Tutorial 5: Vehicle Speed Control and Service Announcement

Vehicle Speed Control and Service Announcement

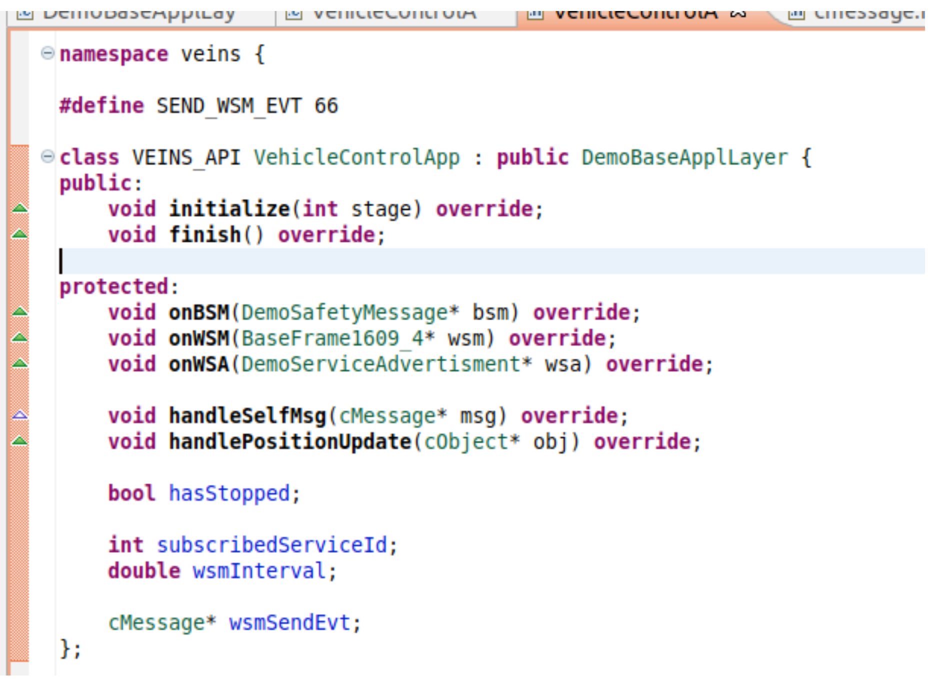
Prof. Sangyoung Park
Module ”Vehicle-2-X: Communication and Control“

Road Network

- Create a 1 km straight stretch of road
- Create an obstacle
- Introduce 5 vehicles that will travel from left to right
- Name the xml files, straight.net.xml, .rou.xml, .poly.xml, .sumocfg

Let‘s make a new WaveApplFile (cc and h)

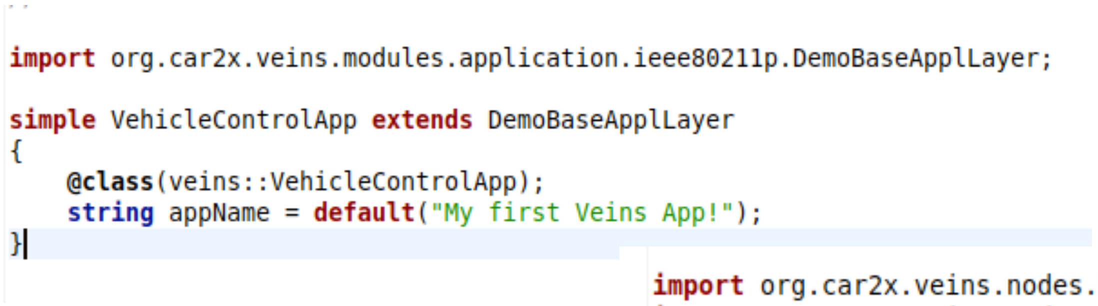
- New-> Class (OMNet++)
  - VehicleControlApp.cc and VehicleControlApp.h are generated
- Let‘s copy the contents from MyVeinsApp.cc/h
  - Veins/src/veins/modules/application/traci/
- But of course, you should change the file content to reflect the name change



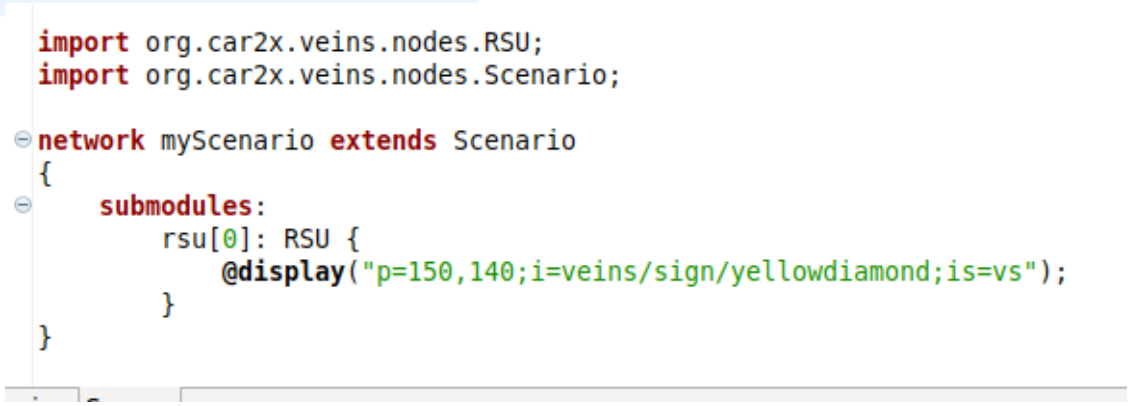
```
namespace veins {

#define SEND_WSM_EVT 66

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

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

    void handleSelfMsg(cMessage* msg) override;
    void handlePositionUpdate(cObject* obj) override;

    bool hasStopped;

    int subscribedServiceId;
    double wsmInterval;

    cMessage* wsmSendEvt;
};
```

Let‘s make a new WaveApplFile (ned)

- New -> Network Description File (NED)
  - Again, copy the contents from MyVeinsApp.ned to VehicleControlApp.ned and fix the names accordingly
- Let‘s not use RSUs this time
  - In myScenario, change rsu[1] to rsu[0], so that 0 RSUs are used



