Documentation: Vehicular Ad-Hoc Networks Simulation Using OMNeT++ and SUMO

Instructor: Tokunbo Makanju

Subject: INCS-870 Project I

Guojin Tang (1248834)

Hedao Tian (1256221)

Weijia Gu (1244996)

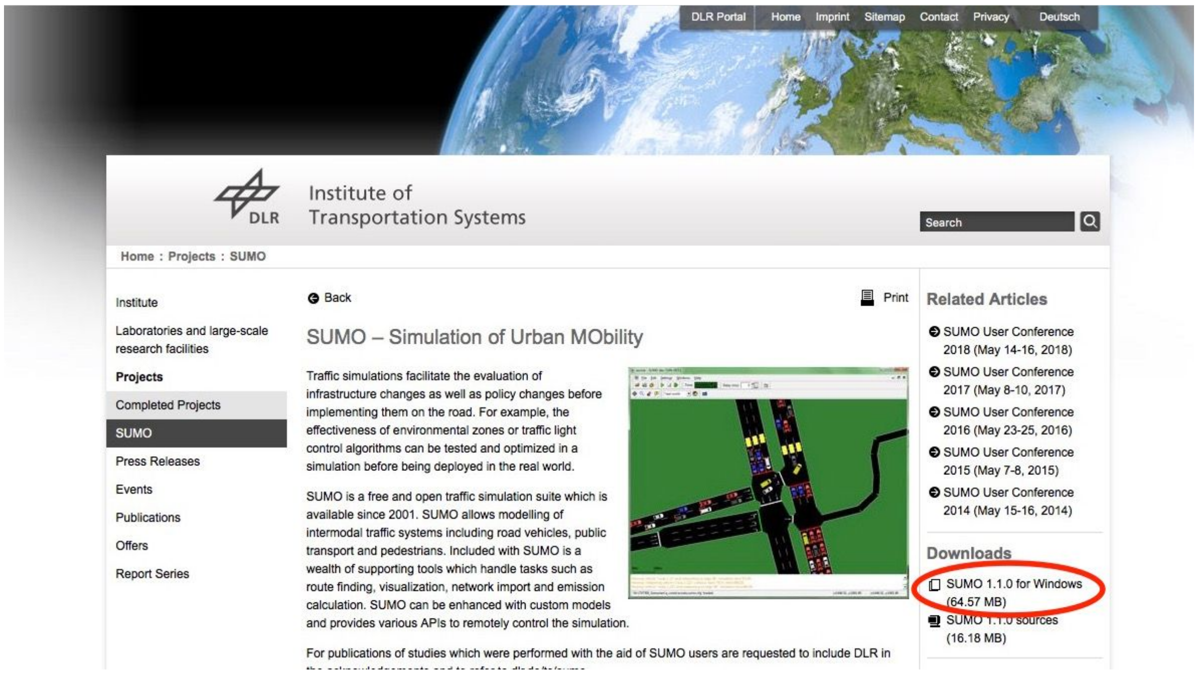
Table of contents

I. Introduction and Installation

A. SUMO

"**S**imulation of **U**rban **MO**bility" (SUMO) is an open source, highly portable, microscopic road traffic simulation package designed to handle large road networks. It allows to simulate how a given traffic demand which consists of single vehicles moves through a given road network. The simulation allows to address a large set of traffic management topics. It is purely microscopic: each vehicle is modelled explicitly, has an own route, and moves individually through the network. Simulations are deterministic by default but there are various options for introducing randomness.

- Step 1: download the SUMO-1.1.0 for Windows and save into the system



SUMO – Simulation of Urban MObility

Traffic simulations facilitate the evaluation of infrastructure changes as well as policy changes before implementing them on the road. For example, the effectiveness of environmental zones or traffic light control algorithms can be tested and optimized in a simulation before being deployed in the real world.

SUMO is a free and open traffic simulation suite which is available since 2001. SUMO allows modelling of intermodal traffic systems including road vehicles, public transport and pedestrians. Included with SUMO is a wealth of supporting tools which handle tasks such as route finding, visualization, network import and emission calculation. SUMO can be enhanced with custom models and provides various APIs to remotely control the simulation.

For publications of studies which were performed with the aid of SUMO users are requested to include DLR in

Related Articles

- SUMO User Conference 2018 (May 14-16, 2018)
- SUMO User Conference 2017 (May 8-10, 2017)
- SUMO User Conference 2016 (May 23-25, 2016)
- SUMO User Conference 2015 (May 7-8, 2015)
- SUMO User Conference 2014 (May 15-16, 2014)

Downloads

- SUMO 1.1.0 for Windows (64.57 MB)
- SUMO 1.1.0 sources (16.18 MB)

- Step 2: extract the zip file
- Step 3: go to the SUMO Installation location and run the executable application “SUMO-GUI.exe”
- Step 4: Now you can run SUMO samples in SUMO-GUI window.

B. OMNeT++

Objective **M**odular **Ne**twork **T**estbed in C**++** (OMNeT++) is a modular, component-based C++ simulation library and framework, primarily for building network simulators.

- Step 1: download the OMNeT++ Source Code
- Step 2: move the OMNeT++ archive to the directory you want to install it
- Step 3: extract the zip file
- Step 4: start the mingwenv.cmd in the by double-clicking it in Windows Explorer and a console with the MSYS *bash* shell will come up
- Step 5: build the OMNeT++by running *./configure*
- Step 6: run *omnetpp* to launch the OMNeT++ IDE

C. Veins

Veins is an Open Source vehicular network simulation framework. It ships as a suite of simulation models for vehicular networking. These models are executed by an event-based network simulator (OMNeT++) while interacting with a road traffic simulator (SUMO). Other components of Veins take care of setting up, running, and monitoring the simulation.

In this project, we use **Instant Veins Virtual Machine**. Instant Veins is a virtual machine used to quickly try out Veins. It is distributed as a single-file virtual appliance, ready for one-click import into software like Oracle VM VirtualBox, VMware Workstation Player, or any software supporting the Open Virtualization Format. Download Instant Veins.

