



Faculty of Sciences,
Technology
and Communication

Introduction to event-based discrete network simulation and to OMNeT++

Middleware Labs

Guillaume-Jean Herbiet
<guillaume.herbiet@uni.lu>



Discrete event-based simulation

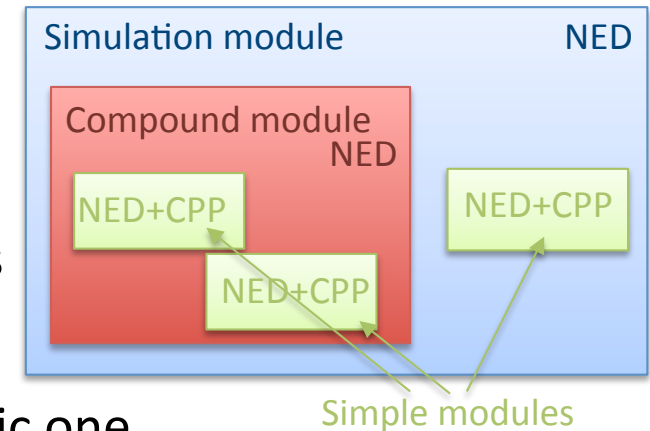
- Events are stored in a single queue
- Events are sorted in chronological order
- Events can generate (schedule) one or more new events
- Simulated time advances to the timestamp of the head event before execution

OMNeT++: Introduction

- OMNeT++ :
 - Discrete event simulator
 - Packet level simulation, possible to go to subpacket modeling
- Available on GNU/Linux, Windows and Mac OS
- Features
 - Open source, dual licensing
 - OMNeT offers the framework for wired simulation only
 - Completed by frameworks
 - Upper-layers models (INET framework)
 - Wireless and ad-hoc models (Mobility Framework)
 - Command line (cmdenv) or GUI (tkenv)
 - Traces and statistics
 - Distributed simulation

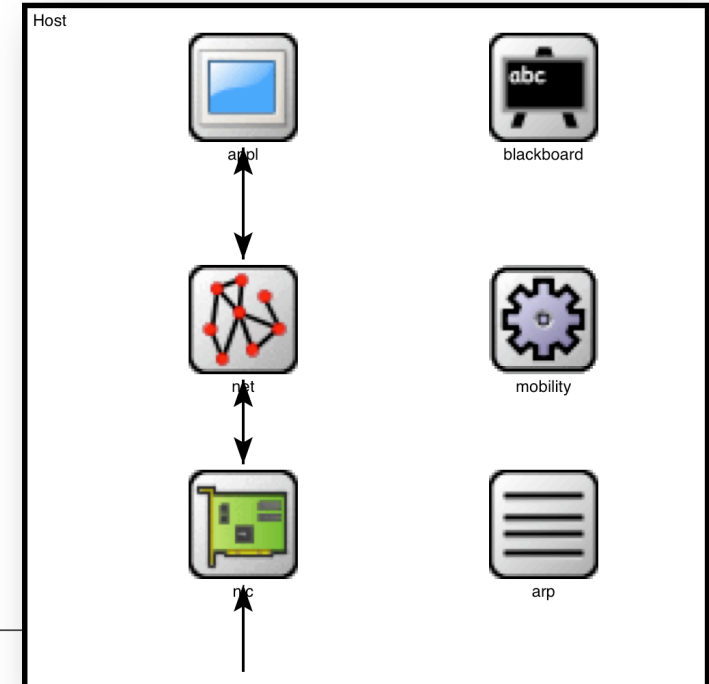
OMNeT++: Paradigm

- Object-oriented programming and modules
 - A module is defined by a NED file
 - Compound module = place holder for sub-modules
 - Simple module = implemented through a C++ class
 - Possible to easily derive modules from more generic one
 - Application: derived module used can be a simulation parameter
- Network configuration and simulation parameters are configured using a text file <omnetpp.ini>
 - All nodes can have different parameters
 - Allow automatic sequential launch of multiple runs with different parameters
 - Possible to call different subconfig file



Mobility Framework

- Provides support for wireless and ad-hoc modelling
 - Mobility management: managed locally by each node by the *mobility* module
 - Wireless connectivity: managed globally by the *channelControl* module
 - Radio broadcast of packets: packets are duplicated and sent concurrently to all nodes in Tx range according to *channelControl*
 - SNR evaluation:
received packets can be dropped if noise is too high
 - Channel access models
(e.g. : 802.11)
 - Cross layering features inside a node through *blackBoard* module



The OMNeT++ API

- OMNeT++ and MF have a well documented API
- Objects all conveniently derive from objects that offer:
 - Easy modules creation, suppression, modification
 - Animation features for use in tkenv
 - Messages creation, management, suppression
- Other classes exist for:
 - Queuing management
 - Finite State Machine (FSM) management
 - Statistics collection



