SUMO mobility in NS3

Adil Alsuhaim (aalsuha@clemson.edu)

September 21, 2020

`SUMO` (Simulation of Urban MObility) is an open source, highly portable, microscopic and continuous multi-modal traffic simulation package designed to handle large networks. It can be used to simulate vehicular traffic, from which we can export a trace of vehicle positions that can be used in `ns-3` simulations. The steps to perform this are:

1. Create a road map in `SUMO`. The format is and `XML` file that we will name `map.net.xml`
2. Define traffic routes that are used in a `SUMO` simulation. These flows can be generated randomly with the `randomTrips.py` tool, or created manually be defining flows. We will name these file with a `.rou.xml` extension.
3. Create a `SUMO` simulation file that points to the map file, and a list of route files.
4. Run a `SUMO` simulation to generate a trace that we name `sumoTrace.xml`
5. Convert the trace to `ns-2` mobility file using `traceExporter.py` tool.
6. Use the generated `ns-2` trace with `ns-3` by configuring it with `Ns2MobilityHelper`.

1 SUMO Overview

An installation of `SUMO` will contain a list of binaries and python scripts. I am mainly interested in the following binaries

- **`sumo`** Runs a `SUMO` simulation in command-line, i.e. no graphics. This is what we use to generate an `XML` trace of a simulation.
- **`sumo-gui`** A GUI tool to run a `SUMO` simulation.
- **`netedit`** A GUI tool to create SUMO maps.
- **`netconvert`** A tool to generate a `SUMO` network map from existing files. This can be used to convert a map in OpenStreetMap format to a `SUMO` map, but it can also be used to generate a map from files representing nodes and edges.

On Ubuntu, if you installed `SUMO` using `apt`, these binaries will be in `/usr/bin` directory, and therefore, they will be part of the `$PATH`. Otherwise, if you have built `SUMO` from sources, then you will have to add it to the path. For example, if we installed it at `$HOME/sumo` then we will need to edit `~/.bashrc` with these lines

```
export SUMO_HOME=$HOME/sumo
export PATH=$PATH:$SUMO_HOME/bin:$SUMO_HOME/tools
```

The `python` scripts that we will use are

- **`randomTrips.py`** Using a `SUMO` map, this tool will generates random trips and creates a `XML` file for those trips. Those random trips start at one junction on the map, and end at another.
- **`traceExporter.py`** We use this tool to convert a `SUMO` trace to `ns-2` trace.

If you have installed `SUMO` using `apt` on Ubuntu, these tools will be under `/usr/share/sumo/tools`. You may want to add the path to your `$PATH`

2 Creating a SUMO map

2.1 Using OpenStreetMap

OpenStreetMap (https://www.openstreetmap.org) is a collaborative project to create a free editable map of the world. Go to the website, and search for a specific area anywhere in the world. You will have too zoom into part of the map so that you will have a smaller map to export. The map view have an Export button which will export the current view as a map named map.osm

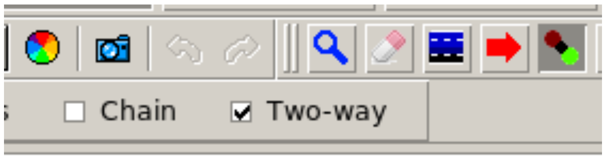
Once you have a map in OpenStreetMap format, convert it to a SUMO map as follows:

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

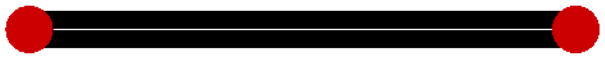
This will generate the file map.net.xml

2.2 Using netedit tool

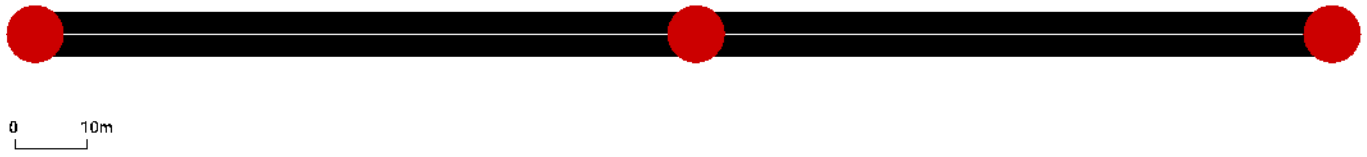
You can create a map using the netedit GUI tool. You can do so many things in this tool. Start by creating a new network with File > New Network to create a new map. Click on the junction tool in the tool bar. Check the "Two-way" box to create a Two-way road.



Click somewhere on the white map area to create a junction, then move the mouse to create another junction. This process will create a road between the two red points.

