a function named "setPredacc" will be called to set this value to the vehicle. This function is defined in "MSVehicle.h" as shown in Figure 25:

```
void setPredacc(double predacc1) {
predacc=predacc1;
}
```

Figure 25: "setPredacc" definition

As seen in Figure 25, in this function we just pass the value of input parameter ("inputStorage.readDouble(), see Figure 24) to the parameter "predacc".

In this way, the value of preceding vehicle's acceleration is passed to the parameter defined in MSVehicle ("predacc") and it contributes to the calculation of reference acceleration as shown in Figure 9.

2.3.3 Iterating vehicles

Since in our experiment, each vehicle uses an independent controller library, which is not available in the existing car-following models in SUMO, we have to manually "tell" each vehicle to use its own controller library. SUMO uses an "iterator" in "sumo/src/microsim/MSLane.cpp" to perform vehicle's mobility starting from the last following vehicle towards the leading vehicle. Therefore, we use the counter named "curr" defined in "sumo/src/microsim/MSLane.cpp" as shown in Figure 26:

```
bool
MSLane::setCritical(SUMOTime t, std::vector<MSLane*> &into) {

    int first2pop = -1;
    int curr = 0;
    bool hadProblem = false;
    VehCont::iterator i;
    std::vector<MSVehicle*> vaporized;
    for (i=myVehicles.begin(); i!=myVehicles.end(); ++i, ++curr) {
    VehCont::const_iterator pred(i + 1);
    cout<<curr<<"\n";

    bool removed;

    removed = (*i)->moveFirstChecked(*pred,curr);


        if (removed) {
            vaporized.push_back(*i);
        }
        MSLane *target = (*i)->getTargetLane();
        if ((removed||target!=0)&&first2pop<0) {
            first2pop = curr;
        }
    }
```

Figure 26: counter "curr" in "MSLane.cpp"

In Figure 26, it is shown what we modified. We use the function "moveFirstChecked(*pred,curr)" to pass the value of "curr" and the pointer *pred) to "sumo/src/microsim/MSVehicle.cpp". This function is defined in "sumo/src/microsim/MSVehicle.cpp".

Note that "*pred" is a pointer defined in SUMO which can be used to refer to preceding vehicle, while "curr" is used in the following way. It uses an initial value of 0, and increases every time by one when the vehicle is iterated in SUMO. In other words, SUMO begins to move a different vehicle in the direction from the last following vehicle to the leading vehicle. After iterating all the vehicles on the road, "curr" will be set to the value to 0 again. Since the original source code is quite large, we will give the simplified pseudo code of how the counter "curr" is used in "sumo/src/microsim/MSVehicle.cpp":

>>switch (curr) {
>>case 0
>>using the library for veh9 (last vehicle);
>>break;

>>case 1
>>using the library for veh8;
>>break;
…
>>case 8
>>using the library for veh1;
>>break;

>>case 9
>>moving leading vehicle as stated in Figure 13/Figure 14;
>>break;

>>default:
>>break;
>>}

The details about definition of the function "moveFirstChecked(*pred,curr)" can be found in the source code of "sumo/src/microsim/MSVehicle.cpp".

In addition to the above, the "omnetpp-4.1/samples/DACS_MiXiM-1.2/examples/Mac80211Beaconing/omnetpp.ini" should be used to replace the "omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd/ omnetpp.ini". Because this omnetpp.ini file is easy to read, but is too large, to get the details of modification of this file, you can refer to its source code. Note that during the implementation procedure, the "lambda_g" defined in "/omnetpp-4.1/samples/mixim-sommer/modules/netw/ BeaconNetwLayer.h" is not used. Instead we defined another variable named "ritama" denoting the beacon sending frequency in the same file. Moreover, in /omnetpp-4.1/samples/mixim-sommer/ modules/netw/BeaconNetwLayer.cc" we use the following statement to make the network model to send specific number(the value of "ritama") of beacons every seconds:
"scheduleAt(simTime() + (1.0/(ritama/100)), tauTimer);"

So far, we have already described how to bidirectional couple OMNeT++ and SUMO by modifying the code. To get more details, one can refer to source code of files listed below. These files were modified or created by us. Some files are not mentioned in the descriptions given above because they were modified just for grammar reasons, e.g. function declaration in head files for the function defined in the source file:

/sumo/bin/hello.nod.xml
/sumo/bin/hello.edg.xml
/omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd/net.net.xml
/omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd/car.ned
/omnetpp-4.1/samples/mixim-sommer/modules/nic/Nic80211p.ned
/omnetpp-4.1/samples/mixim-sommer/modules/netw/BeaconNetwLayer.cc
/omnetpp-4.1/samples/mixim-sommer/modules/netw/BeaconNetwLayer.h
/omnetpp-4.1/samples/mixim-sommer/modules/application/BeaconApplLayer.cc

/omnetpp-4.1/samples/mixim-sommer/modules/application/BeaconApplLayer.h
/omnetpp-4.1/samples/mixim-sommer/modules/mobility/traci/TraCIMobility.h
/omnetpp-4.1/samples/mixim-sommer/modules/mobility/traci/TraCIMobility.cc
/omnetpp-4.1/samples/mixim-sommer/modules/mobility/traci/TraCIScenarioManager.h
/omnetpp-4.1/samples/mixim-sommer/modules/mobility/traci/TraCIScenarioManager.cc
/omnetpp-4.1/samples/mixim-sommer/modules/mobility/traci/TraCIConstants.h"
/sumo/src/traci-server/TraCIConstants.h
/sumo/src/traci-server/TraCIServerAPI_Vehicle.cpp
/sumo/src/traci-server/TraCIServerAPI_Vehicle.cpp
/sumo/src/microsim/MSVehicle.h
/sumo/src/microsim/MSVehicle.cpp
/sumo/src/microsim/MSLane.h
/sumo/src/microsim/MSLane.cpp

Finally, the whole model is built.

In order to make the whole model to work, a script (~/omnetpp-4.1/samples/mixim-sommer/base/
sumo-launchd.py) should be run first, and where the shared libraries' directory should be mentioned. The command used for running SUMO in command line is:

LD_LIBRARY_PATH="/home/…/sumo/src/microsim" ~/omnetpp-4.1/samples/mixim-sommer/base/sumo-launchd.py -vv -c ~/sumo/bin/sumo

One can run the simulation, either by running "omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd/ omnetpp.ini" graphical way, or a command line. For example, if one wants to run the configuration "config1" specified in "omnetpp.ini", this command can be used to run the simulation in command line under the directory "omnetpp-4.1/samples/mixim-sommer/examples/traci_launchd/":

>>./run -c config1_c -u Cmdenv

For other details about SUMO, please refer to:
http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page

For other details about OMNeT++, please refer to:
http://www.omnetpp.org/

Please note that:

- the source code of the modified SUMO model can be supplied together with this Appendix;
- the source code of the modified MiXiM model can be supplied together with this Appendix.