

Simulation Software: Omnet++ GTNetS GlomoSim / QualNet

Holger Füßler

Course overview

1. Introduction

2. Building block: RNG

**3. Building block:
Generating random variates I
and modeling examples**

**4. Building block:
Generating random variates II
and modeling examples**

**5. Algorithmics:
Management of events**

6. NS-2: Introduction

7. NS-2: Fixed networks

8. NS-2: Wireless networks

9. Output analysis

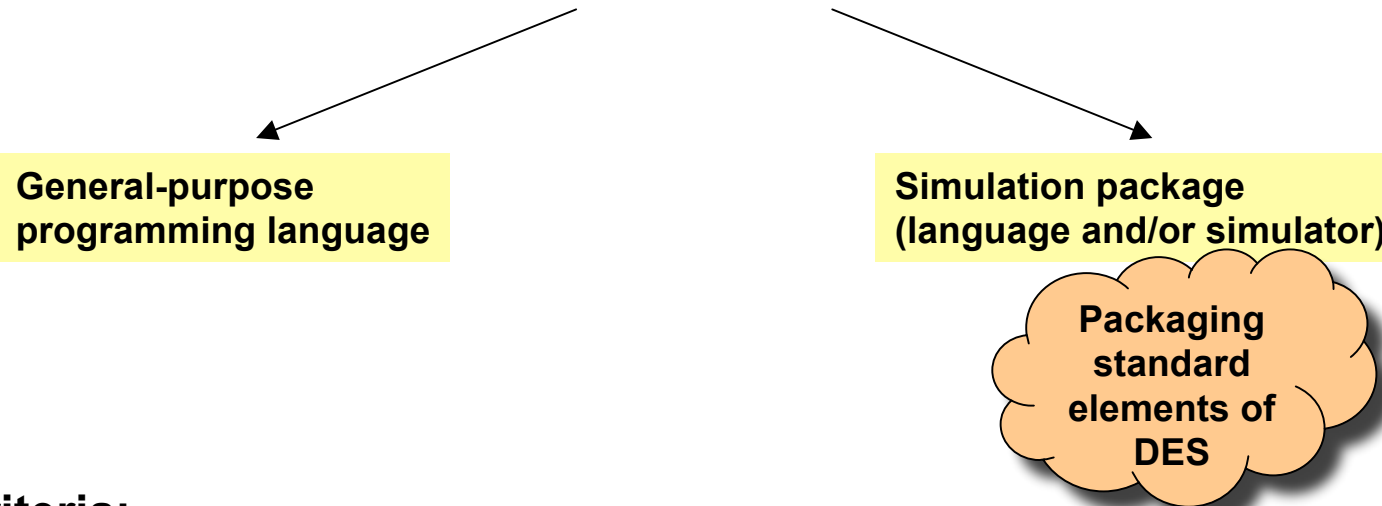
10. Simulation lifecycle, summary

11. Alternative Simulators

Simulation software and tools

» ... where simulation meets software technology

- How to build models and run simulations *conveniently*.



» **Criteria:**

- **General capabilities (flexibility, available models, re-use, ...)**
- **Hardware/software considerations**
- **Graphical facilities**
- **Statistical features**
- **“Learning curve”, documentation, support**
- **Output reports and plots**

Choice of simulation software

» So far we know ns-2

- What do you like about ns-2?
- What do you dislike?

» Many more options:

- CSIM: C-based simulation package (<http://www.atl.external.lmco.com/proj/csim/>)
- JSIM: Java-based simulation package (<http://chief.cs.uga.edu/~jam/jsim/>)
- OMNeT++
- GTNetS
- GloMoSim / QualNet
- OPNET Modeler
- ...