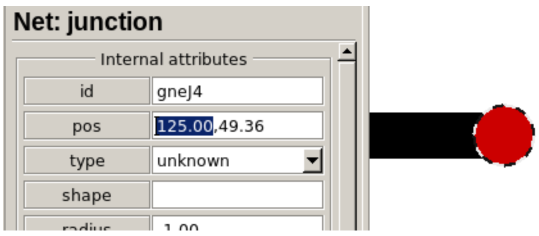


While we still have the junction mode active, click on the second junction created to make, which will make it turn green, then click somewhere else to create a third junction. This should look like this:



This is the bare minimum for a SUMO map to work, an independent set of road segments should at least have three segment to work properly.

Notice the scale (0-10m), indicating that this is a very short road. Let's now click the inspection tool from the tool bar, then click on one of the red edges to change its position. Edit the position by changing 125 to 1000 so that the road length is at about 1000 meters long.



Net: junction

Internal attributes

id	gneJ4
pos	125.00,49.36
type	unknown
shape	
radius	1.00

Save your file as map.net.xml and you are done!

Note. You can make your map more complex if you want to. You can set a number of lanes on a certain edge, or edit junctions and intersections and change their types. An intersection can have a traffic signal, or can have an "all-way stop".

2.3 From scratch by creating nodes & edges

You can create XML files for nodes and edges and then compile it into a SUMO map. In this example, we will create an intersection of three-lane highways with a traffic signal in the middle.

```xml
<nodes>
  <node id="west" x="0.0" y="25.0" />
  <node id="east" x="1000.0" y="25.0" />
  <node id="north" x="500.0" y="-525.0" />
  <node id="south" x="500.0" y="525.0" />
  <node id="mid" x="500.0" y="25.0" type="traffic_light"/>
</nodes>
```

Listing 1: Nodes file nodes.nod.xml

```xml
<edges>
  <!-- West to East-->
  <edge from="west" id="eastbound1" to="mid" numLanes="3" />
  <edge from="mid" id="eastbound2" to="east" numLanes="3" />

  <!-- East to West-->
  <edge from="east" id="westbound1" to="mid" numLanes="3" />
  <edge from="mid" id="westbound2" to="west" numLanes="3" />

  <!-- North to South-->
  <edge from="north" id="southbound1" to="mid" numLanes="3" />
  <edge from="mid" id="southbound2" to="south" numLanes="3" />

  <!-- South to North-->
  <edge from="south" id="northbound1" to="mid" numLanes="3" />
  <edge from="mid" id="northbound2" to="north" numLanes="3" />
</edges>
```

Listing 2: Edges file edges.edg.xml

Then use netconvert to create the SUMO map

```
netconvert --node-files=nodes.nod.xml --edge-files=edges.edg.xml --output-file=map.net.xml
```

3 Creating routes

3.1 Creating flows manually

Alternatively, we can create a route file manually by defining traffic flows. Let's create the file routes.rou.xml manually as follows

```xml
<routes>
  <vtype id="car_type" type="passenger" accel=3.5 decel=2.2 />
  <vtype id="bus_type" type="bus" accel=1.2 decel=1.8 />

  <flows>
    <interval begin="0" end="360">
      <flow id="0" from="genE1" to="genE2" probability="0.9" type="car_type"/>
      <flow id="0" from="-genE2" to="genE1" probability="0.9" type="car_type"/>

      <flow id="0" from="genE1" to="genE2" probability="0.01" type="bus_type"/>
      <flow id="0" from="-genE2" to="genE1" probability="0.01" type="bus_type"/>
    </interval>
  </flows>
</routes>
```

You will have to make sure that the edges in the `from` and `to` fields are valid. Notice that we created two types of vehicles with their corresponding acceleration & deceleration limits. Notice how the bus accelerates slower than a normal car. You can create as many types as you want. Refer to SUMO documentation for information about all kinds of vehicle classes you can use.

3.2 Using randomTrips.py

The easiest way to create routes is by using the `randomTrips.py` tool.

```
randomTrips.py -n map.net.xml -r routes.rou.xml
```

Which will create the routes file `routes.rou.xml` and we are done!

We can pass more parameters to `randomTrips.py` to control the frequency of vehicles generation, vehicle classes and attributes if we want to. I'll touch on that later.