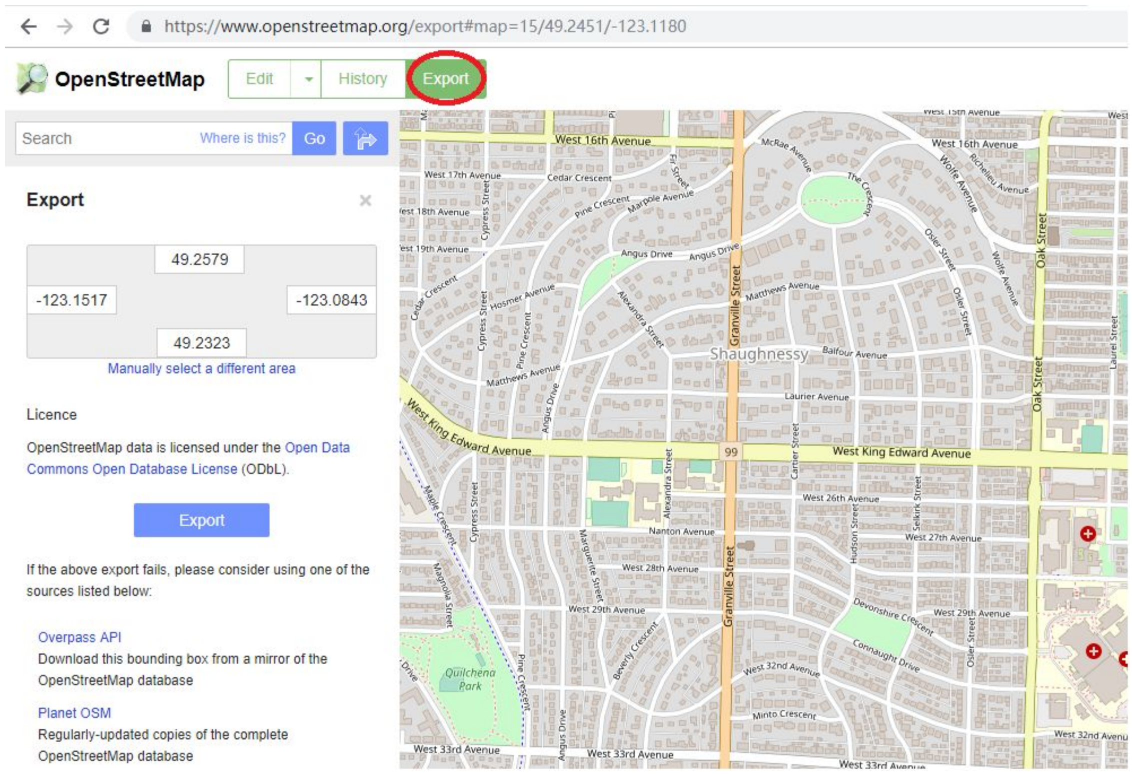
D. TraCI

Veins instantiates one network node per vehicle driving in SUMO. This task is handled by the TraCIScenarioManagerLaunchd module: it connects to a TraCI server (SUMO or sumo-launchd) and subscribes to events like vehicle creation and movement. **Tra**ffic **C**ontrol **I**nterface (TraCI) uses a TCP based client/server architecture to provide access to SUMO, giving access to a running road traffic simulation, it allows to retrieve values of simulated objects and to manipulate their behaviour "on-line".

II. Configuration

A. SUMO

You will need a map to achieve simulation, and there is a website, called OpenStreetMap, can provide this kind of map. Go to OpenStreetMap, press export, and grab the part of map you need.



https://www.openstreetmap.org/export#map=15/49.2451/-123.1180

Export

49.2579
-123.1517
-123.0843
49.2323

Manually select a different area

Licence

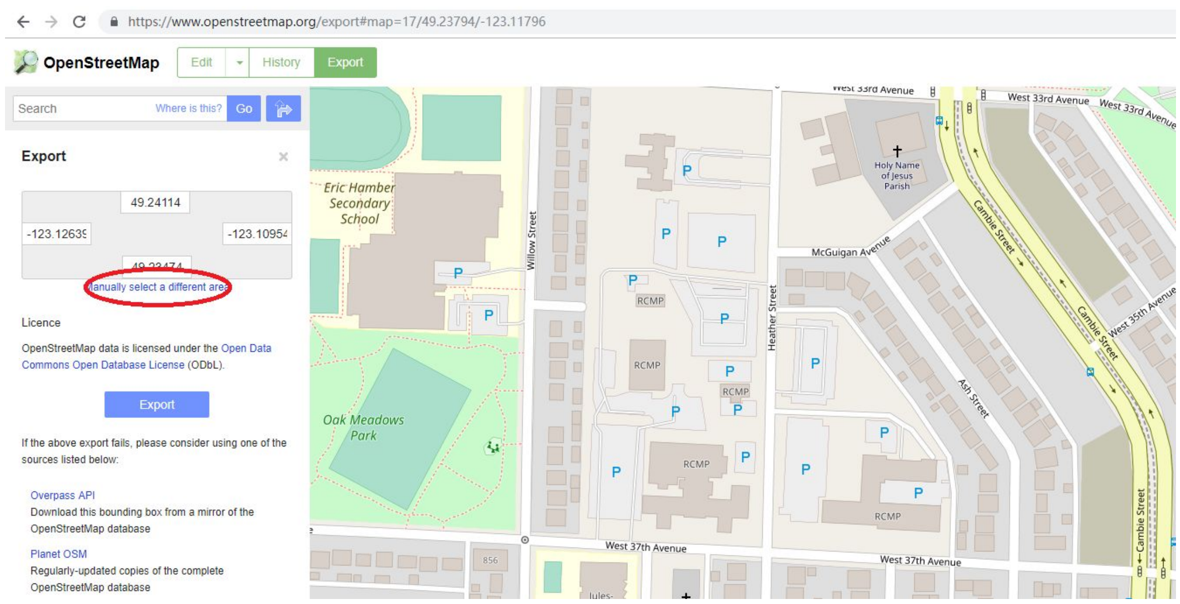
OpenStreetMap data is licensed under the Open Data Commons Open Database License (ODbL).

Export

If the above export fails, please consider using one of the sources listed below:

Overpass API
Download this bounding box from a mirror of the OpenStreetMap database

Planet OSM
Regularly-updated copies of the complete OpenStreetMap database


https://www.openstreetmap.org/export#map=17/49.23794/-123.11796
OpenStreetMap
Edit
History
Export
Search
Where is this?
Go
Export
49.24114
-123.1263
-123.1095
49.23474
Manually select a different area
Licence
OpenStreetMap data is licensed under the Open Data Commons Open Database License (ODbL).
Export
If the above export fails, please consider using one of the sources listed below:
Overpass API
Download this bounding box from a mirror of the OpenStreetMap database
Planet OSM
Regularly-updated copies of the complete OpenStreetMap database
Eric Hamber Secondary School
Oak Meadows Park
Holy Name of Jesus Parish
McGuigan Avenue
West 33rd Avenue
West 35th Avenue
West 37th Avenue
Cambie Street
Ash Street
Heather Street
Willow Street
RCMP

And you will download this part of map as OSM format. Now we need to transfer the format of the map which SUMO uses.

