EINSTEIN CENTER Digital Future

Technische Universität Berlin

# Tutorial 6: Traffic Light Control Using TraCI

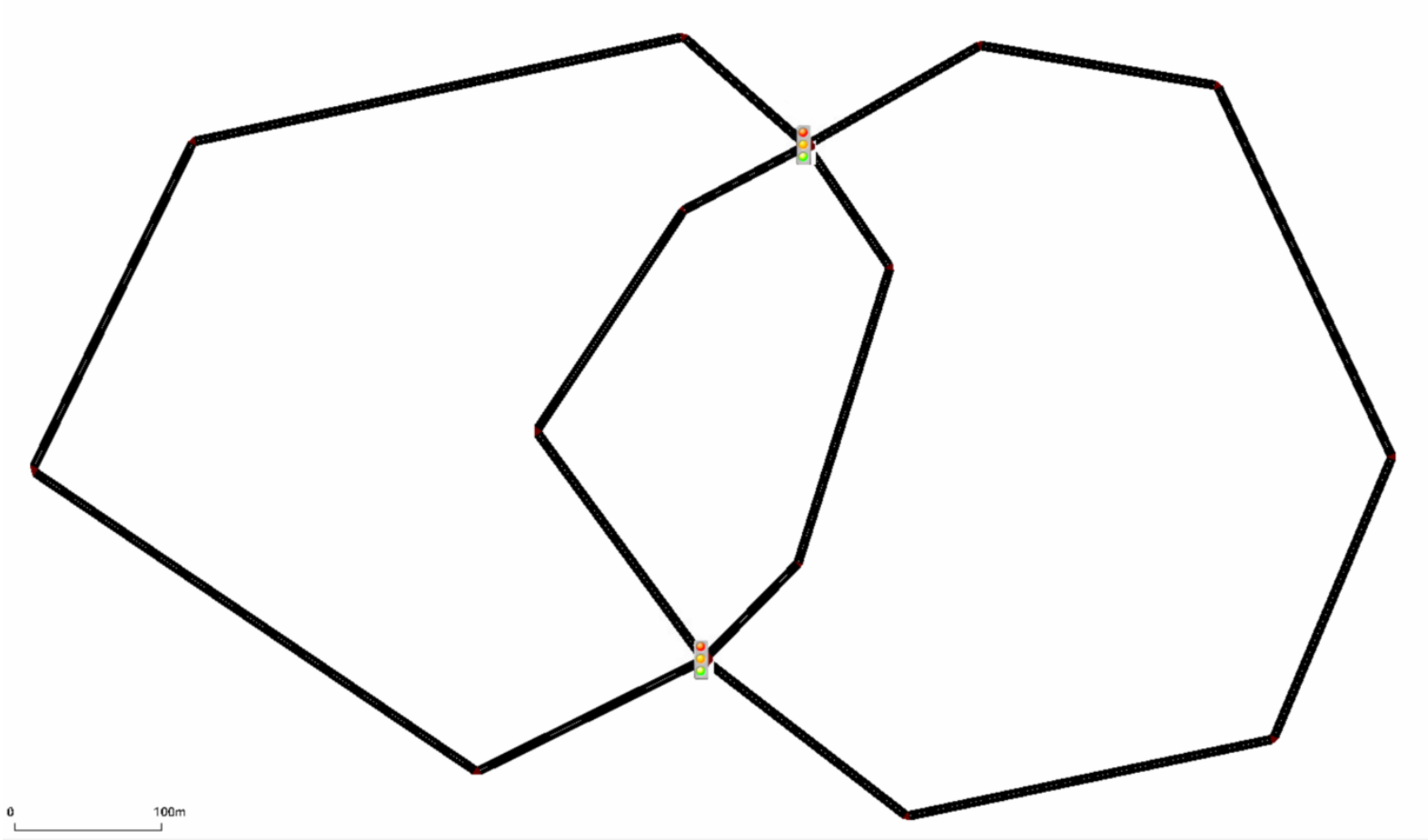
## Traffic Light Control Using TraCI

Prof. Sangyoung Park

Module ”Vehicle-2-X: Communication and Control“

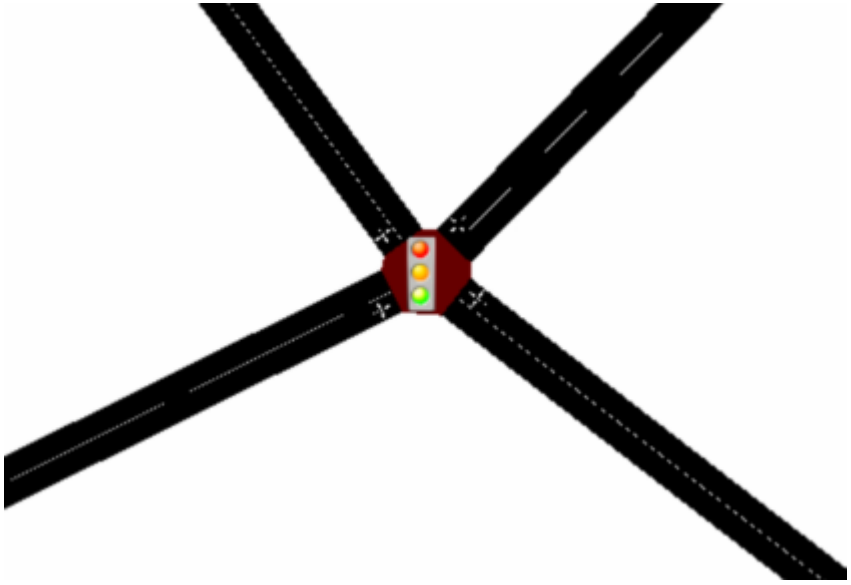
# Let’s Add Traffic Lights in a Road Network

- Let’s make two circular roads with two intersections
- And traffic lights will be automatically generated at the intersections



# Let's Add Traffic Lights in a Road Network

- But be careful, you shouldn't just cross two sections of road using edge
- It will look like an intersection, but it's not
- You connect the edges to the intersection explicitly, then it will look like the figure below



# Let’s make traffic flows

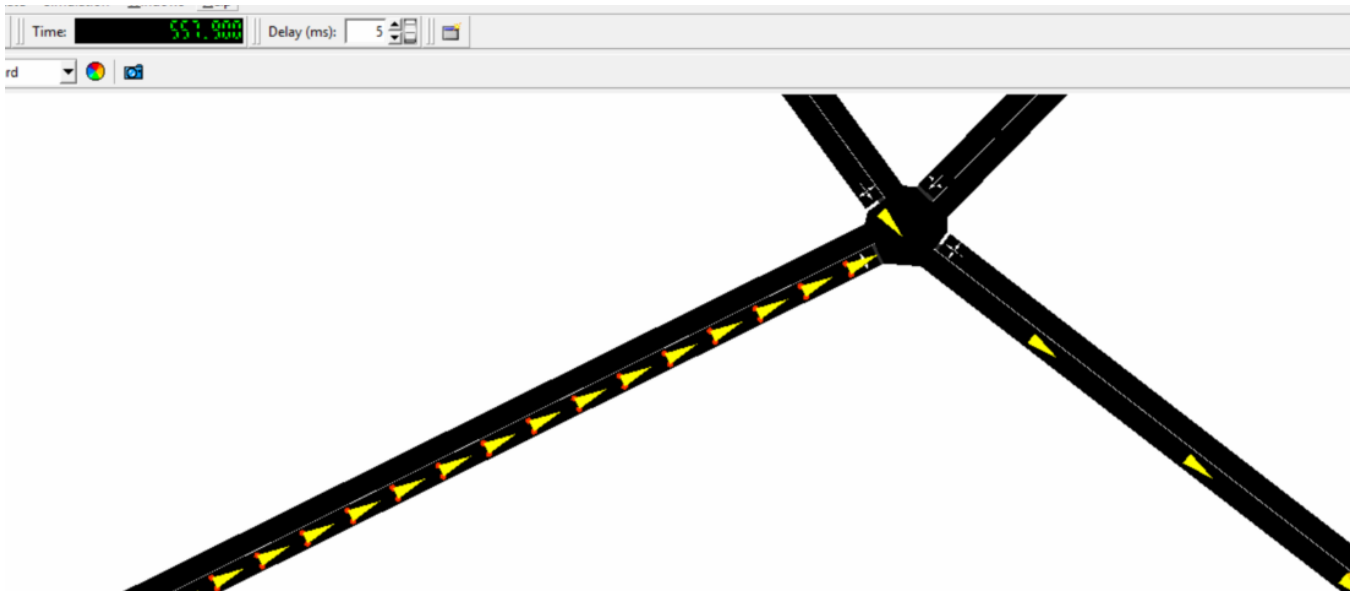
- We would make two traffic flows each going in respective circles using reroute
- Not all details are given, please recall the past tutorials

```
<routes>
  <vType id="car" type="passenger" length="5" accel="3.5" decel="2.2" emergencyDecel="5" sigma="0" maxSpeed="28"/>
  <flow id="carflow1" type="car" beg="0" end="0" number="1000" from="edge1" to="edge2"/>
  <flow id="carflow2" type="car" beg="0" end="0" number="1000" from="edge3" to="edge4"/>
</routes>
```

```
<additionals>
   <rerouter id="rerouter_0" edges="edge1">
                        <interval end="1e9">
                          <destProbReroute id="edge2"/>
                        </interval>
            </rerouter>
   <rerouter id="rerouter_1" edges="edge2">
                        <interval end="1e9">
                          <destProbReroute id="edge1"/>
                        </interval>
            </rerouter>
            <rerouter id="rerouter_2" edges="edge3">
                        <interval end="1e9">
                          <destProbReroute id="edge4"/>
                        </interval>
            </rerouter>
   <rerouter id="rerouter_3" edges="edge4">
                        <interval end="1e9">
                          <destProbReroute id="edge3"/>
                        </interval>
            </rerouter>
</additionals>
```

