# Introduction to Simulation and Data Analysis

By Quentin Bramas

ProgRes 2017

Inspired by slides by Walid Benchaita, and Graham Horton

# Outline

- System, Model, and Simulation
- Discrete-Event Simulation
- OMNET, WSNet, JBOTSIM
- Data Analysis and Visualization

# System, Model, and Simulation

# System: Discrete and Continuous

- System:

– a collection of entities that act and interact together toward the accomplishment of some logical end.

- Discrete system:

– state variables change instantaneously at separated point in time, e.g., a bank, since state variables - number of customers, change only when a customer arrives or when a customer finishes being served and departs

- Continuous system:

– state variable change continuously with respect to time, e.g., airplane moving through the air, since state variables - position and velocity change continuously with respect to time

# Model

- A model:

– is a set of assumptions about the operation of the system

- These assumptions can be:
  - algorithmic (sequence of steps)
  - mathematical (equations)
  - logical (conditions)

- This model can be "run" in order to study the real system

# Simulation

- A definition of simulation:
  1. Imitation of the operation of a real-world system
  2. Generation of an artificial history of a system
  3. Observation of the artificial history
- Simulation is performed using a model

# Advantages of Simulation

- study new designs without interrupting real system
- study new designs without needing extra resources
- Improve understanding of system
- Manipulate time
- Less dangerous / expensive / intrusive

# Abstraction & Idealisation

There are two main techniques for building models:

- Abstraction
- Idealisation

Abstraction means leaving out unnecessary details

- Represent only selected attributes of a customer

Idealisation means replacing real things by concepts

- Replace a set of measurements by a function

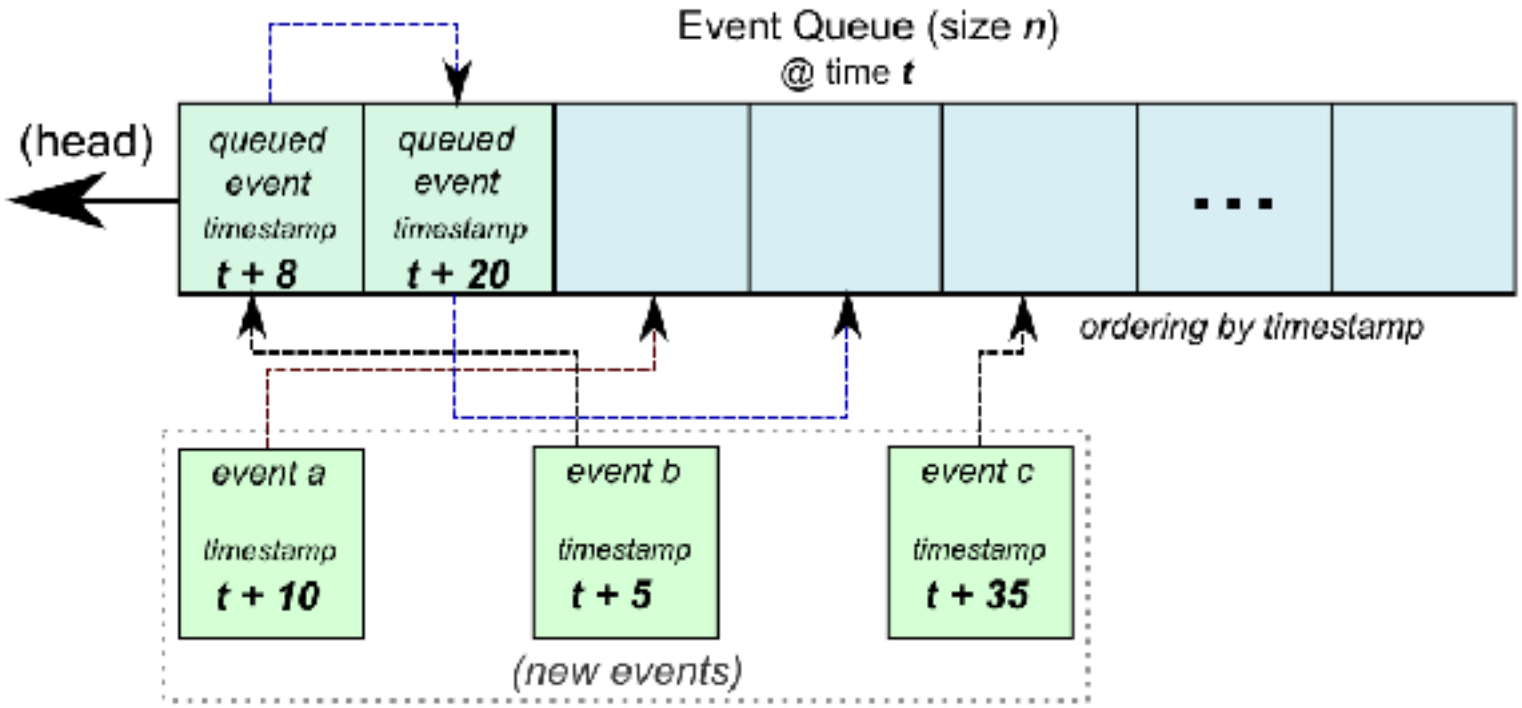
# The Event Queue

The event queue is a list of thing happening in the system, ordered by time of occurrence.

