- *collision free vehicle movement;*
- *several vehicle types;*
- *simulation of traffic lights;*
- *junction-based right-of-way rules* (junctions with streets having equal / different priorities, e.g. right-before-left);
- *lane changing;*
- *lane-to-lane connections;*

MOVE (46) is an extension to SUMO that adds a GUI for describing maps and defining vehicle movement and allows the user to import Google Earth maps. MOVE also includes a visualization tool that allows users to view the generated mobility trace. Basically, MOVE is composed of two components: the road map editor and the vehicle movement editor. The former serves to manually and randomly generate a road map, either from TIGER/line files or Google earth files, whereas the latter allows to specify the properties of each vehicle, like the maximum speed, the acceleration, the probability of turning at crossroads, the path to take etc. The information collected by the two editors are sent to the SUMO compiler, then a trace file in ns-2 or Qualnet format is generated. MOVE has been compared by simulation to RWP using AODV. The results show that MOVE causes low reception rate.

4.3 Network simulators

In this section we describe the most used network simulators in the research field. These applications allow to simulate (at different level of detail) the ISO-OSI layers, taking into account the propagation and fading effect of the radio signals.
As described earlier, a mobility simulator is generally used to produce node movement traces that are then fed to the network simulator. Th e network simulator then controls the communications between the mobile nodes. As these network simulators support wireless communication, most of them include at least a simple node mobility model,

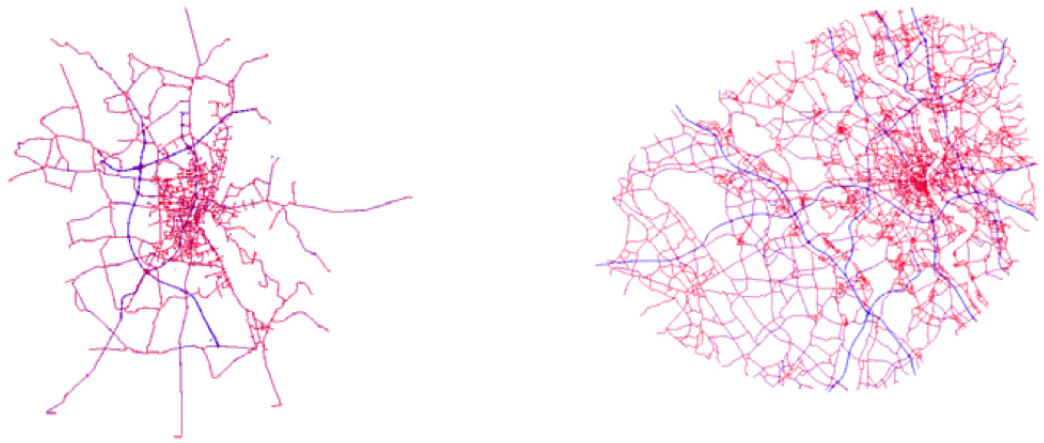

Figure 4.8: SUMO simulated network example. Two example networks as simulated by SUMO; left: the city of Magdeburg from the INVENT-project; right: the area around the city of Cologne.

which includes the following models: *Random Drunken Model*[1], *Random Waypoint Model*, *Trace file*[2].

4.3.1 OPNET

OPNET[3] is a commercial network simulator used for simulations of both wired and wireless networks. It supports a wide range of wireless technologies such as MANETs, IEEE 802.11 wireless LANs, WiMAX, Bluetooth, and satellite networks. OPNET provides a graphical editor interface to build models for various network entities from physical layer modulator to application processes, and includes graphical packages and libraries for presenting simulation scenarios and results. There are three basic phases of the OPNET deployment process. First, choose and configure node models to use in simulations, such as a wireless node, trajectory, and so on. Second, build and organize the network by setting up connections for different entities. Third, select the desired statistics (local or global) to collect during the simulation.
In order to setup a simulation, there are three main configuration files: *network config-*

[1]At each intersection, the vehicle randomly selects from four directions or chooses to remain stationary.

[2]Vehicles' mobility is based on a trace file that can be imported into the simulator.

[3]Website: http://www.opnet.com

uration file (nodes, routers, applications, protocols, radio settings), *node configuration file* (data related to mobile nodes such as IP addresses) and *global parameter file* (simulation time, coordinate system, random seed, protocol stack, statistic filter).

4.3.2 GloMoSim and QualNET

GloMoSim (112) is a network simulator developed at UCLA in 1999 especially designed of MANET and includes a large set of routing protocols and several physical layer implementations. Being an integrated network simulator, it offers also the possibility to generate trace according to some mobility models like, i.e. Random Waypoint, Random Drunken (and, of course, real traces import).
The software was dismissed in 2000 but it is still possible to download only for educational purposes. On the other side, the commercial version of GloMoSim became Qualnet simulator, developed by Scalable Network Technologies.
Qualnet[1] is an extremely powerful and detailed network simulator which provides a large set of wireless physical and MAC layers models and mobility models, allowing reseearchers to develop and and easily simulate protocols for sensor networks, satellite and military warfare. Another important aspect of Qualnet, is the avaliability for Windows and Unix/Linux platforms.

QualNet delivers high fidelity simulations of network devices, transmitters, antennas, terrestrial characteristics, and human interactions, all at real time speed or greater thanks to an efficient parallel kernel and patent-pending, computationally-efficient code, making possible to get the same accurate simulations of wireless and wired networks for 50, 500, or 5,000 nodes. Models in source form provide developers with a solid library on which to build and experiment with new network functionality. The end result is accurate prediction of network performance for a diverse set of application requirements and uses. From wired LANs and WANs, to cellular, satellite, WLANs and mobile ad hoc networks, QualNet's library is extensive and because of its efficient kernel, QualNet models large scale networks with heavy traffic and mobility in reasonable simulation times.
Qualnet Simulator features can be summarized as the follows:

- *instant playback of simulation results to minimize unnecessary model executions;*

[1] http://www.scalable-networks.com

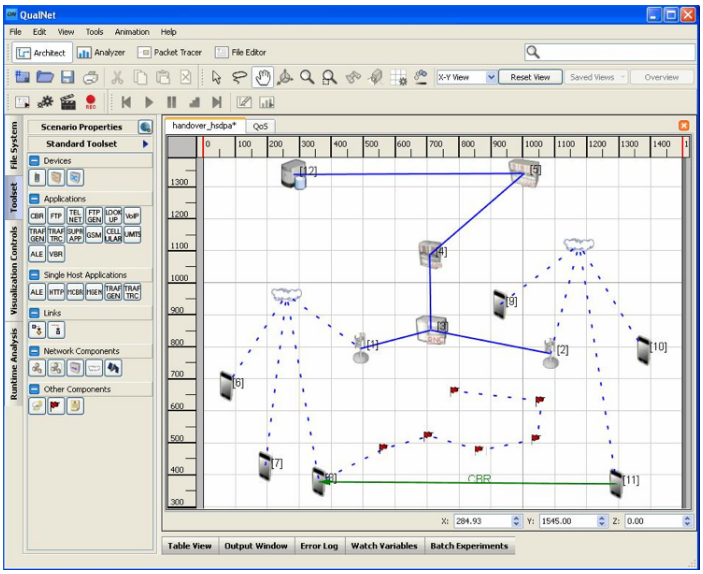

Figure 4.9: Qualnet simulator. Qualnet in visual mode.