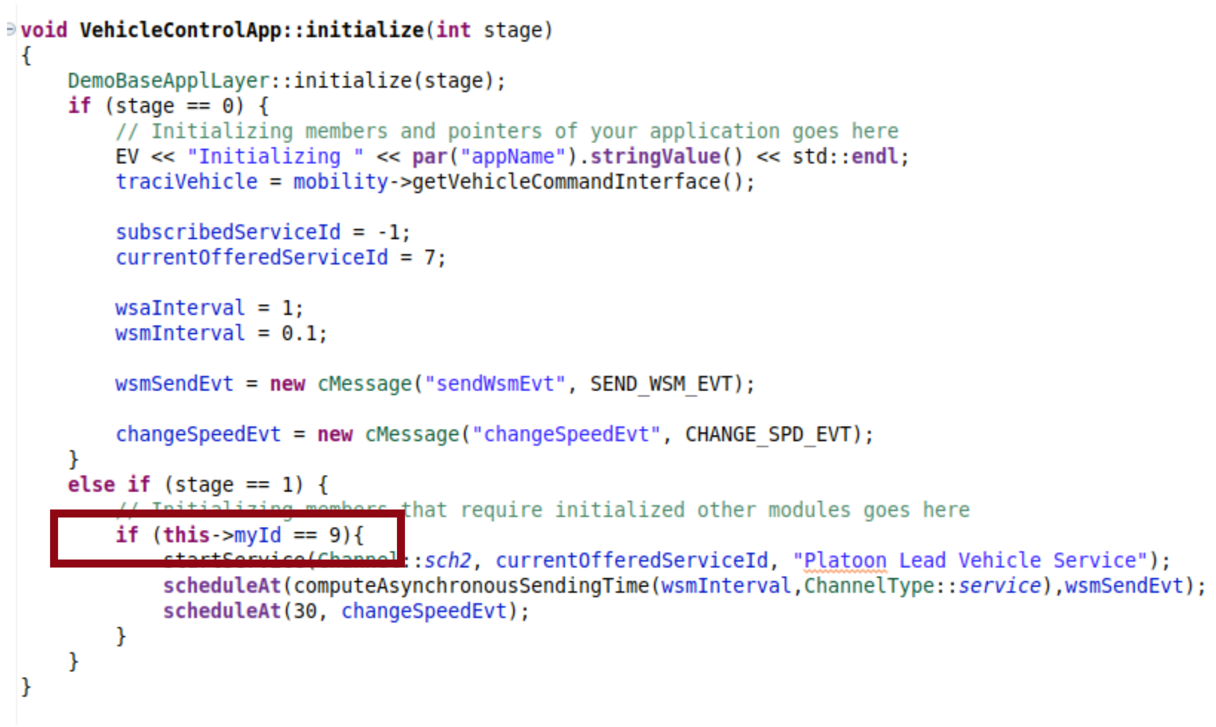
```
import org.car2x.veins.modules.application.ieee80211p.DemoBaseApplLayer;

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



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

network myScenario extends Scenario
{
    submodules:
        rsu[0]: RSU {
            @display("p=150,140;i=veins/sign/yellowdiamond;is=vs");
        }
}
```

Let‘s make Wave Service Announcements (WSA)

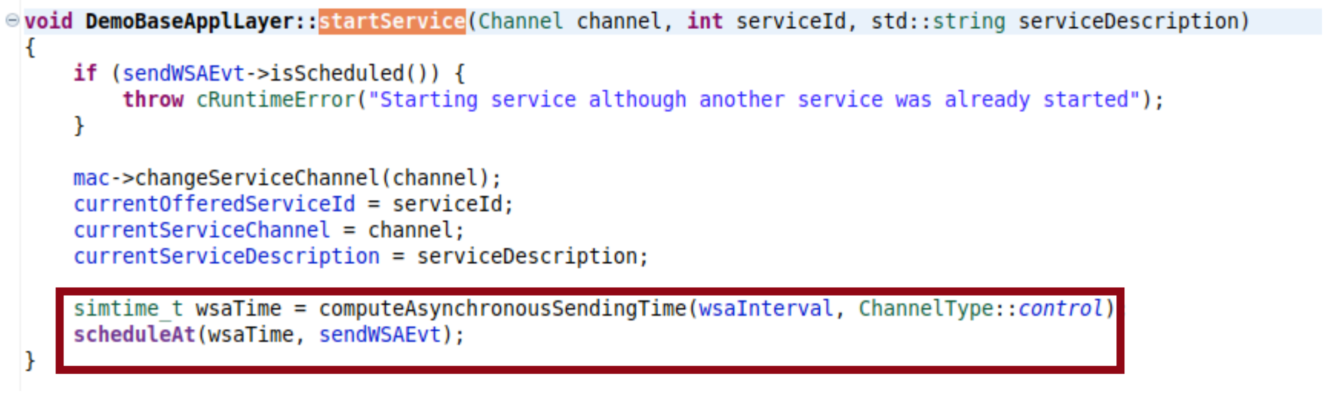
- Let‘s make the first car, which appears on the map, to make the service announcement (WSA)
- We can make use of startService() to start a WAVE service
- However, we don‘t want every car to start their own services



```
void VehicleControlApp::initialize(int stage)
{
    DemoBaseApplLayer::initialize(stage);
    if (stage == 0) {
        // Initializing members and pointers of your application goes here
        EV << "Initializing " << par("appName").stringValue() << std::endl;
        traciVehicle = mobility->getVehicleCommandInterface();

        subscribedServiceId = -1;
        currentOfferedServiceId = 7;

        wsaInterval = 1;
        wsmInterval = 0.1;

        wsmSendEvt = new cMessage("sendWsmEvt", SEND_WSM_EVT);

        changeSpeedEvt = new cMessage("changeSpeedEvt", CHANGE_SPD_EVT);
    }
    else if (stage == 1) {
        // Initializing members that require initialized other modules goes here
        if (this->myId == 9){
            startService(Channel::sch2, currentOfferedServiceId, "Platoon Lead Vehicle Service");
            scheduleAt(computeAsynchronousSendingTime(wsmInterval,ChannelType::service),wsmSendEvt);
            scheduleAt(30, changeSpeedEvt);
        }
    }
}
```

Let‘s make Wave Service Announcements (WSA)

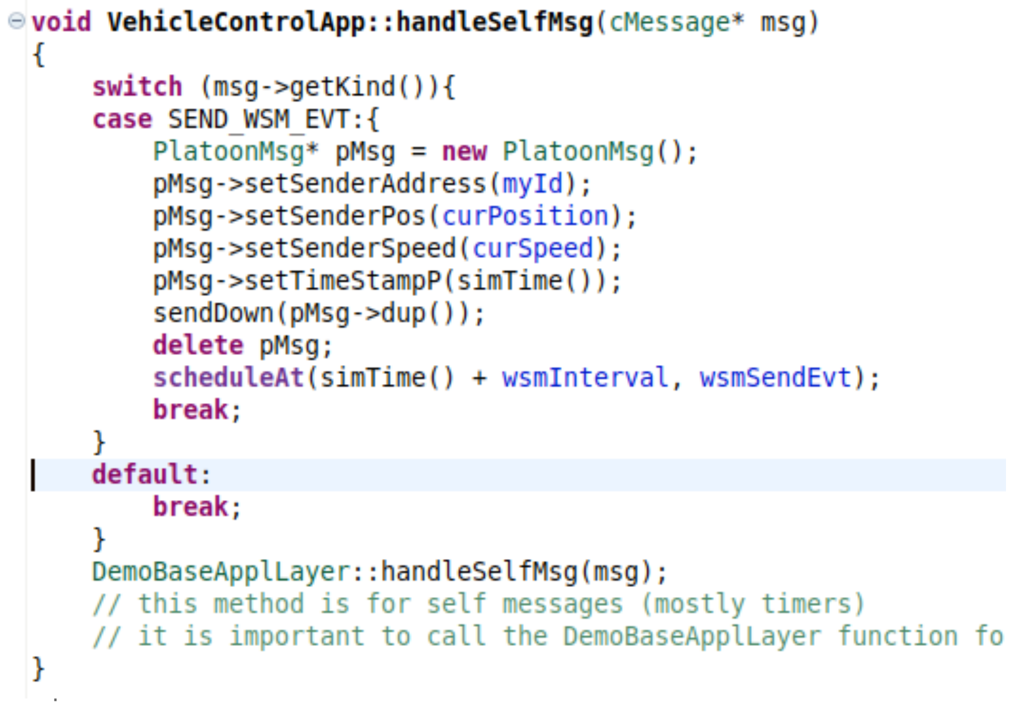
- During the initialization of each car node, we check for the ID by using this->myId
- If it‘s the first car, we call startService(), you can put any number for currentOfferedServiceId
- Types of WAVE messages available in Veins
  - Wave service message (WSM) or BaseFrame1609_4
  - Wave service announcement (WSA) or DemoServiceAdvertisement
  - Basic safety messages (BSM) or DemoSafetyMessage
- If you go inside the function startService(), you will see that WSA will be scheduled using scheduleAt() for the next CCH period



```
void DemoBaseApplLayer::startService(Channel channel, int serviceId, std::string serviceDescription)
{
    if (sendWSAEvt->isScheduled()) {
        throw cRuntimeError("Starting service although another service was already started");
    }

    mac->changeServiceChannel(channel);
    currentOfferedServiceId = serviceId;
    currentServiceChannel = channel;
    currentServiceDescription = serviceDescription;

    simtime_t wsaTime = computeAsynchronousSendingTime(wsaInterval, ChannelType::control);
    scheduleAt(wsaTime, sendWSAEvt);
}
```