Today:

» OMNeT++ / GTNetS / QualNet

Lecture overview: OMNeT++

- » **OMNeT++ overview**
- » **Concept**
- » **Architecture / Steps to follow**
- » **Simulator internals**
- » **Example**
- » **Existing modules**
- » **Differences with ns-2**

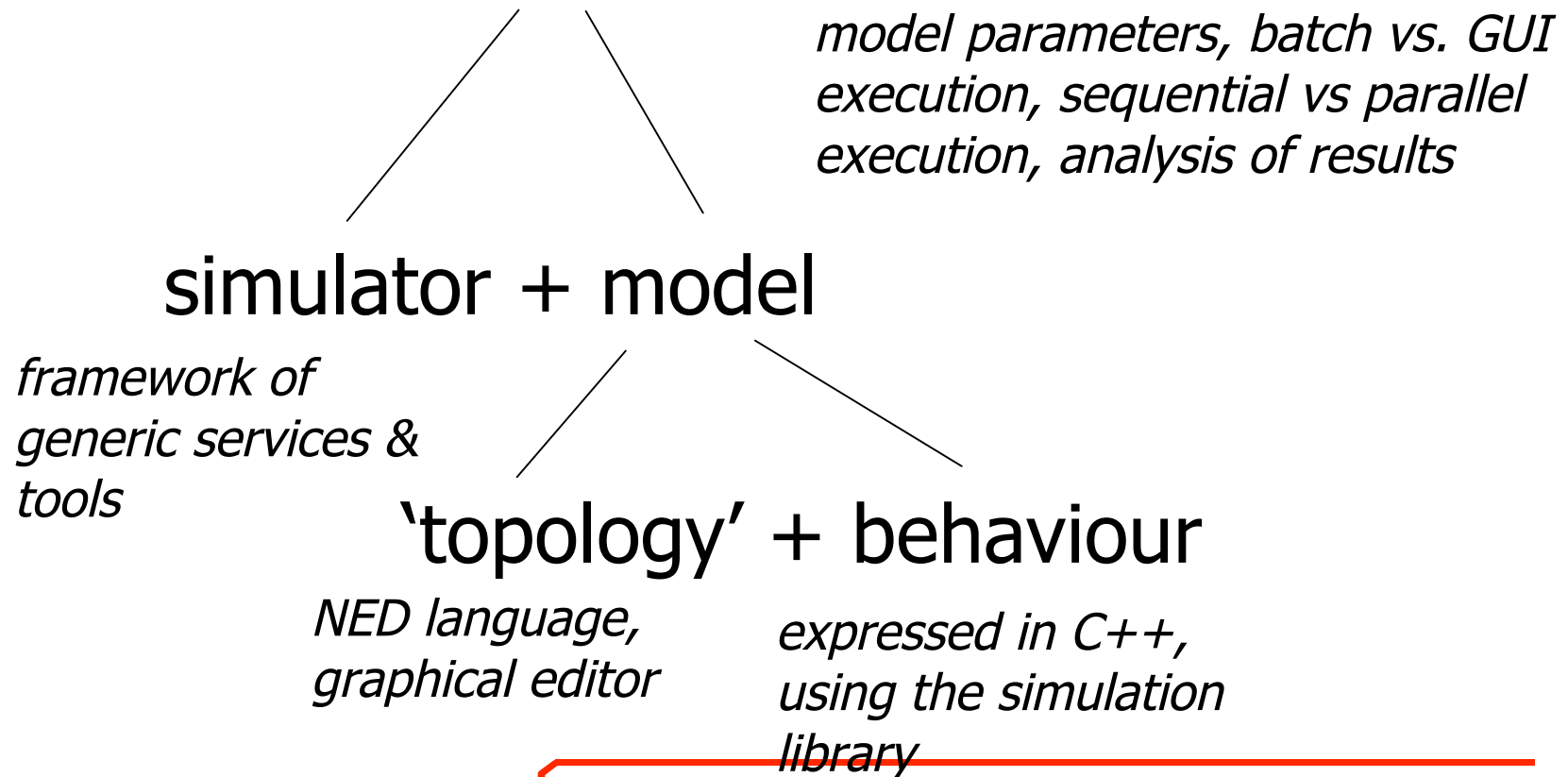
OMNeT++

- » **Open-source, generic simulation framework -- best suited for simulation of communication networks, multiprocessors and other complex distributed systems** (further examples: queuing systems, hardware architectures, server farm, business processes, call centers)
- » **C++ based simulation kernel plus a set of libraries and tools (GUI and command-line)**
- » **Platform: Unix, Windows**
- » **Being developed at BUTE (Technical University of Budapest), CVS at Uni Karlsruhe**
- » **Contributions from worldwide**
- » **Active user community (mailing list has about 240 subscribers)**