1. Find the file you downloaded. Open a terminal and execute the command.

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

   map.osm is the file you download

   You may get a warning said you need to configure SUMO_HOME.

   Find the source path of bin directory in SUMO.

2. Generate .rou.xaml file. It describes the behavior of vehicles in the map. This needs randomTrips.py in sumo to generate random action of vehicles. The command is

   ??/randomTrips.py -n xx.net.xml -e 1000 -l

   ?? means the path of randomTrips.py in sumo directory.

   -*e* is the time limit of the vehicles' action.

   -*l* means longer route has more probability that vehicles drives on.

3. Generate .poly.xml file. It describes terrain. We need to use polyconvert tool.

   polyconvert --net-file xx.net.xml --osm-files map.osm -o xx.poly.xml

4. Write .sumo.cfg file. It is a configuration file.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<configuration            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/sumoConfiguration.xsd">
 <input>
<net-file value="xx.net.xml"/>
<route-files value="xx.rou.xml"/>
<additional-files value="xx.poly.xml"/>
</input>
<time>
        <begin value="0"/>
        <end value="1000"/>
        <step-length value="0.1"/>
</time>
<report>
         <no-step-log value="true"/>
</report>
<gui_only>
         <start value="true"/>
</gui_only>
</configuration>
```

Time tage set the duration of simulation. Input tag needs all files we generated. After that we can use SUMO to open it.

```
sumo-gui xx.sumo.cfg
```

Also, every time you wanna run a simulation, you should run a command in a terminal.

```
src/veins/sumo-launchd.py -vv -c sumo-gui
```

This help you to start the simulation from SUMO and omnet++ at the same time, and start to listen the port.

III. Hands-on Practice with Examples

Veins is the project that we use to build and run a simulation of communicating cars.Veins itself is an open source project that connects two open source simulators:

OMNeT++ for network traffic, and SUMO for vehicle traffic. Writing a Veins project involves knowledge of all three systems. SUMO provides an API for getting and setting information about the cars in it's simulation, called TraCI. Veins will automatically handle the connection to TraCI when we run our project, while running the two simulators in parallel
https://github.com/burtonwilliamt/carlogicapi/tree/master/tutorials/VeinsTutorial.

The authors provided certain research and investigation result about hands-on practice to help students or further researchers to have a better insight before doing any projects on Veins. Below sections concluded practices of OMNet++, SUMO and Veins.

A. OMNet++ - TicToc

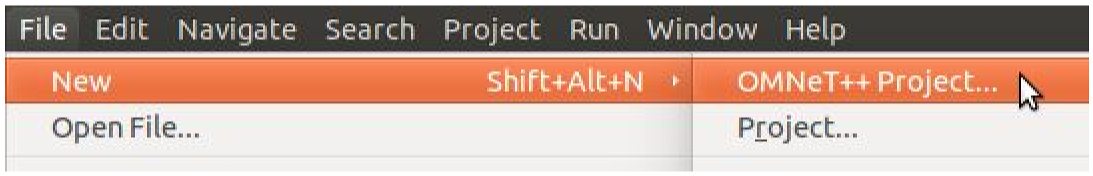
This tutorial guides you through building and working with an example simulation model, showing you along the way some of the commonly used OMNeT++ features.

1.1 The model

For a start, let us begin with a "network" that consists of two nodes. The nodes will do something simple: one of the nodes will create a packet, and the two nodes will keep passing the same packet back and forth. We'll call the nodes tic and toc.

1.2 Setting up the project

Start the OMNeT++ IDE by typing omnetpp in your terminal. Once in the IDE, choose New -> OMNeT++ Project from the menu.



A wizard dialog will appear. Enter tictoc as project name, choose Empty project when asked about the initial content of the project, then click Finish. An empty project will be created, as you can see in the Project Explorer.

The project will hold all files that belong to our simulation. In our example, the project consists of a single directory.

1.3 Adding the NED file

OMNeT++ uses NED files to define components and to assemble them into larger units like networks. We start implementing our model by adding a NED file. To add the file to the project, right-click the project directory in the Project Explorer panel on the left, and choose New -> Network Description File (NED) from the menu. Enter tictoc1.ned when prompted for the file name.

Once created, the file can be edited in the Editor area of the OMNeT++ IDE. The OMNeT++ IDE's NED editor has two modes, Design and Source; one can switch between them using the tabs at the bottom of the editor. In Design mode, the topology can be edited graphically, using the mouse and the palette on the right. In Source mode, the NED source code can be directly edited as text. Changes done in one mode will be immediately reflected in the other, so you can freely switch between modes during editing, and do each change in whichever mode it is more convenient. (Since NED files are plain text files, you can even use an external text editor to edit them, although you'll miss syntax highlighting, content assist, cross-references and other IDE features.)

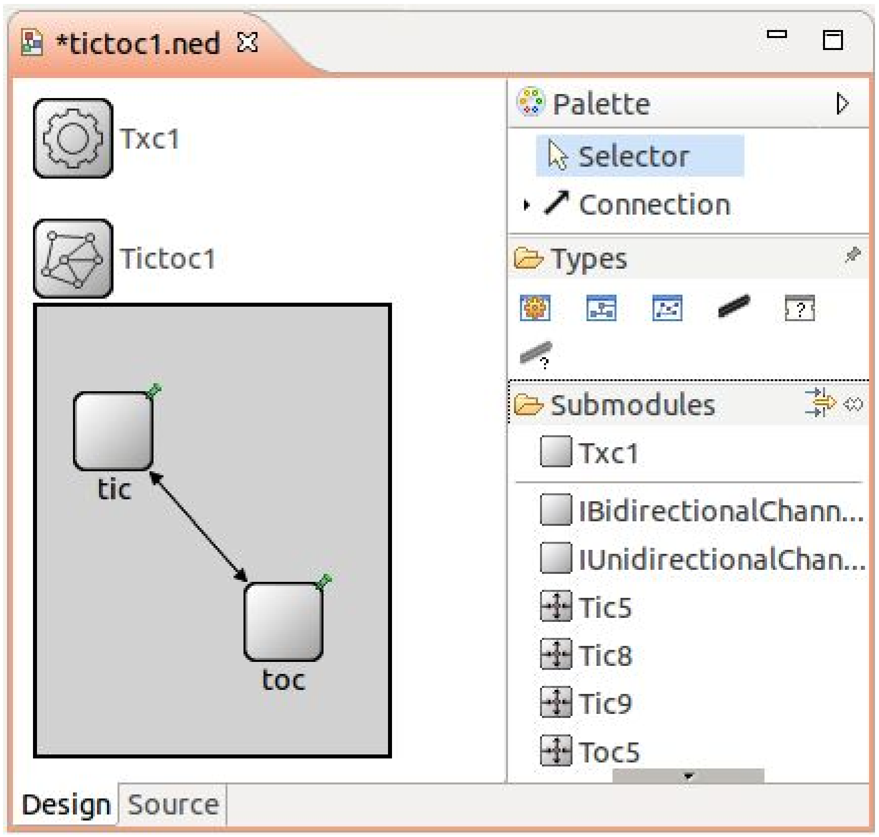
Switch into Source mode, and enter the following:

