UNIVERSITY OF TRENTO
Department of Information Engineering and Computer Science

Wireless Mesh and Vehicular Networks

Simulation of Vehicular Networks

Michele Segata, Renato Lo Cigno - University of Trento

with special thanks to
Falko Dressler, Christoph Sommer, Bastian Bloessl, Stefan Joerer, David Eckhoff

A (rough) outline of the Vehicular Networks topics

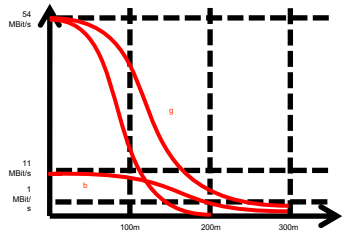
- ~~Application: why VN?~~
- ~~Communication: technologies, alternatives, protocols, challenges~~
- Simulation: evaluating vehicular networks without vehicles and without networks. Tools and models

SIMULATION

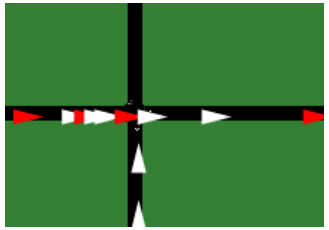
Approaches to Performance Evaluation

- Field Operational Tests
  - + Highest degree of realism
  - - no in-depth investigations of network behavior
  - - Non-suppressible side effects
  - - Limited extrapolation from field operational tests

- Analytical evaluation
  - + Closed-form description allows for far-reaching conclusions
  - - May need to oversimplify complex systems



- Simulation
  - – Can serve as middle ground

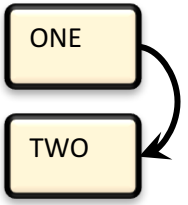


Requirements for Simulation

- Models
  - Network protocol layers
  - Radio propagation
  - Node mobility
  - Model of approach to be investigated (e.g., flooding)
- Scenarios
  - Road geometry, traffic lights, meta information
  - Normal traffic pattern
  - Scenario of use case to be investigated (e.g., accident)
- Metrics
  - Network traffic metrics (delay, load, …)
  - Road traffic metrics (travel time, stopping time, emissions, …)
  - Metric of use case to be investigated (e.g., time until jam resolved)

Modeling Network Protocols

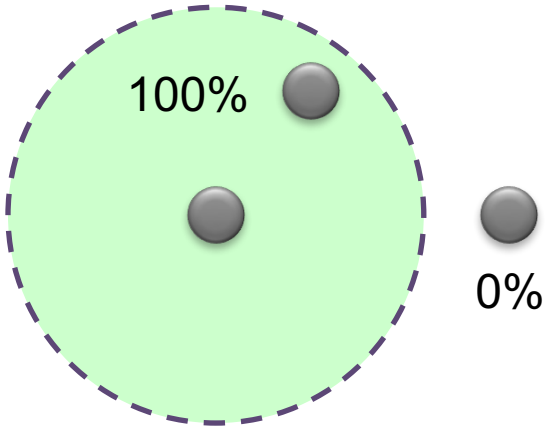
- Dedicated simulation tools
  - Discrete Event Simulation (DES) kernel
  - Manages queue of events (e.g., “an IP fragment was received”)
  - Delivers events to simulation models
- Model libraries
  - Simulate components’ reaction to events
  - E.g., HTTP server, TCP state machine, radio channel, human, …
  - “when enough IP fragments received ⇨ tell TCP: packet received”



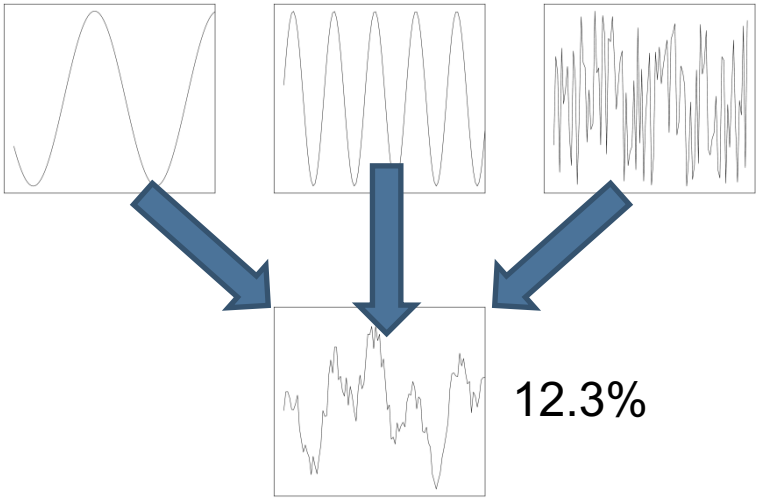
Engine	Language	Library	Language
OMNeT++	C++	MiXiM	C++
ns-2 / ns-3	C++	ns-2 / ns-3	Objective Tcl / Python
JiST	Java	SWANS	Java

Modeling Radio Channel

- Simple model: unit disk
  - Fixed radio “range”
  - Node within range
    ⇔ packet received



- Enhanced models:
  - For each packet, consider
    - Signal strength
    - Interference (other radios)
    - Noise (e.g., thermal noise)
  - Calculate “signal to noise and interference ratio” (SNIR)
  - Derive packet error rate (PER)



Modeling Radio Propagation

- Signal attenuation
  - Received power depends on transmitted power, antenna gains, and path loss

$$P_r[dBm] = P_t[dBm] + G_t[dB] + G_r[dB] - \sum L_x[dB]$$

- Free space path loss

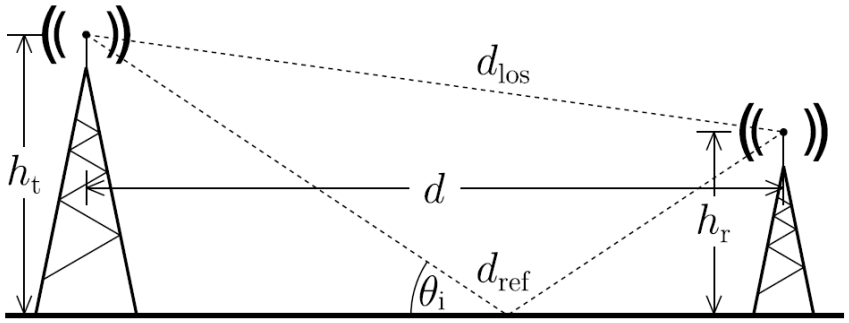
$$L_{\text{freespace}}[\text{dB}] = 20 \lg \left(4\pi \frac{d}{\lambda} \right)$$

- Empirical free space path loss

$$L_{\text{freespace,emp}}[\text{dB}] = 10 \lg \left(4\pi \frac{d}{\lambda} \right)^{\alpha}$$

UNIVERSITY OF TRENTO
Department of Information Engineering and Computer Science

- Two Ray Interference path loss



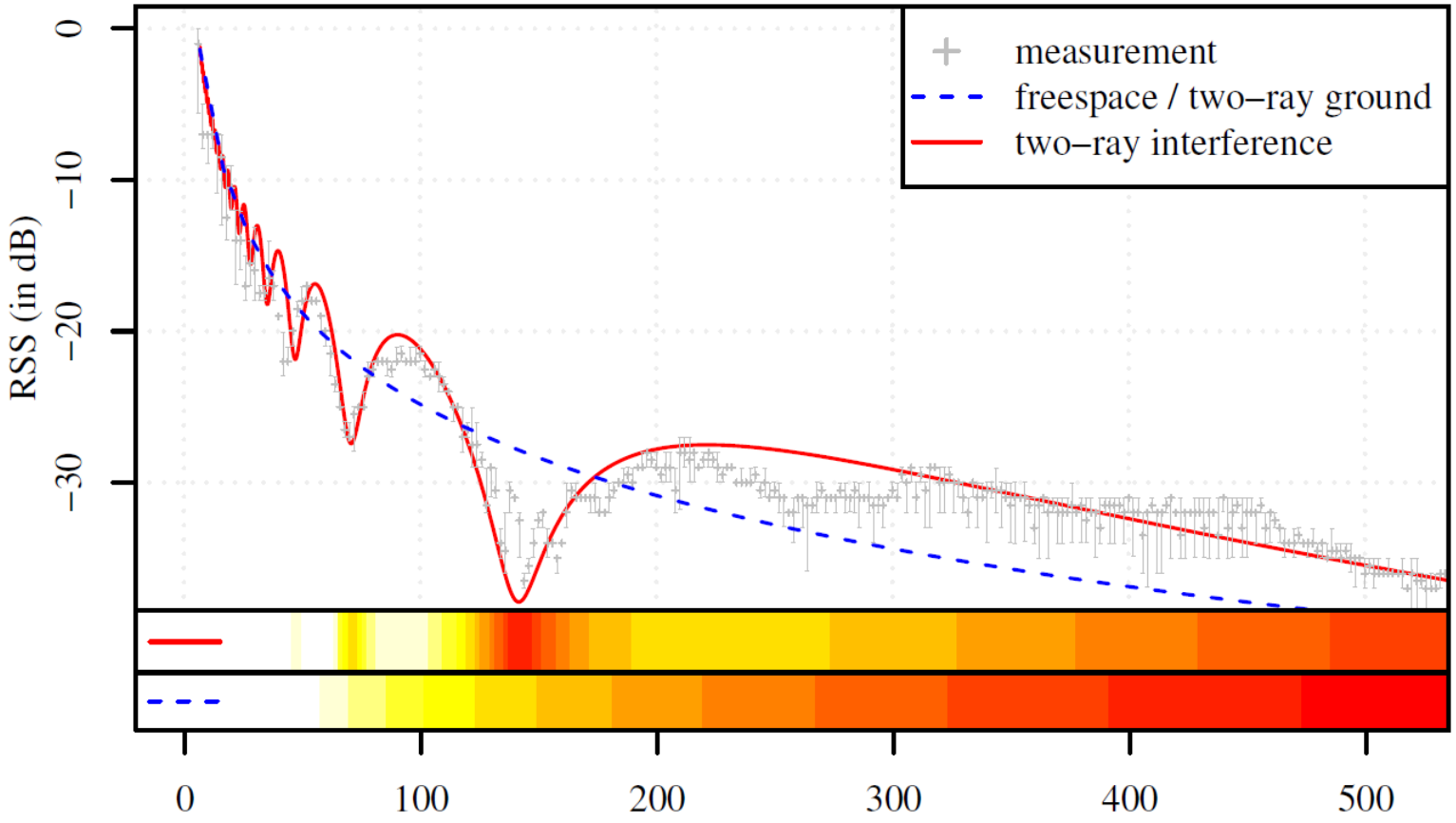
$$L_{\text{tri}}[\text{dB}] = 20 \lg \left(4\pi \frac{d}{\lambda} \left| 1 + \Gamma_{\perp} e^{i\varphi} \right|^{-1} \right), \text{ substituting}$$