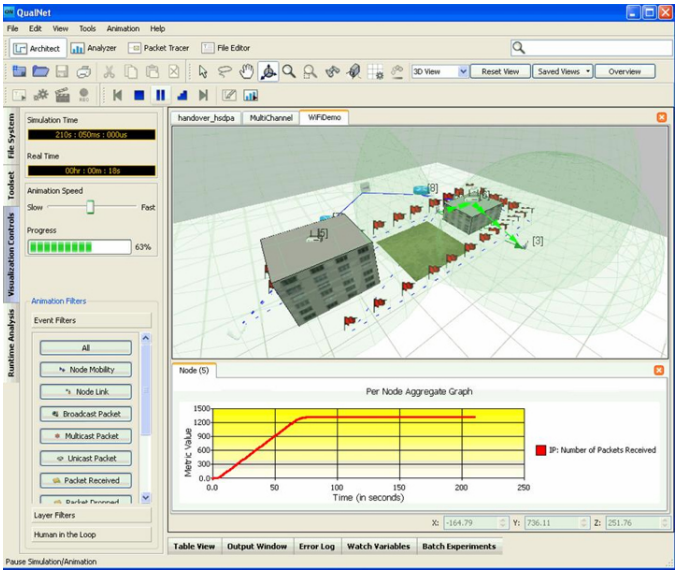

Figure 4.10: Qualnet simulator. Qualnet in visual mode, additional example.

- *fast simulation results for thorough exploration of model parameters;*
- *fast model set up with a powerful Graphical User Interface (GUI) for custom code development and reporting options;*
- *scalable up to tens of thousands of nodes;*
- *real-time simulation for man-in-the-loop and hardware-in-the-loop models;*
- *multi-platform support;*

and it consists of several modules which handle specific functions like:

- *Scenario Designer*
  It is a graphical tool that provides an intuitive model set up capability and is used to create and design experiments in QualNet. The Scenario Designer enables a user to define the geographical distribution, physical connections and the functional parameters of the network nodes, all using intuitive click and drag tools, and to define network layer protocols and traffic characteristics for each node.
- *Animator*
  This module is used to execute and animate experiments created in the Scenario Designer. Using the Animator a user can watch traffic flow through the network and create dynamic graphs of critical performance metrics as a simulation is running.
- *Packet Tracer*
  It is a packet-level visualization tool for viewing the contents of packets as they travel up and down the protocol stack.
- *Analyzer*
  Statistical graphing tool that displays network statistics generated from a QualNet experiment. Using the Analyzer, a user can view statistics as they are being generated, as well as compare results from different experiments.

Due to its powerful features and scalability, we decided to use Qualnet in our research on data dissemination with rateless codes in vehicular networks.

4.3.2.1 Discrete event simulation

QualNet is a discrete-event simulator. In discrete-event simulation, a system is modeled as it evolves over time by a representation in which the system state changes instantaneously when an event occurs, where an event is defined as an instantaneous occurrence that causes the system to change its state or to perform a specific action. Examples of events are: arrival of a packet, a periodic alarm informing a routing protocol to send out routing update to neighbors, and so on. Examples of actions to take when an event occurs are: sending a packet to an adjacent layer, updating state variables, starting or restarting a timer, etc.
In discrete-event simulation, the simulator maintains an event queue. Associated with each event is its event time, i.e., the time at which the event is set to occur. Events in the event queue are sorted by the event time. The simulator also maintains a simulation clock which is used to simulate time. The simulation clock is advanced in discrete steps, as explained below.
The simulator operates by continually repeating the following series of steps until the end of simulation:

1. the simulator removes the first event from the event queue, i.e., the event scheduled for the earliest time;

2. the simulator sets the simulation clock to the event time of the event. This may result in advancing the simulation clock;

3. the simulator handles the event, i.e., it executes the actions associated with the event. This may result in changing the system state, scheduling other events, or both. If other events are scheduled, they may be scheduled to occur at the current time or in the future.

4.3.2.2 Qualnet protocol stack

One of the best feature in Qualnet (very appreciated from researchers) is the protocol stack's implementation, a layered architecture similar to that of the TCP/IP network protocol stack. According to this architecture, data moves between adjacent layers (from top to bottom): Application, Transport, Network, Link (MAC) and Physical Layers. Adjacent layers in the protocol stack communicate via well-defined APIs, and

generally, layer communication occurs only between adjacent layers. For example, Transport Layer protocols can get and pass data to and from the Application and Network Layer protocols, but cannot do so with the Link (MAC) Layer protocols or the Physical Layer protocols. This rule concerning communication only between adjacent layers is not a limit of the simulator because it can be also circumvented by the programmer.

Each protocol operates at one of the layers of the stack. Protocols in QualNet essentially operate as a finite state machine. The occurrence of an event corresponds to a transition in the finite state machine. The interface between the layers is also event based. Each protocol can either create events that make it change its own state (or perform some event handling), or create events that are processed by another protocol. To pass data to, or request a service from, an adjacent layer, a protocol creates an event for that layer.

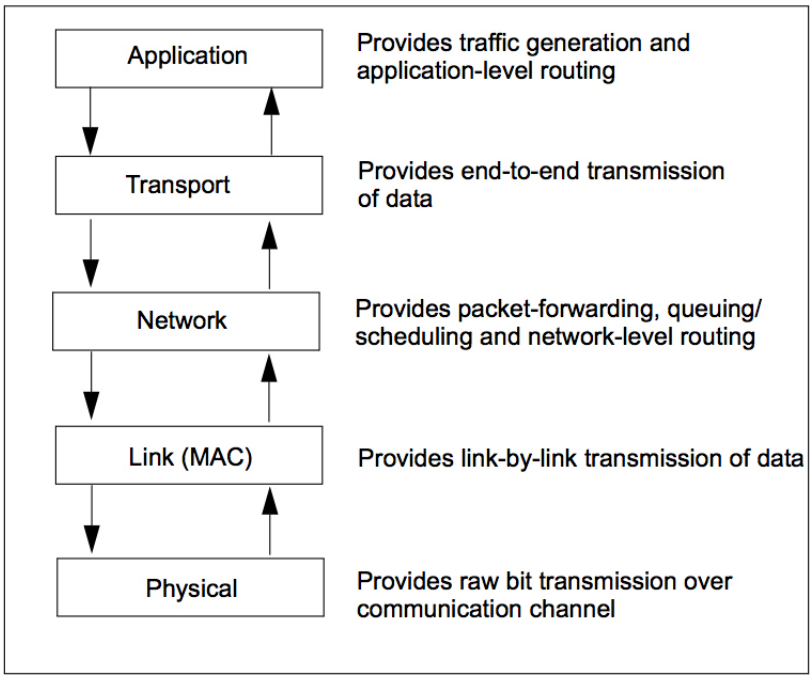

Figure 4.11: Qulanet protocol stack. This picture shows the simulator protocol stack and the general functionality of each layer.