```
simple Txc1
{
    gates:
        input in;
        output out;
}

//
// Two instances (tic and toc) of Txc1 connected both ways.
// Tic and toc will pass messages to one another.
```

```
//
network Tictoc1
{
    submodules:
        tic: Txc1;
        toc: Txc1;
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

When you're done, switch back to Design mode. You should see something like this:



The first block in the file declares Txc1 as a simple module type. Simple modules are atomic on NED level. They are also active components, and their behavior is implemented in C++. The declaration also says that Txc1 has an input gate named in, and an output gate named out.

The second block declares Tictoc1 as a network. Tictoc1 is assembled from two submodules, tic and toc, both instances of the module type Txc1. tic's output gate is connected to toc's input gate, and vica versa. There will be a 100ms propagation delay both ways.

1.4 Adding the C++ files

We now need to implement the functionality of the Txc1 simple module in C++. Create a file named txc1.cc by choosing New -> Source File from the project's context menu (or File -> New -> File from the IDE's main menu), and enter the following content:

```
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;

/**
 * Derive the Txc1 class from cSimpleModule. In the Tictoc1 network,
 * both the `tic' and `toc' modules are Txc1 objects, created by OMNeT++
 * at the beginning of the simulation.
 */
class Txc1 : public cSimpleModule
{
  protected:
    // The following redefined virtual function holds the algorithm.
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

// The module class needs to be registered with OMNeT++
Define_Module(Txc1);
```

```
void Txc1::initialize()
{
   // Initialize is called at the beginning of the simulation.
   // To bootstrap the tic-toc-tic-toc process, one of the modules needs
   // to send the first message. Let this be `tic'.

   // Am I Tic or Toc?
   if (strcmp("tic", getName()) == 0) {
       // create and send first message on gate "out". "tictocMsg" is an
       // arbitrary string which will be the name of the message object.
       cMessage *msg = new cMessage("tictocMsg");
       send(msg, "out");
   }
}

void Txc1::handleMessage(cMessage *msg)
{
   // The handleMessage() method is called whenever a message arrives
   // at the module. Here, we just send it to the other module, through
   // gate `out'. Because both `tic' and `toc' does the same, the message
   // will bounce between the two.
   send(msg, "out"); // send out the message
}
```

The Txc1 simple module type is represented by the C++ class Txc1. The Txc1 class needs to subclass from OMNeT++'s cSimpleModule class, and needs to be registered in OMNeT++ with the Define_Module() macro.

1.5 Adding omnetpp.ini

To be able to run the simulation, we need to create an omnetpp.ini file. omnetpp.ini tells the simulation program which network you want to simulate (as

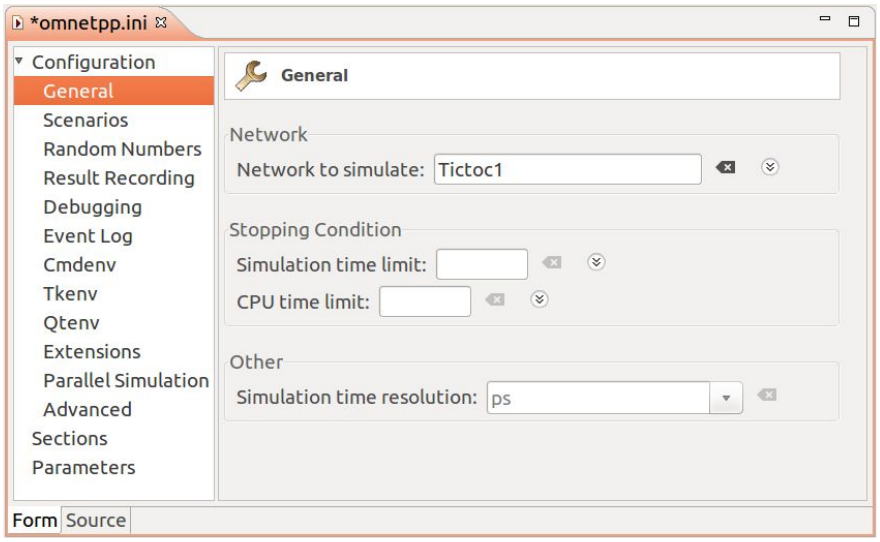
NED files may contain several networks), you can pass parameters to the model, explicitly specify seeds for the random number generators, etc.

Create an omnetpp.ini file using the File -> New -> Initialization file (INI) menu item. The new file will open in an Inifile Editor. As the NED Editor, the Inifile Editor also has two modes, Form and Source, which edit the same content. The former is more suitable for configuring the simulation kernel, and the latter for entering module parameters.

For now, just switch to Source mode and enter the following:

```
[General]
network = Tictoc1
```

You can verify the result in Form mode:



General

Network

Network to simulate: Tictoc1

Stopping Condition

Simulation time limit:

CPU time limit:

Other

Simulation time resolution: ps

tictoc2 and further steps will all share a common omnetpp.ini file.

We are now done with creating the first model, and ready to compile and run it. To run, just hit the Run button on the top left of the screen. For troubleshooting and visit original files, please visit:
https://docs.omnetpp.org/tutorials/tictoc/part2/

B. SUMO

This tutorial is an overview of the Hello Sumo Tutorial and provides step by step tutorial for students to get on board of Sumo.

- Create a directory to place your files in. In my case, I called this Hello-Sumo/.
- The two necessary components to run a simulation are: a road network(described in the .net.xml) and cars to run on the network(described by the .rou.xml file). The first of these is generated from two parts. The first is a list of nodes, or junctions in the road network(a .nod.xml file). The second is a list of edges, which describes how those junctions are connected(a .edg.xml file). Note: these two files are only used to generate the .net.xml file.