NED and MSG files

- NED files define compound or simple modules
 - Gates (=links) to other modules
 - Configuration parameters
- MSG files define messages (or packets)
 - Fields type, default values,...
 - Can also contain complex items (like vectors or C++ classes)

Tracing and statistics

- Easy tracing in cmdenv or tkenv using “ev” macro
- OMNeT++ provides ability to collect statistics as scalars or vectors
 - Can be dynamically displayed during simulation in tkenv
 - Can be displayed later using built-in tools
 - Or parsed later using user-defined scripts (results stored in plain-text files)
- It is also possible to use a user-defined logger or any convenient C++ code that has to be linked at compilation time

Tkenv and user interface

- Easy to check network behavior before going to cmdenv (faster) for extensive simulation campaigns
- Also a good demo tool
 - Displays node mobility and messages paths
 - Shows evolution of node parameters or stored variables
 - e.g.: Correctness of routing table
 - API allows to easily animate network
 - Color nodes depending on states
 - Display “bubbles” or icons on events
 - Colors/thicken links depending on traffic
 - Also possible to use Tcl/tk to easily create custom animations (e.g. : slot allocation in TDMA networks)



(cQueue) multicastNetwork.router1

1 objects

Class	Name	T=0.91
IPDatagram	udpAppData-2	T=0.91

(StandardHost) multicastNetwork.host1

(StandardHost) multicastNetwork.host1 (id=2)

multicastNetwork.host1

blackboard
1+1 routes
routingTable

tcp

pingApp

networkLayer

172.0.0.1 /10M
rcv:7 snt:9
q:0

ppp[0]
(PPPPFrame)udpAppData-2

OMNeT++/Tkenv - multicastNetwork

File Edit Simulate Trace Inspect View Options Help

Run #1: multicastNetwork Event #650 T=0.9110014846 (911ms) Next: multicastNetwork.router1.ppp[0] (id=

Msgs scheduled: 21 Msgs created: 474

Ev/sec: n/a Simsec/sec: n/a

multicastNetwork (MulticastNetwork)

- parameters (cArray)
- gates (cArray)
- host1 (StandardHost)
- host2 (StandardHost)
- host3 (StandardHost)
- host4 (StandardHost)
- host5 (StandardHost)
- host6 (StandardHost)
- router1 (Router)
- router2 (Router)
- router3 (Router)
- router4 (Router)
- scheduled-events (cArray)
- pppEndTxEvent
- pppEndTxEvent
- nnnEndTxEvent

(IPDatagram) ...heduled-eve...

(IPDatagram) simulation.scheduled-events.udpAppData-2

General Sending/Arrival Fields Control Info

short version = 4
short headerLength = 20
IPAddress srcAddress = 172.0.0.3
IPAddress destAddress = 225.0.1.1
int transportProtocol = 17 (IP_PROTO_UDP)
short timeToLive = 31
int identification = 0
bool moreFragments = true
bool dontFragment = false
int fragmentOffset = 10360
unsigned char diffServCodePoint = 0
int optionCode = 0 (IPOPTION_NO_OPTION)

(MulticastNetwork) multicastNetwork

(MulticastNetwork) multicastNetwork (id=1) (ptr00E9D)

multicastNetwork

host1

router1

host2

host3

router2

router3

router4

host4

host6

Simulation workflow



1. An OMNeT++ model is build from components (*modules*) which communicate by exchanging messages. Modules can be nested, that is, several modules can be grouped together to form a *compound module*. When creating the model, you need to map your system into a hierarchy of communicating modules.



2. Define the model structure in the **NED language**. You can edit NED in a text editor or in **GNED**, the graphical editor of OMNeT++.



```
void Computer::  
activity()  
{  
    for (;;)   
        cMessage *msg =
```

3. The active components of the model (*simple modules*) have to be programmed in C++, using the **simulation kernel** and **class library**.

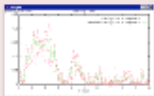


```
[General]  
sim-time-limit =  
random-seed =  
[Parameters]
```

4. Provide a suitable `omnetpp.ini` to hold OMNeT++ configuration and parameters to your model. A config file can describe several simulation runs with different parameters.



5. Build the simulation program and run it. You'll link the code with the OMNeT++ simulation kernel and one of the **user interfaces** OMNeT++ provides. There are command line (batch) and interactive, graphical user interfaces.



6. Simulation results are written into output vector and output scalar files. You can use **Plove** and **Scalars** to visualize them. For more thorough statistical analysis, you can use standalone packages such as R, Octave or Matlab, or even spreadsheets like OpenOffice Calc or Gnumeric.