- *Application Layer*

  The Application Layer is responsible for traffic generation and application level

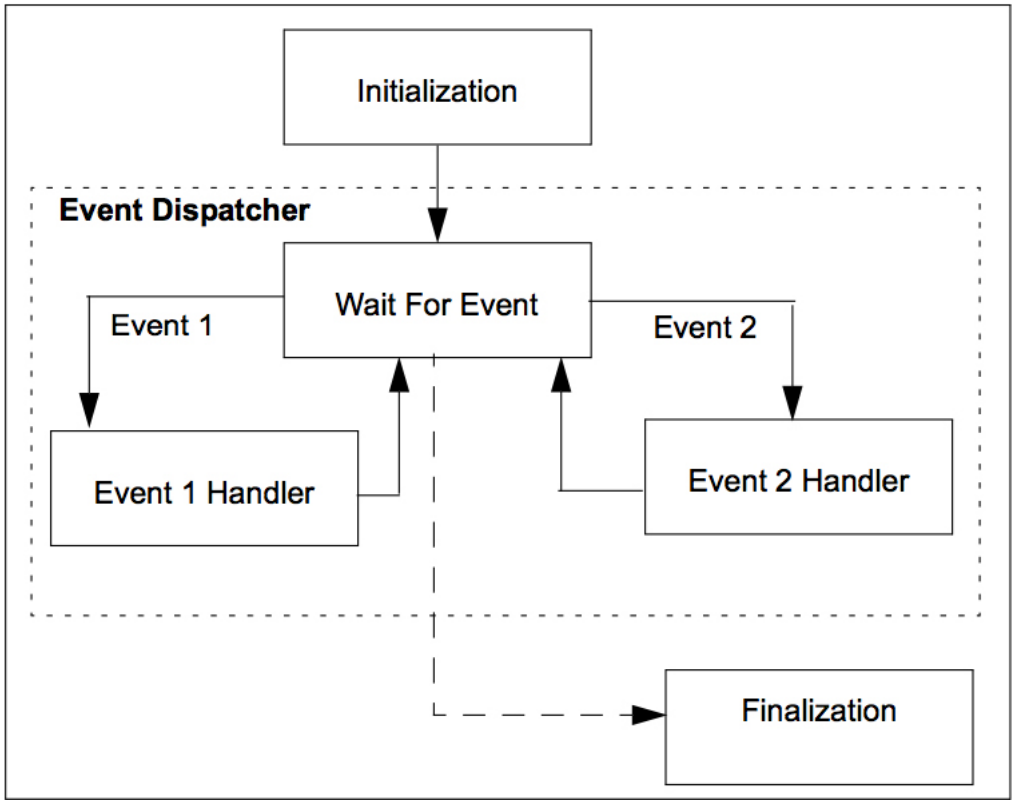

Figure 4.12: Qualnet protocol model. The picture shows the finite state machine representation of a protocol in QualNet. At the heart of a protocol model is an Event Dispatcher, which consists of a Wait For Event state and one or more Event Handler states. In the Wait For Event state, the protocol waits for an event to occur. When an event for the protocol occurs, the protocol transitions to the Event Handler state corresponding to that event (e.g., when Event 1 occurs, the protocol transitions to the Event 1 Handler state). In this Event Handler state, the protocol performs the actions corresponding to the event, and then returns to the Wait For Event state. Actions performed in the Event Handler state may include updating the protocol state, or scheduling other events, or both.

routing. Protocols written at the Application Layer rely on the Transport Layer to deliver application-level data from the source to the destination. Thus, Application Layer protocols pass data down to the Transport Layer at the source node, and receive data from the Transport Layer at the destination node. Examples of traffic- generating Application Layer protocols implemented in QualNet are Constant Bit Rate (CBR), FTP, and Telnet. Examples of Application Layer routing protocols implemented in QualNet are RIP, Bellman-Ford, and BGP.

- *Transport Layer*
  The Transport Layer provides end-to-end data transmission services to the Application Layer. Protocols written at the Transport Layer receive data from the Application Layer and rely on the Network Layer for data forwarding at the source node, and receive data from the Network Layer and pass data to the Application Layer at the destination node. Examples of Transport Layer protocols include UDP, TCP and RSVP-TE.

- *Network Layer*
  The Network Layer is responsible for data forwarding and queuing/scheduling. The Internet Protocol (IP) resides at this layer and is responsible for packet forwarding. At the source node, the Network Layer receives data from the Transport Layer and relies on the Link (MAC) Layer for link-by-link data delivery. At the destination node, the Network Layer receives data from the Link (MAC) Layer and passes the data up to the Transport Layer. The Network Layer also implements certain types of routing protocols. Examples of Network Layer routing protocols implemented in QualNet are AODV, DSR, OSPF, and DVMRP. Examples of queuing/scheduling protocols implemented in QualNet are FIFO, RED, RIO, WFQ, and WRR.

- *Link (MAC) Layer*
  The Link (MAC) Layer provides link-by-link transmission. At the sending side, the Link (MAC) Layer receives data from the Network Layer and passes the data to the Physical Layer for transmission over the wired or wireless channel. At the receiving side, the Link (MAC) Layer receives data from the Physical Layer and forwards the data up to the Network Layer. Examples of protocols at the Link

(MAC) Layer implemented in QualNet are point-to-point, IEEE 802.3, IEEE 802.11, and CSMA.

- *Physical Layer*[1]
  The Physical Layer is responsible for transmitting and receiving raw bits from the wired and wireless channel. At the source node, the Physical Layer receives data from the Link (MAC) Layer and sends the data to the Physical Layer of the destination node. At the destination node, the Physical Layer receives data from the Physical Layer of the source node and passes the data to the Link (MAC) Layer. Examples of Physical Layer protocols implemented in QualNet are wired point-to-point links, IEEE 802.3, and IEEE 802.11.

- *The Communication medium*
  The communication medium transmits signals between nodes. It interfaces with the Physical Layer entities at the nodes. A wireless communication medium model in QualNet simulates the propagation of signals between nodes, taking into account both propagation delays and signal attenuation due to path loss, fading, and shadowing.
  In QualNet, a communication medium model has three components: a path loss model, a fading model, and a shadowing model. Path loss models in QualNet include free space, two ray, and Irregular Terrain Model (ITM). QualNet implements the Ricean fading model. Rayleigh fading is a special case of Ricean fading. QualNet provides models for two shadowing models: constant and lognormal.

4.3.3 NS-2

NS-2 is an open-source discrete event network simulator that supports both wired and wireless networks, including many MANET routing protocols and an implementation of the IEEE 802.11 MAC layer and it is the most widely used simulator for academic networking research. There are implementations of several mobility models available for NS-2, including *Random Trip Mobility*(73) and *Semi-Markov Smooth Mobility*(114). NS-2 simulates the wireless physical layer and the important parameters that influence its behavior (e.g., channel fading).
The core of NS-2 is written in C++, but users interact with NS-2 by writing TCL

[1]For wired networks, the Physical Layer code is incorporated into the Link (MAC) Layer.

scripts. which should contain all of the commands needed to run the simulation (e.g., setting up the topology, specifying wireless parameters, and so on). As with QualNet, several of the mobility simulators can generate node descriptions and movement traces suitable for use in NS-2.
A typical wireless NS-2 simulation produces an event trace file and an animation trace file, used by the included utility nam to provide animation of the simulation. The event trace file includes packet enqueue (transmission), packet dequeue (forwarding), packet drops, and packet reception.