- First create a file called hello.nod.xml, and in it describe three nodes like this:

```
<!-- filename: hello.nod.xml -->
<nodes>
   <node id="1" x="-250.0" y="0.0" />
   <node id="2" x="+250.0" y="0.0" />
   <node id="3" x="+251.0" y="0.0" />
</nodes>
```

These three nodes are junctions in our road. The id is a unique id, and the coordinates are specified.

- Then create your edges file called hello.edg.xml, then describe the two edges like this:

```
<!-- filename: hello.edg.xml -->
<edges>
  <edge from="1" id="1to2" to="2" />
  <edge from="2" id="out" to="3" />
</edges>
```

These two edges are connected the "from" node to the "to" node. CRAZY right? Use the node id to specify this, and assign the edge a unique id as well.

- After you have your nodes and your edges, you can now generate your net.xml file. Do this by running the following command:

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

At this point you can run your net.xml file in sumo to see your road network. Do this with the following command:

```
sumo-gui -n hello.net.xml
```

- Once you're properly satisfied with your majestic road network, you can put a vehicle on it. Do this by creating the hello.rou.xml routes file. Paste the following into it:

```
<!-- filename: hello.rou.xml -->
<routes>
      <vType accel="1.0" decel="5.0" id="Car" length="2.0" maxSpeed="100.0" sigma="0.0" />
   <route id="route0" edges="1to2 out"/>
   <vehicle depart="1" id="veh0" route="route0" type="Car" />
</routes>
```

This file describes three things:

1. A vehicle
2. A route, described by the edges it travels
3. An instance of the vehicle and route, with a start time
4. The last thing we should do is create a sumo config file. Name this hello.sumo.cfg, and past the following into it.

```
<!-- filename: hello.sumo.cfg -->
<configuration>
```

```
 <input>
     <net-file value="hello.net.xml"/>
     <route-files value="hello.rou.xml"/>
     <gui-settings-file value="hello.settings.xml"/>
 </input>
 <time>
     <begin value="0"/>
     <end value="10000"/>
 </time>
</configuration>
```

This file will setup the simulation, pointing the sumo at the files that we want to use. You can see that there is a gui file specified. This file will be called hello.settings.xml. Put the following in that file:

```
<!-- filename: hello.settings.xml -->
<viewsettings>
   <viewport y="0" x="250" zoom="100"/>
   <delay value="100"/>
</viewsettings>
```

- When you're ready to run your simulation, do so by running the following command: sumo-gui -c hello.sumo.cfg This will open up the simulation application, with your sim loaded up. The hello.settings.xml file we made had the setting <delay value="100"/>which set the delay between steps to 100ms. You can see this delay value in the top middle of the toolbar at the top of the page. You can change this value here, then run the simulation with the play button. To the left is a reset button if you want to start your simulation again.

C. Veins

The author has already introduced the configuration of Veins files and the practice of OMNET++ with TicToc example. In this section we will learn the existing veins example’s built inside the VM. Each file will be analyzed.

TutorialScenario.ned

Firstly, we need to define the network that is going to describe our example. This is described in src/TutorialSenario.ned of this project. If you open .ned files in the OMNET++ IDE, you'll probably see a graphical representation of the network; to see the source code, click the Source tab in the bottom left of the window. As you can see, this file doesn't do much except extend Scenario (inherit from Scenario). Open up Scenario.ned from veins/src/veins/nodes. You might need to open as text / source if you are using the OMNeT++ IDE.

Car.ned

The Car class that we are using is located in the package: org.car2x.veins.nodes.Car which is the same place as Scenario.ned. Car is not in fact a simple class. It declares objects for an application, a NIC (Network Interface Card), and mobility. As well as setting up the connections between the NIC and the application.

TraCIMobility

TraCIMobility is the simple class that holds driving-related code. It is in org.car2x.veins.modules.mobility.traci.TraCIMobility and has a .ned, .h, and .cc file. The .ned file pretty much only defines some parameters. But the C++ code is the real deal. The TraCIMobility::changePosition() function moves the node in the OMNeT++ simulation according to the messages from the SUMO sim over TraCI.

TutorialAppl

The application part of the Car is in the TutorialAppl class which is comprised of a .ned, a .h, and a .cpp in the src folder included in the VeinsTutorial project here. It handles the messages from other Cars, sending and receiving them. In this example, I've made the cars slow down or speed up to the surrounding car's speeds when they receive a message from another car. This is supposed to sync the Car's speeds, and it serves as a simple example. Here is a function that gets called everytime SUMO updates the position of the Car.

```
void TutorialAppl::handlePositionUpdate(cObject* obj) {
    BaseWaveApplLayer::handlePositionUpdate(obj);

    //sends message every 5 seconds
    if (simTime() - lastSent >= 5) {
        std::string message = std::to_string(mobility->getSpeed());
        sendMessage(message);
        lastSent = simTime();
    }
}
```

And here is the code that matches the speed upon receiving a message:

```
void TutorialAppl::onData(WaveShortMessage* wsm) {
    //Receive a message with a target speed, slow down / speed up to that speed
    float message_speed = atof(wsm->getWsmData());
    traciVehicle->slowDown(message_speed, 1000); //slow down over 1s
}
```

omnetpp.ini

The omnetpp.ini file is where all the parameters in .ned network files get set. It serves as a configuration file for the Veins simulation, and the file that is Run to start a simulation. The parameters can be confusing to find because of the inheritance structure and the star expansions. Just know that the argument names are dotted off of the variable names, not the classes. And star expansion works in the Unix way. The beginning *. before everything is the Tutorial Scenario object. And ** is any number of nested *.s.

Running the VeinsTutorial Example

Assuming that you have Veins and SUMO installed properly, open up the OMNet++ IDE with the omnetpp terminal command. You can choose the carlogicapi repo as your workspace or make a folder in your Documents for it. Import Veins and this VeinsTutorial projects in the IDE with File > Import > General: Existing Projects into Workspace and select the carlogicapi/tutorials/VeinsTutorial/ directory for the tutorial. Do the same but