The event queue contains all future events that are scheduled

New events are inserted in the queue.
You can also delete an event from the queue

# The Event Queue



# The Event Queue

A simulation consists in executing the first event in the queue, until no event remains.

# The Simulation Algorithm

Remove and process the 1st event:

- Remove 1st event from the queue
- Advance simulation time
- Update state variables
- Execute the event

If the queue is not empty, start again.

# The Simulation Algorithm

**Remark:**
The time to simulate X seconds is the same as simulating X years, if no events occurs.

In both cases its instantaneous.

# Network Simulators

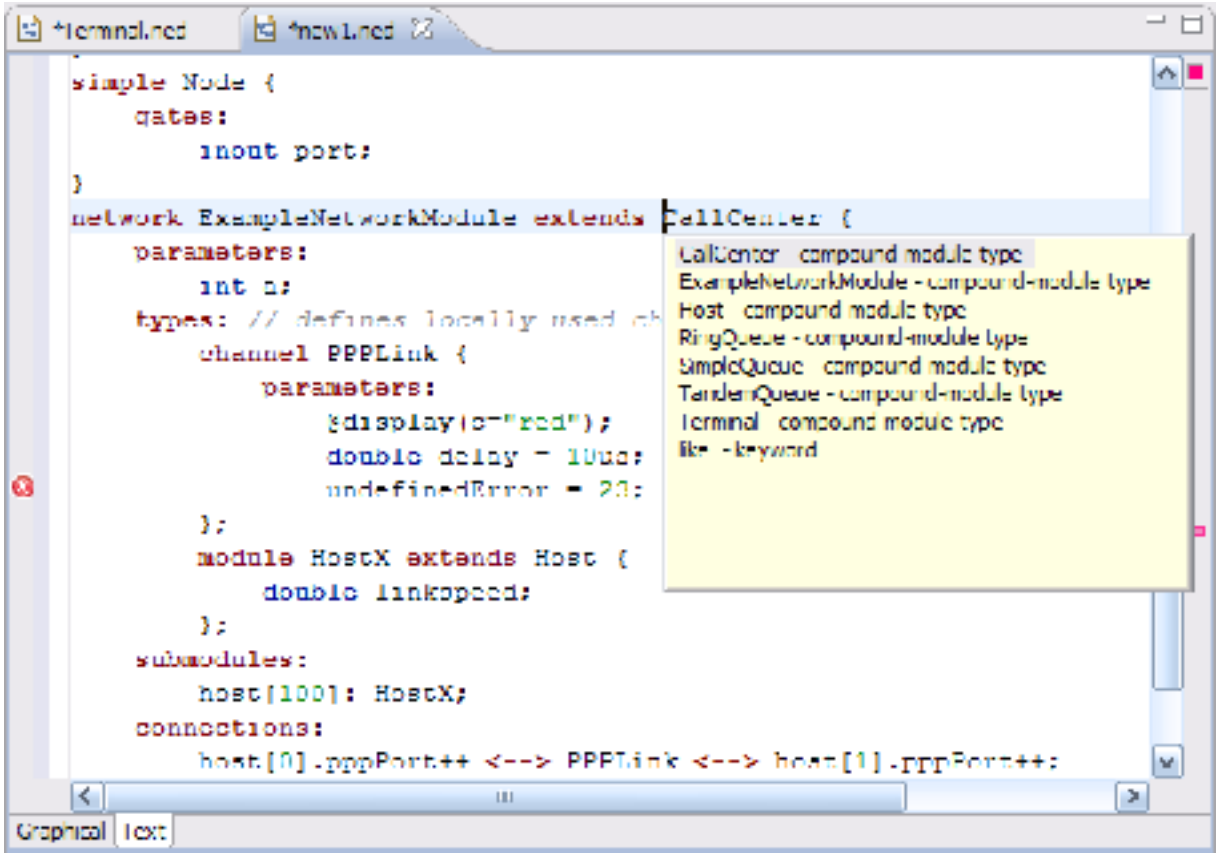
- NS2
- NS3
- OMNeT++
- OPNeT
- WSNeT
- JBotSim

