Tutorial 4: Simple WAVE Application

Simple WAVE Application

Prof. Dr. Sangyoung Park
Module ”Vehicle-2-X: Communication and Control“

Objectives

- Use your own road network for Veins simulation
- Understanding ScenarioManager in Veins
- Defining custom WAVE message
- WAVE communication example
- Vehicle motion data exchange

Make a New Project

- Make another folder in the Veins project just like you did in the last tutorial
- Copy the following files into the new project
  - Veins/examples/veins/antenna.xml, config.xml, omnetpp.ini
  - Copy them into your project folder
  - Select your newly created folder as a NED source folder

New NED File

- We need a network to simulate
- Right click on your project and make a new NED file
  - Let‘s name it CommExample.ned
  - Enter the following code

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


network CommExample extends Scenario
{
    submodules:
        rsu[1]: RSU {
            @display("p=150,140;i=veins/sign/yellowdiamond;is=vs");
        }
}
```

Road Network

- Let‘s make a 1 km stretch of road
- Open „netedit“ and make a simple road network comprising three nodes
    - Node 1: (0,0)
    - Node 2: (1000,0)
    - Node 3: (1010,0)
- Name the edge of the left as “edge1“, and on the right as „edge2“
- And save the network as “straight.net.xml“ in your project folder

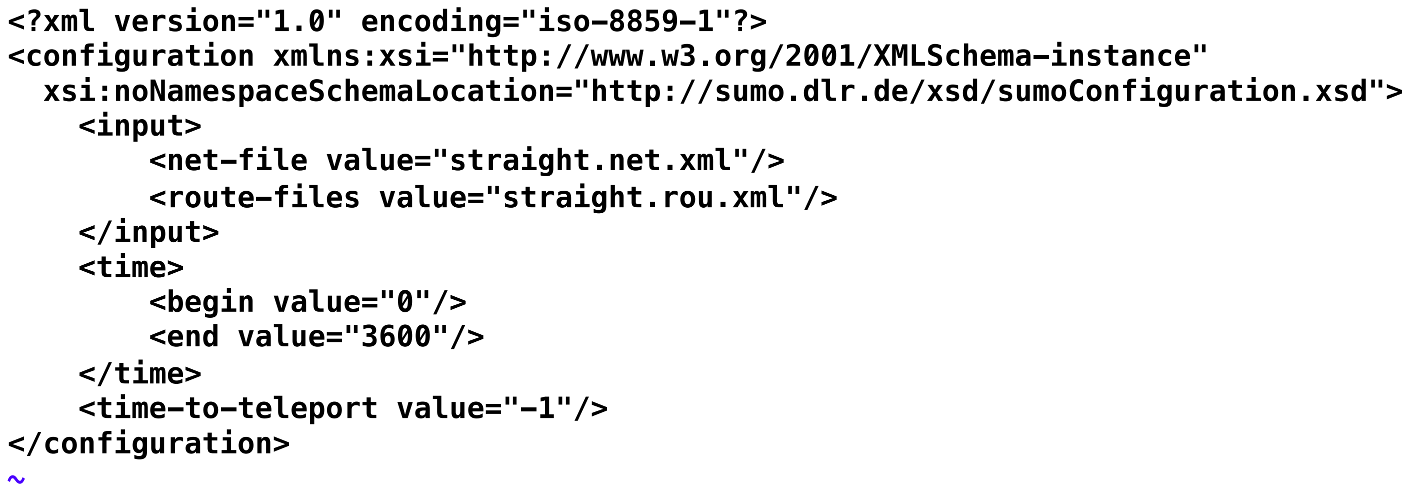
Define Traffic Flow

- Open any text editor of your choice
- Type the following to create straight.rou.xml
- We are sending one vehicle from edge1 to edge2

```
<?xml version="1.0" encoding="UTF-8"?>
<routes>
  <vType accel="3.0" decel="6.0" id="CarA" length="5.0" minGap="2.5" maxSpeed="50.0" sigma="0.5" />
  <route id="route01" edges="edge1 edge2"/>
  <vehicle depart="0" id="veh0" route="route01" type="CarA" color="1,0,0" />
</routes>
~
```

Make .sumocfg File

- We are now letting SUMO know which files we want to use for SUMO simulation
- Make straight.sumocfg file with the following contents