Let‘s make Wave Service Announcements (WSA)

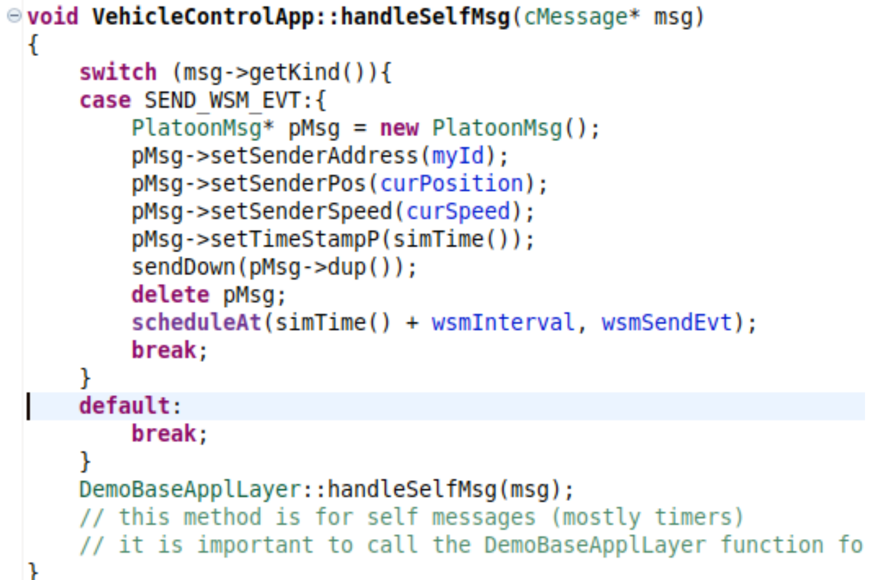
- If you are wondering what scheduleAt() function would end up, it ends up in the following function
- We‘ve used scheduleAt() in initialize() ourselves, and there‘s another scheduleAt() inside the DemoBaseApplLayer::initialize() that we called at the first line of our initialize()
- scheduleAt() invokes the handleSelfMsg() later in simulation time



```
void VehicleControlApp::handleSelfMsg(cMessage* msg)
{
    switch (msg->getKind()){
    case SEND_WSM_EVT:{
        PlatoonMsg* pMsg = new PlatoonMsg();
        pMsg->setSenderAddress(myId);
        pMsg->setSenderPos(curPosition);
        pMsg->setSenderSpeed(curSpeed);
        pMsg->setTimeStampP(simTime());
        sendDown(pMsg->dup());
        delete pMsg;
        scheduleAt(simTime() + wsmInterval, wsmSendEvt);
        break;
    }
    default:
        break;
    }
    DemoBaseApplLayer::handleSelfMsg(msg);
    // this method is for self messages (mostly timers)
    // it is important to call the DemoBaseApplLayer function fo
}
```

Let‘s make Wave Service Announcements (WSA)

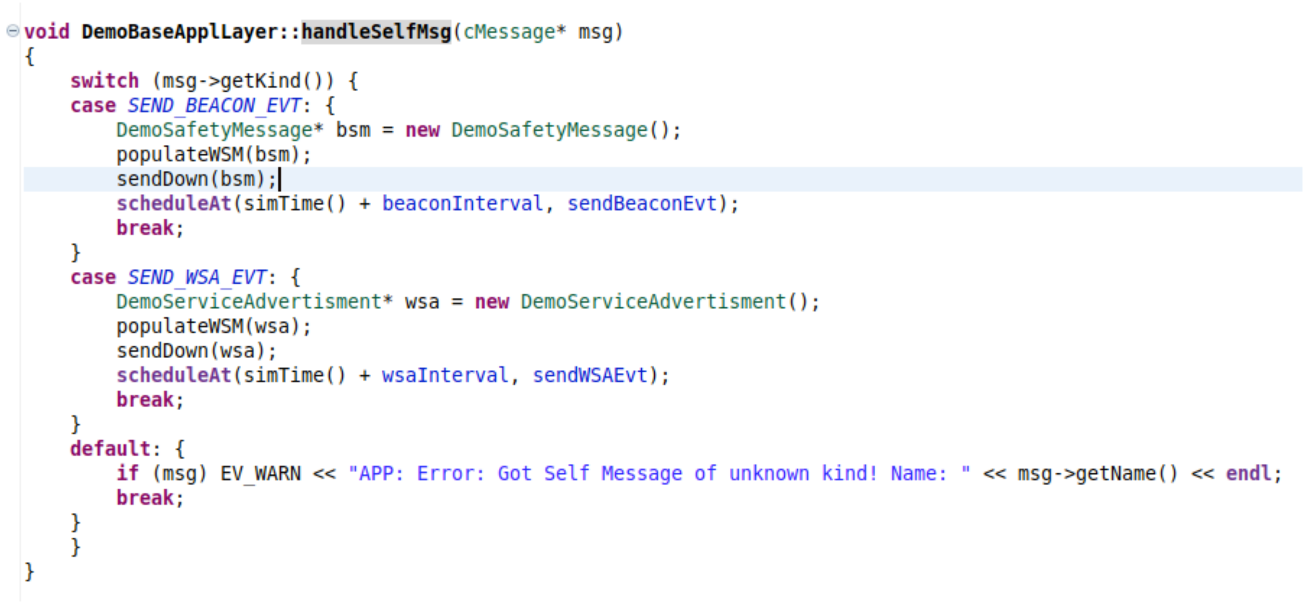
- In this function, we check if the argument passed on to the scheduleAt() was of SEND_WSM does the appropriate task
  - Write senderAddr
  - Write curPosition and curSpeed: This is automatically maintained to have the same value as SUMO simulation, just use the variables
  - send_down a duplicate to the 802.11p MAC layer, which will eventually transmit data, and delete pMsg
  - If you don‘t do this, there‘ll be memory leak (ask me if you want to go deep)
- WSA and BSM will be handled at DemoBaseApplLayer::handleSelfMsg()



```
void VehicleControlApp::handleSelfMsg(cMessage* msg)
{
    switch (msg->getKind()){
    case SEND_WSM_EVT:{
        PlatoonMsg* pMsg = new PlatoonMsg();
        pMsg->setSenderAddress(myId);
        pMsg->setSenderPos(curPosition);
        pMsg->setSenderSpeed(curSpeed);
        pMsg->setTimeStampP(simTime());
        sendDown(pMsg->dup());
        delete pMsg;
        scheduleAt(simTime() + wsmInterval, wsmSendEvt);
        break;
    }
    default:
        break;
    }
    DemoBaseApplLayer::handleSelfMsg(msg);
    // this method is for self messages (mostly timers)
    // it is important to call the DemoBaseApplLayer function fo
}
```