# OMNeT++

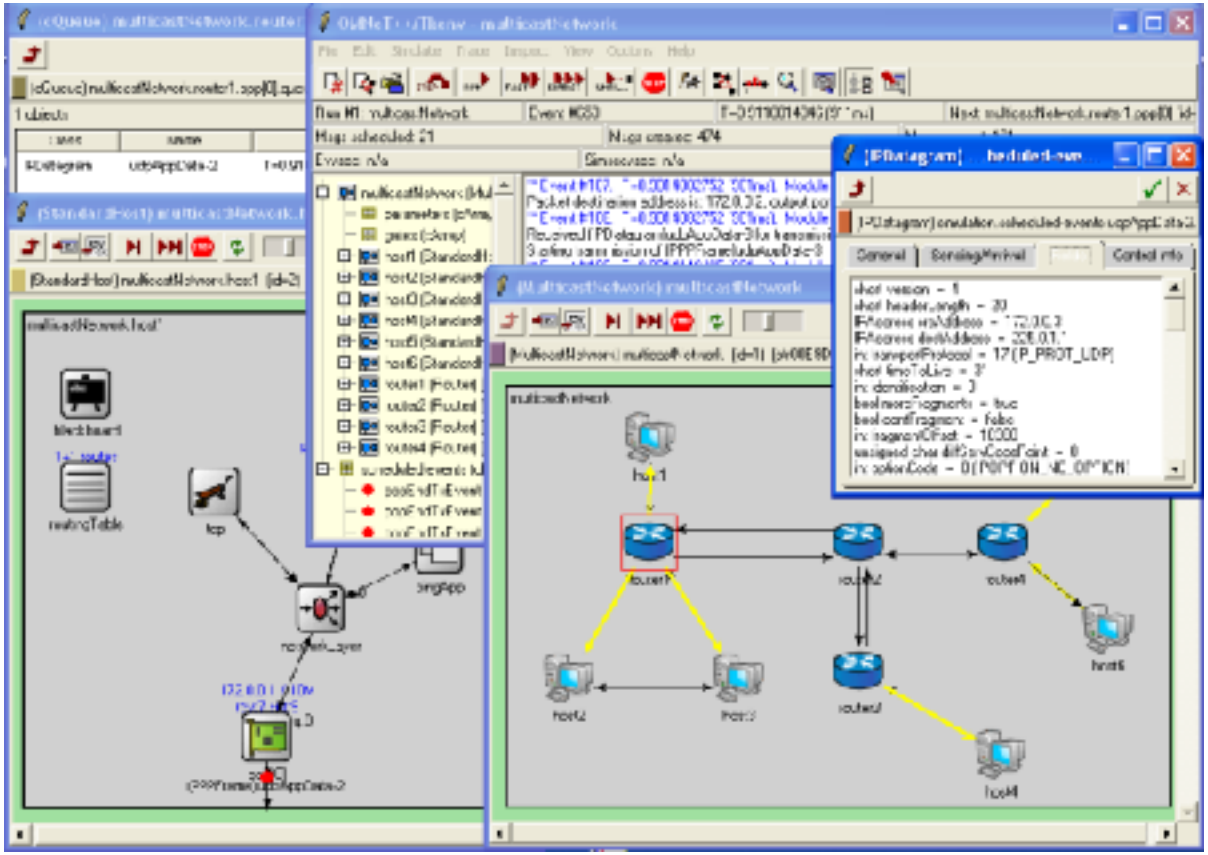
# What is OMNeT++?

- ~~It's a network simulator!!~~
- C++ class library (runs on Windows, Linux, Mac OS X, and other Unix-like systems):
  - Simulation kernel ,
  - Utility classes (random number generation, topology discovery etc.)
- Infrastructure to assemble simulations
  - Network Description Language (NED)
  - .ini files
- Runtime user interfaces: Tkenv, Cmdenv
- Tools to facilitate creating simulations and evaluating results: GNED, Scalars, Plove
- Extension interfaces for real-time simulation, emulation…