```
<?xml version="1.0" encoding="iso-8859-1"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">
    <input>
        <net-file value="straight.net.xml"/>
        <route-files value="straight.rou.xml"/>
    </input>
    <time>
        <begin value="0"/>
        <end value="3600"/>
    </time>
    <time-to-teleport value="-1"/>
</configuration>
~
```

Check SUMO Simulation

- Again, we make a „straight.launchd.xml“ file
- Veins knows which files we need by reading this file.

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

Modify the .ini File

- If you give the .ini file “straight.launchd.xml“ then it knows all the necessary files for Veins simulation

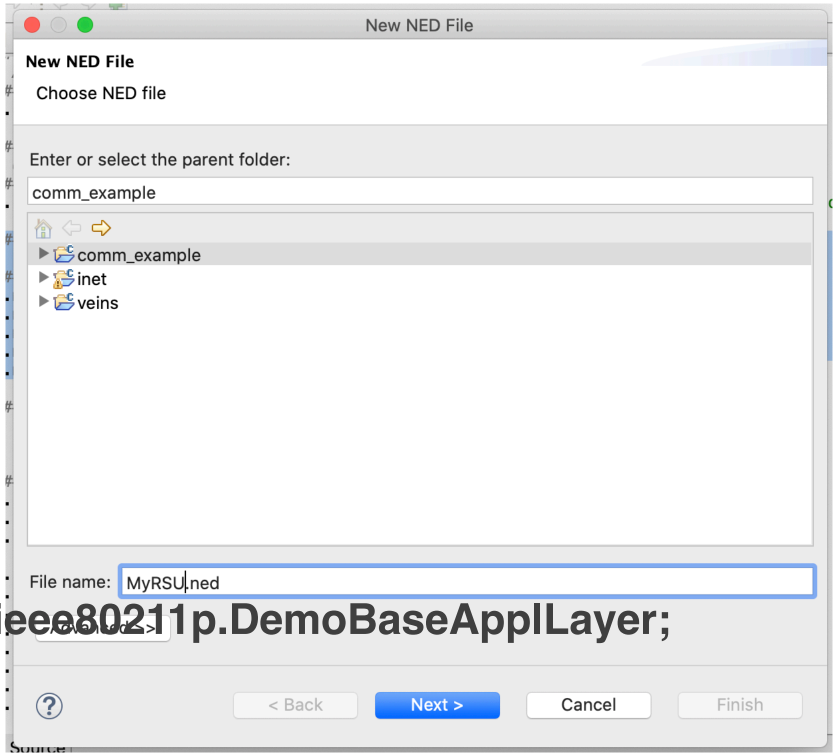
```
##########################################################
#            TraCIScenarioManager parameters              #
##########################################################
*.manager.updateInterval = 1s
*.manager.host = "localhost"
*.manager.port = 9999
*.manager.autoShutdown = true
*.manager.launchConfig = xmldoc("straight.launchd.xml")
```

Application for RSU-Vehicle

- Let‘s make a simple RSU application that would send WSA
- Vehicles passing by subscribe to the service provided by the RSU
- Vehicles then inform the RSU of their current locations using WSM
- Upon receiving WSM from vehicles, RSU calculates the distance to the vehicle and informs the vehicles that how many vehicles are within 200 m of the RSU

RSU Behavior Description

- You create a new NED file
- Let‘s name it
- MyRSU.ned
- RSU will run a (WAVE) „application“ which extends DemoBaseApplLayer



```
import org.car2x.veins.modules.application.ieee80211p.DemoBaseApplLayer;