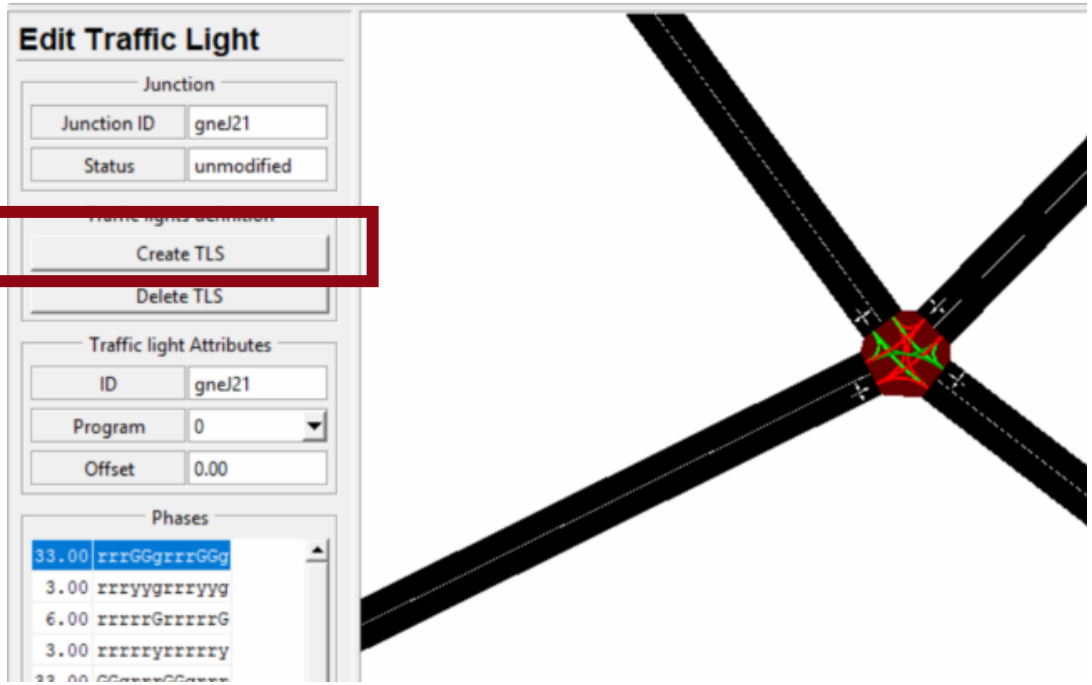
# What happens if we run simulation?

- Modify your sumocfg file accordingly, and then run your simulation
- There is traffic light, but no signal control program
- So…. Traffic accumulates in one direction



