VENTOS User Manual

Version 0.8

Revision 7

To get the latest version of this document, please visit here.

Version History

Version No.	Date	Author	Changes
0.1	September 2016	Mani	Initial version
0.2	September 2016	Mani	Adding examples to Part3
0.3	November 2016	Mani	~~Adding SNMP appendix~~
0.4	January 2017	Mani	Adding nodes into the simulation
0.5	April 2017	Mani	Completely rewriting the V2X part
0.6	May 2017	Mani	How to debug programs
0.7	June 2017	Mani	Adding platoon and speed change
0.8	December 2017	Mani	Adding many V2X examples

Table of Contents

About This Manual

VENTOS is an open-source integrated VANET C++ simulator for studying vehicular traffic flows, collaborative driving, and interactions between vehicles and infrastructure through DSRC-enabled wireless communication capability. VENTOS is useful for researchers in transportation engineering, control theory and vehicular networking fields. VENTOS is being developed in Rubinet Lab, UC Davis since 2013 and is the main tool in the C3PO project.

The aim of this manual is to help you get started with VENTOS. This manual has all the necessary step-by-step information to run a basic simulation and is structured into the following parts:

Part 1: We introduce VENTOS and all of its small building blocks.

Part 2: VENTOS uses SUMO to perform road traffic simulation. In order to create your own simulations on custom road maps, you will need to learn how SUMO works. This part shows you how to get started with SUMO and how to generate your own 'SUMO network file' and 'SUMO traffic demand file'.

In order to get more information on SUMO, you can refer to the corresponding online user documentation in here. You can use the SUMO mailing list at sumo-user@lists.sourceforge.net to ask questions, but first search the sumo-user mailing list in here to see if someone has not already asked your question. Answers to some common questions may also be found in the FAQ.

Part 3: In this part, we will show you how to make your own V2X simulation scenarios using VENTOS. Besides this official documentation and the examples, you can post your VENTOS-related questions on stackoverflow website with the VENTOS tag. You can also open an issue here.

VENTOS makes use of OMNET++ libraries/tools extensively. OMNeT++ is an open-source, extensible, modular, component-based C++ simulation package, primarily for building network simulators. "Network" is meant in a broader sense that includes wired and wireless communication networks, on-chip networks and queueing networks. In order to get more information on OMNET++, you can refer to the online user manuals in here where you can find excellent tutorials, videos and comprehensive user manual. To get help, you can check the OMNET++ Google group in here or you can post your programming-related questions on stackoverflow website with the omnet++ tag.

Part 1: Introduction to VENTOS

1 VENTOS (**VE**hicular **N**e**T**work **O**pen **S**imulator)

VENTOS is an open-source VANET C++ simulator for studying vehicular traffic flows, collaborative driving, and interactions between vehicles and infrastructure through DSRC-enabled wireless communication capability. VENTOS is being developed in Rubinet Lab, UC Davis since 2013 and is the main tool in C3PO project. You can find more information here.

VENTOS supports multi-modal traffic simulation consists of motor vehicles (cars, buses, trucks, motor-cycles, etc.), bikes and pedestrians. You can design and verify many DSRC applications and monitor the message exchange between On-Board Units (OBUs) installed on motor vehicles or bikes, Road-Side Units (RSUs) and pedestrians. VENTOS supports all types of V2X wireless communications such as V2V (Vehicle-to-Vehicle), V2I (Vehicle-to-Infrastructure) and V2P (Vehicle-to-Pedestrians). You can design variety of DSRC safety/efficiency/infotainment applications such as:

- Emergency Electronic Brake Light (EEBL)
- CACC vehicle platooning
- Multi-hop routing protocols in VANET
- Traffic Signal Control (TSC) algorithms that rely on live traffic information from approaching cars

VENTOS is released as an open-source project to support the V2X research community in order to design and evaluate their own algorithms. You can use VENTOS as a toolbox to build a more complex simulation framework by extending its code or adding your own developed algorithm into it as described in Part 3, Section 8. Using a common tool like VENTOS makes the implemented algorithms more comparable.

1.1 VENTOS Architecture

VENTOS is an integrated simulator based on two well-known simulators: **SUMO** and **OMNET++.**

- SUMO (Simulation of Urban Mobility) is an open-source, microscopic, continuous-space, discrete-time C++ road traffic simulator, developed by Institute of Transportation Systems at the German Aerospace Center and adopted as our vehicular traffic simulator. VENTOS is closely coupled with SUMO through TraCI (Traffic Control Interface) and uses the mobility information of cars, bikes and pedestrians to perform realistic simulation. We believe that SUMO is as powerful as a Sumo wrestler.

- OMNET++ is an open-source, extensible, modular, component-based C++ simulation package and captures the wireless communication simulation. IEEE 802.11p physical layer modeling and IEEE 1609.4 are implemented in the Veins (Vehicles in Network Simulation) framework and is used for wireless V2X communication between different modules. Many well-known TCP/IP protocols can be added from INET framework. Figure 1 shows the VENTOS architecture.

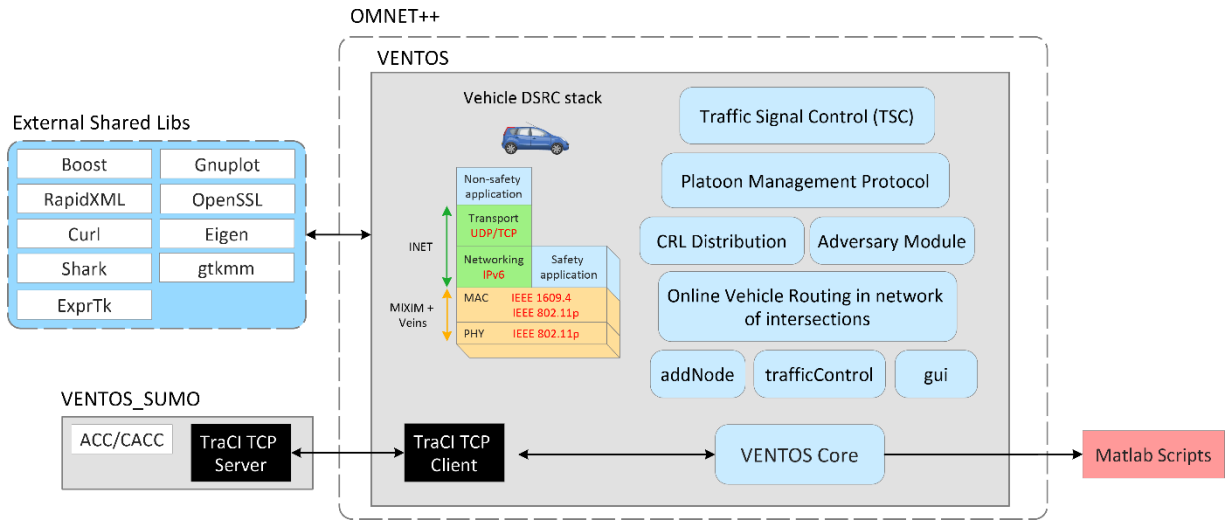


Figure 1: **VENTOS architecture**

The functionality of the VENTOS modules depicted in Figure 1 is described in the following:

- **VENTOS_SUMO:** Implements additional car-following models for ACC/CACC vehicles in SUMO and extends the TraCI commands which connects SUMO to an external application.
- **Traffic Signal Control (TSC):** This module implements many of the Traffic Signal Control (TSC) algorithms that is being used nowadays in intersections such as fix-time and traffic-actuated or algorithms developed in research community such as adaptive Webster, Longest Queue First (LQF), Oldest Job First (OJF), Maximum weight Matching (MWM).

- **Platoon Management Protocol:** This module implements a platoon management protocol that supports different maneuvers such as merge, split, entry, follower leave, platoon leader leave. You can find more details here. You can also check a sample platooning video using VENTOS in here.
- **Certificate Revocation List (CRL) Distribution:** This module implements many of the well-known CRL distribution algorithms in Vehicular PKI architecture such as RSU-only, C2C Epidemic, Most-Pieces Broadcast (MPB), Intelligent CRL Exchange (ICE), etc. You can find more details here.
- **Adversary Module:** You can use this module to study security attacks specially in collaborative driving. You can find more details here.
- **Online Vehicle Routing:** This module implements dynamic traffic routing algorithms using real-time traffic information to reduce the average delay as well as fluctuation of the average speed within the whole network. You can find more details here.
- **addNode:** This module allows you to add different nodes such as RSUs, obstacles, vehicle flow, etc. to the simulation environment.
- **trafficControl:** This module allows you to change the speed of a vehicle or coordinate a platoon
- **gui:** This module allows you to change viewport and track a specific vehicle in the simulation
- **VENTOS Core:** VENTOS core is responsible for orchestrating all internal modules. It is also responsible for initiating the TraCI interface with SUMO as well as logging simulation results into text files. These output files can be fed into Matlab scripts for post-processing.

VENTOS uses the following libraries:

- **Boost:** Boost provides free peer-reviewed portable C++ source libraries. We are using boost filesystem, circular buffer, tokenizer, and graph.
- **RapidXML:** RapidXml is an attempt to create the fastest XML parser possible, while retaining usability, portability and reasonable W3C compatibility. It is an in-situ parser written in modern C++, with parsing speed approaching that of strlen function executed on the same data. RapidXML is a header only library, and is already included in VENTOS source code.
- **Curl:** Curl is a library for transferring data with URL syntax, supporting many different networking protocols. We use this library to download the latest SUMO binaries from Internet.
- **Eigen:** Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. We use this library for matrix calculations.
- **Shark:** Shark is a fast, modular, feature-rich open-source C++ machine learning library. It provides methods for linear and nonlinear optimization, kernel-based learning algorithms, neural networks, and various other machine learning techniques.
- **Gnuplot:** Gnuplot is a portable command-line driven interactive plotting program.
- **OpenSSL:** OpenSSL is an open-source implementation of the SSL and TLS protocols. The core library, written in the C programming language, implements the basic cryptographic functions and provides various utility functions. We use OpenSSL library to sign and verify different messages.
- **gtkmm:** gtkmm is the official C++ interface for the popular GUI library GTK+. We use this library to create the output message log window.
- **ExprTk:** The C++ Mathematical Expression Toolkit Library (ExprTk) is a simple to use, easy to integrate and extremely efficient run-time mathematical expression parser and evaluation engine. It supports numerous forms of functional, logical and vector processing semantics and is very easily extendible. We use this library to evaluate speed functions.

2 Installing VENTOS

We have tested VENTOS on the following operating systems. We recommend installing VENTOS on the latest version of Ubuntu in a dual-boot setup where you will have the most efficient working environment for VENTOS and VENTOS_SUMO.

Ubuntu	Mac OS X
16 x64	Sierra (10.12)
	El Capitan (10.11)
	Yosemite (10.10)

You can also use a Virtual Machine (VM) such as Oracle VirtualBox (video). If you do not use VENTOS often, then the VM would be a good option. This avoids reboots and enables easy data-exchange between the guest and your host OS. Moreover, you can save the VM state using snapshots and roll back to the previous functioning state if something goes wrong. One big disadvantage of using a VM is slow speed because your available computer resources (CPU, RAM, etc.) are shared between two OSes. Hence, if you use VENTOS often, then a dual-boot setup is more suitable. You can install OMNET++ on Windows too, but building VENTOS and VENTOS_SUMO on Windows is very painful and requires several tweaks. Hence, we recommend you to switch to a Unix-like OS such as Ubuntu. Follow these instructions to install VENTOS (and its prerequisites) on your Unix-like OS[1].

Step 1: If you do not have Git installed on your machine, then install it.

[Ubuntu] Run the following command in terminal:

```
sudo apt-get install git
```

[Mac OS X] Type 'git' in terminal and choose the Install button in the window that pops up. This will install 'command line developer tools' that includes Git.

Step 2: Clone the VENTOS repository on a folder that you have write permission such as Desktop or home folder. Choose a folder with no space in its path and is big enough since OMNET++ and SUMO are going to be downloaded in the same folder. **The VENTOS repository is on a private Git server and you need to have a username/password. If you don't, then clone the public repository**.

```
git clone https://github.com/ManiAm/VENTOS
[public] git clone https://github.com/ManiAm/VENTOS_Public
```

You can also download a zip archive of VENTOS from Github. Note that this zip archive is just a snapshot of one particular commit of the repository (usually the tip of the branch), and it does not contain any history. This allows you to download a copy of the source code faster, but if you make any changes to the source code and decide to pull the latest VENTOS changes, then you are in trouble!

Step 3: Go to the VENTOS folder and run the 'runme' script. This bash script checks your system and installs the required packages and libraries. **Do not run the script as sudo/root**.

[1] If you are using Ubuntu, then you can check this video.

```
./runme
```

You need patience and a fast Internet. The installation process might take more than an hour! Note that the script might show you many compile-time warning messages and you can safely ignore many of them. At the end you would have three folders (VENTOS, omnetpp-5.1 and VENTOS_SUMO) plus OMNET++ launcher icon on your desktop as shown in Figure 2.

Figure 2: **All folders used by VENTOS**

Step 4: Open the OMNET++ desktop launcher using the following commands:

```
cd ~/Desktop
gedit ./opensim-ide.desktop
```

Replace the line starting with 'Exec' with the following (change the path to omnetpp accordingly). Save the file and then close it.

```
Exec=bash -i -c '/home/mani/Desktop/omnetpp-5.3/bin/omnetpp;$SHELL'
```

Now double-clicking the OMNET++ desktop shortcut opens the Eclipse IDE in an interactive shell which loads .bashrc with all the environment variables defined in previous step. If you get an error related to '--add-modules=ALL-SYSTEM', then open the ide/omnetpp.ini file and remove the last three lines.

Step 5: You can run the Eclipse IDE using the desktop shortcut or typing 'omnetpp' in terminal. The first time you run OMNET++, Eclipse IDE asks you to select a workspace. Select the folder that you will use to store all your project files. If you have intention to store all your projects on Desktop, then change the workspace to Desktop. Also check "Use this as the default and do not ask again".

"Introduction to OMNET++" page will appear. Click on "Workbench". Then it asks you to install INET framework or import some examples into your workspace. Uncheck them both since we do not need them for the time being.

Step 6: Import VENTOS project into the OMNET++ IDE by choosing "File->Import" from the menu. Choose "General->Existing Projects into Workspace" from the upcoming dialog and proceed with "Next". Choose

"Select root directory" and select the VENTOS folder. "VENTOS" should appear in the "Projects" section. Unselect "Copy project into workspace" if the VENTOS folder is already in your workspace. Click "Finish".

Step 7: Now you can build the VENTOS project. Use Ctrl+B or right-click on the project name and choose "Build Project". Wait for a while and then check the console windows at the bottom of the Eclipse IDE to make sure no errors occurred. If you are curious to know how VENTOS is built under the hood, then read Section 4 for more information.

Mac OS X users

If you get the following compile-time error then the clang compiler does not support OpenMp and you need to remove the -fopenmp flag. Right click on the VENTOS project name and click Properties. Select OMNET++ | Makemake and click on src. Click on the 'Options…' button and click the Custom tab. Remove both lines and click 'ok' two times.

```
clang: error: unsupported option '-fopenmp'
```

You should be able to run the sumo-gui application by typing sumo-gui in the terminal. If you get the following error then you need to log out/in in order to let X11 start automatically. Alternatively, you may start X11 manually by pressing cmd-space and entering "XQuartz".

```
FXApp::openDisplay: unable to open display :0.0
```

Step 8: To make sure that everything is installed correctly, run a sample simulation scenario. Click Run in the menu bar and select 'Run Configurations…'. Choose 'OMNET++ Simulation' and click on 'New launch configuration' button at the top left. Give this configuration a name like myConfig. In 'Executable', choose opp_run and set 'Working dir' to /VENTOS/examples/platoon_cacc. In 'Ini file(s)' choose omnetpp.ini. From 'Config name' choose a configuration from the drop down list like 'CACCVehicleStream1'. Leave the rest of the options to default. Click Apply and then click Run.

Step 9: OMNET++ Qtenv graphical runtime environment and SUMO-GUI window appear. Click on the red play button  on the toolbar to start the simulation.

3 Simulation Model Structure in VENTOS

If you quickly go over the files in the VENTOS project, you will notice that they are either *.ned, *.ini, *.msg, *.xml, *.cc or *.h files. In this section, we will discuss about why do we need all these file types and how they are used in the build stage (Section 4) and execution stage (Section 5) of the simulation model. You can refer to the OMNET++ manual to get more detailed information.

***.ned File**

A simulation model in OMNET++ consists of **modules** (simple or compound) that define the topology of the network. Figure 3 shows a network with three simple modules. Simple modules communicate with message passing (via **gates**) where messages may carry arbitrary data structures. Gates are the input and output interfaces of modules. Messages are sent through output gates and arrive through input gates. An

input gate and output gate can be linked by a **connection**[1]. In the figure below, gates and connections are shown with small squares and arrows respectively. Simple module A has one input gate, simple module B has one output gate and one input/output gate and simple module C has one input/output gate.

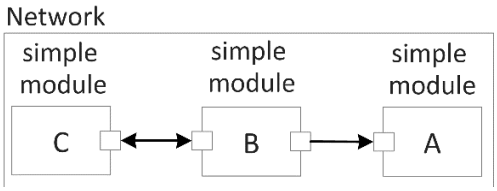


Figure 3: **A network consisting of three simple modules**

Note: Modules not only can pass messages along predefined paths via gates and connections, but they can also pass messages directly to their destination. This is useful for wireless simulations. For example simple module C above can directly send a message to simple module A.

Well-written modules are truly reusable, and can be combined in various ways like LEGO blocks. Simple modules can be grouped into a **compound** module. For example Figure 4 shows a network with a simple module and a compound module where the compound module consists of two simple modules (B and C). Note that the depth of module nesting is not limited.

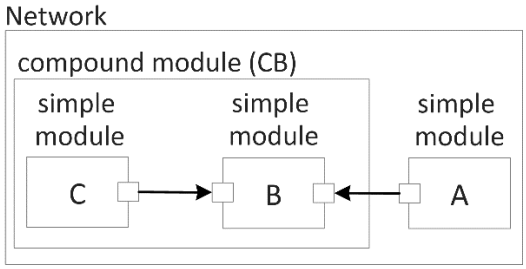


Figure 4: **A network consisting of a simple and compound modules**

Simple or compound modules may have parameters that can be used to customize their behavior and/or to parameterize the model's topology. You can define parameters of type **bool**, **int**, **double**, **string** and **xml** in a module. Parameters are variables belonging to a module that can be used to supply input to C++ code that implements the module as we will see later. Parameters can get their value from NED file or from the configuration file or even interactively from the user. A default value can also be given which is used if the parameter is not assigned otherwise.

OMNET++ uses a domain-specific language called NED (Network Description) language to define a module in a file with **.ned** extension. This is a powerful approach to separate the topology definition from behavior that is lacking in other network simulators such as NS3. For example, go to the 'src/nodes/vehicle' and double click the 'modules.ned' file in order to bring up the NED editor. Multiple modules are defined in this file using the NED language and you can switch between the source code and GUI design by clicking on the 'Design | Source' tab at the bottom of the page as shown in Figure 5.

[1] Parameters such as propagation delay, data rate and bit error rate, can be assigned to connections.

Figure 5: **NED definition of a 'vehicle' module in VENTOS**

As you can see, 'vehicle' is a compound module that consists of three submodules: appl, nic and mobility. The 'appl' and 'mobility' module are simple modules whereas 'nic' is a compound module that is made of two other submodules. You can double click on the 'nic' module to open the NED file that contains the definition of the 'nic' module.

***.h and *.cc Files**

The C++ code (*.h and *.cc) implements the internal behavior of a **simple** module. In other words, a simple module defines a black box with input gates, output gates, parameters, etc. using the NED language and the C++ code defines the internal behavior or algorithm that is running in that black box. By default, OMNeT++ looks for C++ classes of the same name as the NED type. One can explicitly specify the C++ class with the **@class** property.

Note: Compound module doesn't have C++ code.

***.ini File**

The configuration file (usually called omnetpp.ini) contains settings that control how the simulation is executed, values for model parameters, etc. The configuration file can also prescribe several simulation runs; in the simplest case, they will be executed by the simulation program one after another. You can find more information here.

***.msg File**

Message and packet contents can be defined in a syntax resembling C structs in a *.msg file. OMNET++ looks for *.msg files in the project and uses the message compiler (opp_msgc) to produce .c and .h files. You can include the message header file into your code and start using it right away. This makes your life much easier.

Figure 6: **Message files are converted into C++ files**

4 Building VENTOS

OMNET++ generates Makefile for the VENTOS project automatically using opp_makemake and the build process is done using 'make' by reading the generated Makefile(s). You can control the Makefile generation using the GUI interface provided by OMNET++. Right click on the VENTOS project in the Eclipse IDE, and then go to 'Properties | OMNET++ | Makemake'. You can find more information in the OMNET++ manual here.

The build configuration from the GUI interface is saved into the .oppbuildspec file that is later used by the IDE to generate Makefile(s). The following steps show you what happens when you build the VENTOS project from the IDE (Ctrl+B):

1. The .oppbuildspec file is read and opp_makemake is invoked with the correct arguments in order to generate temporary Makefile(s) in memory.

2. The temporary Makefile(s) is compared to the one on the file system. If they are the same, then we go to step 3. If the in-memory Makefile and the one stored on the file system is different[1], then the in-memory Makefile is written out to the file system with the current timestamp. This will ensure that calling 'make' after this will create a full rebuild of the whole project[2]. Then we go to step 3.

3. The IDE invokes the 'make' command for the parent Makefile and the build process starts by calling the sub-Makefile in the 'src' folder. The 'make' is invoked with the following default arguments:

```
make MODE=debug -j2 all
```

4. The build process starts by converting all *.msg files into C++ source files with the help of the message compiler (opp_msgc).

5. Now that all C++ files are ready, they are compiled into object form (*.o) and then linked with the simulation kernel (oppsim) and its dependencies (oppenvir, oppcommon, oppnedxml, etc.) and optionally

[1] The generated Makefile is changed if there is any changes in the .oppbuildspec file, makefrag file or IDE settings or a new source file is added/removed to/from the project.
[2] The build output depends on the Makefile itself too.

with one or more user interface libraries (opptkenv, oppqtenv and oppcmdenv) as well as user libraries (boost, curl, etc.). The build output is generated as a shared file called **libVENTOS.so**.

Figure 7 shows a high-level view of the build process.

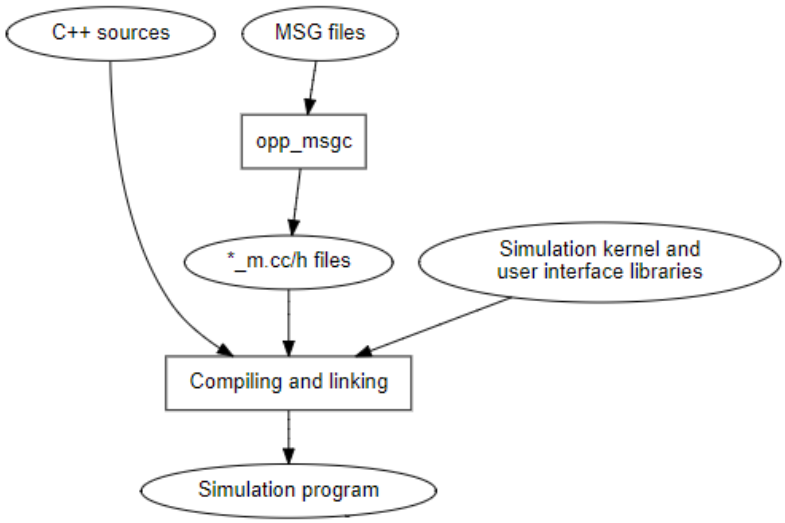


Figure 7: **OMNET++ project build process**

Note that the libVENTOS.so shared library depends on many other shared libraries that they all need to be installed on your system. You can use the 'ldd' command in Linux to print the shared libraries required by libVENTOS.so as shown in Figure 8.

```
maniam@ubuntu: ~/Desktop/VENTOS/out/gcc-debug/src
maniam@ubuntu:~/Desktop/VENTOS/out/gcc-debug/src$ ldd libVENTOS.so
        linux-vdso.so.1 =>  (0x00007ffd8a5aa000)
        libboost_system.so.1.58.0 => /usr/lib/x86_64-linux-gnu/libboost_system.so.1.58.0 (0x00007f493047d000)
        libboost_filesystem.so.1.58.0 => /usr/lib/x86_64-linux-gnu/libboost_filesystem.so.1.58.0 (0x00007f4930264000)
        libboost_serialization.so.1.58.0 => /usr/lib/x86_64-linux-gnu/libboost_serialization.so.1.58.0 (0x00007f4930003000)
        libcurl-gnutls.so.4 => /usr/lib/x86_64-linux-gnu/libcurl-gnutls.so.4 (0x00007f492fd96000)
        libshark_debug.so.0 => /usr/local/lib/libshark_debug.so.0 (0x00007f492f11f000)
        libblas.so.3 => /usr/lib/libblas.so.3 (0x00007f492eee0000)
        liboppenvird.so => /home/maniam/Desktop/omnetpp-5.1/lib/liboppenvird.so (0x00007f492eb04000)
        liboppsimd.so => /home/maniam/Desktop/omnetpp-5.1/lib/liboppsimd.so (0x00007f492e456000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f492e251000)
        libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f492decf000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f492dbc6000)
        libgomp.so.1 => /usr/lib/x86_64-linux-gnu/libgomp.so.1 (0x00007f492d9a3000)
        libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f492d78d000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f492d570000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f492d1a5000)
        libidn.so.11 => /usr/lib/x86_64-linux-gnu/libidn.so.11 (0x00007f492cf72000)
        librtmp.so.1 => /usr/lib/x86_64-linux-gnu/librtmp.so.1 (0x00007f492cd56000)
        libnettle.so.6 => /usr/lib/x86_64-linux-gnu/libnettle.so.6 (0x00007f492cb1f000)
        libgnutls.so.30 => /usr/lib/x86_64-linux-gnu/libgnutls.so.30 (0x00007f492c7ef000)
        libgssapi_krb5.so.2 => /usr/lib/x86_64-linux-gnu/libgssapi_krb5.so.2 (0x00007f492c5a5000)
        liblber-2.4.so.2 => /usr/lib/x86_64-linux-gnu/liblber-2.4.so.2 (0x00007f492c395000)
        libldap_r-2.4.so.2 => /usr/lib/x86_64-linux-gnu/libldap_r-2.4.so.2 (0x00007f492c144000)
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f492bf2a000)
        liblapack.so.3 => /usr/lib/liblapack.so.3 (0x00007f492b745000)
```

Figure 8: **Using 'ldd' to print the shared libraries used by libVENTOS.so**

5 Running VENTOS

As discussed in the previous section, the build output is a share library called libVENTOS.so. Simulations compiled to a shared library can be run using the opp_run program that provides the main() entry point for the simulation. For example, the shared library libVENTOS.so can be run with the following command, where the -l option tells opp_run to load the given shared library.

```
opp_run -l VENTOS
```

In reality, you need to provide more arguments for opp_run. The Eclipse IDE uses the run configuration that you have provided to run your simulation. Let's go back to the sample scenario that we run previously in Section 2. The run configuration looks like Figure 9.

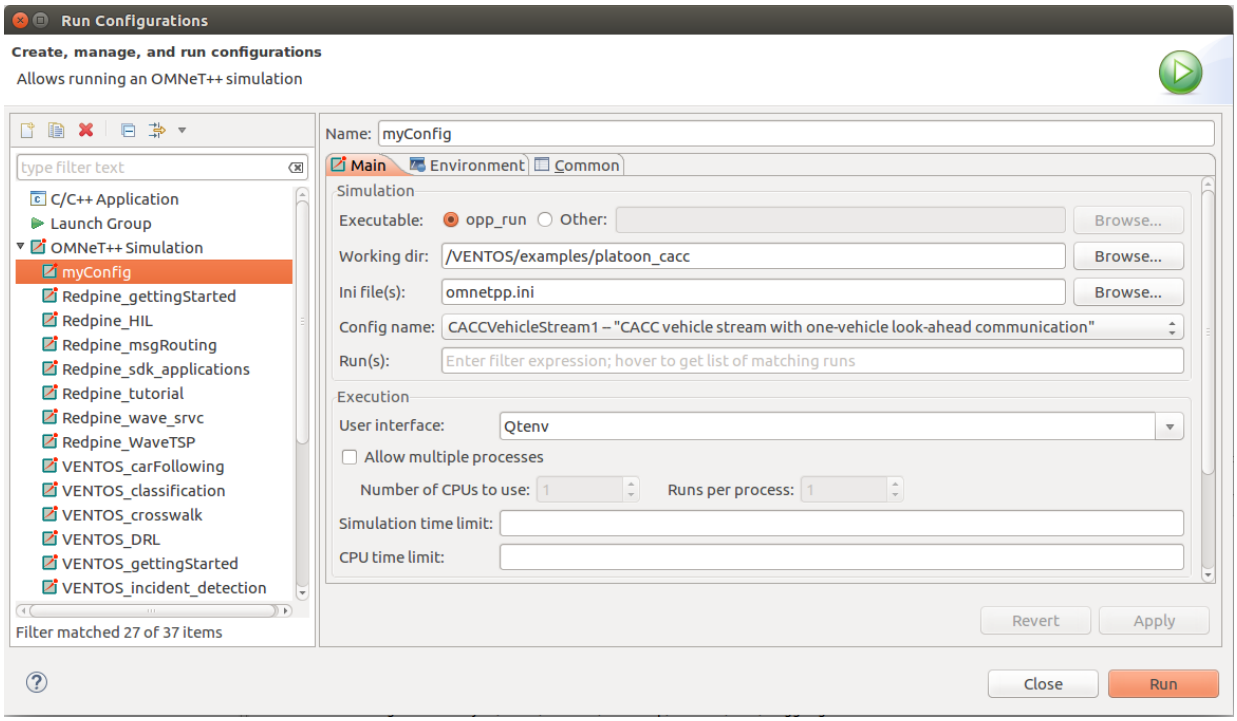


Figure 9: **Sample run configuration in the Eclipse IDE**

When you click 'Run' the Eclipse IDE invokes opp_run with the following arguments:

```
opp_run -m -u Qtenv -c CACCVehicleStream1 -n ../.. -l ../../src/VENTOS omnetpp.ini
```

You can find more information on the opp_run arguments in the OMNET++ manual here. In short:

- `-u Qtenv` tells omnetpp to run under Qtenv
- `-c <configname>` option is used to select a configuration
- `-n` option is used to specify the NED path
- `-l` option is used to load shared libraries

To get the command line used for running the simulation, run the simulation from the IDE. Then click on 'Debug' tab and right click on your program and select 'properties'. The IDE shows the currently used command line. In newer versions of OMNET++ the run command is printed at the beginning of the Console tab like the following:

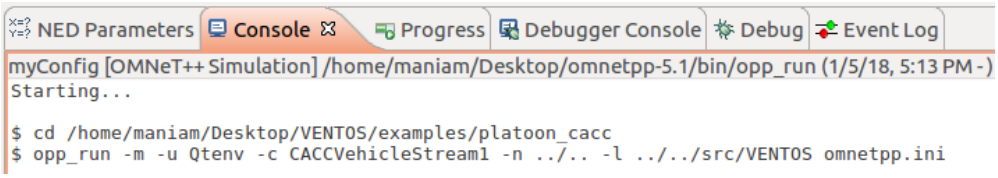


Figure 10 shows a high-level view of how the VENTOS project is being executed. As you can see, NED and ini files do not play a part in the build process, as they are loaded by the simulation program at runtime. In the following we will describe the execution steps with more details.

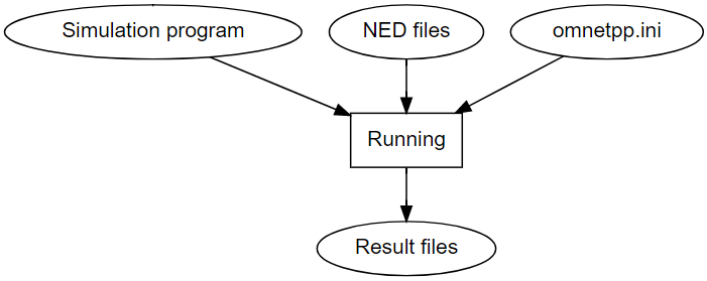


Figure 10: **OMNET++ project execution process**

The main() function of the simulation is in the 'evmain.cc' file located in the 'omnetpp/src/envir' folder. It invokes the setupUserInterface method in the 'startup.cc' file. The following steps are taken:

- It first registers and sets up the appropriate **user interface** (Cmdenv, Tkenv or Qtenv). Note that all user interfaces are derived from the EnvirBase class.

- It then creates a new simulation manager object of type cSimulation as following. The cSimulation is the central class in OMNET++ and stores the active simulation model, and provides methods for setting up, running and finalizing simulations. Most cSimulation methods are not of interest for simulation model code, and they are used internally, e.g. by the user interface libraries to set up and run simulations.

```
simulation = new cSimulation("simulation", app);
cSimulation::setActiveSimulation(simulation);
```

As you can see, the simulation objects takes the environment object of type EnvirBase (pointing to the active user interface) as the second argument.

- Method run() is called in the EnvirBase class that subsequently calls setup() that is responsible for loading NED files. It then calls doRun method in the active user interface (Cmdenv, Tkenv or Qtenv). From now on, it is up to the active user interface to control the simulation. Although they have different implementation details, they use the following lines to get the next event from the FES and then execute it in a while loop.

```
// query which module will execute the next event
cEvent *event = sim->takeNextEvent();

// do a simulation step
sim->executeEvent(event);
```

6 How VENTOS Works (in nutshell)?

Many V2X communication scenarios need to dynamically change the behavior of microscopic traffic simulation. SUMO provides the TraCI (Traffic Control Interface) which allows an external application to retrieve information from the traffic simulation or to influence the simulation behavior. TraCI uses a TCP based client/server architecture to provide access to SUMO where the TCP server is listening on a specific port. SUMO, when started in TraCI mode, only prepares the simulation and waits for an external application to take over the control.

Figure 11 shows a sample simulation scenario in VENTOS. The simulation is performed by running the OMNET++ network simulation and SUMO road traffic simulation in parallel. OMNET++ and SUMO are connected using the TraCI. Movement of nodes (vehicle/bikes/pedestrians) in SUMO is reflected as movements of nodes in OMNET++ simulation. Nodes in OMNET++ can interact with each other using V2X wireless communication and their behavior in SUMO is changed through TraCI.

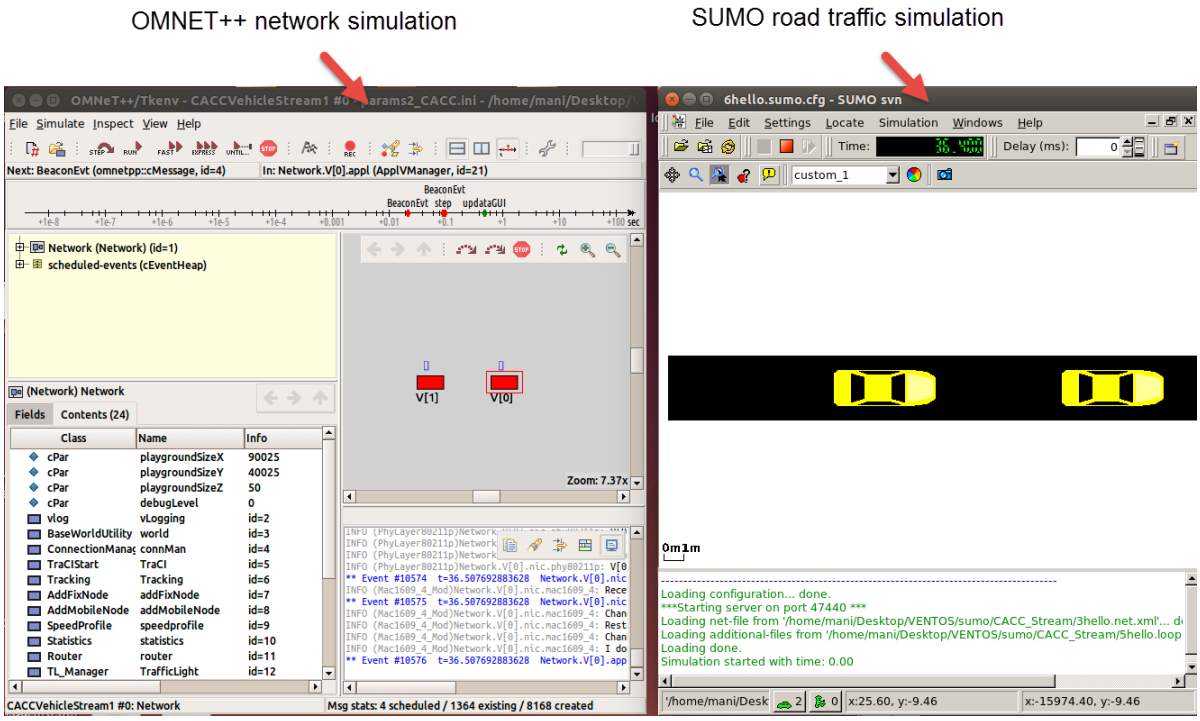


Figure 11: **A scenario showing the interaction between OMNET++ and SUMO**

The following describes the step-by-step procedure:

1. VENTOS starts running.

2. VENTOS uses fork to create a new process and then starts an instance of SUMO in TraCI mode.
3. SUMO prepares the simulation by loading the SUMO configuration file (.sumocfg). It then starts a TCP server and waits for an incoming connection from VENTOS to take over the control.
4. VENTOS connects to the TCP server in SUMO.
5. VENTOS performs network simulations for one time-step and then forces SUMO to perform simulation for the same time-step. For each node inserted into SUMO, a node is mirrored in OMNET++ as shown in Figure 11. Movements of each yellow vehicle in SUMO is reflected as movements of the corresponding red vehicle in OMNET++.

VENTOS uses a timer to call simulationStep TraCI command periodically in order to advance SUMO simulation one step at a time. Figure 12 illustrates this when SUMO time step is 500 ms. You can find more detailed information in Part 3, Section 8.1.

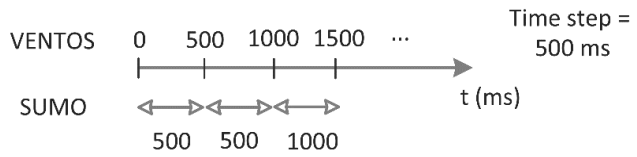


Figure 12: **How VENTOS advances SUMO simulation every time step (500 ms)**

7 How to Debug VENTOS?

Debugging is an integral part of software development lifecycle and involves identifying, locating and finally correcting errors in your code. You can debug your program to find compile-time errors, run-time errors or logic errors.

Compile-time errors can happen due to syntax errors, type-checking errors, etc. Pay close attention to compile-time warnings too. These warnings won't keep your code from compiling (unless you ask the compiler to treat warnings as errors), and you might be tempted to just ignore them, but DON'T. Compiler warnings are often indicators of future bugs. If your program compiles correctly then it is well-formed and you can run it now.

Run-time errors can happen due to division by zero, deferencing a null pointer, running out of memory, overflow, etc. Also there can be errors that are detected by the program itself such as trying to open a file that isn't there. Make sure to test your program with all possible inputs/conditions. Debugging run-time errors can be tough sometimes and debugging tools such as GDB are very useful. The run-time error is caught by the GDB debugger and the simulation is stopped right before the error. You can look at the stack trace[1] and examine variables to find the root cause of the error.

If your program is free of compile-errors and run-time errors then it doesn't mean that it will operate as you expect. A **logic error** is a bug in a program that causes it to operate incorrectly, but not to terminate abnormally (or crash). A program with a logic error is a valid program, though it does not behave as intended. The only clue to the existence of logic errors is the production of wrong output. Multiplying two

[1] The stack trace is a list of all of the methods that were being executed when the program crashed.

numbers instead of adding them together produces a logic error. As another example, missing parenthesis in an arithmetic calculation might produce a wrong output due to operator precedence.

Logic errors are due to flaws in the thinking of the programmer and are often hard to debug. You can monitor the execution of your program so you can locate and remove logic errors. You can go through the output logs (Part 3, Section 13) or use OMNET++ sequence chart. The most useful practice is to add a breakpoint to your program. Breakpoints are special markers that you can set at any executable line of code. You cannot place them on comments or whitespace. When a running program reaches a breakpoint, execution pauses, allowing you to examine the values of variables and view the order in which methods are called to help determine whether logic errors exist. You can also "step" through the program one statement at a time.

7.1 Debugging Example

Let's see how you can debug a VENTOS application in OMNET++ IDE:

Step 1: In the 'Project Explorer' go to src/nodes/vehicle folder and open '01_Beacon.cc file'.

Step 2: Find the 'initialize' method and append these two lines.

```
TraCI2 = NULL;
TraCI->TraCIclosed = false;
```

Step 3: Press Ctrl+B to re-build the project. The C++ compiler shows a compile-time error in the console tab as shown in Figure 13. The error messages are clickable and you can double click on them to go to the error line. The 'TraCI2' is not a valid keyword and you can fix it by renaming it to TraCI. Save the changes and then press Ctrl+B to re-build the project. The program is now compiled without any errors and we can run it in the next step.

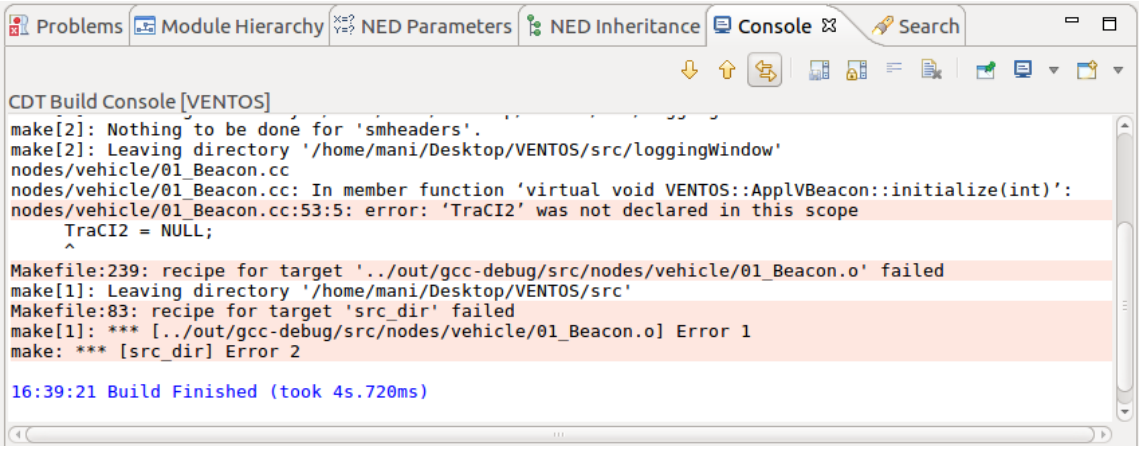


Figure 13: **Compile-time errors are shown in the Console tab**

Step 4: Click Run in the menu bar and select 'Run Configurations…'. Choose 'OMNET++ Simulation' and click on 'New launch configuration' button at the top left. Give this configuration a name like myDebug. In 'Executable', choose opp_run and set 'Working dir' to /VENTOS/examples/platoon_cacc. Choose 'omnetpp.ini' in 'Ini file(s)' and from 'Config name' choose 'CACCVehicleStream1'. Leave the rest of the

options to default. Click Apply and then click Run. Click on the red play button on the toolbar to start the simulation.

Step 5: The simulations stops with an error as shown in Figure 14. The following error message is also shown in the console tab. Unix systems return errono 128+signal when a signal is received. Thus, exit code 139 is signal number 11 which is SIGSEGV (invalid memory reference). This means that the program is using an invalid pointer. We are accessing a variable using the NULL pointer TraCI.

```
Simulation terminated with exit code: 139
```

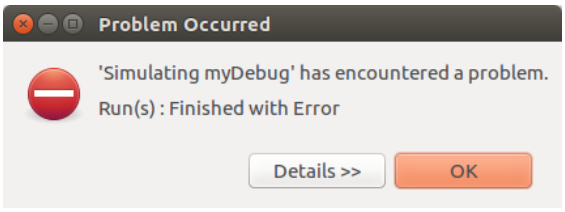


Figure 14: **Simulation error message box**

Step 6: In order to find the error line, you need to run the application in debug mode using GDB. Click Run in the menu bar and this time select 'Debug Configurations…'. Choose the previous configuration 'myDebug' and click Debug. Click on the red play button on the toolbar to start the simulation in debug mode. The run-time error is caught by the GDB debugger and the simulation is stopped right before the error. The IDE asks you to switch to the 'debug' perspective that has views for displaying stack trace, variables and break point management. Click yes and check 'Remember my decision'.

Debugging is only possible when your program is compiled in debug (and not release) mode. Right click on the VENTOS project name and go to 'Build Configurations | Set Active' and make sure that debug is selected. In the debug mode, the IDE invokes 'make' like this:

```
make MODE=debug all
```

As a result, the source code is compiled with -g and -Wall options. The -Wall option turns on all warnings. Warnings are diagnostic messages that report constructions that are not inherently erroneous but that are risky or suggest there may have been an error. The -g option produces debugging information in the operating system's native format. GDB can work with this debugging information.

In the release mode, the IDE invokes 'make' like the following. As a result, the source code is compiled with -O2 and -DNDEBUG=1 options. The -O2 option turns on level 2 optimization. The -DNDEBUG option defines NDEBUG macro that is used internally by the OMNET++ for data logging code.

```
make MODE=release all
```

The compiled files in debug and release mode are stored separately in out/gcc-debug and out/gcc-release folders respectively. So when switching modes, the project is not re-compiled from scratch.

Step 7: In the debug perspective, the stack trace is shown on the top left. The topmost method is the method that got called most recently. Most likely, it's where the error happened. Moving down from there, you can trace the sequence of methods that were called to the first method that was called. In a

typical stack trace, there will be many methods that you did not write near the bottom of the list and they belong to OMNET++. You can safely assume that the OMNET++ code is bug free, so you should begin tackling the stack trace when it first enters one of your methods.

In this particular example the topmost method (ApplVBeacon::initialize) is where the error happened and you can easily see the error line highlighted with green. Hovering your mouse on any variable in the active method will show its value.

Note that in general the method that ran into the error is not necessarily the one that caused it. This happens most often if the method accepts a parameter. For example, if you passed a null value as the actual parameter into a method, an error would result because a null value should never have been passed. You can, however, trace down from this method to figure out where your error was caused.

Step 8: To remove the run-time error simply comment or remove the two newly added lines. Switch from the debug perspective to simulation perspective by clicking on the  icon located in top right.

Step 9: Now let's add a breakpoint at line number 46 in 01_beacon.cc file. Simply double-click on the left margin on line 46 as shown in Figure 15. Run the simulation in debug mode as before and when the program reaches the breakpoint, execution pauses, allowing you to examine the values of variables and view the order in which methods are called. You can also "step" through the program one statement at a time by going to Run and click Step Into, Step Over and Step Return or press the corresponding shortcut keys.

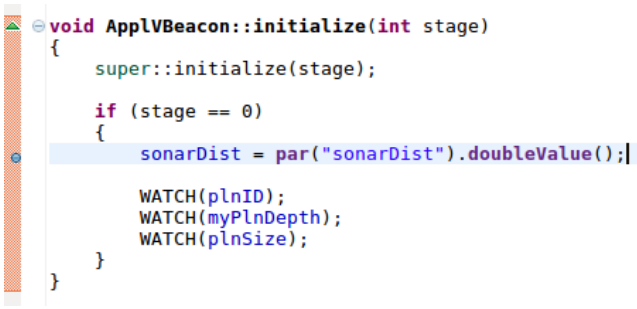


Figure 15: **Adding a breakpoint to a line**

Another interesting feature of the IDE is macro expansion. For example WATCH in the code is a macro defined by OMNET++. Hover your mouse over WATCH to see its underlining code.

7.2 Debug SUMO

You can import VENTOS_SUMO into the IDE and compile/run it similar to VENTOS.

Step 1: Import VENTOS_SUMO project into the OMNET++ IDE by choosing "File->Import" from the menu. Choose "General->Existing Projects into Workspace" from the upcoming dialog and proceed with "Next". Choose "Select root directory" and select the VENTOS_SUMO folder. "VENTOS_SUMO" should appear in the "Projects" section. Unselect "Copy project into workspace" if the VENTOS_SUMO folder is already in your workspace. Click "Finish".

Step 2: Build the project by pressing Ctrl+B or right-click on the project name and choose "Build Project".

Step 3: Click Run in the menu bar and select 'Debug Configurations…'. Choose 'C/C++ Application' and click on 'New launch configuration' button at the top left. Give this configuration a name like 'SUMO Debug'. Set the 'Project' to VENTOS_SUMO and set 'C/C++ Application' to bin/sumo-guiD. Click apply and then run.

Step 4: SUMO-GUI application starts running and you can perform vehicle traffic simulation as before. As an example, go to the 'VENTOS/examples/gettingStarted/1_two_subsequent_streets' folder from the terminal and generate the SUMO network file using the following command:

```
netconvert --node-files=example.nod.xml --edge-files=example.edg.xml --output-
file=example.net.xml
```

From the SUMO-GUI window, open the configuration file located in the folder above and click the green play button to start the simulation. You can go to VENTOS_SUMO/src/microsim and add breakpoints in different files to study the behavior of different classes. If you are looking for the 'main' method of SUMO-GUI, then you can find it in 'src/guisim_main.cpp' file. Similarly, the 'main' method of SUMO (command-line application) is located in 'src/sumo_main.cpp' file.