$$\varphi = 2\pi \frac{d_{\text{los}} - d_{\text{ref}}}{\lambda}, \; \Gamma_{\perp} = \frac{\sin\theta_{\text{i}} - \sqrt{\epsilon_{\text{r}} - \cos^2\theta_{\text{i}}}}{\sin\theta_{\text{i}} + \sqrt{\epsilon_{\text{r}} - \cos^2\theta_{\text{i}}}},$$

$$d_{\text{los}} = \sqrt{d^2 + (h_{\text{t}} - h_{\text{r}})^2}, \; d_{\text{ref}} = \sqrt{d^2 + (h_{\text{t}} + h_{\text{r}})^2},$$

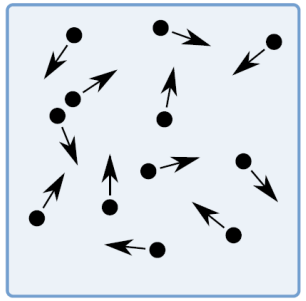
$$\sin\theta_{\text{i}} = (h_{\text{t}} + h_{\text{r}}) / d_{\text{ref}}, \; \cos\theta_{\text{i}} = d / d_{\text{ref}}.$$

- Comparison: Two Ray Interference vs. Free Space

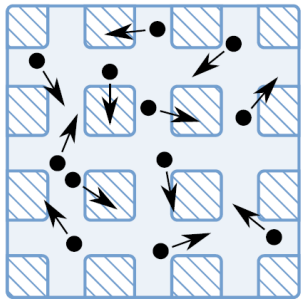


Modeling Mobility

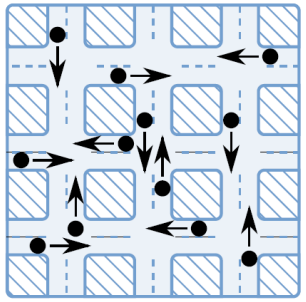
- Traditional approach in network simulation: Random Waypoint (RWP)
  - „pick destination, move there, repeat“



- First adaptation to vehicular movement
  - Add mass, inertia
  - Add restriction to “roads”
  - Add angular restrictions



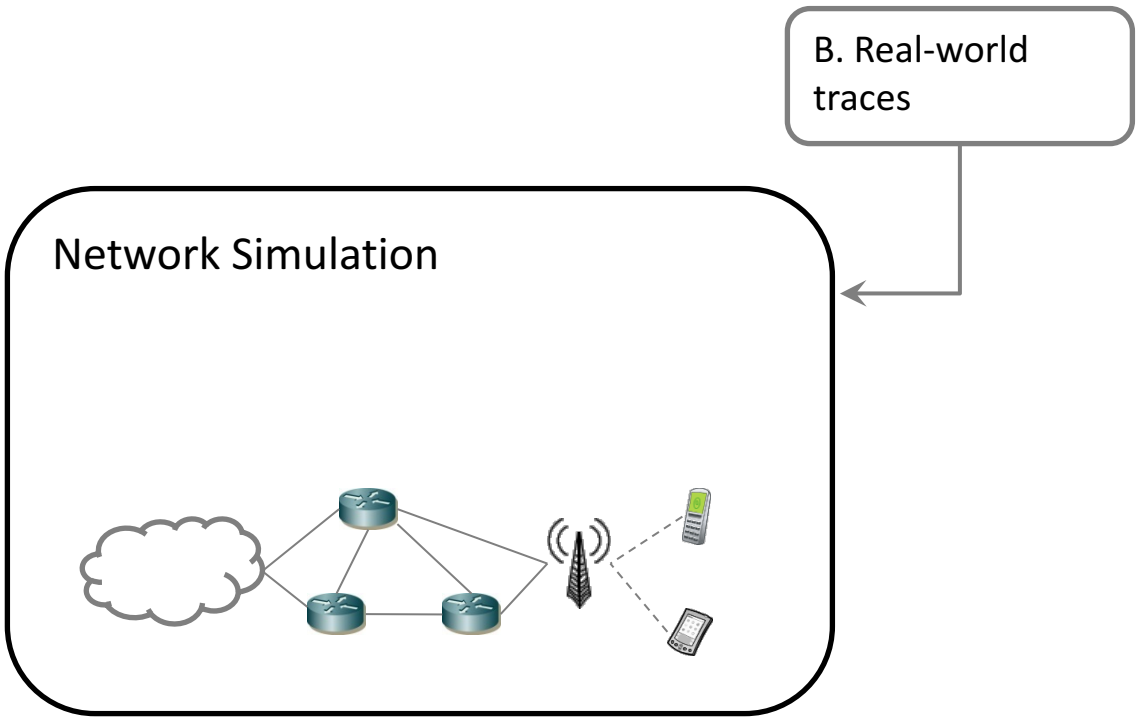
- Problem
  - Very unrealistic (longitudinal) mobility pattern



[1] J. Yoon, M. Liu, and B. Noble, "**Random waypoint considered harmful**," Proceedings of 22nd IEEE Conference on Computer Communications (IEEE INFOCOM 2003), vol. 2, San Francisco, CA, March 2003, pp. 1312-1321

- First approach: Replay recorded trace data
  - Use GPS
  - Install in Taxi, Bus, …
  - Highest degree of realism

- Problems:
  - Invariant scenario
  - No extrapolation
    - To other vehicles (cars, trucks, …)
    - To more vehicles
    - To fewer vehicles

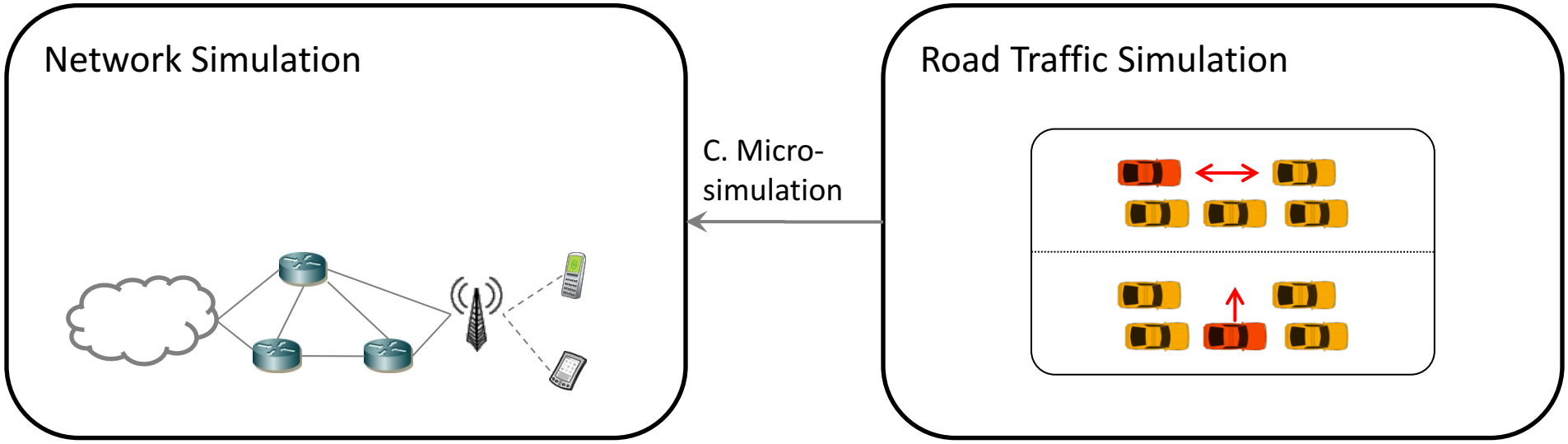


[1] V. Naumov, R. Baumann, and T. Gross, "**An evaluation of inter-vehicle ad hoc networks based on realistic vehicular traces**," Proceedings of 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2006), Florence, Italy, March 2006, pp. 108-119