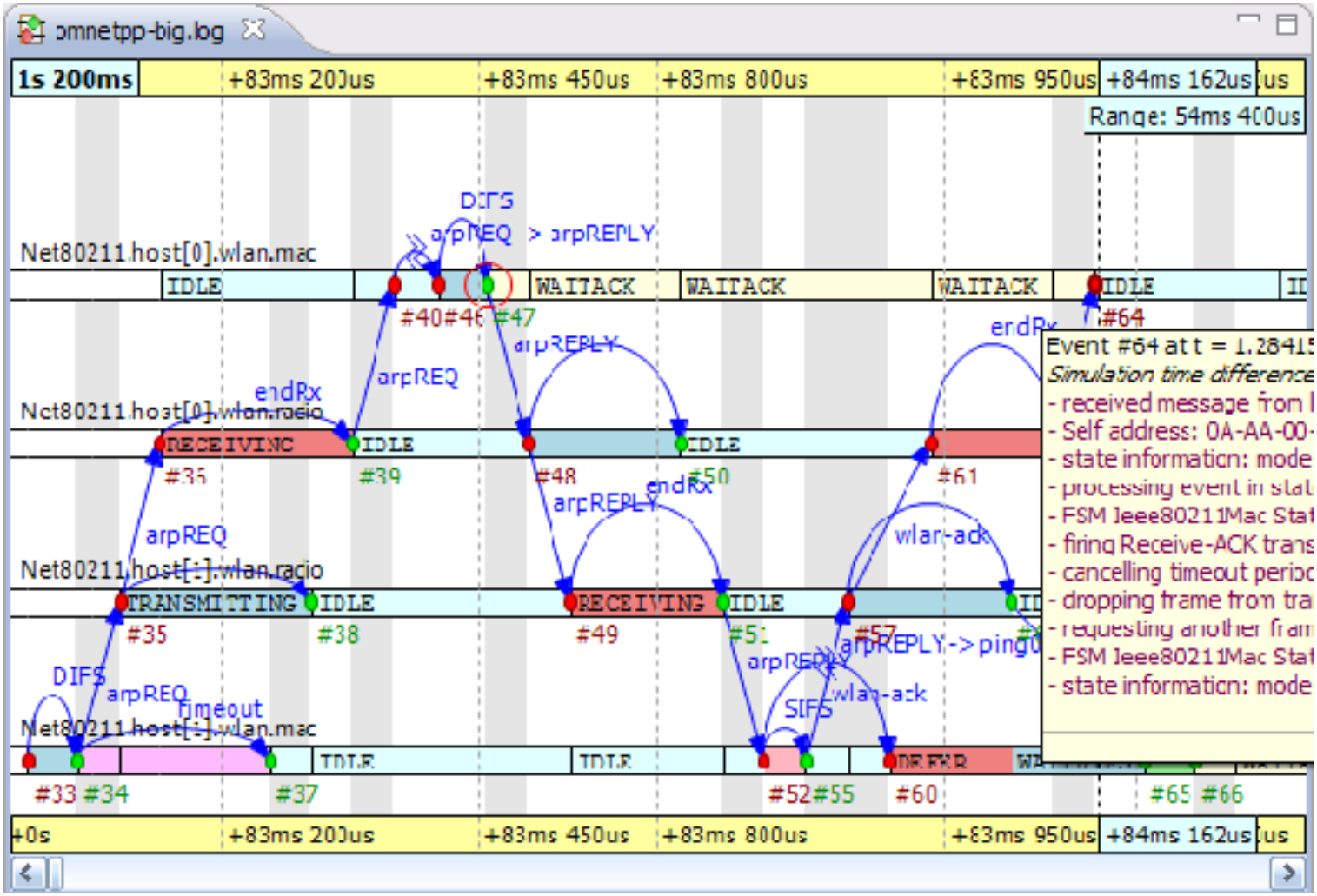
# OMNeT++


*terminal.ned
*new1.ned
simple Node {
gates:
inout port;
}
network ExampleNetworkModule extends CallCenter {
parameters:
int n;
types: // defines locally used ch
channel PPPLink {
parameters:
@display("red");
double delay = 10us;
};
module HostX extends Host {
double linkspeed;
};
submodules:
host[100]: HostX;
connections:
host[0].pppPort++ <--> PPPLink <--> host[1].pppPort++;
CallCenter compound module type
ExampleNetworkModule - compound module type
Host compound module type
RingQueue - compound module type
SimpleQueue compound module type
TandemQueue - compound module type
Terminal compound module type
like - keyword
Graphical
Text