7.3 Debug SUMO in TraCI mode

As previously mentioned in Section 6, SUMO provides the TraCI (Traffic Control Interface) which allows an external application to retrieve information from the traffic simulation or to influence the simulation behavior. TraCI uses a TCP based client/server architecture to provide access to SUMO where the TCP server is listening on a specific port. You can start SUMO in the TraCI mode by passing the following arguments:

```
--remote-port 45585 --configuration-file <absolute_path_to_configuration_file>
```

Step 1: Click Run in the menu bar and select 'Debug Configurations…'. Choose the configuration that we have created in the previous section (SUMO Debug). Click on the 'Arguments' tab and copy/paste the above argument, but make sure to specify the absolute path of the configuration file that we have created in the previous section. Click Apply and close the window (do not run the application for now).

Step 2: SUMO, when started in TraCI mode, only prepares the simulation and waits for an external application such as VENTOS to take over the control. By default, VENTOS uses fork to create a new process and then starts an instance of SUMO in TraCI mode on a random available port. Open 'omnetpp.ini' located in 'VENTOS/examples/gettingStarted' folder and change it to the following configuration:

```
include ../omnetpp_general.ini

[Config tutorial1]
description = "This config shows you how to debug SUMO in TraCI mode"

Network.TraCI.active = true
Network.TraCI.SUMOapplication = "sumo-guiD"
Network.TraCI.SUMOconfig = "1_two_subsequent_streets/example.sumocfg"

Network.TraCI.forkSUMO = false
Network.TraCI.remotePort = 45585
```

Step 3: Create a debug configuration called tutorial (if you haven't done so) to run VENTOS in debug mode. Now we have two debug configurations: 'SUMO Debug' and 'tutorial'.

Step 4: Before running the projects, let's put a breakpoint in the SUMO-GUI code to see if execution will be stopped in the breakpoint. Go to the VENTOS_SUMO/src/microsim/cfmodels/MSCFModel_Krauss.cpp file and add a breakpoint in the first line of the 'followSpeed' method. The default car-following models for vehicles is Krauss and the followSpeed method is used to calculate the following speed.

Step 5: Launch 'SUMO Debug' first to run the SUMO-GUI application in TraCI mode on port 45585 followed by 'tutorial' that runs VENTOS and establishes the TraCI connection. You can use the 'Launch group' and asks the IDE to launch 'SUMO Debug' and 'tutorial' configurations in a sequential order for you.

Click Run in the menu bar and select 'Debug Configurations…'. Choose 'Launch Group' and click on 'New launch configuration' button at the top left. Give this configuration a name like SUMO_VENTOS. Click Add and select 'SUMO Debug' and make sure the 'Launch Mode' is set to debug. Click ok. Click on Add again and select 'tutorial' this time. Also make sure the 'Launch Mode' is set to debug and click ok. This is shown in Figure 16. Click Apply and then click Debug.

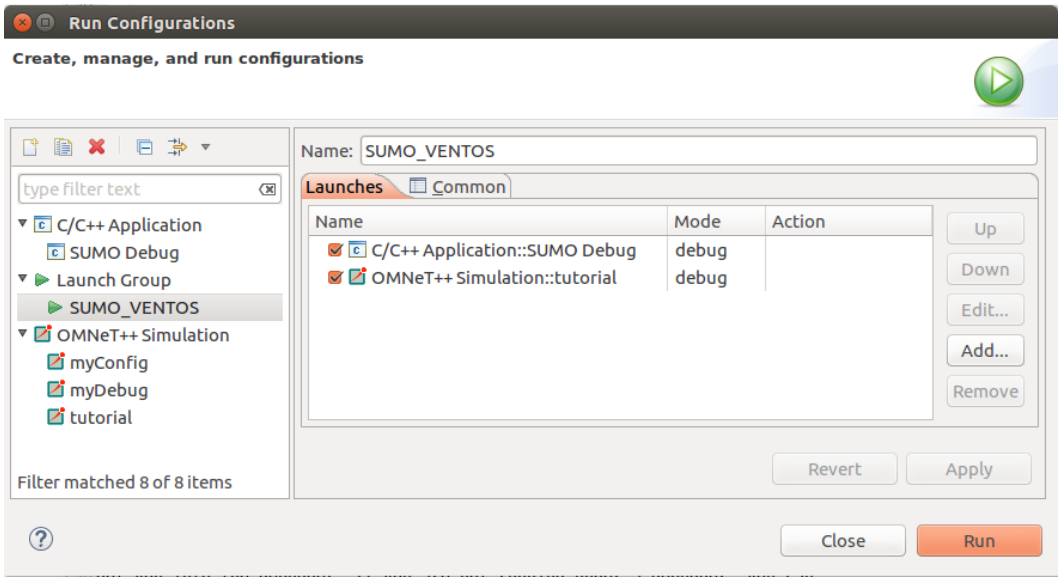


Figure 16: **Creating a launch group in the IDE**

Step 6: SUMO-GUI is executed first, followed by VENTOS. Click the green run button in SUMO-GUI and then click the green run button in OMNET++ Qtenv.

Step 7: At some point, code execution in SUMO pauses right before the breakpoint, allowing you to examine the values of variables and view the order in which methods are called. You can also "step" through the program one statement at a time.

8 Getting the Latest Changes

Many projects including VENTOS use Git as the main Version Control System (VCS). Git is very powerful, but it can be very confusing at first. There are many good online tutorials such as this, this, this and this that guide you through using Git professionally. You can also refer to our Git tutorial document that teaches you the fundamentals of the Git system.

To get the latest VENTOS changes you can go to the VENTOS folder from the terminal and do a 'git pull' and then re-build the project. You can do this inside the Eclipse IDE too by right-clicking the VENTOS project name and go to 'Team | Pull'. You can display the Git commit history by going to 'Team | Show in History'. This will open the History tab as shown in Figure 17. You can navigate between different commits and check which files are modified/created. You can even right-click on a file and click on 'Compare with Previous Version' to open a text compare window as shown Figure 17.

Similarly, to get the latest VENTOS_SUMO changes you can go to the VENTOS_SUMO folder from the terminal and do a 'git pull' and then re-build the project by running the 'runme' script in that folder. Note that the VENTOS_SUMO folder is a clone of the official SUMO SVN repository at here[1] and we are tracking our changes in a Git repository. Although, there is a Git mirror of SUMO in here that gets synchronized hourly using git-svn, we do not use it due to lack of efficiency.

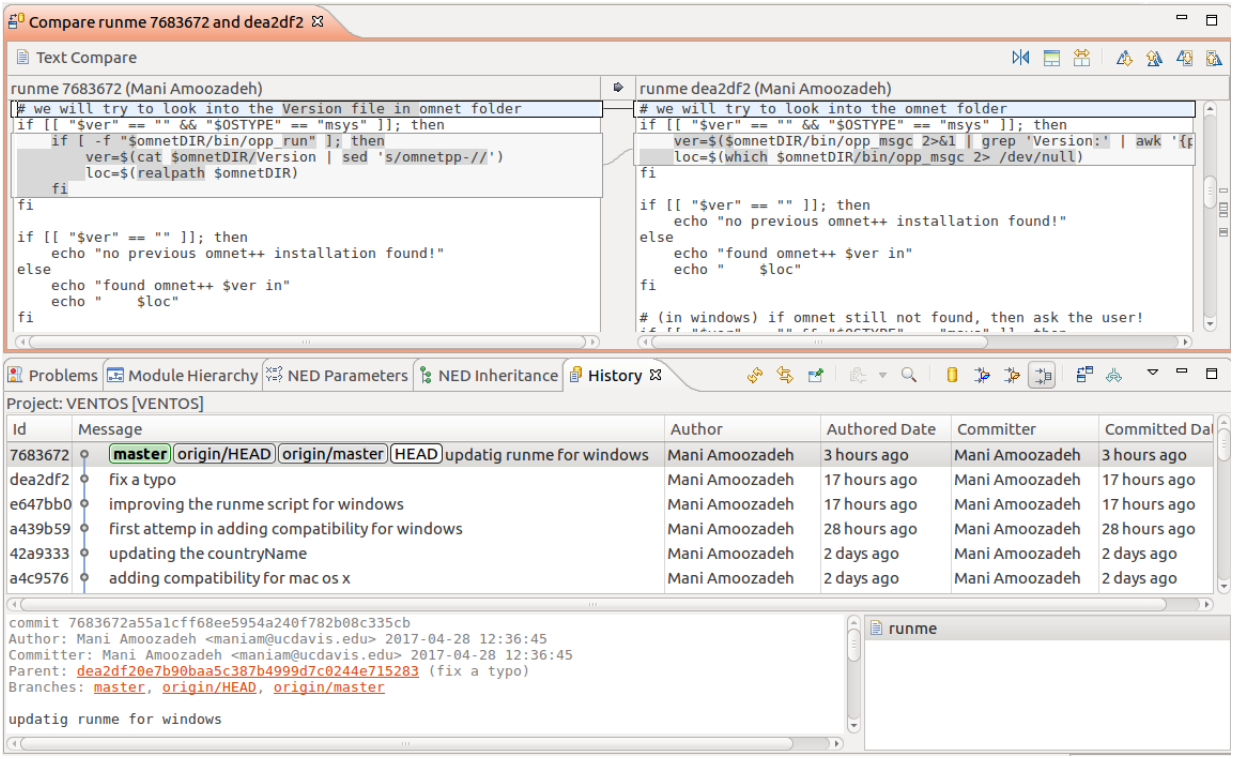
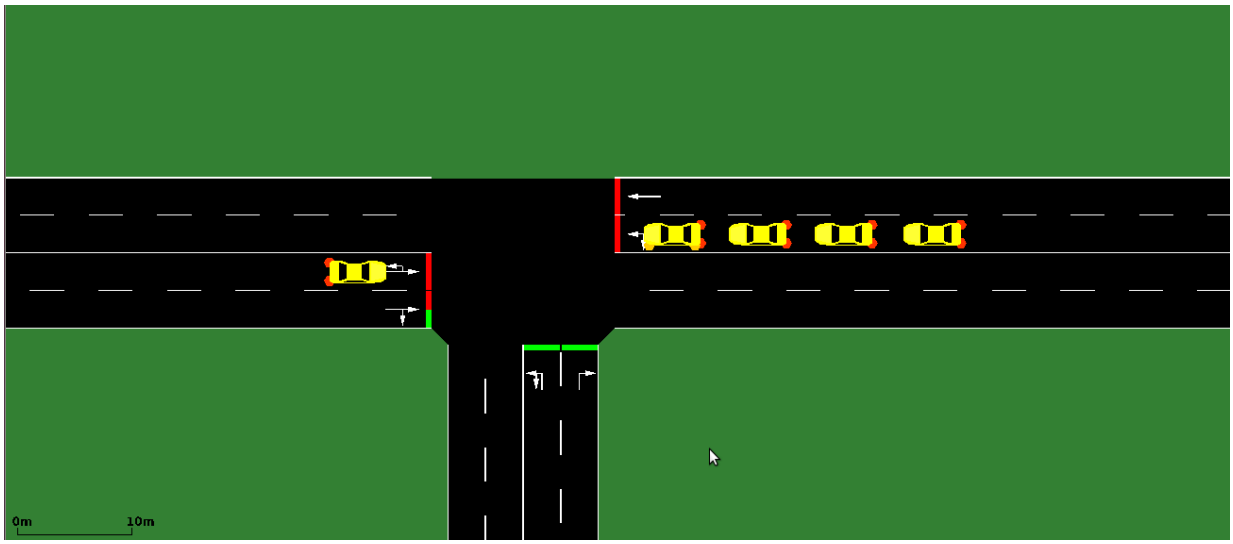


Figure 17: **Git management in OMNET Eclipse IDE**

[1] Unfortunately SUMO developers have not migrated from SVN to Git (yet!)

Part 2: Road Traffic Simulation with SUMO



1 Getting Started with SUMO

VENTOS uses SUMO to perform road traffic simulation. In order to create your own simulations on custom road maps, you will need to learn how SUMO works. SUMO is a microscopic traffic simulator and each vehicle is modelled explicitly, has its own route and moves individually through the road network. VENTOS extends SUMO's functionality by adding new car-following models for ACC/CACC vehicles as-well-as defining new TraCI commands.

SUMO is meant to be used to simulate networks of a city's size, but you can of course use it for smaller networks and larger, too, if your computer power is large enough. There are no hard limits on the number of streets, intersections or vehicles in SUMO. Make sure that you have enough RAM when trying to build or run scenarios beyond city-size. It may be necessary to use the 64bit version of SUMO to make use of available RAM. The maximum length of a simulation scenario is 292 million years [ref].

SUMO comes with many applications (Table 1) and you can find them all in the VENTOS_SUMO/sumo/bin folder. Each application has a certain purpose and runs individually. This is something that makes SUMO different to other simulation packages where, for instance, the dynamical user assignment is made within the simulation itself, not via an external application like here. This split allows an easier extension of each of the applications within the package because each is smaller than a monolithic application that does everything. Also, it allows the usage of faster data structures, each adjusted to the current purpose, instead of using complicated and ballast-loaded ones. Still, this makes the usage of SUMO a little bit uncomfortable in comparison to other simulation packages [ref].

Table 1: **List of SUMO applications**

Name	Description
SUMO	The microscopic simulation with no visualization; command line application
SUMO-GUI	The microscopic simulation with a graphical user interface
NETCONVERT	Network importer and generator; reads road networks from different formats and converts them into the SUMO-format
NETEDIT	A graphical network editor
NETGENERATE	Generates abstract networks for the SUMO-simulation
DUAROUTER	Computes fastest routes through the network, importing different types of demand description. Performs the DUA
JTRROUTER	Computes routes using junction turning percentages
DFROUTER	Computes routes from induction loop measurements
OD2TRIPS	Decomposes O/D-matrices into single vehicle trips
POLYCONVERT	Imports points of interest and polygons from different formats and translates them into a description that may be visualized by SUMO-GUI
ACTIVITYGEN	Generates a demand based on mobility wishes of a modeled population
MESO	A simulation which uses a mesoscopic queue-model, performing simulations up to 100 times faster than the pure microscopic simulation
MESO-GUI	The mesoscopic simulation with a graphical user interface

The two most important applications that are used for microscopic simulation are **SUMO** and **SUMO-GUI**. SUMO-GUI (as shown in the right window of Figure 11) provides a GUI for visualization and is very useful for debugging purposes. On the other hands, SUMO is the command-line version that provides no visualization but is faster and is suitable for long-run and batch simulations. Both SUMO and SUMO-GUI need two input files: **network file** and **traffic demand file** as shown in Figure 18.

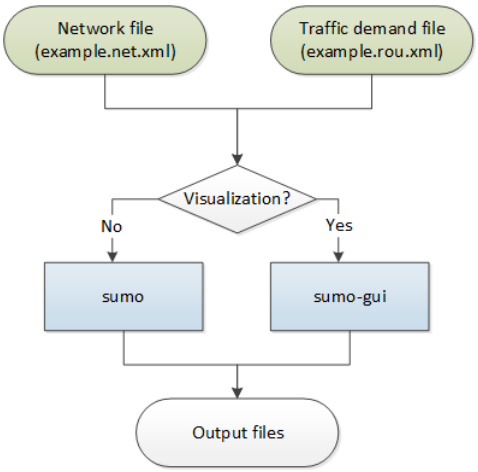


Figure 18: **Network file and Traffic demand file**

SUMO Network File

SUMO network file is encoded as XML file and describes the traffic-related part of a road map and defines the junctions (nodes) and roads (edges). Edges are unidirectional and are defined as a collection of lanes, including the position, shape and speed limit of every lane. Nodes are defined along with their coordinates

and right of way regulation as well as traffic light logic (if present). Although being readable by human beings, a SUMO network file is not meant to be created/edited by hand. Rather you should use one of the tools provides by SUMO to generate the network file as we will discuss in Section 2.

SUMO Traffic Demand File

After having generated a SUMO network file, one could take a look at it using SUMO-GUI, but no cars would be driving around. The next step is to define the **vehicle's type** such as length, acceleration and deceleration capability, and maximum speed in a traffic demand file. This file should also specify the **route** for each vehicle which is defined as all the edges the vehicle will pass. The traffic demand file is not as complex as a network file and you can generate it by hand. For bigger networks, SUMO provides tools to generate routes as we will discuss in Section 3.

2 Generating SUMO Network File

As previously mentioned, the SUMO network file is not meant to be created/edited by hand. Rather you should use one of the tools provided by SUMO such as NETCONVERT or NETGEN to generate the network file for you as shown in Figure 19. We will discuss three methods in the following sections.

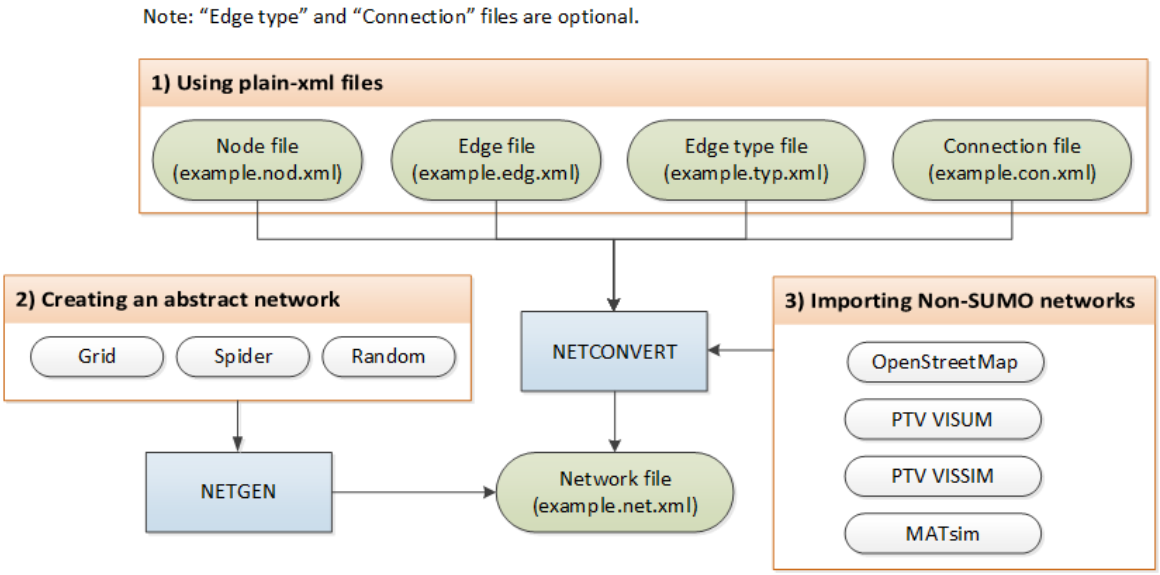


Figure 19: **Methods for generating SUMO network file**

2.1 Method 1: Using plain-xml files

You can define a set of plain-xml files ('node file', 'edge file', 'edge type file' and 'connection file') which describe the network topology and geometry to generate the SUMO network file. You need to make each of these files by hand and then use NETCONVERT to generate the SUMO network file. To generate a network file, 'node file' and 'edge file' are required whereas 'edge type file' and 'connection file' are optional. We will show you how to generate a network file using examples.

2.1.1 Example: Two subsequent streets with two lanes each

Let's generate a simple SUMO network file with two streets, subsequent to each other. As previously mentioned, in SUMO a street network consists of nodes (junctions) and edges (streets connecting the junctions). Thus, if we want to create a network with two streets, subsequent to each other, we need three nodes and two edges. We first try to create 'node file' in step 1 and then create 'edge file' in step 2. In step 3, we will feed these two files into NETCONVERT to generate network file. **You can also find this example in VENTOS/examples/gettingStarted/1_two_subsequent_streets folder.**

Step 1: The 'node file' looks like the following. You can edit a file with a text editor of your choice and save this for instance as example.nod.xml where .nod.xml is the default suffix for SUMO node files. All known extensions in SUMO are listed here.

```
<nodes>
    <node id="1" x="-200.0" y="0.0" />
    <node id="2" x="0.0" y="0.0" />
    <node id="3" x="+200.0" y="0.0" />
</nodes>
```

Each node has an id for future reference and specifies x- and y-coordinate that shows the distance to the origin in meters.

Step 2: The 'edge file' looks like the following. You can edit a file with a text editor of your choice and save this for instance as example.edg.xml where .edg.xml is the default suffix for SUMO edge files. All known extensions in SUMO are listed here.

```
<edges>
    <edge id="street1" from="1" to="2" numLanes="2" speed="30" />
    <edge id="street2" from="2" to="3" numLanes="2" speed="30" />
</edges>
```

An edge in SUMO is unidirectional and starts from 'source node id' and ends at 'target node id'. The length of this edge will be computed as the distance between the starting and the end point. You need to specify a unique id for each edge for future references. For each edge, you can optionally specify the number of lanes using 'numLanes' and maximum speed in all lanes using 'speed'. Otherwise, the edge will have a single lane with maximum allowed speed of 13.9 m/s=50 km/h by default. Lanes of edge "X" are named X_0, X_1, X_2, etc. where X_0 is the right-most lane.

Step 3: Now that we have nodes and edges we can call NETCONVERT to create a network file. Make sure NETCONVERT is somewhere in your PATH.

```
netconvert --node-files=example.nod.xml --edge-files=example.edg.xml
--output-file=example.net.xml
```

You can open the generated SUMO network file using SUMO-GUI as shown in Figure 20. As you can see, both ends of a street needs an according node. Node 1 and Node 2 specify the ends of steet1 whereas Node 2 and Node 3 specify the ends of street 2. Edge street1 has two lanes called street1_0 and street1_1 and edge street2 has also two lanes called street2_0 and street2_1.

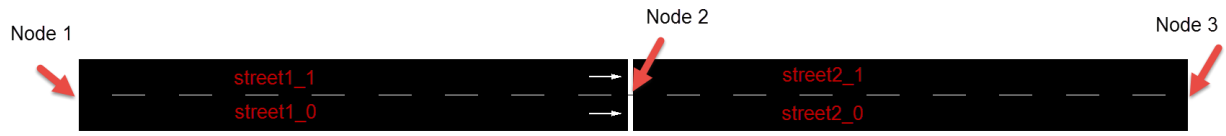


Figure 20: **Visualization of the network file in SUMO-GUI**

2.1.2 Example: 4-way priority junction

In this example we will generate a network file for a 4-way traffic intersection. **You can also find this example in VENTOS/examples/gettingStarted/2_four_way_priority_junction folder.** The 'node file' looks like the following. The first node named "C" is the node in the center of the junction and the other nodes represent four corners of the intersection. Note that the order of defining nodes does not matter.

```
<nodes>
    <node id="C" x="0.00" y="0.00" type="priority" />
    <node id="E" x="100.00" y="0.00" />
    <node id="W" x="-100.00" y="0.00" />
    <node id="N" x="0.00" y="100.00" />
    <node id="S" x="0.00" y="-100.00" />
</nodes>
```

The right-of-way computation at each intersection is based on the optional 'type' of the node. If you leave out the type of the node, it is automatically guessed by NETCONVERT but may not be the one you intended. All possible junction types are listed in Table *2*[1].

Table 2: **Junction types in SUMO**

Junction Type	Description
unregulated	The junction is completely unregulated - all vehicles may pass without braking; this may cause collisions
priority	Vehicles on a low-priority edge have to wait until vehicles on a high-priority edge have passed the junction
priority_stop	This works like a priority-junction but vehicles on minor links always have to stop before passing
traffic_light	The junction is controlled by a traffic light (priority rules are used to avoid collisions if conflicting links have green light at the same time)
traffic_light_unregulated	The junction is controlled by a traffic light without any further rules. This may cause collision if unsafe signal plans are used
traffic_light_right_on_red	The junction is controlled by a traffic light as for type traffic_light. Additionally, right-turning vehicles may drive in any phase whenever it is safe to do so (after stopping once)
right_before_left	Vehicles will let vehicles coming from their right side pass
allway_stop	The traffic approaching this intersection from all directions is required to stop before proceeding through the intersection

[1] Setting the junction type to 'unregulated' or 'traffic_light_unregulated' disables all right-of-way rules within the intersection

zipper	This junction connects edges where the number of lanes decreases and traffic needs to merge zipper-style (late merging)
rail_signal	This junction is controlled by a rail signal. This type of junction/control is only useful for rails
rail_crossing	This junction models a rail road crossing. It will allow trains to pass unimpeded and will restrict vehicles via traffic signals when a train is approaching

In this example, node "C" is given type priority. In a priority junction, vehicles on a low-priority edge have to wait until vehicles on high-priority edge have passed the junction. The allowed speed on the edge and the edge's number of lanes are also used to compute which edge has a greater priority on a junction. Given two edges e1 and e2, the following pseudocode shows how the higher priority edge is calculated. As you can see, the priority information is evaluated, first. Then the speed information, then the lane number.

```
1   if (e1->getPriority() != e2->getPriority())
2       return e1->getPriority() > e2->getPriority();
3
4   if (e1->getSpeed() != e2->getSpeed())
5       return e1->getSpeed() > e2->getSpeed();
6
7   return e1->getNumLanes() > e2->getNumLanes();
```

The 'edge file' looks like the following. EC, WC, NC and SC represent the incoming edges to the intersection and CE, CW, CN and CS represent the outgoing edges from the intersection. All edges have three lanes with maximum speed of 30 m/s. EW is the main street and NS is the side street, thus, EC and WC are given a higher priority than NC and SC.

```
<edges>
    <!-- incoming edges -->
    <edge id="EC" from="E" to="C" numLanes="3" speed="30" priority="3" />
    <edge id="WC" from="W" to="C" numLanes="3" speed="30" priority="3" />
    <edge id="NC" from="N" to="C" numLanes="3" speed="30" priority="2" />
    <edge id="SC" from="S" to="C" numLanes="3" speed="30" priority="2" />

    <!-- outgoing edges -->
    <edge id="CE" from="C" to="E" numLanes="3" speed="30" priority="1" />
    <edge id="CW" from="C" to="W" numLanes="3" speed="30" priority="1" />
    <edge id="CN" from="C" to="N" numLanes="3" speed="30" priority="1" />
    <edge id="CS" from="C" to="S" numLanes="3" speed="30" priority="1" />
</edges>
```

Now that we have nodes and edges we can call NETCONVERT as before to create a network file.

```
netconvert --node-files=example.nod.xml --edge-files=example.edg.xml
--output-file=example.net.xml
```

You can open the generated SUMO network file using SUMO-GUI as shown in Figure 21. WC, NC, EC and SC are incoming edges to the intersection. The right-of-way rules are indicated in SUMO-GUI by the colored bars at the end of each lane.

You can turn on the 'lane to lane connection' in order to see how the end of each lane is connected to the beginning of another lane[1]. For instance incoming lane WC_1 is connected to outgoing lane CE_1. Incoming lane WC_0 is connected to outgoing lane CE_0 and CS_0. This means that vehicles waiting on lane WC_0 can either turn right (CS_0) or go straight (CE_0). Lastly, incoming lane WC_2 connects to CE_2 (straight), CN_2 (left-turn) and CW_2 (U-turn). These lane to lane connections are called links.

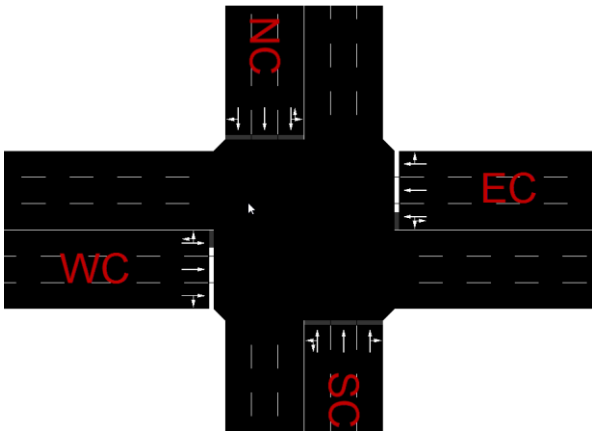


Figure 21: **Visualization of the network file in SUMO-GUI**

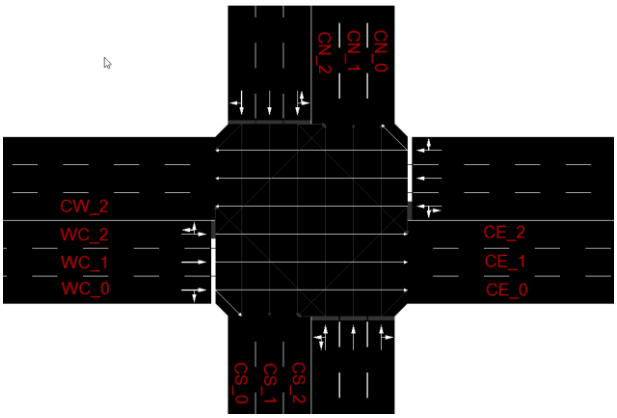


Figure 22: **Lane to lane connection (link) in the junction**

2.1.3 Example: 4-way priority junction with no U-turns

In example 2.1.2, NETCONVERT automatically computed the lane to lane connections in the junction based on heuristics. According to the default setting, U-turns are allowed and the rightmost lane of each link is aligned to the leftmost lane of the respective downstream link. What if you want to override this and for example prohibit U-turns in a junction or to only allow left-turn in a lane? In order to achieve this, you need to create a 'connection file'. This file describes how a node's incoming and outgoing edges are connected. You can specify connections on the edge level or you can declare it in detail which incoming lane shall be connected to which outgoing lanes. **Note that you can also find this example in VENTOS/examples/gettingStarted/3_no_Uturn folder.**

Assume left-most incoming lanes in the junction are used for left turns only and U-turn is not allowed. We can explicitly describe all possible connections as following. You can edit a file with a text editor of your choice and save this for instance as example.con.xml where .con.xml is the default suffix for SUMO connection files. All known extensions in SUMO are listed here.

```
<connections>
    <connection from="EC" to="CW" fromLane="0" toLane="0" />
    <connection from="EC" to="CN" fromLane="0" toLane="0" />
    <connection from="EC" to="CW" fromLane="1" toLane="1" />
    <connection from="EC" to="CS" fromLane="2" toLane="2" />

    <connection from="WC" to="CS" fromLane="0" toLane="0" />
    <connection from="WC" to="CE" fromLane="0" toLane="0" />
    <connection from="WC" to="CE" fromLane="1" toLane="1" />
```

[1] Click on the icon 'change color scheme' then go to junction and check 'show lane to lane connection'

```
    <connection from="WC" to="CN" fromLane="2" toLane="2" />

    <connection from="NC" to="CW" fromLane="0" toLane="0" />
    <connection from="NC" to="CS" fromLane="0" toLane="0" />
    <connection from="NC" to="CS" fromLane="1" toLane="1" />
    <connection from="NC" to="CE" fromLane="2" toLane="2" />

    <connection from="SC" to="CE" fromLane="0" toLane="0" />
    <connection from="SC" to="CN" fromLane="0" toLane="0" />
    <connection from="SC" to="CN" fromLane="1" toLane="1" />
    <connection from="SC" to="CW" fromLane="2" toLane="2" />
</connections>
```

Now that we have nodes, edges and connections files we can call NETCONVERT as before to create a network file.

```
netconvert --node-files=example.nod.xml --edge-files=example.edg.xml
--output-file=example.net.xml --connection-files=example.con.xml
```

You can open the generated SUMO network file using SUMO-GUI as shown in Figure 23. As you can see all of the left-most incoming lanes are left turn only. The connection tag accepts other attributes such as 'pass', 'keepClear', 'uncontrolled', etc. You can get more information from here.

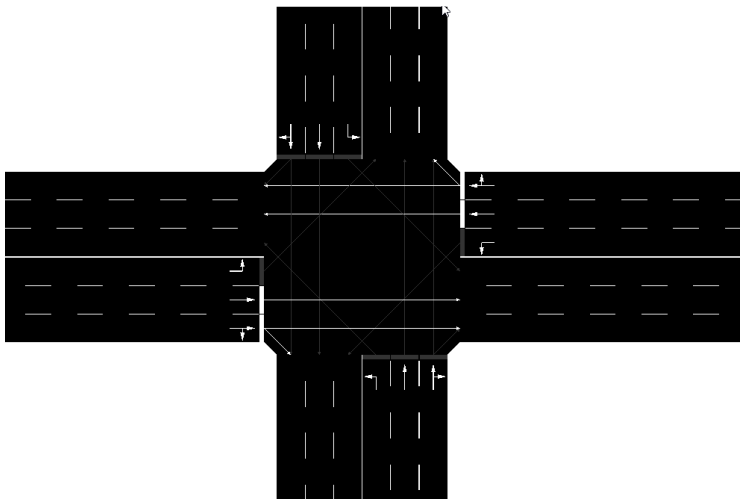


Figure 23: **Visualization of the network file in SUMO-GUI**

2.1.4 Example: 4-way signalized junction

This example is identical to Example 2.1.3, but we are going to replace the priority junction with a traffic light. You can find this example at **VENTOS/examples/gettingStarted/4_four_way_signalized_junction folder.** The new 'node file' looks like the following and we will keep using the old 'edge file' and 'connection file' as before.

```
<nodes>
    <node id="C" x="0.00" y="0.00" type="traffic_light" />
    <node id="E" x="100.00" y="0.00" />
    <node id="W" x="-100.00" y="0.00" />
```

```
    <node id="N" x="0.00" y="100.00" />
    <node id="S" x="0.00" y="-100.00" />
</nodes>
```

Now we can call NETCONVERT to create a network file.

```
netconvert --node-files=example.nod.xml --edge-files=example.edg.xml
--output-file=example.net.xml --connection-files=example.con.xml
```

You can open the generated SUMO network file using SUMO-GUI as shown in Figure 24. Node C is now controlled by a traffic light. NETCONVERT computes a default traffic light program for node C during generating of the network file as shown in Figure 25. Priority rules are used to avoid collisions if conflicting links have green light at the same time. The right-of-way rules are indicated by the colored bars at the end of each lane.

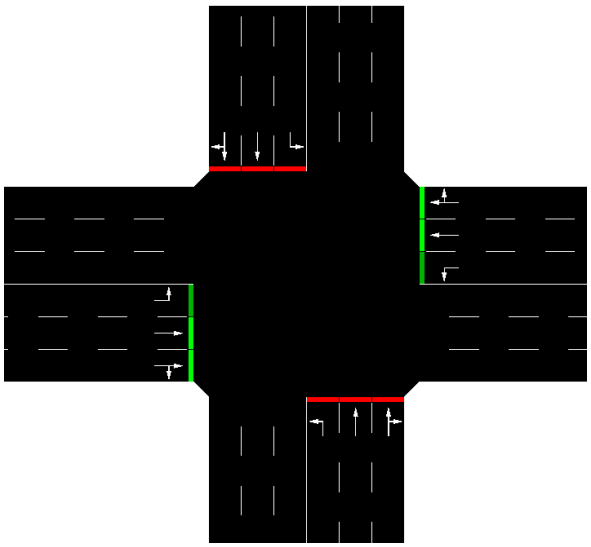


Figure 24: **Visualization of the network file in SUMO-GUI**

```
<tlLogic id="C" type="static" programID="0" offset="0">
    <phase duration="19" state="rrrrGGGgrrrrGGGg"/>
    <phase duration="10" state="rrrryyygrrrryyyg"/>
    <phase duration="6" state="rrrrrrrGrrrrrrrG"/>
    <phase duration="10" state="rrrrrrryrrrrrrry"/>
    <phase duration="19" state="GGGgrrrrGGGgrrrr"/>
    <phase duration="10" state="yyygrrrryyygrrrr"/>
    <phase duration="6" state="rrrGrrrrrrrGrrrr"/>
    <phase duration="10" state="rrryrrrrrrryrrrr"/>
</tlLogic>
```

Figure 25: **Generated (fix-time) traffic light program**

At the current phase, north and south directions are getting red light and must wait. Left turns in east and west directions are **permissive-only left turns**[1] (shown with dark green color bars) and vehicles need to yield to the opposing approaches before proceeding. The other lanes in east and west directions have right of way (shown with light green color bar) and vehicles can pass the junction without stopping.

A traffic light program (as shown in Figure 25) defines the phases of a traffic light at node C. “id” connects a traffic light program to a node and should be identical to the node id. You can define more than one traffic light programs for a node and each should have a unique “programID”[2]. You can switch between programs via WAUT or via the GUI context menu. “offset” defines the initial time offset of the program. “type” specifies the traffic light type and could be **static** or **actuated**.

[1] "Permissive-only" phasing allows two opposing approaches to time concurrently, with left turns allowed after yielding to conflicting traffic and pedestrians.

[2] The generated traffic light program by NETCONVERT usually has programID of "0". Also note that “off” is reserved and switches the traffic light off.

- Static traffic light selects fix-time traffic signal control (also known as pre-timed control) where order and duration of all traffic phases are fixed between cycles.

- Actuated traffic light selects traffic-actuated signal control where green duration can be extended in order to allow more vehicles to pass the intersection. Green extensions is done depending on the traffic measurements obtained from automatically-added induction loops. You can get more information about actuated traffic lights in SUMO from here.

Here we only focus on the fix-time traffic signal control. Each phase[1] in a fix-time traffic signal needs to be defined precisely via the 'phase' tag as shown in Figure 25. 'phase' tag in fix-time control consists of two attributes: **state** and **duration**.

"state" consists of a sequence of characters and each describes the state of a link. Note that in a junction, each direction has multiple incoming edges, each incoming edge has multiple lanes and each lane has one or more outgoing links[2]. In order to see the link numbers and lane to lane connection in SUMO-GUI, click on the icon 'change color scheme' then go to junction and check 'show link tls index' and 'show lane to lane connection'. As you can see in Figure 26, the junction has 16 links that are numbered from 0 to 15. The North direction has three incoming edges. Left-most edge (right-turn or through traffic) has two links (#0 and #1). The middle edge has only one link (#2) and the right-most edge has also one link (#3).

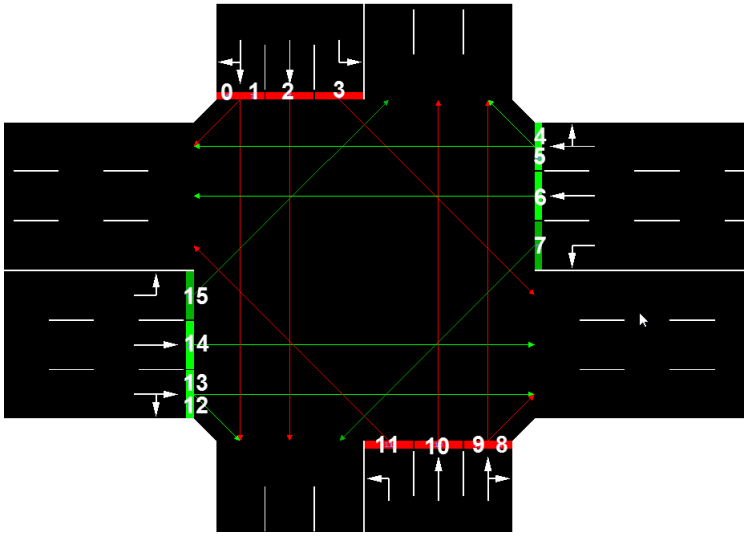


Figure 26: **Links in an intersection**

The junction has 16 links, thus "state" consists of 16 characters. All allowed characters are shown in Table *3*. The current state of the traffic light in Figure 26 can be described as `state="rrrrGGGgrrrrGGGg"`. The left-most character 'r' encodes the red light for link #0, followed by red for link #1 and so on. The duration defines the duration of the phase.

[1] SUMO is not using the correct technical term here. Interval is the correct term and not phase.
[2] This means that a signal does not control lanes, but links.

Table 3: **Possible characters to describe the link's state**

Character	Description
r	'red light' for a signal - vehicles must stop
y	'amber (yellow) light' for a signal - vehicles will start to decelerate if far away from the junction, otherwise they pass
g	'green light' for a signal, no priority - vehicles may pass the junction if no vehicle uses a higher prioterize foe stream, otherwise they decelerate for letting it pass
G	'green light' for a signal, priority - vehicles may pass the junction
u	'red+yellow light' for a signal, may be used to indicate upcoming green phase but vehicles may not drive yet (shown as orange in the gui)
o	'off - blinking' signal is switched off, blinking light indicates vehicles have to yield
O	'off - no signal' signal is switched off, vehicles have the right of way

You can visually check the traffic light program by right-clicking on a colored-bar and select 'show phases'. A window similar to Figure 27 will appear. Y-axis shows the link number and X-axis shows the simulation time. For example at time 0, link 0, 1, 2, 3, 8, 9, 10 and 11 are all getting red light and link 4, 5, 6, 7, 12, 13, 14 and 15 are getting green light.

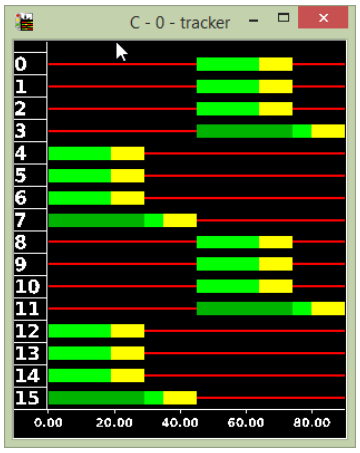


Figure 27: **Visualizing the traffic light program**

Loading a New Program

As discussed before, NETCONVERT generates traffic light programs for junctions during the computation of the networks. These computed programs quite often differ from those found in reality. You can define your own traffic light program and load it as a part of an additional-file into SUMO or SUMO-GUI. Let's define a new fix-time traffic light program that uses protected-only left turns as below. Duration of yellow interval is 4s and duration of red interval is 2s.

```
<additional>
    <tlLogic id="C" type="static" programID="my_prog" offset="0">
        <phase duration="20" state="rrrGrrrrrrrGrrrr"/>
        <phase duration="4" state="rrryrrrrrrryrrrr"/>
        <phase duration="2" state="rrrrrrrrrrrrrrr"/>

        <phase duration="20" state="rGGrrrrrrGGrrrrr"/>
```

```
        <phase duration="4" state="ryyrrrrrryyrrrrr"/>
        <phase duration="2" state="rrrrrrrrrrrrrrr"/>

        <phase duration="20" state="rrrrrrrGrrrrrrrG"/>
        <phase duration="4" state="rrrrrrryrrrrrrry"/>
        <phase duration="2" state="rrrrrrrrrrrrrrr"/>

        <phase duration="20" state="rrrrrGGrrrrrrGGr"/>
        <phase duration="4" state="rrrrryyrrrrrryyr"/>
        <phase duration="2" state="rrrrrrrrrrrrrrr"/>
    </tlLogic>
</additional>
```

You can edit a file with a text editor of your choice and save the above code for instance as tls.add.xml where .add.xml is the default suffix for SUMO additional files. After having defined a traffic light program, it can be loaded as an additional-file like the following (more on this later in Section 6).

```
sumo --net-file=example.net.xml --route-files=example.rou.xml --additional-
file=tls.add.xml
```

When the simulation starts running, two traffic light programs will be loaded. 1) The program with id "0" that is automatically computed by NETCONVERT and is stored in the example.net.xml file. 2) The new program with id "my_prog" that we manually defined. When more than one traffic light program is loaded for a single node, then the last loaded program will be used. The traffic light program in additional file will be loaded after the network file (as described in Section 6.1), thus "my_prog" will be used.

2.1.5 Example: Two-way street with pedestrian crossing

In this example we are going to build a two-way street with sidewalks and a pedestrian crossing. Figure 28 shows how the SUMO network will look like. Pedestrian crossing is a place designed for pedestrians to safely cross a road. Sidewalk is used for the pedestrian path beside a road. The area that connects sidewalks with crossings is modeled by special lanes of the type walking area. **You can find this example in VENTOS/examples/gettingStarted/5_pedestrian_crossing folder.**

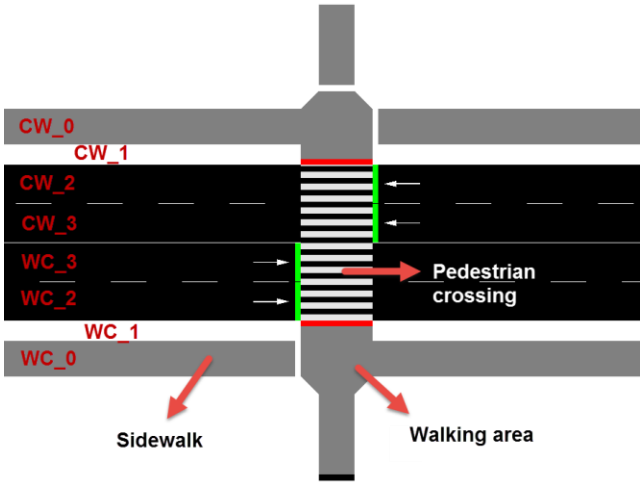


Figure 28: **Two-way street with sidewalks and crossing**

The new ‘node file’ looks like the following which is very similar to the previous node files except the N and S nodes are closer to node C.

```
<nodes>
    <node id="C" x="0.00" y="0.00" type="traffic_light" />
    <node id="E" x="100.00" y="0.00" />
    <node id="W" x="-100.00" y="0.00" />
    <node id="N" x="0.00" y="20.00" />
    <node id="S" x="0.00" y="-20.00" />
</nodes>
```

The new ‘edge file’ looks like the following. Each edge consists of four lanes and we have defined the characteristics of each lane separately. Using the ‘allow’ and ‘disallow’ attributes you can explicitly allow or disallow vehicle classes that are given in a comma separated list. For example consider edge CW. The right-most lane (#0) which serves as our sidewalk only allows class ‘pedestrian’ and has width of 3 meters[1]. Lane #1 disallows all vehicle and pedestrian classes and serves as a divider. Lane #2 and #3 allow all vehicle classes except pedestrians.

```
<edges>
    <edge id="EC" from="E" to="C" priority="2" numLanes="4" speed="30">
        <lane index="0" allow="pedestrian" width="3"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" disallow="pedestrian"/>
        <lane index="3" disallow="pedestrian"/>
    </edge>

    <edge id="CW" from="C" to="W" priority="2" numLanes="4" speed="30">
        <lane index="0" allow="pedestrian" width="3"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" disallow="pedestrian"/>
        <lane index="3" disallow="pedestrian"/>
    </edge>

    <edge id="WC" from="W" to="C" priority="2" numLanes="4" speed="30">
        <lane index="0" allow="pedestrian" width="3"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" disallow="pedestrian"/>
        <lane index="3" disallow="pedestrian"/>
    </edge>

    <edge id="CE" from="C" to="E" priority="2" numLanes="4" speed="30">
        <lane index="0" allow="pedestrian" width="3"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" disallow="pedestrian" />
        <lane index="3" disallow="pedestrian"/>
    </edge>

    <edge id="NC" from="N" to="C" priority="1" numLanes="1" speed="5.00"
spreadType="center" width="3.00" allow="pedestrian"/>
    <edge id="CS" from="C" to="S" priority="1" numLanes="1" speed="5.00"
spreadType="center" width="3.00" allow="pedestrian"/>
```

[1] A sidewalks is a lane which allows only the sumo vClass pedestrian. In this case it is important to disallow pedestrians on all other lanes.

```
</edges>
```

Lastly, the ‘connection file’ looks like the following:

```
<connections>
    <connection from="EC" to="CW" fromLane="2" toLane="2"/>
    <connection from="EC" to="CW" fromLane="3" toLane="3"/>

    <connection from="WC" to="CE" fromLane="2" toLane="2"/>
    <connection from="WC" to="CE" fromLane="3" toLane="3"/>

    <crossing node="C" edges="CE EC" priority="1" width="6.00"/>
</connections>
```

The pedestrian crossing is defined in the ‘connection file’ using the element ‘crossing’. Note that priority is a boolean value and specifies if the pedestrians have priority over the vehicles (pedestrians are free to walk without considerations for vehicular traffic). This can be used to model the behavior at controlled intersections and at marked pedestrian crossings. At uncontrolled intersections pedestrians may only cross when there is a suitable gap in road traffic.

Now that we have ‘node file’, ‘edge file’ and ‘connection file’ ready we can call NETCONVERT to generate the network file similar to Figure 28.

```
netconvert --node-files=example.nod.xml --edge-files=example.edg.xml
--output-file=example.net.xml --connection-files=example.con.xml
```

2.1.6 Example: 4-way signalized junction with pedestrian crossings and bike lanes

Previous intersections only support motor-vehicles (cars, buses, trucks, motor-cycles, etc.). In this section we are going to generate a junction that supports multi-modal traffic, namely motor-vehicles, bikes and pedestrians. For bikes, we need to add bike lanes and for pedestrians, we need to add sidewalks, crossings and walking areas. Figure 29 shows how the SUMO network will look like.