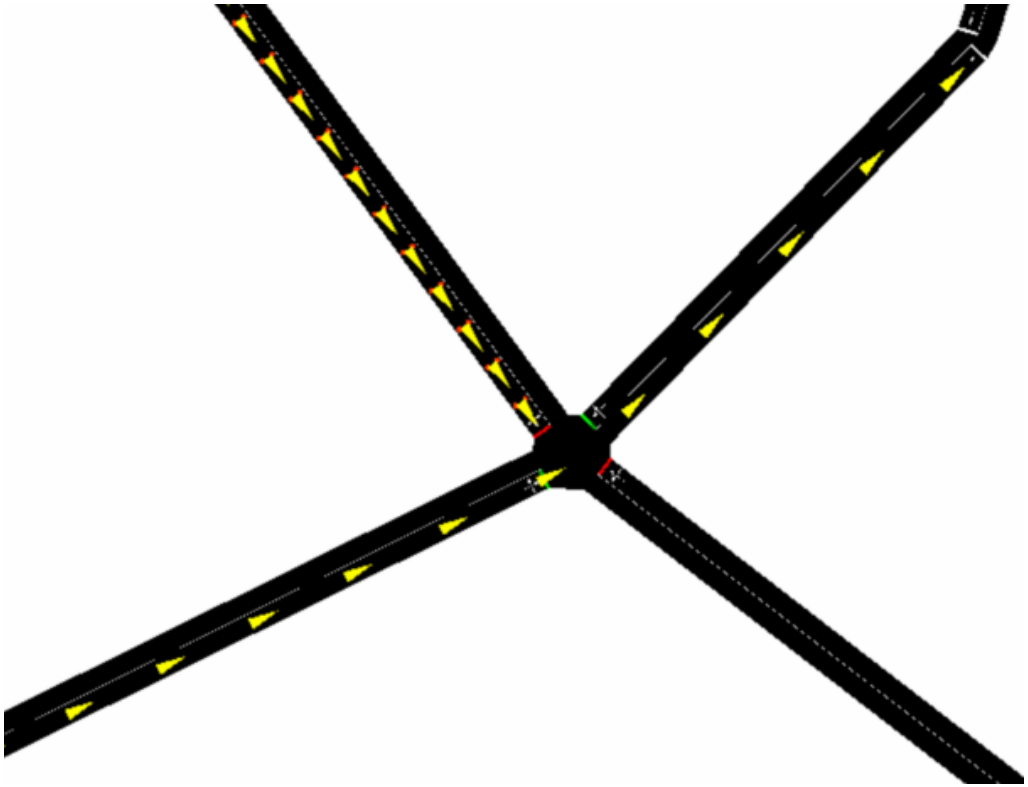
# Generate Traffic Light Control

- Edit -> Traffic Light -> Click on junction
- Click create TLS and you will see default program generated
- What does “rrrGGgrrrGGg” mean?
  - When you click on the phases, you signals will be highlighted on the junctions



# Let‘s Run Simulation Again

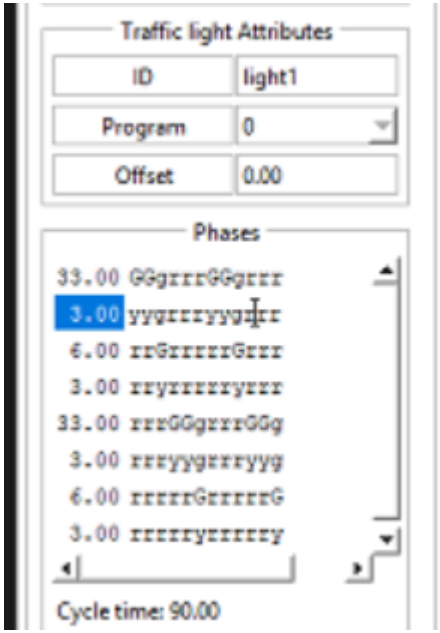
- Now you will see green and red lights distributed evenly across the two roads



# Take a look at net.xml file

- You will find a section in .net.xml file with the following text
- You can see that this corresponds to the information on netedit GUI
- You can of course modify the text to change the traffic program if you want to (or you could use GUI in netedit as well)

```
<tlLogic id="light1" type="static" programID="0" offset="0">
    <phase duration="33" state="GGgrrrGGgrrr"/>
    <phase duration="3" state="yygrrryygrrr"/>
    <phase duration="6" state="rrGrrrrrGrrr"/>
    <phase duration="3" state="rryrrrrryrrr"/>
    <phase duration="33" state="rrrGGgrrrGGg"/>
    <phase duration="3" state="rrryygrrryyg"/>
    <phase duration="6" state="rrrrrGrrrrrG"/>
    <phase duration="3" state="rrrrryrrrrry"/>
  </tlLogic>
  <tlLogic id="light2" type="static" programID="0" offset="0">
    <phase duration="33" state="rrrGGgrrrGGg"/>
    <phase duration="3" state="rrryygrrryyg"/>
    <phase duration="6" state="rrrrrGrrrrrG"/>
    <phase duration="3" state="rrrrryrrrrry"/>
    <phase duration="33" state="GGgrrrGGgrrr"/>
    <phase duration="3" state="yygrrryygrrr"/>
    <phase duration="6" state="rrGrrrrrGrrr"/>
    <phase duration="3" state="rryrrrrryrrr"/>
  </tlLogic>
```



# Let‘s modify the .rou.xml file

- Let‘s redue the number of cars on the carflow2 to be 1
- Circular road on the left has 10 cars circulating
- Circular road on the right has only one car re-routed
- Let‘s control the traffic lights such that carflow2 is not interrupted!
  - When carflow2 is near a traffic light, carflow2 is given a green light
  - Otherwise, carflow1 is always given the green light

```
<routes>
  <vType id="car" type="passenger" length="5" accel="3.5" decel="2.2" emergencyDecel="5" sigma="0" maxSpeed="28"/>
  <flow id="carflow1" type="car" beg="0" end="0" number="10" from="edge1" to="edge2"/>
  <flow id="carflow2" type="car" beg="0" end="0" number="1" from="edge3" to="edge4"/>
</routes>
```