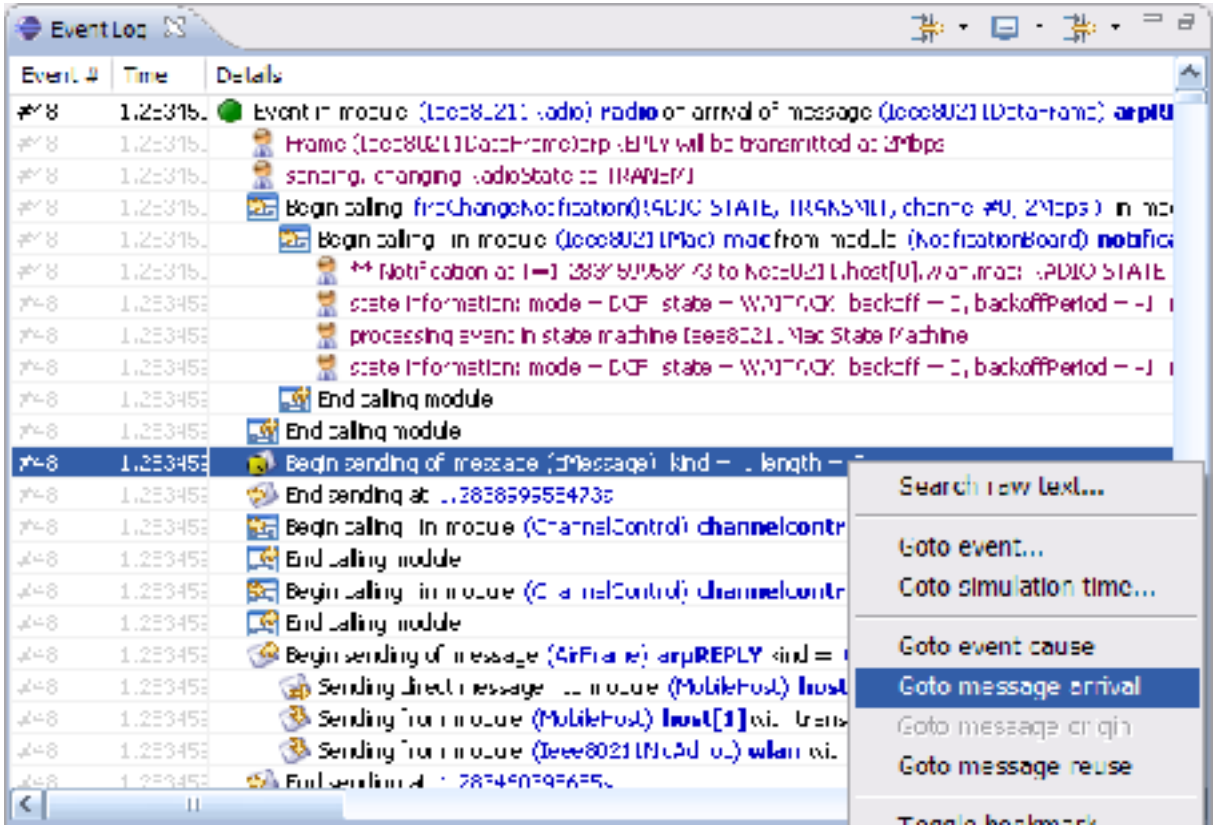
# OMNeT++



# OMNeT++


omnetpp-big.log
1s 200ms
+83ms 200us
+83ms 450us
+83ms 800us
+83ms 950us
+84ms 162us
Range: 54ms 400us
Net80211.host[0].wlan.mac
IDLE
WAITACK
DIFS
arpREQ
arpREPLY
#40
#46
#47
#64
Net80211.host[0].wlan.radio
RECEIVING
endRx
#35
#39
#48
#50
#61
Net80211.host[1].wlan.radio
TRANSMITTING
#38
#49
#51
#57
wlan-ack
arpREPLY -> ping
SIFS
timeout
Net80211.host[1].wlan.mac
#33
#34
#37
#52
#55
#60
#65
#66
Event #64 at t = 1.28415
Simulation time difference
- received message from
- Self address: 0A-AA-00-
- state information: mode
- processing event in stat
- FSM Ieee80211Mac Stat
- firing Receive-ACK trans
- cancelling timeout perio
- dropping frame from tra
- requesting another fram
- FSM Ieee80211Mac Stat
- state information: mode
+0s

# OMNeT++

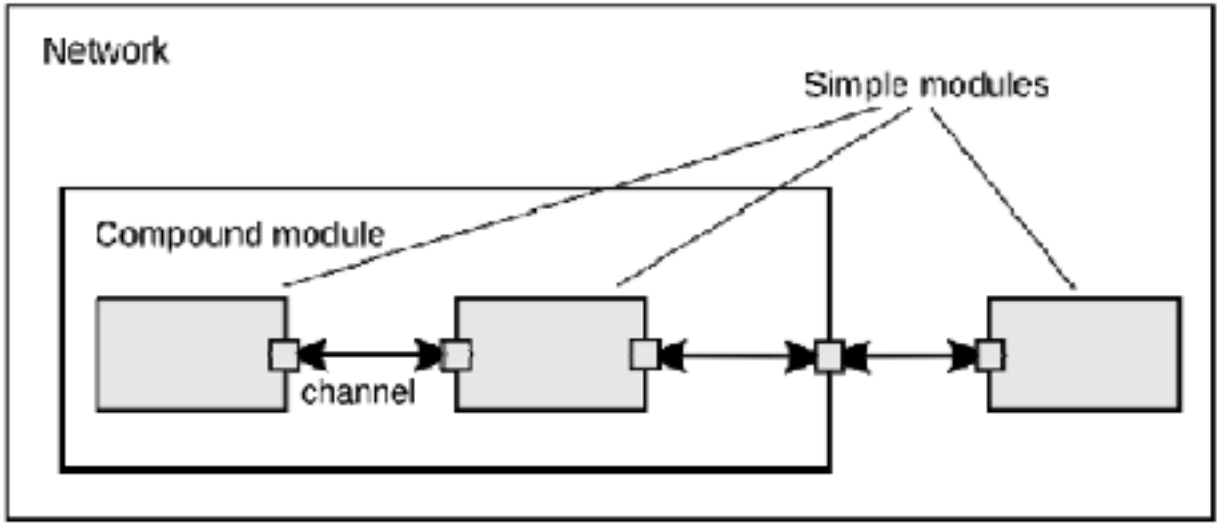

Event Log
Event #
Time
Details
Search raw text...
Goto event...
Goto simulation time...
Goto event cause
Goto message arrival
Goto message origin
Goto message reuse

# Flexibility

- Core framework for discrete event simulation.
  - Different add-ons for specific purposes.
- Fully implemented in C++.
- Functionality added by deriving classes following specified rules.

# Programming model

- Simulated objects are represented by modules
  - Simple or Compound
  - Communicate by messages (directly or via gates)



# Model management

- Clear separation among simulation kernel and developed models.
- Easiness of packaging developed modules for reuse.
- No need for patching the simulation kernel to install a model.