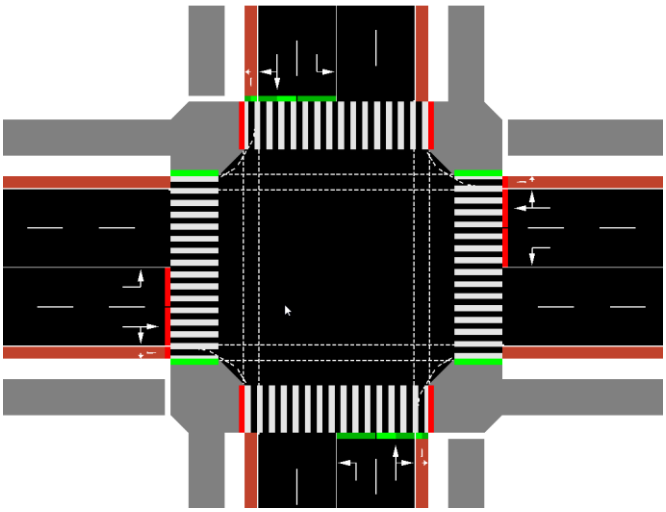


Figure 29: **4-way signalized junction that supports multi-modal traffic**

You can find this example in VENTOS/examples/gettingStarted/6_multimodal_signalized_junction folder. The 'node file' will be identical to Example 2.1.4, but we need to modify the 'edge file' and 'connection file'. The new 'edge file' looks like the following. Each edge consists of five lanes and we have defined the characteristics of each lane separately. Lane-specific definition is very similar to Example 2.1.5, and we have added an extra lane (#2) for bikes that only allows class 'bicycle'.

```
<edges>
    <edge id="CE" from="C" to="E" priority="2" numLanes="5" speed="30">
        <lane index="0" allow="pedestrian" width="3"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" allow="bicycle" width="1.00"/>
        <lane index="3" disallow="bicycle pedestrian"/>
        <lane index="4" disallow="bicycle pedestrian"/>
    </edge>

    <edge id="CN" from="C" to="N" priority="2" numLanes="5" speed="30">
        <lane index="0" allow="pedestrian" width="3.00"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" allow="bicycle" width="1.00"/>
        <lane index="3" disallow="bicycle pedestrian"/>
        <lane index="4" disallow="bicycle pedestrian"/>
    </edge>

    <edge id="CS" from="C" to="S" priority="2" numLanes="5" speed="30">
        <lane index="0" allow="pedestrian" width="3.00"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" allow="bicycle" width="1.00"/>
        <lane index="3" disallow="bicycle pedestrian"/>
        <lane index="4" disallow="bicycle pedestrian"/>
    </edge>

    <edge id="CW" from="C" to="W" priority="2" numLanes="5" speed="30">
        <lane index="0" allow="pedestrian" width="3.00"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" allow="bicycle" width="1.00"/>
        <lane index="3" disallow="bicycle pedestrian"/>
        <lane index="4" disallow="bicycle pedestrian"/>
    </edge>

    <edge id="EC" from="E" to="C" priority="2" numLanes="5" speed="30">
        <lane index="0" allow="pedestrian" width="3.00"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" allow="bicycle" width="1.00"/>
        <lane index="3" disallow="bicycle pedestrian"/>
        <lane index="4" disallow="bicycle pedestrian"/>
    </edge>

    <edge id="NC" from="N" to="C" priority="2" numLanes="5" speed="30">
        <lane index="0" allow="pedestrian" width="3.00"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" allow="bicycle" width="1.00"/>
        <lane index="3" disallow="bicycle pedestrian"/>
        <lane index="4" disallow="bicycle pedestrian"/>
    </edge>

    <edge id="SC" from="S" to="C" priority="2" numLanes="5" speed="30">
```

```
        <lane index="0" allow="pedestrian" width="3.00"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" allow="bicycle" width="1.00"/>
        <lane index="3" disallow="bicycle pedestrian"/>
        <lane index="4" disallow="bicycle pedestrian"/>
    </edge>

    <edge id="WC" from="W" to="C" priority="2" numLanes="5" speed="30">
        <lane index="0" allow="pedestrian" width="3.00"/>
        <lane index="1" disallow="all" width="1.50"/>
        <lane index="2" allow="bicycle" width="1.00"/>
        <lane index="3" disallow="bicycle pedestrian"/>
        <lane index="4" disallow="bicycle pedestrian"/>
    </edge>
</edges>
```

The new 'connection file' looks like the following. Note that we have defined 4 crossings in each direction using the element 'crossing'.

```
<connections>
    <connection from="CE" to="EC" fromLane="4" toLane="4" />

    <connection from="CS" to="SC" fromLane="4" toLane="4" />

    <connection from="EC" to="CN" fromLane="2" toLane="2" />
    <connection from="EC" to="CW" fromLane="2" toLane="2" />
    <connection from="EC" to="CN" fromLane="3" toLane="3" />
    <connection from="EC" to="CW" fromLane="3" toLane="3" />
    <connection from="EC" to="CS" fromLane="4" toLane="4" />

    <connection from="NC" to="CW" fromLane="2" toLane="2" />
    <connection from="NC" to="CS" fromLane="2" toLane="2" />
    <connection from="NC" to="CW" fromLane="3" toLane="3" />
    <connection from="NC" to="CS" fromLane="3" toLane="3" />
    <connection from="NC" to="CE" fromLane="4" toLane="4" />

    <connection from="SC" to="CE" fromLane="2" toLane="2" />
    <connection from="SC" to="CN" fromLane="2" toLane="2" />
    <connection from="SC" to="CE" fromLane="3" toLane="3" />
    <connection from="SC" to="CN" fromLane="3" toLane="3" />
    <connection from="SC" to="CW" fromLane="4" toLane="4" />

    <connection from="WC" to="CS" fromLane="2" toLane="2" />
    <connection from="WC" to="CE" fromLane="2" toLane="2" />
    <connection from="WC" to="CS" fromLane="3" toLane="3" />
    <connection from="WC" to="CE" fromLane="3" toLane="3" />
    <connection from="WC" to="CN" fromLane="4" toLane="4" />

    <crossing node="C" edges="CN NC" priority="1" />
    <crossing node="C" edges="CE EC" priority="1" />
    <crossing node="C" edges="CS SC" priority="1" />
    <crossing node="C" edges="CW WC" priority="1" />
</connections>
```

Now we can call NETCONVERT to generate the network file similar to Figure 29.

```
netconvert --node-files=example.nod.xml --edge-files=example.edg.xml
--output-file=example.net.xml --connection-files=example.con.xml
```

2.2 Method 2: Creating an abstract network

NETGENERATE allows to generate geometrically simple, abstract road maps. It supports grid-networks, spider-networks and random networks by passing one of the following switches: --grid-net, --spider-net or --random-net. Check here for more information.

2.3 Method 3: Importing Non-SUMO networks

NETCONVERT can imports digital road networks from different sources and generates road networks compatible with SUMO. It supports many external formats such as OpenSteetMap, VISUM, Vissim, etc. Check here for more information. At this section we will show you how to convert OpenSteetMap data into SUMO network file.

OpenSteetMap is a free editable map of the whole world. It is made by people like you. OpenSteetMap creates and provides free geographic data such as street maps to anyone who wants them. The project was started because most maps you think of as free actually have legal or technical restrictions on their use, holding back people from using them in creative, productive, or unexpected ways. The amount of data contained within OpenSteetMap, and their quality, is amazing. The quality of the road networks makes it worth to regard OpenSteetMap as a data source for traffic simulations.

This page describes different ways to obtain OpenSteetMap data. We will use the OpenSteetMap website to download a rectangular area. Click on the 'Export' button at the top of the page to open the export sidebar. Find the area you want and click on the blue 'Export' link[1]. This will allow you to export all the current raw OpenSteetMap data (nodes, ways, relations-keys and tags) in an .osm file with XML format. It does this by actually pointing your browser directly at the OpenSteetMap API to retrieve a bounding box of data (a "map call").

The next step is to import the OSM file into SUMO using NETCONVERT as following:

```
netconvert --osm-files map.osm -o map.net.xml
```

Several aspects of the imported network may have to be modified to suit your needs. The recommended NETCONVERT options are listed below. You can get more information from here.

```
--geometry.remove --roundabouts.guess --ramps.guess --junctions.join
--tls.guess-signals --tls.discard-simple --tls.join
```

A sample exported OSM map and the resulting SUMO network file are shown in Figure 30 and Figure 31 respectively. The red line in Figure 31 shows a river trail. OSM-data not only contains the road network but also a wide range of additional polygons such as building and rivers. These polygons can be imported using another application called POLYCONVERT with the following command:

[1] Note that if you select a big area, the blue 'Export' link will not appear until you select a smaller region.

```
polyconvert --net-file map.net.xml --osm-files map.osm -o poly.add.xml
```

The created polygon file, poly.add.xml, can then be added to a SUMO-GUI configuration as we will see later. The resulting network file with polygons is show in Figure 32. **You can find this example in VENTOS/examples/gettingStarted/openStreetMap folder.**

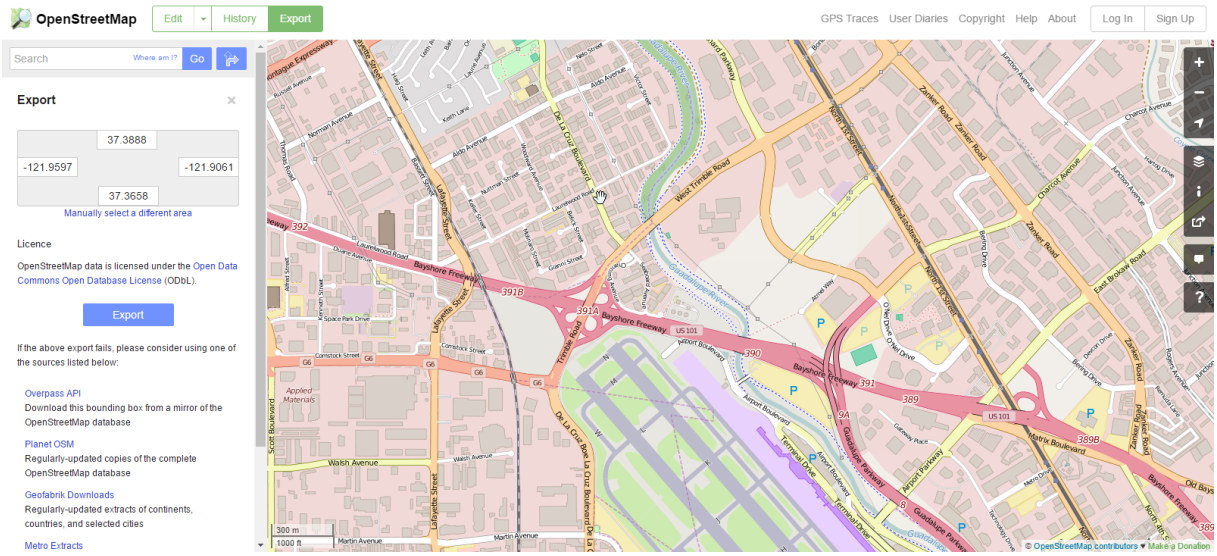


Figure 30: **Exported OSM file from OpenSteetMap website**

Figure 31: **Imported OSM file into SUMO using NETCONVERT**

Figure 32: **Adding polygon data into the network file**

2.4 Converting a SUMO Network File into Plain-XML Files

We have shown so far that a SUMO network file can be generated using NETCONVERT with a set of plain XML files including 'node file', 'edge file' and 'connection file'. Interestingly, the reverse is also true meaning that NETCONVERT can convert a SUMO network file into the plain XML files without information loss. For instance you can use the following command to convert the generated SUMO network file in Example 2.1.6 into its plain XML files.

```
netconvert --sumo-net-file=example.net.xml --plain-output-prefix plain
```

This will give you the files: plain.edg.xml, plain.nod.xml, plain.con.xml, plain.tll.xml

This is extremely useful in modifying your network. Often, importing non-SUMO networks leaves some aspect of deficient network quality. This manifests as unexpected/unrealistic traffic jams and teleporting vehicle errors. You will have to modify your network to add missing roads, prohibit some turns, correct the number of lanes and add/remove some traffic lights. Using the above method, you can convert your network to plain-xml files, modify them and then re-assemble the network.

3 Generating Traffic Demand File

In Section 2 we described three different methods for generating the SUMO network file. Now we need to define a traffic demand file that describes the vehicle type as well as the routes. The following shows a sample traffic demand file for Example 2.1.1. You can edit a file with a text editor of your choice and save this for instance as example.rou.xml where .rou.xml is the default suffix for SUMO traffic demand files. All

known extensions in SUMO are listed here. Note that the dashed-line box is not part of the code is used to show different logical sections of the file.

```
<routes>

  <!-- passenger vehicle type -->
  <vType id="passenger" length="5.0" minGap="2.0" maxSpeed="30.0" vClass="passenger" >
      <carFollowing-Krauss accel="3.0" decel="5.0" sigma="0" tau="1.64" />
  </vType>

  <!-- bus vehicle type -->
  <vType id="bus" length="10.0" minGap="4.0" maxSpeed="15.0" vClass="bus" >
      <carFollowing-Krauss accel="2.0" decel="3.0" sigma="0" tau="2.0" />
  </vType>

  <!-- route definition -->
  <route id="route0" edges="street1 street2" />

  <vehicle id="veh0" type="passenger" color="1,0,0" route="route0" depart="0"
departSpeed="0" departPos="0" departLane="first" />

  <vehicle id="veh1" type="passenger" color="1,0,0" route="route0" depart="8"
departSpeed="0" departPos="0" departLane="first" />

  <vehicle id="veh2" type="passenger" color="1,0,0" route="route0" depart="16"
departSpeed="0" departPos="0" departLane="first" />

</routes>
```

A traffic demand file consists of three sections:

1. **Vehicle type** that describes the vehicles physical properties such as length, color[1], maximum speed and minimum gap as well as car-following model and lane-changing model parameters. In the above example, two vehicle types called 'passenger' and 'bus' are defined. The length attribute describes the length of the vehicle itself and the minGap attribute describes the offset to the leading vehicle when standing in a jam as shown in Figure 33. Within the simulation, each vehicle needs 'length' + 'minGap' (when ignoring the safe gap), but only 'length' of the road should be marked as being occupied.

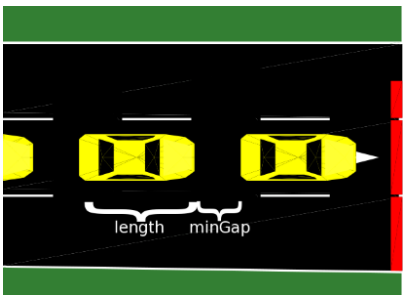


Figure 33: **Length and minGap for a vehicle**

[1] Color is defined in the normalized RGB space with three number (between 0 and 1) for red, green and blue respectively. Values are separated by comma and in the quotes with no space between the values. For example "1,0,0" represents the red color or "0,1,0" represents the green color.

A SUMO vehicle may be assigned to an "abstract vehicle class", defined by using the attribute vClass. They are "abstract" in the means that they are just names only (one could build a .5m long bus). These classes are used in lane definitions and allow/disallow the usage of lanes for certain vehicle types as we have seen in Example 2.1.5 and Example 2.1.6. All abstract vehicle classes are listed here.

2. **Vehicle route.** Route is defined as all the edges the vehicle will pass[1]. In the above example, only one route is defined called 'route0' that includes edges street1 and strret2. Note that a route must contain at least one edge and adjacent edges must be connected.

3. **Vehicle definition**. In the above example, SUMO builds three red vehicles of type 'passenger'[2] named 'veh0', 'veh1' and 'veh2' that all take route 'route0'. They depart at simulation time 0, 8 and 16 respectively with initial speed of 0 m/s from the beginning of starting edge and drive along street1 and street2. The 'departLane' determines the lane on which a vehicle shall be inserted (more info here).

Note 1

Vehicle definitions should be sorted by 'depart' time. In the above example, veh0 departs at time 0, then veh1 departs at time 8 and veh2 departs at time 16. If you swap the vehicle definition of veh0 and veh1, SUMO will give you the following warning:

Warning: Route file should be sorted by departure time, ignoring 'veh1'!

The reason behind sorted routes is that SUMO can simulate large road networks with up to millions of routes. Using a plain PC this is only possible if you do not keep all routes in memory. The simulation parameter --route-steps, which defaults to 200, defines the size of the time interval with which the simulation loads its routes. That means by default at startup only routes with departure time <200 are loaded, if all the vehicles have departed, the routes up to departure time 400 are loaded etc. This works only if the route file is sorted. This behavior may be disabled by specifying --route-steps 0 in which all routes will be loaded completely.

Note 2

Vehicles are inserted into the simulation at the time of their departure. You can add a random offset that is chosen uniformly from [0, TIME] to the departure time of each vehicle using --random-depart-offset switch. If there is insufficient space for inserting the vehicle at departure time, then the vehicle is put into an insertion queue and insertion is repeatedly attempted in subsequent simulation steps. You can use the --max-depart-delay <TIME> switch to discard vehicles after <TIME> seconds.

There are two methods for inserting vehicles into the simulation:

1. Each vehicle is checked separately for insertion on an edge (using the --eager-insert switch)

2. Insertion on an edge shall not be repeated in the same time-step once failed (using --sloppy-insert switch)

[1] It is up to the lane-changing model to determine the lane choice on multi-lane edges and perform speed adjustments related to lane changing (ref).
[2] It is not mandatory to define a vehicle type. If not given, a default type (DEFAULT_VEHTYPE) is used.

In an uncongested networks these methods behave similar but in a congested network with lots of vehicles which cannot be inserted variant 2) is much faster. Since version 0.18.0, variant 2) is the default behavior in SUMO.

3.1 Creating Vehicle Trip

As we have seen before, vehicle definition should contain a route (complete list of edges) that the vehicle traverses. Defining a **trip** allows us to specify only the origin/destination edges instead of a complete list of edges. In this case the simulation performs fastest-path routing based on the traffic conditions found in the network at the time of departure to compute the route. This is particularly useful in big networks that we only care about the origin/end edges of a vehicle and not the exact route. For example the following line defines 'veh0' of type 'passenger' with color yellow that departs at time 0 from edge '157751094#0' and ends the trip at edge '-8933185'.

```
<trip id="veh0" type="passenger" color="1,1,0" from="157751094#0" to="-8933185"
depart="0" />
```

Optionally, a list of intermediate edges can also be specified with the 'via' attribute.

```
<trip id="trip1" type="passenger" color="1,1,0" from="157751094#0" to="-8933185"
via="416214087" depart="0" />
```

3.2 Creating Vehicle Flow

A **flow** of vehicles consists of more than one vehicle that have the same starting and ending edge. Vehicles in a flow have the same vehicle type, but each has a unique id of the form "flowId.runningNumber". Each vehicle has a different depart time based on the flow type and can be defined using **either** of vehsPerHour, period, or probability tags.

The following line defines a flow of vehicles of type 'passenger' with color 'yellow' that all take route0. The flow id is 'flow1' and the vehicles' id are 'flow1.0', 'flow1.1', etc. The flow starts at time 0 and ends at time 100 with period of 10. This means that first vehicle departs at 0, second vehicle departs at 10, third vehicle departs at 20, fourth vehicle departs at 30 and so on.

```
<flow id="flow1" type="passenger" color="1,1,0" route="route0" begin="0" end= "100"
period="10" />
```

The following line defines a flow that departs vehicles with intensity of 300.5 veh/h.

```
<flow id="flow1" type="passenger" color="1,1,0" route="route0" begin="0" end= "100"
vehsPerHour="300.5" />
```

You can also define the probability for emitting a vehicle in each second using probability tag.

```
<flow id="flow1" type="passenger" color="1,1,0" route="route0" begin="0" end= "100"
probability="0.3" />
```

In order to limit the number of vehicles in a flow, you can remove the 'end' tag and use the 'number' tag instead as shown in the following.

```
<flow id="flow1" type="passenger" color="1,1,0" route="route0" begin="0" number="10"
probability="0.3" />
```

Flows with Incomplete Routes

Route information for a flow may also take the form of origin/destination edges (similar to a trip) instead of a complete list of edges. In this case the simulation performs fastest-path routing based on the traffic conditions found in the network at the time of departure/flow begin to compute the route.

```
<flow id="flow1" type="passenger" color="1,1,0" from="157751094#0" to="-8933185"
begin="0" number="10" probability="0.3" />
```

Optionally, a list of intermediate edges can also be specified with the 'via' attribute.

```
<flow id="flow1" type="passenger" color="1,1,0" from="157751094#0" to="-8933185"
via="416214087" begin="0" number="10" probability="0.3" />
```

3.3 Creating Traffic Routes

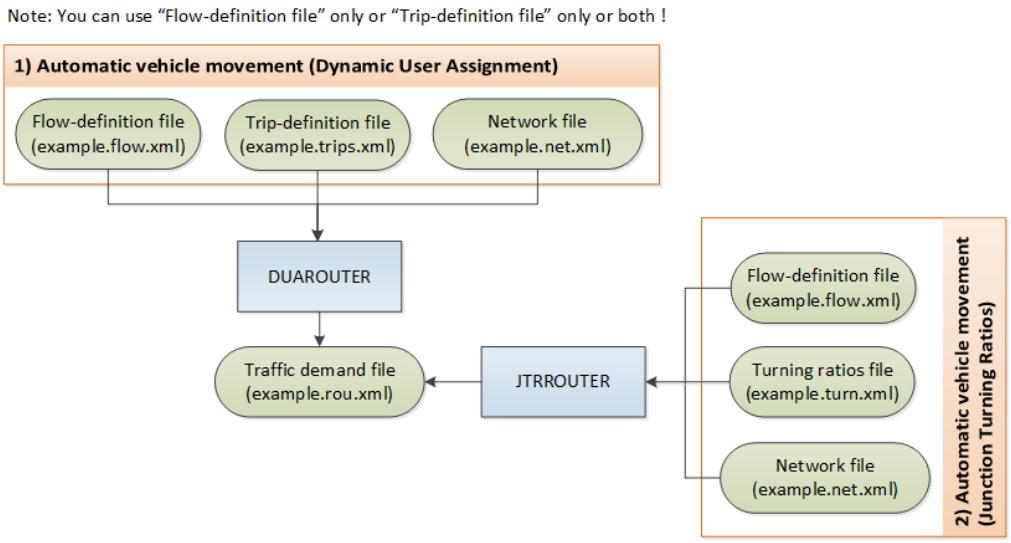


Figure 34: **Methods for generating SUMO traffic demand file**

Check here:
http://sumo.dlr.de/wiki/Demand/Introduction_to_demand_modelling_in_SUMO

http://sumo.dlr.de/wiki/Demand/Shortest_or_Optimal_Path_Routing

3.4 Adding Persons

Adding person into the junction
Adding person into sidewalk

```
<person id="ped4" type="pedestrian" depart="40">
    <walk from="EC" to="CW" departPos="-100" />
</person>
```

4 How SUMO Calculates the Next Vehicle's Speed

Let's consider a very long road with single-lane A with maximum speed of v_{max}^{A}. Vehicle V that has maximum speed of v_{max} and maximum acceleration of a_{max} is traveling on lane A. The vehicle's speed in the next time step, denoted by v_{next}, is calculated as following,

$$v_{next} = \min[v + a_{max} \times \Delta t, v_{max}] \quad (1)$$

Where v is the current speed and Δt is simulation time step. The vehicle's speed increases with a_{max} in each time step until it reaches v_{max}. The vehicle's next speed cannot exceed the lane's maximum speed, thus we have:

$$v_{next} \leftarrow \min[v_{next}, v_{max}^{A}] \quad (2)$$

By combining (1) and (2), we have:

$$v_{next} \leftarrow \min[min[v + a_{max} \times \Delta t, v_{max}], v_{max}^{A}] \quad (3)$$

Based on v_{next}, SUMO updates current acceleration (a) and current position (x) as following:

$$a_{next} = \frac{v_{next} - v}{\Delta t}, \; x_{next} = x + v_{next} \times \Delta t \quad (4)$$

Thus, in the next simulation step we have:

$$v \leftarrow v_{next} \quad (5)$$
$$a \leftarrow a_{next} \quad (6)$$
$$x \leftarrow x_{next} \quad (7)$$

Note that calculation of next position (x_{next}) does not depend on the current acceleration and SUMO refers to it as Euler update. A more accurate formula is ballistic update that yields more realistic results and is calculated as following. Ballistic update can be activated using option --step-method.ballistic.

$$x_{next} = x + \frac{1}{2} \times a \times \Delta t^2 + v_{next} \times \Delta t \quad (8)$$

In general, next speed calculation is complicated and is not as simple as equation (3). Many other factors can also bound the speed of a vehicle. One notable example is a car-following scenario that we will discuss in the next section.

4.1 Car-following Models in SUMO

Assume that vehicle V_2 is following vehicle V_1 on a single-lane of roadway. The car-following model (a.k.a longitudinal control model) determines the V_2's follow speed in relation to the vehicle ahead of it (V_1). Car following is of interest because it is relatively simple compared to other driving tasks, has been successfully described by mathematical models, and is an important facet of driving. Thus, understanding car following contributes significantly to an understanding of traffic flow [ref].

Let's consider a road with single-lane A with maximum speed of v_{max}^{A}. Assume that vehicle V is following vehicle V_l as shown in Figure 35. The following vehicle should adapt its speed to the leading vehicle in order to avoid collision.

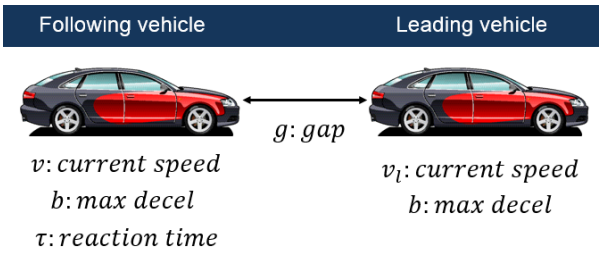


Figure 35: **Follower-leader example**

The speed of the following vehicle is equal to the following, where v_{next} is the next speed calculated from equation (3) and 'followSpeed' depends on the car-following model being used.

$$v_{next} \leftarrow \min[v_{next}, followSpeed] \quad (9)$$

The car-following models currently implemented in SUMO is listed in the following table. Note that not all models are fully functional. Moreover, each model uses its own set of parameters and you can find more information here. VENTOS has added four new models and are described in Part 3, Section 24.

Car-following Name	Attribute	Description
KRAUSS_ORIG1	KraussOrig1	The original Krauß-model which is a simplified version of Gipps model
KRAUSS	Krauss	The Krauß-model with some modifications
KRAUSE_PLUS_SLOPE	KraussPS	The default Krauss model with consideration of road slope
SMART_SK	SmartSK	Variant of the default Krauss model
PWAGNER2009	PWagner2009	Scalable model based on Krauß by Peter Wagner, using Todosiev's action points
IDM	IDM	The Intelligent Driver Model by Martin Treiber
IDMM	IDMM	Variant of the IDM model
WIEDEMANN	Wiedemann	The psycho-physical model by Wiedemann
DANIEL1	Daniel1	Car following model by Daniel Krajzewicz
BKERNER	BKerner	A model by Boris Kerner

You can select a car-following model when defining a vehicle type in the SUMO traffic demand file as described before in Section 3. For example the following lines define a vehicle type named 'passenger' with car-following model Krauss.

```
<vType id="passenger" length="5.0" minGap="2.0" maxSpeed="30.0" vClass="passenger" >
    <carFollowing-Krauss accel="3.0" decel="5.0" sigma="0" tau="1.64" />
</vType>
```

Note that the default car-following model in SUMO is **Krauss**. If you do not explicitly specify a car-following model for a vehicle type, then its car-following model is Krauss and its parameters are chosen based on

the vehicle's class. For example the following line defines a vehicle type named 'passenger' with car-following model Krauss and parameter accel=2.6, decel=4.5, sigma=0.5 and tau=1.

```
<vType id="passenger" length="5.0" minGap="2.0" maxSpeed="30.0" vClass="passenger" />
```

Follow Speed in Krauss_Orig1

Let's consider Krauss_Orig1 car-following model which is a simplified version of Gipps car-following model and is defined by Stefan Krauss in this paper. In this model, the followSpeed, denoted by v_{safe} is calculated as following:

$$v_{safe}(g, v_l) = -b\tau + \sqrt{(b\tau)^2 + v_l^2 + 2bg} \quad (10)$$

Where:

g: Space-gap between following and leading vehicles (m)
v_l: Current speed of the leading vehicle (m/s)
b: Maximum deceleration of the following/leading vehicle (m/s^2)
τ: Reaction time of the following vehicle (s)

Now let's see how v_{safe} is calculated. Consider the worst-case scenario where the leading vehicle breaks with maximum deceleration. The breaking distance that the leading vehicle needs to come to a complete stop is calculated as following:

$$\left.\begin{aligned} &\Delta x = -\frac{1}{2}bt^2 + v_0 t \quad (11) \\ &v_0 = v \quad (12) \\ &-b = \frac{0-v}{t} \rightarrow t = \frac{v}{b} \quad (13) \end{aligned}\right\} \Delta x_l = \frac{v_l^2}{2b} \quad (14)$$

The stopping distance for the following vehicle is:

$$\Delta x = v\tau + \frac{v^2}{2b} \quad (15)$$

Where the first term is the **reaction distance** and the second term is the **breaking distance**. The space gap, g, should be greater than or equal to $\Delta x - \Delta x_l$ in order to maintain a safe gap and prevent collision.

$$g \geq \Delta x - \Delta x_l = v\tau + \frac{v^2}{2b} - \frac{v_l^2}{2b} \quad (16)$$

That is equal to the following quadratic equation:

$$v^2 + (2b\tau)v - \left(v_l^2 + 2bg\right) \leq 0 \quad (17)$$

Solving the above equation for v would be:

$$0 < v \leq v_{safe} = -b\tau + \sqrt{(b\tau)^2 + v_l^2 + 2bg} \quad (18)$$

In other words, the maximum safe speed for the following vehicle denoted by v_{safe} is:

$$v_{safe} = -b\tau + \sqrt{(b\tau)^2 + v_l^2 + 2bg} \quad (19)$$

Substituting followSpeed (19) into (9) gives us:

$$v_{next} \leftarrow \min\left[v_{next}, -b\tau + \sqrt{(b\tau)^2 + v_l^2 + 2bg}\right] \quad (20)$$

Substituting v_{next} (3) into the above equation gives us:

$$v_{next} \leftarrow \min\left[\min[min[v + a_{max} \times \Delta t, v_{max}], v_{max}^A], -b\tau + \sqrt{(b\tau)^2 + v_l^2 + 2bg}\right] \quad (21)$$

The Krauss_Orig1 model assumes that a driver is not perfect in realizing the v_{next}. Instead, the chosen speed is smaller and is calculated as following, where 'sigma' denotes the driver's imperfection between 0 and 1 where 0 means a perfect driver.

$$v_{next} \leftarrow max[0,\ v_{next} - (sigma \times a_{max} \times Rnd\,[0,1)) \times \Delta t] \quad (22)$$

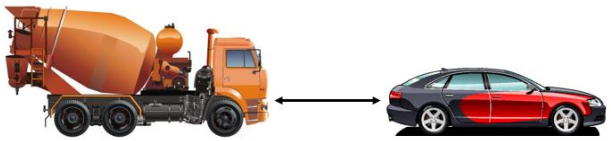
The following lines define a vehicle type 'Type1' with Krauss_orig1 car-following model:

```
<vType id="Type1" length="5.0" minGap="2.0" maxSpeed="28.0" vClass="passenger">
    <carFollowing-KraussOrig1 accel="2.0" decel="5.0" sigma="0" tau="1.64" />
</vType>
```

Attribute	Unit	Description
accel	m/s^2	Maximum acceleration
decel	m/s^2	Maximum deceleration (absolute value)
sigma	-	The driver imperfection [0,1]
tau	s	The driver reaction time

Follow Speed in Krauss

In previous section, we described how the follow speed (a.k.a safe speed) is calculated in the Krauss_Orig1 car-following model. If you have noticed, Krauss_Orig1 assumes that following and leading vehicles both have the same maximum deceleration. Imagine a heavy truck with deceleration of 2 is following a passenger car with deceleration of 5. The Krauss_Orig1 model running in the truck thinks that the leading vehicle has maximum deceleration of 2 which is wrong! If the leading vehicle suddenly breaks with maximum deceleration, then the truck will crash into it. As an example, you can have a look at the 'KraussOrig1' scenario in 'examples/carFollowing/KraussOrig1' folder.



$b = 2\ \frac{m}{s^2}$ $b = 5\ \frac{m}{s^2}$

The Krauss model solves this problem by considering the maximum deceleration in both vehicles. In this model, safe speed is calculated as following where b_f and b_l are maximum deceleration of following and leading vehicle respectively and $b = \min(b_f, b_l)$.

$$v_{safe} = -b\tau + \sqrt{(b\tau)^2 + \frac{b}{b_l}v_l^2 + 2b(g - G_{min})}$$

The v_{safe} is calculated as before. Consider the worst-case scenario where the leading vehicle breaks with maximum deceleration. The breaking distance that the leading vehicle needs to come to a complete stop is calculated as following:

$$\Delta x_l = \frac{v_l^2}{2b_l}$$

The stopping distance of the following vehicle is:

$\Delta x = v\tau + \frac{v^2}{2b}$ where $b = \min(b_f, b_l)$

The space gap, g, should be greater than or equal to $\Delta x - \Delta x_l + G_{min}$ in order to maintain a safe gap and prevent collision. Adding G_{min} is necessary to ensure an accident-free model in the presence of numerical errors arising from discretization. Solving this equation for v would give us the v_{safe} above.

$$g \geq (\Delta x - \Delta x_l) + G_{min} = v\tau + \frac{v^2}{2b} - \frac{v_l^2}{2b_l} + G_{min}$$

The following lines define a vehicle type 'Type1' with Krauss car-following model:

```
<vType id="Type1" length="5.0" minGap="2.0" maxSpeed="28.0" vClass="passenger">
    <carFollowing-Krauss accel="2.0" decel="5.0" sigma="0" tau="1.64" />
</vType>
```

Attribute	Unit	Description
accel	m/s^2	Maximum acceleration
decel	m/s^2	Maximum deceleration (absolute value)
sigma	-	The driver imperfection [0,1]
tau	s	The driver reaction time

4.2 Approaching to a Stop Point

A vehicle should decrease its speed when approaching a stop point such as a bus stop, dead-end street, signalized intersection, etc.

non-moving obstacle

5 XML Schema Resolution

Since version 0.20.0 sumo performs XML-validation on input files with schema information. This helps to detect common mistakes during manual preparation of XML input files such as spelling mistakes and attributes which should have been placed within another element.

The schema files can be found in the **VENTOS_SUMO/data/xsd** directory of your SUMO installation. If the environment variable SUMO_HOME is set, these files will be used when validating inputs. Otherwise, the files are loaded online from **http://sumo.dlr.de/xsd/SCHEMA_FILE** which may slow down the application (or fail if there is no internet connection).

Files that are written by one of the SUMO applications automatically receive the appropriate schema declaration. For example open the network file that was generated by NETCONVERT in any of the previous examples. You can see the schema declaration in the root element as following:

```
<net version="0.27" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/net_file.xsd">
```

When writing an input file from scratch the schema declaration must be added manually to the root element like above. The general format of schema declaration is as follows where ROOT_ELEMENT and SCHEMA_FILE should be set according to Table 4.

```
<ROOT_ELEMENT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/SCHEMA_FILE">
```

Table 4: **Schema declaration**

Application Option	ROOT_ELEMENT	SCHEMA_FILE
--route-files}, --trip-files, , --flow-files	routes	routes_file.xsd
--additional-files	add	additional_file.xsd
--node-files	nodes	nodes_file.xsd
--edge-files	edges	edges_file.xsd
--connection-files	connections	connections_file.xsd
--tllogic-files	tlLogics	tllogic_file.xsd
--type-files	types	types_file.xsd

Schema validation may slowdown XML parsing considerably and is therefore disabled for the network input by default (because networks should not be edited by hand and therefore be valid anyway). If you have large auto-generated inputs you may consider disabling schema validation altogether. This can be done using the option **--xml-validation never** or by deleting the schema information at the top of the XML input file(s). Note that when setting the validation option to **--xml-validation always**, it is an error to omit the schema declaration.

6 Running Microscopic Simulation

Once you generate the SUMO network file (as discussed in Section 2) and SUMO traffic demand file (as discussed in Section 3), you can start the microscopic simulation with SUMO or SUMO-GUI. SUMO-GUI provides a GUI for visualization and is very useful for debugging purposes. On the other hands, SUMO is

the command-line version that provides no visualization but is faster and is suitable for long-run and batch simulations.

6.1 SUMO (command-line)

SUMO needs at least two input parameters, one for network file and one for traffic demand file. You can run SUMO as following:

```
sumo --net-file=map.net.xml --route-files=map.rou.xml
```

You can also load one or more 'additional files' using the '--additional-file' switch option in order to add a wide range of network elements such as the followings:

- Infrastructure related: traffic light programs, induction loops and bus stops
- Additional visualization: POIs and polygons (rivers, buildings, etc.)
- Dynamic simulation control structures: variable speed signs and re-routers
- Demand related entities: vehicle types and routes

For instance to add the polygon file, the following command is used:

```
sumo --net-file=map.net.xml --route-files=map.rou.xml --additional-file=poly.add.xml
```

6.1.1 Order of File Parsing

SUMO accepts different input files as shown in previous section such as network file (--net-file), route file (--route-files) and additional file (--additional-file). To ensure the correct resolution of references, it is important to know what input file is loaded when. Whenever a simulation object A (such as a vehicle) defined in an input file refers to another simulation object B (such as a vehicle type), the corresponding object B must be defined before A either by being put into a file that is loaded earlier, or by being put into the same file on an earlier line. The order is as follows:

1. Network file is loaded
2. Additional files are loaded in the order in which they are given in the option
3. Route files are opened and the first n time-steps are read
4. Each n time-steps, the routes for the next n time steps are loaded

6.1.2 Defining Simulation 'Time Period' and 'Step Length'

Using the option --begin and --end, you can define the simulation time period as following:

```
sumo --net-file=map.net.xml --route-files=map.rou.xml --additional-file=poly.add.xml
--begin=0 --end=1000
```

The simulation starts at the time given in --begin, which defaults to 0. All vehicles with a departure time (depart) lower than the --begin time are discarded. The simulation performs each time step one-by-one. The simulation ends in the following cases:

- The final time step was given using --end and this time step was reached.

- No value for --end has been given and all vehicles have been simulated.

SUMO uses a simulation time step of 1s by default which means that in each simulation time step, one second of real time is simulated. You may override this using the --step-length <TIME> option. <TIME> is here given in seconds, but you may enter a real number.

Note

SUMO accepts many options that define the application's behavior and you can check them all here.

6.1.3 Configuration File

Because the list of options may get very long, configuration files were introduced. You can set up a configuration file which contains all the parameters you want to start the SUMO application with. A configuration file is an XML-file that has a root element named configuration. The options are written as element names, with their value being stored in the attribute 'value'.

In the above example, the configuration file would look like the following. You can use almost all SUMO options (input, output, time, processing, routing, report, etc.) defined here, but remember to remove the leading --.

```
<configuration>
    <input>
        <net-file value="map.net.xml" />
        <route-files value="map.rou.xml" />
        <additional-files value="poly.add.xml" />
    </input>

    <time>
        <begin value="0"/>
        <end value="1000"/>
        <step-length value="0.1"/>
    </time>
 </configuration>
```

Note that sections 'input' and 'time' have only documentation purposes and no functional meaning. You can edit a file with a text editor of your choice and save this for instance as map.sumocfg where .sumocfg is the default suffix for SUMO configuration files. All known extensions in SUMO are listed here. Now you can call SUMO as following. This means that instead of listing the parameters one by one in command line, we only give the name of the configuration file.

```
sumo --configuration-file map.sumocfg
```

In addition to a configuration file, further command line parameters can also be given on the command line. If a parameter is set within the named configuration file as well as given on the command line, the value given on the command line is used (overwrites the one within the configuration file). If you want to disable a Boolean option which was enabled in the configuration file, you need to give the "false" value on the command line explicitly, like --verbose false

6.2 SUMO-GUI

SUMO-GUI is basically the same application as SUMO, just extended by a graphical user interface. During the execution, each vehicular movement and the traffic progression can be observed and the possible bottlenecks can be visually identified. To open a simulation, click on File | Open Simulation or press Ctrl+O and select an existing SUMO configuration file such as map.sumocfg. Note that a configuration file is required in SUMO-GUI. SUMO reads the configuration file and loads all the necessary files. If the SUMO configuration file is erroneous, the errors are reported, otherwise your network referenced within the configuration file will be shown as in Figure 36.

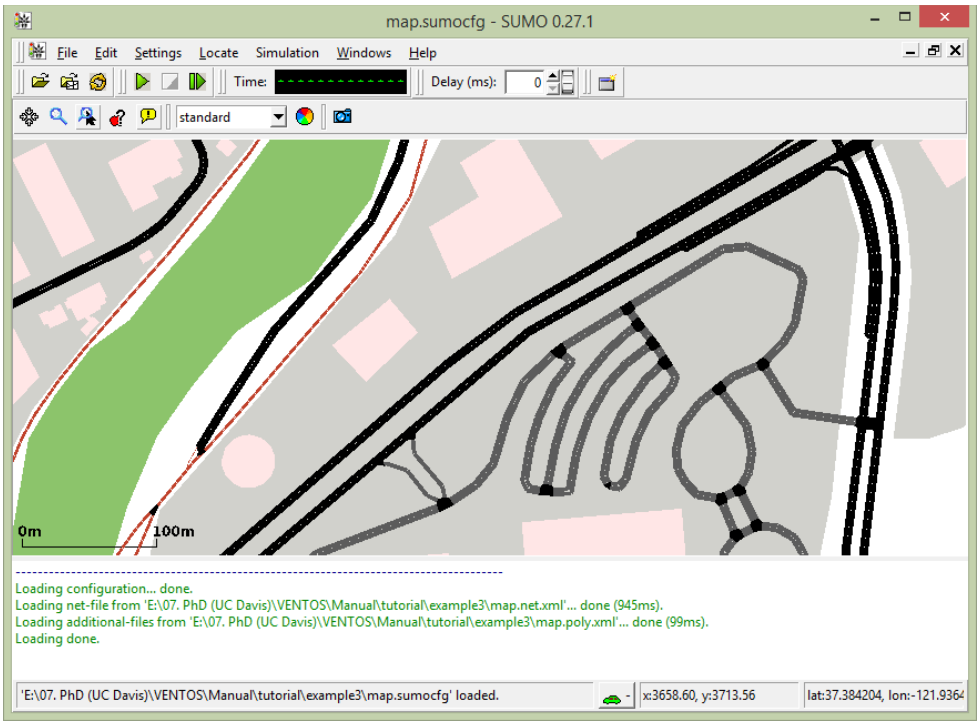


Figure 36: **SUMO-GUI window**

Navigation through the network is possible with the mouse only. One can drag the network with the left mouse button pressed into all directions and zoom either by using the mouse wheel or by pressing the right mouse button and moving the mouse up and down. For fine-grained zooming (half zoom speed) press the "Control" key while using the mouse wheel, for double-speed use "Shift". You can change the viewport by clicking on in order to change zoom and camera position (X, Y).

You can control the simulation using these controls:

Icon	Name	Shortcut	Description
	Play	Ctrl + A	Start/Resume the simulation
	Stop	Ctrl + S	Stop the simulation
	Single-step	Ctrl + D	Perform one step simulation

The current simulation second is shown in the "digital digits" field, right to "Time:". By clicking on the word "Time:", the display can be toggled between showing <seconds> and <hour:minute:seconds>. Next to the time display is the delay control. This allows you to slow down the simulation by waiting for the given number of milliseconds between simulation steps (by default the delay is set to 0. This can result in a simulation that runs too fast to see any vehicles. Increase the delay value if this happens).

Visualization setting allows you to change and customize the simulations' appearance and visualization. For customizing the simulation one can make changes e.g. to the background coloring, streets and vehicle appearance, etc. Click on the icon to access the visualization settings. Click on the Vehicles tab on the left and change 'Show As' from 'triangles' to 'raster images'. This will change the vehicle's icon from a triangle (Figure 37) to car image (Figure 38). To show the vehicle's id you can select 'Show vehicle name' as illustrated in Figure 39. You can choose the vehicle coloring schemes, edge/lane visualization setting, etc. as described in here.

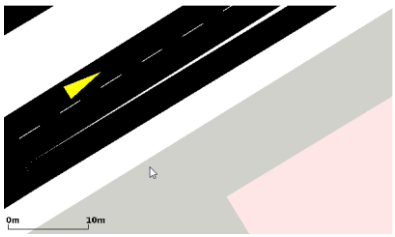


Figure 37: Default vehicle shape

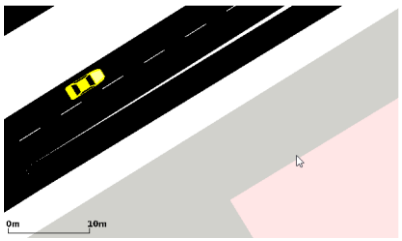


Figure 38: Raster image

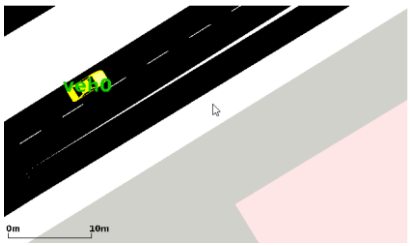


Figure 39: Showing vehicle id

You can save your modified visualization setting by pressing the save button at the top of the window[1]. Give it a name such as gui-settings.cfg and then update the SUMO configuration file as following. This allows SUMO to load the previously saved GUI setting on each simulation run.

```
<configuration>
    <input>
        <net-file value="map.net.xml" />
        <route-files value="map.rou.xml" />
        <additional-files value="poly.add.xml" />
        <gui-settings-file value="gui-settings.cfg"/>
    </input>

    <time>
        <begin value="0"/>
        <end value="1000"/>
        <step-length value="0.1"/>
    </time>
 </configuration>
```

You can open multiple views in the same simulation using the button. The views are called View #0, View #1 and so on. Figure 40 shows two different views from the same simulation. View #0 (on the left) shows veh0 driving on the left-most lane. View #1 (on the right) shows a zoom-out view with the nearby intersection that veh0 is approaching.

[1] The exported file does not contain 'Viewport', 'Delay' and 'Decals' settings by default. You need to enable each of the corresponding check-boxes in front of 'Export includes:' name.

The viewing windows can be managed using the Windows menu in the main menu bar. The visualization settings can be set independently for each view. When passing multiple files to the SUMO-option --gui-settings-file, one viewing window is opened for each file at the start of the simulation.

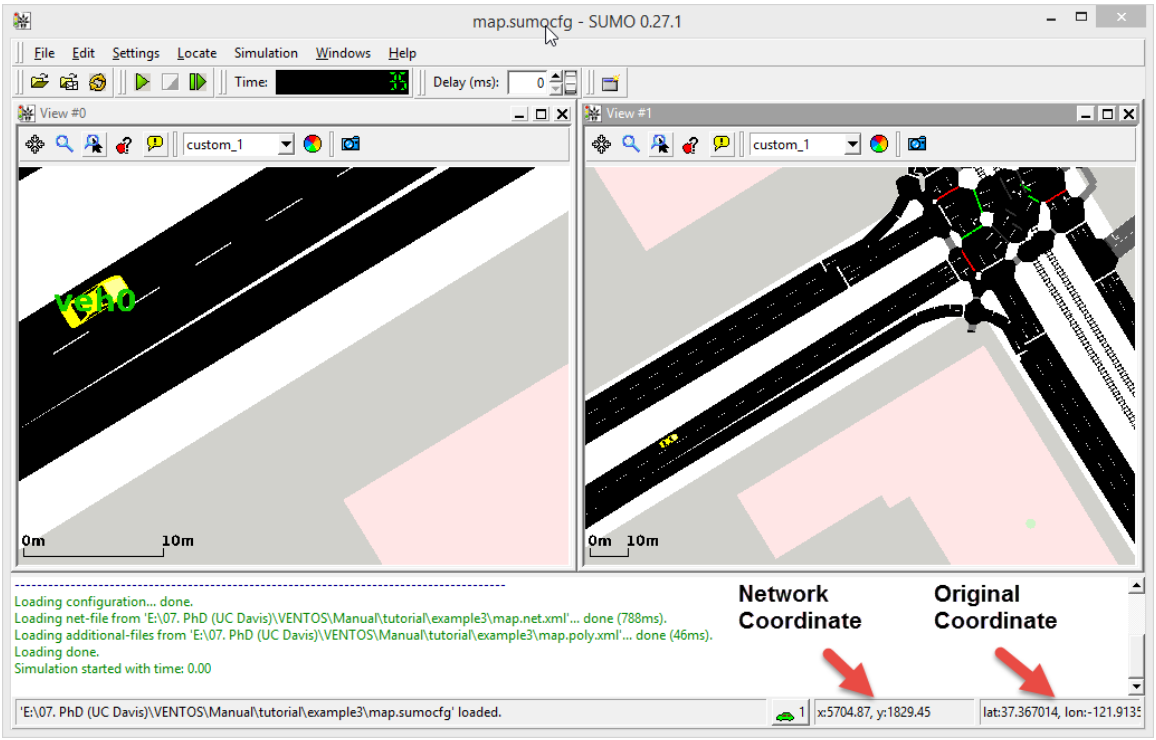


Figure 40: **Multiple views in the same simulation**

SUMO Coordinates

Note that networks in SUMO are using Cartesian, metric coordinates where the leftmost node is at x=0 and the node being most at the bottom is at y=0.

(If SUMO is compiled in debug mode, then they are called **SUMOD** and **SUMO-GUID**).

Check here:
http://sumo.dlr.de/wiki/Networks/SUMO_Road_Networks#Network_Format

7 SUMO Traffic Control Interface (TraCI)

Many scenarios need to dynamically change the behavior of microscopic traffic simulation. SUMO provides the TraCI (Traffic Control Interface) which allows an external application to retrieve information from the traffic simulation or to influence the simulation behavior. TraCI uses a TCP-based client/server architecture where SUMO acts as a TCP server and the external application is the TCP client. SUMO, when started in TraCI mode, only prepares the simulation and waits for an external application to take over the control.