# Let‘s control the traffic light using TraCI

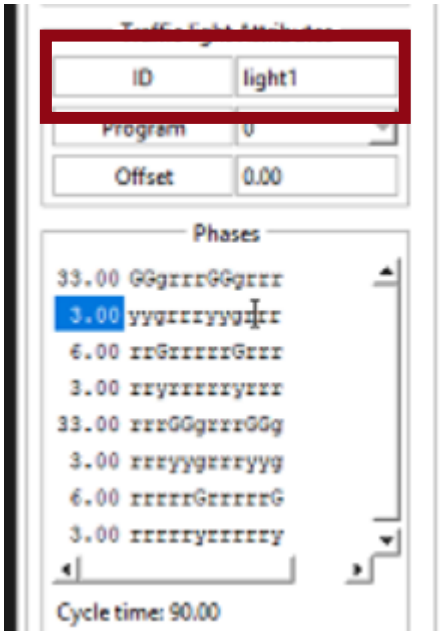
- Let’s create another cc file for traffic light RSU
- New -> Create OMNet++ class -> TrafficLightRsuApp.cc & .h
- Let it inherit BaseWaveApplLayer again
- But this time, be aware of the content in the red box

```
#include "veins/modules/application/ieee80211p/BaseWaveApplLayer.h"
#include "veins/modules/mobility/traci/TraCIScenarioManager.h"
#include "veins/modules/mobility/traci/TraCICommandInterface.h"

namespace Veins{
class TrafficLightRsuApp  : public BaseWaveApplLayer {
    protected:
        virtual void initialize(int stage);
        virtual void onWSM(WaveShortMessage* wsm);
        virtual void onWSA(WaveServiceAdvertisment* wsa);
        virtual void onBSM(BasicSafetyMessage * bsm);
        virtual void handleSelfMsg(cMessage* msg);

        TraCIScenarioManager* manager;
        std::string trafficLightId;

        cMessage* initMsg;
        cMessage* phaseMsg;
};
}
```

# What is TraCIScenarioManager?

- We need this to get access to TraCI from our RSU
- Basically, the following code gives you access to a particular traffic light
- You need to have a traffic light called „light1“ in your .net.xml file

```
manager = TraCIScenarioManagerAccess().get();
traci = manager->getCommandInterface();
trafficLightId = "light1";
traci->trafficlight(trafficLightId).setProgram("program2");
```




- Then, where do we define „program2“?
- We will come to that soon

# Where should we insert the code?

- At first, I tried TrafficLightRsuApp::initialize() just like for vehicles, a similar code exists inside BaseWaveApplLayer::initialize()
- But for some reason, traci connection with SUMO is not established yet when initialize() is called

```
manager = TraCIScenarioManagerAccess().get();
traci = manager->getCommandInterface();
```

- So, I had to call it after the simulation has alread run for some time
- How do we do it? We use scheduleAt() function
  - Number 77 is randomly chosen
  - You can choose any other number
  - Number 88 is also random

```
void TrafficLightRsuApp::initialize(int stage){
    BaseWaveApplLayer::initialize(stage);
    if (stage == 0) {

    }
    else if (stage == 1){
        initMsg = new cMessage("traffic light init",77);
        phaseMsg = new cMessage("phase msg",88);
        scheduleAt(0.1, initMsg);
    }
}
```

# Getting TraCI in the RSU

- I know.., this is a bit tricky to understand.. So, this time, I uploaded my source files where you can take a look
  - Please find TrafficLightRsuApp.cc & .h files on ISIS to take a look
- We‘ve just schedule something at simulation time 0.1 seconds
- At 0.1 second handleSelfMsg() will be called (do you remember OMNet++ example?)
- See the next page for source code
  - Do you see the number 77?
- Here, we are now able to get access to traCI as we have already established connection with SUMO
  - (This took me a lot of time to figure out, sorry for the delay...)

# Getting TraCI in the RSU

```
void TrafficLightRsuApp::handleSelfMsg(cMessage* msg){
    BaseWaveApplLayer::handleSelfMsg(msg);
    switch (msg->getKind())
    {
        case 77:
            manager = TraCIScenarioManagerAccess().get();
            traci = manager->getCommandInterface();
            switch (myId)
            {
            case 7: // first traffic light
                trafficLightId = "light2";
                traci->trafficlight(trafficLightId).setProgram("program2");
                break;
            case 8: // second traffic light
                trafficLightId = "light1";
                traci->trafficlight(trafficLightId).setProgram("program2");
                break;
            default:
                assert(0); // something wrong, it's not a traffic light, crash the program
                break;
            }
            break;
        case 88:
            traci->trafficlight(trafficLightId).setProgram("program2");
            break;
        default:
            assert(0);
            break;
    }

}
```