Periodic Transmission of WAVE Messages

- Remember „scheduleAt()“ function from the OMNet++ Tictoc tutorial?
  - scheduleAt() is used for self-messages
- In at DemoBaseApplLayer.cc, there is function handleSelfMsg()
  - Once scheduleAt is used with either SEND_BEACON_EVT or SEND_WSA_EVT kind of cMessages, it‘s going to be re-scheduled periodically because scheduleAt() is called again inside handleSelfMsg()



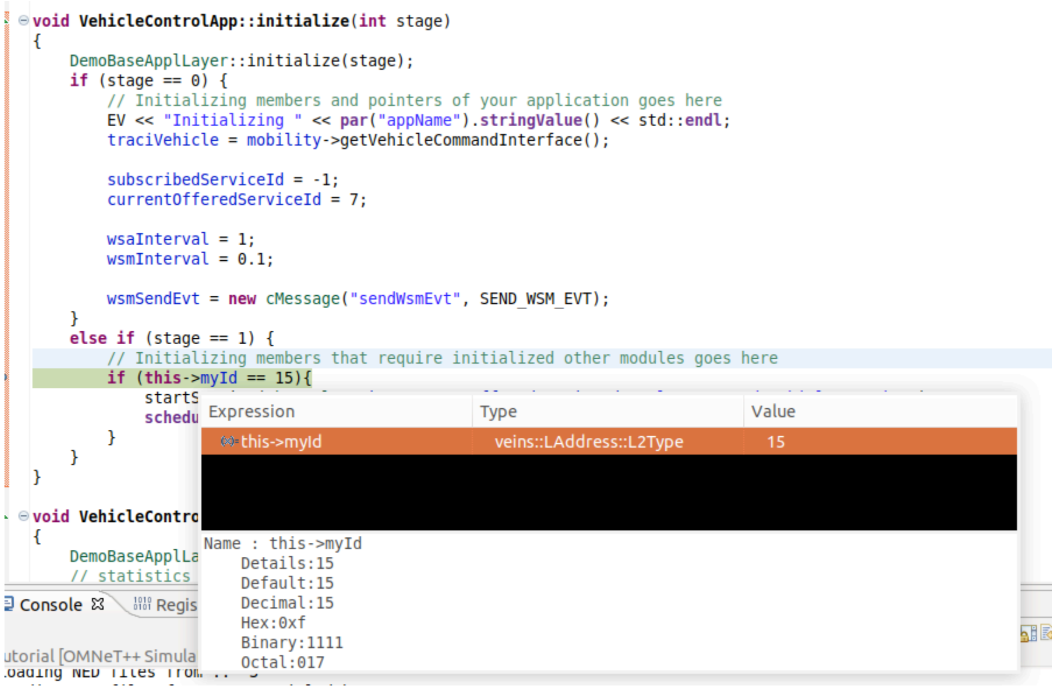
```
void DemoBaseApplLayer::handleSelfMsg(cMessage* msg)
{
    switch (msg->getKind()) {
    case SEND_BEACON_EVT: {
        DemoSafetyMessage* bsm = new DemoSafetyMessage();
        populateWSM(bsm);
        sendDown(bsm);
        scheduleAt(simTime() + beaconInterval, sendBeaconEvt);
        break;
    }
    case SEND_WSA_EVT: {
        DemoServiceAdvertisment* wsa = new DemoServiceAdvertisment();
        populateWSM(wsa);
        sendDown(wsa);
        scheduleAt(simTime() + wsaInterval, sendWSAEvt);
        break;
    }
    default: {
        if (msg) EV_WARN << "APP: Error: Got Self Message of unknown kind! Name: " << msg->getName() << endl;
        break;
    }
    }
}
```

Using Debugger

- Wait, how did I know the ID of the first car would be 9?
- Let‘s use the debugger
- Add the line in the red rectangle to the source code
- Double click on the left to create a „breakpoint“
  - A small blue dot will appear
- Omnetpp.ini (right click) -> debug as -> omnet++ simulation

Using Debugger

- The perspective of the Omnet IDE changes to the „debug perspective“
- If you run, the program will stop at the breakpoint
- If you lay your mouse cursor on top of idDebug, you will be able to see the value of the variable
  - Or, you can look into the sub-window in the top-right corner to find „variables“ window to read the value of the variables or the member variables of the current object (this)
- In my case, the value was 9



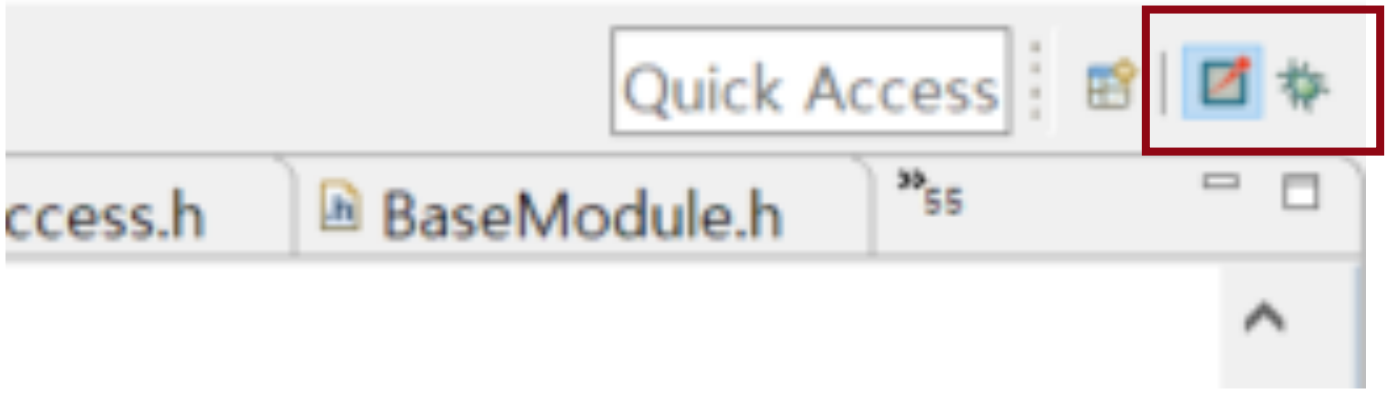
```
void VehicleControlApp::initialize(int stage)
{
    DemoBaseApplLayer::initialize(stage);
    if (stage == 0) {
        // Initializing members and pointers of your application goes here
        EV << "Initializing " << par("appName").stringValue() << std::endl;
        traciVehicle = mobility->getVehicleCommandInterface();

        subscribedServiceId = -1;
        currentOfferedServiceId = 7;

        wsaInterval = 1;
        wsmInterval = 0.1;

        wsmSendEvt = new cMessage("sendWsmEvt", SEND_WSM_EVT);
    }
    else if (stage == 1) {
        // Initializing members that require initialized other modules goes here
        if (this->myId == 15){
            startS
            schedu
        }
    }
}

void VehicleContro
{
    DemoBaseApplLa
    // statistics
```