[2] M. Fiore, J. Härri, F. Filali, and C. Bonnet, "**Vehicular Mobility Simulation for VANETs**," Proceedings of 40th Annual Simulation Symposium (ANSS 2007), March 2007, pp. 301-309

[3] H-Y. Huang, P-E. Luo, M. Li, D. Li, X. Li, W. Shu, and M-Y. Wu, "**Performance Evaluation of SUVnet With Real-Time Traffic Data**," IEEE Transactions on Vehicular Technology, vol. 56 (6), pp. 3381-3396, November 2007

- Replay artificial trace data
  - Microsimulation of road traffic
  - Pre-computation or live simulation
  - Problem: how to investigate traffic information systems (TIS)?

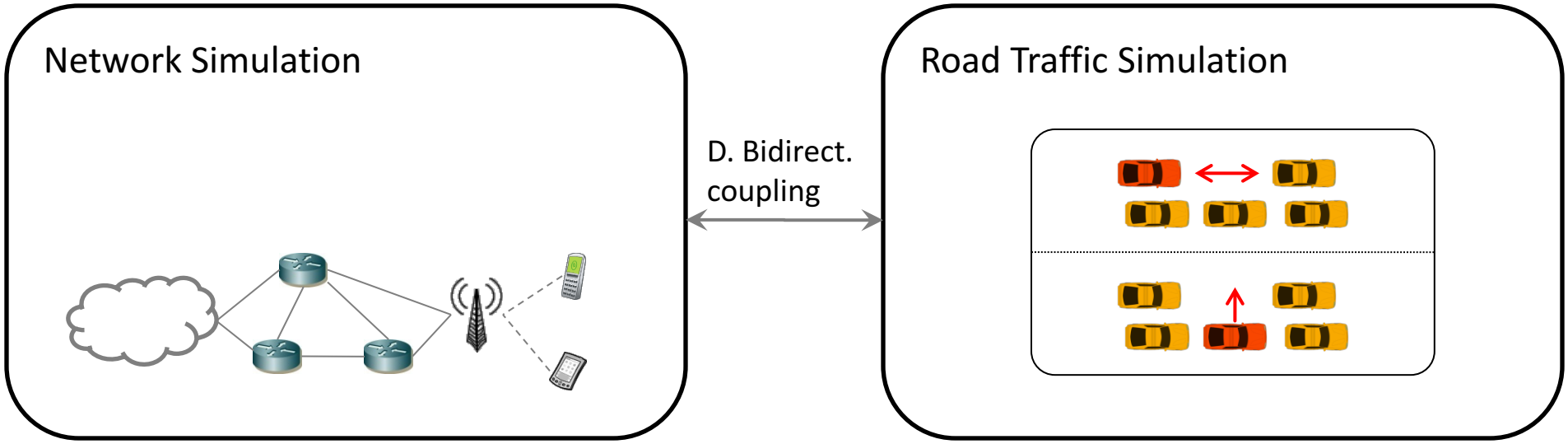


[1] C. Sommer, I. Dietrich, and F. Dressler, "**Realistic Simulation of Network Protocols in VANET Scenarios**," Proceedings of 26th IEEE Conference on Computer Communications (INFOCOM 2007): IEEE Workshop on Mobile Networking for Vehicular Environments (MOVE 2007), Poster Session, Anchorage, AK, May 2007, pp. 139-143

[2] B. Raney, A. Voellmy, N. Cetin, M. Vrtic, and K. Nagel, "**Towards a Microscopic Traffic Simulation of All of Switzerland**," Proceedings of International Conference on Computational Science (ICCS 2002), Amsterdam, The Netherlands, April 2002, pp. 371-380

[3] M. Treiber, A. Hennecke, and D. Helbing, "**Congested Traffic States in Empirical Observations and Microscopic Simulations**," Physical Review E, vol. 62, pp. 1805, 2000

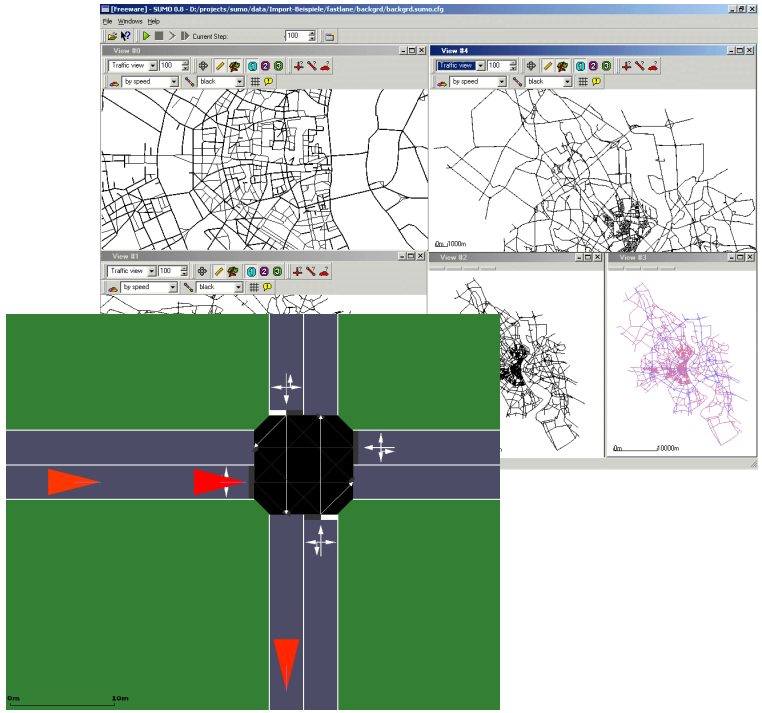
- Bidirectional coupling
  - Network traffic can influence road traffic



[1] C. Sommer, Z. Yao, R. German, and F. Dressler, "**On the Need for Bidirectional Coupling of Road Traffic Microsimulation and Network Simulation**," Proceedings of 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc 2008): 1st ACM International Workshop on Mobility Models for Networking Research (MobilityModels 2008), Hong Kong, China, May 2008, pp. 41-48

[2] C. Sommer, R. German, and F. Dressler, "**Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis**," IEEE Transactions on Mobile Computing, 2010. (to appear)

- Road traffic microsimulation
  - Ex.: SUMO – Simulation of Urban Mobility
  - Time discrete microsimulation
  - Car following models (Krauss, IDM)
  - Lane change models
  - Road topology
    - Speed limits
    - Traffic lights
    - Access restrictions
    - Turn restrictions
    - ...



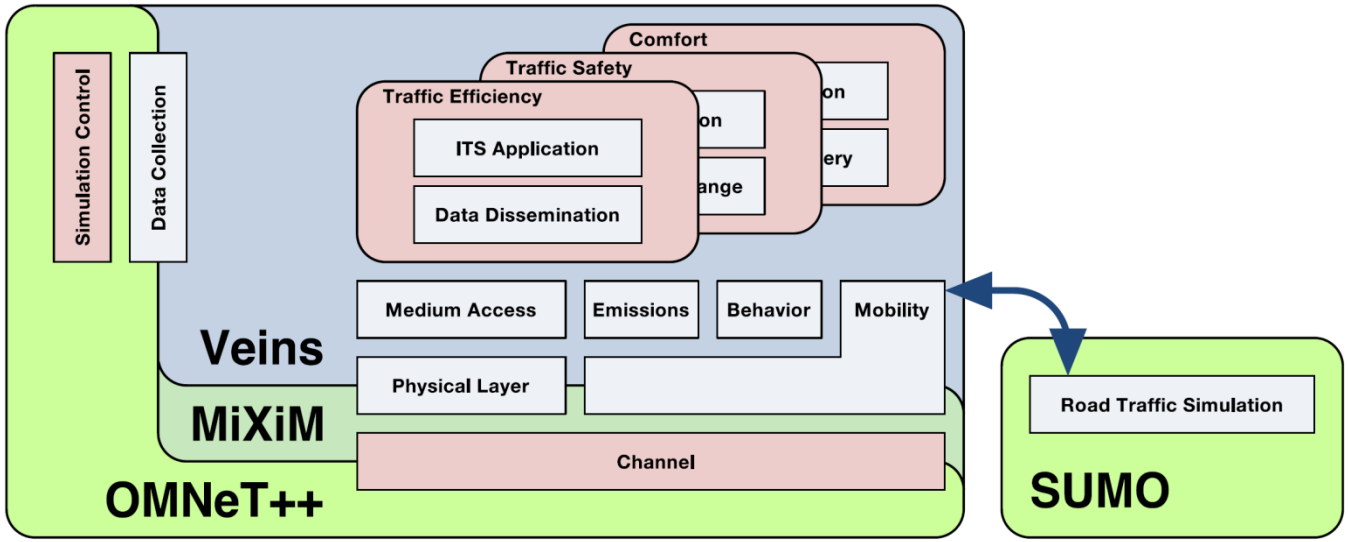
[1] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner, "**SUMO (Simulation of Urban MObility); An open-source traffic simulation**," Proceedings of 4th Middle East Symposium on Simulation and Modelling (MESM2002), Sharjah, United Arab Emirates, September 2002, pp. 183-187