simple MyRSU extends DemoBaseApplLayer
{
    @class(veins::MyRSU);
    @display("i=block/app2");
}
```

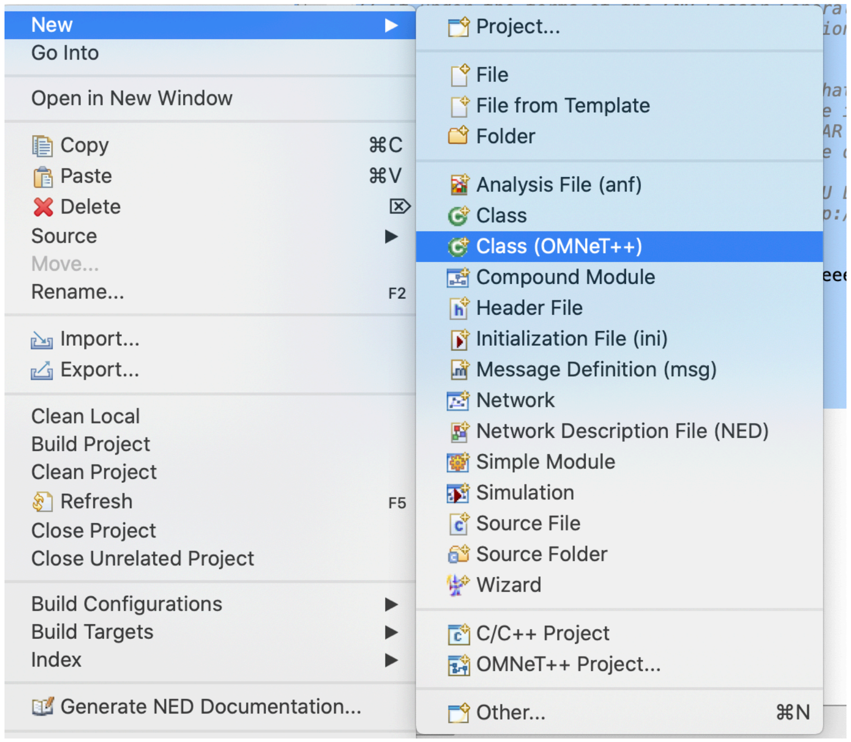
Vehicle Behavior Description

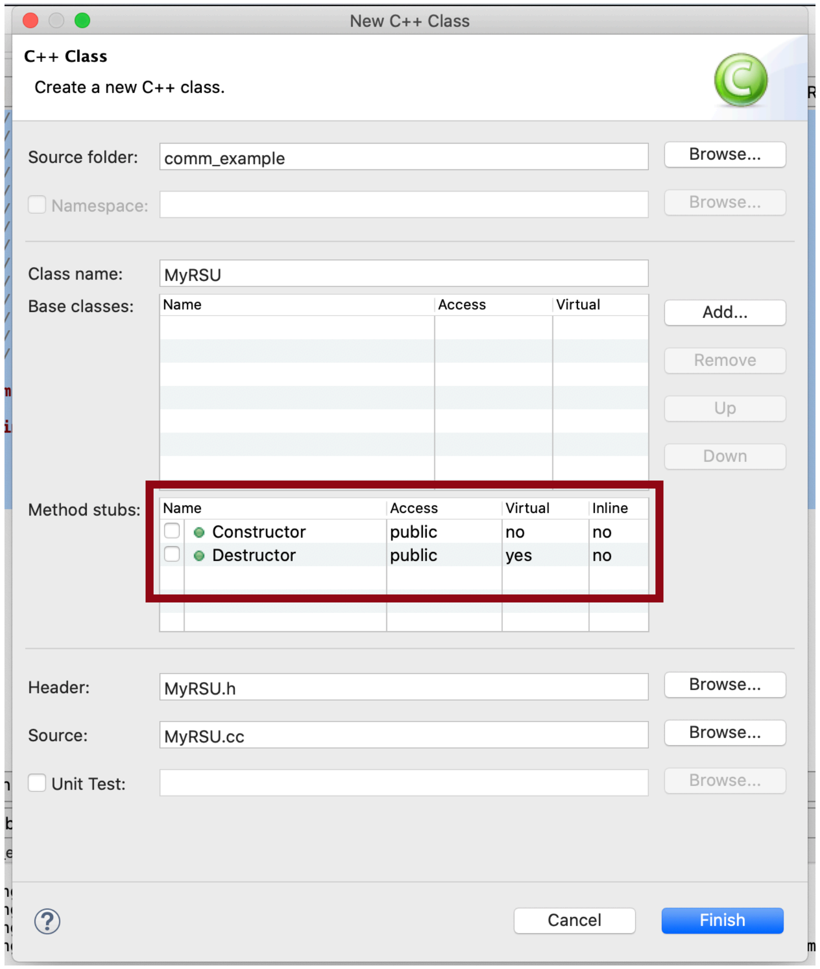
- We create NED also for vehicles
- Note @class(veins::MyVehicles)
  - This line designates the C++ class which will describe the vehicle behaviors at the application level (same goes for the RSU NED file we created from the previous slide)

```
//
import org.car2x.veins.modules.application.ieee80211p.DemoBaseApplLayer;


simple MyVehicle extends DemoBaseApplLayer
{
    @class(veins::MyVehicle);
    @display("i=block/app2");
}
```

C++ Files for RSUs

- We create a OMNeT++ class
- Let‘s name it MyRSU
- We don‘t need constructors and destructors so un-check (red box)




New C++ Class

C++ Class

Create a new C++ class.

Source folder: comm_example

Namespace:

Class name: MyRSU

Base classes:

Name	Access	Virtual

Method stubs:

	Name	Access	Virtual	Inline
☐	Constructor	public	no	no
☐	Destructor	public	yes	no

Header: MyRSU.h

Source: MyRSU.cc

☐ Unit Test:

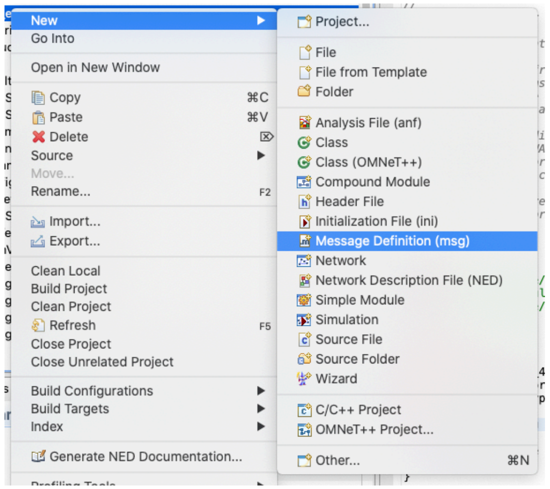
MyRSU.h

- For now, let‘s define two functions
- Initialize()
  - RSU will start service (WSA)
- onWSM()
  - RSU will collect info from vehicles