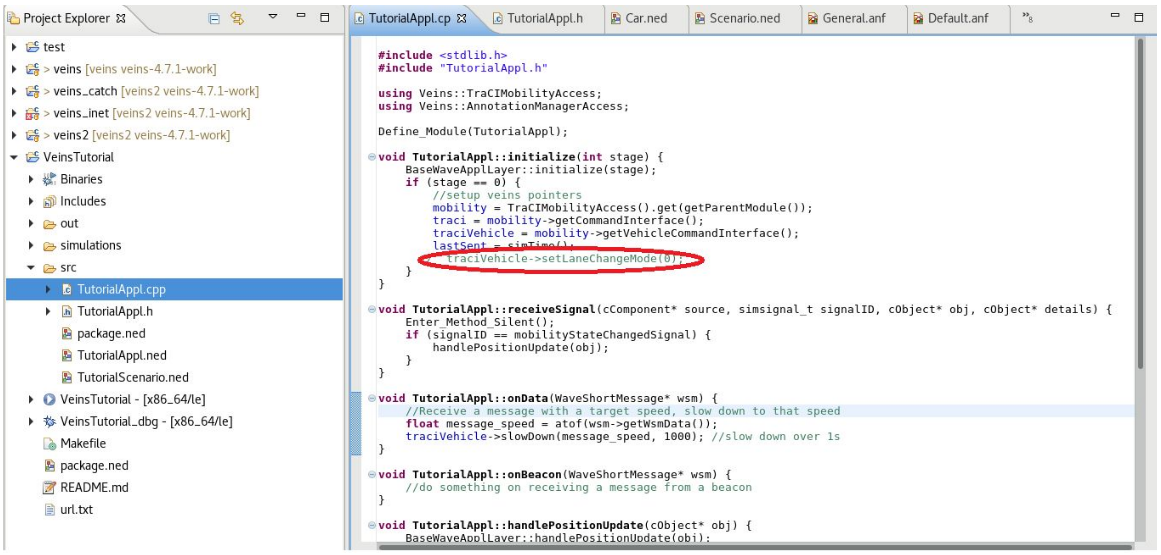
Import Veins from its source location. Build the VeinsTutorial project with Ctrl-B or Project > Build All. Before we run our project, we need to run a script that will start a parallel SUMO simulation when we start our Veins/OMNeT++ sim. To do this, open a seperate terminal and run: ./sumo-launchd.py -vv from the root of the Veins directory. Then, you are ready to run the example. Right click on VeinsTutorial/simulations/omnetpp.ini in the Project Explorer and Run As > OMNet++ Simulation. Click the Run button or press F5 to start the simulation in the window that pops up.

D. Troubleshooting

No such file "BaseWaveAppLayer.h"

This was caused by Veins versions mismatch.
Solution:

1. right click on Veins Tutorial Project Folder > Properties > OMNeT++ > makemake > Options > Compile Tab and include the path to the veins-4.6 src folder.
2. Note: There is a function does not exist in the original file of veins but was written in the code. It seems to set some parameters to be 0. Delete it and you can run the simulation.



```
#include <stdlib.h>
#include "TutorialAppl.h"

using Veins::TraCIMobilityAccess;
using Veins::AnnotationManagerAccess;

Define_Module(TutorialAppl);

void TutorialAppl::initialize(int stage) {
    BaseWaveApplLayer::initialize(stage);
    if (stage == 0) {
        //setup veins pointers
        mobility = TraCIMobilityAccess().get(getParentModule());
        traci = mobility->getCommandInterface();
        traciVehicle = mobility->getVehicleCommandInterface();
        lastSent = simTime();
        // traciVehicle->setLaneChangeMode(0);
    }
}

void TutorialAppl::receiveSignal(cComponent* source, simsignal_t signalID, cObject* obj, cObject* details) {
    Enter_Method_Silent();
    if (signalID == mobilityStateChangedSignal) {
        handlePositionUpdate(obj);
    }
}

void TutorialAppl::onData(WaveShortMessage* wsm) {
    //Receive a message with a target speed, slow down to that speed
    float message_speed = atof(wsm->getWsmData());
    traciVehicle->slowDown(message_speed, 1000); //slow down over 1s
}

void TutorialAppl::onBeacon(WaveShortMessage* wsm) {
    //do something on receiving a message from a beacon
}

void TutorialAppl::handlePositionUpdate(cObject* obj) {
    BaseWaveApplLayer::handlePositionUpdate(obj);
```

For more troubleshooting info and visit original files, please visit

(https://github.com/burtonwilliamt/carlogicapi/tree/master/tutorials/VeinsTutorial)

IV. Author Organized Useful Links

A. Creating Routes from Scratch in SUMO (Chinese/English version)

http://www.voidcn.com/article/p-pactztyv-bec.html

http://alibalador.blogspot.com/2013/02/working-with-sumo.html?view=sideb
ar

B. Car Behaviour Models (lane changing, following...)

https://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes

C. Guide of generating random route and its framework

http://alibalador.blogspot.com/2013/03/generating-route-file-for-using-with.h
tml?view=sidebar:

https://github.com/GRCDEV/mixim-sommer

D. Veins=SUMO+OMNET++

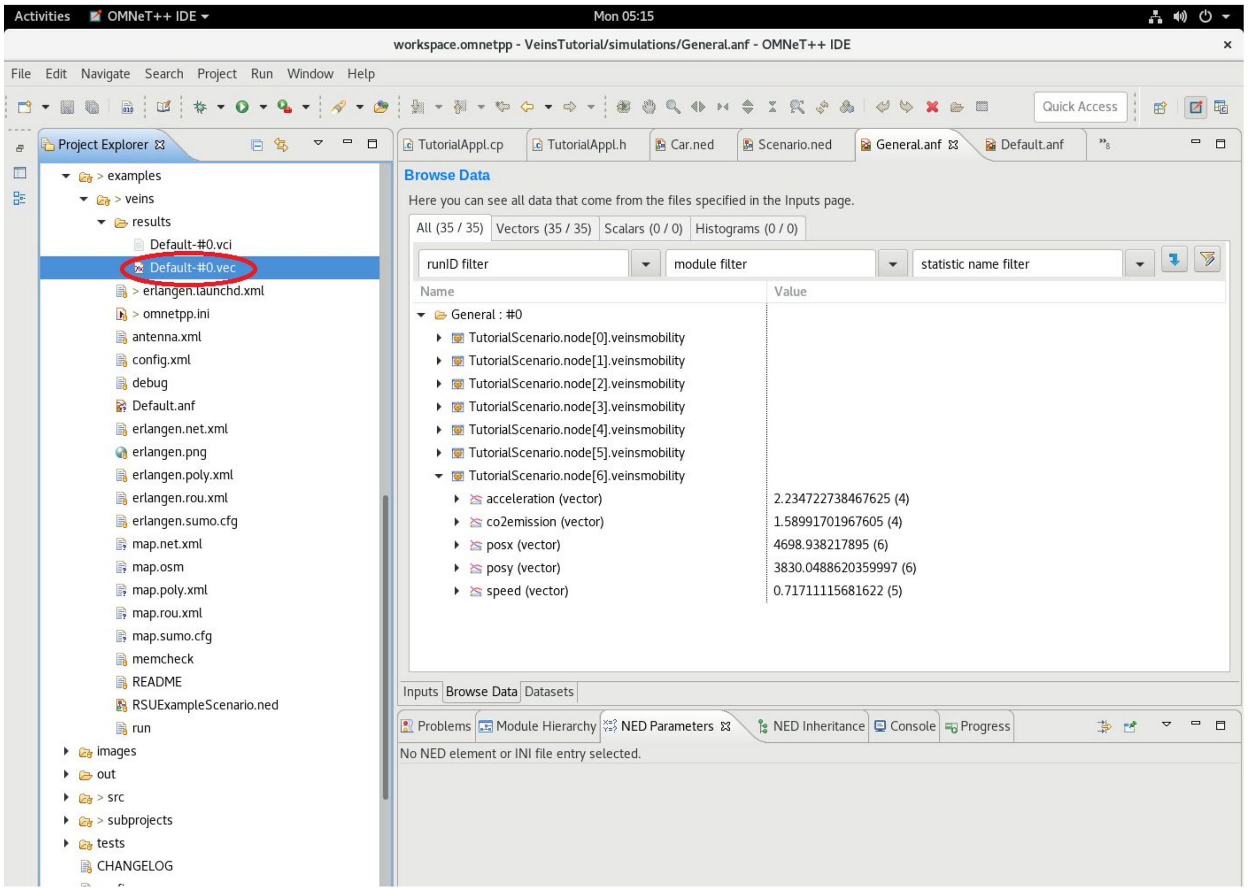
https://www.youtube.com/watch?v=39FufLl-Gso