# Debugging and tracking

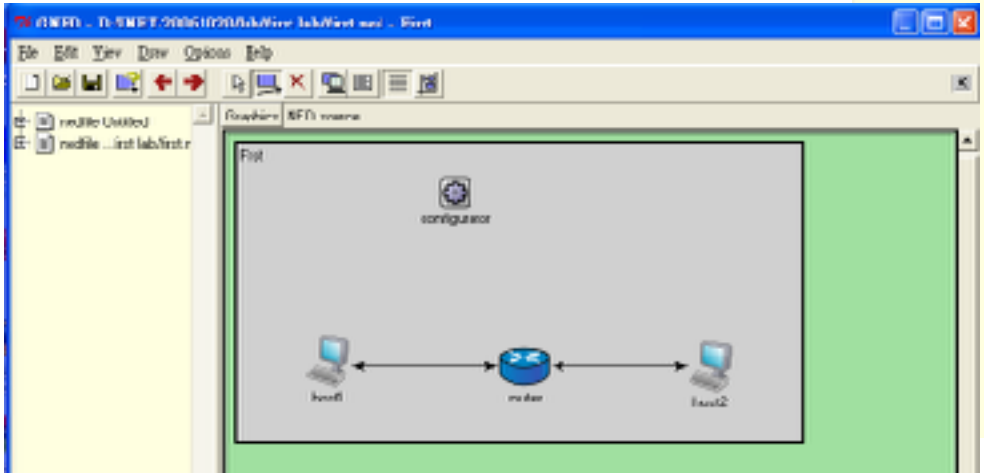
- Supports
  - Recording data vectors and scalars in output files
  - Random numbers with several distributions and different starting seeds
  - displaying info about the module's activity, snapshots, breakpoints
- Easy to configure using .ini file
- Batch execution of the same simulation for different parameters is also included

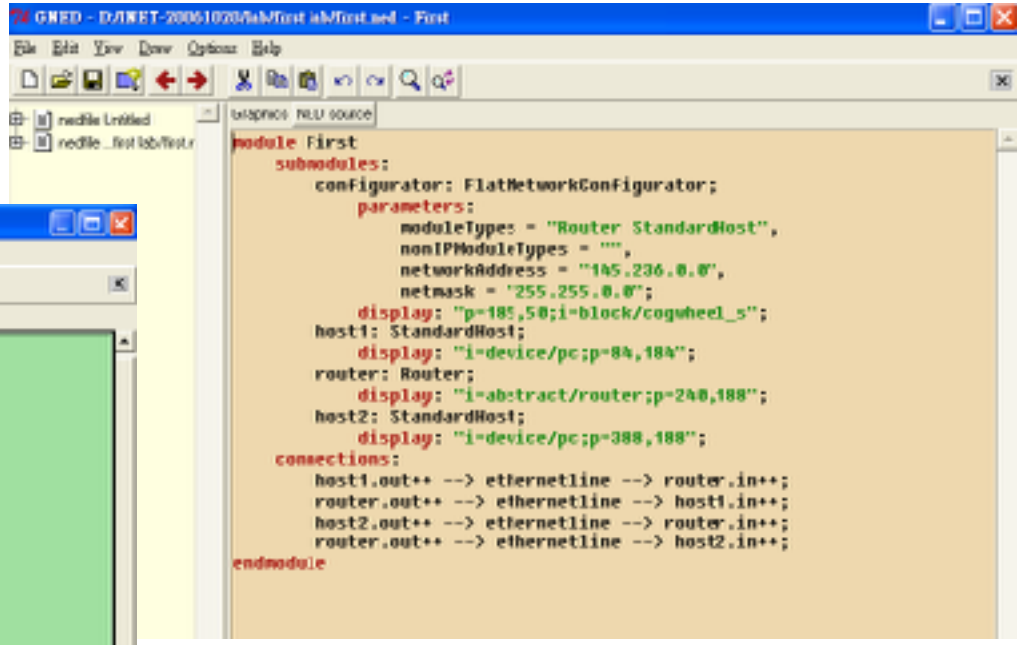
# Simulation Modes

- Command line
- Interactive GUI
  - Tcl/Tk windowing, allows view what's happening and modify parameters at run-time.

# Topology: NED Overview

- The topology of a model is specified using the NED language.
- Edit it with GNED or other text editor.



# NED description

- Import directives
- Channel definitions
- Simple and compound module definitions
- Network definitions


```
//
// A network
//
network Network
{
    submodules:
        node1: Node;
        node2: Node;
    connections:
        node1.port++ <--> {datarate=100Mbps;} <--> node2.port++;
}
```

# Behavior

- C++ Code automatically generated.
- Only methods describing behavior are needed to be redefined.