SUMO provides and maintains the TraCI client library in C++ and Python. Other people have developed the TraCI client library in Java (TraCI4J), Matlab (TraCI4Matlab) and using SOAP (TraaS). Among all these libraries, the Python library is being tested and updated daily and supports all TraCI commands. Here we will provide a simple C++ program that uses the SUMO C++ TraCI client library. This program connects to SUMO at line 18 and asks SUMO to run for 5 simulation time steps at line 25. At the end, SUMO TraCI interface is closed in line 31.

```
1   #include <iostream>
2   #include <TraCIAPI.h>
3
4   // derive a class called Client from TraCIAPI
5   class Client : public TraCIAPI
6   {
7   public:
8       Client() {};
9       ~Client() {};
10  };
11
12  int main(int argc, char* argv[])
13  {
14      // create object client of class Client
15      Client client;
16
17      // connect to TraCI server in SUMO
18      client.connect("localhost", 1337);
19
20      // ask the current simulation time from SUMO
21      std::cout << "time in ms: " << client.simulation.getCurrentTime() << "\n";
22
23      // ask SUMO to run 5 time steps
24      std::cout << "run 5 steps ...\n";
25      client.simulationStep(5 * 1000);
26
27      // ask the current simulation time from SUMO again
28      std::cout << "time in ms: " << client.simulation.getCurrentTime() << "\n";
29
30      // disconnect client
31      client.close();
32  }
```

Save the above program in a file such as **traci_test.cpp** and put it in the VENTOS_SUMO directory. Then compile/build the program using the following command (in Linux). This will create an executable file traci_test, but do not run it yet!

```
g++ -o traci_test traci_test.cpp src/utils/traci/libtraci.a
src/foreign/tcpip/storage.o src/foreign/tcpip/socket.o -I src/utils/traci -I
src
```

Open a new terminal and run SUMO in TraCI mode by using the following command. You can use any of the previous SUMO configuration files. Note that passing the --remote-port 1337 switch starts SUMO in the TraCI mode by running a TCP server on port number 1337. This is shown in Figure 41.

```
sumo -c map.sumocfg --remote-port 1337
```

Now open another terminal and run traci_test as shown in Figure 42. Note that line 31 in the C++ code disconnects the client from SUMO and if the simulation is on-going then SUMO will show an error.

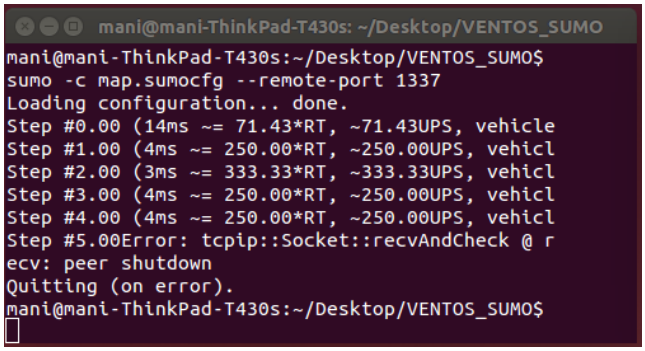


Figure 41: **Running sumo**

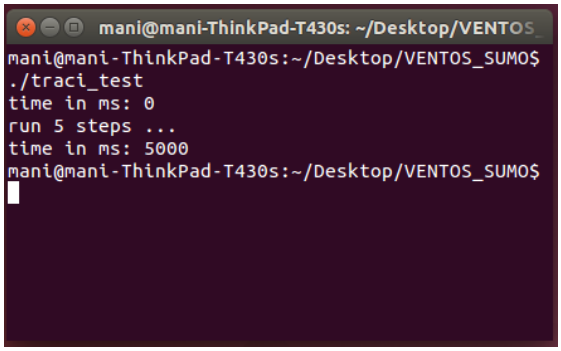


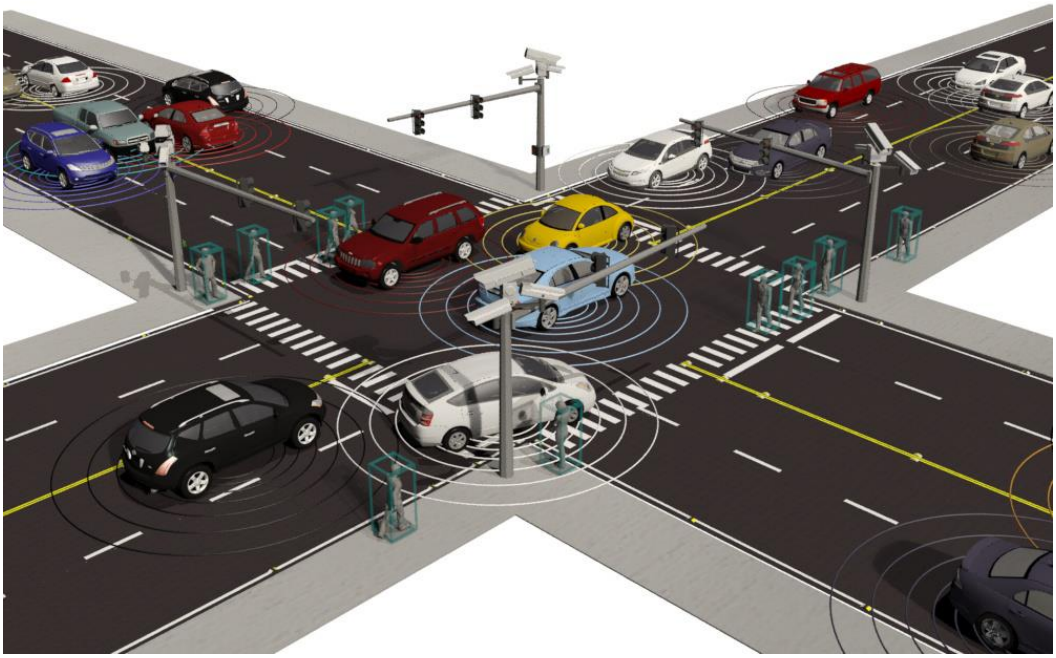
Figure 42: **Running traci_test**

While the simulation is running, many information can be retrieved, both static (e.g. the road network topology) and dynamic (e.g. position and speed of vehicles). All available TraCI commands are defined here and you can get the C++ API documentation from here. Note that using TraCI slows down the simulation speed. The amount of slow-down depends on many factors:

- Number of TraCI function calls per simulation step
- Types of TraCI functions being called (some being more expensive than others)
- Computation within the TraCI script/method
- Client language (C++ client is faster than Python in general)

VENTOS provides another implementation of C++ TraCI client library and allows you to develop more complex programs with less hassle. We will walk you through using VENTOS in Part 3.

Part 3: V2X Simulation with VENTOS



1 Getting Started with VENTOS

If you have not installed VENTOS on your machine yet, then follow the installation instructions from Part 1, Section 2 to do so. VENTOS folder is structured as following:

- **src:** contains all the C++ source codes
- **examples:** contains sample examples for you to run and get idea.
- **libs:** contains header-only libraries needed by VENTOS.
- **scripts:** contains scripts used for post-processing of simulation output

Each sub-folder in the example folder has the following structure:

- **results:** simulation output for this example are saved in this folder
- **sumocfg:** SUMO-related files to model roads, junctions, vehicles, etc.
- **omnetpp.ini:** configuration file that provides input data for simulation

- **addNode.xml:** (optional) file used by the AddNode module to add nodes into simulation
- **trafficControl.xml:** (optional) file used by the TrafficControl module to control node's behavior
- **gui.xml:** (optional) file used by the GUI module to control the viewport of SUMO GUI

The src folder has the following structure:

- **addNode:** contains the code that is used for adding nodes (vehicles, bikes, pedestrians)
- **baseAppl:** contains the base application layer common in all nodes
- **gettingStarted:** contains codes that help you get started with VENTOS
- **global:** contain the code that is being used globally in the project
- **logging:** contains the code that is responsible for logging all activities
- **loggingWindow:** standalone C++ application for showing logs in a window
- **MIXIM_veins:** contains the code for veins project that implements IEEE 802.11p
- **mobility:** contains the code responsible for moving the nodes in OMNET++
- **msg:** contains the code that defines message frame structure
- **nodes:** contain the code for vehicle, rsu, pedestrian, bicycle, etc.
- **router:** contains the code for dynamic routing of vehicles in the network
- **traci:** contains the code for Traffic Control Interface to be able to connect to SUMO
- **trafficlight:** contains the code for traffic signal control on intersections
- 'Network.ned' describes the network topology in OMNET++ NED format

Once you compile/build the VENTOS project, a shared library called libVENTOS.so is generated in the src folder. 'opp_run' later loads the shared library and starts the simulation (opp_run is part of OMNET++).

2 Building a V2X Simulation Scenario

A V2X simulation is built using the following steps:

1. Generating a road network consisting of junctions (nodes) and roads (edges) in SUMO. You can use plain-xml files or use abstract network or import non-SUMO networks such as Open Street Map. This is already covered in Part 2, Section 2.
2. Generating multi-modal DSRC-enabled traffic including motor vehicles, bikes, person, etc. You can define a 'SUMO traffic demand file' as discussed in Part 2, Section 3. VENTOS also provides the 'addNode' module to help you add different node types into the simulation easily as will be discussed in Section 3. The third option is to add/remove nodes dynamically during simulation through TraCI as discussed in Section 9.
3. (Optionally) placing one or more Road-Side Units (RSUs) on the road network using the addNode module as discussed in Section 3.2.
4. Write your own algorithm (Section 7 and 8) that uses the simulation data and can potentially affects the simulation behavior.
5. Run the simulation and collect data (Section 29) that is later used for post-processing.

You can also have a look at the 'examples' folder as a starting point or to get idea.

3 Adding Nodes to Simulation

VENTOS provides the 'addNode' module to help you add different node types into the simulation easily. You can add fix nodes such as RSUs, obstacles, etc. as well as mobile nodes such as motor vehicles, bikes and person. All nodes are defined in the 'addNode.xml' file located in the same folder as omnetpp.ini configuration file. The addNode module reads this file and decides which nodes should be inserted at the beginning of the simulation.

Note: 'addNode.xml' file is not a replacement for SUMO traffic demand file and you can still define nodes in that file. In fact, 'addNode.xml' complements the SUMO traffic demand file by allowing the user to add more node types. In some cases, defining the SUMO traffic demand file is necessary (to define vehicle type, route type, etc.).

Setting up an Example

We are using the example in Part 2, Section 2.1.6 to generate a 4-way signalized junction with pedestrian crossing and bike lane. Go to the 'VENTOS/examples/gettingStarted/6_multimodal_signalized_junction' folder and generate the SUMO network file using the following command (if you have not done it before). Make sure NETCONVERT is somewhere in your PATH.

```
netconvert --node-files=example.nod.xml --edge-files=example.edg.xml
--output-file=example.net.xml --connection-files=example.con.xml
```

Open the 'omnetpp.ini' located in VENTOS/examples/gettingStarted folder and copy/paste the following configuration:

```
include ../omnetpp_general.ini

[Config tutorial1]
description = "This config shows you how to add nodes into simulation"

Network.TraCI.active = true
Network.TraCI.SUMOapplication = "sumo-guiD"
Network.TraCI.SUMOconfig = "6_multimodal_signalized_junction/example.sumocfg"
Network.TraCI.terminateTime = 600s

Network.addNode.id = "add_0"
```

The TraCI module is responsible for establishing the TraCI interface and accepts many parameters. The 'active' parameter is set to 'true' to turn the module on. The 'SUMOapplication' parameter specifies which SUMO application needs to run: sumoD (command line) or sumo-guiD (visualization) as discussed in Part 2, Section 6. Note that you cannot use the official SUMO binaries since VENTOS has extended the TraCI commands, car-following models, etc. Using the official SUMO binaries will probably give you run-time error. 'SUMOconfig' specifies the relative path to the SUMO configuration file (.sumocfg) in the VENTOS project. Parameter 'terminateTime' specifies the maximum simulation time in seconds. The addNode module reads the 'addNode.xml' file looking for the addNode XML element with id "add_0".

In the menu bar, click on Run | Run Configuration and choose 'OMNET++ Simulation' and click on 'New launch configuration' button at the top left. Give this configuration a name like tutorial1. In Executable,

choose opp_run[1] and change 'Working dir' to '/VENTOS/examples/gettingStarted'. In 'Ini file(s)' click on Browse button and find 'omnetpp.ini'. From 'Config name' choose tutorial1 from the drop down list. Leave the rest of the options to default. Click Apply and then click Run.

OMNET++ Qtenv graphical runtime environment appears (you can find more information about Qtenv window in Section 30). Click on the red play button on the toolbar to start the simulation. The following warning message is shown since we have not added any nodes. The simulation keeps running until maximum simulation time (=600s as specifies using parameter 'terminateTime') is reached.

```
WARNING: Add node with id 'add_0' is empty!
```

In the following sections, we'll show you how to define different nodes in the 'addNode.xml' file.

3.1 Default Modules

Open 'src/addNode/modules.ned' and look for addNode module definition. Each node has its own default module type, name and display string in OMNET. For example, if you add a vehicle into the simulation, then VENTOS uses the 'vehicle' module type defined in the 'VENTOS/src/nodes/vehicle/modules' file, with the default module name of 'V' and a red rectangle as its geometric shape. You can also find the default module for bike, person, obstacle and RSU.

3.2 Adding Road-side Units (RSUs)

To define an RSU, use the "rsu" element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
id	string	"RSU0"	Yes
pos	double, double, double	"900,800,0"	Yes
drawMaxIntfDist	true/false	"false"	No (default: "true")
color	string	"red"	No (default: "green")
filled	true/false	"true"	No (default: "false")

The following line defines an RSU with name "my_RSU" at SUMO coordinate "100,100,0".

```
<rsu id="my_RSU" pos="100,100,0" />
```

As shown in Figure 43, RSU[0] is created in OMNET++ and it is also drawn in SUMO. The circle around the RSU shows the maximum interference distance (more on this in Section 15). You can turn it off by setting the 'drawMaxIntfDist' attribute to 'false'. You can also change the circle's color using the 'color' attribute or draw a filled circle by using the 'filled' attribute. You can find the RSU code in src/nodes/rsu folder.

Note: If you want to use an RSU to control an intersection, then you need to place the RSU as close as possible to the center of the intersection. Moreover, the RSU id should be identical to the intersection id. We will provide more details later.

[1] The VENTOS project is compiled as a shared library.

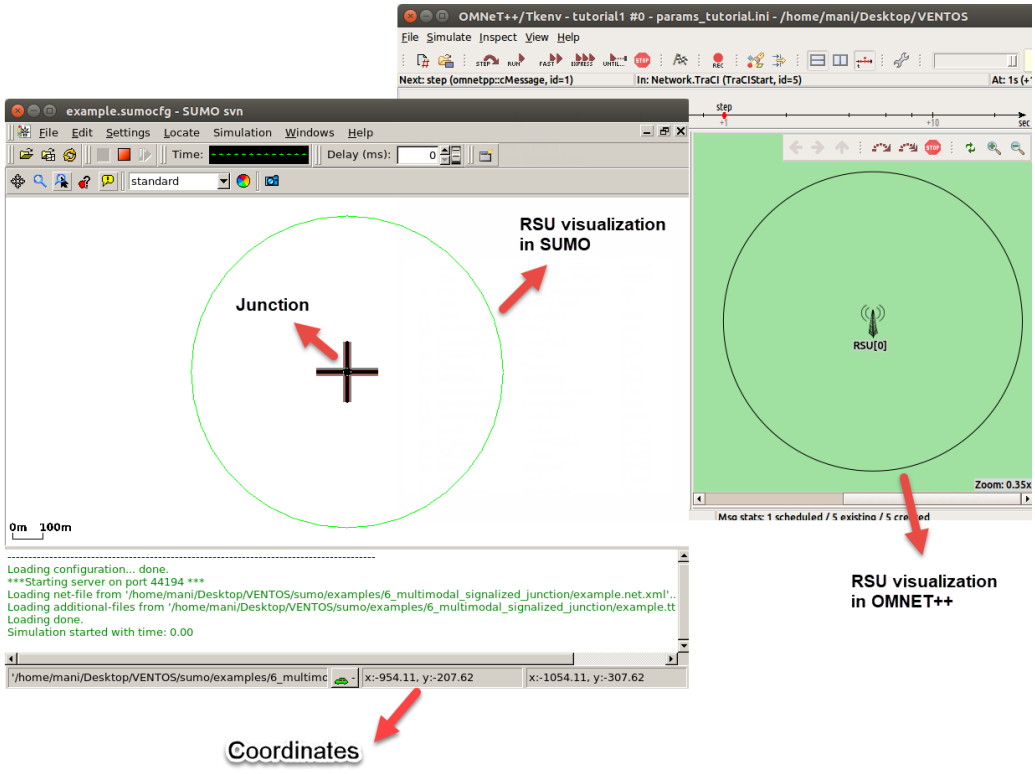


Figure 43: **Visualization of an RSU in OMNET++ and SUMO**

To place an RSU in the network, you need to find a x,y coordinate by opening your SUMO network file in the SUMO-GUI and then hovering the mouse cursor on a position in the road network. The SUMO x,y coordinates under the mouse cursor are shown in the SUMO-GUI status bar as shown in Figure 43.

You can place an RSU randomly in the network by using 'rnd' for x or y coordinates. As an example the following line chooses a random x and a random y for RSU named 'my_RSU'. The random coordinate is chosen uniformly from the network boundaries reported by SUMO.

```
<rsu id="my_RSU" pos="rnd,rnd,0" />
```

The following line places 'my_RSU' in x=100 and z=0, but a random y value. Note that the z coordinate does not support rnd.

```
<rsu id="my_RSU" pos="100,rnd,0" />
```

3.3 Adding Obstacles

Obstacles are non-moving objects that can block a roadway and are useful in simulating an accident. To define an obstacle, use the "obstacle" element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
id	string	"OBS0"	Yes
edge	string	"street_1"	Yes

lane	int	"1"	*
lanePos	double	"320"	Yes
onRoad	true/false	"false"	No (default: "true")
length	double (m)	"10"	No (default: "5.0")
color	string	"green"	No (default: "red")
begin	double (s)	"5"	No (default: "0")
end	double (s)	"20"	No (default: sim end)
duration	double (s)	"10"	No (default: "-1")
* See the text for more information.			

The following line defines an obstacle with name “my_OBS” at position 50 of lane 3 of edge WC as shown in Figure 44. Note that obstacles can also be DSRC-enabled.

```
<obstacle id="my_OBS" edge="WC" lane="3" lanePos="50" />
```

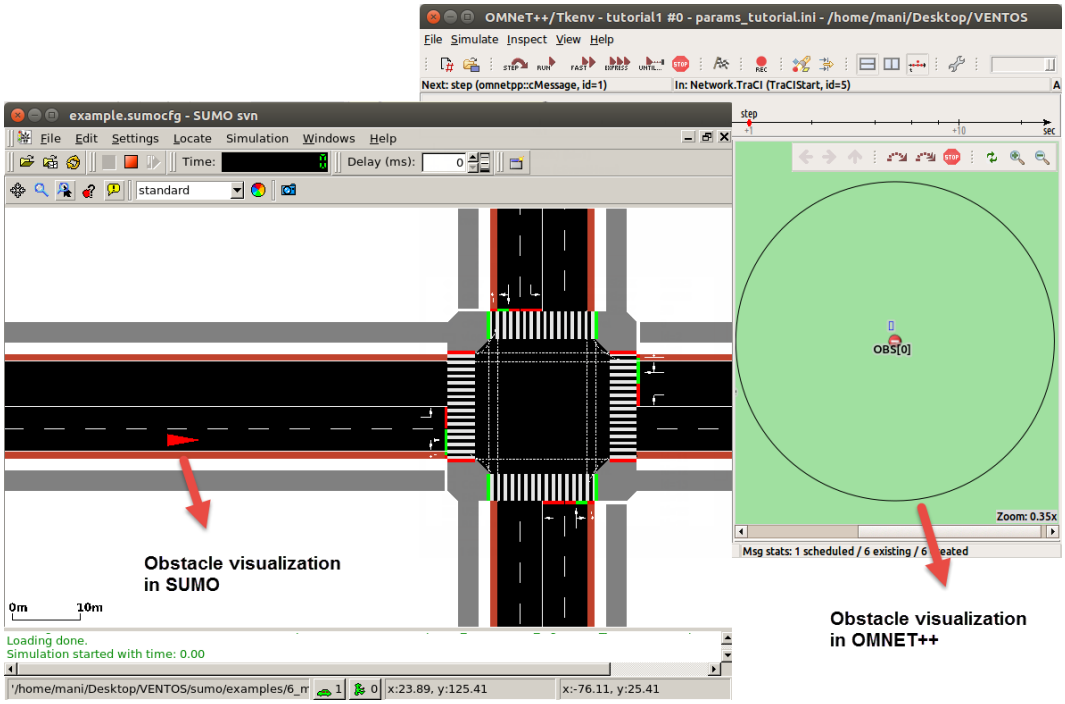


Figure 44: **Visualization of an obstacle in OMNET++ and SUMO**

You can set the color and length of the obstacle. You can also place the obstacle beside the road by setting the ‘onRoad’ attribute to ‘false’. As an example, we add two obstacles as shown in Figure 45. Obstacle ‘my_OBS’ is red and placed on the road whereas obstacle ‘my_OBS2’ is placed beside the road and is green. Both of these obstacles are non-moving and they can potentially send/receive V2X messages. The difference is that the former obstacle obstructs the vehicles flow, but the later doesn’t.

```
<obstacle id="my_OBS" edge="WC" lane="3" lanePos="50" />
<obstacle id="my_OBS2" edge="WC" lanePos="60" color="green" onRoad="false" />
```

The ‘begin’ attribute allows you to specify the time of insertion. The ‘end’ and ‘duration’ attributes allow you to remove the obstacle after a certain time. Note that you are not allowed to use ‘end’ and ‘duration’ together. You can find the obstacle’s application layer code in src/nodes/obstacle folder.

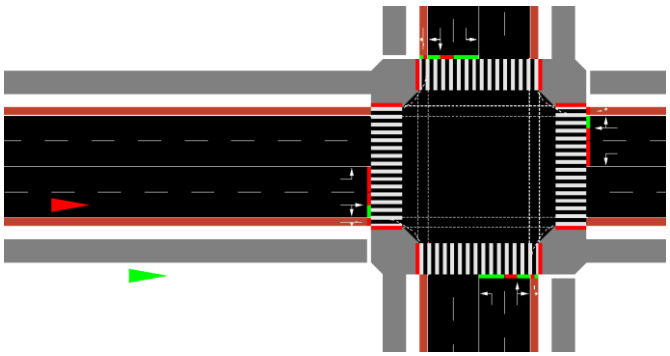


Figure 45: **Obstacles can be inserted on the road or beside the road**

3.4 Adding Motor-vehicles and Bikes

To insert a motor vehicle or bike, use the “vehicle” element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
id	string	"veh_2"	Yes
type	string	"TypeManual"	Yes
route	string	"route0"	*
from	string	"edge1"	*
to	string	"edge10"	*
via	string/string list	"edge1,edge2"	*
color	string	"green"	No (default: "yellow")
depart	double (s)	"30"	No (default: "0")
departLane	int	"320"	No (default: "-5")
departPos	double (m)	"10"	No (default: "0")
departSpeed	double (m/s)	"5"	No (default: "0")
laneChangeMode	int	"533"	No (default: "597")
status	stopped/parked	"parked"	No (default: "")
duration	double (s)	"10"	No (default: "-1")
DSRCprob	double	"0.5"	No (default: config)
* See the text for more information.			

You can control the vehicle’s departure using ‘depart’, ‘departLane’, ‘departPos’ and ‘departSpeed’ that is almost identical to SUMO-defined attributes described here. The ‘depart’ parameter is a real number that shows the time at which the vehicle enters the network. Note that if there is not enough space in the network, the actual depart time may be later. It also takes the following special values:

-1	**DEPART_TRIGGERED:** the departure is person triggered
-2	**DEPART_CONTAINER_TRIGGERED:** the departure is container triggered
-3	**DEPART_NOW:** discards the vehicle if it cannot be inserted immediately

The ‘departLane’ determines the lane index (starting with rightmost=0) that the vehicle is tried to be inserted. It also takes the following special values:

-2	**DEPART_LANE_RANDOM:** a random lane is chosen. Note that vehicle insertion is not retried if it could not be inserted
-3	**DEPART_LANE_FREE:** the most free (least occupied) lane is chosen
-4	**DEPART_LANE_ALLOWED_FREE:** the "free" lane of those that allow the vehicle’s class
-5	**DEPART_LANE_BEST_FREE:** the "free" lane of those that allow the vehicle the longest ride without the need to change lane
-6	**DEPART_LANE_FIRST_ALLOWED:** the rightmost lane the vehicle may use

The ‘departPos’ is a real number that determines the position on the chosen departure lane at which the vehicle is tried to be inserted. It also takes the following special values:

-2	**DEPART_POS_RANDOM:** a random position is chosen. Note that vehicle insertion is not retried if it could not be inserted
-3	**DEPART_POS_FREE:** a free position (if existing) is used
-4	**DEPART_POS_BASE:** the vehicle is tried to be inserted at the position that allows its back to be at the beginning of the lane (vehicle's front position = vehicle length)
-5	**DEPART_POS_LAST:** the vehicle is inserted with the given speed as close as possible behind the last vehicle on the lane. If the lane is empty it is inserted at the end of the lane instead
-6	**DEPART_POS_RANDOM_FREE:** at first, the "random" position is tried, then the "free", if the first one failed

The ‘departSpeed’ is a real number that determines the speed of the vehicle at insertion. It also takes the following special values:

-2	**DEPART_SPEED_RANDOM:** A random speed between 0 and MIN (vehicle's maximum velocity, lane's maximum velocity) is used
-3	**DEPART_SPEED_MAX:** The maximum velocity allowed for the vehicle on the chosen departure lane is used

Let’s see an example. The following line defines vehicle “veh_1” of type “passenger” and with route id “movement8”. Note that vehicle type and route id should already be defined in the SUMO traffic-demand file[1]. The rest of the attributes are set to default.

```
<vehicle id="veh_1" type="passenger" route="movement8" />
```

Now let’s define a second vehicle with id “veh_2” of type “passenger” and with route id “movement3”. This vehicle is green and departs at time 5s from position 6 of lane 3.

```
<vehicle id="veh_2" type="passenger" route="movement3" depart="5"
departPos="6" departLane="4" color="green" />
```

The route of veh_1 (yellow) and veh_2 (green) is shown in Figure 46.

[1] example.rou.xml

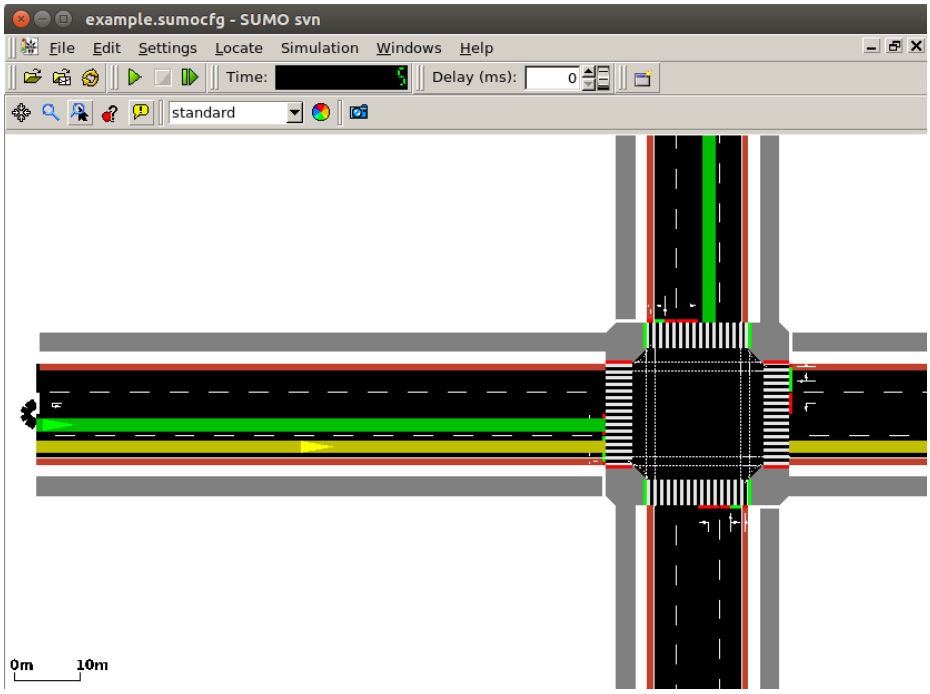


Figure 46: **Route of 'veh_1' (yellow) and 'veh_2' (green) at time 5s**

Vehicles with Incomplete Routes

As we have seen before, vehicle definition should contain a route id (complete list of edges) that the vehicle traverses. Using 'from'/'to' attributes allows us to specify only the origin/destination edges instead of a complete list of edges. In this case the simulation performs fastest-path routing based on the traffic conditions found in the network at the time of departure to compute the route. This is particularly useful in big networks that we only care about the origin/end edges of a vehicle and not the exact route. For example the following line defines 'veh_3' of type 'passenger' that departs from edge 'WC' and ends the trip at edge 'CS'.

```
<vehicle id="veh_3" type="passenger" from="WC" to="CS" />
```

Optionally, a list of intermediate edges (separated by comma) can also be specified with the 'via' attribute. In the following example, veh_4 departs from edge WC and ends the trip at edge CS like before, but it also passes edge EC along its route. Under the hood, the shortest path from WC to EC is calculated first (**WC -> CE -> EC**), then the shortest path from EC to CS is calculated (**EC -> CS**). The final route is calculated by merging the two smaller routes (**WC -> CE -> EC -> CS**).

```
<vehicle id="veh_4" type="passenger" from="WC" to="CS" via="EC" />
```

Adding Truck/Bus/Bike

So far we focused on adding vehicles of type 'passenger'. The 'passenger' type is defined in the SUMO traffic demand file as following. We have covered vehicle type here. You can define other vehicle types for bus, emergency vehicle, truck, bicycle, etc.

```
<vType id="passenger" length="5.0" minGap="2.0" maxSpeed="30.0" vClass="passenger">
   <carFollowing-Krauss accel="3.0" decel="5.0" sigma="0" tau="1.64" />
</vType>
```

As an example let's add three vehicles of type passenger, truck and bicycle. By default, SUMO shows all vehicles with a triangle. You can show the vehicles as raster images as shown in Figure 47. Click on the 'change color scheme' icon, then go to vehicles and select 'raster images' from the drop down menu.

```
<vehicle id="veh_1" type="passenger" route="movement3" />
<vehicle id="truck_1" type="truck" route="movement8" />
<vehicle id="bike_1" type="bicycle" route="movement8" />
```

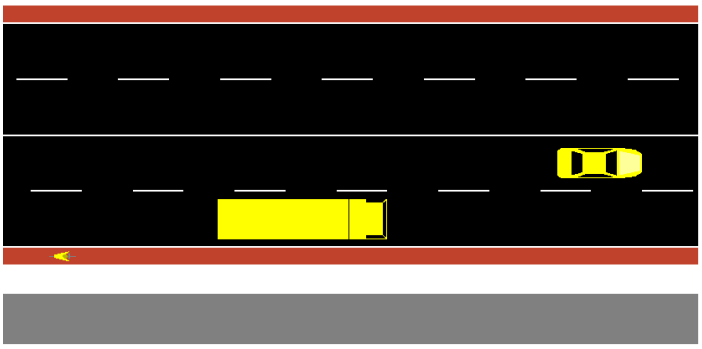


Figure 47: **Adding bike, truck and passenger vehicle into the simulation**

Adding a Stopped/ Parked Vehicle

Setting the 'status' attribute to 'stopped', inserts a non-moving vehicle **on the road**. The following line inserts veh_1 at position 30 of lane WC_4 with speed zero. The brake light of veh_1 is on and it stays there until the end of simulation. You can control the stopped duration using the 'duration' attribute. The vehicle is stopped for the 'duration' period and then starts moving.

```
<vehicle id="veh_1" type="passenger" route="movement3" departLane="4"
departPos="30" status="stopped" />
```

Setting the 'status' attribute to 'parked', inserts a non-moving vehicle **besides the road**[1]. The following line inserts veh_1 at position 30 of lane WC_4 with speed zero at parking mode. Parked vehicle veh_1 stays there until the end of simulation and cannot affect the traffic flow. You can control the parked duration using the 'duration' attribute. The vehicle is parked for the 'duration' period and then starts moving.

```
<vehicle id="veh_1" type="passenger" route="movement3" departLane="4"
departPos="30" status="parked" />
```

DSRC Probability

The 'DSRCprob' attribute is a real value in the [0,1] range and determines the probability of a vehicle being DSRC-enabled. '1' means that the vehicle is DSRC-enabled and can participate in V2X communication

[1] Vehicles are parked on the right hand side of the road relative to the traffic direction.

whereas ‘0’ means the vehicle is not DSRC-enabled. The probability follows a uniform distribution with the simulation run number used as the seed.

3.5 Adding Vehicle Flow

A **flow** of vehicles consists of more than one vehicle that have the same starting and ending edge. Each vehicle in a flow has a unique id of the form “flowId.xxx” and has a different depart time based on the distribution type. Vehicles in a flow can have different types. This definition is more general than how SUMO defines a vehicle flow as discussed in Part 2, Section 3.2. To insert a vehicle flow, use the “vehicle_flow” element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
id	string	"veh_2"	Yes
type	string/string list	"TypeManual"	Yes
typeDist	double list	"90,10"	*
route	string	"route0"	*
from	string	"edge1"	*
to	string	"edge10"	*
via	string/string list	"edge1,edge2"	*
color	string	"green"	No (default: "yellow")
departLane	int	"3"	No (default: "-5")
departPos	double (m)	"60"	No (default: "0")
departSpeed	double (m/s)	"20"	No (default: "0")
laneChangeMode	int	"533"	No (default: "597")
begin	double (s)	"5"	No (default: "0")
number	int	"10"	No (default: "-1")
end	double (s)	"100"	No (default: "-1")
seed	int	"43"	No (default: "0")
distribution	string	"poisson"	Yes
period	double (s)	"1"	*
lambda	double (veh/h)	"500"	*
probability	double	"0.7"	*
DSRCprob	double	"0.5"	No (default: config)
***** See the text for more information.			

Vehicle departure distribution

The “distribution” attribute in a flow specifies the distribution of vehicle departure time and it can be either ‘deterministic’, ‘Poisson’ or ‘uniform’. In the deterministic distribution, one vehicle is departed in every ‘period’ seconds. In the Poisson distribution, vehicle’s departure time follows a Poisson distribution with parameter ‘lambda’. In the uniform distribution, a vehicle is departed in each second with probability of ‘probability’ that is between 0 and 1. Let’s see some examples.

We define vehicle flow ‘flow1’. Vehicles in this flow are of type ‘passenger’ with color ‘red’ that all take ‘route1’ route. The vehicles’ id are ‘flow1.0’, ‘flow1.1’, etc. The flow begins at time 0 and ends at time 400s with period of 1s. This means that the first vehicle departs at 0, second vehicle departs at 1s, third vehicle departs at 2s, fourth vehicle departs at 3s and so on.

```
<vehicle_flow id="flow1" type="passenger" color="red" route="route1"
begin="0" end="400" distribution="deterministic" period="1" />
```

Let's define a second flow called 'flow2'. Vehicles in this flow are of type 'passenger' with color 'green' that all take 'route3' route. The vehicles' id are 'flow2.0', 'flow2.1', etc. The flow begins at time 0 and ends at time 400s with period of 2s. This means that the first vehicle departs at 0, second vehicle departs at 2s, third vehicle departs at 4s, fourth vehicle departs at 6s and so on.

```
<vehicle_flow id="flow2" type="passenger" color="green" route="route3"
begin="0" end="400" distribution="deterministic" period="2" />
```

Figure 48 shows 'flow1' and 'flow2'. 'flow1' consists of red vehicles going from north to west. 'flow2' consists of green vehicles going from west to south.

The following line defines 'flow3' with Poisson distribution and parameter lambda=300.5 veh/h. The default seed value is zero, but you can override this using the 'seed' attribute.

```
<vehicle_flow id="flow3" type="passenger" color="red" route="route1"
begin="0" end="400" distribution="poisson" lambda="300.5" seed="43" />
```

The following line defines 'flow4' with uniform distribution and parameter probability=0.2. The default seed value is zero, but you can override this using the 'seed' attribute.

```
<vehicle_flow id="flow4" type="passenger" color="red" route="route1"
begin="0" end="400" distribution="uniform" probability="0.2" seed="43" />
```

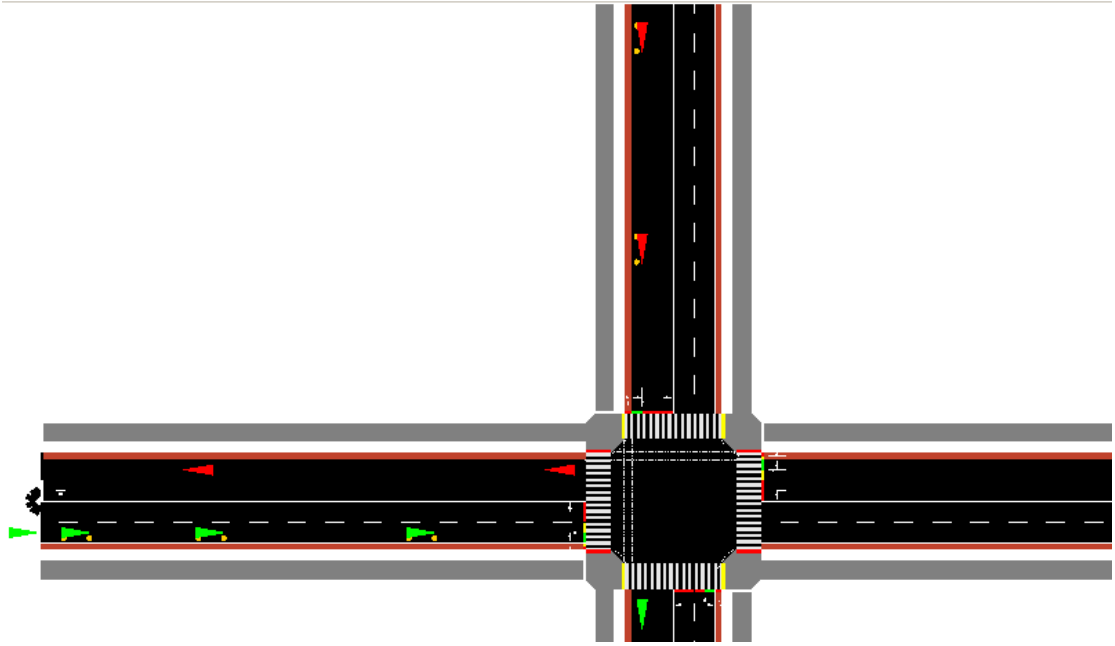


Figure 48: **Traffic simulation with two flows**

Begin, End and Number Attributes

The vehicle flow starts from the time 'begin' and finishes at time 'end'. If you remove both of these attributes, then the flow begins on simulation start and ends on simulation end. In order to limit the number of vehicles in a flow, you can remove the 'end' tag and use the 'number' tag instead, as shown in the following.

```
<vehicle_flow id="flow5" type="passenger" color="green" route="route3"
begin="0" number="20" distribution="deterministic" period="2" />
```

Flows with Incomplete Routes

Route information for a flow may also take the form of origin/destination edges instead of a complete list of edges. In this case the simulation performs fastest-path routing based on the traffic conditions found in the network at the time of flow departure to compute the route.

```
<vehicle_flow id="flow6" type="passenger" color="red" from="WC" to="CS"
distribution="deterministic" period="1" />
```

Optionally, a list of intermediate edges can also be specified with the 'via' attribute.

```
<vehicle_flow id="flow7" type="passenger" color="red" from="WC" to="CS"
via="EC" distribution="deterministic" period="1" />
```

Flows with Different Vehicle Types

You can define a flow of vehicles of different types. You need to list all vehicle types that a flow can have in the 'type' attribute (separated with comma), and then use the 'typeDist' attribute to specify the percent of each vehicle type (separated with comma). For example the following line defines 'flow_8' that consists of two vehicle types: 'passenger' and 'truck' where 70 percent of vehicles are of type 'passenger' and 30 percent of vehicles are of type 'truck'.

```
<vehicle_flow id="flow8" type="passenger,truck" typeDist="70,30"
route="route3" distribution="deterministic" period="2" />
```

3.6 Adding Vehicle Multi-Flow

The "vehicle_multiflow" element allows you to define multiple flows that have the same starting edge. It's very similar to the "vehicle_flow" described in the previous section, but you can specify a list of route ids separated with comma in the "route" attribute and then use the "routeDist" to specify the distribution of routes. Note that all route IDs listed in the "route" attribute should have the same starting edge.

In our four-way signalized intersection example, assume that the traffic demand approaching from north is 200 veh/h between time 0 and 400s. 90% percent of vehicles are 'passenger' and the remaining 10% are 'truck'. 70% of vehicles go straight, 30% turn left and 0% turn right. The following line defines such a traffic using "vehicle_multiflow" element.

```
<vehicle_multiflow id="multiFlow1" type="passenger,truck" typeDist="90,10"
route="movement2,movement5,route1" routeDist="70,30,0" begin="0" end="400"
distribution="poisson" lambda="200" seed="43" />
```

3.7 Adding Vehicle Platoon

Vehicle platooning is a technique where highway traffic is organized into groups of close-following vehicles called **platoon** or **convoy**. Platooning enables vehicles to drive closer (maintain a smaller headway) than normal vehicles with the same speed. Vehicle platooning improves traffic throughput, homogeneity and safety and at the same time reduces fuel consumption and emissions. You can add vehicle platoons using the "vehicle_platoon" element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
id	string	"plt1"	Yes
type	string	"TypeManual"	Yes
size	int	"5"	Yes
route	string	"route0"	*
from	string	"edge1"	*
to	string	"edge10"	*
via	string/string list	"edge1,edge2"	*
depart	double (s)	"10"	No (default: "0")
departLane	int	"3"	No (default: "0")
departPos	double (m)	"60"	No (default: "0")
platoonMaxSpeed	double (m/s)	"20"	No (default: "10")
color	string	"red"	No (default: "yellow")
fastCatchUp	bool	"true"	No (default: "false")
interGap	double (s)	"3.0"	No (default: config)
pltMgmtProt	true/false	"false"	No (default: "false")
optSize	int	"5"	No (default: config)
maxSize	int	"10"	No (default: config)

We will switch to the example located in **examples/7_platooning** to demonstrate platooning. So change the SUMOconfig configuration in your omnetpp.ini file to "07_platooning/example.sumocfg".

3.7.1 Homogeneous Vehicle platooning

The following line inserts a vehicle platoon 'plt1' into the simulation. Platoon 'plt1' is red and has size 5 and departs at time 0 (The platoon size should be greater than or equal to 1). All vehicles in a platoon have the **same** type 'passenger' and follow the same route. Platoon leader has depth of 0, and it increases as we go farther. Vehicles in the platoon are named: plt1, plt1.1, plt1.2, plt1.3, plt1.4. Note that the platoon leader has the same id as the platoon name.

```
<vehicle_platoon id="plt1" type="passenger" size="5" route="route0"
depart="0" departLane="1" departPos="40" platoonMaxSpeed="20" color="red" />
```

The 5-vehicle platoon looks like Figure 49. The 'departPos' specifies the position that the platoon leader is inserted. If there is not enough space for all vehicles to be inserted, then you will get error. If you do

not specify the ‘departPos’ then the platoon leader is placed at the nearest position from the start of the lane so that all followers fit on the lane. Note that lane changing is disabled for vehicles in a platoon. This means that you need to insert the platoon on the correct lane (by changing the ‘departLane’ parameter) if you want the platoon to make a left or right turn or to go straight when approaching an intersection.

Figure 49: **Platoon of 5 vehicles**

The platoon leader speeds up from 0 m/s to the ‘platoonMaxSpeed’. Note that the platoon leader’s maximum speed is bounded to the ‘maxSpeed’ attribute in the vehicle type. The “**intraGap**” (time-gap between vehicles in a platoon) is specified in the car-following model which is defined in the vehicle type using the ‘tau’ attribute. In this example, ‘tau’ is equal to 1.64s which means when the platoon reaches its maximum speed of 20m/s, the intraGap would be around 20 m/s * 1.64 s = 32.8m.

Note that if you set ‘platoonMaxSpeed’ too high, then followers might catch-up slowly. You can fix this (potentially with sacrificing realism) by setting the ‘fastCatchUp’ attribute to ‘true’ in order to disregard the maximum acceleration safety check in the followers. In this case, following vehicles can accelerate higher that the maximum acceleration specified in ‘accel’ attribute in the vehicle type.

Attribute “**interGap**” shows the time-gap in seconds that the platoon leader tries to maintain with the front vehicle. As an example, consider platoon plt1 and plt2 that are inserted back to back on the same lane. Platoon ‘plt1’ is red with size 5 and type ‘passenger’ whereas ‘plt2’ is green with size 5 and type ‘truck’. The result is shown in Figure 50.

```
  <vehicle_platoon id="plt1" type="passenger" size="5" route="route0"
depart="0" departLane="0" departPos="200" platoonMaxSpeed="20" color="red"
interGap="3.5" />

  <vehicle_platoon id="plt2" type="truck" size="5" route="route0" depart="0"
departLane="0" departPos="160" platoonMaxSpeed="20" color="green"
interGap="3.5" />
```

As you can see, the gap between vehicles in ‘plt2’ (green) is bigger since the ‘truck’ vehicle type has a bigger ‘tau’. The time-gap between platoons (interGap) is 3.5s; thus the platoon leader of plt2 tries to maintain a time-gap of 3.5s with the front platoon.

Figure 50: **Two 5-vehicle platoons traveling with 20 m/s**

Note: While adding platoons on the same lane, make sure that they do not overlap with other vehicles. The ‘addNode’ module performs a basic overlap check for platoons. If the lane is occupied, then ‘addNode’ will try insertion repeatedly in subsequent time steps. However, the platoon overlap check is not rigorous and it might fail sometimes.

Note: The default car-following model in SUMO, Krauss, is not suited for platooning purposes. The platooning is best realized with CACC car-following model. CACC needs V2V wireless communication to exchange different control parameters between vehicles. Check Section 24.3 for more information on CACC car-following model defined by VENTOS.

3.7.2 Non-homogeneous Vehicle platooning

You can insert a platoon of vehicles that each has a different type. The index attribute shows the location of a vehicle in the platoon starting from 0 that is reserved for the platoon leader. For example the following code defines platoon 'plt1' of 5 vehicles with default type of 'passenger'. Platoon leader and the first follower have a different vehicle type. As shown in Figure 51, 'truck' represents a truck that has a higher minimum gap and maintains a bigger gap with the front vehicle for safety.

```
<vehicle_platoon id="plt1" type="passenger" size="5" route="route0"
departLane="0" departPos="200" >
      <member index="0" type="emergency" />
      <member index="1" type="emergency" />
      <member index="2" type="truck" />
</vehicle_platoon>
```

Figure 51: **Non-homogeneous platoon of 5 vehicles**

Note: Be cautious when mixing vehicles with different car-following models as they might not be compatible with each other and it could cause platoon instability or even rear-end collision.

3.7.3 Platoon Management Protocol

Vehicle platooning requires cooperation between vehicles with the help of a platoon management protocol. This allows platoons to perform different maneuvers such as merge, split and leave. The platoon management protocol implemented in VENTOS is described in this paper. You can turn this protocol on by setting the 'pltMgmtProt' attribute to 'true'. Here is a simple example:

```
<vehicle_platoon id="plt1" type="passenger" size="10" route="route0"
depart="0" departLane="0" departPos="100" platoonMaxSpeed="20"
pltMgmtProt="true" optSize="3" maxSize="20" />
```

As you have noticed, there are three different attributes related to platoon size. The 'size' attribute shows the initial platoon size that will be inserted on the road. The 'optSize' attribute shows the optimal platoon size that the platoon management protocol tries to maintain using merge/split maneuvers. The 'maxSize' attribute shows the maximum platoon size. Any merge request that increases the target platoon size beyond 'maxSize' will be rejected.

The above platoon looks like Figure 52. When the platoon management protocol is active, the platoon leader and its followers are marked with red and blue color respectively. As we go farther from the platoon

leader, the blue color fades to show the platoon depth. Platoon coordination is done with 'trafficControl' module discussed in Section 4.3. You can find a platooning example in Section 25.

Figure 52: **Platoon of 10 vehicles with active platoon management protocol**

3.8 Adding Adversary Node

You can add an adversary node to the simulation to perform different security attacks on the wireless communication. To define an adversary, use the "adversary" element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
id	string	"adv0"	Yes
pos	double, double, double	"900,800,0"	Yes
drawMaxIntfDist	true/false	"false"	No (default: "true")
color	string	"red"	No (default: "green")
filled	true/false	"true"	No (default: "false")

The following line defines an adversary with name "jammer" at SUMO coordinate "1890,0,0".

```
<adversary id="jammer" pos="1890,0,0" />
```

You can find the adversary's application layer code in src/nodes/adversary folder.