4 SUMO simulation file

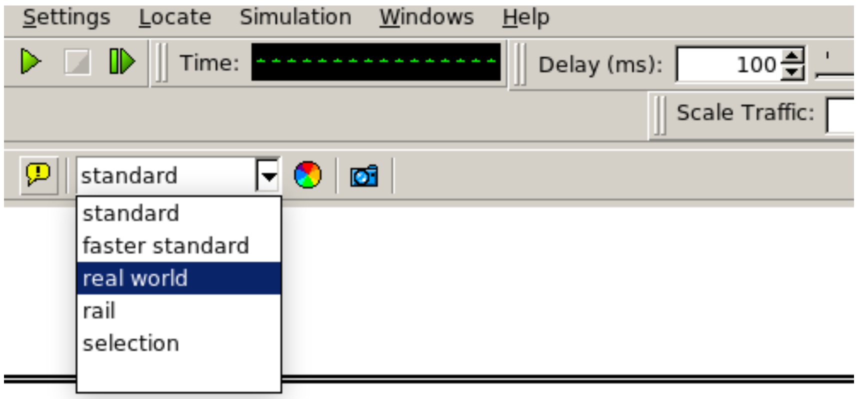
Once you have at least a map and a routes file, you can create a SUMO simulation file. We will name if `sim.sumocfg`. The simplest file looks like

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

Visualize your simulation by running it in `sumo-gui` tool

```
sumo-gui sim.sumocfg
```

Before you run it by clicking on the play green button ▶, let's first change the "Delay (ms)" from 0 to 100 so that simulation does not go faster than we can visualize it. Also, from the drop down menu that shows "standard" select "real world" to have vehicles drawn instead of triangles.



Delay (ms): 100

standard

real world

Once you click on the play button ▶, simulation will start, and you can see vehicles moving in your map. Zoom in to get a better view.

However, we would like to run the SUMO simulation, and store it in a trace file as follows:

```
sumo -c sim.sumocfg --fcd-output trace.xml
```

and then export the resulting sumoTrace.xml file to an ns-2 file as follows:

```
traceExporter.py --fcd-input sumoTrace.xml --ns2mobility-output ns2mobility.tcl
```

The resulting ns-2 mobility file is in a language called Tcl, which ns-2 uses. The ns-2 file looks like this, showing the first and last few lines:

```
$node_(0) set X_ 994.9
$node_(0) set Y_ 50.96
$node_(0) set Z_ 0
$ns_ at 1.0 "$node_(0) setdest 994.9 50.96 0.00"
$node_(1) set X_ -49.73
$node_(1) set Y_ 47.76
$node_(1) set Z_ 0
$ns_ at 1.0 "$node_(1) setdest -49.73 47.76 0.00"
$ns_ at 2.0 "$node_(0) setdest 992.56 50.96 2.34"
...
$ns_ at 2314.0 "$node_(1447) setdest 995.57 47.76 13.55"
$ns_ at 2314.0 "$node_(1448) setdest 972.99 47.76 12.24"
$ns_ at 2315.0 "$node_(1448) setdest 985.68 47.76 12.70"
$ns_ at 2316.0 "$node_(1448) setdest 998.19 47.76 12.50"
```

As you can see, the file generates 1449 nodes (from 0 to 1448) and simulation time runs for 2316 seconds.

5 Using SUMO-exported trace in ns-3

ns3 has a Mobiliy model that can be used with a C++ class called ns3::Ns2MobilityHelper that would use the ns2 mobility trace file generated earlier. In your ns-3 simulation, you have to create nodes and install mobility to them. You will have to make sure that the number of ns-3 simulation nodes that you created are equal to the number of simulation nodes created by SUMO in the ns-2 trace. You also want to make sure your ns-3 simulation lasts as long as the SUMO simulation as well.

If we know that our simulation will run for 2316 seconds using 1449 nodes, a minimalist example using C++ should look like the following

```
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);
  uint32_t node_count = 1449;
  double sim_time = 2316;

  NodeContainer nodes;
  nodes.Create(node_count);

  //The path to the file is relative to the root ns3 directory
  Ns2MobilityHelper ns2("path/to/ns2mobility.tcl");
  //this installs the mobility in all nodes in NodeList global container.
  ns2.Install();
  //proceed configuring devices, etc..
  ...
  Simulator::Stop ( Seconds (sim_time));
  Simulator::Run ();

}
```

If `node_count` is less than the nodes in the `SUMO` generated trace, the simulation will work, but only the first `node_count` nodes will have that mobility installed. If `node_count` is greater than `SUMO` nodes, the simulation will have a run-time error.

5.1 But, there is a problem!

In `SUMO`, vehicles enter into the simulation environment periodically over time, and leave at varying times as well when they reach their destinations. In this simplistic `ns-3` simulation, all nodes are entered into the simulation at time 0, and they will remain stationary at their starting position until they move (according to the `SUMO` generated trace). This is problematic when you vehicles are equipped with radios such as `WaveNetDevice`, that broadcast periodically, and will cause congestion in the network which is unrealistic.