# OMNeT++ ; A Framework Approach

- Various simulation models and frameworks
  - For Specific Application Areas
  - Mostly Open Source
  - Developed Independently of OMNeT++

# OMNET++ Frameworks

- Simulation Frameworks Based on OMNeT++:
  - Mobility Framework
    - Mobile & Wireless Simulations
  - INET Framework
    - Wired & Wireless TCP/IP Based Simulations
  - Castalia
    - Wireless Sensor Networks
  - MiXiM
    - Mobile & Wireless Simulations

# Simulators Based on OMNeT++

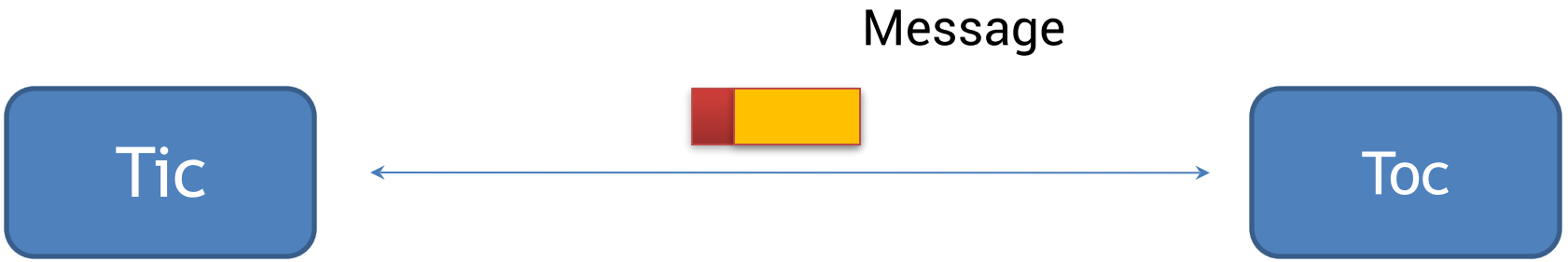
- OverSim
  - Overlay and Peer-to-Peer Networks (INET-based)
- NesCT
  - OS simulations
- Consensus Positif and MAC Simulator
  - for sensor networks
- CDNSim
  - content distribution networks, youtube
- PAWiS
  - Power Aware Wireless Sensor Networks Simulation Framework
- Other:
  - FIELDBUS, ACID SimTools, X-Simulator

# OMNET++

Example: Tic Toc

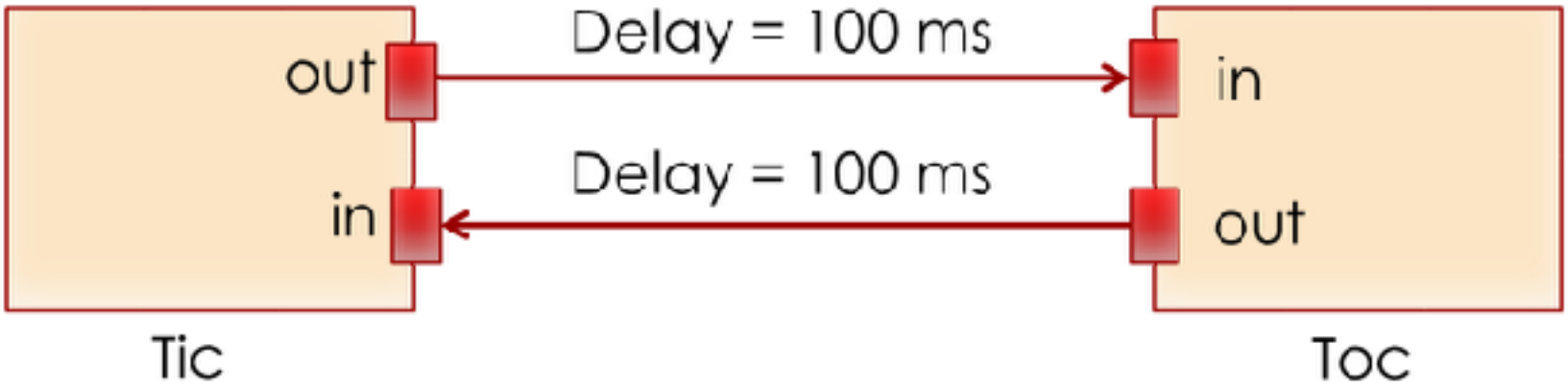
# OMNeT Tic Toc

- Tic Toc is a network with two nodes
- Tic send a message to Toc, Toc resend to Tic and vise versa



# OMNeT Tic Toc

– Ned file



# OMNeT Tic Toc (Ned)

```
simple Txc1
{
        gates:
                input in;
                output out;
}
network Tictoc1
{
                submodules:
                        tic: Txc1;
                        toc: Txc1;
                connections:
                        tic.out --> { delay = 100ms; } --> toc.in;
                        tic.in <-- { delay = 100ms; } <-- toc.out;
}
```

# OMNeT Tic Toc (C++)

```
#include <string.h>
#include <omnetpp.h>

class Txc1 : public cSimpleModule
{
        protected:
                virtual void initialize();
                virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc1);

void Txc1::initialize()
{
        if (strcmp("tic", getName()) == 0)
        {
                cMessage *msg = new cMessage("tictocMsg");
                send(msg, "out");
        }

}

void Txc1::handleMessage(cMessage *msg)
{
        send(msg, "out");
}
```

# OMNeT Tic Toc

Create a file omnetpp.ini

```
[General]
network = Tictoc1
```

We now create the Makefile which will help us to compile and link our program to create the executable tictoc:

```
opp_makemake
```

Compile and link the simulation by issuing the make command:

```
make
```

Launch the simulation:

```
./tictoc
```

# OMNeT Tic Toc

We can add debug message in the source code:

```
EV << "Received message `" << msg->getName() << "', sending it out again\n";
```