4 Controlling Vehicular Traffic

We introduced the 'addNode' module in Section 3 that allows you to add different node types such as RSUs, obstacles, vehicle flow, etc. into the simulation. The next step is to control the vehicular traffic by changing vehicle's speed, perform platooning maneuvers, etc. This is achieved through the 'trafficControl' module. All nodes are defined in the 'trafficControl.xml' file located in the same folder as omnetpp.ini configuration file. The trafficControl module reads this file and decides how/when to affect the vehicular traffic. In the following sections, we'll show you how to write a 'trafficControl.xml' file.

4.1 Changing Vehicle's Speed

You can change a vehicle's speed using the "speed" element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
id	string	"veh0"	Yes
begin	double	"70.3"	No (default: "-1")
end	double	"100"	No (default: sim end)
duration	double	"30"	No (default: "-1")
edgeId	string	"1to2"	No (default: "")

edgePos	double	"20.4"	No (default: "-1")
laneId	string	"1to2_0"	No (default: "")
lanePos	double	"16.5"	No (default: "-1")
value	string	"20" or "sin(t)"	Yes
maxAccel	double	"5"	No (default: default)
maxDecel	double	"3"	No (default: default)
file	string (file path)	"trajectory.txt"	No (default: "")
headerLines	int	"4"	No (default: "-1")
columns	int,int	"1,2"	No (default: "")

4.1.1 Changing Vehicle's Speed from a Specific Time

The vehicle's speed is set to 5 m/s starting from 70s. The vehicle either speeds up with the maximum acceleration or slows down with the maximum deceleration in order to reach 5 m/s and maintains the speed until the end of simulation. Note that the vehicle's speed is affected by the maximum vehicle speed, lane's maximum allowed speed, presence of an obstacle or front vehicle, etc. Refer to Section 4.2 for practical examples.

```
<speed id="veh.0" begin="70" value="5" />
```

You can override the default's maximum acceleration and maximum deceleration of a vehicle using the 'maxAccel' and 'maxDecel' attributes respectively. For example, the following line causes 'veh.0' to stop with higher deceleration of -7 m/s^2 at 70s. From now on, veh.0 has maximum deceleration of -7 m/s^2. Note that setting the 'maxDecel' to a high value might cause rear-end collision in the vehicle that is following veh.0.

```
<speed id="veh.0" begin="70" value="0" maxDecel="7" />
```

The 'end' attribute allows you to specify an ending time for speed change. For example the following line sets the vehicle's speed to 5 m/s from 70s to 100s. After the ending time, the vehicle's speed reverts back to the old value before speed change.

```
<speed id="veh.0" begin="70" end="100" value="5" />
```

Alternatively, you can use the 'duration' attribute. Note that you are not allowed to use 'end' and 'duration' attributes together.

```
<speed id="veh.0" begin="70" duration="30" value="5" />
```

Note: Make sure that speed changes for a vehicle is conflict free. If you are trying to change the speed of a vehicle multiple times at the same time, VENTOS throws you an error. For example consider the following lines. We are trying to change the speed of vehicle 'veh.0' to two different values at time 40s that is not allowed.

```
<speed id="veh.0" begin="40" value="5" />
<speed id="veh.0" begin="40" value="10" />
```

As another example consider the following lines. The first line is trying to change the speed of 'veh.0' at time 50s to 10 m/s for duration of 10s. This means that at 60s, speed of veh.0 will revert back to its old value. On the other hands, the second line is trying to change the speed at 60s that is not allowed.

```
<speed id="veh.0" begin="50" duration="10" value="10" />
<speed id="veh.0" begin="60" value="20" />
```

To fix the above error, you can begin the second speed change in the next simulation time step (assuming that the time step is 0.1):

```
 <speed id="veh.0" begin="50" duration="10" value="10" />
<speed id="veh.0" begin="60.1" value="20" />
```

4.1.2 Changing Vehicle's Speed in a Specific Edge

You can change the speed of a vehicle, once it reaches to position 'edgePos' of edge 'edgeId' as following:

```
<speed id="veh.0" edgeId="1to2" edgePos="200" value="5" />
```

It goes without saying that the vehicle's route should include 'edgeId', otherwise the above line would never execute. Also note that 'edgeId' and 'edgePos' triggers the start point for speed change. This means that the speed of 'veh.0' is retained even after leaving edge '1to2'. This is the main difference with Variable Speed Signs (VSS) that assign speed limit to lanes and not vehicles.

If you do not specify the 'edgePos' then the vehicle's speed changes as soon as it reaches to that edge.

```
<speed id="veh.0" edgeId="1to2" value="5" />
```

Similarly, you can use 'duration' and 'end' attributes as well as 'maxAccel' and 'maxDecel'.

```
<speed id="veh.0" edgeId="1to2" edgePos="200" duration="10" value="5" />
```

Note: Make sure that speed changes for a vehicle is conflict free. If you are trying to change the speed of a vehicle multiple times at the same location, VENTOS throws you an error. For example consider the following lines. We are trying to change the speed of vehicle 'veh.0' to two different values at the same location that is not allowed.

```
<speed id="veh.0" edgeId="1to2" value="5" />
<speed id="veh.0" edgeId="1to2" edgePos="0" value="10" />
```

4.1.3 Changing Vehicle's Speed in a Specific Lane

You can change the speed of a vehicle, once it reaches to position 'lanePos' of lane 'laneId' as following. Note that 'laneId' and 'lanePos' triggers the start point for speed change. This means that the speed of 'veh.0' is retained even after leaving lane '1to2_0'.

```
<speed id="veh.0" laneId="1to2_0" lanePos="200" value="5" />
```

Negative 'lanePos' values count from the end of the edge.

```
<speed id="veh.0" laneId="1to2_0" lanePos="-30" value="5" />
```

If you do not specify the 'lanePos' then the vehicle's speed changes as soon as it reached to that lane.

```
<speed id="veh.0" laneId="1to2_0" value="5" />
```

Similarly, you can use 'duration' and 'end' attributes as well as 'maxAccel' and 'maxDecel'.

```
<speed id="veh.0" laneId="1to2_0" lanePos="200" duration="10" value="5" />
```

Note: Make sure that speed changes for a vehicle is conflict free. If you are trying to change the speed of a vehicle multiple times at the same location, VENTOS throws you an error. For example consider the following lines. We are trying to change the speed of vehicle 'veh.0' to two different values at the same location that is not allowed (length of lane 1to2_0 is 2000m).

```
<speed id="veh.0" laneId="1to2_0" lanePos="1980" value="5" />
<speed id="veh.0" laneId="1to2_0" lanePos="-20" value="10" />
```

4.1.4 Changing Vehicle's Speed Based on a Function

As we have seen so far, you can set the vehicle's speed as a double number using the 'value' attribute. You can set the vehicle's speed as a function of time (denotes by t) that allows you to change the speed in a complex behavior. Evaluation of the speed function is done using the ExprTk library that supports many mathematical operators/functions as following:

Operators:

+	-	*	/	%	^

Functions:

min	max	avg	sum	abs	ceil	floor	round	roundn

exp	log	log10	logn	pow	root	sqrt	clamp	inrange	swap

Trigonometry:

sin	cos	tan	acos	asin	atan	atan2	cosh	cot

csc	sec	sinh	tanh	d2r	r2d	d2g	g2d	hyp

Consider the following example. Starting from 60s the vehicle's speed follows $v(t) = \frac{1}{2} \times (t - 60)$, where 't' is the simulation time step. This means that the vehicle speeds up with slope (acceleration) of 0.5 until it reaches to the maximum speed. It goes without saying that the slope cannot exceed the maximum acceleration of the vehicle. **Note that 'end' or 'duration' attribute is mandatory when using a speed function**.

```
<speed id="veh.0" begin="60" end="1000" value="0.5*(t-60)" />
```

You can change the speed based on a sinusoid function as below. This is a periodic function with amplitude of 30 m/s, offset of 60s and period of $\frac{2\pi}{0.1} \cong 62.8s$. Note that vehicle's speed cannot be negative; thus negative values are interpreted as zero speed.

```
<speed id="veh.0" begin="60" end="1000" value="30*sin(0.1*(t-60))" />
```

Another useful periodic function is a square wave (a.k.a pulse train) that consists of instantaneous transitions between two levels. The following square wave has amplitude of 30 m/s, offset of 60s and period of 60s. Note that instantaneous transitions between speed levels require an infinite maximum acceleration and deceleration. Since the acceleration/deceleration is bounded, transition slope is also bounded.

```
<speed id="veh.0" begin="60" end="1000" value="30*(-1)^floor(2(t-60)/60)" />
```

The following example shows a saw-tooth wave that consists of an infinite sequence of truncated ramp functions concatenated together. This function has amplitude of 30 m/s, period of 60s and offset of -1.

```
<speed id="veh.0" begin="60" end="1000" value="30*((t/60-1)-floor(t/60-1))" />
```

Refer to Section 4.2 for practical examples and to see how the speed/acceleration profile looks like.

4.1.5 Changing Vehicle's Speed from a Dump File

You can change a vehicle's speed based on a speed dump file. For example, the following line changes the speed of vehicle 'veh.0' starting from 10s by reading the 'trajectory.txt' file that is located in the current example directory.

```
<speed id="veh.0" begin="10" file="trajectory.txt" />
```

The trajectory.txt file looks like the following. It has a single column and each row shows a speed in m/s. Starting from 10s, the first line of the file is read and the speed of 'veh.0' is set to 24.5 m/s. In the next time step, the second row is read and the speed of 'veh.0' is set to 24.25 m/s. The process is repeated in each time step until all rows are read. Note that empty rows are skipped.

10	24.5
10+Δt	24.25
10+2Δt	24
10+3Δt	23.75
10+4Δt	23.5
10+5Δt	23.25
10+6Δt	23
10+7Δt	22.75
10+8Δt	22.5
10+9Δt	22.25
	…

At the beginning of the simulation, VENTOS shows you the maximum acceleration or deceleration needed by the vehicle in order to follow the speed from the 'trajectory.txt' file as following:

Speed change of vehicle 'veh.0' using file 'trajectory.txt' requires maximum acceleration of '1.5'
Speed change of vehicle 'veh.0' using file 'trajectory.txt' requires maximum deceleration of '-2.5'

You need to make sure that the speed of 'veh.0' at time 10s (when speed changes begins) is close to 24.5 m/s; otherwise the speed change will not follow the 'trajectory.txt' closely. Assume that 'veh.0' is traveling with 5 m/s and has maximum acceleration of 3 m/s^2. The first line of the 'trajectory.txt' file sets the speed to 24.5 m/s, but the vehicle cannot jump to that speed in the next time step. Jumping from 5 m/s to 24.5 m/s requires an acceleration of at least 195 m/s^2, assuming that the simulation time step is 0.1s.

A feasible approach looks like the following. We first set the speed of veh.0 to 24.5 m/s and then start following the speed in 'trajectory.txt' file.

```
<speed id="veh.0" begin="5" value="24.5" />
<speed id="veh.0" begin="10" file="trajectory.txt" />
```

HeaderLines

You can skip the first 'n' lines of the file using the 'headerLines' attribute. For example, the following line skips the first three lines of the 'trajectory.txt' file.

```
<speed id="veh.0" begin="10" file="trajectory.txt" headerLines="3" />
```

Speed Values with Timestamp

The dump file can consist of <time, speed> pairs that are saved in two separate columns as shown below. Note that the time column should be sorted in an ascending order.

Time	Speed
12	24.5
12.1	24.25
12.2	24
12.3	23.75
…	…

As an example, the following line changes the speed of vehicle 'veh.0' based on the <time, speed> pair defined in the 'trajectory.txt' file starting from 1s.

```
<speed id="veh.0" begin="1" file="trajectory.txt" />
```

Column Selection

If your input file has multiple columns, then you can specify the location of speed and time columns using the 'columns' attribute. Assume that the input file has the following format:

Time	Accel	Speed
12	-2	24.5
12.1	-2.5	24.25
12.2	-2	24
12.3	-2.5	23.75
…	…	…

The following line uses the 'columns' attribute to specify the location of time and speed columns as a comma separated string. Note that column numbering starts from 1.

```
<speed id="veh.0" begin="1" file="trajectory.txt" columns="1,3" />
```

Specifying a single value for 'columns' attribute denotes the speed column.

CSV Support

The input file can also be of type CSV (Comma-Separated Values) with **.csv** extension. Note that if the file has multiple columns, then you need to specify the location of speed and time columns with 'columns' attribute as discussed in the previous section. Also note that spreadsheet data generated with Microsoft Excel or LibreOffice Calc can be saved as CSV files.

```
<speed id="veh.0" begin="10" file="trajectory.csv" />
```

4.2 Changing Vehicle's Speed: An Example

In the previous section we showed you how to use the 'trafficControl' module to set the speed of a vehicle. Here we will show you how the speed/acceleration profile of a vehicle looks like after speed change. You can refer to the example located in the 'examples/trafficControl' folder. This example shows a one-lane freeway where vehicles are inserted from the left.

Open the omnetpp.ini configuration file. The TraCI module is active and 'SUMOconfig' is pointing to the SUMO configuration file located in the 'sumocfg' folder. The 'addNode' module is active and points to the 'add_00' configuration that inserts a single vehicle 'veh.0' with departure speed of 20 m/s. On top of that the 'trafficControl' module is also active and points to the 'control_0' configuration. The 'record_stat' is active on all vehicles and four data (vehId, lanePos, speed and accel) is collected from each vehicle. The output is written to the 'results/000_vehicleData.txt' file. We then use the Matlab script 'SpeedProfile.m' located in the scripts folder to read the output file and plot different figures.

```
include ../omnetpp_general.ini

[Config TrafficControl]
description = "This scenario shows you how trafficControl module works"

Network.TraCI.active = true
Network.TraCI.SUMOconfig = "sumocfg/6hello.sumo.cfg"
Network.TraCI.terminateTime = 800s

Network.addNode.id = "add_00"
Network.trafficControl.id = "control_0"
Network.gui.id = "gui_0"

Network.V[*].record_stat = true
Network.V[*].record_list = "vehId | lanePos | speed | accel"
```

The 'control_0' in the 'trafficControl.xml' looks like this:

```
<speed id="veh.0" begin="40" value="5" />
<speed id="veh.0" begin="70" duration="20" value="30" />
<speed id="veh.0" begin="100" value="20" maxAccel="5" />
<speed id="veh.0" begin="120" value="5" maxDecel="7" />
```

Run the simulation and wait for 'veh.0' to exit the simulation. Run the Matlab script to get two plots as shown in Figure 53.

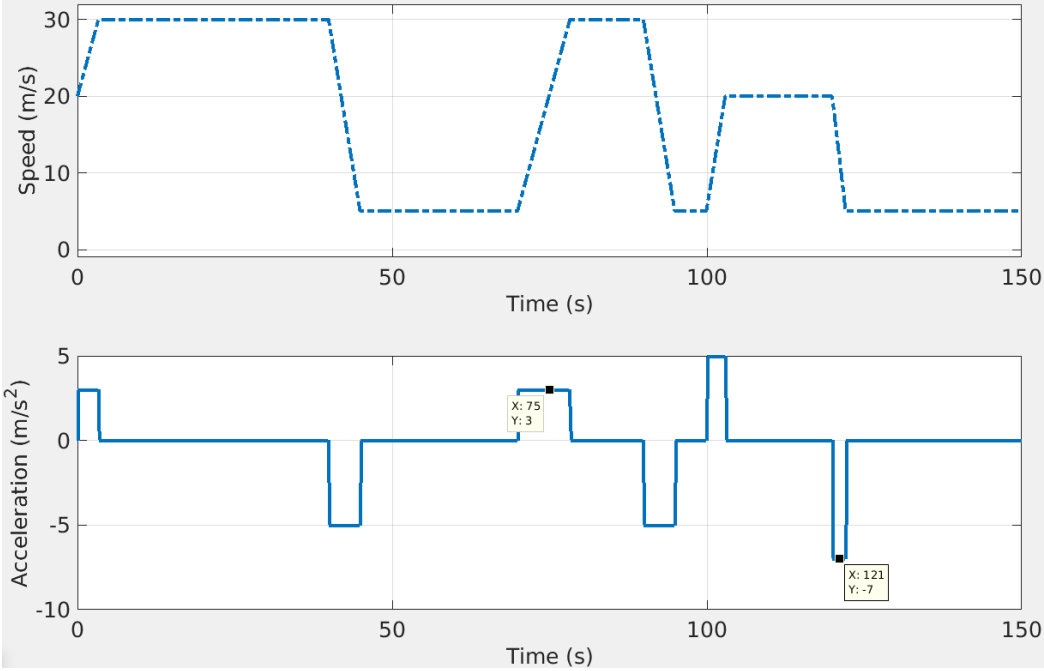


Figure 53: **Speed and acceleration of 'veh.0' over time**

Let's interpret the plots more closely:

1. 'veh.0' is departed at time 0 with departure speed of 20 m/s
2. The maximum speed of 'veh.0' is 30 m/s; thus it speeds up to 30 m/s with acceleration of 3 m/s^2
3. Starting from 40s, the speed of 'veh.0' is changed to 5 m/s. The vehicle slows down with deceleration of -5 m/s^2 that is the default deceleration as specified in the vehicle's type
4. Starting from 70s, the speed of 'veh.0' is changed to 30 m/s for a duration of 20s. The vehicle speeds up with acceleration of 5 m/s^2 and at 90s its speed changes back to 5 m/s
5. Starting from 100s, the speed of 'veh.0' is changed to 20 m/s with acceleration of 5 m/s^2
6. Starting from 120s, the speed of 'veh.0' is changed to 5 m/s with deceleration of -7 m/s^2
7. The vehicle retain 5 m/s speed until it arrived to the destination

As you can see by mixing different speed controls, you can generate any arbitrary speed profiles. Periodic speed profiles are very common in stability testing of a platoon of ACC/CACC vehicles as we will see in Section 24.2 and Section 24.3. In the remaining of this section we generate multiple periodic speed profiles such as sinusoid, square and saw-tooth waves.

Change the trafficControl id to the following:

```
Network.trafficControl.id = "control_1"
```

Where it looks like this:

```
<speed id="veh.0" begin="20" value="0" />
<speed id="veh.0" begin="60" end="150" value="0.5*(t-60)" />
<speed id="veh.0" begin="190" end="1000" value="30*sin(0.1*(t-190))" />
```

Run the simulation and wait for 'veh.0' to exit the simulation. Run the Matlab script to get two plots as shown in Figure 54.

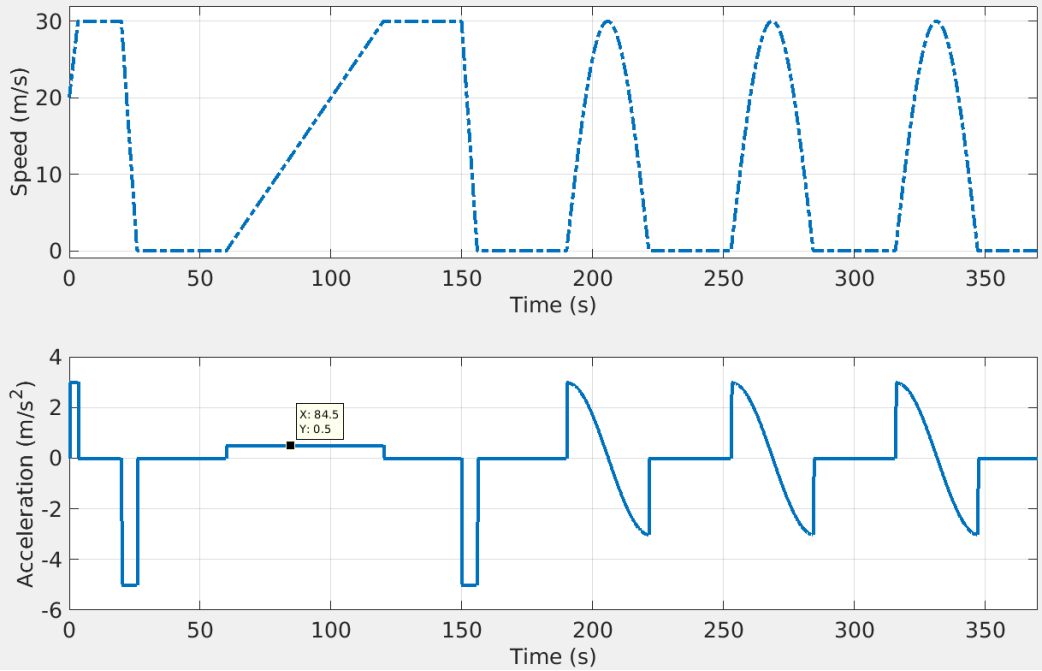


Figure 54: **Speed and acceleration of 'veh.0' over time**

Let's interpret the plots more closely:

1. 'veh.0' is departed at time 0 with departure speed of 20 m/s
2. The maximum speed of 'veh.0' is 30 m/s; thus it speeds up to 30 m/s with acceleration of 3 m/s^2
3. Starting from 20s, the speed of 'veh.0' is changed to zero. The vehicle slows down with deceleration of -5 m/s^2 that is the default deceleration as specified in the vehicle's type
4. Starting from 60s, the speed of 'veh.0' follows $v(t) = \frac{1}{2} \times (t - 60)$. Thus, it speeds up with slope of 0.5 until it reaches the maximum speed of 30 m/s. The speed remains at 30 m/s until 150s where the vehicle speed reverts back to 0 m/s
5. Starting from 190s, the speed of 'veh.0' follows $v(t) = 30 \times \sin(0.1 \times (t - 190))$. Note that vehicle's speed cannot be negative

Change the trafficControl id to the following:

```
Network.trafficControl.id = "control_2"
```

Where it looks like this:

```
<speed id="veh.0" begin="20" value="0" />
<speed id="veh.0" begin="60" end="1000" value="30*(-1)^floor(2(t-60)/60)" />
```

Run the simulation and wait for 'veh.0' to exit the simulation. Run the Matlab script to get two plots as shown in Figure 55.

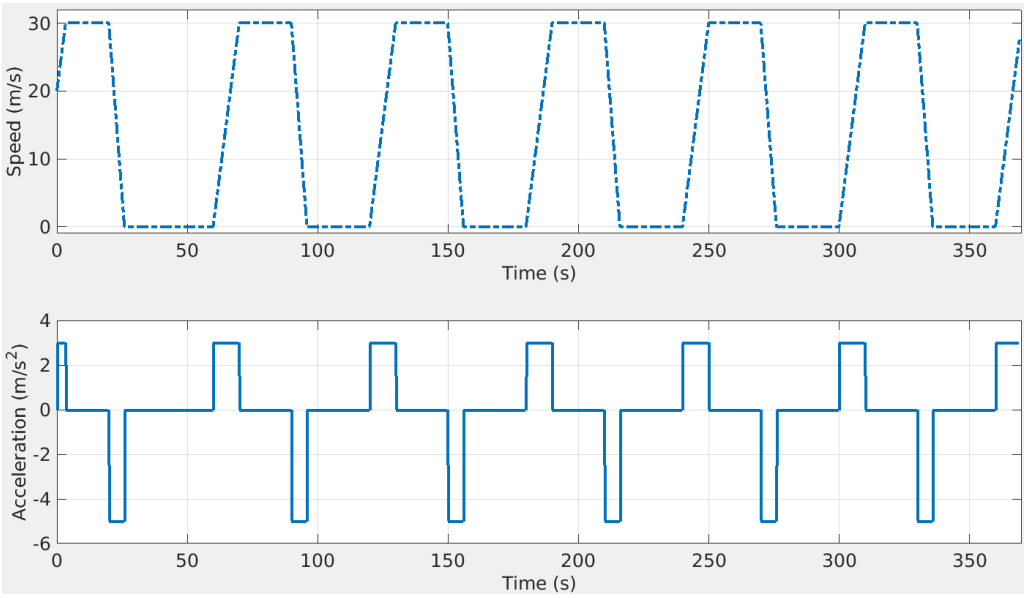


Figure 55: **Speed and acceleration of 'veh.0' over time**

As the last example, change the trafficControl id to the following:

```
Network.trafficControl.id = "control_3"
```

Where it looks like this. The plot is shown in Figure 56.

```
<speed id="veh.0" begin="20" value="0" />
<speed id="veh.0" begin="60" end="1000" value="30*((t/60-1)-floor(t/60-1))" />
```

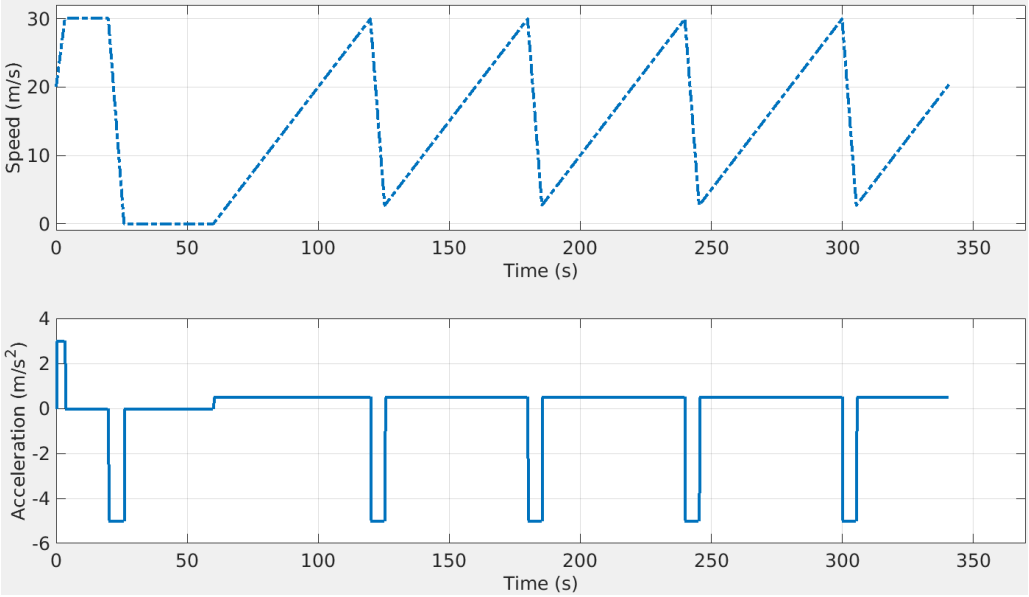


Figure 56: **Speed and acceleration of 'veh.0' over time**

4.3 Platoon Coordination

In Section 3.7.3, we discussed how to insert a platoon that is running a platoon management protocol. The 'trafficControl' module allows you to perform different maneuvers (merge, split, leave) on a platoon. We emphasize that platooning maneuvers are only allowed when a platoon management protocol is running, meaning that the 'pltMgmtProt' attribute is set to 'true'. You can refer to Section 25 for practical examples on platoon coordination.

4.3.1 Changing Platoon Optimal size

You can change the optimal platoon size in one or more platoons using the "optSize" element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
pltId	string	"plt1"	No (default: "")
begin	double	"70.3"	Yes
value	string	"5"	Yes

For example, the following line changes the optimal platoon size of 'plt1' to 9 starting from 10s. Make sure 'plt1' is the id of a platoon otherwise you will get error.

```
<optSize pltId="plt1" begin="10" value="9" />
```

If you do not specify a platoon id, then all platoons are affected.

```
<optSize begin="10" value="9" />
```

4.3.2 Platoon Merge

You can merge a platoon into the front platoon with the “pltMerge” element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
pltId	string	"plt1"	Yes
begin	double	"70.3"	Yes

For example, the following line asks platoon ‘plt1’ to send a merge request to the front platoon at time 70s. The front platoon rejects the merge request if the final platoon size exceeds ‘maxSize’. The merge request is also rejected if the front platoon is busy performing a maneuver. Once the merge maneuver is finished, the optimal platoon size will be equal to the final platoon size.

```
<pltMerge pltId="plt1" begin="70" />
```

You can refer to Section 4.1 in this paper to get more detailed information about the merge maneuver. For now we do not support side merging and all merges are done from behind.

4.3.3 Platoon Split

You can split a platoon into two smaller platoons using the “pltSplit” element with the following attributes:

Attribute Name	Value Type	Value Example	Mandatory?
pltId	string	"plt1"	*
splitIndex	int	"3"	*
splitVehId	string	"plt1.3"	*
begin	double	"70.3"	Yes
* See the text for more information.			

You can specify the splitting vehicle Id inside a platoon as below:

```
<pltSplit splitVehId="plt1.3" begin="70" />
```

Or you can specify a splitting location inside a platoon such as below:

```
<pltSplit pltId="plt1" splitIndex="3" begin="70"/>
```

Note that after the split maneuver, the optimal platoon size will be equal to the current platoon size. You can refer to Section 4.2 in this paper to get more detailed information about the merge maneuver.

4.3.4 Platoon Leave

You can initiate a leave maneuver using the “pltLeave” element with the following attributes. You can ask a platoon follower or even the platoon leader to leave the platoon.

Attribute Name	Value Type	Value Example	Mandatory?
pltId	string	"plt1"	*
leaveIndex	int	"3"	*
leaveVehId	string	"plt1.3"	*
begin	double	"70.3"	Yes
leaveDirection	string	"left/right/free"	No (default: "free")
* See the text for more information.			

For example, the following line asks the last platoon follower to leave the 5-vehicle platoon 'plt1'. You can refer to Section 5.2 in this paper to get more detailed information.

```
<pltLeave pltId="plt1" leaveIndex="4" begin="70"/>
```

As another example you can initiate a platoon follower leave that is in the middle of the 5-vehicle platoon.

```
<pltLeave pltId="plt1" leaveIndex="2" begin="70"/>
```

You can also initiate a platoon leader leave as following. The first platoon follower will take over the platoon leader role. You can refer to Section 5.1 in this paper to get more detailed information.

```
<pltLeave pltId="plt1" leaveIndex="0" begin="70"/>
```

Alternatively, instead of specifying the platoon id and leave index, you can directly specify the leaving vehicle id in a platoon as following:

```
<pltLeave leaveVehId="plt1.2" begin="70"/>
```

You can specify the side of a lane change using the 'leaveDirection' attribute. It accepts 'left', 'right' and 'free' values. For example, the following line asks vehicle 'plt1.2' to leave the platoon from the left lane.

```
<pltLeave leaveVehId="plt1.2" leaveDirection="left" begin="70"/>
```

4.3.5 Enable/Disable Maneuvers

Using the "maneuver" element, you can disable or enable platooning maneuvers. This element supports the following attributes. Note that all platooning maneuvers are active by default.

Attribute Name	Value Type	Value Example	Mandatory?
pltId	string	"plt1"	No (default: "")
begin	double	"70.3"	Yes
merge	true/false	"false"	No (default: "true")
split	true/false	"false"	No (default: "true")
leaderLeave	true/false	"true"	No (default: "true")
followerLeave	true/false	"false"	No (default: "true")
entry	true/false	"true"	No (default: "true")

For example, the following line disables the merge maneuver in 'plt1' starting from 10s. This means that platoon 'plt1' does not accept any merge request and does not merge into a front platoon.

```
<maneuver pltId="plt1" begin="10" merge="false" />
```

If you do not specify a platoon id, then all platoons are affected.

```
<maneuver begin="10" merge="false" />
```

5 GUI Module

The GUI module defined in VENTOS allows you to change the viewport in SUMO-GUI or track a vehicle or a group of vehicles. The GUI module reads the 'gui.xml' file located in the same folder as omnetpp.ini configuration file and decides what to do. Note that the GUI module in not active in command-line mode.

5.1 Change Viewport

When a network is first loaded into the SUMO-GUI, the GUI gives a global view of the network and the zoom level is 100 by default. Note that the zoom level is relative to the network size and 100 means to show the whole network. You can navigate through the network using mouse or zoom in/out using the mouse wheel to bring a specific area of your network into view. Repeating this process at the beginning of each simulation run can be tiresome. The 'viewport' element defined in the 'gui.xml' file enables you to set a predefined zoom and offset for a specific view every time the simulation starts. This element supports the following attributes.

Attribute Name	Value Type	Value Example	Mandatory?
viewId	string	"View #1"	No (default: " View #0")
zoom	double	"1200"	No (default: "100")
offsetX	double	"500"	No (default: "0")
offsetY	double	"-30"	No (default: "0")
begin	double	"10"	No (default: "0")
steps	int	"20"	No (default: "0")

For example, the following line sets the zoom level and offset at the beginning of the simulation:

```
<viewport zoom="15159" offsetX="100" offsetY="-5" />
```

You can use the 'begin' attribute to specify when the viewport should change:

```
<viewport zoom="15159" offsetX="100" offsetY="-5" begin="5" />
```

You can only change one attribute such as the X offset:

```
<viewport offsetX="100" />
```

The default view in SUMO-GUI is 'View #0'. You can change the viewport in each view by specifying the view id as below. Note that the view id is in the form 'View #x', where the last digit counts the open sub-windows in the SUMO-GUI.

```
<viewport viewId="View #0" zoom="15159" offsetX="100" offsetY="-5" />
<viewport viewId="View #1" zoom="200" offsetY="55" />
```

You can zoom-in or zoom-out slowly to get an animation effect using the 'step' attribute as shown below. In this example, it takes 200 simulation steps until we reach the 15159 zoom level.

```
<viewport zoom="15159" steps="200" offsetX="100" offsetY="-5" />
```

5.2 Vehicle Tracking

The viewport element as discussed in the previous section, allows you to set zoom level and offset for a particular view. The viewport is fixed and does not move. Sometimes it is useful to track a vehicle or a group of vehicles in SUMO-GUI. In order to track a vehicle, you can located it in the network (in the menu bar click 'Locate' and then 'Locate Vehicles') and then right-click on the vehicle and select 'Start Tracking'. The camera is moved with the vehicle so that the vehicle is always at the center of the window. You can still zoom in/out using the mouse wheel, but you cannot drag the network.

The 'track' element defined in the 'gui.xml' file gives you more control over the tracking and supports the following attributes.

Attribute Name	Value Type	Value Example	Mandatory?
viewId	string	"View #1"	No (default: " View #0")
vehId	string list	"veh.0"	No (default: "")
begin	double	"10"	No (default: "0")

For example, the following like tracks vehicle 'veh' starting from time 5s:

```
<track vehId="veh" begin="5" />
```

6 Record a Video from Simulation

In order to record a video from your simulation, you can use a screen recording software. If you are using Ubuntu, we suggest installing 'Simple Screen Recorder' as following:

```
sudo add-apt-repository ppa:maarten-baert/simplescreenrecorder
sudo apt-get update
sudo apt-get install simplescreenrecorder
# if you want to record 32-bit OpenGL applications on a 64-bit system:
sudo apt-get install simplescreenrecorder-lib:i386
```

This program is easy to use and you can customize it a lot. It is fast and can encode the video on the fly. It supports multi-monitor systems and can record the entire screen or part of it. Moreover, it uses libav / ffmpeg libraries for encoding and supports many different codecs and file formats such as MP4, MKV, MOV, etc. Check this tutorial video for more information.

7 Program Skeleton in VENTOS

As mentioned in Section 2, once a road network is generated and nodes are added into the simulation, you can start writing your own algorithm. In this section, we are going to introduce the program skeleton in VENTOS. Expand the VENTOS project in the Eclipse 'project explorer' by clicking on the small triangle icon and then navigate to src/gettingStarted folder. This folder contains 3 files: **tutorial.h**, **tutorial.cc** and **modules.ned** which show the program skeleton in VENTOS.

Double clicking on the modules.ned file brings up the NED editor. Click on the Source at the button of the window to show the NED source code as below. It defines a simple module called tutorial. The associated class name is 'VENTOS::tutorial' where 'VENTOS' is namespace and 'tutorial' is the class name. Using parameter 'active' (of type bool) you can turn-on/off the module. Module 'tutorial' is added into the Network module located at 'src/Network.ned'. This tells OMNET++ that 'tutorial' is part of the simulation and needs to be loaded at the simulation start.

```
    // modules.ned
1   package VENTOS.src.gettingStarted;
2
3   simple tutorial
4   {
5       parameters:
6           @class(VENTOS::tutorial);
7
8           bool active = default(false);
9   }
```

The content of tutorial.h and tutorial.cc is shown in the following. Note that in OMNET++, the C++ classes need to be sub-classed from the cSimpleModule class. In the following example, the inheritance hierarchy of tutorial class is like this:

tutorial --> BaseApplLayer --> BaseLayer --> BaseModule --> cSimpleModule

tutorial.h:

```
1   #ifndef TUTORIAL_H  // #include guard
2   #define TUTORIAL_H
3
4   #include "baseAppl/03_BaseApplLayer.h"  // base class for all programs
5   #include "TraCICommands.h"  // header that defines TraCI commands
6
7   // program is in the VENTOS namespace
8   namespace VENTOS {
9
10  // tutorial class is inherited from BaseApplLayer class
11  class tutorial : public BaseApplLayer
12  {
13  private:
14      // You can access the TraCI interface using this pointer
15      TraCI_Commands *TraCI = NULL;
16      // Controls if the module should be active or not
17      bool active;
```

```
18      // This module is subscribed to the following two signals. The
    first signal notifies the module that the TraCI interface has been
    established. The second signal notifies the module that one simulation
    time step is executed
19      omnetpp::simsignal_t Signal_initialize_withTraCI;
20      omnetpp::simsignal_t Signal_executeEachTS;
21
22  public:
23      // class destructor
24      virtual ~tutorial();
25      // OMNET++ invokes this method when tutorial module is created
26      virtual void initialize(int);
27      // This method should return the number of initialization stages.
    By default, OMNET++ provides one-stage initialization and this method
    returns 1. In multi-stage initialization, this method should return
    the number of required stages (stages are numbers from 0)
28      virtual int numInitStages() const { return 1; }
29      // OMNET++ invokes this method when simulation has terminated
    successfully
30      virtual void finish();
31      // OMNET++ invokes this method with a message as parameter
    whenever the tutorial module receives a message
32      virtual void handleMessage(omnetpp::cMessage *);
33      // This method receives emitted signals that carry a long value.
    There are multiple overloaded receiveSignal methods for different
    value types
34      virtual void receiveSignal(omnetpp::cComponent *,
    omnetpp::simsignal_t, long, cObject* details);
35
36  private:
37      // This method is similar to the initialize method but you have
    access to the TraCI interface to communicate with SUMO. In the
    initialize method, TraCI interface is still closed
38      void initialize_withTraCI();
39      // This method is called in every simulation time step
40      void executeEachTimestep();
41  };
42
43  }
44
45  #endif   // end of #include guard
```

tutorial.cc:

```
1   #include "tutorial.h"  // including the header file above
2
3   namespace VENTOS {
4
5   // Define_Module macro registers this class with OMNET++
6   Define_Module(VENTOS::tutorial);
7
8   tutorial::~tutorial()
9   {
10
11  }
12
```

```
13  void tutorial::initialize(int stage)
14  {
15      if(stage == 0)
16      {
17          active = par("active").boolValue();
18
19          if(active)
20          {
21              // get a pointer to the TraCI module
22              TraCI = TraCI_Commands::getTraCI();
23
24              // subscribe to initializeWithTraCISignal
25              Signal_initialize_withTraCI =
    registerSignal("initializewithTraCISignal");
26              omnetpp::getSimulation()->getSystemModule()-
    >subscribe("initializeWithTraCISignal", this);
27
28              // subscribe to executeEachTimeStepSignal
29              Signal_executeEachTS =
    registerSignal("executeEachTimeStepSignal");
30              omnetpp::getSimulation()->getSystemModule()-
    >subscribe("executeEachTimeStepSignal", this);
31          }
32      }
33  }
34
    void tutorial::finish()
35  {
36      if(!active)
37          return;
38
39      // unsubscribe from initializeWithTraCISignal
40      if(omnetpp::getSimulation()->getSystemModule()-
    >isSubscribed("initializeWithTraCISignal", this))
41          omnetpp::getSimulation()->getSystemModule()-
    >unsubscribe("initializeWithTraCISignal", this);
42
43      // unsubscribe from executeEachTimeStepSignal
44      if(omnetpp::getSimulation()->getSystemModule()-
    >isSubscribed("executeEachTimeStepSignal", this))
45          omnetpp::getSimulation()->getSystemModule()-
    >unsubscribe("executeEachTimeStepSignal", this);
46  }
47
48  void tutorial::handleMessage(omnetpp::cMessage *msg)
49  {
50
51  }
52
53  void tutorial::receiveSignal(omnetpp::cComponent *source,
    omnetpp::simsignal_t signalID, long i, cObject* details)
54  {
55      Enter_Method_Silent();
56
57      // if Signal_executeEachTS is received, then call
    executeEachTimestep() method
58      if(signalID == Signal_executeEachTS)
```

```
59          {
60              tutorial::executeEachTimestep();
61          }
62          // if Signal_initialize_withTraCI is received, then call
    initialize_withTraCI() method
63          else if(signalID == Signal_initialize_withTraCI)
64          {
65              tutorial::initialize_withTraCI();
66          }
67      }
68
69      void tutorial::initialize_withTraCI()
70      {
71
72      }
73
74      void tutorial::executeEachTimestep()
75      {
76
77      }
78
79      }
```

You can change the simulation input parameters using configuration files (with ini extension). The configuration file for the tutorial example is 'omnetpp.ini' located in 'examples/gettingStarted' folder.

8 Writing your First Algorithm in VENTOS

In the previous section we showed you the basic program skeleton in VENTOS. In this section, we are going to write our first algorithm in VENTOS. Here we will use the SUMO files in Part 2, Example 2.1.1 that is located in 'VENTOS/examples/gettingStarted/1_two_subsequent_streets' folder.

Step 1: We need to call NETCONVERT to generate the network file. Open a terminal and navigate to the VENTOS/examples/gettingStarted/1_two_subsequent_streets folder. Then run NETCONVERT application as below to generate 'example.net.xml' file. Make sure NETCONVERT is somewhere in your PATH.

```
netconvert --node-files=example.nod.xml --edge-files=example.edg.xml --output-
file=example.net.xml
```

Step 2: Go to src/gettingStarted folder and open tutorial.cc source file. Add the following code into the tutorial::executeEachTimestep() method to print the total number of departed vehicles and total number of arrived vehicles at the end of each time step on the output console. We are using static variables to save the value for the next method call.

```
1       static int departedVehs = 0;
2       // get number of departed vehicles in the current time step
3       departedVehs += TraCI->simulationGetDepartedVehiclesCount();
4
5       static int arrivedVehs = 0;
6       // get the number of arrived vehicles in the current time step
7       arrivedVehs += TraCI->simulationGetArrivedNumber();
8
```

```
9     std::cout << "\ntime step: " << omnetpp::simTime().dbl();
10    std::cout << ", departed vehs: " << departedVehs;
11    std::cout << ", arrived vehs: " << arrivedVehs;
12    std::cout << "\n" << std::flush;
```

Step 3: Open the 'omnetpp.ini' file in 'examples/gettingStarted' and change it to the following:

```
include ../omnetpp_general.ini

[Config tutorial1]
description = "writing your first algorithm"

Network.TraCI.active = true
Network.TraCI.SUMOapplication = "sumo-guiD"
Network.TraCI.SUMOconfig = "1_two_subsequent_streets/example.sumocfg"
Network.TraCI.terminateTime = 600s

Network.tutorial.active = true
```

Step 4: To start the simulation, click on Run | Run Configuration and select the configuration you created in Section 3 (tutorial1). Click on Run and then click the red play button  on the toolbar to start the simulation. Simulation is terminated if all the vehicles arrive or maximum simulation time (=600s as specifies using parameter 'terminateTime') is reached.

8.1 Under the Hood

In the previous section, we generated a simple SUMO traffic model and asked VENTOS to load and run it for us. VENTOS opens the TraCI interface automatically and controls the simulation under the hood. We used the TraCI to get information about the simulation (number of departed and arrived vehicles) at each time step. Under the hood, when the simulation starts, the following steps are performed:

- OMNET++ reads the 'src/Network.ned' file and creates the sub-modules that are listed under the 'submodules:' tag, one by one from top to bottom. For each module, OMNET++ invokes the initialize method as shown in the 'log viewer' area in Figure 57[1]. For example when the tutorial module is being initialized (as highlighted in Figure 57) the method tutorial::initialize is invoked (line 13 in tutorial.cc). In this method, we get a pointer to the TraCI class (note that the TraCI interface has not been established yet) and then subscribe to two signals.
- VENTOS forks a new process and runs the SUMO application in TraCI mode. VENTOS also passes the 'example.sumocfg' configuration file to SUMO. This is logged in the output console as shown in Figure 58.
- SUMO reads the configuration file and loads the SUMO network file, SUMO traffic demand file and other optional files if present.
- The TraCI interface is established and VENTOS emits **initializeWithTraCISignal**. Since the module 'tutorial' is subscribed to this signal, tutorial::receiveSignal method is invoked and subsequently tutorial::initialize_withTraCI method is called.

[1] OMNET++ supports multi-stage initialization. You can get more information here.

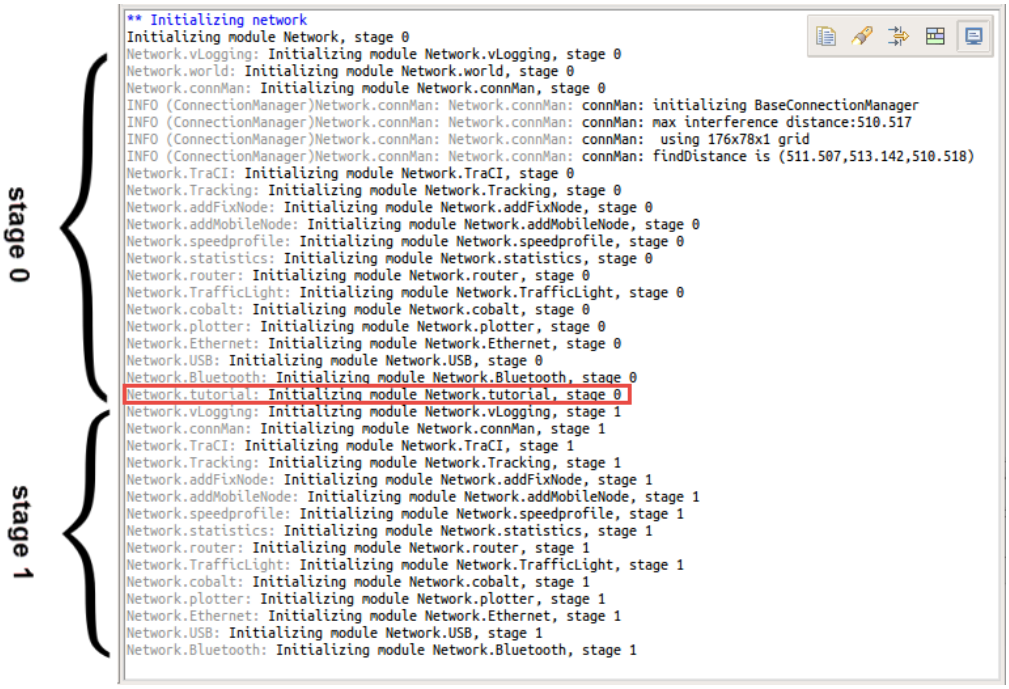


Figure 57: **OMNET++ module 2-stage initialization**

```
>>> Starting SUMO process ...
    Executable file: /home/mani/Desktop/VENTOS_SUMO/bin/sumo-guiD
    Config file: /home/mani/Desktop/VENTOS/examples/v2x/sumocfg/stock.sumo.cfg
    Switches: --start --no-step-log --quit-on-end --seed 0
    TraCI server is listening on port 40674
    SUMO has started successfully in process 4779
    SUMO gui Version dev-SVN-r22948
Loading configuration... done.

>>> Connecting to TraCI server on port 40674 ...
    TraCI server "SUMO dev-SVN-r22948" reports API version 14
    Simulation time step is 0.100000 seconds
    TraCI reports network boundaries (0,0)-(5604.28,3196.46)
    Initializing modules with TraCI support ...
```

Figure 58: **Console window showing SUMO execution and TraCI establishment**

- VENTOS uses a timer to periodically advance SUMO simulation one time step at a time. It then emits **executeEachTimeStepSignal** and since module 'tutorial' is subscribed to this signal, the tutorial::receiveSignal method is invoked and subsequently tutorial::executeEachTimestep method is called[1].
- Once a vehicle is inserted into SUMO, VENTOS creates a mirror module in OMNET++ that follows the mobility of the vehicle in SUMO. Simulation is terminated if all the vehicles arrive or maximum simulation time is reached.
- OMNET++ invokes the finish method in all modules. This method is normally used to record statistics information accumulated in data members of the class at the end of the simulation.

[1] 'tutorial::initialize_withTraCI' method is called only once at the start of simulation when the TraCI interface is established whereas 'tutorial::executeEachTimestep' method is called at each time step.

9 Adding/Removing Nodes to Simulation Dynamically

As we have seen so far, you can add nodes (motor-vehicles, bikes and person) directly from the SUMO traffic demand file (Part 2, Section 3) or you can use the 'addNode' module (Section 3). In either cases, the nodes are added at the beginning of the simulation. In some situations you need to dynamically add/remove nodes. For example, when the number of cars in the simulation reaches 20, then you may insert a bicycle or you may remove a car, when a specific condition happens. TraCI commands can be used to dynamically add/remove nodes into/from the simulation.

Dynamic vehicle insertion is done by calling the 'vehicleAdd' method. This method takes 7 parameters: 'vehicle name'[1], 'vehicle type', 'vehicle route', 'depart time', 'depart position', 'depart speed' and 'depart lane'. Dynamic vehicle removal is done by calling the 'vehicleRemove' method. Persons can be added using 'personAdd' method and should be followed by appending stages or the person will immediately vanish on departure.

Let's write an example that inserts vehicles in SUMO through the TraCI interface.