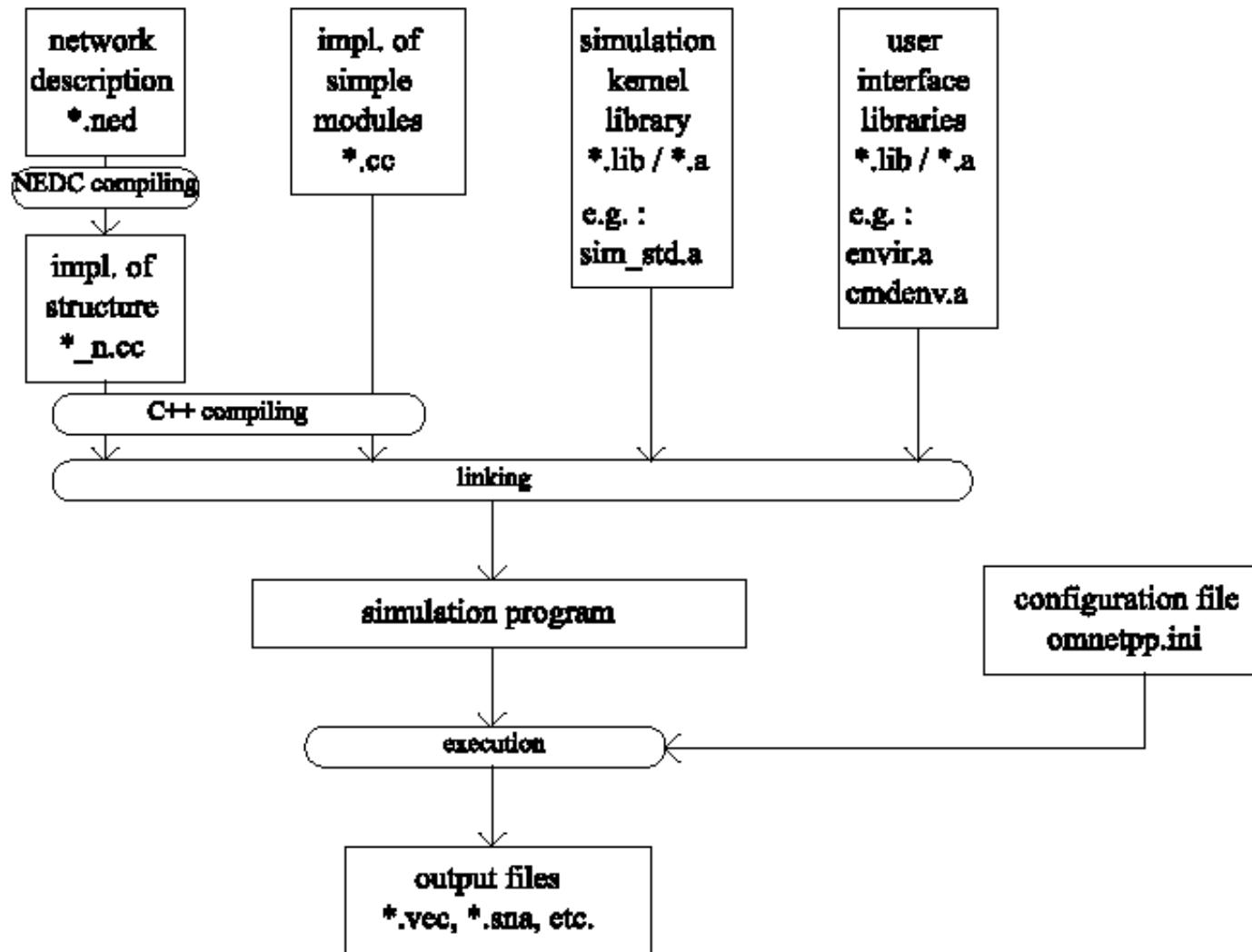
Concept

Separation of concerns:

simulation = sim. program + experiments



Architecture



Steps to follow

- 1. Map your system into a set of communicating modules**
- 2. Use NED (or GNED) to define the model's structure**
- 3. Using C++, describe the active components of the model as concurrent processes**
- 4. Provide a suitable configuration file containing options of OMNeT++ and parameters to your model**
- 5. Build the simulation program and run it**
- 6. Analyze results written into output vector files.**

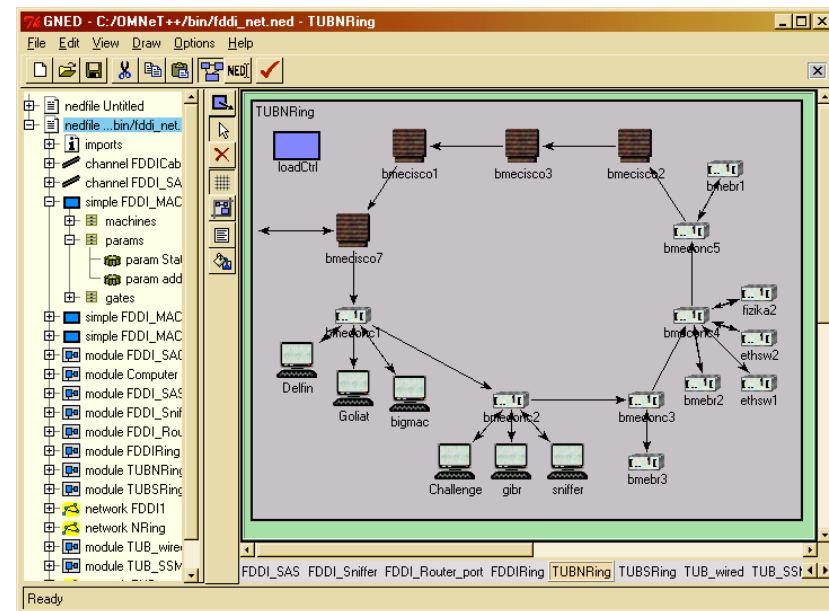
Defining the topology

» NED – Network Description Language

- declares simple modules with their interfaces
- defines compound modules (submodules, interconnection)
- defines the network as instance of a module type