4.3.4 J-Sim

J-Sim is an open-source simulation environment, developed entirely in Java. J-Sim provides two mobility models: trajectory-based and random waypoint. J-Sim is presented as an alternative to ns-2, because it is designed to be easier to use. In J-Sim, applications are built as a set of components that can be designed and tested separately. J-Sim can take a TCL fi le as input, similar to ns-2, but with a diff erent format. Like ns-2, J-Sim produces an event trace file and an animation file, suitable for use in nam.

4.3.5 OMNeT++

OMNeT++[1] is an open-source simulation environment. The primary simulation applications are Internet simulations, mobility, and ad hoc simulations. OMNeT++ has a component-based design, meaning that new features and protocols can be supported through modules. OMNeT++ supports network and mobility models through the independently developed Mobility Framework and INET Framework modules. Simulation design in OMNeT++ is GUI-based, and output data can be plotted through the GUI as well. OMNEST[2] is the commercial version of OMNeT++, offered by Simulcraft , Inc.

4.3.6 SWANS

SWANS(13) (Scalable Wireless Ad hoc Network Simulator) was developed to be a scalable alternative to ns-2 for simulating wireless networks. Based on comparisons of SWANS, GloMoSim, and ns-2, (45) SWANS was determined to be the most scalable

[1]OMNET++ website: http://www.omnetpp.org/
[2]OMNEST website: http://www.omnest.com/

and the most efficient in memory usage with the fastest runtime. Along with better performance, SWANS delivered similar results as ns-2, at least for the network components that were implemented in both. The input for SWANS is a Java file that creates the nodes and specifies how these nodes should move (the node movement scenario) and how they should communicate (the communication scenario). The user can select any of the ready-made applications in SWANS and associate it with any node(s) to execute it at the node application layer. Also, SWANS gives the user the flexibility to build a custom application and execute it at the application layer of any node.

4.4 Tightly Integrated Simulators

As mentioned in 4.1, the simulation of VANET applications not only requires simulating the wireless communication between the vehicles, but also requires simulating the mobility of the vehicles. Unfortunately, these two aspects of VANET simulation have often been decoupled. Both vehicular mobility and wireless communication have large communities concerned with their modeling and simulation, so high-quality simulators exist in each of these areas, as discussed in the previous sections. The problem is how to merge the two types of simulators (network simulator and mobility simulator).
A simple method to achieve this goal is to implement mobility models in a network simulator, but without allowing the network messages to feed back to the mobility model. This kind of simulation is called one-way communication (from mobility model to network). These types of simulators are suitable for simulating infotainment-related VANET applications, including Internet connectivity, multimedia applications, and peer-to-peer applications, where the communication does not affect vehicles movements. In contrast, tightly integrated simulators, that offer two-way communication, usually consist of two sub-simulators (network and mobility) which can communicate with each other. These simulators are more appropriate for safety-related and traffic information applications that assume that feedback from the network will affect vehicles movements. In these types of applications, the traffic simulator feeds the network simulator with position information, speed, acceleration, direction, and so on. The VANET application that runs at the top level of the network simulator incorporates this information with surrounding vehicles information in order to notify the driver of upcoming congestion or a possible collision. Based on this notification, driving decisions (i.e., vehicle

mobility) may be affected. For example, in a congestion notification system, the driver may choose to change lanes or take a different path. These decisions need to propagate back to the mobility simulator to be reflected in the vehicle mobility information.

Usually, any simulator has an events queue to store the events that should be executed according to their scheduled execution time. In case of two-way communication simulators, each sub-simulator has its own events queue; these two events queues can be combined into one events queue, or they can be kept separate, which implies that extra overhead will be needed for synchronization. Based on this decision, the two-way communication simulators are separated into two categories: *those with a single events queue* and *those with two events queues*. Having a single events queue can be achieved through implementing one of the sub-simulators in the other. Often, the vehicular mobility sub-simulator is implemented in the network sub-simulator, as in ASH (7) or the two simulators can be highly integrated together, as in NCTUns (99). The advantages of having a single events queue are that the vehicles mobility events and the network events will be inserted in the same queue, which removes the burden of synchronizing the two types of events. In addition, the simulation will be more efficient from the execution time and memory consumption perspectives. The main disadvantage of having a single events queue is that the process of maintaining and extending such simulators is not easy. With two events queues, two-way communication is achieved through an interface that is implemented between the network sub-simulator and the mobility sub-simulator itself. The main function of that interface is to update each sub-simulator with the recent events in the other sub-simulator and, moreover, it synchronizes the event execution in each of the events queues. The main disadvantages are that these types of simulators consume more memory and execution time.

In this section, tightly integrated simulators with one-way communication will be discussed first. However, only those simulators that attempt to package the mobility and network simulator in a single program are included, rather than those that manually feed the mobility simulators movement trace to the network simulator. Next will come an overview of tightly integrated simulators with two-way communication and two events queues, and finally tightly integrated simulators with two-way communication and a single events queue.

4.4.1 SWANS++

SWANS++[1] extends the network simulator SWANS by adding a GUI to visualize the scenario and a mobility model, STRAW (21) for the vehicles movement in street scenarios. STRAW uses the simple random waypoint mobility model, but it restricts the vehicles movement to real street boundaries, loaded from TIGER/Line data files. STRAW consists mainly of three components: intrasegment mobility, intersegment mobility, and route management and execution. In intersegment mobility, the vehicles move according to a car-following model and change their speed only in certain situations:

1. when the vehicle arrives at an intersection and the next segment is full, the vehicle stops until the next segment has a free slot;

2. when the vehicle has a vehicle in front of it, the vehicle adjusts its speed accordingly in order to maintain a certain distance in between;

3. when the vehicle arrives at a traffic control or stop sign;

4. when the vehicle makes a turn.

For intersegment mobility, according to the system design, there is either a traffic control sign or a stop sign at each intersection that forces the vehicle to alter its speed. The mobility model implemented in STRAW (and therefore, SWANS++) does not support lane changing. The route management and execution (RME) module is responsible for determining the vehicles routes during the simulation. The RME has two techniques to fulfill its task. The first technique is simple intersegment mobility (simple STRAW) at which the vehicles next segment is determined stochastically, while the second technique is origindestination mobility (STRAW OD), in which the vehicles route is predetermined based on the shortest path between the origin and destination. SWANS++ is a tightly integrated simulator, but it does not provide feedback between the mobility and networking modules.

[1] SWANS++ is avaliable at http://www.aqualab.cs.northwestern.edu/projects/swans++/

4.4.2 GrooveNeT

GrooveNet[1] (63) (originally known as GrooveSim (64)) is an integrated network and mobility simulator that allows communication between real and simulated vehicles. Originally, GrooveNet extended the open-source simulator *RoadNav* [2] by adding a