Step 1: We are going to use the same SUMO files created in Section 8, but with a small modification. Go to 'examples/gettingStarted/1_two_subsequent_streets' folder and open 'example.rou.xml'. Delete the line at the bottom of the file that defines a flow of vehicles in order to prevent SUMO from inserting vehicles into the simulation. We are going to manually insert vehicles in the C++ code.

Step 2: Go to src/gettingStarted folder and open tutorial.cc source file. Add the following code into the tutorial::initialize_withTraCI method in order to add 100 vehicles into SUMO at the beginning of the simulation. This is done by calling vehicleAdd method in a 'for' loop for 100 times. Note that in each loop iteration, only 'vehicle name' and 'depart time' are changed.

```
1    int depart = 0;
2    const int interval = 1000;
3
4    for(int i=1; i<=100; i++)
5    {
6        char vehicleName[90];
7        sprintf(vehicleName, "veh_set1_%d", i);
8
9        TraCI->vehicleAdd(vehicleName, "passenger", "route0", depart, 0
     /*pos*/, 0 /*speed*/, 0 /*lane*/);
10
11       depart += interval;
12   }
```

Step 3: Append the following code into the tutorial::executeEachTimestep() method. When the number of departed vehicles reaches 10, then five red vehicles are inserted into lane 1.

```
1    static bool wasExecuted = false;
2    if(!wasExecuted && departedVehs == 10)
3    {
4        int depart = omnetpp::simTime().dbl() * 1000;
```

[1] 'vehicleName' specifies the vehicle id in SUMO and not OMNET++.

```
5       const int interval = 1000;
6
7       for(int i=1; i<=5; i++)
8       {
9           char vehicleName[90];
10          sprintf(vehicleName, "veh_set2_%d", i);
11
12          TraCI->vehicleAdd(vehicleName, "passenger", "route0", depart,
    0 /*pos*/, 0 /*speed*/, 1 /*lane*/);
13
14          // change color to red
15          RGB newColor = Color::colorNameToRGB("red");
16          TraCI->vehicleSetColor(vehicleName, newColor);
17
18          depart += interval;
19      }
20
21      wasExecuted = true;
22  }
```

Step 4: Use the same configuration in Section 8. To start the simulation, click on Run | Run Configuration and select the run configuration you created in Section 8 (tutorial1). Click on Run and then click the red play button  on the toolbar to start the simulation.

10 Controlling the TraCI connection

VENTOS relies heavily on the Traffic Control Interface (TraCI) to connect to SUMO. The TraCI module in VENTOS (located in traci/modules.ned file) is responsible for opening the SUMO application in a separate process and then establishing the TraCI connection. This module accepts the following parameters:

- **active**: If 'true', VENTOS connects to SUMO and establishes the TraCI. Otherwise, the simulation is performed without SUMO.
- **remotePort**: the TCP port that SUMO TraCI starts listening to. Setting remotePort to -1 causes VENTOS to choose an available/free port.
- **forkSUMO**: If 'true', VENTOS automatically runs the SUMO application in a separate process. Otherwise, VENTOS connects to the external SUMO application (with maximum of 10 retries). This is particularly useful in debugging the SUMO code where you are running the SUMO application inside an IDE as explained in Part 1, Section 7.3.
- **SUMOapplication**: This parameter determines which SUMO application needs to be started and it can be 'sumo', 'sumo-gui', 'sumoD' or 'sumo-guiD'. SUMO applications with letter 'D' at the end are compiled with debug flag. Note that the SUMO application should be in your PATH variable, otherwise VENTOS cannot find it.
- **SUMOconfig**: This parameter points to the SUMO configuration file (with .sumocfg extension). The path is relative starting from the current working directory. SUMO application accepts many input parameters that are all defined here. You can set/unset them in the SUMO configuration file. You can find more information about SUMO configuration file in Part 2, Section 6.1.3.

- **SUMOcommandLine**: In addition to the SUMO configuration file, further SUMO options can also be given on the command line using this parameter. If a parameter is set within the named configuration file as well as given on the SUMOcommandLine, the value given on the command line is used (overwrites the one within the configuration file).
- **terminateTime**: This parameter specifies the maximum simulation time in seconds.
- **autoTerminate**: If 'true', the simulation terminates automatically when all nodes arrive to their destinations and no more nodes are waiting for insertion.
- **equilibrium_vehicle**: if 'true' the arrived vehicles are re-inserted into the simulation in order to achieve an equilibrium in the number of active vehicles in the simulation. In other words, the number of active vehicles in the simulation is always fixed (assuming that no more vehicles are inserted in the middle).
- **record_TraCI_activity**: This parameter allows you to monitor the TraCI messages exchanged between VENTOS and SUMO as explained in Section 29.2.
- **roiRoads**: Setting region of interest (ROI) with roads/edges. More information in Section 20.1.
- **roiRects**: Setting region of interest (ROI) with rectangle(s). More information in Section 20.1.
- **roiRectsRSU**: Setting region of interest (ROI) around RSUs. More information in Section 20.1.

11 VENTOS TraCI APIs

The Veins framework has implemented a C++ TraCI API. VENTOS rewrote the API from scratch and extended the official SUMO TraCI API by providing new methods. The traci/TraCICommands.h file has all the TraCI APIs (a.k.a commands) supported by VENTOS. All related commands are grouped together and you can use the TraCI pointer to invoke them. As shown in the VENTOS program skeleton in Section 7, the TraCI pointer can be obtained using the following line.

```
// get a pointer to the TraCI module
auto TraCI = VENTOS::TraCI_Commands::getTraCI();
```

Thanks to the Eclipse's code completion, once you type `TraCI->`, all available commands are listed as shown in Figure 59.

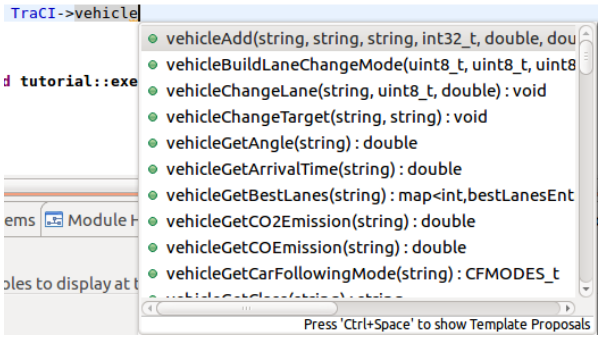


Figure 59: **Code completion in Eclipse IDE**

The explanation for all official TraCI commands can be found here. In the remainder of this section, we will discuss the new TraCI commands added by VENTOS.

Simulation:

`std::string simulationGetStartTime()`
Returns a string in the ''YYYYMMDD-HH:MM:SS" format that shows the simulation start time.

`std::string simulationGetEndTime()`
Returns a string in the ''YYYYMMDD-HH:MM:SS" format that shows the simulation end time. You can call this method in the finish method of your code. Note that while the simulation is running, this method returns an empty string.

`std::string simulationGetDuration()`
Returns a string in the "x hour, y minute, z second" format that shows the total duration of the simulation.

`void simulationTerminate()`
Your program can call this method to terminate the simulation. This method records the simulation ending time and then invokes 'endSimulation()' to notify OMNET++ to finish the simulation.

Vehicle:

`double vehicleGetCurrentAccel(std::string vehID)`
Returns the current acceleration of a vehicle.
`bool vehicleExist(std::string vehID)`
Check if a vehicle exists in the simulation

`double vehicleGetDepartureTime(std::string vehID)`
Returns the departure time of a vehicle. If the vehicle does not exist, -1 is returned.

`double vehicleGetArrivalTime(std::string vehID)`
Returns the arrival time of a vehicle. If the vehicle does not exist or hasn't arrived yet, -1 is returned.

`std::string vehicleGetCarFollowingModelName(std::string vehID)`
Returns the car-following model name of a vehicle. Refer to Table 5 for a list of car-following models in SUMO. Car-following models are discussed in Part 2, Section 4.1. The last four models (OptimalSpeed, KraussFixed, ACC and CACC) are added by VENTOS and you can find more information in Section 24.

Car-following model name	Car-following model ID
KRAUSS	0
KRAUSE_PLUS_SLOPE	1
KRAUSS_ORIG1	2
SMART_SK	3
DANIEL1	4
IDM	5
IDMM	6

Car-following model name	Car-following model ID
PWAGNER2009	7
BKERNER	8
WIEDEMANN	9
OPTIMALSPEED	10
KRAUSS_FIXED	11
ACC	12
CACC	13

Table 5: **Car-following models in SUMO**

`carFollowingModel_t vehicleGetCarFollowingModelID(std::string vehID)`
Returns the car-following model ID of a vehicle.

`int vehicleGetCACCStrategy(std::string vehID)`
Returns the platooning strategy in a CACC system. Refer to Section 24.3 for more information.

```
CFMODES_t vehicleGetCarFollowingModelMode(std::string vehID)
```
Returns the current mode/stage that the ACC/CACC car-following model is currently at. The modes are defined as following. Refer to the platooning paper for more information.

```
typedef enum CFMODES {
    Mode_Undefined,
    Mode_NoData,
    Mode_DataLoss,
    Mode_SpeedControl,
    Mode_GapControl,
    Mode_EmergencyBrake,
    Mode_Stopped
} CFMODES_t;
```

```
std::string vehicleGetPlatoonId(std::string)
```
Returns the platoon id that this vehicle is part of. If the vehicle is not in a platoon, then an empty string is returned.

```
int vehicleGetPlatoonDepth(std::string)
```
Returns the platoon depth of this vehicle. Platoon leader has depth of 0, first followers has depth of 1 and so forth. If the vehicle is not in a platoon, then -1 is returned.

RSU:

```
std::vector<std::string> rsuGetIDList()
```
Returns the list of RSUs in the network.

```
uint32_t rsuGetIDCount()
```
Returns the number of RSUs in the network.

```
TraCICoord rsuGetPosition(std::string rsuID)
```
Returns the x,y SUMO coordinates of an RSU.

Obstacle:

```
std::vector<std::string> obstacleGetIDList()
```
Returns the list of obstacles in the network.

```
uint32_t obstacleGetIDCount()
```
Returns the number of obstacles in the network.

```
TraCICoord obstacleGetPosition(std::string obstacleID)
```
Returns the x,y SUMO coordinates of an obstacle.

```
std::string obstacleGetEdgeID(std::string obstacleID)
```
Returns the edge id that obstacle is currently on it.

```
std::string obstacleGetLaneID(std::string obstacleID)
```
Returns the lane id that the obstacle is currently on it.

```
uint32_t obstacleGetLaneIndex(std::string obstacleID)
```
Returns the lane index that obstacle is currently on it.

```
double obstacleGetLanePosition(std::string obstacleID)
```
Returns the current lane position of an obstacle.

Platoon:

```
std::vector<std::string> platoonGetIDList()
```
Returns the list of platoons in the network. A platoon is uniquely identified by the vehicle id of its platoon leader.

```
uint32_t platoonGetIDCount()
```
Returns the number of platoons in the network.

```
omnetpp::cModule* platoonGetLeaderModule(std::string platoonID)
```
Returns the module object of the platoon leader.

```
uint32_t platoonGetSize(std::string platoonID)
```
Returns the platoon size.

```
uint32_t platoonGetOptSize(std::string platoonID)
```
Returns the optimal platoon size.

```
uint32_t platoonGetMaxSize(std::string platoonID)
```
Returns the maximum platoon size.

```
std::vector<std::string> platoonGetMembers(std::string platoonID)
```
Returns member of this platoon (platoon leader and all followers).

```
bool platoonIsPltMgmtProtActive(std::string platoonID)
```
Is the platoon management protocol running in this platoon?

Route:

```
std::vector<std::string> routeShortest(std::string from_edge, std::string to_edge)
```
Calculates the shortest route between ‘from_edge’ and ‘to_edge’ and returns an edge vector.

Edge:

```
uint32_t edgeGetLaneCount(std::string edgeID)
```
Returns the number of lanes in an edge

```
std::vector<std::string> edgeGetAllowedLanes(std::string edgeID, std::string vClass)
```
Returns all lanes in an edge that allow vehicle class ‘vClass’.

SUMO-OMNET Conversion:

```
std::string convertId_traci2omnet(std::string SUMOid) const
```
Converts SUMO id to OMNET++ id

```
std::string convertId_omnet2traci(std::string omnetid) const
```
Converts OMNET++ id to SUMO id

`Coord convertCoord_traci2omnet(TraCICoord coord) const`
Converts TraCI coordinates to OMNET++ coordinates

`TraCICoord convertCoord_omnet2traci(Coord coord) const`
Converts OMNET++ coordinates to TraCI coordinates

`double convertAngle_traci2omnet(double angle) const`
Converts TraCI angle to OMNeT++ angle (in rad)

`double convertAngle_omnet2traci(double angle) const`
Converts OMNET++ angle (in rad) to TraCI angle

SUMO PATH:

`boost::filesystem::path getFullPath_SUMOApplication()`
Returns the absolute path of the SUMO application specified in 'SUMOapplication' parameter.

`boost::filesystem::path getFullPath_SUMOConfig()`
Returns the absolute path of the SUMO configuration file specified in 'SUMOconfig' parameter.

12 Signals Defined by VENTOS

Simulation signals (or just signals) provide a way of publish-subscribe communication for modules. Signals are emitted on a module or channel, and propagate on the module hierarchy up to the tree. At any level, one may register listeners, that is, objects with callback methods. These listeners will be notified (their appropriate methods called) whenever a signal value is emitted. The result of upwards propagation is that listeners registered at a compound module can receive signals from all components in that submodule tree. You can find more detailed information here.

OMNET++ has two built-in signals, PRE_MODEL_CHANGE and POST_MODEL_CHANGE that are emitted before and after model change and you can find more information here. Veins defines multiple signals, among which, the following two signals are notable:

- mobilityStateChangedSignal: This signal is emitted from the mobility module of a vehicle every time its position is changed.
- parkingStateChangedSignal: This signal is emitted from the mobility module of a vehicle when it goes into/out of parking.

In addition to the above signals, VENTOS has defined the following signals:

- initializeWithTraCISignal: This signal is emitted when the TraCI is established and ready to use.
- executeEachTimeStepSignal: This signal is emitted at the end of each simulation time step.

- vehicleDepartedSignal: This signal is emitted when a vehicle is departed. The signal carries a string value showing the SUMO ID of the departed vehicle.
- vehicleArrivedSignal: This signal is emitted when a vehicle is arrived to its destination. The signal carries a string value showing the SUMO ID of the arrived vehicle.

- vehicleModuleDeletedSignal: This signal is emitted when a vehicle module is deleted. The signal carries an object of type cModule that points to the soon be deleted OMNET++ module.
- vehicleModuleAddedSignal: This signal is emitted when a vehicle module is created. The signal carries an object of type cModule that points to the newly created OMNET++ module.

- personModuleDeletedSignal: This signal is emitted when a person module is deleted. The signal carries an object of type cModule that points to the soon be deleted OMNET++ module.
- personModuleAddedSignal: This signal is emitted when a person module is created. The signal carries an object of type cModule that points to the newly created OMNET++ module.

Note that 'ModuleDeletedSignal' and 'ModuleAddedSignal' might be called several times for each vehicle or person when you have Region of Interest (ROI) in your simulation as described in Section 20.1. At any module, you can subscribe to a signal with 'cComponent::subscribe()' method. For example to subscribe to the 'executeEachTimeStepSignal', you can use the following line in your code as shown previously in program skeleton in Section 7. This will register a listener for the signal at the system module that can receive the signal from the whole simulation.

```
omnetpp::getSimulation()->getSystemModule()->subscribe("executeEachTimeStepSignal", this);
```

Signals are identified by names, but internally numeric signal IDs are used for efficiency. The registerSignal method takes a signal name as parameter, and returns the corresponding simsignal_t value.

```
auto sigEachTS = registerSignal("executeEachTimeStepSignal");
```

In order to receive a signal you need to use the receiveSignal method in your code. Several overloaded receiveSignal methods are defined, one for each data type. Whenever a signal is emitted, the matching receiveSignal method is invoked on the subscribed listeners. The 'executeEachTimeStepSignal' carries a long value (that is not used and can be ignored), thus the overloaded receiveSignal method for long value should be used as following:

```
1   void tutorial::receiveSignal(omnetpp::cComponent *source, omnetpp::simsignal_t
    signalID, long i, cObject* details)
2   {
3       Enter_Method_Silent();
4
5       // if executeEachTimeStepSignal is received
6       if(signalID == sigEachTS)
7       {
8           // do something
9       }
10  }
```

13 Data Logging

Data logging is a means of tracking events when your code runs. Data logging is very useful in figuring out if your code is actually doing what it is supposed to do. It is an important technique for troubleshooting and can save long debugging sessions and dramatically increases the maintainability of your code. Data logging is particularly useful in OMNET++ simulation that often consists of many modules that are simultaneously carrying out different tasks.

The easiest way of data logging in your C++ code is to use the standard output (stdout/stderr). You need to include the 'iostream' header and send output to std::cout or std::cerr. OMNET++ provides its own C++ API for data logging that is controllable from the configuration file and is more convenient to view using Tkenv/Qtenv. You can find more detailed information here and here.

VENTOS also provides its own data logging API. All logging must be categorized into one of the predefined log levels. The assigned log level determines how important and how detailed a log statement is. VENTOS defines the following four log levels which is a simplified version of OMNET++ eight log level.

- **WARNING:** This log level is used when you want to inform the user about anything that can potentially cause oddities, but is automatically recoverable. For example when a frame is received with bit-error, you can record it as a warning log.
- **ERROR:** This log level is used for non-recoverable issues that prevents your application from further operation. For example when a user provides a badly-formatted input then you can record it as an error log and possibly terminate the application execution.
- **INFO:** This log level is used for messages that are most likely important/useful for the users.
- **DEBUG:** This log level is used to log implementation-specific technical details that are most likely important for the developers of the application for debugging purposes.

You need to include the following header to your code:

```
#include "logging/VENTOS_logging.h"
```

Then you can use one of the following macros to log your data at a specific level. Each macro acts like a C++ output stream. This means that you can use the << operator to write to it.

```
LOG_WARNING << "queue is full, discarding packets. \n";
LOG_ERROR << "connection to server is lost! \n";
LOG_INFO << "packet received of size " << pktSize << " KB \n";
LOG_DEBUG << "data logging in DEBUG level \n";
```

You can also use output manipulators such as std::endl and std::flush as following:

```
LOG_WARNING << "this is a sample string" << std::endl;
LOG_WARNING << "this is a sample string" << std::flush;
```

All macros print the data logs into the output stream (stdout) similar to std::cout. The difference is that you have more control over displaying the data log levels. For example you can define the log prefix format using the following parameters:

Parameter	Description
printTimeStep	The simulation time is shown in the log prefix
printLogLevel	The log level is shown in the log prefix
printFileName	The file name that this log message belongs to is shown
printLineNumber	The line number that this log message is called is shown

By default, all of these parameters are set to 'false'. If we enable them all,

```
Network.vLogging.printTimeStep = true
Network.vLogging.printLogLevel = true
Network.vLogging.printFileName = true
Network.vLogging.printLineNumber = true
```

Then the following command:

```
LOG_WARNING << "queue is full, discarding packets. \n";
```

Is shown similar to the following where the log prefix is highlighted:

```
[12.00000000] WARNING tutorial.cc (112): queue is full, discarding packets.
```

The log prefix is printed once for each line. For example, the following command

```
LOG_WARNING << "first line" << std::endl << "second line \n";
```

Prints this:

```
[12.00000000] WARNING tutorial.cc (110): first line
second line
```

You can control if the log messages should be printed in the command-line mode using 'logRecordCMD' parameter. By default, it is set to 'false', but you can change it using the following configuration:

```
Network.vLogging.logRecordCMD = true
```

You can enable global runtime filtering using the 'systemLogLevel' parameter. This parameter allows you specify which log level produce output. It is a bitset with the following additive components. As an example, to print the ERROR logs only, set systemLogLevel to 2. To print the WARNING and ERROR logs, set it to 3 (1+2). To enable all logs, set it to 15 (1+2+4+8). To disable all logs, set it to 0. By default, log filtering is disabled (it is set to 15).

```
1: WARNING  (= WARNING_LOG_VAL)
2: ERROR    (= ERROR_LOG_VAL)
4: INFO     (= INFO_LOG_VAL)
8: DEBUG    (= DEBUG_LOG_VAL)
```

Sometimes the computation of log statement parameters may be very expensive, and thus it must be avoided if possible. In this case, it is a good idea to make the log statement conditional on whether the output is actually being displayed anywhere. The `LOG_ACTIVE(logLevel)` macro returns false when a particular log level is disabled. As an example, in the following code if the DEBUG level is not enabled, then the 'if' body that contains the expensiveComputation method will not be executed.

```
if(LOG_ACTIVE(DEBUG_LOG_VAL))
    LOG_DEBUG << "the result is " << expensiveComputation() << std::endl;
```

Last but not least, you can save the log data into a file at the end of the simulation by setting the parameter 'saveLog2File' to true.

14 GUI-based Data Logging

The data logging described in Section 13 enables the user to log data into one of the four log levels and print it in the standard output. You can enable/disable displaying of log prefix and define log filtering. Printing data logs into the standard output can be messy sometimes. GUI-based data logging solves this problem by using a log window that contains multiple tabs/panes. First, you need to include the following header to your code:

```
#include "logging/VENTOS_logging.h"
```

Then you can use the `GLOG` macro to log your data. This macro acts like a C++ output stream and you can use the << operator to write to it. It takes two string arguments that specify tab name and pane name respectively. For example, the following commands create two tabs: 'tab1' and 'tab2'. The first tab has two panes as shown in Figure 60 and the second tab has only one pane as shown in Figure 61. Note that there is no limit on the number of created tabs/panes.

```
GLOG("tab1", "pane1") << "message in tab1 pane1 \n" << std::flush;
GLOG("tab1", "pane1") << "another message in tab1 pane1 \n" << std::flush;
GLOG("tab1", "pane2") << "message in tab1 pane2 \n" << std::flush;
GLOG("tab2", "pane1") << "message in tab2 pane1 \n" << std::flush;
```

Figure 60: **tab1 contains two panes**

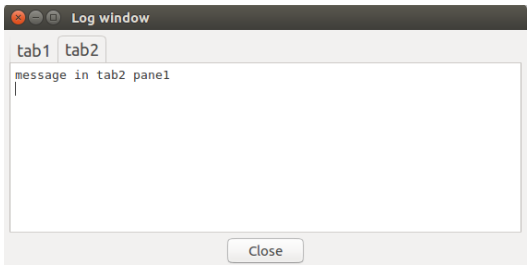


Figure 61: **tab2 contains one pane only**

You can also have multi-line tab names by using the '\n' character. Line 2 and above are shown in blue with smaller font. Moreover, syntax highlighting is active by default and you can display some texts with different fonts and colors in order to improve the readability of the text. The following words are displayed with a different color: 'error' in red, 'warning' in orange and 'ok' in green. As an example the following log message is displayed similar to Figure 62.

```
GLOG("tab1", "pane1") << "error, warning and ok are highlighted \n" << std::flush;
```

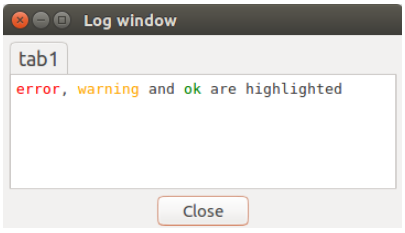


Figure 62

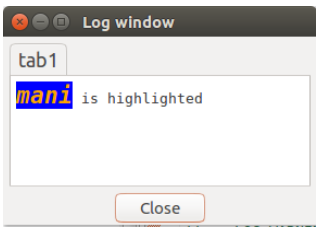


Figure 63

You can define your own syntax highlighting using the 'syntaxHighlightingExpression' parameter that takes the following attributes.

Attribute Name	Value Type	Mandatory?	Description
text	string	Yes	The text that you want to show with a different format (text cannot contain white spaces).
caseSensitive	true/false	No (default: "false")	Is the text case-sensitive
fcolor	string	No (default: "black")	foreground color of the text
bcolor	string	No (default: "blank")	background color of the text
size	integer string	No (default: "normal")	font size of the text
style	string	No (default: "empty")	font style of the text (bold, italic or oblique)

As an example, the following configuration defines a syntax highlighting for text 'mani'. This text is shown with orange foreground color, blue background color, font size 16 and is bold and italic. A sample message is shown in Figure 63. Note that the attribute's value is enclosed in single quotation mark and there is no space before/after the = operator.

```
Network.vLogging.syntaxHighlightingExpression = "text='mani' fcolor='orange'
bcolor='blue' size='16' style='bold,italic'";
```

You can define more than one syntax highlighting syntax using the | operator. For example, the following configuration defines syntax highlighting for 'mani' and 'nima'.

```
Network.vLogging.syntaxHighlightingExpression = "text='mani' fcolor='orange'
bcolor='blue' size='16' style='bold,italic' | text='nima' fcolor='yellow'
bcolor='blue' size='20' style='italic'";
```

15 V2X Communication in VENTOS

Two or more DSRC-enabled nodes in VENTOS can send/receive messages when they are in the radio range of each other. The 'DSRCenabled' parameter in each node is a Boolean value that determines if that node is DSRC-enabled or not. By default, this parameter is 'true' in vehicles, RSUs, pedestrians and bicycles and is 'false' in obstacles. You can overwrite the default value in the configuration file. For example to disable DSRC in V[0] only, use the following command. This means that V[0] cannot receive/send any data over wireless. You will get run-time error if your application tries to send a data in a non-DSRC node.

```
Network.V[0].DSRCenabled = false
```

A circle is drawn around a DSRC-enabled node in OMNET++ Qtenv that shows the maximum interference distance. As an example, we added an RSU (RSU[0]), an obstacle (OBS[0]), an adversary (ADV[0]) and a vehicle (V[1]) into the simulation as shown in Figure 64. The obstacle node has no circle because it is not

DSRC-enabled by default. Note that V[1] and ADV[0] can communication with each other, but none of them can communication with RSU[0], because it is far away. OBS[0] is not able to communicate with any of the nodes since it is not DSRC-enabled.

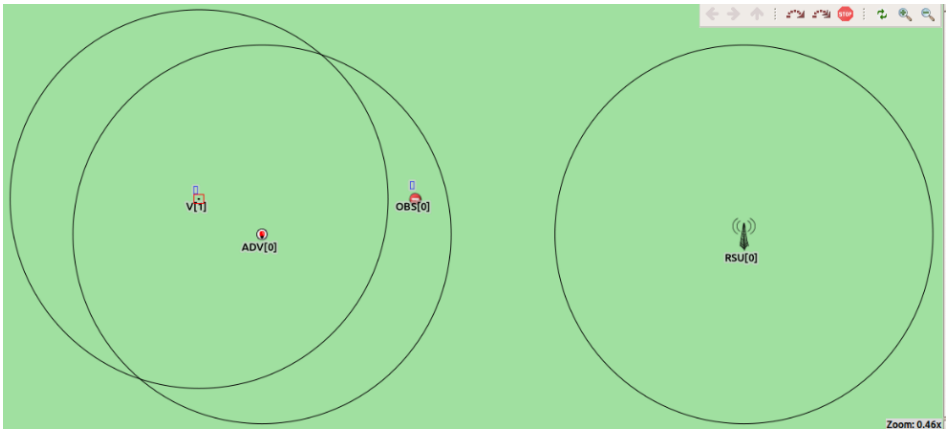


Figure 64: **Circle around a node shows the maximum interference distance**

You can hide the circle by setting the 'drawMaxIntfDist' parameter to 'false' in the configuration file. For example to hide the circle in V[1] use the following command. If you want to hide the circle in all vehicles then replace V[0] with V[*]. Note that this is for visualization purpose only and hiding the circle does not have any effect on the communication.

```
Network.V[1].nic.drawMaxIntfDist = false
```

The maximum interference distance is calculated based on the free pace pass loss and depends on 'pMax', 'sat', 'alpha' and 'carrierFrequency' parameters that are defined in the connMan module. You can find the calculation in src/MIXIM_veins/ConnectionManager.cc file. Based on the following default values, the maximum interference distance is calculated around 510.5 meters. In other words, the radius of the circle is 510.5 meters.

```
# Maximum Tx power possible [mW]
Network.connMan.pMax = 20mW

# Minimum signal attenuation threshold [dBm]
Network.connMan.sat = -89dBm

# Minimum path loss coefficient
Network.connMan.alpha = 2.0

# Minimum carrier frequency of the channel [Hz]
Network.connMan.carrierFrequency = 5.890e9Hz
```

Note that changing the above parameters affects the maximum interference distance in **all** DSRC-enabled nodes. You can manually set your own maximum interference distance by setting the 'maxIntfDist' parameter as shown below that overrides the calculated value above from the free space loss formula.

```
Network.connMan.maxIntfDist = 100
```

16 Wireless Channels in IEEE 802.11p

In October 1999, the United States Federal Communications Commission (FCC) allocated 75 MHz of spectrum in the 5.9 GHz band to be used by intelligent transportation systems (ITS). In August 2008, the European Telecommunications Standards Institute (ETSI) allocated 30 MHz of spectrum in the 5.9 GHz band for ITS.

The DSRC/WAVE standard has defined two types of channels: Control channel (**CCH**) and Service channel (**SCH**). The 75 MHz band is distributed into 1 central Control Channel (CCH) and 6 Service Channels (SCHs) as shown in Figure 65. Service channels are used for non-safety data transfer, while the CCH is used to broadcast safety and control messages (i.e., service advertisement messages).

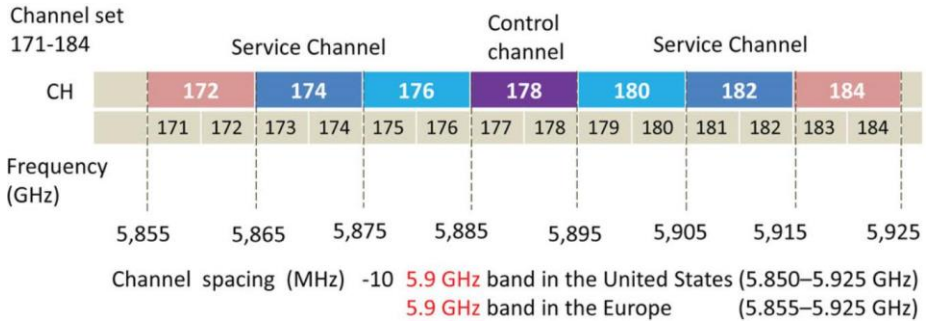


Figure 65: **Distribution of CCH and SCH channels in 75 MHz band**

Seven 10-MHz channels (172, 174, 176, 178, 180, 182, and 184) are defined as in shown Figure 65. The pair of channels (channel 174 and 176, and channel 180 and 182) can be combined to form a single 20-MHz channel, channel 175 and 181 respectively. The channel number (CN) is derived by counting the number of 5-MHz spectrum in the frequency band from 5000 MHz to the center frequency f (CN) of the channel CN, that is:

$$f\ (CN) = 5000\ +\ 5 \times CN \quad (MHz)$$

The following table shows the channel numbers with their respective central frequencies in 5 MHz, 10 MHz and 20 MHz bands.

Channel Numbers in 20 MHz band	Channel Numbers in 10 MHz band	Channel Numbers in 5 MHz band
175 (5875 MHz)	172 (5860 MHz)	171 (5855 MHz)
181 (5905 MHz)	174 (5870 MHz)	172 (5860 MHz)
	176 (5880 MHz)	173 (5865MHz)
	178 (5890 MHz)	174 (5870MHz)
	180 (5900 MHz)	175 (5875 MHz)
	182 (5910 MHz)	176 (5880 MHz)
	184 (5920 MHz)	177 (5885 MHz)
		178 (5890 MHz)
		179 (5895 MHz)
		180 (5900 MHz)

		181 (5905 MHz)
		182 (5910 MHz)
		183 (5915 MHz)
		184 (5920 MHz)

The following table shows the supported data rates in each band. As you can see, higher frequency bands support higher data rates.

Data Rates (Mbps) in 20 MHz band	Data Rates (Mbps) in 10 MHz band	Data Rates (Mbps) in 5 MHz band
6	3	1.5
9	4.5	2.25
12	6	3
18	9	4.5
24	12	6
36	18	9
48	24	12
54	27	13.5

VENTOS/Veins supports the channel numbers / data rates in the 10 MHz band.

17 WSM Transmission and Reception Flow

Transmission and reception flow of a Wave Short Message (WSM) is shown in Figure 66. Application layer and MAC layer are connected directly to each other and can exchange messages using two separate data channels[1]. MAC and PHY layers are also connected using two separate pairs of data and control channels.

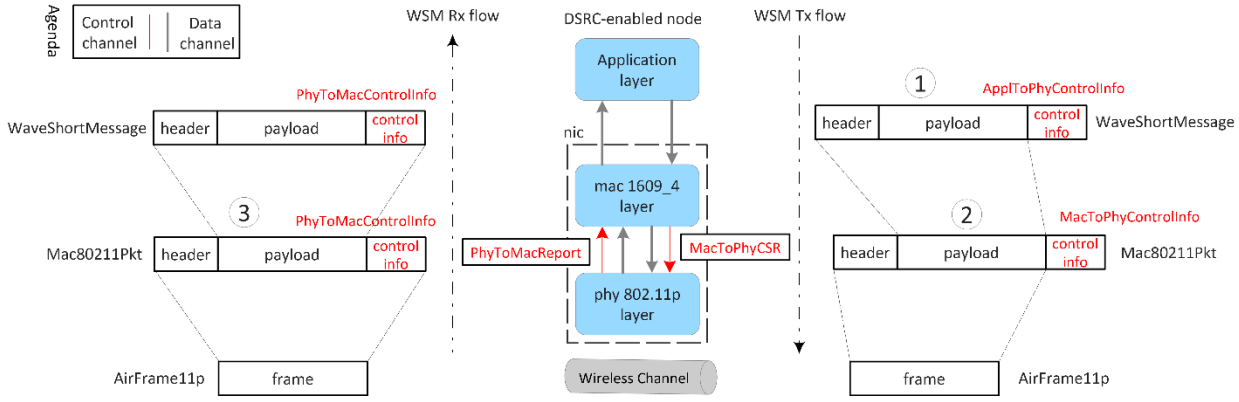


Figure 66: **Transmission and reception flow of a WSM**

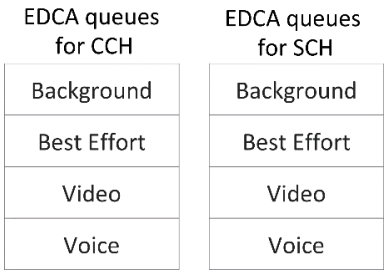
[1] If you need transport layer (TCP/UDP) and network layer (IPv4, IPv6) then the INET project should be added.

17.1 WSM Transmission Flow

The MAC layer implements the EDCA (Enhanced Distributed Channel Access) mechanism for sharing the channel access between multiple stations. Two separate EDCA mechanisms is used for CCH and SCH and each ECDA includes **four** queues with the parameters shown in Table *6*.

Table 6: **EDCA Parameters for each access category**

Access Category	CWmin	CWmax	AIFSN
Background	15	1023	9
Best Effort	15	1023	6
Video	7	15	3
Voice	3	7	2



1. Assume that the application layer generates a Wave Short Message (WSM) as marked with number 1 in Figure 66. A control data of type 'ApplToPhyControlInfo' can be (optionally) attached to the WSM message that specifies transmission power, data rate and modulation/coding scheme. These parameters are later used by the PHY layer. The application layer sends the WSM message down to the MAC layer.

2. In the MAC layer, the WSM message is inserted into one of the queues depending on the priority and the channel type (CCH or SCH). Priority 0, 1, 2 and 3 are mapped to background, best effort, video and voice queue respectively. As long as EDCA queues are non-empty, a WSM message is selected from the queues' front and then encapsulated into a Mac80211Pkt packet as marked with number 2 in Figure 66. A control data of type 'MacToPhyControlInfo' is attached to the MAC packet that contains transmission power, data rate, modulation/coding scheme as well as the active channel's frequency. The resulting MAC frame is sent down to the PHY layer.

3. The PHY layer encapsulates the data into an AirFrame11p frame. The header of the AirFrame11p is used by the receiver to process the received frame and consists of the following fields:

- **signal:** Stores the physical representation of the signal of an air frame such as propagation delay, sender's transmission power, sender's bitrate, etc.
- **state:** This field is used by the receiver's physical layer as state machine for frame processing and can be either START_RECEIVE, RECEIVING, or END_RECEIVE.
- **type:** Specifies the frame's type as either data or control. Value 0 means that the frame is of type data and value 1 means that the frame is of type control where the 'id' field specifies the control type.
- **id:** Unique ID of the AirFrame used as identifier for related control-AirFrames.
- **protocolId:** The id of the PHY protocol of this airframe.
- **channel:** The channel of the radio used for this transmission.
- **duration:** Stores the frame duration (frame transmission time) that is the time needed to send all bits of the frame.
- **sendingTime:** Stores the start time of frame transmission. This field is updated by the sender's node.
- **arrivalTime:** Stores the time when the first bit of this frame is received. This field is updated by the receiver's node.

4. The resulting AirFrame11p is sent on the channel. Note that the PHY layer can transmit only one frame at a time and it is the responsibility of the MAC layer to send the messages one by one in the right order to the PHY layer.

17.2 WSM Reception Flow

Let's go over the WSM reception flow in this section:

1. The AirFrame11p is received by all the nodes that are in the maximum interference distance of the sender and each node processes the received frame independently. The frame is sent to the decider to be decoded by evaluating noise, interference, etc.

2. If the frame's decoding is successful, then the AirFrame11p is de-capsulated and the Mac80211Pkt is extracted. A control data of type 'PhyToMacControlInfo' is attached to the extracted MAC message as marked with number 3 in Figure 66. The control data contains the decider results for the received frame. The MAC layer sends the MAC message directly to the application layer.

3. If frame's decoding is not successful, then the PHY layer informs the MAC layer about the error using a message of type 'PhyToMacReport'. The error can be one of the followings:

BITERROR	The frame is lost due to bit error
COLLISION	The frame is lost due to collision
RECWHILESEND	The frame is lost due to receive while sending
DROPPED	The frame is dropped by the PHY layer

By default, the PHY layer simulation is not active to speed up the simulation. This means that the received frame is always healthy and the de-capsulated message is sent up to the MAC layer without evaluating noise, interference, etc. PHY layer simulation can be activated with 'emulationActive' parameter as discussed in Section 20. Moreover, all frame exchanges can be recorded as explained in Section 29.10.

17.3 PHY Layer Frame Transmission

In the previous sections, we discussed how a WSM is encapsulated/de-capsulated while moving down / up the DSRC/WAVE protocol stack in a node. In this section we will show you how a frame is being transmitted from the PHY layer as illustrated in Figure 67.

- $\mathbf{t_1}$: PHY layer decides to send a frame.
- $\mathbf{t_2}$ (Sending start): After RADIODELAY (= 1 us), frame transmission starts.
- $\mathbf{t_3}$ (Sending end): After DURATION, the last bit of the frame is transmitted and the PHY layer notifies the MAC layer that the frame transmission is over.
- $\mathbf{t_4}$ (Reception start): The first bit of the frame is received after the propagation delay that is the time it takes for a bit to get to the receiver in direct path. Propagation delay is inversely proportional to the distance between the sender and receiver.
- $\mathbf{t_5}$ (Reception end): The last bit of the frame is received after frame DURATION.

Reception start = Sending start + Propagation delay
Reception end = Reception start + Transmission time (DURATION)

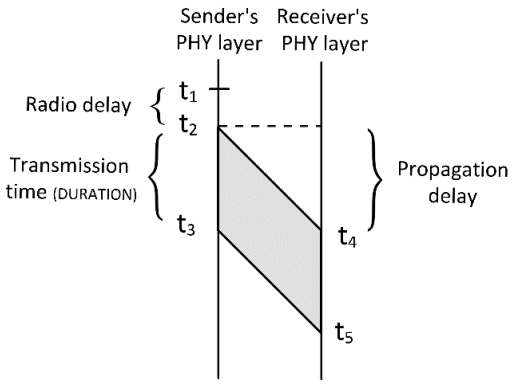


Figure 67: **PHY layer transmission**

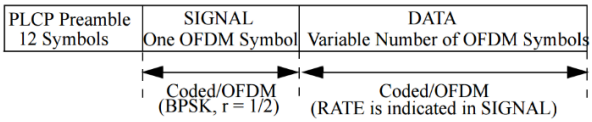


Figure 68: **PHY frame format**

The format of the PHY layer frame is shown in Figure 68. The transmission time (frame duration) is calculated according to equation 17-29 in the IEEE 802.11-2007 standard as following:

$$T_{TR} = T_{PREAMBLE} + T_{SIGNAL} + (T_{SYM} \times N_{SYM})$$

Where, $T_{PREAMBLE}$ denotes the PLCP preamble duration and is 320 ms. T_{SIGNAL} denotes the duration of the signal BPSK-OFDM symbol and is 8 us. T_{SYM} denotes the symbol interval and is 8 us. N_{SYM} denotes the number of OFDM symbols and is calculated according to equation 17-11 in the IEEE 802.11-2007 standard as following:

$$N_{SYM} = ceil(\frac{16 + bitLength + 6}{N_{DBPS}})$$

The bitLength denotes the frame's payload size in bits and N_{DBPS} denotes the number of data bits per OFDM symbol. N_{DBPS} is derived from the data rate as shown in the following table.

Data rates (Mbps)	N_{DBPS}
3	24
4.5	36
6	48
9	72
12	96
18	144
24	192
27	216

The number of bits in the payload shall be multiple of 48, 96, 192 or 288 bits (the number of coded bits in an OFDM symbol). To achieve that, the length of the message is extended so that it becomes a multiple of N_{DBPS}. At least 6 bits are appended to the message, in order to accommodate the TAIL bits as described in section 17.3.5.2 of standard.

17.4 PHY Layer Frame Transmission: An Example

Imagine two cars traveling on a straight highway and one of them broadcasts a frame. The aim is to calculate t_1, t_2, t_3 and t_4 as explained in the previous section. The frame's **payload** size is 43 bytes (=344 bits) and the data rate is 6 Mbps, then the transmission time is calculated as following:

$$T_{TR} = 32\ \mu s + 8\ \mu s + 8\ \mu s \times ceil\left(\frac{16 + 344 + 6}{48}\right) = 104\ \mu s$$

If t_1=38.791734 s then the sending start time is t_2=t_1+RADIODELAY=38.791735 s. Frame transmission ends at t_3=t_1+T_{TR}=38.791734 s+104 us. Assume that the Euclidean distance between the sender and receiver is 9 meters that is calculated from bumper to bumper as shown in Figure 69.

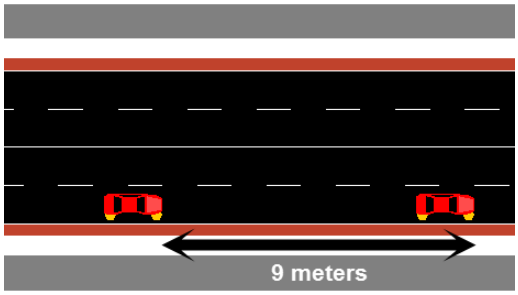


Figure 69: **Euclidean distance between two vehicles**

The propagation delay is calculated as following:

$$T_{PR} = \frac{distance}{speed\ of\ light} = \frac{9\ m}{299792458\ m/s} = 30.021\ ns$$

The first bit of the frame is received after the propagation delay at:

$t_4 = t_2 + T_{PR}$ = 38.791735 s + 30.021 ns = 38.791735030021 s

Simulation of propagation delay in a node can be disabled using the 'usePropagationDelay' parameter. For example, use the following line if V[0] needs to transmit frames with zero propagation delay.

```
Network.V[0].nic.phy80211p.usePropagationDelay = false
```

18 Extracting Control Information from the Received WSM

As explained in Section 17.2, the received AirFrame11p is sent to the decider to be decoded. If decoding is successful, then the MAC layer frame is de-capsulated and a control data of type 'PhyToMacControlInfo' is attached to the extracted MAC frame. The control data contains the decider results for the received frame and includes:

- **Bitrate:** the bit-rate used for frame transmission
- **SNR:** the signal-to-noise ratio of the transmission
- **Received Power:** the received power in dBm (Decibel-milliwatt)

Note that the received power is **not** the RSSI (Received Signal Strength Indication). The RSSI is an indicator of the signal quality and is not standardized and different vendors can define different indicators. The received power value indicates the power that the frame had when received by the NIC (Network Interface Card), without noise floor and without interference.

Also note that the received power is represented in -dBm format that is the power ratio in decibels of the measured power referenced to one milliwatt. For example, 0 dBm equals 1 mW of power, -10 dBm equates to 0.1 mW, -20 dBm equates to 0.01 mW, and so forth. It's a lot easier, and more useful in some calculations, to describe the received power as -100 dBm as opposed to 0.0000000001 mW. The closer the value is to 0, the stronger the signal. For example, -41dBm is better signal strength than -61dBm.

The noise level indicates the amount of background noise in the environment. The higher the noise level, the more likely hood of degraded strength and performance for the wireless signal strength. The noise level is represented in -dBm and the closer the value to 0, the greater the noise level. For example, -96dBm is a lower noise level than -20dBm.

The signal-to-noise ratio (SNR) is the power ratio between the signal strength and the noise level and it is unit-less (when you divide the signal by the noise they have the same units, thus canceling the units out). As an example, if you have a -41 dBm signal strength, and a -50 dBm noise level, this results in a SNR of +9 dBm. Note that subtracting/adding the decibels always corresponds to dividing/multiplying the power ratios and that's why scientists love decibel!

An example of extracting control information from a received WSM message is shown in Section 21.4.

19 Multichannel Operation

As mentioned earlier, 75 MHz of spectrum in the 5.9 GHz band is allocated to one control channel (CCH) and six service channels (SCH). The IEEE 1609.4 standard provides enhancements to the IEEE 802.11 Media Access Control (MAC) to support WAVE operations and has defined four multichannel operation schemes as shown in Figure 70 that specifies how nodes can coordinate safety and non-safety services over a single CCH and the six SCHs.

Continuous access is the most basic scheme that permits a node to always stay tuned to the CCH in order to exchange only safety-related information. Thus this scheme is not appropriate for nodes who are particularly interested in both safety and non-safety applications.

Alternating access alternates between CCH and SCH channels. In this scheme, the channel time is divided into synchronization intervals with a fixed length of 100ms, consisting of a CCH interval and a SCH interval, each of 50 ms. All nodes should tune to CCH during all CCH intervals, where high priority packets and management packets are exchanged. During SCH intervals, nodes can optionally switch to one of six SCHs (negotiated during the CCH interval) in order to perform non-safety applications. This scheme allows node to perform non-safety applications on SCHs without missing important messages on CCH.

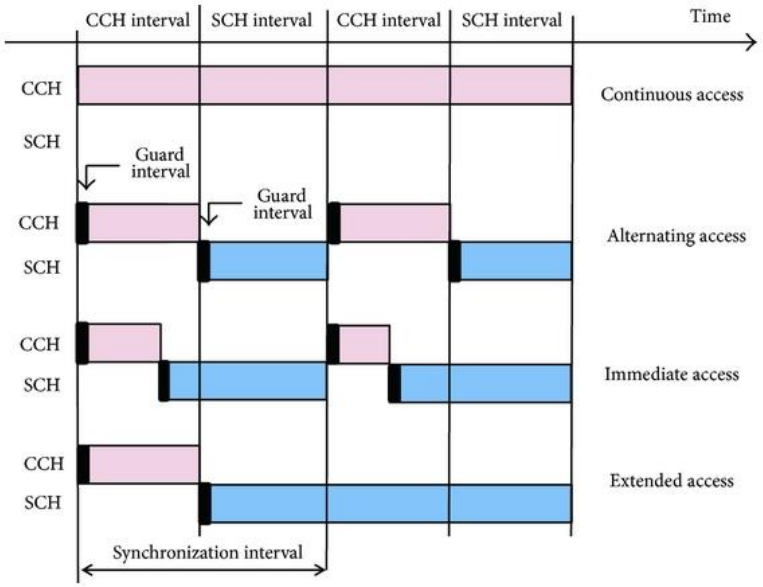


Figure 70: **IEEE 1609.4 multichannel operation strategy**

The black rectangle between each CCH and SCH denotes the guard interval (GI) where radio switching and time synchronization takes place and is between 4 ms to 6 ms. Throughout the GI, nodes are not permitted to send or receive data packets.

The guard interval ensures that the beginning of each and every time interval is well synchronized among the entire common channel. For vehicles to avoid any possible communication delays or losses, IEEE 1609.4 employs an external time reference UTC (Coordinated Universal Time) often provides by GPS to synchronize the nodes. If one of the devices fail to get UTC from its local GPS, then it can contact other nodes over the air in order to ensure it gets synced to other device. IEEE 802.11p specifies something called as WTA (Wave Time Advertisement) frames.

Immediate access is very similar to the alternating access, but it allows nodes to start communications over the SCH immediately without waiting for the next SCH interval.

Extended access is very similar to the alternating access, but it allows nodes to keep communicating over the SCH without switching to the CCH.

Together immediate and extended access schemes are basically intended to improve the performance of applications like video streaming, map updates etc. that consume extensive bandwidth. Nevertheless, these two schemes could be only favorable to those node that are not interested in safety applications.

VENTOS/Veins supports continuous and alternating access only. If 'useServiceChannel' parameter is false then the MAC layer uses continuous access; otherwise the MAC layer uses alternating access. We will present an example in Section 21.2.

20 PHY Layer Simulation

'emulationActive' is false by default!
Explain what happens when this parameter is on.