Veins

UNIVERSITY OF TRENTO
Department of Information Engineering and Computer Science

Vehicles in Network Simulation

http://veins.car2x.org



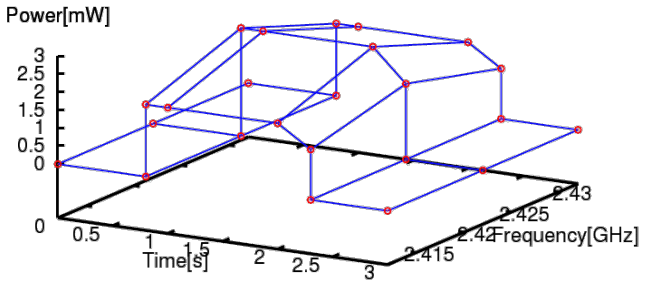
[1] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," IEEE Transactions on Mobile Computing, vol. 10, no. 1.

[2] C. Sommer, Z. Yao, R. German, and F. Dressler, "Simulating the Influence of IVC on Road Traffic using Bidirectionally Coupled Simulators," in 27th IEEE Conference on Computer Communications (INFOCOM 2008), Phoenix, AZ: IEEE, April 2008.

Veins

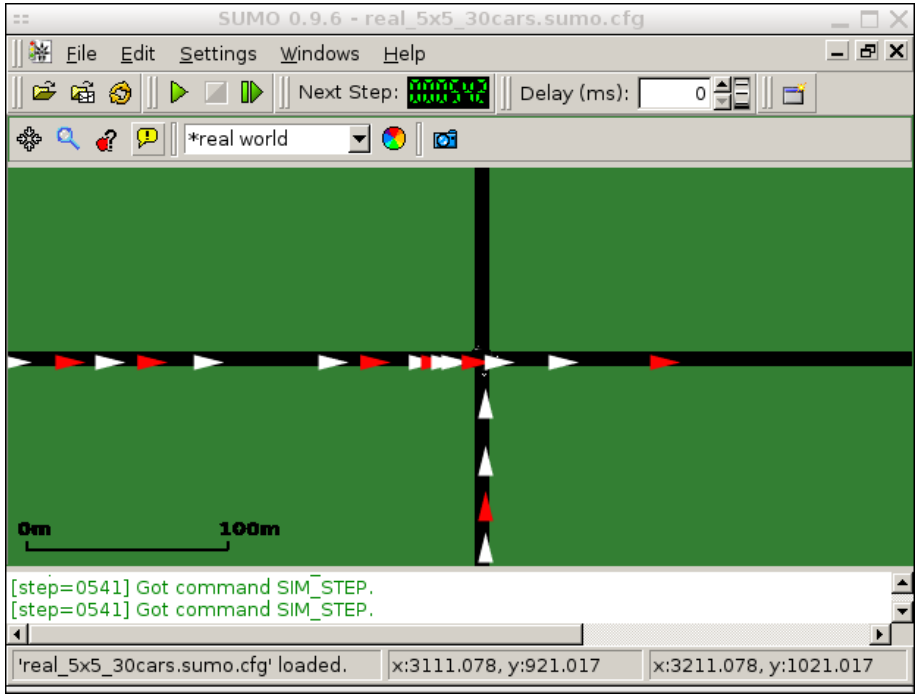
- OMNeT++
  - Discrete-Event Simulation (DES) kernel
    - Simulate model's reaction to queue of events
  - Main use case: network simulation
    - e.g., MANETs, Sensor nodes

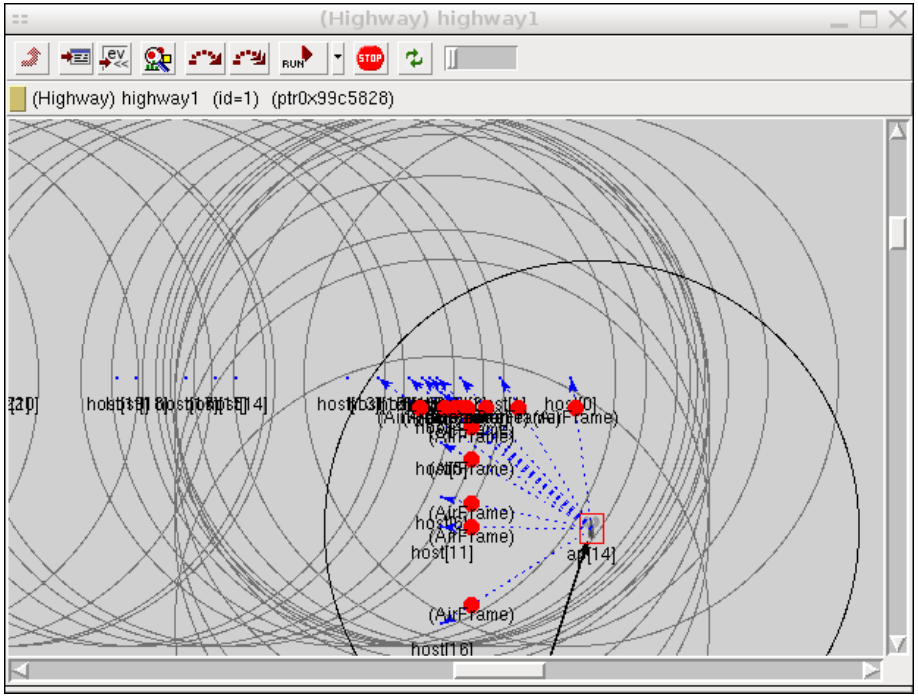
- (Fork of) MiXiM
  - Model library for OMNeT++ for PHY layer and mobility support
  - Event scheduling
  - Signal propagation
  - SINR / bit error calculation
  - Radio switching
  - …



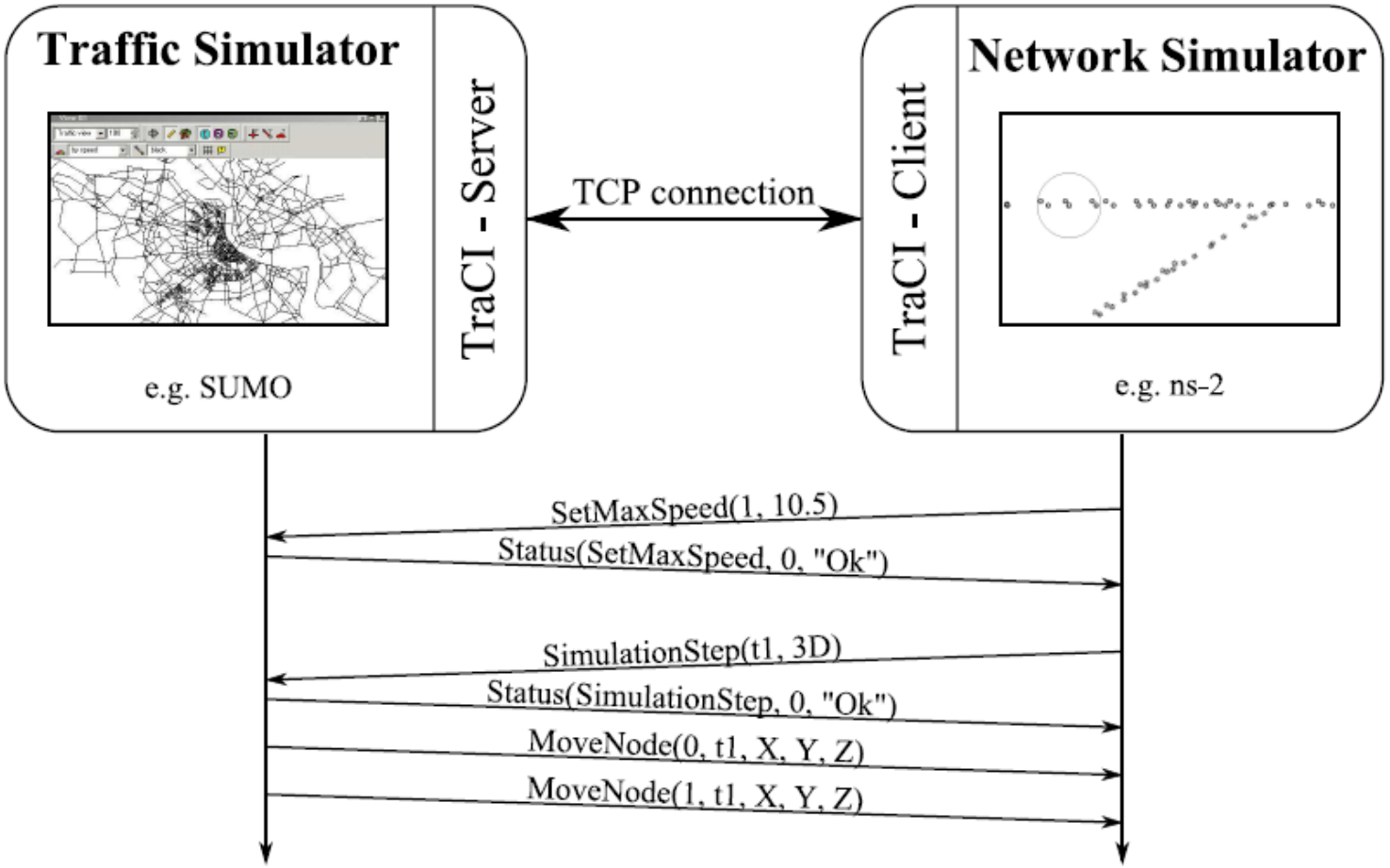
[1] A. Varga, "**The OMNeT++ Discrete Event Simulation System**," Proceedings of European Simulation Multiconference (ESM 2001), Prague, Czech Republic, June 2001