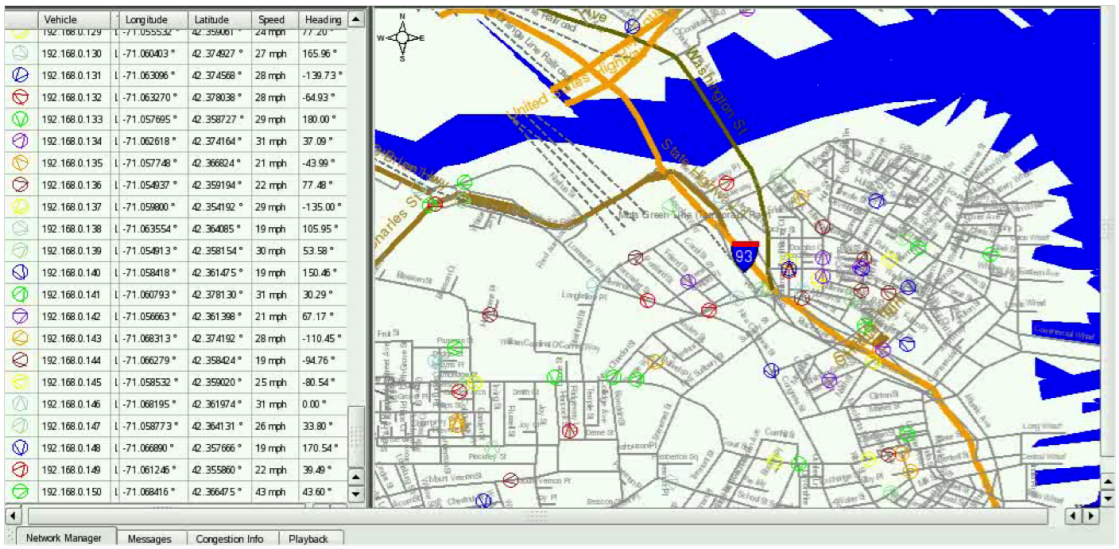

Figure 4.13: Groovenet. An example of GrooveNet simulation, where vehicles are moving in a real map with their coordinates being updated almost in realtime.

network model and a GUI based on Qt. GrooveNet can load real street maps from the TIGER/Line database in order to simulate vehicles mobility on real roads, including fixed mobility, street speed, uniform speed, and car-following mobility models. GrooveNet also supports many operational modes such as drive mode, simulation mode, playback mode, hybrid simulation mode, and test generation mode. Furthermore, the simulator has capabilities such as communicating with surrounding vehicles

[1] http://www.seas.upenn.edu/ rahulm/Research/GrooveNet/

[2] RoadNav is an open source street navigation solution capable of running on a variety of operating systems. It can obtain your position from a GPS unit, plot a map of your area, and provide directions to locations in the USA. It can also verbalize directions using Microsoft SAPI 5.1, Festival, flite, and OS X's built in text to speech engine.
Roadnav uses the free TIGER/Line (Topologically Integrated Geographic Encoding and Referencing) files from the US Census Bureau to build the maps, along with the GNIS state and topical gazetteer data from the USGS to identify locations. It has experimental support for scripting, LCDproc, importing OpenStreetMap data, and importing GPX waypoints and tracks.

to get real traffic information, executing a specific scenario, mixing between the previous two modes, and finally playing back the logfile generated during any of the modes operation for further analysis. GrooveNets unique ability to integrate simulated vehicles with real vehicles allows it to function as testbed software as well as a simulator.

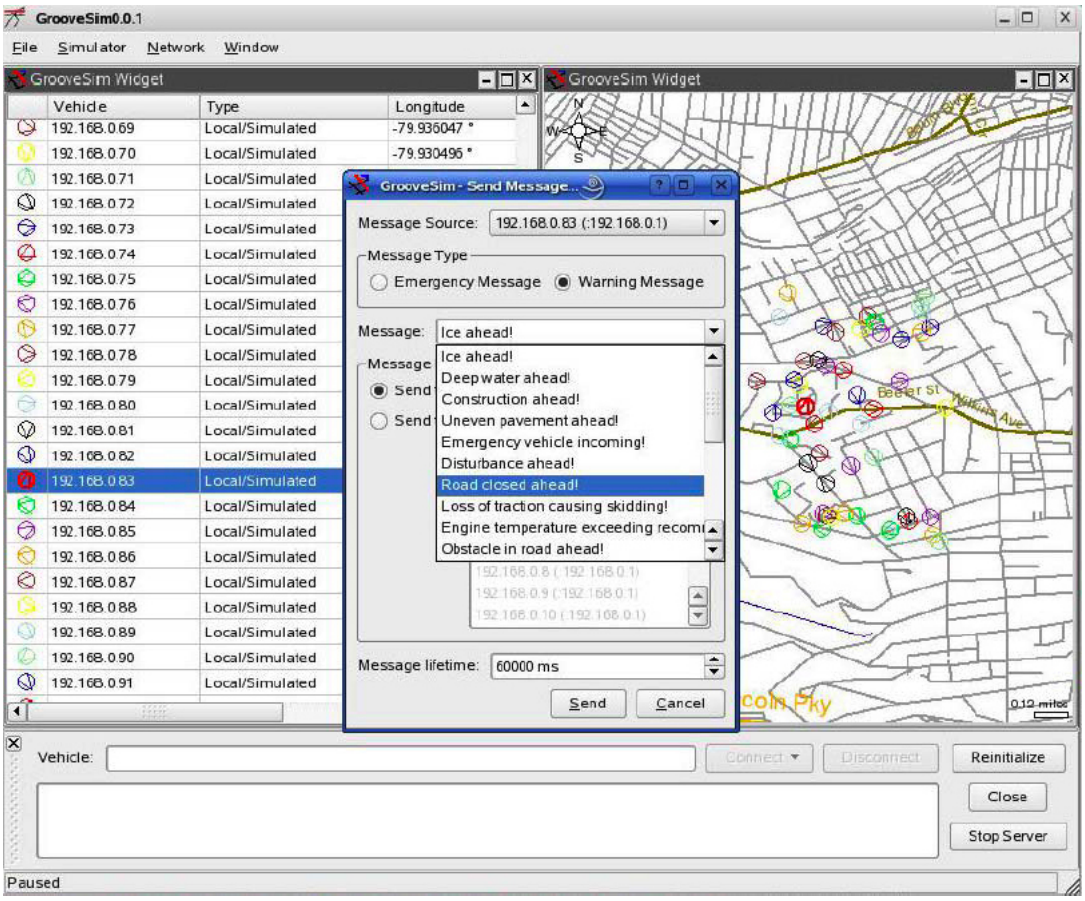

Figure 4.14: Groovenet. The GrooveNet simulator allows to broadcast emergency alerts or road warnings, thus analyzing the effect on the vehicles in realtime. This unique characteristic makes GrooveNet suitable both for simulations and real research testbeds.

4.4.3 TraNS

TraNS[1] (Traffic and Network Simulation Environment) (76) can be called the first VANET simulator. It was the first work to combine a network simulator, ns-2, with a

[1] http://trans.epfl.ch/

vehicular mobility simulator, SUMO, and to provide feedback from the network simulator to the mobility simulator. TraNS can operate in two modes: *network-centric mode* and *application-centric mode*. In the network-centric mode, there is no feedback provided from ns-2 to SUMO, so the vehicles mobility trace file can be pre-generated and fed to the network simulator later. The link between the two simulators in this case is done through a parser that analyzes the mobility trace file generated by SUMO and converts it to a suitable format for ns-2.