» GNED -- Graphical Network Editor

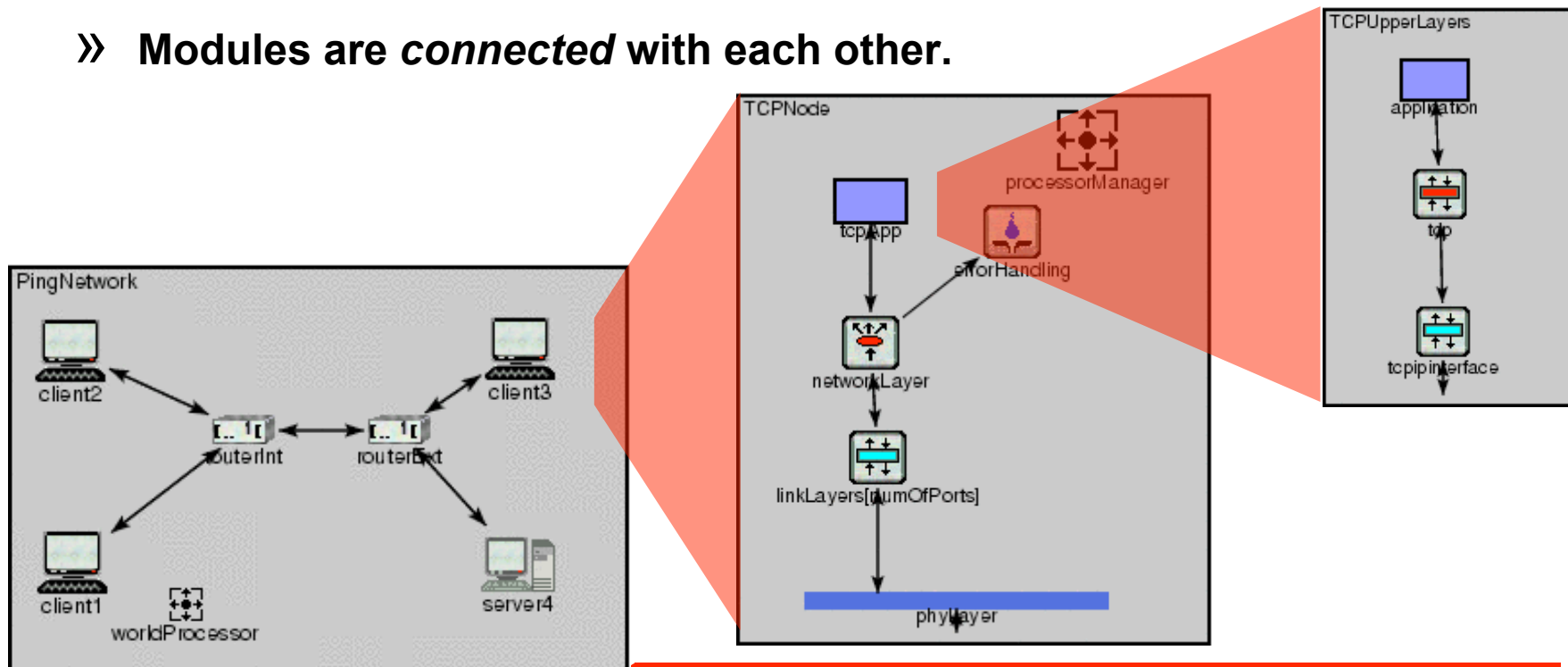
- works directly with NED files
- two-way tool: you may edit in NED sources or graphical view – they are automatically kept consistent



Model structure

Component-oriented approach (Hierarchically nested modules):

- » The basic building block is a simple module (programmed in C++).
- » Simple modules can be grouped to form compound modules.
- » Modules are *connected* with each other.



Defining the behaviour

- » **Behaviour is encapsulated in simple modules, C++.**
- » **A simple module:**
 - **sends messages, reacts to received messages**
 - **collects statistics**
- » **Gates are the input and output interfaces for messages.**
- » **Connections (links) are established between modules, characterized by:**
 - **Propagation delay**
 - **Bit error rate**
 - **Data rate**

Simulation library

- » **Simulation class library covers commonly needed functionality, such as:**
 - *random number generation*
 - *queues and other containers*
 - *support for topology discovery and routing*
 - *recording simulation results (output vectors)*
 - *statistics collection and estimation (histograms, etc)*

The GUI

The screenshot displays the OMNeT++/Tkenv interface for a simulation named 'BusNet'. The main window includes a menu bar (File, Edit, Simulate, Trace, Inspect, View, Options, Help) and a toolbar with simulation controls like STEP, RUN, FAST, EXPRESS, UNTIL..., and STOP. The status bar shows 'Run #5: BusNet', 'Event #429', 'T=0.0902266843 (90ms)', and 'Next: BusNet.bus (id=30)'. Below this, statistics for 'Msgs scheduled: 12', 'Msgs created: 299', 'Msgs present: 42', 'Ev/sec: n/a', 'Simsec/sec: n/a', and 'Ev/simsec: n/a' are provided.