20.1 Region of Interest (ROI)

Physical layer simulation is a CPU-intensive task, especially in large scale networks with many DSRC-enabled nodes. If you are want to perform the simulation for a small area of your network, then defining one or more region of interests can speed up your simulation significantly. The following three parameters can be used to define a region of interest.

```
Network.TraCI.roiRects
Network.TraCI.roiRectsRSU
Network.TraCI.roiRoads
```

The 'roiRects' is a string value that defines one or more region of interests in **SUMO coordinates**. For example the following line defines a rectangle with **bottom left** and **top right** coordinates of (0,0) and (10,10) respectively.

```
Network.TraCI.roiRects = "0,0-10,10"
```

Multiple rectangles can also be defined using space as separator:

```
Network.TraCI.roiRects = "0,0-10,10 20,20-30,30"
```

The 'roiRectsRSU' is a string that defines a region of interest for each RSU in the network. The rectangle is centered on the RSU and has the `"x,y"` format, where x is the length and y is the width of the region. This parameter makes your life easier when you want to perform PHY simulation near RSUs.

```
Network.TraCI.roiRectsRSU = "0,10"
```

Last but not least, the 'roiRoads' parameter defines the roads (edges) that are considered to constitute the region of interest. Note that the edge ids should exist in the network, otherwise you will get error.

```
Network.TraCI.roiRoads = "hwy1 hwy2"
```

21 V2X Communication: Example

You can find an example in ‘examples/v2x’ folder that shows you how to setup a simple network consisting of DSRC-enabled vehicles and RSUs and how V2V and V2I communication is used to exchange messages between different nodes. In this example, we exported a small region of the Stockholm city using OpenStreetMap and converted it into SUMO network file as shown in Figure 71. You can find more information in Part 2, Section 2.3.

In the project explorer pane, go to ‘examples/v2x’ folder and open addNode.xml. This XML file tells VENTOS what nodes should be added to the network at the beginning of the simulation and you can find more information in Section 3. We have added six RSUs in different locations as shown with green circles in Figure 71. We have also added three vehicles: veh0, veh1 and veh2 with (Origin1, Destination1) pair and a vehicle flow called ‘flow1’ consisting of 5 vehicles with (Origin2, Destination2) pair.

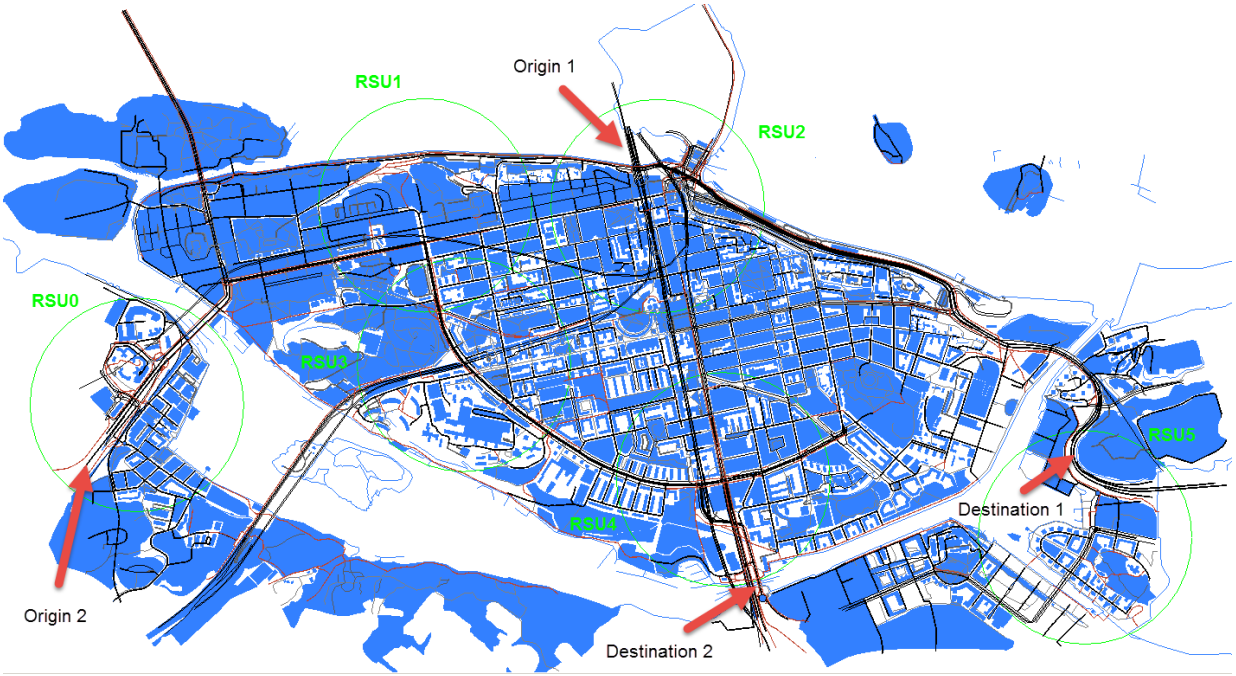


Figure 71: **SUMO network in V2X example with six RSUs and two OD pairs**

Click Run in the menu bar and select ‘Run Configurations…’. Choose 'OMNET++ Simulation' and click on 'New launch configuration' button at the top left. Give this configuration a name like v2x. In ‘Executable’, choose opp_run and set ‘Working dir’ to /VENTOS/examples/v2x. In 'Ini file(s)' choose omnetpp.ini. From 'Config name' choose V2X_Example from the drop down list. Leave the rest of the options to default. Click Apply and then click Run.

OMNET++ Qtenv graphical runtime environment and SUMO-GUI window appear. You can find more detailed information such as the SUMO PID as well as TraCI server API version in the IDE console tab as explained in Section 8.1. The AddNode module shows that 6 RSUs, 3 vehicles and 1 vehicle flow are added into the simulation and presents a summary at the end. The summary includes all defined routes, vehicle types and loaded vehicle count. Click on the red play button on the Qtenv’s toolbar to start the

simulation. The simulation terminates after all vehicles reach to their destinations. In SUMO-GUI, you can locate the vehicles in the network by going to 'Locate | Locate Vehicles' in the menu bar.

You can configure the simulation using SUMO configuration file (stock.sumo.cfg) as well as OMNET++ configuration file (omnetpp.ini). Open the SUMO configuration file located in 'examples/v2x/sumocfg'. SUMO network and traffic demand files are specified as input and the simulation time step is set to 0.1s. You can find more information in Part 2, Section 6.1.3.

Open the OMNET++ configuration file, omnetpp.ini, located in 'examples/v2x' that allows you to configure many aspects of the simulation. Let's go over the OMNET++ configuration file in this example. The TraCI module is responsible for establishing the TraCI interface and accepts many parameters as discussed before in Section 10.

```
Network.TraCI.active = true
Network.TraCI.SUMOconfig = "sumocfg/stock.sumo.cfg"
Network.TraCI.terminateTime = 800s
```

VENTOS provides the 'addNode' module to help you add different node types into the simulation easily. All nodes are defined in the 'addNode.xml' file located in the same folder as omnetpp.ini configuration file. The addNode module reads this file looking for the addNode XML element with id "**add_0**" and then decides which nodes should be inserted at the beginning of the simulation. You can find more information in Section 3.

```
Network.addNode.id = "add_0"
```

The following configuration turns the beaconing on for all vehicles and sets the beaconing interval to 0.1s. The beacon's message format is defined in 'src/msg/BeaconVehicle.msg' file. You can add your own fields into the beacon format, but be cautious when removing the existing fields because some examples rely on them. You can find the beaconing code in 'src/nodes/vehicle/01.Beacon.cc' file.

```
# turn on beaconing in vehicles
Network.V[*].appl.sendBeacons = true
Network.V[*].appl.beaconInterval = 0.1s
```

Similarly, the following configuration turns the beaconing on for all RSUs and sets the beaconing interval to 1s. The beacon's message format is defined in 'src/msg/BeaconRSU.msg' file. You can add your own fields into the beacon format, but be cautious when removing the existing fields because some examples rely on them. You can find the beaconing code in 'src/nodes/rsu/01.Beacon.cc' file.

```
# turn on beaconing in RSUs
Network.RSU[*].appl.sendBeacons = true
Network.RSU[*].appl.beaconInterval = 1s
```

The following configuration sets the 'record_stat' parameter in all vehicles to 'true'. VENTOS records the data listed in the 'record_list' parameter and saves the output in the 'results/000_vehicleData.txt' file. You can find more information in Section 29.3.

```
Network.V[*].record_stat = true
Network.V[*].record_list = "vehId | lanePos | speed | accel"
```

The ‘record_frameTxRx’ parameter in the ‘nic’ module of each DSRC-enabled node enables you to track each frame from the sender to all receivers. The result is saved in the ‘results/000_FrameTxRxdata.txt’ file. You can find more information in Section 29.10. You can also record many different aspects of the simulation as explained in Section 29.

```
Network.V[*].nic.phy80211p.record_frameTxRx = true
Network.RSU[*].nic.phy80211p.record_frameTxRx = true
```

21.1 Sending Data over CCH

In this section, we are going to change the previous example in order to make vehicles broadcast data packets whenever they receive a beacon from an RSU. Beacon and data packets are all broadcasted on the Control Channel (CCH). We will walk you through the changes that we have already made.

1. Create a new message for data packets. Right click on the ‘src/msg’ folder and then click 'New | Message Definition (msg)'. Give the message a name such as ‘dataMsg.msg’ and click next. Select ‘Empty Message File’ and then click Finish.

2. Open the ‘src/msg/dataMsg.msg’ file and define your data packet following the guideline described in the OMNET++ manual here. Your message should be inherited from the WaveShortMessage class, otherwise the MAC layer does not accept this message for transmission. We have added two fields into the dataMsg, a string and an integer as shown below. During compilation, OMNET++ invokes the message compiler (opp_msgc) to translate the ‘dataMsg.msg’ file into C++ code (.h and .cc).

```
cplusplus {{
    #include "WaveShortMessage_m.h"
}}

class Veins::WaveShortMessage;
namespace VENTOS;

packet dataMsg extends WaveShortMessage
{
    string sender;
    int target;
};
```

3. Open the ‘04_BaseWaveApplLayer.h’ file located in the src/baseAppl folder and add a new entry such as TYPE_GENERIC_DATA to the WaveApplMessageTypes enum. TYPE_GENERIC_DATA will later serve as the message kind.

4. Go to the nodes/vehicle folder and open ‘myCode.h’ file. Notice that the following header is added to the top of the file.

```
#include "msg/dataMsg_m.h"
```

5. Open the ‘myCode.cc’ file and go to method ‘onBeaconRSU’. This method is called upon reception of a beacon from an RSU. The received beacon is passed to the method as a pointer of type BeaconRSU. Every time an RSU beacon is received in a vehicle, we generate a data packet by calling generateData method and then send it to the lower layer (MAC layer) using the following command:

```
send(data, lowerLayerOut);
```

6. Look at how generateData() method generates a WSM (Wave Short Message). It first creates a new object of type dataMsg with message name ‘genericData’ and message kind TYPE_GENERIC_DATA.

```
dataMsg* data = new dataMsg("genericData", TYPE_GENERIC_DATA);
```

Message name is a string that is displayed at many places in the graphical runtime interface, so it is generally useful to choose a descriptive name. Message kind is an integer field. Some negative values are reserved by the simulation library, but zero and positive values can be freely used in the model for any purpose. Message kind is typically used to carry a value that conveys the role, type, category or identity of the message.

7. The next step is to fill the dataMsg header and payload fields. The important header fields are channel number and message priority. The channel number can be any of the following values (one channel for CCH and five channels for SCH) in the 10 MHz band as shown before in Figure 65. In this example beacons and data packets are sent on the CCH channel.

```
enum ChannelNumber {
       CRIT_SOL = 172, // 5.86e9 GHz, Critical Safety of Life
       SCH1 = 174,     // 5.87e9 GHz
       SCH2 = 176,     // 5.88e9 GHz
       CCH = 178,      // 5.89e9 GHz
       SCH3 = 180,     // 5.90e9 GHz
       SCH4 = 182,     // 5.91e9 GHz
       HPPS = 184      // 5.92e9 GHz, High Power Public Safety
};
```

The message priority is a number in the range [0,3] and determines how urgent a message is. Priority 0 translates to background traffic, priority 1 translates to best effort traffic, priority 2 translates to video traffic and priority 3 translates to voice traffic. Each WSM message is placed in the corresponding queue based on its priority as discussed previously in Section 17.1.

8. Until now, we generate a data packet called ‘data’ of type dataMsg and filled it in, but the packet size is zero. A packet of size zero makes the simulation unrealistic, since it is used in modeling the transmission time, propagation delay, collision, etc. You need to use addBitLength or addByteLength method in order to change the packet length by the given value. Every time you call these methods, the packet length is appended to the current packet size.

9. Add the following line to the previous ‘V2X_Example’ configuration:

```
# vehicles send data upon reception of RSU beacons
Network.V[*].appl.sendingData = true
```

10. Run the simulation and make sure everything is working as expected. Figure 72 shows a small part of the frame transmission and reception output where RSU[2] broadcasts a beacon message around 5.89s that is first received by V[1] (that is closer to RSU[2]) and then V[2]. Vehicle V[1] broadcasts a data packet in response that is received by both V[2] and RSU[2] in error (look at the last column that shows frame reception status). Then V[2] broadcasts a data packet and is received by both V[1] and RSU[2] in error.

1823	beaconRSU	RSU[2]	V[1]	58	5.89178667	390	6.00
1822	beaconRSU	RSU[2]	V[2]	64	5.89178667	390	6.00
1841	genericData	V[1]	V[2]	64	5.89195188	390	6.00
1843	genericData	V[1]	RSU[2]	28	5.89195188	390	6.00
1842	genericData	V[2]	V[1]	58	5.89195204	390	6.00
1844	genericData	V[2]	RSU[2]	28	5.89195204	390	6.00
1864	beaconVehicle	V[2]	V[1]	58	5.96632004	390	6.00
1865	beaconVehicle	V[2]	RSU[2]	28	5.96632004	390	6.00

Figure 72: **Simulation output in the 000_FrameTxRxdata.txt file**

Question: Can you figure out why the data frame broadcasted by V[1] and V[2] are both received in error?

Answer: Look at the sending time of the data frames. Vehicle V[1] and V[2] broadcast the data packet almost instantaneously that causes both frames to get corrupted. You can fix this issue by introducing a random delay before broadcasting of a data frame using the sendDelayed() method below:

```
sendDelayed(data, smallRandomDelay, lowerLayerOut);
```

21.2 Sending Data over SCH

In the previous example, we showed you how to make vehicles to send data packets over the CCH channel. The CCH is used to broadcast safety and control messages while the SCH is mostly used for non-safety data transfer. In this section, we are going to modify the previous example in order to send data frames in a service channel.

1. Open 'yourCode.cc' file located in the 'src/nodes/vehicle' folder and go to generateData() method. Change the following line:

```
data->setChannelNumber(Veins::Channels::CCH);
```

To this:

```
data->setChannelNumber(Veins::Channels::SCH1);
```

With this, we are asking the MAC layer to send the data packet to the SCH1 channel that corresponds to channel number 174.

2. Then you need to set the 'useServiceChannel' to 'true' in vehicles. This will change the MAC layer access scheme from **continuous** to **alternating** as discussed in Section 19.

```
Network.V[*].nic.mac1609_4.useServiceChannel = true
Network.RSU[*].nic.mac1609_4.useServiceChannel = true
```

The MAC layers in all nodes tune to the CCH channel (with a small offset) and switch to the SCH channel after 50 ms. The channel alternates between CCH and SCH every 50 ms. Figure 73 shows channel switching in the 6 RSUs. The offset is chosen randomly from (0, 0.0003] seconds. The maximum offset used by the MAC layer in each node is controllable using the 'syncOffset' parameter:

```
Network.V[*].nic.mac1609_4.syncOffset = 0.0003s
```

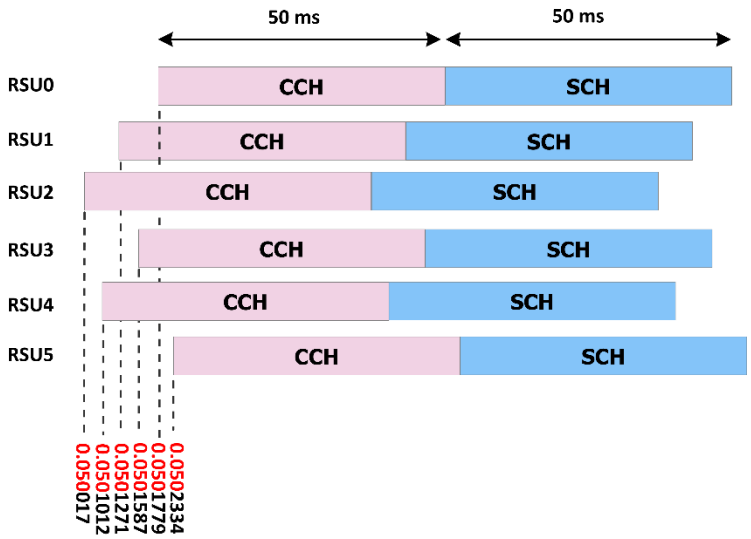


Figure 73: **Alternate channel switching scheme in RSUs**

3. Run the simulation and make sure everything is working as expected. Figure 74 shows a small part of the frame transmission and reception output. As you can see the ChannelNum column shows 174 for data frames and 178 for beacons.

```
genericData        V[3]              V[0]              52              10.65413489          390              174
genericData        V[3]              RSU[0]            16              10.65413489          390              174

genericData        V[0]              V[3]              70              10.65429936          390              174
genericData        V[0]              RSU[0]            16              10.65429936          390              174

beaconVehicle      V[2]              V[4]              76              10.72545063          390              178
beaconVehicle      V[2]              V[1]              58              10.72545063          390              178
beaconVehicle      V[2]              RSU[2]            28              10.72545063          390              178
```

Figure 74: **Simulation output in the 000_FrameTxRxdata.txt file**

21.3 Changing the Transmission Power and Data Rate

As explained in Section 17.1, the application layer can attach a control data of type 'ApplToPhyControlInfo' to the WSM message to specify the transmission power, data rate and modulation/coding scheme. This means that you can change the Tx power and data rate of each individual WSM message. If no control information is attached to the WSM message, then the values are read from the user configuration as shown below.

```
Network.V[*].nic.mac1609_4.bitrate = 4 Mbps
Network.V[*].nic.mac1609_4.txPower = 10 mW
```

If the user has not specified any values then Tx power of 20 mW and data rate of 6 Mbps are chosen by default by the MAC layer. In this section we will show you how to attach a control information to a WSM message in order to set Tx power and data rate.

1. Go to the 'yourCode.cc' located in the 'nodes/vehicle' folder and find the generateData() method. Add the following code before the 'return' statement.

```
1   // create a new control information
2   VENTOS::ApplToPhyControlInfo *controlInfo = new VENTOS::ApplToPhyControlInfo();
3
4   // set Tx power and modulation (bitrate)
5   controlInfo->setTxPower_mW(10);
6   controlInfo->setMcs(Veins::MCS_OFDM_QPSK_R_3_4);
7
8   // attach control information to the data packet
9   data->setControlInfo(controlInfo);
```

In line 2, we have instantiated object 'controlInfo' and then set the Tx power and modulation scheme in line 5 and 6 respectively. Finally, we attach the controlInfo to the data packet. Supported modulation schemes are listed in the following table.

Modulation Scheme	Bitrate (Mbps)
MCS_OFDM_BPSK_R_1_2	3
MCS_OFDM_BPSK_R_3_4	4.5
MCS_OFDM_QPSK_R_1_2	6
MCS_OFDM_QPSK_R_3_4	9
MCS_OFDM_QAM16_R_1_2	12
MCS_OFDM_QAM16_R_3_4	18
MCS_OFDM_QAM64_R_2_3	24
MCS_OFDM_QAM64_R_3_4	27

2. Make sure 'sendingData' is 'true' in the 'V2X_Example' configuration:

```
Network.V[*].appl.sendingData = true
```

3. Run the simulation and make sure everything is working as expected. Figure 75 shows a small part of the frame transmission and reception output. As you can see the Tx power and bitrate of data packets are changed to 10 mW and 9 Mbps respectively. Beacons are still using the default 20 mW and 6 Mbps for their Tx power and bitrate.

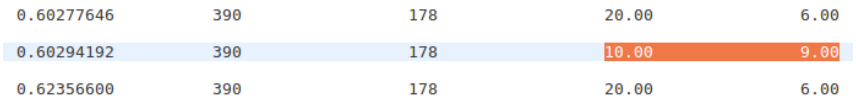


Figure 75: **Simulation output in the 000_FrameTxRxdata.txt file**

21.4 Extracting Control Information: An Example

Section 18 described what type of control information can be extracted from a received WSM message. In this section, we show you an example that is located in the 'examples/receivedSignal' folder. Two vehicles are constantly beaconing with interval of 0.1s. They are approaching to each other from adjacent lanes as shown in Figure 76. There is no V2V communication between the vehicles while they are far away, but they can exchange data once they enter into each other's interference distance.

Figure 76: **Two vehicles are approaching to each other from adjacent lanes**

As soon as a WSM message is received by the application layer of a vehicle, its control information is extracted and are shown in the GUI log window. As vehicles become closer to each other, the 'received power' values increases (approach to 0) that subsequently increases the SNR values.

Also have a look at the '000_PHYdata.txt' output file as shown in Figure 77. Both vehicles broadcast 383 frames since the network is symmetric. While the vehicles are in the radio range of each other, they each receive 170 frames where they are decoded correctly.

vehicleName	lastStatTime	NumSentFrames	NumReceivedFrames	NumLostFrames_BiteError	NumLostFrames_Collision	NumLostFrames_TXRX
V[0]	38.26028128	383	170	0	0	0
V[1]	38.26235958	383	170	0	0	0

Figure 77: **Simulation output in the 000_PHYdata.txt file**

21.5 V2V Communication: Send/Receive Example

In Section 21, we showed you how to setup an example with vehicles and RSUs where all nodes broadcast beacon periodically. In Section 21.1 and 21.2 we changed the example in order to make vehicles send data packets over CCH and then SCH channel whenever they receive a beacon from an RSU. In Section 21.3, we showed you how to change the Tx power and bit rate of each individual WSM message. In Section 21.4 we focused on the receiver and showed you how to extract useful control information from every received WSM message such as the bit rate, received power and SNR.

Many different parameters can affect frame transmission and in this section, we'll go over useful scenarios that show the effect of changing these parameter on frame transmission. We'll use the same example in Section 21.4 located in the 'examples/receivedSignal' folder where two vehicles are approaching to each other. The default Tx power and bitrate are 20 mW and 6 Mbps respectively.

Scenario 1

In the first scenario, we are going to turn off the 'avoidBeaconSynchronization' using the following line:

```
Network.V[*].appl.avoidBeaconSynchronization = false
```

Run the simulation with configuration 'Scenario_1' and check the '000_PHYdata.txt' output file. As shown in Figure 78, the number of broadcasted frames from each vehicle has not changed, but all WSM frames are received with error (RECWHILESEND). This is due to the fact that both vehicles broadcast beacon at the exact same time that corrupt both frames. When 'avoidBeaconSynchronization' parameter is 'true', a very small random offset is added to the first beacon broadcast in a node to prevent artificial synchronization between beacons broadcast in different nodes.

vehicleName	lastStatTime	NumSentFrames	NumReceivedFrames	NumLostFrames_BiteError	NumLostFrames_Collision	NumLostFrames_TXRX
V[0]	38.20000263	383	0	0	0	170
V[1]	38.20000263	383	0	0	0	170

Figure 78: **Simulation output in the 000_PHYdata.txt file**

Scenario 2

In this scenario, the maximum interference distance is reduced to 20 meters using the following line:

```
Network.connMan.maxIntfDist = 20
```

Run the simulation with configuration 'Scenario_2' and check the simulation output files. Since the maximum radio range of all nodes are lowered, then vehicles are in contact with each other in a smaller time window and each vehicle receives lower number of beacons.

Scenario 3

The 'thermal noise' is the only noise that is simulated in the framework. This means that setting the 'useThermalNoise' parameter to 'false' as shown below, causes all received SNR to infinity.

```
Network.V[*].nic.phy80211p.useThermalNoise = false
```

In this scenario, we increase the thermal noise from the default value of -110 dBm to -90 dBm:

```
Network.V[*].nic.phy80211p.thermalNoise = -90dBm
```

Run the simulation with configuration 'Scenario_3' and check the '000_PHYdata.txt' output file. As shown in Figure 79, the number of broadcasted frames from each vehicle has not changed, but around 40% of the frames (~68 out of 170) are received in error. This is due to the increased thermal noise in the wireless channel that makes some of the frame un-decodable.

vehicleName	lastStatTime	NumSentFrames	NumReceivedFrames	NumLostFrames_BiteError	NumLostFrames_Collision	NumLostFrames_TXRX
V[0]	38.26028128	383	102	68	0	0
V[1]	38.26235958	383	103	67	0	0

Figure 79: **Simulation output in the 000_PHYdata.txt file**

Scenario 4

The default sensitivity level of the physical layer is -89 dBm that is equal to the following milliwatt:

$$-89\ dBm = 10\ \times \log_{10}\frac{x}{1\ mW} \rightarrow x = 10^{\frac{-89}{10}}\ mW = 10^{-8.9}\ mW = 1.2589 \times 10^{-9}\ mW$$

If the received power is lower than $1.2589 \times 10^{-9}\ mW$, then the receiver cannot recognize the frame. In this scenario, we change the sensitivity of the physical layer from the default value of -89 dBm to -60 dBm. Basically, we are making the physical layer less sensitive. In other words, the received signal power should be more powerful to be detected by the physical layer in the receiver.

Run the simulation with configuration ‘Scenario_4’ and check the ‘000_PHYdata.txt’ output file. As shown in Figure 80, the number of received frames are much lower. Since we made the physical layer less sensitive, the vehicles should be closer to each other so that the received power is high enough in order to be detectable by the physical layer.

vehicleName	lastStatTime	NumSentFrames	NumReceivedFrames	NumLostFrames_BiteError	NumLostFrames_Collision	NumLostFrames_TXRX
V[0]	38.26028125	383	6	0	0	0
V[1]	38.26236115	383	6	0	0	0

Figure 80: **Simulation output in the 000_PHYdata.txt file**

Scenario 5

Until now, all the beacon messages were broadcasted on the control channel (CCH). In this scenario, vehicle V[1] broadcasts the beacons on SCH1 while V[0] is still using the CCH for beacon broadcast. Go to ‘src/nodes/vehicle’ and open the 01_Beacon.cc source file. Look for the generateBeacon() method and change the following line:

```
wsm->setChannelNumber(Veins::Channels::CCH);
```

To

```
if(SUMOID == "veh0")
    wsm->setChannelNumber(Veins::Channels::CCH);
else
    wsm->setChannelNumber(Veins::Channels::SCH1);
```

Run the simulation with configuration ‘Scenario_5’ and check the ‘000_PHYdata.txt’ output file. Vehicle V[0] received all beacons from V[1] and vehicle V[1] received all beacons from V[0]. This is because, the ‘useServiceChannel’ parameter is set to ‘true’ in both vehicles that enables alternating channel scheme. Both vehicles alternate between CCH and SCH channels every 50 ms and they can receive WSM messages from both CCH and SCH channels.

When you are done with this scenario, do not forget to revert back the changes.

Scenario 6

In this scenario, we are going to lower the transmission power of vehicle V[1] from the default value of 20 mW to 1 mW. Go to the ‘src/nodes/vehicle’ folder and open the 01_Beacon.cc source file. Look for the generateBeacon() method and append the following lines before the return statement:

```
if(SUMOID == "veh1")
{
    // create a new control information
    VENTOS::ApplToPhyControlInfo *controlInfo = new VENTOS::ApplToPhyControlInfo();
    // set Tx power
    controlInfo->setTxPower_mW(1);
    // attach control information to the WSM message
    wsm->setControlInfo(controlInfo);
}
```

Run the simulation with configuration ‘Scenario_6’ and check the ‘000_PHYdata.txt’ output file. As shown in Figure 81, vehicle V[0] has received much lower WSM frames than V[1], because transmission power of beacon messages broadcasted from V[1] is lowered to 1 mW. Vehicles should be closer to each other in order for V[1]’s messages to be decodable in V[0].

vehicleName	lastStatTime	NumSentFrames	NumReceivedFrames	NumLostFrames_BiteError	NumLostFrames_Collision	NumLostFrames_TXRX
V[0]	38.26029051	383	38	0	0	0
V[1]	38.26236421	383	170	0	0	0

Figure 81: **Simulation output in the 000_PHYdata.txt file**

When you are done with this scenario, do not forget to revert back the changes.

Scenario 7

This scenario is similar to the previous scenario, but instead of changing the Tx power in V[1], we are going to change the bitrate in V[1] from the default value of 6 Mbps to 24 Mbps. Go to the ‘src/nodes/vehicle’ folder and open the 01_Beacon.cc source file. Look for the generateBeacon() method and append the following lines before the return statement:

```
if(SUMOID == "veh1")
{
    // create a new control information
    VENTOS::ApplToPhyControlInfo *controlInfo = new VENTOS::ApplToPhyControlInfo();
    // set modulation (bitrate)
    controlInfo-> setMcs(Veins::MCS_OFDM_QAM64_R_2_3);
    // attach control information to the WSM message
    wsm->setControlInfo(controlInfo);
}
```

Run the simulation with configuration ‘Scenario_7’ and check the ‘000_PHYdata.txt’ output file. This time all the broadcasted beacons from each vehicle are received in the other vehicle, but the broadcasted beacons from vehicle V[1] have lower transmission time. This means that the physical layer of vehicle V[1] transmits bits of each frame with higher speed that decreases the transmission time. Note that the propagation time of bits will not get affected.

22 Analog model

23 Get Real-Time PHY layer Statistics

As discussed in Section 29.10, you can ask VENTOS to collect different physical layer statistics by setting the ‘record_stat’ parameter in a node. These statistics are written to a file (xxx_PHYdata.txt) at the end of the simulation. Sometimes, your program needs to access these statistics in real-time while the simulation is running. You can achieve this by getting a pointer of the PHY module and then access

24 Car-following Models Defined by VENTOS

Car-following models in SUMO are described in Part 2, Section 4.1. This document describes how to add a new car-following model into SUMO. VENTOS has added the following new models.

Car-following Name	Attribute	Description
KRAUSS_FIXED	KraussFixed	Fixes a bug in the original Krauß-model
OPTIMAL_SPEED	OptimalSpeed	Optimal speed
ACC	ACC	Adaptive Cruise Control
CACC	CACC	Cooperative Adaptive Cruise Control

24.1 Optimal Speed

The Krauss car-following model is based on the Gipps model where the follow speed depends on the speed of the leading vehicle (v_l) and space-gap (g). In the optimal speed model, the follow speed is a function of g and is calculated as following:

$$v_{follow} = \max[0, \min\left[\frac{g - G_{min}}{T_g}, v_{max}\right]]$$

Where,

g: Space-gap between following and leading vehicles
G_{min}: Minimum space-gap between following and leading vehicles
T_g: Desired time-gap
v_{max}: Maximum speed of the vehicle

When the vehicle is traveling with no leading vehicle ($g \to \infty$), $v_{following}$ is bounded and cannot exceed v_{max}. When the vehicle is very close to the leading vehicle ($g \to G_{min}$), $v_{following}$ approaches to zero. Imagine $G_{min} = 2m$, $T_g = 1.64s$ and $v_{max} = 30m/s$. Then we have

$$v_{follow} = \max[0, \min\left[\frac{g - 2}{1.64}, 30\right]]$$

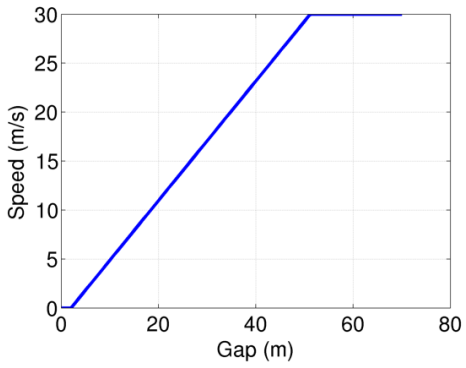


Figure 82: **Speed-Gap diagram**

As you can see, the follow speed $v_{following}$ is a function of g and is plotted in Figure 82. If g is less than or equal to G_{min}, then the follow speed is zero. If g is greater than G_{min}, then by increasing g, the follow speed increases with slope 1/1.64 until it reaches 30 m/s.

We can introduce sensitivity parameter τ that determines the response time of the vehicle. If the current speed of the vehicle is v and the next speed is v_{follow}, then the acceleration $\ddot{x}$ with sensitivity parameter τ is equal to

$$\ddot{x} = \frac{2}{\tau}(v_{follow} - v)$$

And the new follow speed would be:

$$v_{follow} \leftarrow v_{follow} + \ddot{x} \times \Delta t$$

Thus, v_{next} in Optimal speed car-following model after substitution into (3) in Section 4 would be:

$$v_{next} \leftarrow \min\left[\min[min[v + a_{max} \times \Delta t, v_{max}], v_{max}^{A}], v_{follow}\right]$$

Optimal speed is a simple car-following model that is easy to implement, but suffers from unrealistic large acceleration and deceleration. Moreover, rear-end collision might occur with bounded acceleration and deceleration. The following lines define a vehicle type 'Type1' with optimal speed car-following model:

```
<vType id="Type1" length="5.0" minGap="2.0" maxSpeed="28.0" vClass="passenger">
    <carFollowing-OptimalSpeed accel="100" decel="100" sigma="0" tau="1.64"
sensitivity="1.21" />
</vType>
```

Attribute	Unit	Description
accel	m/s^2	Maximum acceleration
decel	m/s^2	Maximum deceleration (absolute value)
sigma	-	The driver imperfection [0,1]
tau	s	Desired time-gap
sensitivity	s	Vehicle's response time

24.2 ACC

The ACC car-following model has multiple operation modes, and the system transits between these modes in order to generate the desired acceleration. You can refer to this paper for more information. You can watch this video to understand what ACC is.

The following lines define a vehicle type 'TypeACC' with ACC car-following model:

```
<vType id="TypeACC" length="5.0" minGap="2.0" maxSpeed="30.0" vClass="passenger">
    <carFollowing-ACC accel="3.0" decel="5.0" tau="1.54" controllerDelay="0.5"
ComfAccel="2.0" ComfDecel="3.0" K_sc="0.4" K_v="1.0" K_g="5.0" V_int="30.0"/>
</vType>
```

Attribute	Unit	Description
accel	m/s^2	Maximum acceleration
decel	m/s^2	Maximum deceleration (absolute value)
tau	s	Time-gap
controllerDelay	s	Lower controller delay
ComfAccel	m/s^2	Comfortable acceleration
ComfDecel	m/s^2	Comfortable deceleration
K_sc	1/s	Speed control gain
K_v	1/s	Speed gain
K_g	$1/s^2$	Gap gain
V_int	m/s	Intended speed

At each time step, the ACC car-following model running on a vehicle needs to know 1) space-gap to the front vehicle and 2) speed of the front vehicle in order to calculate the follow speed. We assume that a 'radar' is installed on the front bumper of the ACC vehicle that can measure 1) and 2). On the other hand, ACC car-following model needs the maximum deceleration of the front vehicle to calculate the safe gap that can be obtained using different approach.

ACC Vehicle Platooning Example 1

You can find an example of ACC vehicle platooning in the 'examples/carFollowing/ACC' folder. Open the omnetpp.ini file and find the ACC configuration.

The TraCI module is active and 'SUMOconfig' is pointing to the SUMO configuration file located in the 'ACC' folder. The 'addNode' module is active and points to the 'add_00' configuration that inserts a 10-vehicle platoon of type 'TypeACC'. Note that the 'platoonMaxSpeed' attribute is zero that causes the platoon to stop moving after insertion. The vehicle type 'TypeACC' is defined in the traffic demand file, located in 'ACC/4.hello.rou.xml'. You can find more detailed information on how to insert vehicle platoons in Section 3.7.

```
<vehicle_platoon id="veh" type="TypeACC" size="10" route="route1"
departPos="100" platoonMaxSpeed="0" />
```

The 'trafficControl' module is also active and points to the following 'control_0' configuration. Beginning from 60s the speed of veh (platoon leader) changes following a square wave where the speed of veh transitions between 0 m/s and 28 m/s. Note that 28 m/s is lower than the maximum vehicle speed so that followers be able to catch-up. You can find detailed information on how the following speed change works in Section 4.1.

```
<speed id="veh" begin="60" end="1000" value="28*(-1)^floor(2(t-60)/200)"/>
```

The 'record_stat' is active on all vehicles and four data (vehId, lanePos, speed and accel) is collected from each vehicle. The output is written to the 'results/000_vehicleData.txt' file. We then use the Matlab script 'SpeedProfile.m' located in the scripts folder to read the output file and plot different figures as shown in Figure 83. Make sure the 'path' variable in the 'SpeedProfile.m' script points to the output file path.

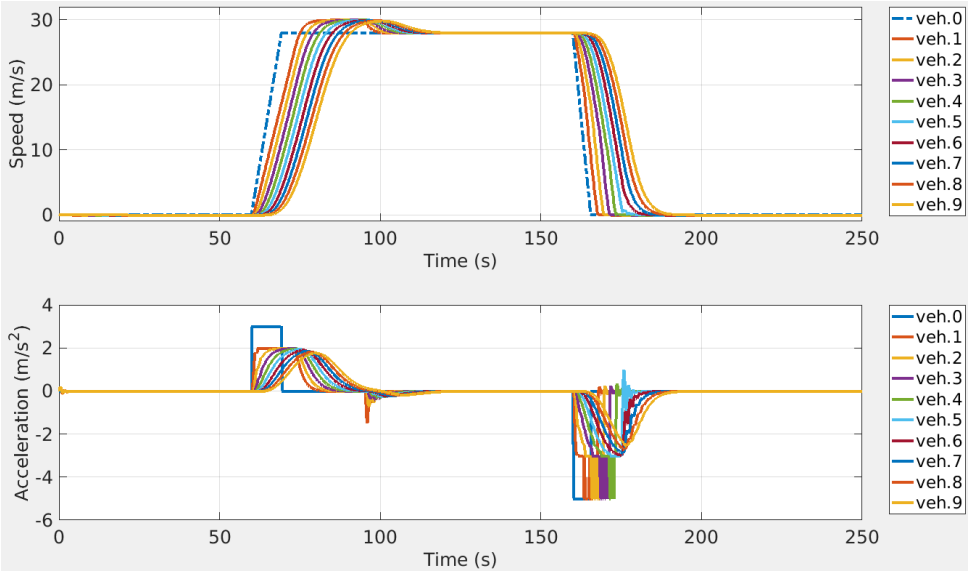


Figure 83: **Speed and acceleration of 10-vehicle ACC platoon over time**

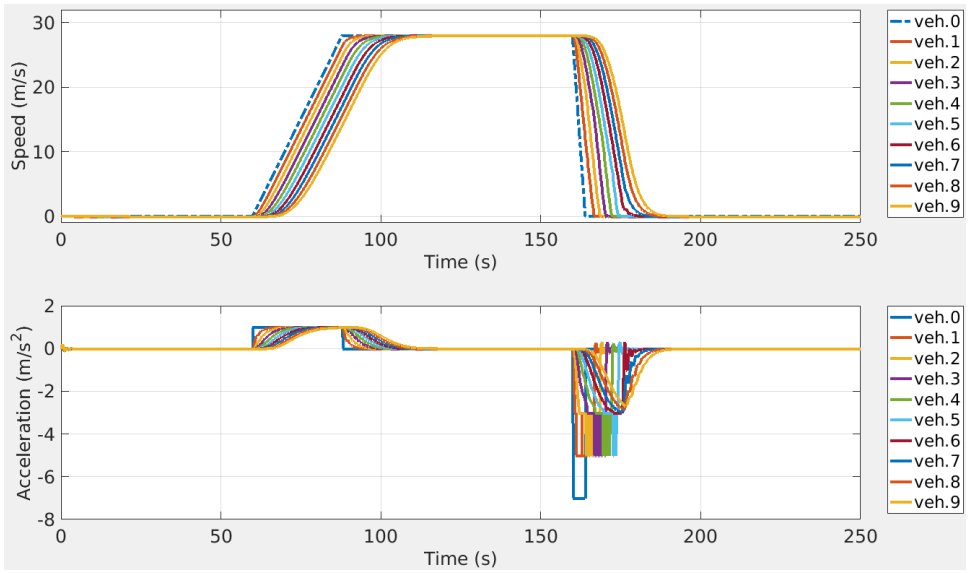


Figure 84: **Speed and acceleration of 10-vehicle ACC platoon over time**

ACC Vehicle Platooning Example 2

In the ACC configuration, change the trafficControl id to “control_1”. Now we have the following speed change for veh (platoon leader) that is identical to the last example, but we override the maximum acceleration and deceleration. The maximum acceleration is now lower than the default (3.0) and the maximum deceleration is now higher than the default (5).

```
<speed id="veh" begin="60" end="1000" value="28*(-1)^floor(2(t-60)/200)"
maxAccel="1" maxDecel="7" />
```

Run the simulation again and plot the new speed/acceleration profile as shown in Figure 84. As you can see, veh accelerates slower and breaks harder. Hopefully, following vehicles are able to react fast-enough to prevent rear-end collision.

ACC Vehicle Platooning Example 3

For demonstration purpose, we change the speed of veh (platoon leader) following a sinusoid function. In the ACC configuration, change the trafficControl id to "control_2" where we have:

```
<speed id="veh" begin="60" end="1000" value="18+10*sin(0.1*(t-60))" />
```

The speed and acceleration profile of the vehicles is shown in Figure 85. As you can see the veh's speed is oscillating between 28 m/s and 8 m/s with period of $\frac{2\times\pi}{0.1} \cong 62.8s$. After a warm-up time, every follower in the platoon perfectly reproduces what the leader is doing.

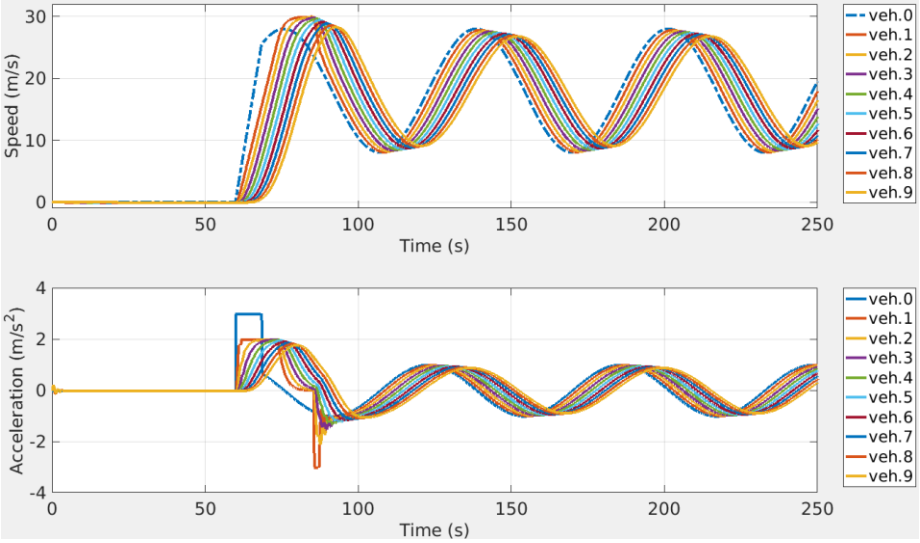


Figure 85: **Speed and acceleration of 10-vehicle ACC platoon over time**

ACC Vehicle Platooning Example 4

In previous examples, we assumed that space-gap and speed measurement are both accurate. You can add inaccuracy into space-gap and speed measurement using 'errorGap' and 'errorRelSpeed' parameters respectively. Both of these parameters can take double values in the [0,1] interval. Note that if the error value is high, then rear-collision might happen. The 'ACC_MeasurementError' in the same omnetpp.ini file shows you how to get started.

24.3 CACC

The CACC car-following model is based on the ACC model and allows vehicles to drive with even closer space gap. This is achievable by parameter sharing using the V2V wireless communication. You can also check this video for more information. Each CACC vehicle broadcasts beacons periodically (every 0.1s by default). A beacon message contains valuable data such as vehicle id, current speed, current acceleration,

maximum deceleration capability, current position, platoon id, platoon depth, etc. We are using a custom message format for beacons, although Basic Safety Message (BSM) is the most commonly used format.

CACC vehicles in a platoon listen to all broadcasted beacons and build a 'view' from the vehicles in the platoon. The 'view' structure looks like the following. Each vehicle updates its view from the platoon when a beacon is received. These collected data are then used by the CACC car-following model in order to calculate an accurate and safe follow speed.

```
typedef struct platoonConfig
{
    double timestamp;
    std::string vehId;
    double speed;
    double accel;
    double maxDecel;
} platoonConfig_t;

typedef struct platoonView
{
    std::string platoonId;
    int platoonSize;
    int platoonDepth;
    std::map<int /*depth*/, platoonConfig_t> platoonConfiguration;
} platoonView_t;
```

The following lines define a vehicle type 'TypeCACC1' with CACC car-following model:

```
  <vType id="TypeCACC1" length="5.0" minGap="2.0" maxSpeed="30.0" vClass="passenger">
      <carFollowing-CACC strategy="1" accel="3.0" decel="5.0" tau="0.71"
controllerDelay="0.5" ComfAccel="2.0" ComfDecel="3.0" K_sc="0.4" K_v="0.99" K_g="4.08"
K_a="0.66" V_int="30.0" degradeToACC="0" invalidTimer="0.1" />
  </vType>
```

In addition to the ACC attributes, the CACC controller supports these attributes:

Attribute	Unit	Description
degradeToACC	-	Should the CACC controller downgrade to ACC
invalidTimer	s	When should a 'platoon view' considered stall
strategy	-	Platooning strategy
K_a	1/s	Acceleration gain

The 'degradeToACC' attribute determines if the CACC vehicle should downgrade itself to ACC in case of packet loss. One or more packet losses cause the platoonConfiguration data to go out of date. In this case, the vehicle will not have an accurate view of the platoon and the follow speed calculation might be unsafe. In this case, the vehicle degrades to ACC if 'degradeToACC' is on. Otherwise, the vehicle keeps using the stall information. The 'invalidTimer' attribute shows how many seconds, since seeing the last data, to consider the platoonConfiguration invalid. If you set 'invalidTimer' to be equal to the beaconing rate, then a single beacon loss will invalidate the platoonConfiguration.

The 'strategy' attribute determines the platooning strategy in CACC vehicles that is used for parameter sharing. Many different platooning strategies can be used in a CACC vehicle platoon. The traditional strategy is the one-vehicle look-ahead communication where a following CACC vehicle is only influenced

by its front CACC vehicle as shown in Figure 86a. Hence a complete view of the platoon is not needed. More complicated platooning strategies take the data from more than one vehicle in the platoon into account. In this case, the following vehicles can also influence the lead vehicle that is shown to produce more stable platoons. Figure 86b-d show some examples.

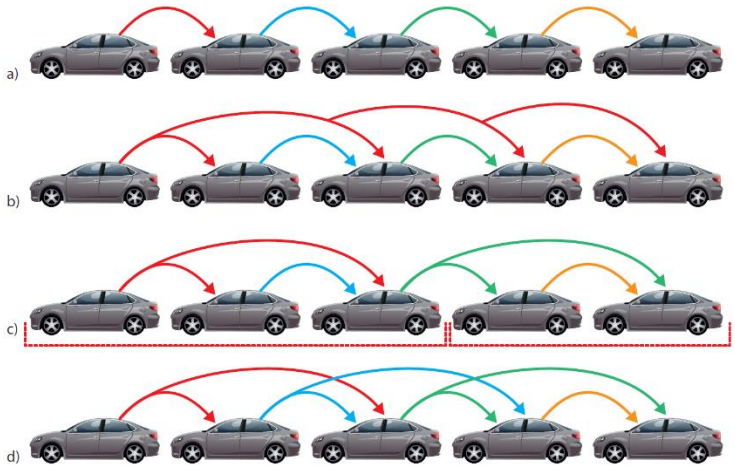


Figure 86: **Platooning strategies in CACC vehicles** [Page 54 in ref]

VENTOS has implemented the following platooning strategies for CACC vehicles. You can potentially have a CACC platoon that vehicles do not have an identical 'strategy' value, but make sure that you know what you are doing!

- strategy = 1: one-vehicle look-ahead communication (Figure 86a). Acceleration and maximum deceleration of the front vehicle is extracted from the beacon message. Speed of the front vehicle and the space-gap to the front vehicle is still obtained from the radar (similar to ACC).
- strategy = 2: one-vehicle look-ahead communication plus acceleration from the platoon leader (Figure 86b) - Under development

CACC Vehicle Platooning Example 1

You can find an example of CACC vehicle platooning in the 'examples/carFollowing/CACC' folder. Open the omnetpp.ini file and find the CACC configuration. The CACC configuration is very similar to the ACC configuration in Section 24.2. All platoon members are of type 'TypeCACC1' and the platooning strategy is one-vehicle look-ahead communication. Note that the platoon leader always falls back to ACC system.

```
<vehicle_platoon id="veh" type="TypeCACC1" size="10" route="route1"
departPos="100" platoonMaxSpeed="0" />
```

The CACC vehicles send beacon messages periodically every 0.1s. The beacon's message format is defined in 'src/msg/BeaconVehicle.msg' file. You can add your own fields into the beacon format, but be cautious when removing the existing fields because some examples rely on them. You can find the beaconing code in 'src/nodes/vehicle/01.Beacon.cc' file.