- Coupling OMNeT++ and SUMO
  - Synchronize time steps
  - Exchange commands and status information

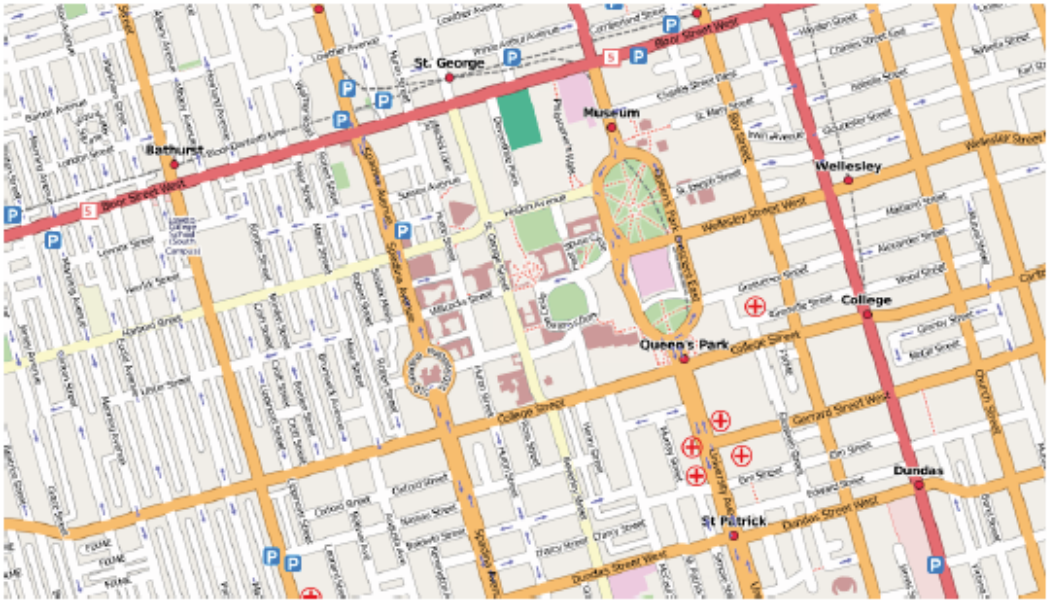


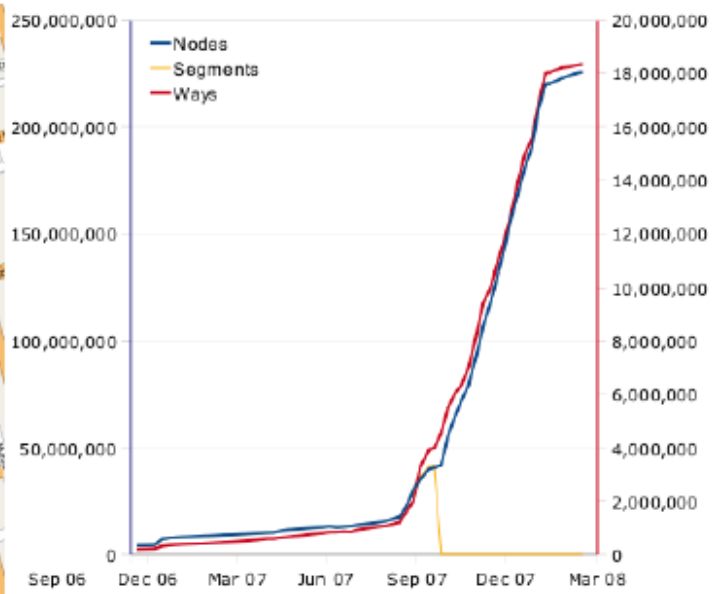


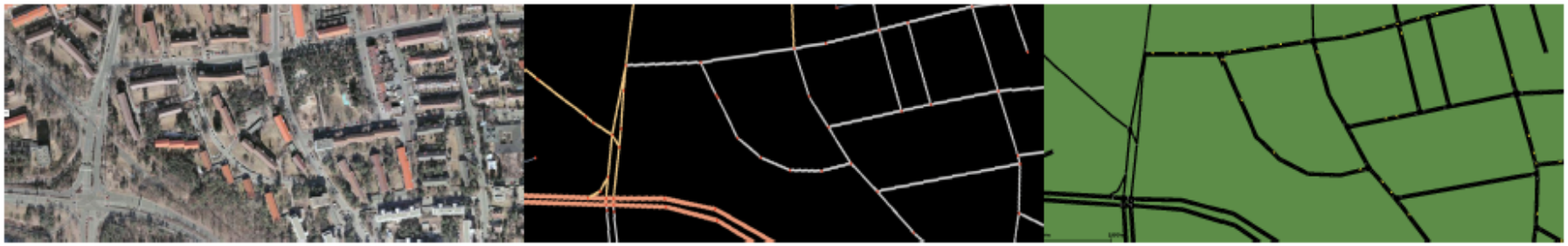
- TraCI: Message Sequence Chart



- Freely available road topology information
  - Geodatabase of OpenStreetMap project







A little bit of background

DISCRETE EVENT SIMULATION

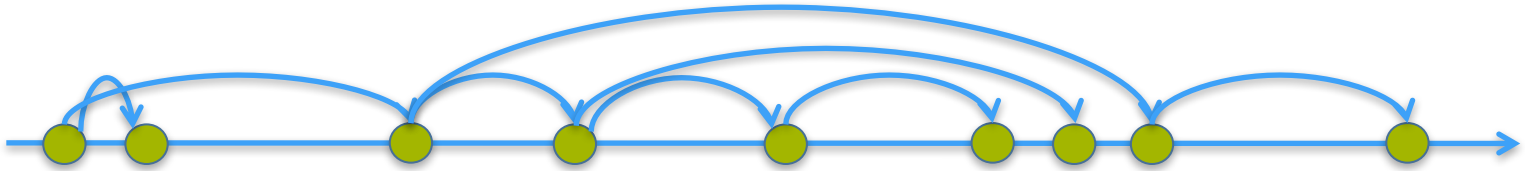
What is a discrete simulation?

- Simulation: reproducing the behavior of a real-world system
  - mathematical
    - $a(t) = a0$
    - $v(t) = a0 * t + v0$
    - $x(t) = a0/2 * t^2 + v0 * t + x0$
  - numerical
    - $a[k] = a0$
    - $v[k] = v[k-1] + a0 * dT$, with $v[0] = v0$
    - $x[k] = x[k-1] + (v[k] + v[k-1])/2 * dT$, with $x[0] = x0$

What is a discrete simulation?