Expression	Type	Value
this->myId	veins::LAddress::L2Type	15

```
Name : this->myId
    Details:15
    Default:15
    Decimal:15
    Hex:0xf
    Binary:1111
    Octal:017
```

Using Debugger

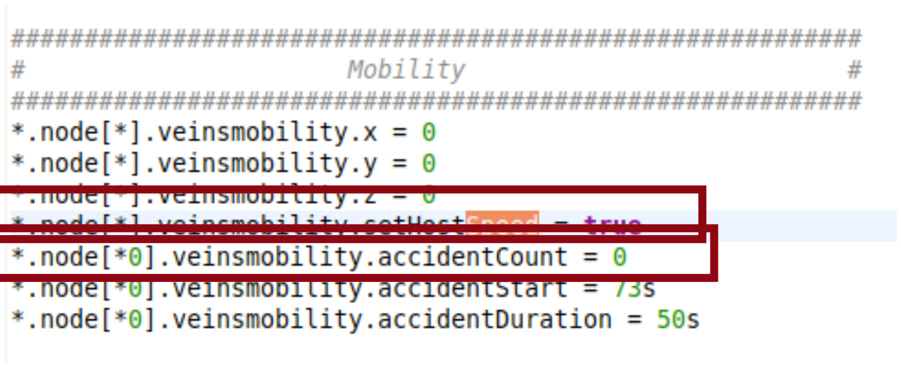
- Changing perspectives, if you want to exit the debugger perspective, you can click on the small buttons on the top-right corner to change perspectives



Oops, errors exist

- We haven‘t allowed the usage of SCH
  - We can configure such parameters in the omnetpp.ini file
- We also need to configure veins simulator to update curSpeed every time step
  - Without this, curSpeed is always 0
- Let‘s also remove the accident

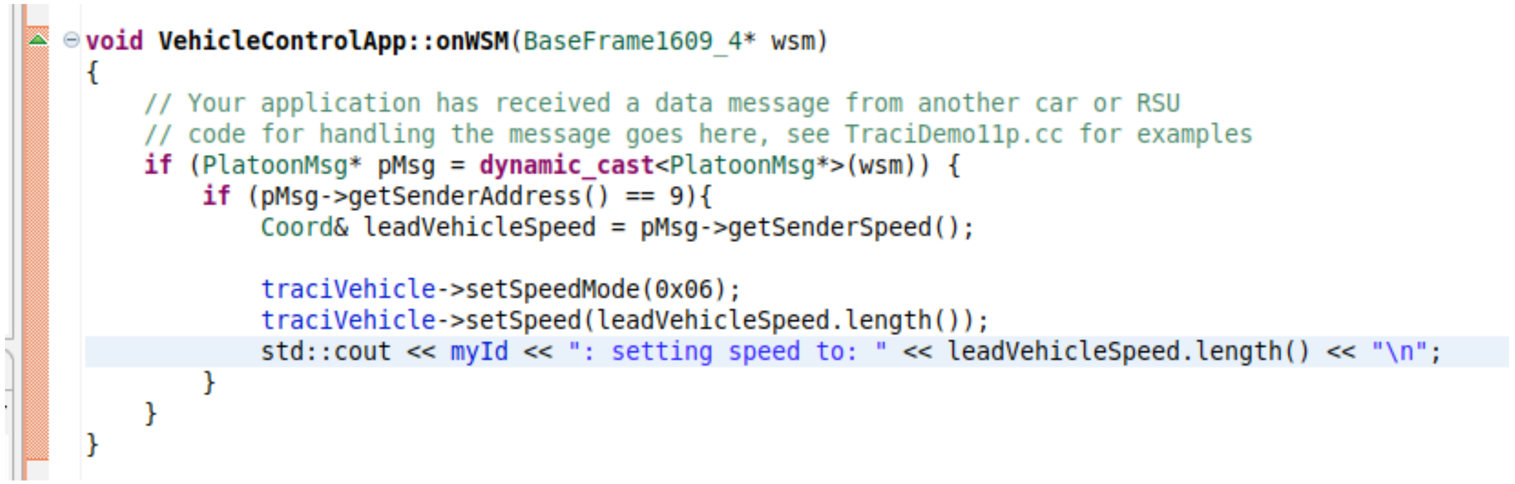
```
##############################################################
#              11p specific parameters                       #
#                                                            #
#                        NIC-Settings                        #
##############################################################
*.connectionManager.sendDirect = true
*.connectionManager.maxInterfDist = 2600m
*.connectionManager.drawMaxIntfDist = false

*.**.nic.mac1609_4.useServiceChannel = true

*.**.nic.mac1609_4.txPower = 20mW
*.**.nic.mac1609_4.bitrate = 6Mbps
*.**.nic.phy80211p.sensitivity = -89dBm

*.**.nic.phy80211p.useThermalNoise = true
*.**.nic.phy80211p.thermalNoise = -110dBm

*.**.nic.phy80211p.decider = xmldoc("config.xml")
*.**.nic.phy80211p.analogueModels = xmldoc("config.xml")
*.**.nic.phy80211p.usePropagationDelay = true

*.**.nic.phy80211p.antenna = xmldoc("antenna.xml",
"/root/Antenna[@id='monopole']")
```



```
#########################################################
#                       Mobility                        #
#########################################################
*.node[*].veinsmobility.x = 0
*.node[*].veinsmobility.y = 0
*.node[*].veinsmobility.z = 0
*.node[*].veinsmobility.setHostSpeed = true
*.node[*0].veinsmobility.accidentCount = 0
*.node[*0].veinsmobility.accidentStart = 73s
*.node[*0].veinsmobility.accidentDuration = 50s
```

Behavior of vehicles upon receiving a WSM

- Upon receiving the WSM from the leading vehicle, we can adjust the speed of the vehicle (let‘s match the leader‘s speed for now)
- onWSM will be invoked upon receiving a PlatoonMsg because it‘s a child class
  - We check whether the receive message is of type PlatoonMsg using dynamic_cast -> Google it and ask me if you don‘t understand!
- Why do I use .length() for speed?
  - Try to find out by exploring the „Coord“ class (hint: it‘s a vector)