Setting the 'SUMOvehicleDebug' parameter to 'true' displays the detailed parameter exchange in the SUMO log window. This makes the simulation slow, but is very helpful in debugging your code. Similar to

the ACC example, the collected data from all vehicles are stored in the 'results/000_vehicleData.txt' file. You can use the 'SpeedProfile.m' Matlab script located in the scripts folder to read the output file and plot different figures.

CACC Vehicle Platooning Example 2

Beaconing is an integral part in CACC vehicle platooning. Packet loss can affect the performance of a CACC platoon and causes platoon instability or even rear-end collision. Scenario 'CACC_PacketLoss' will give you an idea of how packet loss affects CACC platooning. The following two parameters control packet drops:

```
Network.V[*].appl. dropStartTime = 50s
Network.V[*].appl.plr = 60
```

The 'dropStartTime' specifies when the packet drop starts in a vehicle and 'plr' denotes packet loss ratio and is a real number in the [0,100] range. In the above example, all vehicles start dropping packets from 50s with probability of 60 percent. Packet drop is simulated in the application layer of each vehicle and follows a uniform random distribution. After 50s, SUMO starts showing the following warnings:

Warning: SimTime=56.70: vehicle 'veh.2' has outdated data from the front vehicle 'veh.1' (invalidTimer=0.100000)
The last beacon from the front vehicle was received at time '56.462500'

Warning: vehicle 'veh.2' continue using the CACC controller (Down-grade to ACC is off)

You can increase the 'invalidTimer' value in order to decrease the sensitivity to packet loss. You can also turn the 'degradeToACC' attribute. Open the 'examples/CACC/4.hello.rou.xml' file and look for vehicle type 'TypeCACC1'. You can find the 'invalidTimer' and 'degradeToACC' attributes there.

Last but not least, since speed of the front vehicle and the space-gap to the front vehicle is still obtained from the radar (similar to ACC), you can add inaccuracy into them using 'errorGap' and 'errorRelSpeed' parameters as shown in ACC example 4 before.

25 Vehicle Platooning with Platoon Management Protocol

Vehicle platooning is a technique where highway traffic is organized into groups of close-following vehicles called **platoon** or **convoy**. Platooning enables vehicles to drive closer (maintain a smaller headway) than normal vehicles with the same speed. Vehicle platooning improves traffic throughput, homogeneity and safety and at the same time reduces fuel consumption and emissions. Vehicle platooning requires cooperation between vehicles with the help of a **platoon management protocol**. This allows platoons to perform different maneuvers such as merge, split and leave.

In Section 3.7.3 we showed you how to add a vehicle platoon with an active platoon management protocol running on all platoon members using the 'addNode' module. In Section 4.3, we then showed you how to use the 'trafficControl' module to perform different maneuvers (merge, split, leave) on a platoon. In this section we are going to give a platooning example that helps you to get started. You can find this example in 'examples/platoon_management' folder.

Click Run in the menu bar and select ‘Run Configurations…’. Choose 'OMNET++ Simulation' and click on 'New launch configuration' button at the top left. Give this configuration a name like myPlatooning. In ‘Executable’, choose opp_run and set ‘Working dir’ to /VENTOS/examples/platoon_management. In 'Ini file(s)' choose omnetpp.ini. From 'Config name' choose ‘Platooning’ configuration from the drop down list. Leave the rest of the options to default. Click Apply and then click Run. OMNET++ Qtenv graphical runtime environment and SUMO-GUI window appear. Click on the express run button on the toolbar to start the simulation.

A 6-vehicle platoon is inserted into the middle lane of a 3-lane highway as shown in Figure 87. Since the platoon management protocol is active, the platoon leader and its followers are marked with red and blue color respectively. As we go farther from the platoon leader, the blue color fades to show the platoon depth. Also note that the lane changing in all platoon members are inactive that prohibits automatic lane change. You can however issue a leader/follower leave that forces a lane change in that vehicle. The platoon performs difference maneuvers along its way to the destination.

Figure 87: **Platoon of 6 vehicles with active platoon management protocol**

The platoon configuration at the time of insertion is shown in the following table. The dash ‘-’ means that the vehicle does not have that information. As you can see, platoon members only know their platoon id and depth and they are not aware of platoon size, optimal size and maximum size.

vehId	pltId	pltDepth	pltSize	pltOptSize	pltMaxSize
veh	veh	0	6	4	10
veh.1	veh	1	-	-	-
veh.2	veh	2	-	-	-
veh.3	veh	3	-	-	-
veh.4	veh	4	-	-	-
veh.5	veh	5	-	-	-

The ‘record_platoon_stat’ parameter enables you to collect very useful data from platoons. You can set it to ‘true’ in all vehicles as shown below. Two files are generated at the end of a simulation that shows all platooning messages exchanged between vehicles, platoon configuration at each time step and duration of each platooning maneuver. You can find more information in Section 29.11.

```
Network.V[*].appl.record_platoon_stat = true
```

Example Explained

The platoon is inserted using the following line:

```
<vehicle_platoon id="veh" type="TypeACC" size="6" route="route1"
departPos="100" departLane="1" platoonMaxSpeed="0" pltMgmtProt="true"
optSize="4" maxSize="10" />
```

The platoon name is 'veh' and it is a homogeneous platoon; meaning that all platoon members are of the same type 'TypeACC'[1]. Platoon size is 6 and it is inserted on lane 1 at depart position 100m. The platoon leader has depth of 0, first followers has depth of 1 and so forth. Platoon members are named as **veh**, **veh.0**, **veh.1**, **veh.2**, **veh.3**, **veh.4** and **veh.5**. Note that the platoon name is the vehicle id of the platoon leader. The maximum platoon speed is 0 m/s meaning that it will not move. The platoon management protocol is active and optimal and maximum size are 4 and 6 respectively. You can find more information about the attributes of 'vehicle_platoon' in Section 3.7.

The platoon size is 6, but the optimal platoon size is 4. Thus, the platoon leader tries to perform a platoon split maneuver to reduce the platoon size from 6 to 4 as shown in Figure 88. After split completion, we have two separate platoons; platoon 'veh' of size 4 and platoon 'veh.4' of size 2.

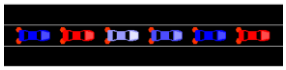


Figure 88: **Platoon split**

The platoon configuration at this time is shown in the following table. Note that the rear platoon inherits 'optimal platoon size' and 'maximum platoon size' from the original platoon.

vehId	pltId	pltDepth	pltSize	pltOptSize	pltMaxSize
veh	veh	0	4	4	10
veh.1	veh	1	-	-	-
veh.2	veh	2	-	-	-
veh.3	veh	3	-	-	-
veh.4	veh.4	0	2	4	10
veh.5	veh.4	1	-	-	-

The trafficControl module controls the behavior of the platoon. The following line changes the speed of vehicle 'veh' (platoon leader in the front platoon) to 20 m/s starting from 10s as shown in Figure 89. You can find more information about speed change in Section 4.1.

```
<speed id="veh" begin="10" value="20" />
```

Figure 89: **Two vehicle platoons traveling at speed of 20 m/s**

This above picture shows the **inter-platoon gap** (the gap between two platoons) and **intra-platoon gap** (the gap between vehicles in a platoon). The platoon is defined without any 'interGap' attribute, thus the default inter-platoon gap of 3.5s is used. On the other hand, the 'tau' attribute that is defined in the car-following model determines the intra-platoon gap. In this example, 'tau' is equal to 1.2s that is higher than the 'tau' in CACC platooning.

[1] CACC is more suited for vehicle platooning, but ACC is used here for simplicity.

The following line increases the optimal platoon size in both platoons to 5 starting from 40s. This does not have any effects on the platoons. The rear platoon cannot merge into the front platoon, because the new platoon size will exceed 5. You can find more information about change the optSize in Section 4.3.1.

```
<optSize begin="40" value="5" />
```

The following line decreases the optimal platoon size in both platoons to 3 starting from 60s:

```
<optSize begin="60" value="3" />
```

The front platoon first performs a split maneuver to reduce its platoon size from 4 to 3. Now we have 3 platoons: veh (front), veh.3 (middle) and veh.4 (rear). The rear platoon starts a merge maneuver and changes it platoon size from 2 to 3. Now we have 2 platoons: veh (front), veh.3 (rear). Both platooning maneuvers are shown in Figure 90.

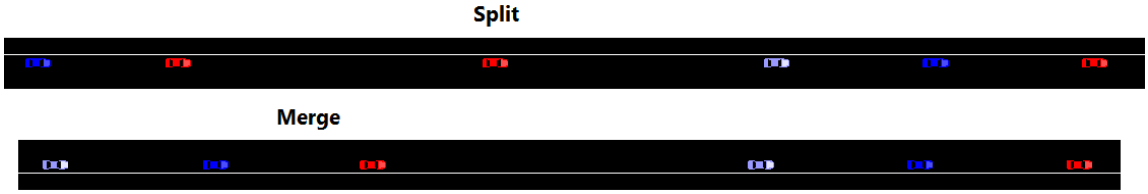


Figure 90: **Changing the optimal platoon size to 3 causes a split followed by a merge**

The platoon configuration after the split is shown in the following table:

vehId	pltId	pltDepth	pltSize	pltOptSize	pltMaxSize
veh	veh	0	3	3	10
veh.1	veh	1	-	-	-
veh.2	veh	2	-	-	-
veh.3	veh.3	0	1	3	10
veh.4	veh.4	0	2	3	10
veh.5	veh.4	1	-	-	-

The platoon configuration after the merge is shown in the following table:

vehId	pltId	pltDepth	pltSize	pltOptSize	pltMaxSize
veh	veh	0	3	3	10
veh.1	veh	1	-	-	-
veh.2	veh	2	-	-	-
veh.3	veh.3	0	3	3	10
veh.4	veh.3	1	-	-	-
veh.5	veh.3	2	-	-	-

The following line asks platoon ‘veh.3’ to merge into the front platoon starting from 80s. The final platoon is shown in Figure 91. You can find more information about merge in Section 4.3.2.

```
<pltMerge pltId="veh.3" begin="80" />
```

Figure 91: **Merging the rear platoon into the front platoon**

The platoon configuration at this time is shown in the following table. Note that the optimal platoon size is now equal to the platoon size.

vehId	pltId	pltDepth	pltSize	pltOptSize	pltMaxSize
veh	veh	0	6	6	10
veh.1	veh	1	-	-	-
veh.2	veh	2	-	-	-
veh.3	veh	3	-	-	-
veh.4	veh	4	-	-	-
veh.5	veh	5	-	-	-

Following to the merge maneuver, we have the following two split maneuvers:

```
<pltSplit pltId="veh" splitIndex="2" begin="100" />
<pltSplit splitVehId="veh.5" begin="120" />
```

The first line splits the platoon ‘veh’ from depth 2 beginning at 100s. After the split, the front platoon size is 2 and the rear platoon size is 4. The optimal platoon size in each platoon is equal to the platoon size. The second line performs a split maneuver from vehicle ‘veh.5’. The final platoons are shown in Figure 92. You can find more information about split in Section 4.3.3.

Figure 92: **Resulting platoons after two split maneuvers**

Then we form a single platoon of 6 vehicles. We can either perform two merge maneuvers or change the optimal platoon size to 6 in all three platoons as following. The platoon management protocols running in each platoon will coordinate to form a single 6-vehicle platoon.

```
<optSize begin="140" value="6" />
```

Finally, we perform the following three leave maneuvers. Starting from 180s, the last platoon follower, ‘veh.5’, performs a platoon leave maneuver and changes lane to the left. Starting from 200s, vehicle ‘veh.3’ that is the middle follower performs a leave maneuver and changes lane to the right. Starting from 240s, the platoon leader performs a leave maneuver and changes lane to the right. The first follower takes over the platoon leader role. You can find more information about platoon leave in Section 4.3.4.

```
<pltLeave pltId="veh" leaveIndex="5" begin="180" leaveDirection="left"/>
<pltLeave pltId="veh" leaveIndex="3" begin="200"/>
<pltLeave pltId="veh" leaveIndex="0" begin="240"/>
```

26 Simulating a Blocked Street

In order to study traffic congestion, you might need to add a stopped or crashed car on the road. You can use the "obstacle" element as explained in Section 3.3 to stop a vehicle at a predefined point in time. You can refer to 'examples/vehCrash' to get an idea. This example shows a two-lane highway where a stopped vehicle is inserted into lane 0 and leaving the other lane free. You can find the following scenarios:

StoppedVehicle: A red vehicle is stopped on lane 0 of edge 1to2 at position 200. Approaching vehicles first decelerate and then change lane to avoid collision.

StoppedVehicleWithResume: A red vehicle is stopped on lane 0 of edge 1to2 at position 200 starting from 10s. Approaching vehicles first decelerate and then change lane to avoid collision. The stopped vehicle disappears after 20s.

StoppedVehicleWithHeavyTraffic: This scenario is similar to the first scenario, but vehicles are inserted in shorter intervals to increase the traffic flow. Some vehicles start accumulating behind the stopped vehicle, forming a queue. These vehicles want to overtake (left-turn signal is on), but the continuously approaching vehicles on the neighbor lane (lane 1) do not leave enough room to safely merge into. Once no more vehicle is inserted, the queued vehicles get a chance to change lane.

To prevent vehicles from lining up behind the stopped vehicle,

- You can decrease the allowed speed on the neighbor lane after the accident to allow a safe gap for lane change. This is doable using variable speed signs or trafficControl module (Section 4.1).
- Another solution is to decrease the traffic flow intensity.

27 Forcing a Rear-end Car Accident

Sometimes the accident itself is required instead of the effects of an accident. SUMO tracks gaps between vehicles that are on the same edge either fully or partially. By default, whenever these gaps are reduced to below a vehicles minGap a collision is registered. If you want to make sure that vehicles intersect closely you should set the minGap attribute to zero. Note that SUMO cannot (and is not designed to) simulate the crash physics. Flanking collisions are more difficult to create and head-on collisions are currently not possible. Here, we present three approaches to force a rear-end car accident.

27.1 Designing a new car-following model

A car-following model, as discussed in Part 2, Section 4.1, determines the follow speed in relation to the vehicle ahead. The default car-following model in SUMO, Krauss, aims to be accident free and it is supposed to never reduce the space gap below the minGap. You can create a new car-following model by

modifying the Krauss model to return a larger value for 'follow speed'. Vehicles drive in an unsafe manner and rear-end accident might happen if the leading vehicle brakes hard.

On the other hand, the Krauss_Orig1 car-following model allows rear-end accident to happen because the following vehicle thinks that the leading vehicle has the same maximum deceleration as we saw in Part 2, Section 4.1.

27.2 Disabling safety check in Krauss Car-following Model

Although, Krauss car-following model is designed to be accident free, you can still force an accident by disabling the safety check using TraCI. Let's see how it works. In order to make a vehicle stop, you can use the following TraCI command. The vehicle starts decelerating with maximum deceleration 'decel' that you have specified in the vehicle type until its speed reaches zero.

```
TraCI->vehicleSetSpeed(std::string vehID, double speed)
```

In order to force an accident, the vehicle should stop with much higher deceleration. You can use the vehicleSetSpeedMode command to set how the values set by vehicleSetSpeed above shall be treated.

```
TraCI->vehicleSetSpeedMode(std::string vehID, uint32_t bitset)
```

To disregard the maximum deceleration, we need to unset bit2; thus bitset would be 0b11011 = 27. This means that when you set the vehicle speed to zero, the vehicle stops immediately in the next time step. In a multi-lane street, the following vehicle changes lane to avoid the rear-end collision. Thus, you need to turn off lane changing using the vehicleSetLaneChaneMode command and set bitset to 0.

```
TraCI->vehicleSetLaneChangeMode(std::string vehID, uint32_t bitset)
```

An Example

Let's write an example that shows how a rear-end car accident can happen. We are going to use the SUMO files located in 'examples/vehCrash/sumocfg' that inserts two vehicles: veh0 and veh1 into a two-lane street where veh1 is following veh0.

Step 1: Go to src/gettingStarted folder and open tutorial.cc source file. Add the following code into the tutorial::initialize_withTraCI method in order to disable lane changing in veh0 and veh1.

```
TraCI->vehicleSetLaneChangeMode("veh0", 0);
TraCI->vehicleSetLaneChangeMode("veh1", 0);
```

Step 2: Add the following code into the tutorial::executeEachTimeStep method. At simulation time 20, the speed mode of veh0 will be changed to 27 that disregards maximum deceleration. Then we set the veh0's speed to zero to make it stop immediately.

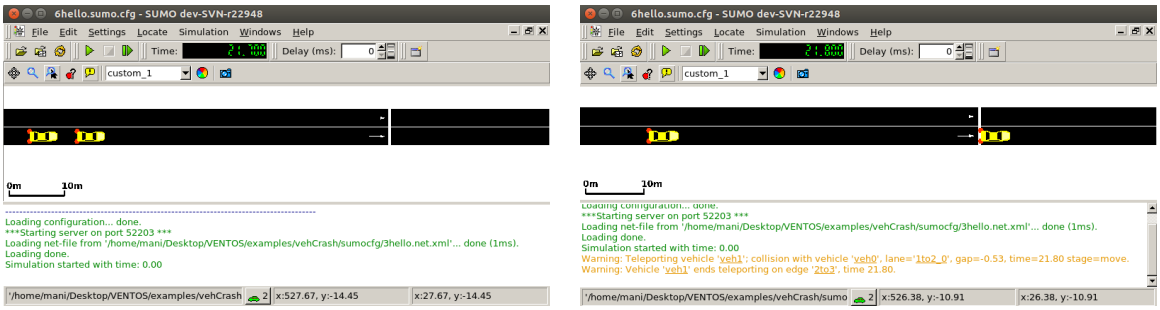
```
if(omnetpp::simTime().dbl() == 20)
{
    TraCI->vehicleSetSpeedMode("veh0", 27);
    TraCI->vehicleSetSpeed("veh0", 0);
}
```

Step 3: Run the simulation as before, pointing the run configuration to ‘examples/vehCrash’. Then choose the ‘ForceCrash’ scenario from the drop down menu.

Step 4: Both vehicles, veh0 and veh1, are inserted into lane 0 where veh1 is following veh0. At time 20.0 the front vehicle (veh0) stops immediately. Figure 93a shows the moment right before the accident. At time 21.8, veh1 crashes into veh0 and SUMO shows a warning message as shown in Figure 93b:

Warning: Teleporting vehicle 'veh1'; collision with vehicle 'veh0', lane='1to2_0', gap=-0.53, time=21.80 stage=move.
Warning: Vehicle 'veh1' ends teleporting on edge '2to3', time 21.80.

When an accident happens, SUMO **teleports** the crashed vehicle to the next edge on its route. You can change the default behavior using --collison.action option as discussed in Section 27.4.



a) b)

Figure 93: **Rear-end car collision in SUMO**

27.3 Modifying the ‘tau’ Value

Parameter ‘tau’ is intended to model a driver’s reaction time (in seconds) and it is used by all car-following models. Driver’s attempt to maintain a minimum time gap of ‘tau’ between the rear bumper of their leader and their own front bumper + minGap. The reaction time is fundamentally limited by the simulation step size and a value of ‘tau’ below step-size may lead to collisions if the leader suddenly starts braking hard with maximum deceleration. You can try this in the last example:

Step 1: Go to the tutorial::executeEachTimeStep method and comment the following line:

```
TraCI->vehicleSetSpeedMode("veh0", 27);
```

Step 2: Go to ‘examples/vehCrash’ folder and open ‘4hello.rou.xml’ file. Change the ‘tau’ parameter from 1.64 to 0.01.

Step 3: Run the simulation as before. You will notice that SUMO shows you a warning message at the start of the simulation.

Warning: Value of tau=0.01 in car following model 'carFollowing-Krauss' lower than simulation step size may cause collisions

At time 20.0 the front vehicle (veh0) starts decelerating and veh1 eventually crashes into veh0. Note that this time the teleporting message is slightly different:

Warning: Teleporting vehicle 'veh1'; collision with vehicle 'veh0', lane='2to3_0', gap=-0.05, time=24.50 stage=move.
Warning: Vehicle 'veh1' teleports beyond arrival edge '2to3', time 24.50.

The collision happens on lane ‘2to3_0’ and there is no next edge for veh1 to be teleported into. That’s why SUMO teleported veh1 beyond edge ‘2to3’ and removes it from the simulation.

27.4 Collision Action

You can configure the consequence of a collision using the option --collision.action.

- **teleport:** By default, the crashed vehicle is teleported meaning that it is moved (teleported) to the next edge on its route. In other words, a vehicle teleports by going “under” the road, and popping back up at the first chance it gets when the road ahead is clear
- **warn:** a warning is issued
- **none:** no action is take
- **remove:** both vehicles are removed from the simulation

28 Measuring Real-time Intersection Parameters

29 Simulation Output

VENTOS can collect different aspects of the simulation and save them into text files at the end of the simulation[1]. The simulation output files are stored in the 'results' folder for each individual examples and have the following name format, where xxx shows the simulation run number.

```
xxx_name.txt
```

All output files have a common header that shows the simulation configuration like the following:

```
configName      tutorial1
iniFile         omnetpp.ini
processID       4777
runID           tutorial1-0-20170301-09:27:48-4777
totalRun        1
currentRun      0
currentConfig   $repetition=0
sim timeStep    100 ms
startDateTime   20170301-09:27:48
endDateTime     20170301-09:27:54
duration        0 hour, 0 minute, 6 second
```

Once the simulation output files are generated, you can use Matlab, R, Python, etc. to post-process the results and draw figures. You can find sample scripts (mostly in Matlab) in the 'scripts' folder. Note that VENTOS collects simulation information per node per simulation time-step. In order to get an aggregated output (for example average speed in all vehicles), you need to write a script to process the output and calculate the aggregated value. Collecting simulation output is disabled by default, and you need to enable it individually from the configuration file as we will show in the following sections.

29.1 Collect Simulation information

You can collect the following data from the simulation in 'results/xxx_simData.txt' file.

Name	Type	Description
loaded	int	Number of loaded vehicles in this time step
departed	int	Number of departed vehicles so far
arrived	int	Number of arrived vehicles so far
running	int	Number of running vehicles in this time step
waiting	int	Number of vehicles that are waiting for insertion in this time step

All you need to do is to set the 'record_sim_stat' parameter to 'true' and list all the data that you want to collect in the 'record_sim_list' parameter separated with '|' character. For instance, the following configuration collects the accumulated number of departed vehicles, number of running vehicles and accumulated number of arrived vehicles in each time step. The output file will contain five columns: index, timeStep, departed, running and arrived with the same order specified in the 'record_sim_list' parameter. Note that the first row with timeStep = -1 denotes the moment right after TraCI establishment.

[1] You can also ask SUMO directly to collect different simulation data. You can find more information here.

```
Network.statistics.record_sim_stat = true
Network.statistics.record_sim_list = "departed | running | arrived"
```

29.2 Collect TraCI information

VENTOS relies heavily on the TraCI interface to connect to SUMO. You can monitor the exchange of TraCI commands between VENTOS and SUMO during the simulation. In the configuration file, you need to set the 'record_TraCI_activity' parameter to true as following:

```
Network.TraCI.record_TraCI_activity = true
```

The output is saved in 'results/xxx.TraCIActivity.txt' file and contains the following columns:

Name	Type	Description
sentAt	double (s)	Simulation time that the TraCI command is sent
duration	double (ms)	The time it takes for the command to run
cmdName	string	Name of the command
cmdGroupId	hex int	Group id of the command
cmdId	hex int	Id of the command in the corresponding group id

29.3 Collect Vehicle/Bike information

You can collect the following data from vehicles in 'results/xxx_vehicleData.txt' file.

Name	Type	Description
vehId	string	SUMO vehicle id
vehType	string	SUMO vehicle type
lane	string	The lane id that the vehicle is on
lanePos	double (m)	Position of the vehicle relative to the current lane
pos	double,double,double	(x,y,z) SUMO position of the vehicle
speed	double (m/s)	Vehicle speed
accel	double (m/s^2)	Vehicle acceleration (can be positive or negative)
departure	double (s)	The time that the vehicle is departed (inserted)
arrival	double (s)	The time that the vehicle is arrived to the destination
route	string	Current route of the vehicle (edge list)
routeDuration	double (s)	The time that the vehicle is on the network
drivingDistance	double (m)	The distance that the vehicle has driven so far
CFMode	string	Car-following mode of the vehicle
timeGapSetting	double (s)	Time-gap that the car-following model tries to maintain
timeGap	double (s)	Time-gap to the front vehicle
frontSpaceGap	double (m)	Space-gap to the front vehicle
rearSpaceGap	double (m)	Space-gap to the rear vehicle
nextTLId	string	Id of the next traffic light along the route
nextTLLinkState	char	Link state (green/yellow/red) of the next traffic light

Vehicle data collection can be activated/deactivated for each individual vehicle. For instance, the following configuration collects selected data from V[0] only where V denotes the OMNET++ module for vehicles. If you want to collect data from all vehicles then use V[*] instead.

```
Network.V[0].record_stat = true
Network.V[0].record_list = "vehId | lane | speed | accel | drivingDistance"
```

You can also customize data collection for each vehicle. For example the following configuration collects two different data from V[0] and V[1]. Note that the output text file contains nine columns (including index and timeStep). Also note that defining region of interest (ROI) as described in Section 15 will affect the simulation output data.

```
Network.V[0].record_stat = true
Network.V[0].record_list = "vehId | lane | speed | accel | drivingDistance"

Network.V[1].record_stat = true
Network.V[1].record_list = "vehId | vehType | lane | lanePos"
```

You can collect data from bicycles too by replacing V with B where B denotes the OMNET++ module for bicycles.

29.4 Collect Vehicle's Emission

You can collect the following pollutant/noise emission from vehicles in 'results/xxx_vehicleEmission.txt' file.

Name	Type
vehId	string
emissionClass	string
CO2	double (mg)
CO	double (mg)
HC	double (mg)
PMx	double (mg)
NOx	double (mg)
fuel	double (ml)
noise	double (dBA)

Vehicle emission data collection can be activated/deactivated for each individual vehicle. For instance, the following configuration collects selected data from V[0] only where V denotes the OMNET++ module for vehicles. If you want to collect data from all vehicles then use V[*] instead. You can collect data from bicycles too by replacing V with B where B denotes the OMNET++ module for bicycles. Also note that defining region of interest (ROI) as described in Section 15 will affect the simulation output data.

```
Network.V[0].record_emission = true
Network.V[0].emission_list = "vehId | emissionClass | CO2 | CO | HC | NOx"
```

When a vClass is specified for a vehicle type, this information is used by SUMO to assign a default emission class from the HBEFA version 3.1 model. Here are some examples.

Vehicle Class	Emission Class
passenger	HBEFA3/PC_G_EU4
bus	HBEFA3/Bus
truck	HBEFA3/HDV
bicycle	HBEFA3/zero

If you are not happy with the default emission class, then you can assign your own to a vehicle by using the vehicle type attribute "emissionClass". You can find all HBEFA3 emission classes here. SUMO supports other emission classes and you can find more information here. SUMO also provides small-sized tools and tests that help you develop and evaluate different emission models.

29.5 Collect loop detector information

First add loop detector through SUMO file
Then enable the parameter

29.6 Collect Vehicle Queue Size in an Intersection

29.7 Collect Vehicle Delay in an Intersection

29.8 Collect Traffic Light Timing Information

29.9 Collect DSRC MAC layer information

You can collect these statistics from the MAC layer of different nodes in 'results/xxx_MACdata.txt' file.

Name	Type	Description
NumDroppedFrames	int	Number of dropped frames due to full buffer
NumTooLittleTime	int	Number of too little time that occurs when remaining time in the current slot is smaller than the frame sending duration.
NumInternalContention	int	Number of contentions between EDCA queues. If more than one queues are ready to send, then the high priority queue is selected and lower priority queues get contention.
NumBackoff	int	Number of back offs due to channel being busy.
SlotsBackoff	int	Accumulated back off values
TotalBusyTime	double	The total time when channel was busy

In the configuration file, you need to set the 'record_stat' parameter in MAC layer of the node that you are interested to 'true' as following:

```
Network.V[0].nic.mac1609_4.record_stat = true
```

29.10 Collect DSRC PHY layer information

You can collect these statistics from the PHY layer of different nodes in 'results/xxx_PHYdata.txt' file.

Name	Type	Description
NumSentFrames	int	Number of total frames sent
NumReceivedFrames	int	Number of total frames (correctly) received
NumLostFrames_BitError	int	Number of total frames lost due to bit error
NumLostFrames_Collision	int	Number of total frames lost due to collision
NumLostFrames_TxRx	int	Number of total frames lost due to receive while sending

In the configuration file, you need to set the 'record_stat' parameter in PHY layer of the node that you are interested to 'true' as following:

```
Network.V[0].nic.phy80211p.record_stat = true
```

The 'record_frameTxRx' parameter enables you to track each frame from the sender to all receivers. The result is saved in the 'results/xxx_FrameTxRxdata.txt' file. The following statistics are collected:

Name	Type	Description
MsgId	int	The unique message id assigned to this frame by OMNET++
MsgName	string	Message name assigned by the application layer
SenderNode	string	Sender full name
RecieverNode	string	Receiver full name
ReceiverGateId	int	Receiver's radio gate id that received the frame
SendingStartAt	double (s)	The time that the sender starts transmitting the frame

FrameSize	int (byte)	Total frame size
ChannelNum	int	WAVE channel number that this frame is sent
TransmissionPower	double (mW)	Transmission power used for the WSM
TransmissionSpeed	double (Mbps)	The speed that the sender transmits the frame
TransmissionTime	double (ms)	The time it takes for the sender to transmit all frame's bits
DistanceToReceiver	double (m)	Distance to the receiver in direct path
PropagationDelay	double (ms)	The time it takes for a bit to get to the receiver in direct path
ReceptionEndAt	double (s)	The time that the received frame is sent up to MAC layer
FrameRxStatus	string	The condition of the received frame: HEALTHY/BITERROR/COLLISION/RECWHILESEND/DROPPED

As an example consider the network shown in Figure 94 where V[2] disseminates a frame using flooding at time 4.00011413 s. The direct distance between V[2] and all other vehicles is also shown in the picture. Note that the 'emulationActive' parameter in all vehicles is set to 'true'.

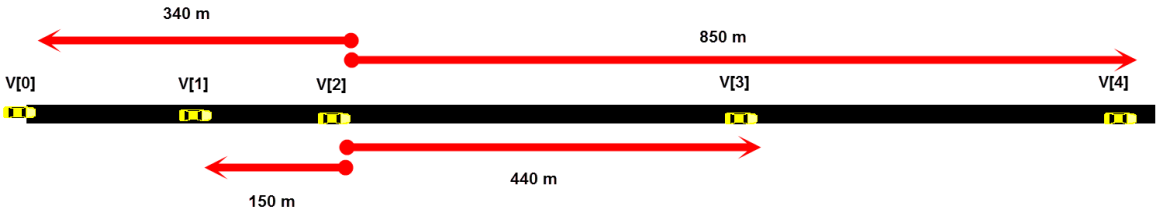


Figure 94: **A scenario of five vehicles where V[2] disseminates a frame using flooding.**

This communication is recorded in the 'FrameTxRxdata.txt' file and a minimum sample output is shown in the following table. V[1], V[0] and V[3] are in the radio range of V[2] and they all receive the frame. Note that V[4] is far away from V[2] and doesn't receive the frame[1]. V[1] is the closest vehicle to V[2] and receives the frame first, followed by V[0] and V[3]. V[3] receives the frame healthy and sends it up to the MAC layer. V[0] receives the frame with error and discards it. Finally, V[3] receives the frame while sending and is not decodable.

MsgId	Sender Node	Receiver Node	Sending StartAt	Propagation Delay	Reception EndAt	Frame RxStatus
60	V[2]	V[1]	4.00011413	0.0000005004670	4.00016263	RECWHILESEND
61	V[2]	V[0]	4.00011413	0.0000011343320	4.00016327	BITERROR
62	V[2]	V[3]	4.00011413	0.0000014676820	4.00016360	HEALTHY

29.11 Collect Vehicle Platooning Data

The 'record_platoon_stat' allows you to collect many useful data from vehicle platoons. You need to set the 'record_platoon_stat' parameter in all vehicles in the configuration file as following:

```
Network.V[*].appl.record_platoon_stat = true
```

[1] The default 'maximum interference distance' is 510 meters as calculated in Section 15.

The output is saved into the following text files. Note that file 2 is only generated if platoon management protocol is active in a platoon.

1) **xxx_plnConfig.txt:** Contains platoon configuration of each platoon in every time step such as platoon id, platoon depth, platoon optimal size, etc.

2) **xxx_plnData.txt**: Contains the data exchange between platoon members, vehicle state transition and start/end time of each platooning maneuvers.

The first file contains the following statistics:

Name	Type	Description
vehId	string	SUMO vehicle id
pltMode	int	Platooning mode id
pltId	string	Platoon id
pltDeph	int	Platoon depth
pltSize	int	Platoon size
pltOptSize	int	Platoon optimal size
pltMaxSize	int	Platoon maximum size

pltMode=1: no platooning
pltMode=2: platooning with no management protocol
pltMode=3: platooning with a management protocol

The second file contains the following statistics:

Name	Type	Description
vehId	string	Sender SUMO id
commandSent	string	The platooning command name
receiverId	string	Receiver SUMO id
senderPltId	string	Sender platoon id
receiverPltId	string	Receiver platoon id
fromState	string	Old vehicle's state
toState	string	New vehicle's state
maneuverStartEnd	string	Start/end point of a maneuver

Figure 95 shows a sample output from the 'xxx_plnData.txt' file. At time 0.21s, vehicle 'veh' goes from state 'state_waitForSplitReply' to 'state_makeItFreeAgent'. It then sends a CAHNGE_PL message to vehicle 'veh.4' in the same platoon and changes state to 'state_waitForAck'. At time 0.22, vehicle 'veh.4' receives the CHANGE_PL message and changes from 'state_waitForCHANGEPL' to 'state_sendingACK'. It then sends an ACK message to vehicle 'veh' and changes state to 'state_waitForSplitDone'.

The last column, maneuverStartEnd, shows the start and end time of a platooning maneuver. For example, at time 0.21s a split maneuver starts and then ends at 0.92s.

timeStep	vehId	fromState	toState	commandSent	receiverId	senderPltId	receiverPltId	maneuverStartEnd
0.10	veh	state_idle	state_platoonLeader	-	-	-	-	-
0.10	veh.1	state_idle	state_platoonFollower	-	-	-	-	-
0.10	veh.2	state_idle	state_platoonFollower	-	-	-	-	-
0.10	veh.3	state_idle	state_platoonFollower	-	-	-	-	-
0.10	veh.4	state_idle	state_platoonFollower	-	-	-	-	-
0.10	veh.5	state_idle	state_platoonFollower	-	-	-	-	-
0.10	veh	state_platoonLeader	state_sendSplitReq	-	-	-	-	-
0.10	veh	state_sendSplitReq	state_waitForSplitReply	SPLIT_REQ	veh.4	veh	veh	-
0.11	veh.4	state_platoonFollower	state_waitForCHANGEPL	SPLIT_ACCEPT	veh	veh	veh	-
0.21	veh	state_waitForSplitReply	state_makeItFreeAgent	-	-	-	-	Split_Start
0.21	veh	state_makeItFreeAgent	state_waitForAck	CHANGE_PL	veh.4	veh	veh	-
0.22	veh.4	state_waitForCHANGEPL	state_sendingACK	-	-	-	-	-
0.22	veh.4	state_sendingACK	state_waitForSplitDone	ACK	veh	veh.4	veh.4	-
0.26	veh	state_waitForAck	state_changePL	-	-	-	-	-
0.26	veh	state_changePL	state_waitForAllAcks2	CHANGE_PL	veh.5	veh	veh	-
0.35	veh.5	state_platoonFollower	state_platoonFollower	ACK	veh	veh.4	veh	-
0.38	veh	state_waitForAllAcks2	state_splitDone	-	-	-	-	-
0.38	veh	state_splitDone	state_platoonLeader	CHANGE_Tg	multicast	veh	veh	-
0.38	veh	state_splitDone	state_platoonLeader	SPLIT_DONE	veh.4	veh	veh	-
0.42	veh.4	state_waitForSplitDone	state_waitForGap	-	-	-	-	-
0.92	veh.4	state_waitForGap	state_platoonLeader	GAP_CREATED	veh	veh.4	veh	Split_End

Figure 95: **Sample output from xxx_plnData.txt**

30 OMNET++ Runtime Environment

30.1 Tkenv Runtime Environment

Tkenv is a **graphical** runtime interface for simulations in OMNET++. Tkenv supports interactive simulation execution, animation, inspection, tracing and debugging. It is also useful for presentation and educational purposes, since it allows the user to get a detailed picture of the state and history of the simulation at any point of its execution. You can find detailed information about Tkenv in section 7 of OMNET++ User Guide. OMNET++ Tkenv window is shown in Figure 96.

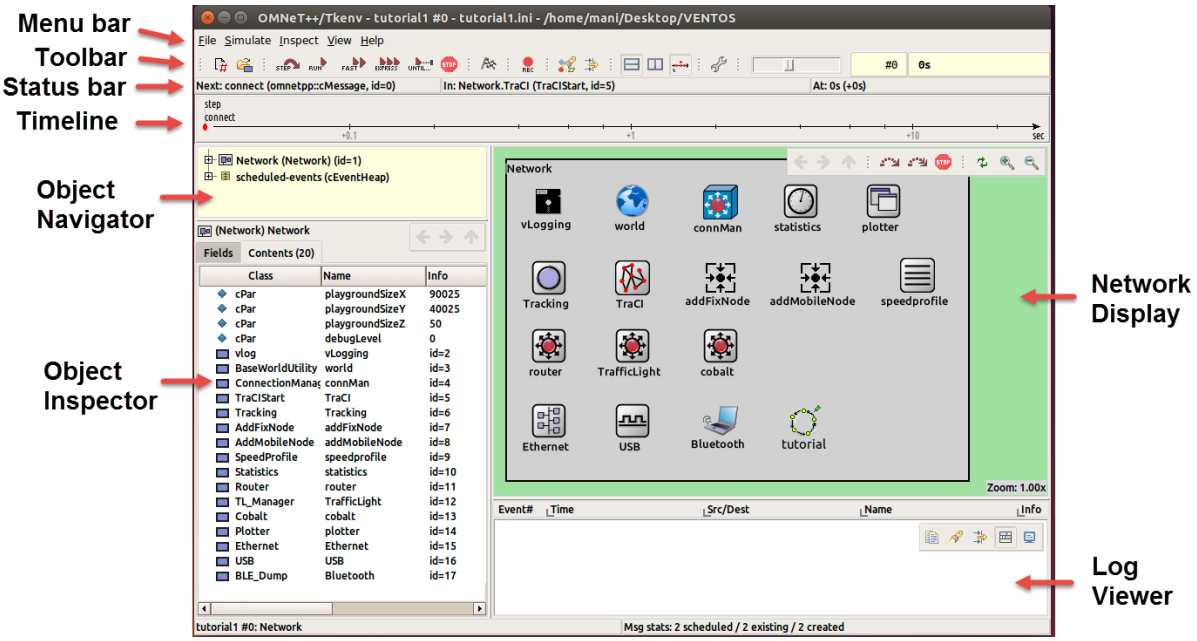


Figure 96: **OMNET++ Tkenv simulation environment**

Toolbar: The toolbar lets you access the most frequently used functions, such as controlling the simulation (stepping, running, stopping, etc.) and the current event number and simulation time.

Status bar: The status bar displays information about the next simulation event. When the simulation is running, it displays performance data like the number of events processed per second.

Timeline: Timeline displays the contents of the Future Events Set (FES) on a logarithmic time scale. Each dot represents a message (event). Clicking an event will focus it in the Object Inspector.

Object Navigator Area: Displays the hierarchy of objects in the current simulation and in the FES.

Object Inspector Area: Displays the contents and properties of the selected object.

Network Display Area: Displays the network or any module graphically. This is also where animation takes place.

Log Viewer Area: Displays the log of packets or messages sent between modules, or log messages output by modules during simulation.

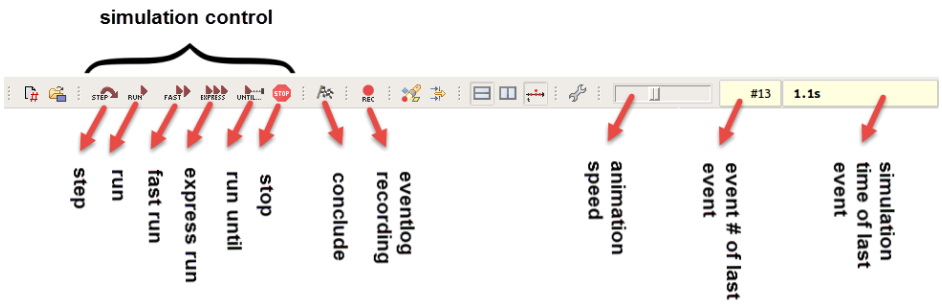


Figure 97: **Toolbar**

Figure 98: **Status bar**

Figure 99: **Timeline**

30.1.1 Simulation Control

Tkenv window provides six buttons on the toolbar to control the simulation as shown in Figure 97.

Step: Step lets you execute one simulation event at the front of Future Events Set (FES). The next event is shown on the status bar. The module where the next event will be delivered is also shown on the status bar and is highlighted with a red rectangle in the network display area.

Run (or Normal Run): In Run mode, the simulation runs with all tracing aids on. Message animation is active and inspector windows are updated after each event. You can change the animation speed using the slider on toolbar. Output messages are displayed in the main window and module output windows. You can fully interact with the user interface while the simulation is running (e.g. you can open inspectors, etc.).

Fast Run: In Fast mode, animation is turned off. The inspectors and the message output windows are updated every 500 milliseconds (the actual number can be set in File|Preferences). Fast mode is several times faster than the Run mode; the speed can increase by up to 10 times (or up to the configured event count).

Express Run: In Express mode, the simulation runs at about the same speed as with Cmdenv, all tracing disabled. Module log is not recorded. You can interact with the simulation only once in a while, thus the

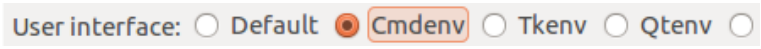
run-time overhead of the user interface is minimal. You have to explicitly click the 'Update inspector' button if you want a display update.

Run Until: You can run the simulation until a specified simulation time, event number or until a specific message has been delivered or canceled. This is a valuable tool during debugging sessions. It is also possible to right-click on an event in the simulation timeline and choose 'Run until Delivery of Message' menu item.

Stop: This button stops the running simulation. You can resume the simulation by clicking on either of the above buttons.

30.2 Cmdenv Runtime Environment

Cmdenv is a **command-line** runtime interface for simulations in OMNET++. You need to choose Cmdenv in your 'Run Configuration' as shown below:



This runtime environment allows batch execution. When running simulation batches, you can specify the number of processes allowed to run in parallel, so you can take advantage of multiple processors or processor cores. The progress of the batch can be monitored, and you can also kill processes from the batch if needed. You can find more information here.

31 Running Simulation from Terminal

The Eclipse IDE launcher helps you start the simulation easily in Tkenv or Cmdenv by building the command line and the environment for your program automatically. Sometimes it would be useful to start the simulation from outside the IDE. To get the command line used for running the simulation, run the simulation from the IDE as before. Then click on 'Debug' tab and right click on your program and select 'properties'. The IDE shows the currently used command line such as:

```
opp_run -u Cmdenv -c TrafficSignalControl -n ../.. -l ../../src/VENTOS omnetpp.ini -r 0
```

- `opp_run` allows starting simulation models that are linked as shared libraries
- `-u Cmdenv` tells omnetpp to run under Cmdenv
- `-c <configname>` option is used to select a configuration
- `-n` option is used to specify the NED path
- `-l` option is used to load additional shared libraries
- `-r <runnumber>` allows you to select runs

OMNET has a utility program called opp_runall that allows you to execute a simulation batch in Cmdenv. You must specify the whole command line you would use to run your batch in Cmdenv. For example:

```
opp_runall -j24 opp_run -u Cmdenv -c TrafficSignalControl -n ../.. -l ../../src/VENTOS
omnetpp.ini -r 0..30
```

The -j option is used to specify the maximum number of parallel runs allowed. opp_runall basically expands the -r option; for example, a simulation command line containing -r 0..2,8,10 will generate 5 commands with the -r 0, -r 1, -r 2, -r 8 and -r 10 options, respectively.

32 Running Simulation on a Remote Server

You can run the simulation on a remote server (Ubuntu Server preferably).

Step 1: Make sure that a ssh server is running on the remote server. To check if a ssh server is running on the server, type the following command in the server:

```
ssh localhost
```

If you receive the following message then the ssh server is not installed.

```
ssh: connect to host localhost port 22: Connection refused
```

Install openssh-server using the following command:

```
sudo apt-get install openssh-server
```

Now if you type `ssh localhost`, you should get the following message (type no):

```
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is fc:7e:9a:c7:33:86:47:4b:35:00:22:b7:be:5c:d1:c7.
Are you sure you want to continue connecting (yes/no)?
```

Step 2: SSH to the remote server as following. `mars.ece.ucdavis.edu` is the name of the server (you can use the server's IP address as well) and rubinet is the username.

```
ssh -X rubinet@mars.ece.ucdavis.edu
```

Note 1: The -X flag enables X11 forwarding and forwards the program that uses a GUI through the SSH connection. Whenever you launch a program that uses a GUI, it will pop up as if you were physically sitting at that computer (this method is preferable to using VNC which forwards the entire desktop).

Note 2: The `X server` should be running on **your** machine. Use the following command to check if X server is running on your machine:

```
xset q
```

If X server is not installed on your machine, then install it:

```
sudo apt-get install xorg openbox
```

Step 3: Follow the instructions in Part 1, Section 2 to install VENTOS on the remote server.

Step 4: While using OMNET++, the SSH session should be open. Closing the terminal windows or restarting your machine terminates the SSH session and the running simulation will be closed. To keep the SSH session open, you need to use `screen` or `tmux`. `tmux` (terminal multiplexer) is more powerful and we recommend it over `screen`. You can find more information here and here. With the help of tmux you can create sessions with multiple windows and panes like Figure 100.

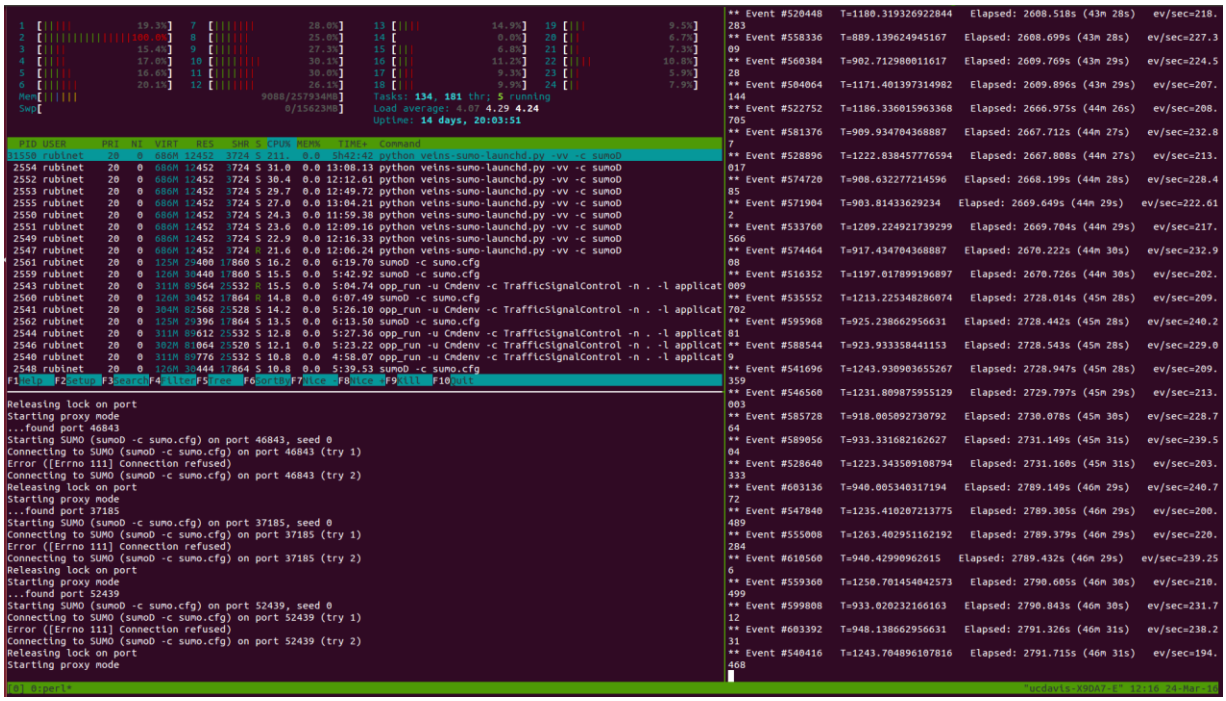


Figure 100

After `detaching` your tmux session, you can safely logoff from the remote machine; your process will keep running inside tmux. When you come back again and want to check the status of your process you can `attach` to your tmux session.

Note 1: The simulation saves multiple output txt files. In order to exchange files between your computer and remote server, you can use `filezilla`.

```
sudo add-apt-repository ppa:n-muench/programs-ppa
sudo apt-get update
sudo apt-get install filezilla
```