E. TraCI - Sumo: Functions Explained

https://sumo.dlr.de/wiki/TraCI

V. Data Collection

After you running the simulation, the data should be automatically collected, and it should be stored in results directory.



workspace.omnetpp - VeinsTutorial/simulations/General.anf - OMNeT++ IDE

Browse Data

Here you can see all data that come from the files specified in the Inputs page.

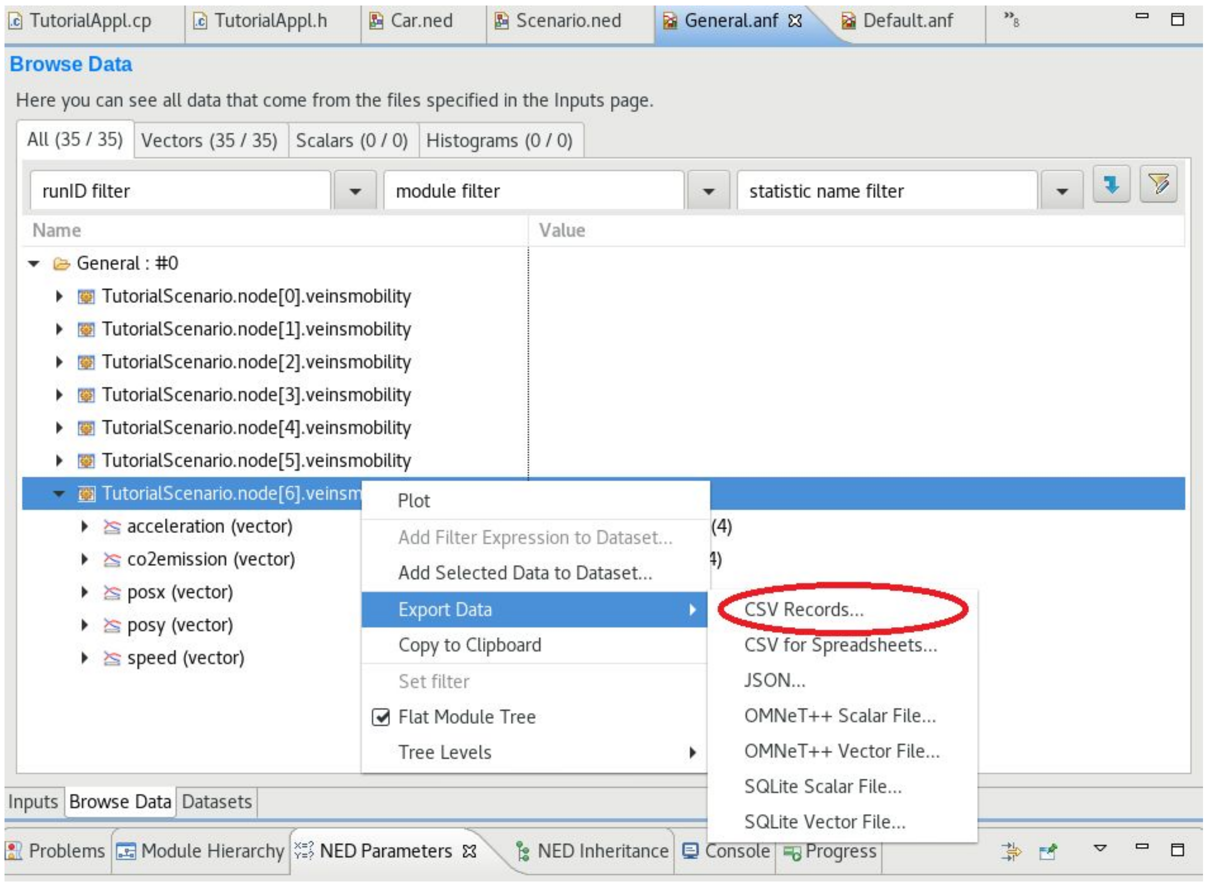
All (35 / 35) | Vectors (35 / 35) | Scalars (0 / 0) | Histograms (0 / 0)

```
Name                                          Value
General : #0
  TutorialScenario.node[0].veinsmobility
  TutorialScenario.node[1].veinsmobility
  TutorialScenario.node[2].veinsmobility
  TutorialScenario.node[3].veinsmobility
  TutorialScenario.node[4].veinsmobility
  TutorialScenario.node[5].veinsmobility
  TutorialScenario.node[6].veinsmobility
    acceleration (vector)                     2.234722738467625 (4)
    co2emission (vector)                      1.58991701967605 (4)
    posx (vector)                             4698.938217895 (6)
    posy (vector)                             3830.0488620359997 (6)
    speed (vector)                            0.71711115681622 (5)
```

```
examples
  veins
    results
      Default-#0.vci
      Default-#0.vec
    erlangen.launchd.xml
    omnetpp.ini
    antenna.xml
    config.xml
    debug
    Default.anf
    erlangen.net.xml
    erlangen.png
    erlangen.poly.xml
    erlangen.rou.xml
    erlangen.sumo.cfg
    map.net.xml
    map.osm
    map.poly.xml
    map.rou.xml
    map.sumo.cfg
    memcheck
    README
    RSUExampleScenario.ned
    run
images
out
src
subprojects
tests
CHANGELOG
```

No NED element or INI file entry selected.