# myId of RSUs

- There are two intersections and traffic lights, so let‘s have two RSUs
- Just like I figured out myIds for vehicles, I figured out myId of RSUs using the debugger
  - It‘s 7 and 8 for the first two RSUs
- Now, we associate the RSUs with the traffic lights in the .net.xml file
  - It‘s nothing fancy, we just store the names of the traffic lights that we defined in the .net.xml file
  - For RSU of myId 7, we associate with traffic light2
  - For RSU of myId 8, we associate with traffic light1

```
case 7: // first traffic light
        trafficLightId = "light2";
        traci->trafficlight(trafficLightId).setProgram("program2");
        break;
case 8: // second traffic light
        trafficLightId = "light1";
        traci->trafficlight(trafficLightId).setProgram("program2");
        break;
```

# Where are the traffic programs?

- We can define it in a separate file
  - We can also define it in the .net.xml file as well (we‘ve already seen one generated by netedit on slide 8)
- Make a file called tls_program.tls.xml with the following contents
- You see that there are traffic programs for each traffic lights

```
<tls>
        <tlLogic id="light1" type="static" programID="program1" offset="0">
                <phase duration="999" state="GGgrrrGGgrrr"/>
                <phase duration="999" state="GGgrrrGGgrrr"/>
        </tlLogic>
        <tlLogic id="light1" type="static" programID="program2" offset="0">
                <phase duration="999" state="rrrGGgrrrGGg"/>
                <phase duration="999" state="rrrGGgrrrGGg"/>
        </tlLogic>

        <tlLogic id="light2" type="static" programID="program1" offset="0">
                <phase duration="999" state="rrrGGgrrrGGg"/>
                <phase duration="999" state="rrrGGgrrrGGg"/>
        </tlLogic>
        <tlLogic id="light2" type="static" programID="program2" offset="0">
                <phase duration="999" state="GGgrrrGGgrrr"/>
                <phase duration="999" state="GGgrrrGGgrrr"/>
        </tlLogic>

</tls>
```

# We need to let SUMO know that a new file exists

- In .launchd.xml file you add the file

```
<?xml version="1.0"?>
<!-- debug config -->
<launch>
        <copy file="traffic_lights.net.xml" />
        <copy file="traffic_lights.rou.xml" />
        <copy file="traffic_lights.add.xml" />
        <copy file="tls_program.tls.xml" />
        <copy file="traffic_lights.sumocfg" type="config" />
</launch>
```

- In .sumocfg file

```
<input>
    <net-file value="traffic_lights.net.xml"/>
    <route-files value="traffic_lights.rou.xml"/>
    <additional-files value="traffic_lights.add.xml tls_program.tls.xml"/>
  </input>
```

# Where are the traffic programs?

- The traffic program looks difficult, but it‘s essentially two programs for two traffic lights where you allow green lights for one street while giving red light for the other
- I configured the programs in the way that „program2“ will give green light to the traffic which goes around the left circle
- So the source code on page 15 shows that left circle will have default green light for two traffic lights
- You will be able to check it graphically later

# Now, we want our application to change the signals

- Let‘s make a traffic signal control which gives green light to the right circles only when the vehicle (single vehicle we configured on .rou.xml file) is near the traffic light
- I‘ve already found out that the vehicle on the right has the myId of 25
- So, whenever the RSU receives a BSM from car 25, it checks for the distance whether it‘s closer than 20 meters, and changes the traffic light to program1
- After 5 seconds, we want to switch back to program2 because 5 second sis enough for car 25 to pass through the intersection

```
void TrafficLightRsuApp::onBSM(BasicSafetyMessage * bsm){
    if (bsm->getSenderAddress() == 25) {
        if ((curPosition-bsm->getSenderPos()).length() < 20) {
            if (!phaseMsg->isScheduled()) {
                traci->trafficlight(trafficLightId).setProgram("program1");
                scheduleAt(simTime()+5,phaseMsg);
            }
        }
    }
}
```

# Now, we want our application to change the signals

- So, we schedule a phaseMsg after 5 seconds (scheduleAt() function call)
- However, BSM is sent every 0.1 seconds, we want to change traffic program only once when the vehicle approaches
- So, we will check whether phaseMsg is already scheduled first and then execute the code
- When the vehicle is within 20 meters of the traffic signal for the first time, the code enters inside the if clause
- Every 0.1 second after that, phaseMsg is already scheduled so we don‘t enter the if clause

```
void TrafficLightRsuApp::onBSM(BasicSafetyMessage * bsm){
    if (bsm->getSenderAddress() == 25) {
        if ((curPosition-bsm->getSenderPos()).length() < 20) {
            if (!phaseMsg->isScheduled()) {
                traci->trafficlight(trafficLightId).setProgram("program1");
                scheduleAt(simTime()+5,phaseMsg);
            }
        }
    }
}
```