To fix this, I created a `C++` class that reads the `ns-2` file to determine: the total number of nodes, the total simulation time, and the times each node enters and exits the simulation. I call the tool `Ns2NodeUtility` and an example is posted to my `github` repository here:

`https://github.com/addola/NS3-HelperScripts/tree/master/examples/SUMOTraceExample`

The code reads the `ns-2` mobility file, and uses regular expressions to determine that information (which takes long to run, by the way). The `ns-3` nodes are configured with applications (`CustomApplication` as a sub-class of `ns3::Application`) that when started, it would periodically broadcast data until the application stops. We use the data we obtained from the `Ns2NodeUtility` to determine the start and stop time of those applications.

6 A More complex SUMO scenario

This is a general example that you can use with any OpenStreetMap (`osm`) map. It creates random routes for vehicles of different classes: passenger cars, buses, trucks, delivery vehicles and trailers. For each class, a separate routes file is created. Since the `randomTrips.py` is used multiple times, we end up with files having vehicle ID values starting at 0 for each file, so we use `sed` to change vehicle IDs by adding a prefix to make them unique. We can parameterized the call to `randomTrips.py` to specify the interval between the times a vehicle enters the simulation (with `-p`) and also specify some attributes such as vehicle color, acceleration/deceleration, etc.

Use OpenStreetMap to export a map in `osm` format, and save it as `map.osm`. Then create this Makefile.

```
SIM_TIME=600

#The value after -p is the period at which vehicles are generated in seconds. Change this as you need
PASS_PARAMS=-e $(SIM_TIME) -p 1 --vehicle-class passenger --trip-attributes="color=\"255,255,255\""
BUS_PARAMS=-e $(SIM_TIME) -p 30 --vehicle-class bus --trip-attributes="accel=\"0.8\""
TRUCK_PARAMS=-e $(SIM_TIME) -p 15 --vehicle-class truck --trip-attributes="color=\"179,223,183\""
TRAILER_PARAMS=-e $(SIM_TIME) -p 150 --vehicle-class trailer --trip-attributes="color=\"223,179,180\"
↪  accel=\"0.5\""
DELIVERY_PARAMS=-e $(SIM_TIME) -p 30 --vehicle-class delivery --trip-attributes="color=\"115,211,230\""

all: osm trips

trace:
  sumo -c sim.sumocfg --fcd-output sumoTrace.xml
  traceExporter.py --fcd-input sumoTrace.xml --ns2mobility-output ns2mobility.tcl

trips:
  # Generate random trips for different vehicle types
  randomTrips.py -n map.net.xml -r bus_routes.rou.xml  -o bus_trips.xml $(BUS_PARAMS)
  randomTrips.py -n map.net.xml -r truck_routes.rou.xml  -o truck_trips.xml $(TRUCK_PARAMS)
  randomTrips.py -n map.net.xml -r delivery_routes.rou.xml  -o delivery_trips.xml $(DELIVERY_PARAMS)
  randomTrips.py -n map.net.xml -r passenger_routes.rou.xml  -o passenger_trips.xml $(PASS_PARAMS)
  randomTrips.py -n map.net.xml -r trailer_routes.rou.xml  -o trailer_trips.xml $(TRAILER_PARAMS)

  #Vehicles should have unique IDs, so I'll add a prefix to the ID according to type.
  sed -i "s/vehicle id=\"/vehicle id=\"bus/g" bus_routes.rou.xml
  sed -i "s/vehicle id=\"/vehicle id=\"truck/g" truck_routes.rou.xml
  sed -i "s/vehicle id=\"/vehicle id=\"pass/g" passenger_routes.rou.xml
  sed -i "s/vehicle id=\"/vehicle id=\"deliv/g" delivery_routes.rou.xml
  sed -i "s/vehicle id=\"/vehicle id=\"trailer/g" trailer_routes.rou.xml

osm:
  netconvert  --osm-files map.osm -o map.net.xml
  #To create a nicer looking map, we'll also obtain buildings information
  polyconvert --net-file map.net.xml --osm-files map.osm -o map.poly.xml

sim:
  sumo-gui sim.sumocfg

clean:
  rm -f sumoTrace.xml ns2mobility.tcl
  rm -f *.rou.xml *.rou.alt.xml *trips.xml
  rm -f map.net.xml map.poly.xml
```

Create a `SUMO` configuration file as follows:

```
<configuration>
  <input>
    <net-file value="map.net.xml"/>
    <route-files value="truck_routes.rou.xml, passenger_routes.rou.xml, bus_routes.rou.xml,
    ↪  trailer_routes.rou.xml, delivery_routes.rou.xml"/>
    <additional-files value="map.poly.xml"/>
  </input>
</configuration>
```

Run it as follows to start a simulation in `sumo-gui`

```
make osm
make trips
make sim
```

If you want to generate an `ns-2` mobility trace, then run the rule `trace`

```
make trace
```