Also, you can export these data to any format you want. Usually, python can handle with CSV file.



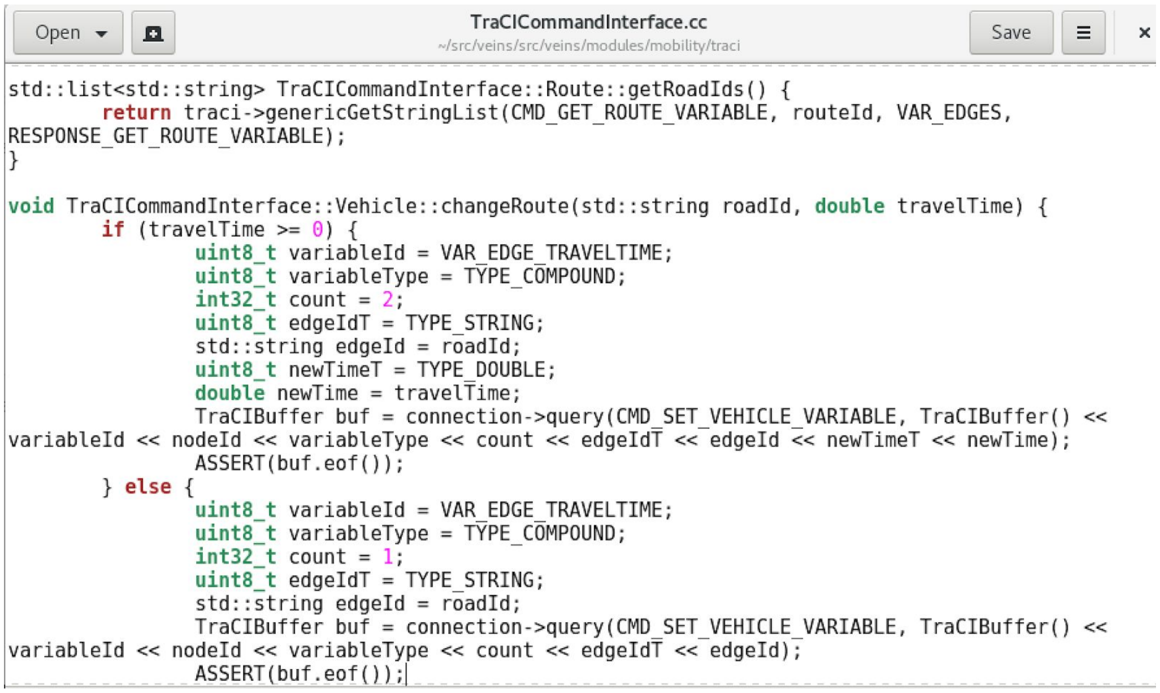
VI. <u>Conclusion & Suggestion</u>

The complete simulation process is:

1. SUMO configuration, you need all file to complete simulation.
2. OMNET++, configure network. You can go to read the part OMNET++ tictoc.
3. Use Traci to archive some function, this will need you code something. You can read the documentation of SUMO to complete this part. The website is the first URL in the reference. The possible function you need to use for congestion attack can be found in https://sumo.dlr.de/wiki/TraCI.
4. Start the simulation and collect the data.

   Note: You can replace veins example's SUMO file to your SUMO file to run a simple simulation.

Authors have done all the previous provided exercises and built our own maps with pre planned routes as well as system generated random routes. By going through some troubleshooting steps and building up the connection between SUMO and OMNet++ via Veins TraCI. We were able to simulate the traffic as we defined. However, to simulate the congestion attack or other related vehicular attacks as the research papers provided, it requires further investigation with certain programming skills especially focusing on C++ on with an eye of modification of TraCI files and OMNet++ files if possible. For example, a part of malware can be written into below's function to make one of the input parameter "roadId" to be always constant, thus all route change will leads to one defined route to cause huge amount of congestion on the simulation map.



```
std::list<std::string> TraCICommandInterface::Route::getRoadIds() {
        return traci->genericGetStringList(CMD_GET_ROUTE_VARIABLE, routeId, VAR_EDGES,
RESPONSE_GET_ROUTE_VARIABLE);
}

void TraCICommandInterface::Vehicle::changeRoute(std::string roadId, double travelTime) {
        if (travelTime >= 0) {
                uint8_t variableId = VAR_EDGE_TRAVELTIME;
                uint8_t variableType = TYPE_COMPOUND;
                int32_t count = 2;
                uint8_t edgeIdT = TYPE_STRING;
                std::string edgeId = roadId;
                uint8_t newTimeT = TYPE_DOUBLE;
                double newTime = travelTime;
                TraCIBuffer buf = connection->query(CMD_SET_VEHICLE_VARIABLE, TraCIBuffer() <<
variableId << nodeId << variableType << count << edgeIdT << edgeId << newTimeT << newTime);
                ASSERT(buf.eof());
        } else {
                uint8_t variableId = VAR_EDGE_TRAVELTIME;
                uint8_t variableType = TYPE_COMPOUND;
                int32_t count = 1;
                uint8_t edgeIdT = TYPE_STRING;
                std::string edgeId = roadId;
                TraCIBuffer buf = connection->query(CMD_SET_VEHICLE_VARIABLE, TraCIBuffer() <<
variableId << nodeId << variableType << count << edgeIdT << edgeId);
                ASSERT(buf.eof());
```

When Veins is managing the simulation, Sumo is listening on the port as a server and wait for the command to manipulate the car's movements, thus OMNet++ can be the subject that manipulate the whole scenario when Veins TraCI is providing the initialized functions for them to do. For example, TraCI provides the original function/definition for a node/car to transmit messages, when OMNet++ is actually using that function to transmit messages on running or in the source code.

Reference

https://sumo.dlr.de/wiki/TraCI/Interfacing_TraCI_from_Python

https://sumo.dlr.de/wiki/Tutorials

https://www.openstreetmap.org/#map=3/71.34/-96.82

https://sumo.dlr.de/wiki/TraCI

https://blog.csdn.net/renguoqing1001/article/details/52743179

https://sumo.dlr.de/wiki/SUMO_User_Documentation

https://docs.omnetpp.org/tutorials/tictoc/part1/