# What happens after 5 seconds?

```
void TrafficLightRsuApp::handleSelfMsg(cMessage* msg){
    BaseWaveApplLayer::handleSelfMsg(msg);
    switch (msg->getKind())
    {
        case 77:
            manager = TraCIScenarioManagerAccess().get();
            traci = manager->getCommandInterface();
            switch (myId)
            {
            case 7: // first traffic light
                trafficLightId = "light2";
                traci->trafficlight(trafficLightId).setProgram("program2");
                break;
            case 8: // second traffic light
                trafficLightId = "light1";
                traci->trafficlight(trafficLightId).setProgram("program2");
                break;
            default:
                assert(0); // something wrong, it's not a traffic light, crash the program
                break;
            }
            break;
        case 88:
            traci->trafficlight(trafficLightId).setProgram("program2");
            break;
        default:
            assert(0);
            break;
    }

}
```

**Back to handleSelfMsg()**
**Remember number 88 from page 12?**

# Now, we need to let RSU to use our application

- In the myTestNetwork.ned file
- We are now adding TWO RSUs!! (See the red rectangle)

```
package newTest;
import org.car2x.veins.nodes.RSU;
import org.car2x.veins.nodes.Scenario;

network myTestNetwork extends Scenario
{
    submodules:
        rsu[2]: RSU {
            @display("p=50,50;i=veins/sign/yellowdiamond;is=vs");
        }

}
```

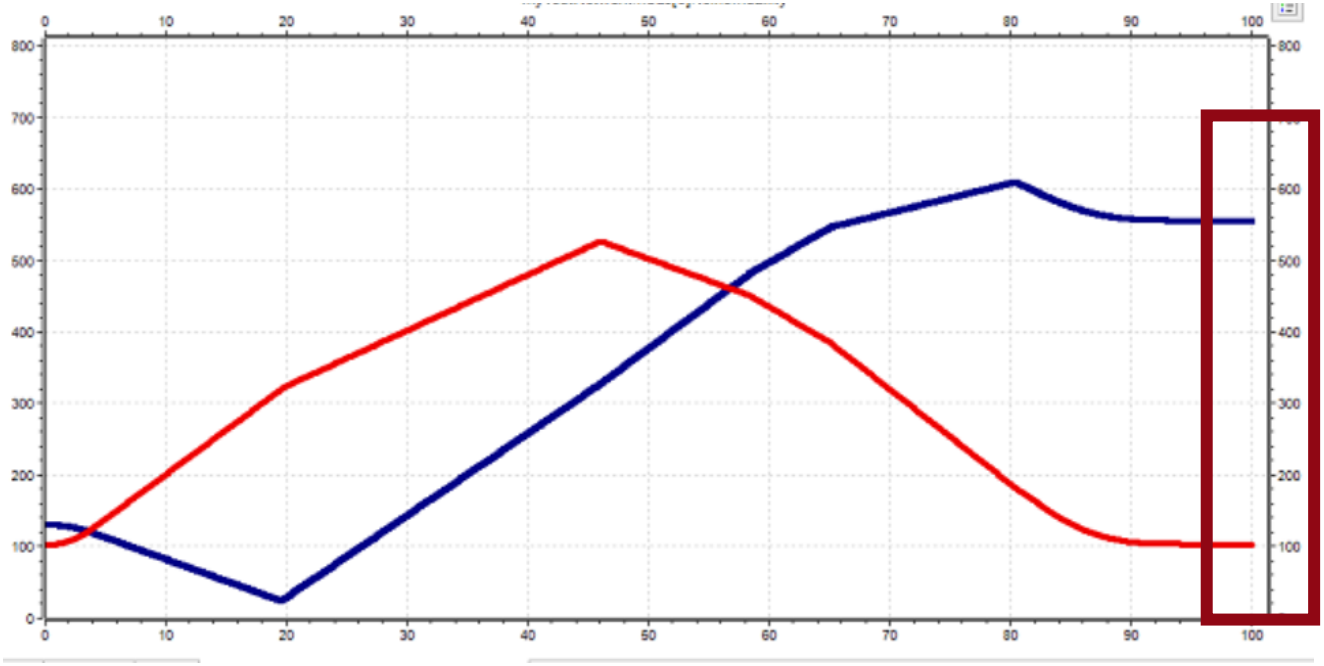
# Now, we need to let RSU to use our application

- And, in the .ini file, we need to designate the location of the RSUs
- I‘ve added text in the rectangles
- We designate our just created app
- We also designate the coordinates of our RSUs

```
##########################################################
#                        RSU SETTINGS                    #
#                                                        #
#                                                        #
##########################################################
*.rsu[0].mobility.x = 485
*.rsu[0].mobility.y = 445
*.rsu[0].mobility.z = 3

*.rsu[1].mobility.x = 556
*.rsu[1].mobility.y = 103
*.rsu[1].mobility.z = 3

*.rsu[*].applType = "TrafficLightRsuApp"
```