In the application-centric mode, the feedback between ns-2 and SUMO is provided

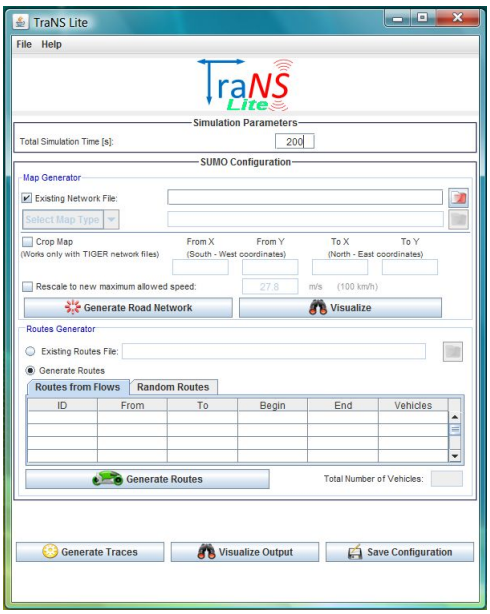

Figure 4.15: TraNS simulator. Screenshot of a light version of TraNS, called TraNS Lite.

through an interface called TraCI (100). In this mode the two simulators (SUMO and ns-2) must run simultaneously. TraCI achieves the link between ns-2 and SUMO by converting the mobility commands coming from ns-2 to a sequence of mobility primitive commands such as stop, change lane, change speed, and so on that can be sent to SUMO. As both simulators are running separately at the same time, the two-way communication in application-centric mode uses two separate events queues.

4.4.4 Veins

Veins (Vehicles in Network Simulation) (90) is another simulator that couples a mobility simulator with a network simulator: SUMO is paired with OMNeT++ by extending SUMO to allow it to communicate with OMNeT++ through a TCP connection. In order to create a bidirectional communication between the two simulators, OMNeT++ has also been extended by adding a module that allows all participating nodes (vehicles) to send commands via the established TCP connection to SUMO. In this case, the two extensions represent the interface between the network simulator and the mobility simulator. Thus, the network simulator can react to the received mobility trace from the mobility simulator by introducing new nodes, by deleting nodes that have reached their destination, and by moving nodes according to the instructions from the mobility simulator.

In Veins, there is a manager module that is responsible for synchronizing the two simulators. At regular intervals, the manager module triggers the execution of one timestep of the traffic simulation, receives the resulting mobility trace, and triggers position updates for all modules it had instantiated. Thus, as with TraNS, this simulator has two separate events queues.

4.4.5 NCTUns

NCTUns[1] (National Chiao Tung University Network Simulation) (99) implements two-way communication with a single events queue. NCTUns 1.0 was developed only as a network simulator, but the most recent version, NCTUns 6.0 (released on 15 January, 2010), integrates some traffic simulation capabilities, such as designing maps and controlling vehicles mobility. A large variety of maps can be designed using different types of supported road segments (e.g., single-lane roads, multilane roads, crossroads, T-shape roads, and lane-merging roads). Also, NCTUns includes a GUI to aid in the design process of the maps.

The supported vehicular movement has two modes, *prespecified* and *autopilot*. In the prespecified movement mode, the scenario designer specifies the moving path and the speed for each vehicle. In autopilot mode, the scenario designer specifies the following

[1] http://nsl10.csie.nctu.edu.tw/

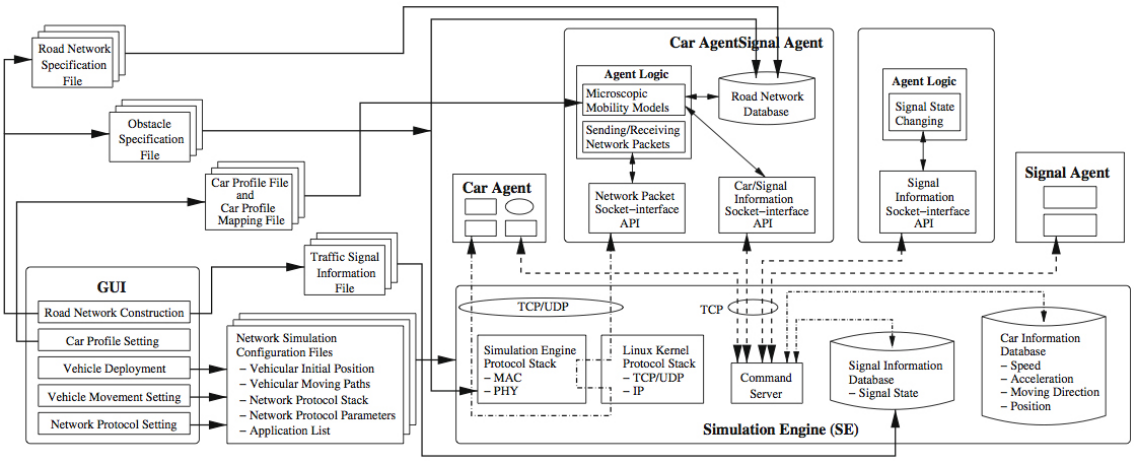

Figure 4.16: NCTUns simulator. The architecture of NCTUns 6.0 network simulator.

parameters for each vehicle: initial speed, maximum speed, initial acceleration, maximum acceleration, maximum deceleration, and so on. Then, the autopilot selects the best route to navigate in the map. Autopilot mode is also capable of performing car following, lane changing, overtaking, turning, and traffic light obeying. The best feature avaliable in NCTUns is that its network protocol stacks includes the Linux kernel protocol stack, including TCP/IP and UDP/IP, and the user level protocol stack and the MAC and PHY layer protocols. *This means that it is possible to run on the simulated nodes whatever application which runs on Linux!* Furthermore, as the simulator works directly on the Linux kernel's network stack, it is not needed the protocol validation (the first version of NCTUns did not rely on real network stack, thus requiring a validation). However, the main disadvantage of NCTUns is tha the code for the vehicles movement logic is integrated with the network simulation code, which makes it difficult to extend. Another disadvantage of this simulator is that, at the moment, it is released for one Linux distribution, Fedora 12 (as it is based on the linux kernel, the portability suffers a bit).

4.4.6 ASH

ASH (Application-aware SWANS with Highway mobility) (7) is an extension of the wireless network simulator SWANS that implements the IDM vehicular mobility model and MOBIL lane changing. ASH supports feedback between the vehicular mobility sub-

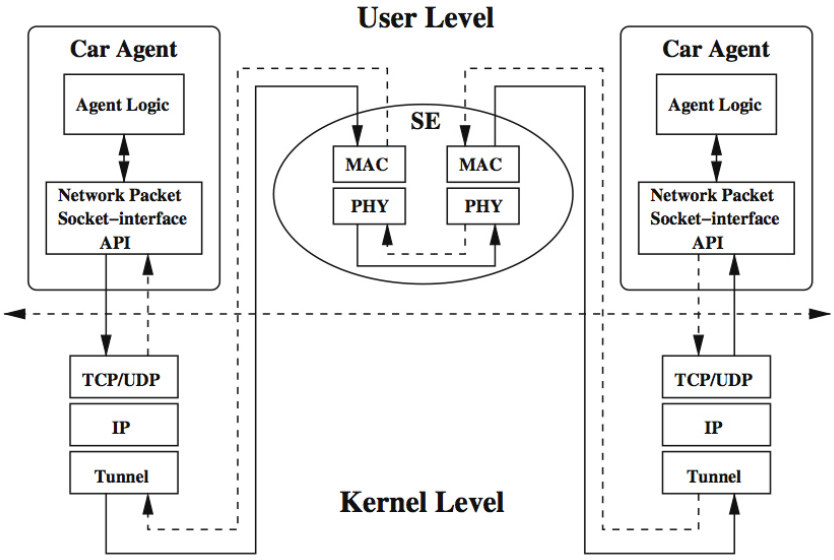

Figure 4.17: NCTUns simulator. The architecture of NCTUns 6.0 network protocol simulation.

simulator and the network subsimulator, making it one of the two-way communication simulators with a single events queue. ASH allows users to design a simple highway segment and customize it by specifying the directions (one-way or two-way), the number of lanes, the number of entries and exits and their corresponding locations along the segment.