- Discrete simulation: simulation "exists" only in specific time moments
  - time driven: sampled with a certain frequency (e.g., 10 Hz)

  - event driven: evolution by the generation and the consumption of events



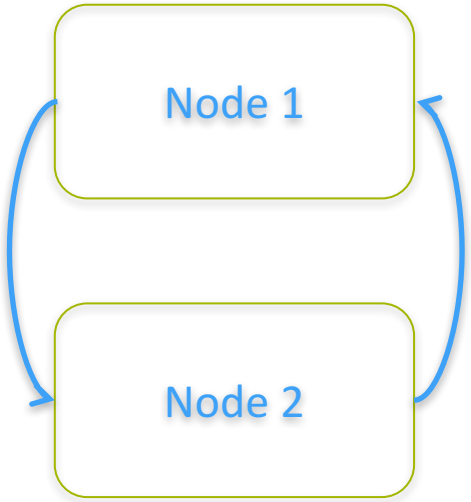
Discrete Event Simulation: An example

- Very easy example: two nodes communication

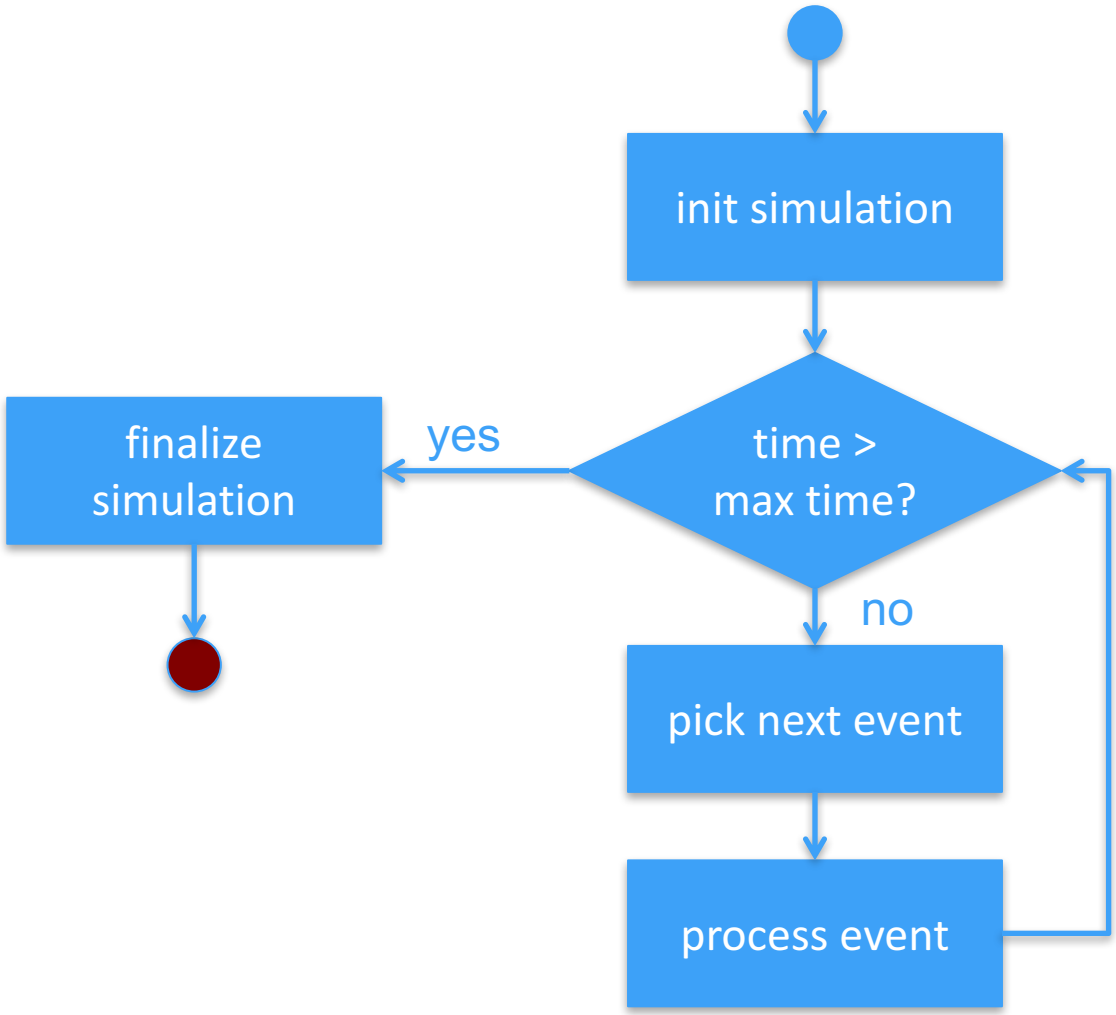
```
on init:
  scheduleEvent(sendMsg, now + exp(1))
  messageCount = 0

on event(event):
  if (event == sendMsg) {
    send(packet)
    scheduleEvent(sendMsg, now + exp(1))
  } else {
    if (random() > 0.5)
      messageCount++
  }

on finish:
  saveToFile(messageCount)
```



DES – A generic view



- Be careful: philosophy change needed
  - EVERYTHING is an event
    - schedule events
    - handle events
  - events are atomic
    - no duration

WRONG!

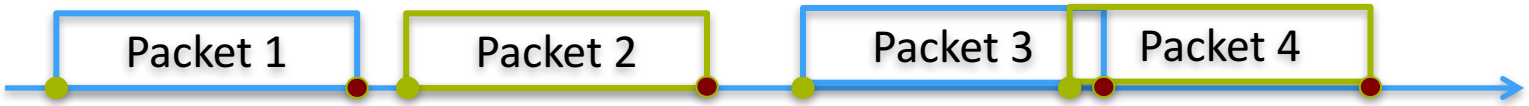
```
onStartRx:
  beginRx = now
  wait(endOfTransmission)
  rxDuration = now - beginRx
```

CORRECT!

```
onStartRx:
  beginRx = now

onEndRx:
  rxDuration = now - beginRx
```

- Consider again a network simulation
  - managing collisions: when two packets overlap, they both can't be received




```
onInit:
  state = IDLE
  recvPackets = {}
```

```
onStartRx(packet):
  recvPackets.add(packet)
  if (state == IDLE)
    state = RX
  else
    for p in recvPackets
      p.setLost()
```

```
onEndRx(packet):
  if (not packet.isLost())
    sendUp(packet)
  recvPackets.del(packet)
  if (|recvPackets| == 0)
    state = IDLE
```

The core network simulator

OMNET++

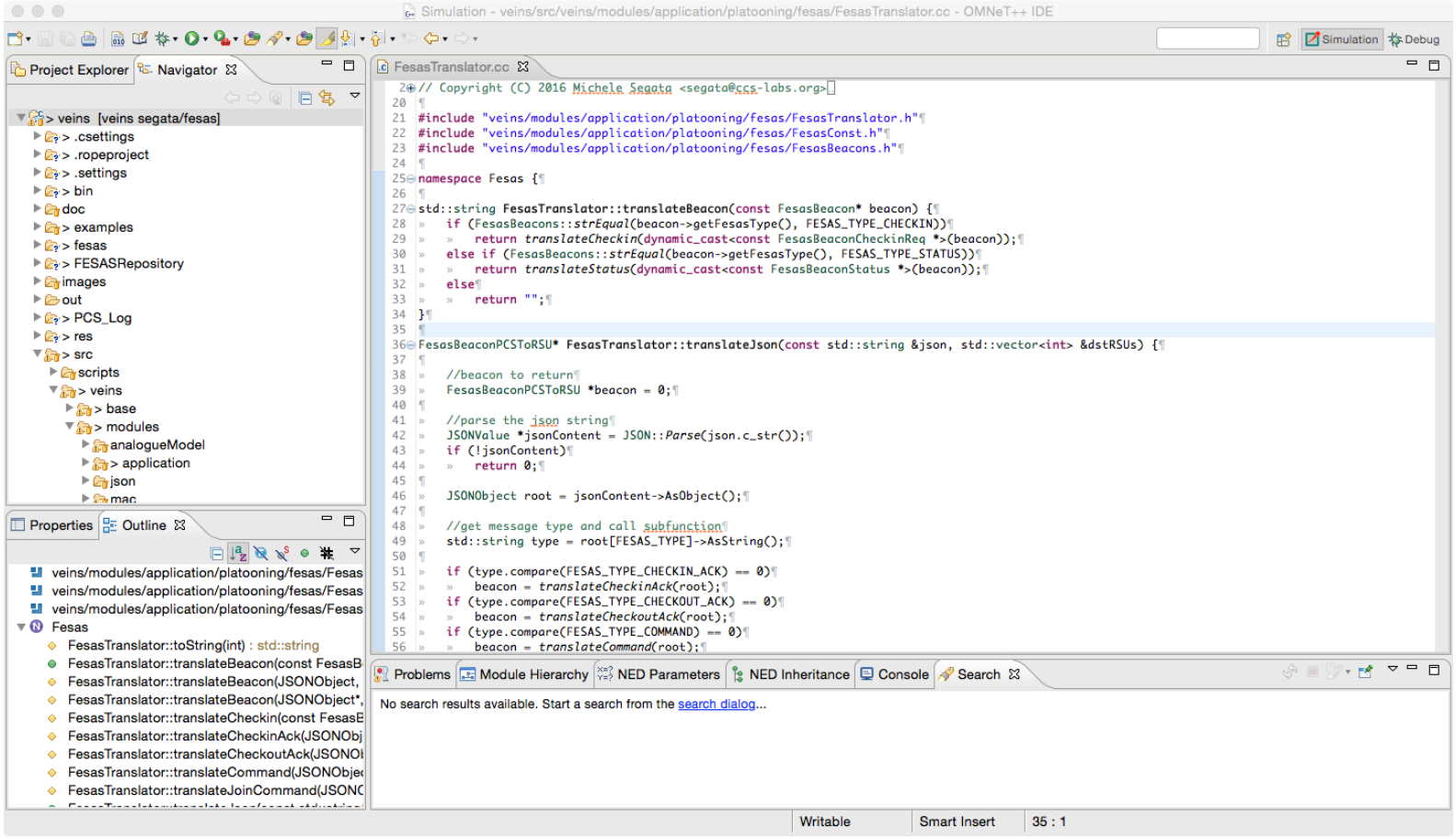
What is OMNeT++?