# How do we know the coordinates of the RSUs?

- We can‘t automatically detect the coordinates of the RSUs
- Coordinate systems for SUMO and Veins are unfortunatey not aligned
- We need to manually find out
- How I did it is that I let the vehicles stop at the traffic light and read the coordinates in the .anf file




**Graph of posx and posy**

**Vehicle stopped. This is (556, 103) The value I used in .ini file**

# Don‘t forget the .ned file!

- TrafficLightRsuApp.ned
- There are occasions where the simulator doesn‘t find the class because the different namespaces
- If you encounter such errors, try adding veins:: in front of the class names

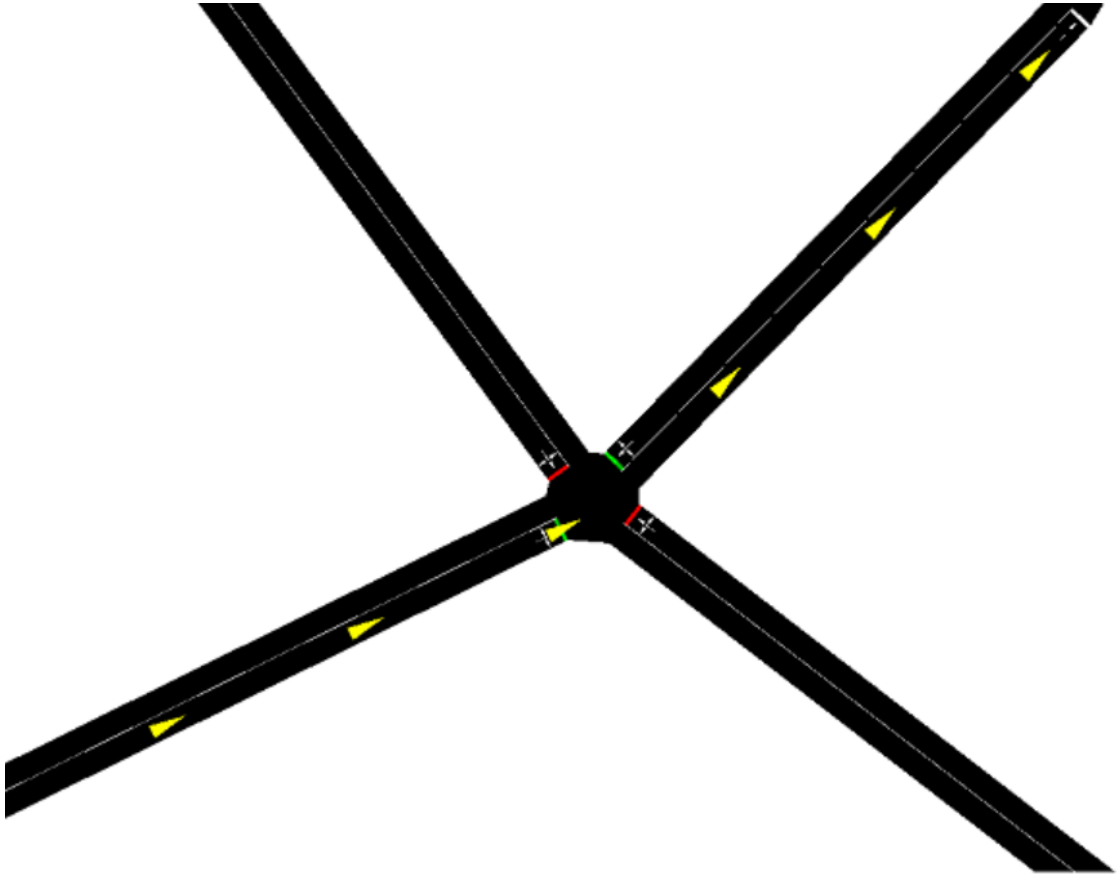
```
simple TrafficLightRsuApp extends BaseWaveApplLayer
{
    @class(Veins::TrafficLightRsuApp);
    string appName = default("My first Veins App!");
}
```

# Let‘s run the simulation!

- Traffic lights are not well visualized in Veins GUI
- So, let‘s run sumo-gui this time to see the traffic lights
- In the Msys terminal, we use sumo-gui.exe instead of sumo.exe
- sumo-launchd.py -vv -c {YOURPATH}/sumo-gui.exe
- You run simulation with „Express speed“
- And SUMO-GUI  will be launched
  - You need to click „start“ in SUMO-GUI as well

# Traffic lights are controlled as we want!

- You will see the traffic lights which are red, will turn green only when a car passes by from the right circle!



# Questions?

- If you have a lot of questions, drop by my office (H4133) or wait for tutorial time
- Send me email if you have short simple questions