```
#pragma once

#include "veins/modules/application/ieee80211p/DemoBaseApplLayer.h"

namespace veins {

/**
 * Small RSU Demo using 11p
 */
class VEINS_API MyRSU : public DemoBaseApplLayer {
public:
    void initialize(int stage) override;

protected:
    void onWSM(BaseFrame1609_4* wsm) override;
};

} // namespace veins
```

Defining Message Format

- It is useful to define a message format
- C++ class can be automatically generated by OMNeT++ (remember TicToc tutorial?)
- We only have to write .msg file
- Make „NumVehicleMsg.msg“
- timeStampP is not typo
  - timeStamp is a reserved word



```
cplusplus {{
#include "veins/base/utils/Coord.h"
#include "veins/modules/messages/BaseFrame1609_4_m.h"
#include "veins/base/utils/SimpleAddress.h"
}}


namespace veins;


class BaseFrame1609_4;
class noncobject Coord;
class LAddress::L2Type extends void;


packet NumVehicleMsg extends BaseFrame1609_4 {
    Coord senderPos;
    int numVehicles;
    simtime_t timeStampP;
    LAddress::L2Type senderAddress = -1;
}
```

Defining Message Format

- The format contains
  - senderPos of Coord type
  - numVehicles of int type
  - senderAddress of LAddress::L2Type
- In order to know what Coord type is, you can look up „Coord.h“
  - Right click -> Show Decl.
  - It‘s basically a 3D vector
- When you compile the project next time, two files named „NumVehicleMsg_m.cc“ and „NumVehicleMsg_m.h“ will be generated

```
cplusplus {{
#include "veins/base/utils/Coord.h"
#include "veins/modules/messages/BaseFrame1609_4_m.h"
#include "veins/base/utils/SimpleAddress.h"
}}


namespace veins;


class BaseFrame1609_4;
class noncobject Coord;
class LAddress::L2Type extends void;


packet NumVehicleMsg extends BaseFrame1609_4 {
   Coord senderPos;
   int numVehicles;
   simtime_t timeStampP;
   LAddress::L2Type senderAddress = -1;
}
```

MyRSU.cc

- Let‘s write the description of RSU behavior
- initialize() is called when the RSU initializes
- onWSM() is called when the RSU receives a WSM
- We‘re printing the distance to the sender vehicle

```
#include "MyRSU.h"
#include "NumVehicleMsg_m.h"

using namespace veins;

Define_Module(veins::MyRSU);

void MyRSU::initialize(int stage)
{
    DemoBaseApplLayer::initialize(stage);
    if (stage == 0) {
        currentOfferedServiceId = 17; // whatever number you'd like
    }
    else if (stage == 1){
        //startService(Channel::sch2, currentOfferedServiceId, "NumVehicle Service");
        // Let‘s not use the SCH for now
    }
}

void MyRSU::onWSM(BaseFrame1609_4* frame)
{
    if (NumVehicleMsg* wsm = check_and_cast<NumVehicleMsg*>(frame)){
        // check_and_cast will return NULL if frame is not in type NumVehicleMsg
        Coord senderPos = wsm->getSenderPos();
        double distance = (senderPos - curPosition).length();
        std::cout << "CarId: " << wsm->getSenderAddress() << " Distance: " << distance << "\n";
    }
}
```

MyVehicle Class

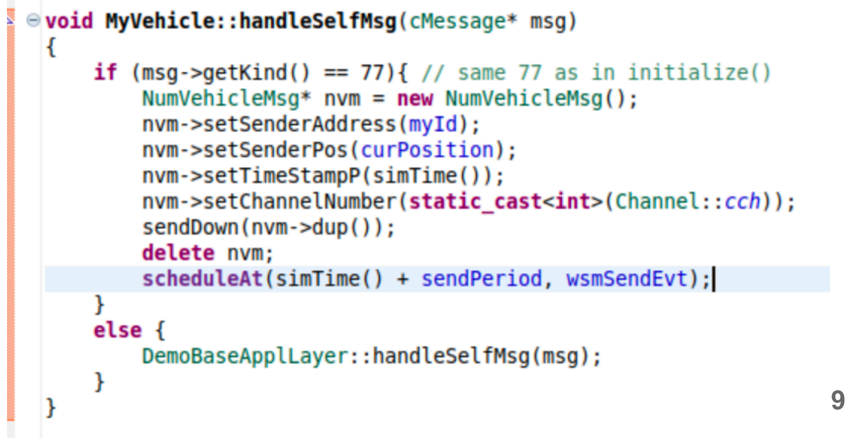
- Let‘s make MyVehicle class which will describe the behavior of the vehicle
- MyVehicle.h
- We want the vehicle to periodically send WSM
- Don‘t forget NED file as well