The left sidebar shows a tree view of the simulation components: BusNet (BusNetwork), parameters (CA), gates (cArray), class-members, hostA (EtherHost), hostB (EtherHost), hostC (EtherHost), hostD (EtherHost), bus (Bus) (id=3), and scheduled-events. The main area shows a self-message log for 'BusNet.hostB' with events #423, #424, #425, and #426, detailing frame reception and transmission states.

A central window titled '(EtherHost) BusNet.hostB' displays a network diagram. It shows two generators, 'gen[0]' and 'gen[1]', connected to a central bus. The bus is connected to two sinks, 'sink[0]' and 'sink[1]', and an interface labeled 'etherframe-21-11'.

Two floating windows provide additional data. The top window, '(cOutVector) ...t.hostA.mac.class-members.bytesSent', shows a line graph of bytes sent over time. The x-axis represents time from 15.8317926 to 16.6777926. The y-axis represents bytes sent, with values 1154404, 1192273, and 1230142. The graph shows a steady increase in bytes sent. Below the graph, it states 'Last value: t=16.677783 (16.67s) value=1.23014e+06' and includes an 'Options...' button.

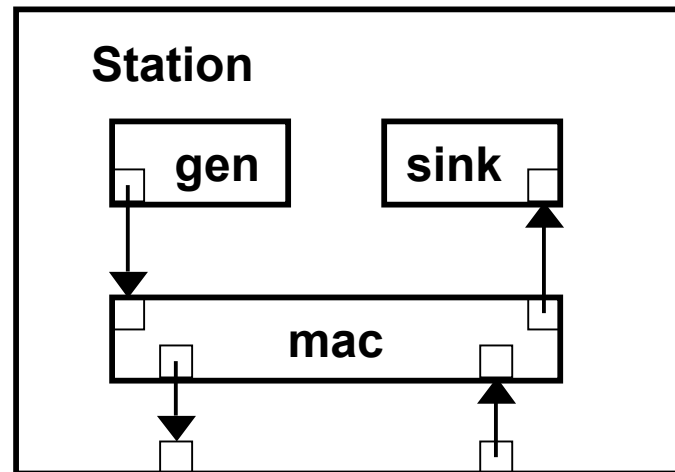
The bottom window, '(EtherMAC) BusNet.hostB.mac', shows a log of self-messages and frame reception events. It includes details such as 'Self-message (cMessage)EndReception received', 'Frame reception complete', and 'Received frame from upper layer: (EtherFrame)etherframe-14-1689'. It also shows transmission states and queue lengths.

Running under the GUI

- » **Run or single-step the simulation**
- » **Monitor state of simulation and execution speed**
- » **Examine model object tree**
- » **Explore modules and see message flow**
- » **Examine scheduled events**
- » **Trace what one module is doing**
- » **Step to next event in a module**
- » **Look at state variables and statistics**
- » **Find out pointer values for C++ debugging (gdb)**
- » **Look at results being recorded**

Simple example

» Station:



Submodels can be connected to each other or to parent module

Simple definition in NED

```
simple MAC  
  
parameters: address;  
  
gates:  
  
in: from_higher_layer, from_network;  
  
out: to_higher_layer, to_network;  
  
endsimple
```

A compound model

```
module Station
  parameters: mac_address;
gates:
  in: in; out: out;
submodules:
  mac: MAC
      parameters:
        address=mac_address;
  gen: Generator;
  sink: Sink;
connections:
  mac.to_network --> out,
  mac.from_network <-- in,
  mac.to_higher_layer --> sink.in,
  mac.from_higher_layer <-- gen.out;
endmodule
```

Existing modules

» Simulation Models TCP/IP networks:

- IPSuite
- IPv6Suite

» LAN/MAN protocols:

- Ethernet
- FDDI
- Token Ring

» Wireless LAN protocols:

- 802.11
- Hiperlan/2

» Mobility and ad-hoc frameworks:

- Mobility Framework
- An AODV framework

OMNeT++ vs NS-2 (seen from OMNeT's perspective)