In addition to adding highway mobility models to SWANS, ASH extends the node types available:

1. *mobile communicating node:* this represents a participating vehicle that should execute a user-defined appl ication, which specifies how the vehicle should behave;

2. *mobile silent node:* this represents a non participating vehicle that should execute a null application so that it will not be able to send or receive any messages;

3. *static communicating node:* this represents road-side infrastructure that should execute a user-defined application, which specifies how the road-side unit should behave. Also, this kind of node may have different physical layer characteristics (e.g., transmission power) than the mobile communicating nodes;

Attribute	SUMO/Move TraNS	VanetMobiSim	NCTuns
Custom graphs	supported	supported	supported
Random graphs	grid based	voronoi graph	shape-file
Map based graphs	Tiger DB	GDF	bitmap image
Multilane graphs	supported	supported	supported
Start/End position	AP, random	AP, random	random
Trip	random start-end	random start-end	random
Path	random walk, Dijkstra	random walk, Dijkstra	random walk
Velocity	road dependent smooth	road dependent smooth	road dependent smooth

Table 4.1: Traffic level features in SUMO/Move/TraNS, VanetMobiSim and NCTuns mobility simulators.

Attribute	SUMO/Move/TraNS	VanetMobiSim	NCTuns
Human patterns	car following models	IDM, IDM-IM IDM-LC	IDM, IDM-IM IDM-LC
Intersection management	Stoch turn	traffic lights and signs	traffic lights
Lane changing	not supported	supported	supported
Radio obstacles	not supported	supported	supported

Table 4.2: Motion level features in SUMO/Move/TraNS, VanetMobiSim and NCTuns mobility simulators.

4. *static silent node:* this represents a road obstacle that should execute a null application.

In particular, the addition of the mobile silent node is important to allow testing of protocols under different penetration rates, where not all vehicles are equipped with communication devices. The location of the static silent node can either be predetermined before running the simulation or can be determined at runtime, in order to simulate an accident, for example.
Because most VANET applications use flooding-based techniques to disseminate data, ASH also implements the Inter-Vehicle Geocast protocol (IVG) (12). Moreover, it supports a probabilistic version of IVG to take the surrounding traffic density into account.

Attribute	SUMO/Move/TraNS	VanetMobiSim	NCTuns
GUI	not supported	supported	supported
Output	ns-2, GlomoSim Qualnet	ns-2, GlomoSim Qualnet	ns-2
Level of integration	federated/integrated	separated	integrated

Table 4.3: Other features in SUMO/Move/TraNS, VanetMobiSim and NCTuns mobility simulators.

ASH also implements statistical and logging utilities to support simulations. The utilities provide information about the simulation entities at different granularity. This information can be retrieved at the simulation level, lane level, vehicle level, or message type level. The statistics utility provides statistics about all possible events that can occur in the simulation.
ASH accepts a configuration file for the highway scenario. The nodes creation and the communication scenario should be specified in a Java file as done in SWANS.

Mobility Model Class	Integrated Framework Support	Benefits	Drawbacks
Random Movement	Virtually All	⊕ Straightforward, intuitive ⊕ Readily available	⊖ Imprecise ⊖ Potentially unstable
Real-World Traces	GloMoSim, QualNet, OPNET, ns-2, Shawn, JiST/SWANS, OMNeT++/INET Framework	⊕ Most realistic node movement ⊕ Re-usable traces	⊖ Costly and time-consuming ⊖ No free parameterization
Artificial Mobility Traces	GloMoSim, QualNet, OPNET, ns-2, Shawn, JiST/SWANS, OMNeT++/INET Framework	⊕ Realistic node movement ⊕ Free parameterization ⊕ Re-usable traces	⊖ No feedback on driver behavior
Bidirectionally Coupled Simulators	Ongoing efforts for: ns-2, Shawn, JiST/SWANS, OMNeT++/INET Framework	⊕ Realistic node movement ⊕ Free parameterization ⊕ Feedback on driver behavior	⊖ No re-usable traces

Figure 4.18: Benefits and drawbacks. Benefits and drawbacks of several VANETs simulator.

4.5 Scalability of VANET simulators

This section aims to provide a brief overview of network simulators scalabilty, with an experimental result on Qualnet and Ns2, two simulators which have recently gained attention in the research community. When simulating a complex scenario including vanet networks, with both mobility and signal transmission and propagation models, it

might be valuable to understand how far you can go with your simulation. This means that the most important parameters that every researcher should keep in mind before approaching any kind of simulation are *scalability* and *overall performance*. Scalability, which is, generally speaking, the ability to either handle growing amounts of work in a graceful manner or to be readily enlarged, gives to the researcher an idea on the maximum scenario he is allowed to simulate, in terms of number of mobile nodes, terrein dimensions, number of simultaneus protocols and details' level of the simulation's outcome. For example, one may simply think to simulate a large network of thousands of nodes with many details per network layer but if the simulation time or the memory consumption start growing too fast, this may turn out to be unfeasible.
Trying to compare two network simulators is not an easy task to accomplish, first of all beacuse of the different implementations of both kernel and protocols, which may result in completely different resource consumption. Secondly, it should also be considered the detail's level of the simulation itself. In order to gain some experimental results of the performance trend in Qualnet and Ns2 (for a quick comparison), we implemented in both the simulators the following simulation set-up:

- *simulation time:* 600 secs;
- *simulation area:* 2000x2000 sqm.;
- *number of mobile nodes:* up to 1000 in Ns2, up to 4000 in Qualnet;
- *mobility:* random waypoint with initial uniform distribution;
- *wireless protocol:* 802.11b with 150 m nominal radio range, two-ray propagation pathloss model and constant shadowing model;
- *routing protocol:* AODV
- *traffic:* a couple of nodes always involved in UDP CBR (*"Constant Bit Rate"*) transfer with a packet size of 1460 bytes.