```
#pragma once

#include
"veins/modules/application/ieee80211p/DemoBaseApplLayer.h"
#include "NumVehicleMsg_m.h"

namespace veins {

class VEINS_API MyVehicle : public DemoBaseApplLayer {
public:
   void initialize(int stage) override;

protected:
   int currentSubscribedServiceId;
   cMessage* wsmSendEvt;
   simtime_t sendPeriod;

protected:
   void onWSM(BaseFrame1609_4* wsm) override;
   void onWSA(DemoServiceAdvertisment* wsa) override;


   void handleSelfMsg(cMessage* msg) override;
};
} // namespace veins
```

MyVehicle Class

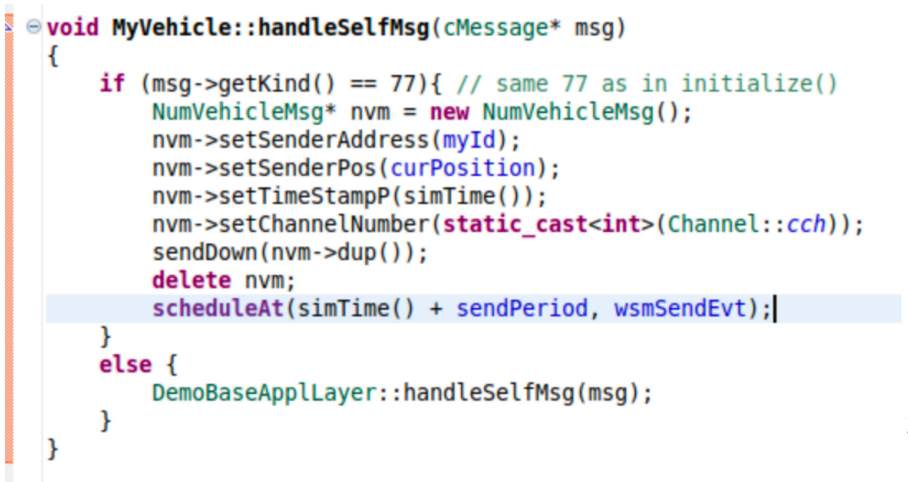
- Mechanism for periodic tasks
- scheduleAt() and handleSelfMsg()
- Do you remember from TicToc?
- You schedule the initial event at initialize()
- The repeating events will be handled in handleSelfMsg()

```
#include "MyVehicle.h"

using namespace veins;

Define_Module(veins::MyVehicle);

void MyVehicle::initialize(int stage){
   DemoBaseApplLayer::initialize(stage);
   if (stage == 0) {
      currentSubscribedServiceId = -1;
      sendPeriod = 0.5;
      wsmSendEvt = new cMessage("wsm send task", 77); //77 is an arbitrary number
   }
   else if (stage == 1){
      scheduleAt(simTime()+sendPeriod, wsmSendEvt);
   }
}

void MyVehicle::onWSA(DemoServiceAdvertisment* wsa){
          /* if (currentSubscribedServiceId == -1) {
             mac->changeServiceChannel(static_cast<Channel>(wsa->getTargetChannel()));
             currentSubscribedServiceId = wsa->getPsid();
           }   */ // we don't need this.. Yet as we are using CCH only
}

void MyVehicle::onWSM(BaseFrame1609_4* frame){
}
```



```
void MyVehicle::handleSelfMsg(cMessage* msg)
{
    if (msg->getKind() == 77){ // same 77 as in initialize()
        NumVehicleMsg* nvm = new NumVehicleMsg();
        nvm->setSenderAddress(myId);
        nvm->setSenderPos(curPosition);
        nvm->setTimeStampP(simTime());
        nvm->setChannelNumber(static_cast<int>(Channel::cch));
        sendDown(nvm->dup());
        delete nvm;
        scheduleAt(simTime() + sendPeriod, wsmSendEvt);
    }
    else {
        DemoBaseApplLayer::handleSelfMsg(msg);
    }
}
```

- And write the channel number in nvm->setChannelNumber()
- There‘s also delete nvm
  - It will cause memory leak without it
  - If you‘re curious, ask me in class
- sendDown() sends the packet created in the WAVE application down to MAC layer (802.11p)
- And MAC layer will handle everything afterwards and send the packet