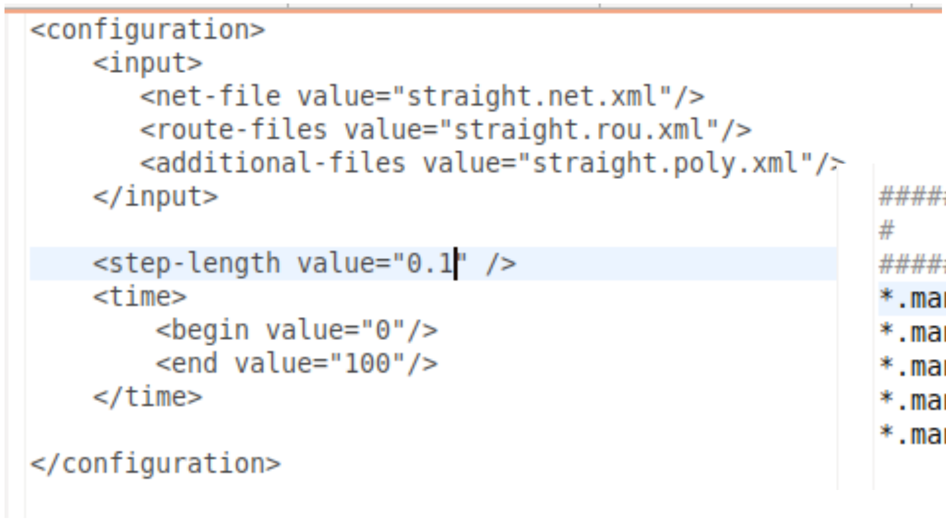
```
void VehicleControlApp::onWSM(BaseFrame1609_4* wsm)
{
    // Your application has received a data message from another car or RSU
    // code for handling the message goes here, see TraciDemo11p.cc for examples
    if (PlatoonMsg* pMsg = dynamic_cast<PlatoonMsg*>(wsm)) {
        if (pMsg->getSenderAddress() == 9){
            Coord& leadVehicleSpeed = pMsg->getSenderSpeed();

            traciVehicle->setSpeedMode(0x06);
            traciVehicle->setSpeed(leadVehicleSpeed.length());
            std::cout << myId << ": setting speed to: " << leadVehicleSpeed.length() << "\n";
        }
    }
}
```

Simulation Speed

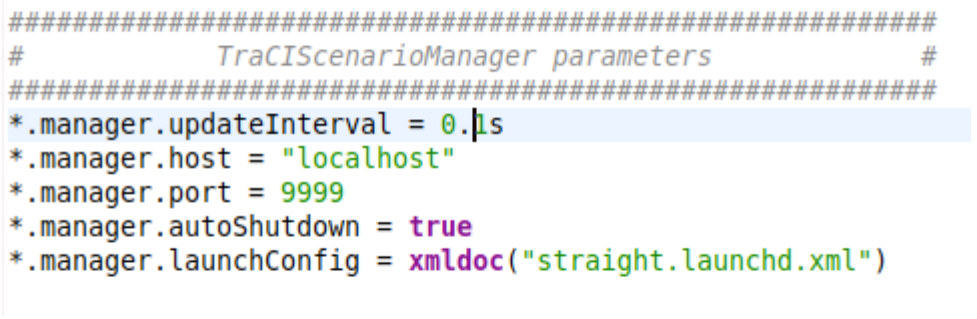
- Now, the speed of the lead vehicle is shared with other vehicles every 0.1 s
- There‘s too much animation going on to describe the packet movements
- We can speed up the simulation by adjusting the amount of animation we want to view
- In the omnet simulation environment
  - Simulate -> Fast run / Express run
  - Simulation will be performed with increased speed without executing the animations

Enhance the time granularity of SUMO

- By default, SUMO time granularity is 1 second, which is too slow for vehicle control
- We can change that in sumocfg file <step-length value="0.1">
- Also Veins should get updated from SUMO more frequently
- In the omnetpp.ini file you can do that *.manager.updateInterval



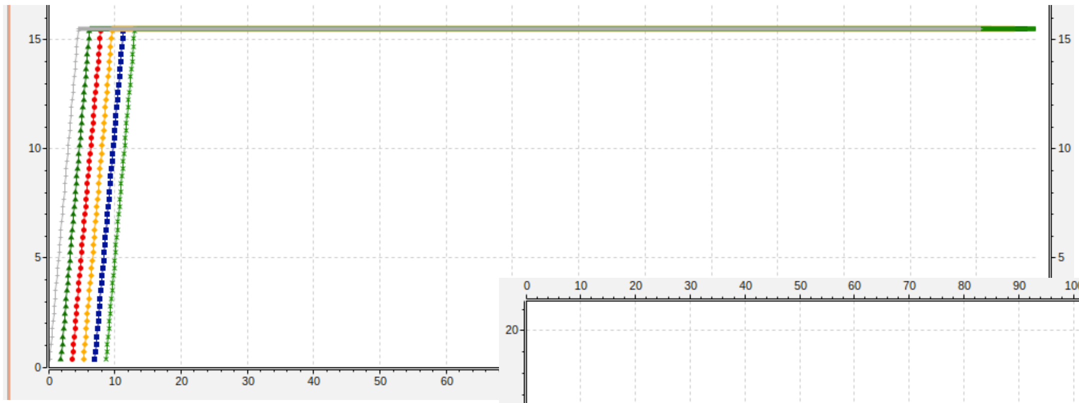
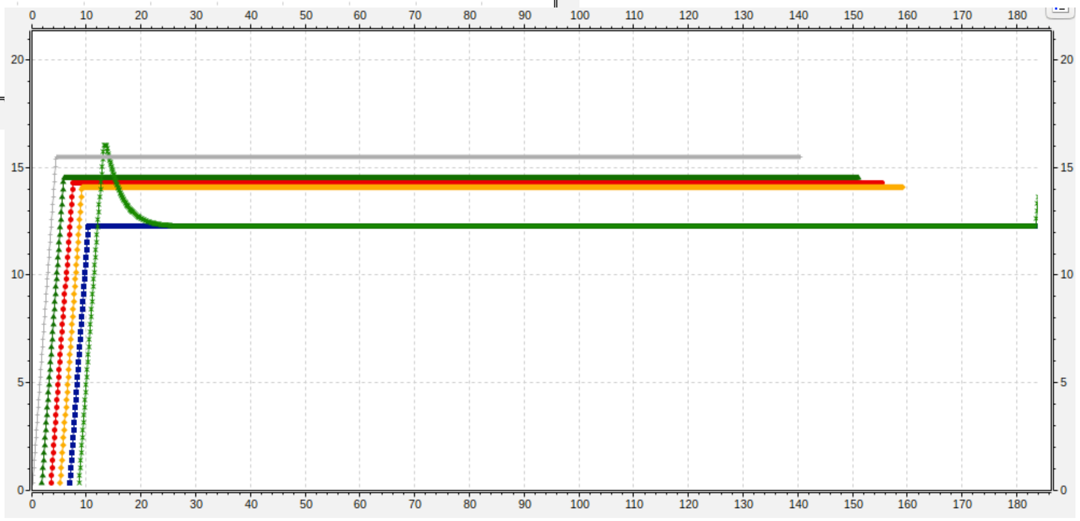
```
<configuration>
    <input>
       <net-file value="straight.net.xml"/>
       <route-files value="straight.rou.xml"/>
       <additional-files value="straight.poly.xml"/>
    </input>

    <step-length value="0.1" />
    <time>
        <begin value="0"/>
        <end value="100"/>
    </time>

</configuration>
```



```
##########################################################
#              TraCIScenarioManager parameters           #
##########################################################
*.manager.updateInterval = 0.1s
*.manager.host = "localhost"
*.manager.port = 9999
*.manager.autoShutdown = true
*.manager.launchConfig = xmldoc("straight.launchd.xml")
```