- Discrete Event Simulator
  - mainly used for network simulations
  - free for academic usage
  - multi-platform and open source
  - provides IDE for programming (if needed)
  - provides GUI for simulations (if needed)

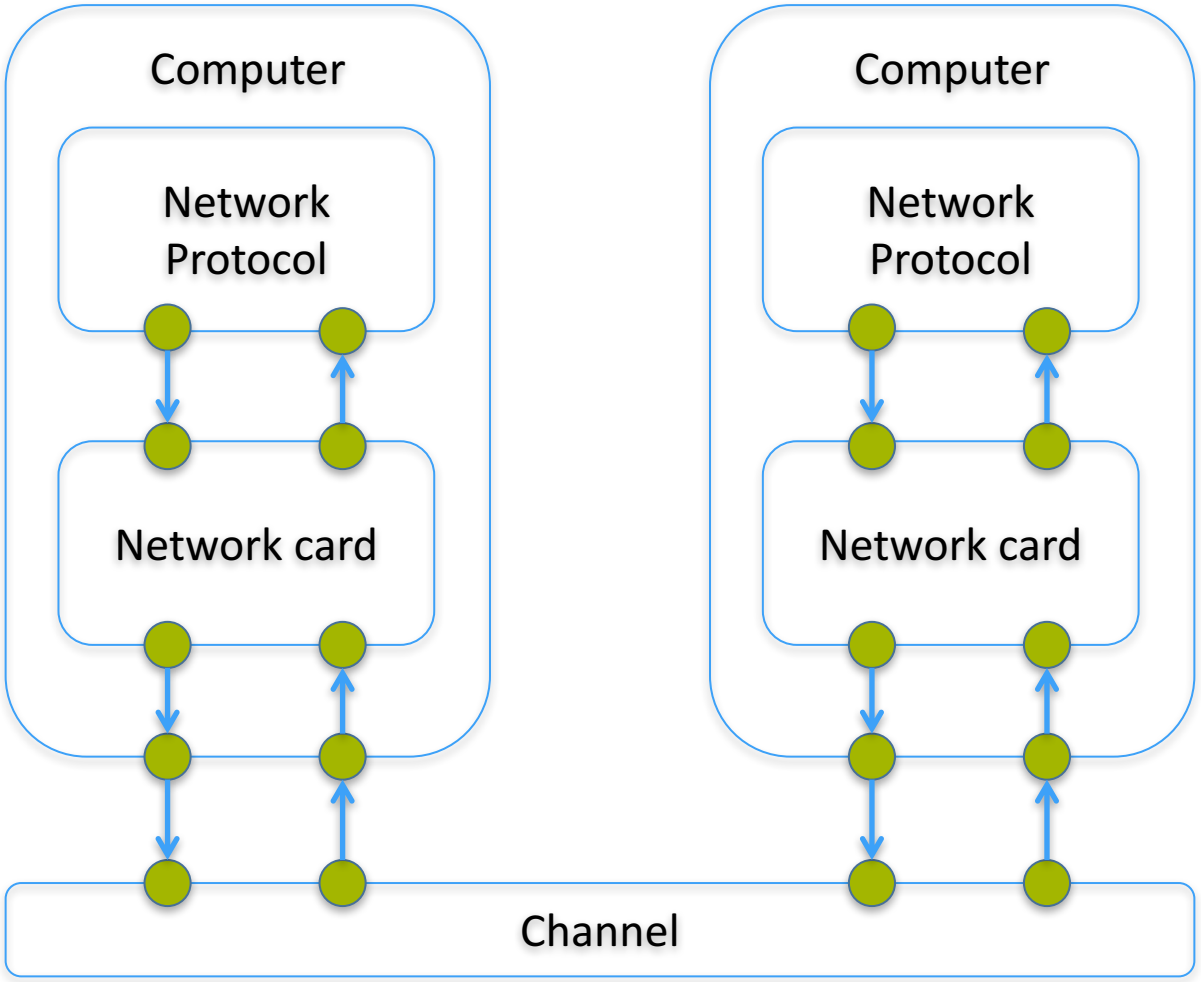
Programming OMNeT++

- Programming OMNeT++ means creating modules:
  - NED: textual files defining parameters and gates of the module
  - C++: source file defining the behavior of the module
  - example: Protocol.ned, Protocol.h, Protocol.cc
  - in addition: .msg (message files)
- Modules interconnect through gates
- You can define compound modules that do not require a C++ implementation
- You can implement C++ classes with no NED
- NED files support inheritance

- Good for coding, but NEVER build source code from there or run them, except for debugging



Module example



NED syntax example

```
package org.car2x.veins.modules.application.platooning;

import org.car2x.veins.modules.application.ieee80211p.BaseWaveApplLayer;

simple UnicastProtocol extends BaseWaveApplLayer
{
    parameters:
        //maximum queue size. set to 0 for infinite queue
        int queueSize = default(0);
        //maximum number of attempts
        int maxAttempts = default(16);
        //ack timeout
        double ackTimeout @unit(s) = default(1ms);
        //packet loss rate (between 0 and 1)
        double packetLossRate = default(0);

        @class(UnicastProtocol);
        @display("i=msg/mail");

    gates:
        input upperControlIn;
        output upperControlOut;
        input upperLayerIn;
        output upperLayerOut;
}
```

Compound module example

```
import org.car2x.veins.base.modules.IBaseApplLayer;
[…]
import org.car2x.veins.modules.nic.Nic80211p;

module Car {
    parameters:
        string scenario_type;
        string helper_type;
        string appl_type;
        string protocol_type;
    gates:
        input radioIn; // gate for sendDirect
    submodules:
        […]
        appl: <appl_type> like BaseApp {}
        prot: <protocol_type> like BaseProtocol {}
        unicast: UnicastProtocol {}
        nic: Nic80211p {}
        mobility: TraCIMobility {}
    connections allowunconnected:
        unicast.upperControlIn <-- prot.lowerControlOut;
        unicast.upperControlOut --> prot.lowerControlIn;
        unicast.upperLayerIn <-- prot.lowerLayerOut;
        unicast.upperLayerOut --> prot.lowerLayerIn;
        nic.upperLayerIn <-- unicast.lowerLayerOut;
        nic.upperLayerOut --> unicast.lowerLayerIn;

        radioIn --> nic.radioIn;
}
```

OMNeT++ messages

- Defined in .msg files

```
message MyMessage {
  int sourceId;
  string information;
}
```

- Automatically transformed into C++ code

```
class MyMessage : public ::cPacket {
  protected:
    int sourceId_var;
    opp_string information_var;
  public:
    virtual int getSourceId() const;
    virtual void setSourceId(int sourceId);
[…]
```

Recording data

- In OMNeT++ you can record data for statistics
  - vector: time sequence of values recorded multiple times
    - cOutVector speed;
    - speed.setName(“speed”);
    - speed.record(<value>);
  - scalars: single value usually recorded in the finish() method
    - recordScalar(“avgSpeed”, <value>);
- To extract data from vectors please see the scripts at
  - https://github.com/michele-segata/plexe-data-extraction

C/C++ module implementation

- Defines the following methods:
  - initialize(int stage)
  - finish()
  - handleMessage(cMessage *msg)
- Uses the following methods:
  - simTime()
  - scheduleAt(SimTime t, cMessage *msg)
  - send(cMessage *msg, int gateId)
  - cancelEvent(cMessage *msg)
  - par(const char* parName)
- Many more things…
  - refer to OMNeT++ documentation

Runtime parameters: omnetpp.ini

```
*.node[*].nic.mac1609_4.txPower = 100mW
*.node[*].nic.mac1609_4.bitrate = 6Mbps

*.node[*].nic.phy80211p.useThermalNoise = true
*.node[*].nic.phy80211p.thermalNoise = -95dBm

[Config ConfigName]
extends = OtherConfig

repeat = 10

*.node[*].module.param1 = ${p1 = 1,2}
*.node[*].module.param2 = ${p2 = 3,4}
*.node[*].module.param3 = ${p3 = 7,8 ! p1}

output-vector-file = ${resultdir}/${configname}_${p1}_${p2}_${repetition}.vec
output-scalar-file = ${resultdir}/${configname}_${p1}_${p2}_${repetition}.sca
```

Example tutorial

- Download source code from course website
- Building the example

```
unzip tictoc.zip
cd tictoc
make -j <number of cores> MODE=<release,debug>
```

- Running the examples (live demo)

```
cd simulations
./run [-u Cmdenv] -c TxcSimulation -r 0
./run [-u Cmdenv] -c Txc2Simulation -r 0
```

Getting the code in the IDE

- Open the OMNeT++ IDE (type omnetpp in your terminal)
- Click on File -> Import… -> General -> Existing projects into workspace
- Choose the folder where the project is located
- Click Open!