```
void MyVehicle::handleSelfMsg(cMessage* msg)
{
    if (msg->getKind() == 77){ // same 77 as in initialize()
        NumVehicleMsg* nvm = new NumVehicleMsg();
        nvm->setSenderAddress(myId);
        nvm->setSenderPos(curPosition);
        nvm->setTimeStampP(simTime());
        nvm->setChannelNumber(static_cast<int>(Channel::cch));
        sendDown(nvm->dup());
        delete nvm;
        scheduleAt(simTime() + sendPeriod, wsmSendEvt);
    }
    else {
        DemoBaseApplLayer::handleSelfMsg(msg);
    }
}
```

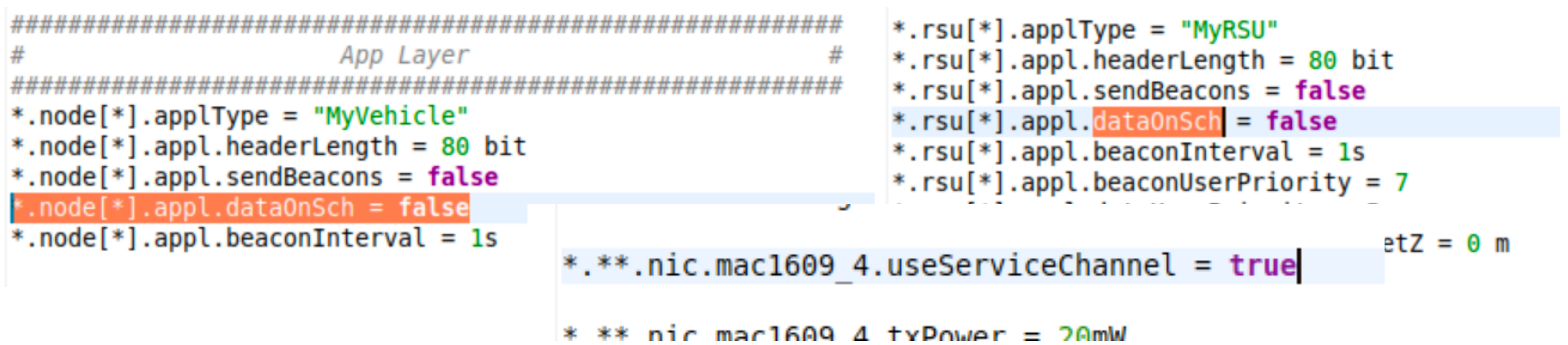
Back to .ini file

- Assign the applications to RSUs and cars (node)

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

*.node[*].applType = "MyVehicle"
```

Back to .ini file

- If we are not using SCH, we should declare as follows



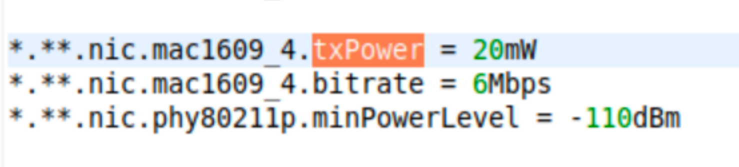
```
##########################################################
#                      App Layer                         #
##########################################################
*.node[*].applType = "MyVehicle"
*.node[*].appl.headerLength = 80 bit
*.node[*].appl.sendBeacons = false
*.node[*].appl.dataOnSch = false
*.node[*].appl.beaconInterval = 1s
```

```
*.rsu[*].applType = "MyRSU"
*.rsu[*].appl.headerLength = 80 bit
*.rsu[*].appl.sendBeacons = false
*.rsu[*].appl.dataOnSch = false
*.rsu[*].appl.beaconInterval = 1s
*.rsu[*].appl.beaconUserPriority = 7
```

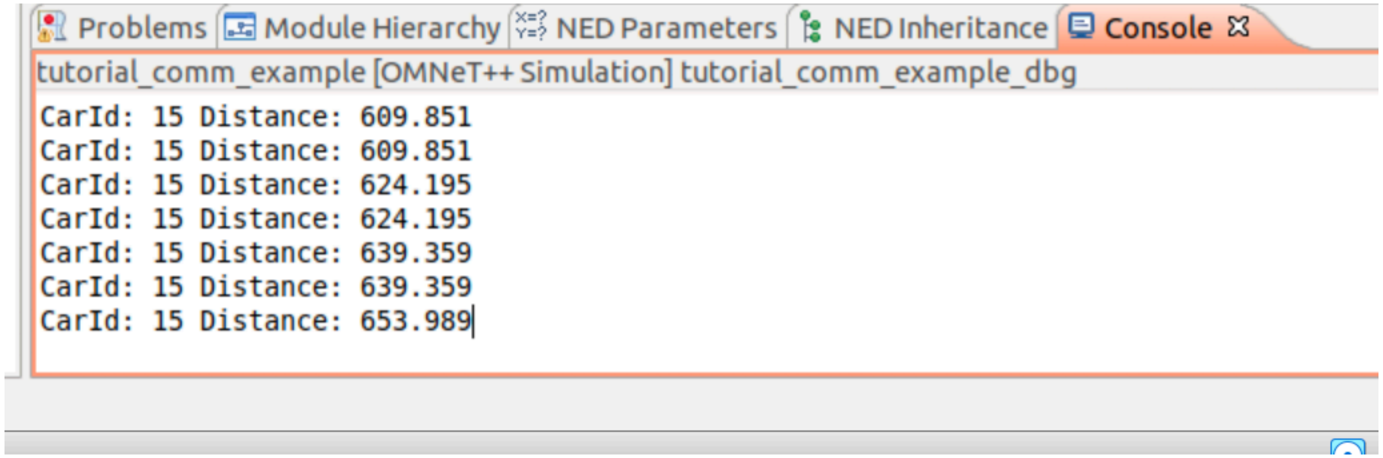
```
etZ = 0 m
*.**.nic.mac1609_4.useServiceChannel = true

*.**.nic.mac1609_4.txPower = 20mW
```