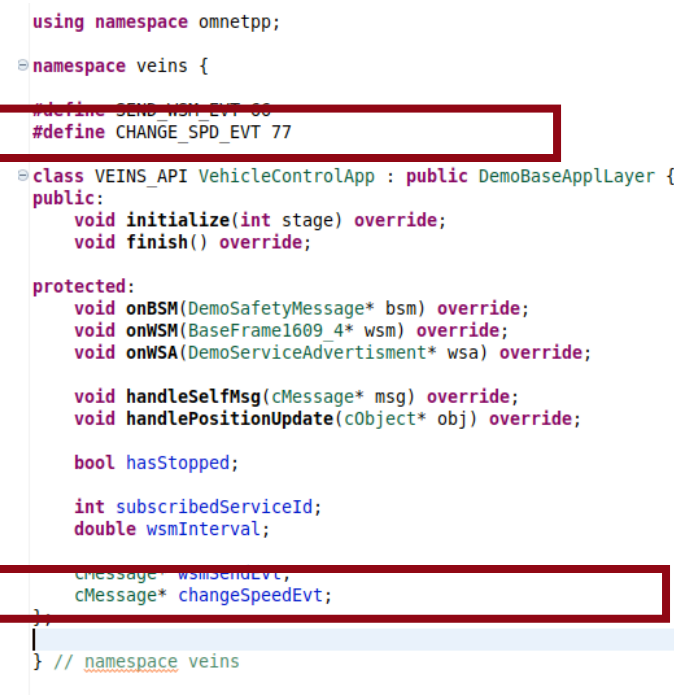
Result: Speed Profile

- It's honestly a bit boring. Could you compare it with a case where you don't update the speed? (comment out the part where you set speed)
  - Then default driver model (SUMO Krauss) kicks in

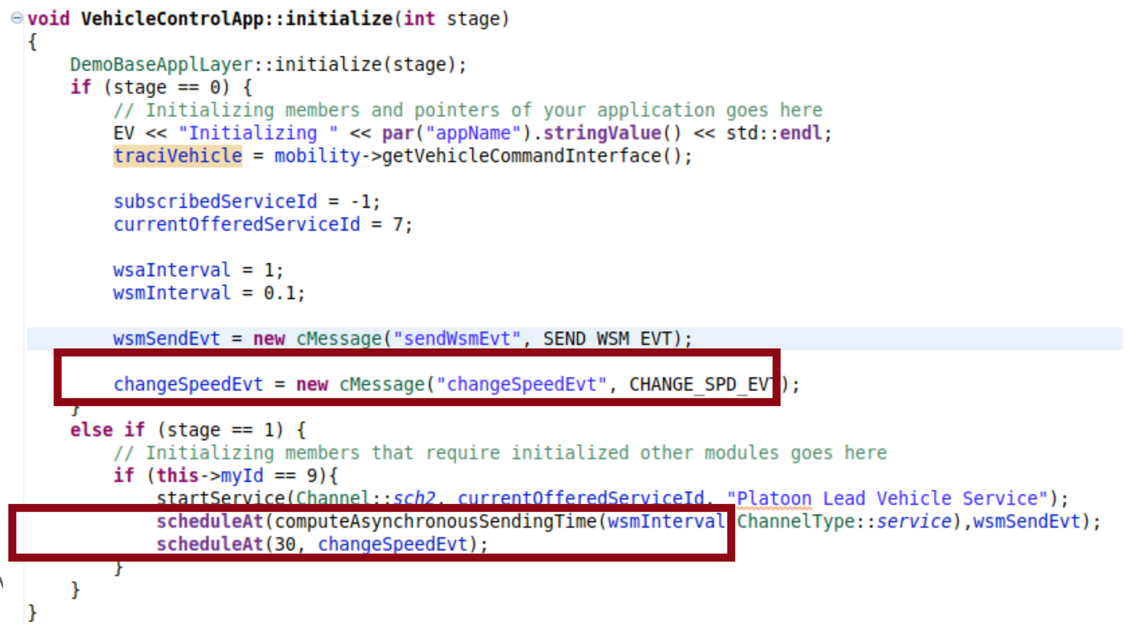



Play around with leader speed

- Shall we try to modify the lead vehicle’s speed profile?
- Let’s add changeSpeedEvt to .h file
- And define CHANGE_SPD_EVT
- We initialize changeSpeedEvt in .cc file
- And scheduleAt at 30 seconds (only for lead vehicle)