The difference in the range of mobile nodes is due to the limitation in the Ns2 scalability which, compared to Qualnet, does not allow simulation of thousands of nodes. In fact, in (110) it was proved that Ns2 does not scale well for wireless sensor network (and then it makes sense to extend the results for Vanet, which is a specialization of a

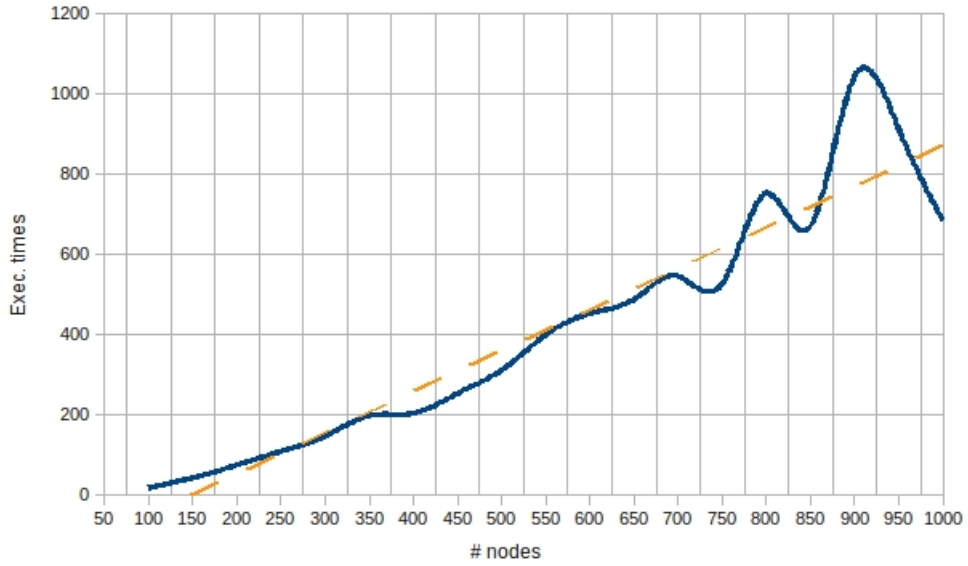

Figure 4.19: Simulation time in Ns2. Simulation time in Ns2 (version 2.34) with mobile nodes ranging from 50 up to 1000 in a 2000x2000 sqm area.

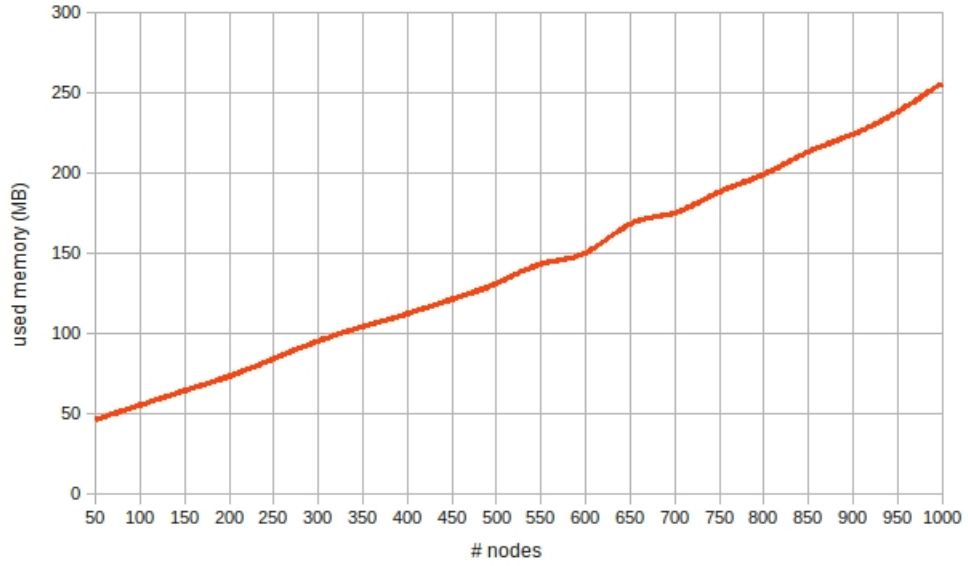

Figure 4.20: Memory memory consumption in Ns2. Memory consumption in Ns2 (version 2.34) with mobile nodes ranging from 50 up to 1000 in a 2000x2000 sqm area.

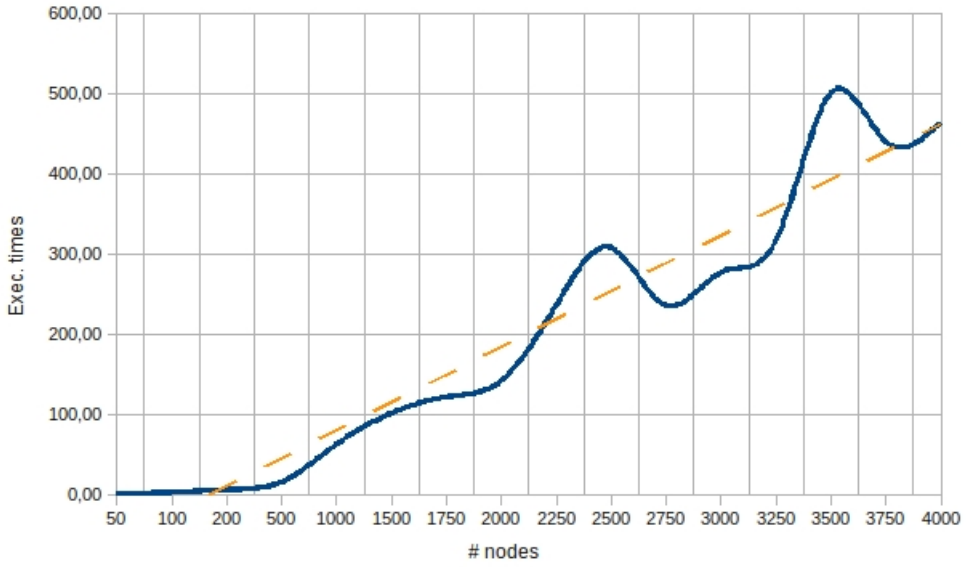

Figure 4.21: Simulation time in Qualnet. Simulation time in Qualnet (version 4.5) with mobile nodes ranging from 50 up to 4000 in a 2000x2000 sqm area.

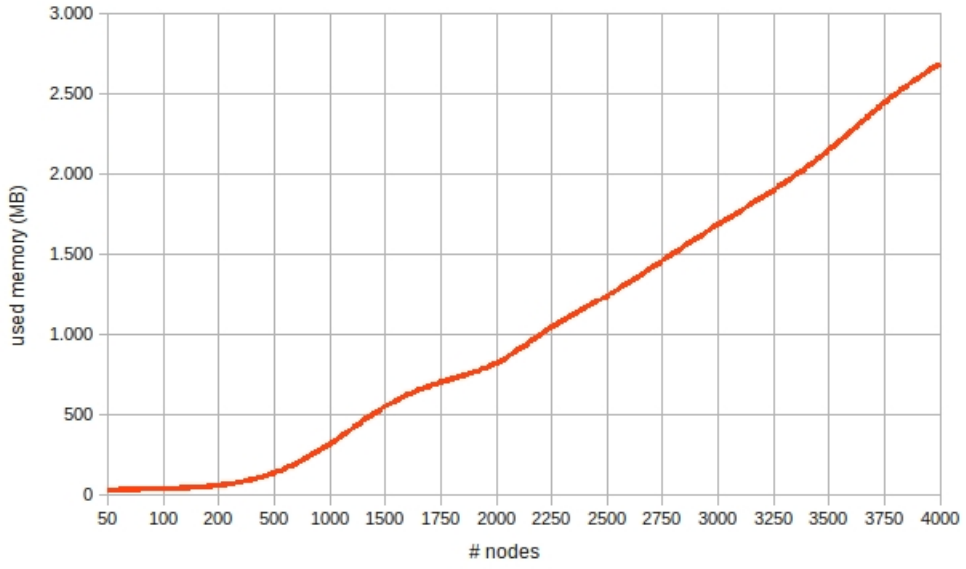

Figure 4.22: Memory memory consumption in Qualnet. Memory consumption in Qualnet (version 4.5) with mobile nodes ranging from 50 up to 4000 in a 2000x2000 sqm area.