	OMNeT++	NS-2
Flexibility	Generic simulation framework	Good for IP networks
Topology Description	NED or GUI	OTcl
Model Management	Models independent of simulation kernel	Monolithic
Hierarchical Models	Hierarchical module structure	“Flat” models
Debugging	Tkenv	None
Models Available	Few computer systems	Rich in communication protocols
Scalability	Limit is the virtual memory of computer	Some problems in large networks
Parallel Simulation	PDES: Parallel Discrete Event Simulation	Developed in Georgia Tech
Embeddability	Simulation kernel can be embedded in other applications	None

References

» Home page: www.omnetpp.org

- Downloadable
- Tutorials (M/M/1 queue!)
- Manual
- Mailing List
- Models
- ...

» Commercial version also exists: www.omnest.com

GTNetS – The Georgia Tech Network Simulator

- » complete new design (pure C++)
- » main design goals: scalability, performance
- » Download:
<http://www.ece.gatech.edu/research/labs/MANIACS/gtnets.htm>
- » a lot of protocols as network primitives (802.3/11 / IP / TCP)
- » primitives for statistics generation / tracing
- » natural support for distributed execution
- » Mobility: RWP / specified waypoint
- » Radio Channel Modelling: ?

The GTNetS Process

Develop Model in C++ (Algorithms)



Write int main() yourself



compile and link against GTNetS object files



run

GTNetS – Discussion

- » **pure C++**
 - **IMHO very nice**
 - **BUT: have to provide Functions for reading scenarios etc.**
- » **potentially a lot faster than ns-2**
- » **BUT: lots of people still focus on ns-2 → newer protocols available, more used means usually mor debugged**

References - GTNetS

Download:

<http://www.ece.gatech.edu/research/labs/MANIACS/gtnets.htm>

» **Riley, George F. : “The Georgia Tech Network Simulator”, p. 5-12, In Proc. of SIGCOMM 2003, Karlsruhe, Germany**

GlomoSim / QualNet

- » **C++ / ParSec (language for description of parallel processes)**
- » **ParSec has to be installed separately (bad license for commercial use)**
- » **QualNet is commercial spin-off, GlomoSim free but no longer maintained**

The GlomoSim / QualNet Process

Develop Model in C++ (Algorithms)



Create config.in File



run (no tracing, statistics collected on-line)