- `./run –a`
  - Lists all the configurations with the number of runs
- `./run –x <CONFIGNAME> -g`
  - Lists all the runs for a particular config
  - For each run, it shows the parameters for that specific run

Example tutorial

- Content of the tutorial:
  - Simple tic-toc message exchange between two nodes
- Version 1:
  - Upon reception of a message, a response is immediately sent back
- Version 2:
  - Upon reception of a message, a node waits for some time before replying
- Features explored:
  - C/C++ and NED modules, with parameters and gates
  - TicToc.ned network
  - Scheduling events and sending messages through gates
  - omnetpp.ini configuration file with features

VEINS AND SUMO

Getting it all together

- OMNeT++:
  - a Discrete Event Simulator mainly used for network simulations
- SUMO:
  - a time-driven discrete simulator of vehicular mobility
- Veins:
  - a vehicular networking simulation framework
  - couples OMNeT++ and SUMO

Veins node to vehicle mapping

1. A vehicle is created in SUMO
   - via SUMO configuration
   - via Veins manual injection
2. Veins is notified
3. Veins creates an OMNeT++ module
   - module type and name defined inside omnetpp.ini or ned files
     - *.manager.moduleType = "Car”
     - *.manager.moduleName = "node”
   - possibility to map SUMO vehicle type to specific module
     - *.manager.moduleType = ”vtypeauto=Car vtypeother=OtherCar”
     - *.manager.moduleName = ”vtypeauto=node vtypeother=other”

Getting Veins

- Download Veins source code and compile it (after OMNeT++)

```
git clone https://github.com/sommer/veins.git
cd veins
./configure
make -j <number of cores> MODE=<release,debug>
```

- Install SUMO (version 0.30.0)

```
sudo add-apt-repository ppa:sumo/stable
sudo apt-get update
apt-get -s install sumo #check that this would install 0.30.0
sudo apt-get install sumo
```

- Check that SUMO is working by typing:

```
sumo-gui
```

The Veins Example

- In one terminal start the launch daemon (we'll see how to avoid this)

```
cd veins
./sumo-launchd.py -c sumo-gui
```

- In a second terminal launch the example simulation

```
cd veins/examples/veins
./run -u Cmdenv -c WithoutChannelSwitching -r 0
```

- The example simulates vehicles which
  - When stopping for more than 10 s, send a "Traffic congestion" message
  - Upon receiving such message a vehicle
    - Re-sends it to other vehicles
    - Chooses another route to its destination

Core Components/Files

- `examples/veins` folder:
  - Configuration files: omnetpp.ini (next slide), SUMO config, etc.
- `src/veins/modules/mobility/traci/TraciScenarioManagerLaunchd.ned`
  - Default values for moduleType and moduleName
    - `string moduleType = default("org.car2x.veins.nodes.Car");`
    - `string moduleName = default("node");`
- `src/veins/nodes/Car.ned`
  - Definition of a “car” node in OMNeT++
    - Compound module: application, NIC card, mobility module

omnetpp.ini

```
[General]
[…]
network = RSUExampleScenario
[…]
*.manager.updateInterval = 1s
*.manager.host = "localhost"
*.manager.port = 9999
*.manager.launchConfig = xmldoc("erlangen.launchd.xml")
[…]
*.rsu[0].mobility.x = 2000
*.rsu[0].mobility.y = 2000
*.rsu[0].mobility.z = 3
*.rsu[*].applType = "TraCIDemoRSU11p"
[…]
```

Definition of the simulation network, which statically creates an 802.11p base station.

Nodes position update interval, and launch daemon parameters. The "forker" is a more handy way of launching the simulations.

Change `src/veins/nodes/Scenario.ned`:

- `import org.car2x.veins.modules.mobility.traci.TraCIScenarioManagerForker;`
- `manager: TraCIScenarioManagerForker {`

Change omnetpp.ini configuration:

- `*.manager.configFile = "erlangen.sumo.cfg"`
- `*.manager.commandLine = "sumo-gui --remote-port $port --seed $seed --configuration-file $configFile"`

Sets the position of the RSU and which application it runs. You have to search for the `TraCIDemoRSU11p.{ned,h,cc}` files

omnetpp.ini

```
[…]
*.**.nic.mac1609_4.txPower = 20mW
*.**.nic.mac1609_4.bitrate = 6Mbps
*.**.nic.phy80211p.sensitivity = -89dBm
*.**.nic.phy80211p.useThermalNoise = true
*.**.nic.phy80211p.thermalNoise = -110dBm
[…]
*.node[*].applType = "TraCIDemo11p"
[…]
[Config WithoutChannelSwitching]
```

Sets MAC/PHY layer parameters for ALL nodes, both RSU and cars. Tricky question: Why?

Sets the application run by car nodes. You have to search for the `TraCIDemo11p.{ned,h,cc}` files

Create a configuration named `WithoutChannelSwitching`. If no `extends` is specified, the configuration inherits from `General`

TraCIDemo11p Source Code

```
#include "veins/modules/application/traci/TraCIDemo11p.h"

Define_Module(TraCIDemo11p);

void TraCIDemo11p::initialize(int stage) {
    BaseWaveApplLayer::initialize(stage);
    if (stage == 0) {
        sentMessage = false;
        lastDroveAt = simTime();
        currentSubscribedServiceId = -1;
    }
}
```

TraCIDemo11p Source Code

```
void TraCIDemo11p::handlePositionUpdate(cObject* obj) {
    BaseWaveApplLayer::handlePositionUpdate(obj);

    // stopped for for at least 10s?
    if (mobility->getSpeed() < 1) {
        if (simTime() - lastDroveAt >= 10 && sentMessage == false) {
            sentMessage = true;

            WaveShortMessage* wsm = new WaveShortMessage();
            populateWSM(wsm);
            wsm->setWsmData(mobility->getRoadId().c_str());

            //host is standing still due to crash
            if (dataOnSch) {
                startService(Channels::SCH2, 42, "Traffic Information Service");
                //started service and server advertising, schedule message to self to send later
                scheduleAt(computeAsynchronousSendingTime(1,type_SCH),wsm);
            }
            else {
                //send right away on CCH, because channel switching is disabled
                sendDown(wsm);
            }
        }
    }
    else {
        lastDroveAt = simTime();
    }
}
```

TraCIDemo11p Source Code

```
void TraCIDemo11p::onWSA(WaveServiceAdvertisment* wsa) {
    if (currentSubscribedServiceId == -1) {
        mac->changeServiceChannel(wsa->getTargetChannel());
        currentSubscribedServiceId = wsa->getPsid();
        if  (currentOfferedServiceId != wsa->getPsid()) {
            stopService();
            startService((Channels::ChannelNumber) wsa->getTargetChannel(), wsa->getPsid(), "Mirrored Traffic
Service");
        }
    }
}
void TraCIDemo11p::onWSM(WaveShortMessage* wsm) {
    findHost()->getDisplayString().updateWith("r=16,green");

    if (mobility->getRoadId()[0] != ':') traciVehicle->changeRoute(wsm->getWsmData(), 9999);
    if (!sentMessage) {
        sentMessage = true;
        //repeat the received traffic update once in 2 seconds plus some random delay
        wsm->setSenderAddress(myId);
        wsm->setSerial(3);
        scheduleAt(simTime() + 2 + uniform(0.01,0.2), wsm->dup());
    }
}
```

TraCIDemo11p Source Code

```
void TraCIDemo11p::handleSelfMsg(cMessage* msg) {
    if (WaveShortMessage* wsm = dynamic_cast<WaveShortMessage*>(msg)) {
        //send this message on the service channel until the counter is 3 or higher.
        //this code only runs when channel switching is enabled
        sendDown(wsm->dup());
        wsm->setSerial(wsm->getSerial() +1);
        if (wsm->getSerial() >= 3) { //stop service advertisements
            stopService();
            delete(wsm);
        }
        else {
            scheduleAt(simTime()+1, wsm);
        }
    }
    else {
        BaseWaveApplLayer::handleSelfMsg(msg);
    }
}
```