Results

- Distance is printed
- You can also see how far a WSM packet delivers
  - In my simulation it was 653 meters
- Play around with yourself
  - What happens if you move the obstacle in the path?
  - What happens if you change the transmit power? (omnetpp.ini file)
  - Why is the distance repeated twice?
    - We'll come to that later



```
*.**.nic.mac1609_4.txPower = 20mW
*.**.nic.mac1609_4.bitrate = 6Mbps
*.**.nic.phy80211p.minPowerLevel = -110dBm
```



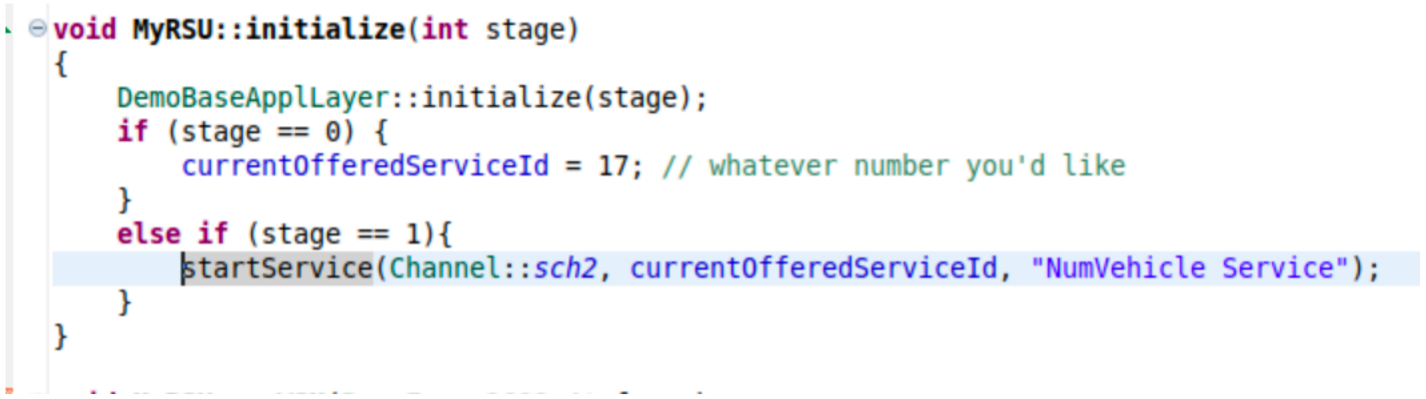
```
tutorial_comm_example [OMNeT++ Simulation] tutorial_comm_example_dbg
CarId: 15 Distance: 609.851
CarId: 15 Distance: 609.851
CarId: 15 Distance: 624.195
CarId: 15 Distance: 624.195
CarId: 15 Distance: 639.359
CarId: 15 Distance: 639.359
CarId: 15 Distance: 653.989
```

WSA and SCH?