GlomoSim / QualNet - Discussion

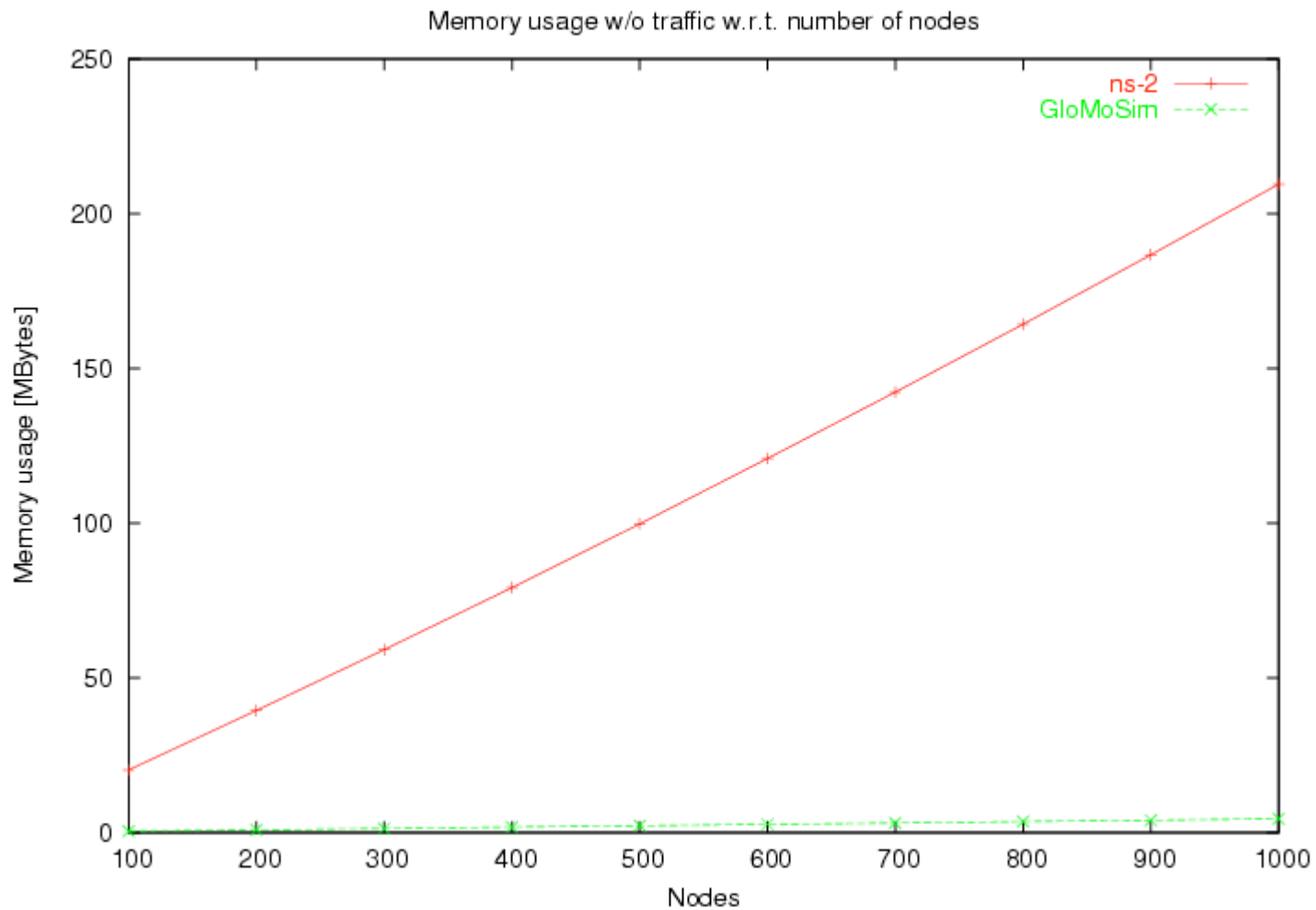
» Strengths:

- Ad-Hoc Networking (routing etc.)
- Radio Channel Modeling (Directional Antennas)
- a lot of nodes possible

» Weaknesses:

- needs PARSEC
- licensing, sourcing (QualNet) / up-to-dateness (GlomoSim)
- tracing (for debugging)

GlomoSim vs. ns-2 – Memory Scalability



References – GlomoSim / Qualnet

- » **GlomoSim:** <http://pcl.cs.ucla.edu/projects/glomosim/>
- » **QualNet:** <http://www.scalable-networks.com>

SimpleSim

- » **eigenes Projekt am Lehrstuhl**

- » **Idee: Sacrifice Performance for**
 - **Stability**
 - **Reproducibility**
 - **Understandability**

- » **So far:**
 - **Pure Java**
 - **Linear Movements**
 - **UDG Radio Propagation**
 - **MAC802.11 / ALOHA MACs**
 - **AODV / DSR Implementation**

- » **In Progress:**
 - **Sim Distribution (SimpleGrid)**

Wrap-Up – Which Simulator should I use?

» Criteria re-visited:

- **General capabilities (flexibility, available models, re-use, ...)**
 - which specific problem / class of problems do I want to tackle?
 - which orders of magnitude for simulation size?
- **Hardware/software considerations**
 - which OS is available / needed?
 - which compilers etc.?
- **Graphical facilities**
 - educational / scientific purpose?
- **Statistical features**
 - tracing vs. inline statistics
- **“Learning curve”, documentation, support**
 - how many languages do I have to learn?
- **Output reports and plots**
- **What do the others in my community use?**