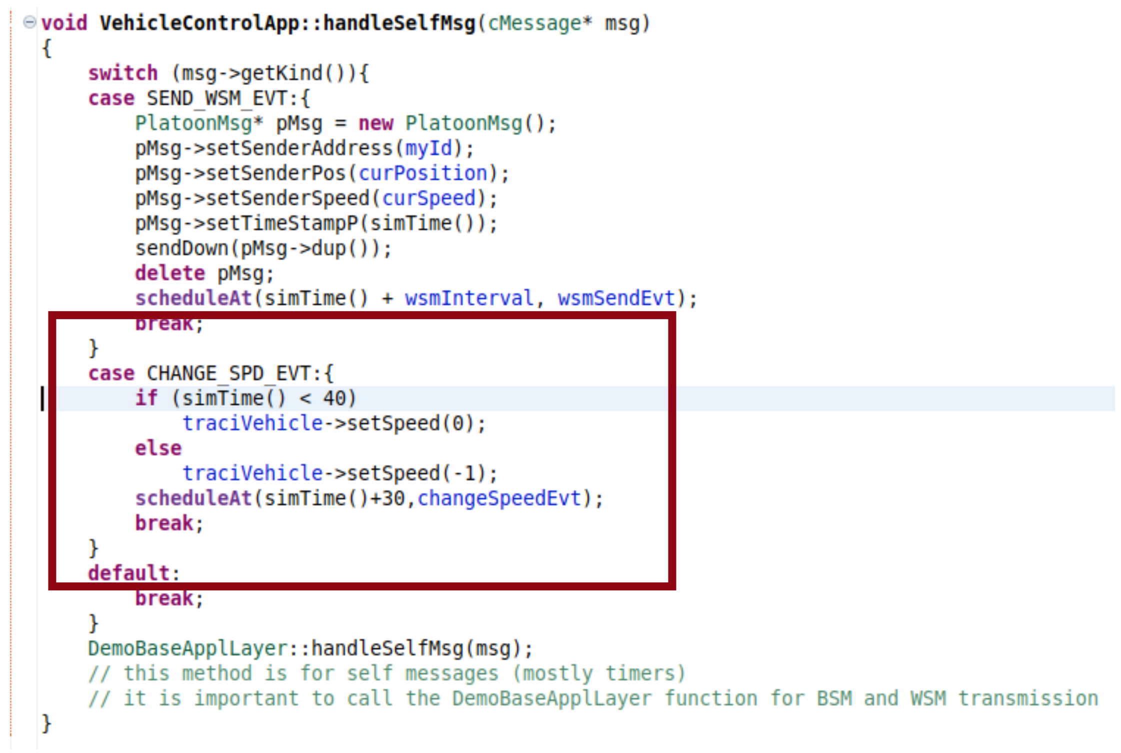
```
using namespace omnetpp;

namespace veins {

#define SEND_WSM_EVT 66
#define CHANGE_SPD_EVT 77

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

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

    void handleSelfMsg(cMessage* msg) override;
    void handlePositionUpdate(cObject* obj) override;

    bool hasStopped;

    int subscribedServiceId;
    double wsmInterval;

    cMessage* wsmSendEvt;
    cMessage* changeSpeedEvt;
};

} // namespace veins
```



```
void VehicleControlApp::initialize(int stage)
{
    DemoBaseApplLayer::initialize(stage);
    if (stage == 0) {
        // Initializing members and pointers of your application goes here
        EV << "Initializing " << par("appName").stringValue() << std::endl;
        traciVehicle = mobility->getVehicleCommandInterface();

        subscribedServiceId = -1;
        currentOfferedServiceId = 7;

        wsaInterval = 1;
        wsmInterval = 0.1;

        wsmSendEvt = new cMessage("sendWsmEvt", SEND_WSM_EVT);

        changeSpeedEvt = new cMessage("changeSpeedEvt", CHANGE_SPD_EVT);
    }
    else if (stage == 1) {
        // Initializing members that require initialized other modules goes here
        if (this->myId == 9){
            startService(Channel::sch2, currentOfferedServiceId, "Platoon Lead Vehicle Service");
            scheduleAt(computeAsynchronousSendingTime(wsmInterval, ChannelType::service),wsmSendEvt);
            scheduleAt(30, changeSpeedEvt);
        }
    }
}
```

Play around with leader speed

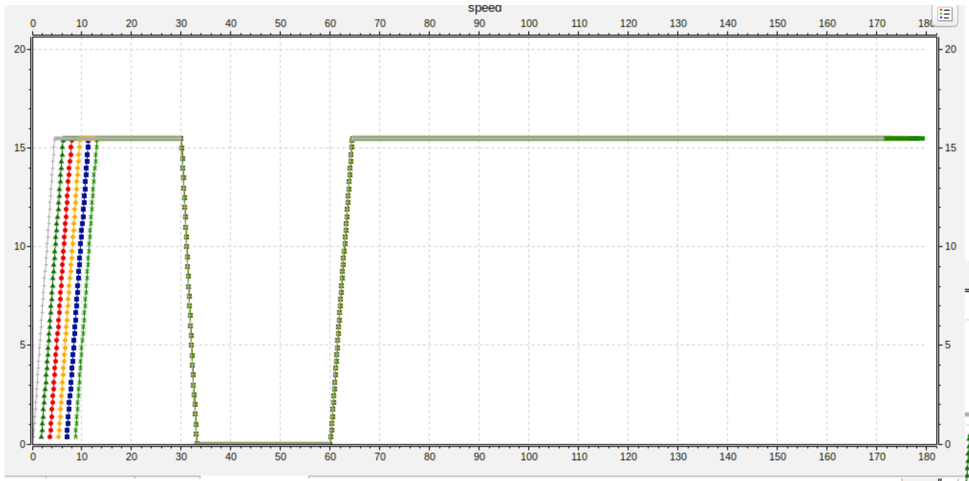
- When CHANGE_SPD_EVT happens, let's stop the lead vehicle
- Lead vehicle will stop at 30 second and resume driving at 60 (try to understand why, I coded in a mediocre way)

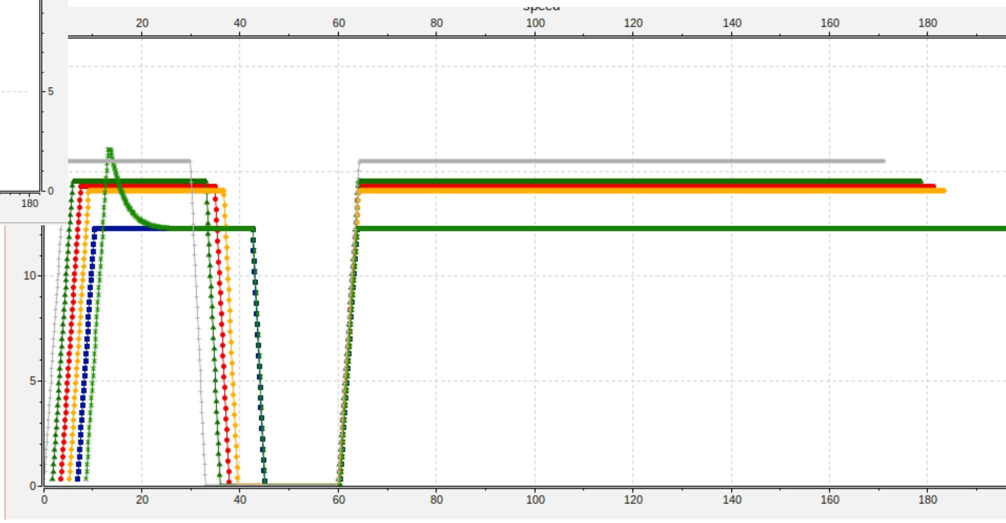


```
void VehicleControlApp::handleSelfMsg(cMessage* msg)
{
    switch (msg->getKind()){
    case SEND_WSM_EVT:{
        PlatoonMsg* pMsg = new PlatoonMsg();
        pMsg->setSenderAddress(myId);
        pMsg->setSenderPos(curPosition);
        pMsg->setSenderSpeed(curSpeed);
        pMsg->setTimeStampP(simTime());
        sendDown(pMsg->dup());
        delete pMsg;
        scheduleAt(simTime() + wsmInterval, wsmSendEvt);
        break;
    }
    case CHANGE_SPD_EVT:{
        if (simTime() < 40)
            traciVehicle->setSpeed(0);
        else
            traciVehicle->setSpeed(-1);
        scheduleAt(simTime()+30,changeSpeedEvt);
        break;
    }
    default:
        break;
    }
    DemoBaseApplLayer::handleSelfMsg(msg);
    // this method is for self messages (mostly timers)
    // it is important to call the DemoBaseApplLayer function for BSM and WSM transmission
}
```

Results

- With & without setting vehicle speed the same as the lead vehicle speed





Implementation of Simple Platooning Algorithm

- Let‘s start with something simple
- Let‘s read the distance to the preceding vehicle only and try to adjust the acceleration of the current vehicle
- Would you be able to implement this? (next Tutorial)
- $a = p \cdot (d - d_{desired})$