- We are not yet using SCH
- We‘ll check that next lecture

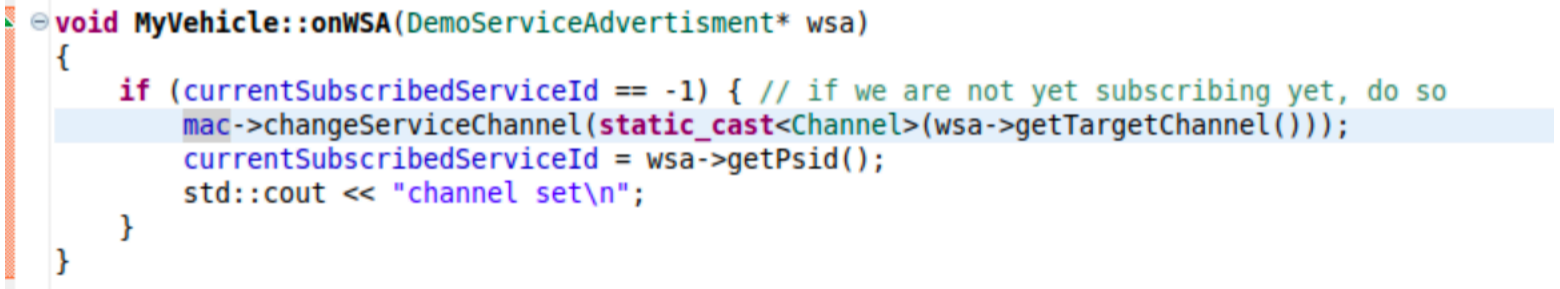
WSA and SCH?

- We‘ve been using only one channel (CCH), and it‘s a waste of bandwidth
- Let‘s roll back changes we‘ve made in the last tutorial
  - From the omnetpp.ini file, make all dataOnSch=true & .userServiceChannel=true
- From MyRSU,



```
void MyRSU::initialize(int stage)
{
    DemoBaseApplLayer::initialize(stage);
    if (stage == 0) {
        currentOfferedServiceId = 17; // whatever number you'd like
    }
    else if (stage == 1){
        startService(Channel::sch2, currentOfferedServiceId, "NumVehicle Service");
    }
}
```

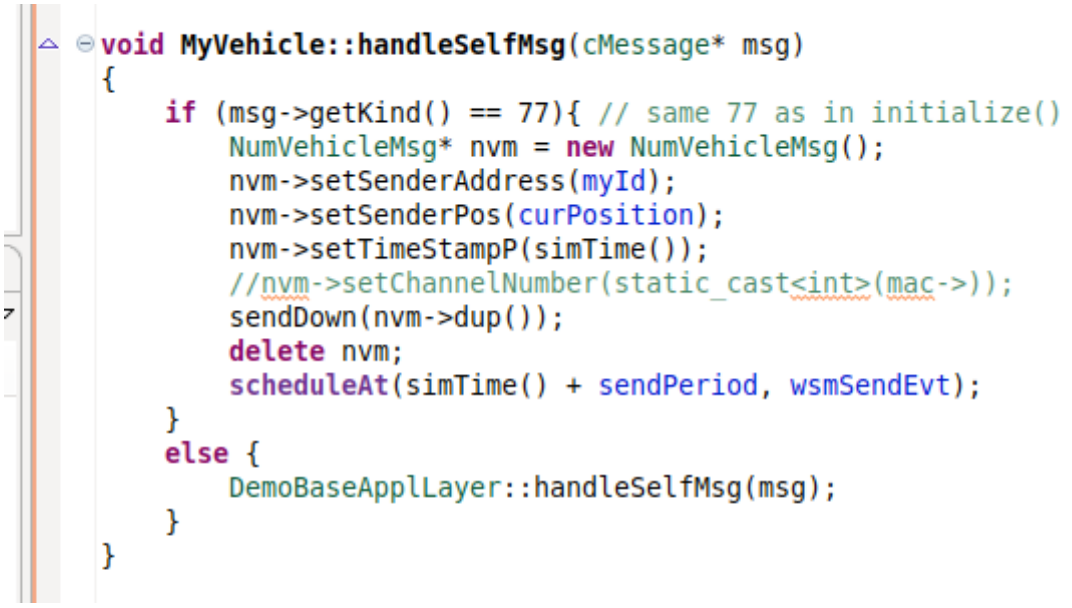
- From MyVehicle.



```
void MyVehicle::onWSA(DemoServiceAdvertisment* wsa)
{
    if (currentSubscribedServiceId == -1) { // if we are not yet subscribing yet, do so
        mac->changeServiceChannel(static_cast<Channel>(wsa->getTargetChannel()));
        currentSubscribedServiceId = wsa->getPsid();
        std::cout << "channel set\n";
    }
}
```

WSA and SCH?

- Now you don‘t have to write the channel on vehicle, but it‘s automatically going to be assigned to the channel you set



```
void MyVehicle::handleSelfMsg(cMessage* msg)
{
    if (msg->getKind() == 77){ // same 77 as in initialize()
        NumVehicleMsg* nvm = new NumVehicleMsg();
        nvm->setSenderAddress(myId);
        nvm->setSenderPos(curPosition);
        nvm->setTimeStampP(simTime());
        //nvm->setChannelNumber(static_cast<int>(mac->));
        sendDown(nvm->dup());
        delete nvm;
        scheduleAt(simTime() + sendPeriod, wsmSendEvt);
    }
    else {
        DemoBaseApplLayer::handleSelfMsg(msg);
    }
}
```