We add the counter as a class member:

```
class Txc3 : public cSimpleModule
{
  private:
      int counter;  // add a counter here
  Protected:
```

We set the variable to 10 in initialize() and decrement in handleMessage(), that is, on every message arrival. After it reaches zero, the simulation will run out of events and terminate. And we add :

```
 WATCH(counter);
```

# JBotSim

# JBotSim

JBotSim is a simulation library (in Java) for distributed algorithms in dynamic networks

```
import jbotsim.Topology;
import jbotsim.ui.JViewer;

public class HelloWorld{
    public static void main(String[] args){
        new JViewer (new Topology());
    }
}
```

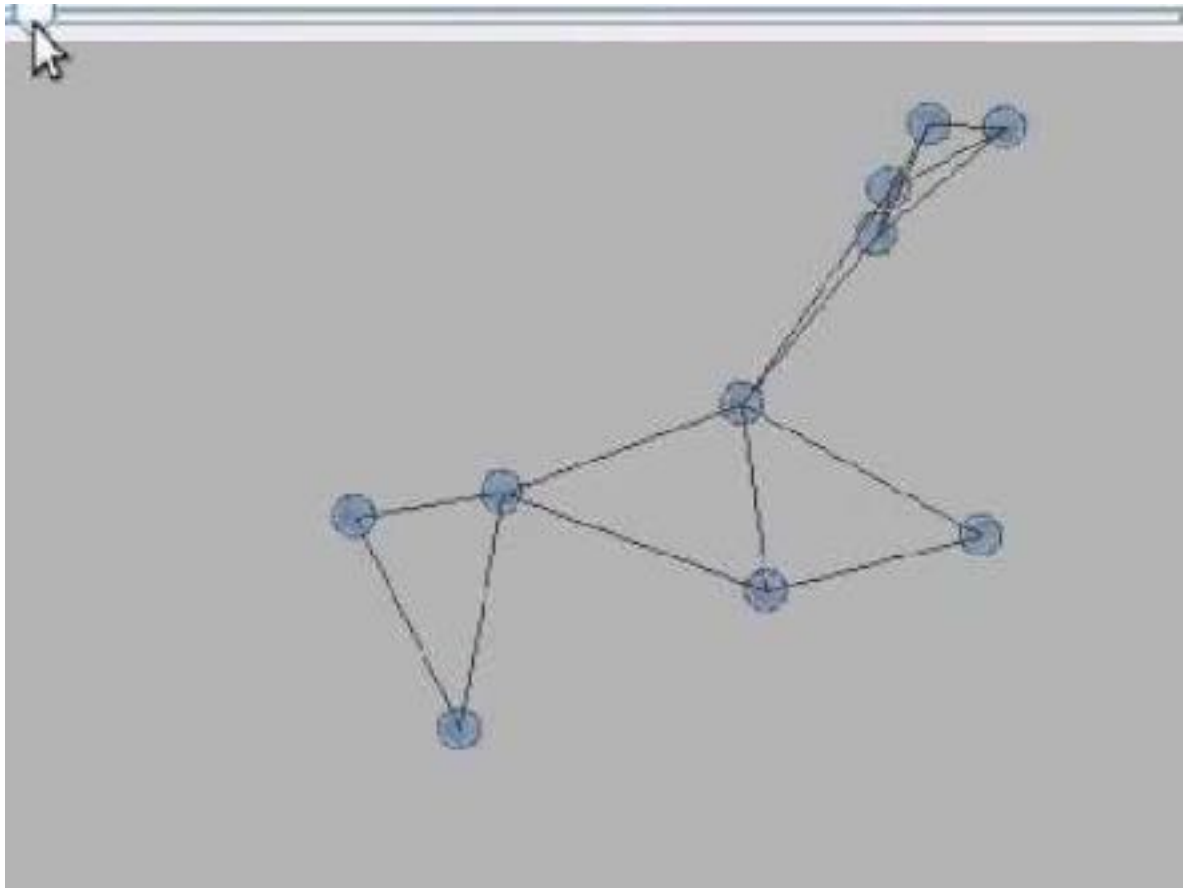
# JBotSim

```
import jbotsim.Node;

public class MovingNode extends Node{
    @Override
    public void onStart(){
        setDirection(Math.random()*2*Math.PI);
    }
    @Override
    public void onClock(){
        move();
        wrapLocation();
    }
}
```

```
public static void main(String[] args) {
    Topology tp = new Topology(400, 300);
    tp.setDefaultNodeModel(MovingNode.class);
    new JViewer(tp);
}
```

# JBotSim



# JBotSim

**Sending messages:**

Sending a message can be done by calling one of the parent methods send() or sendAll().

Two parameters are required for the send() method : the destination of the message (typically, another Node), and the message itself (of type Message).

If the sendAll() primitive is used, the message will be delivered to all the neighbors of this node (at the beginning of the next round)

# JBotSim

**Receiving messages:**

The simplest way to receive messages is to overrides the onMessage() method, as in this example

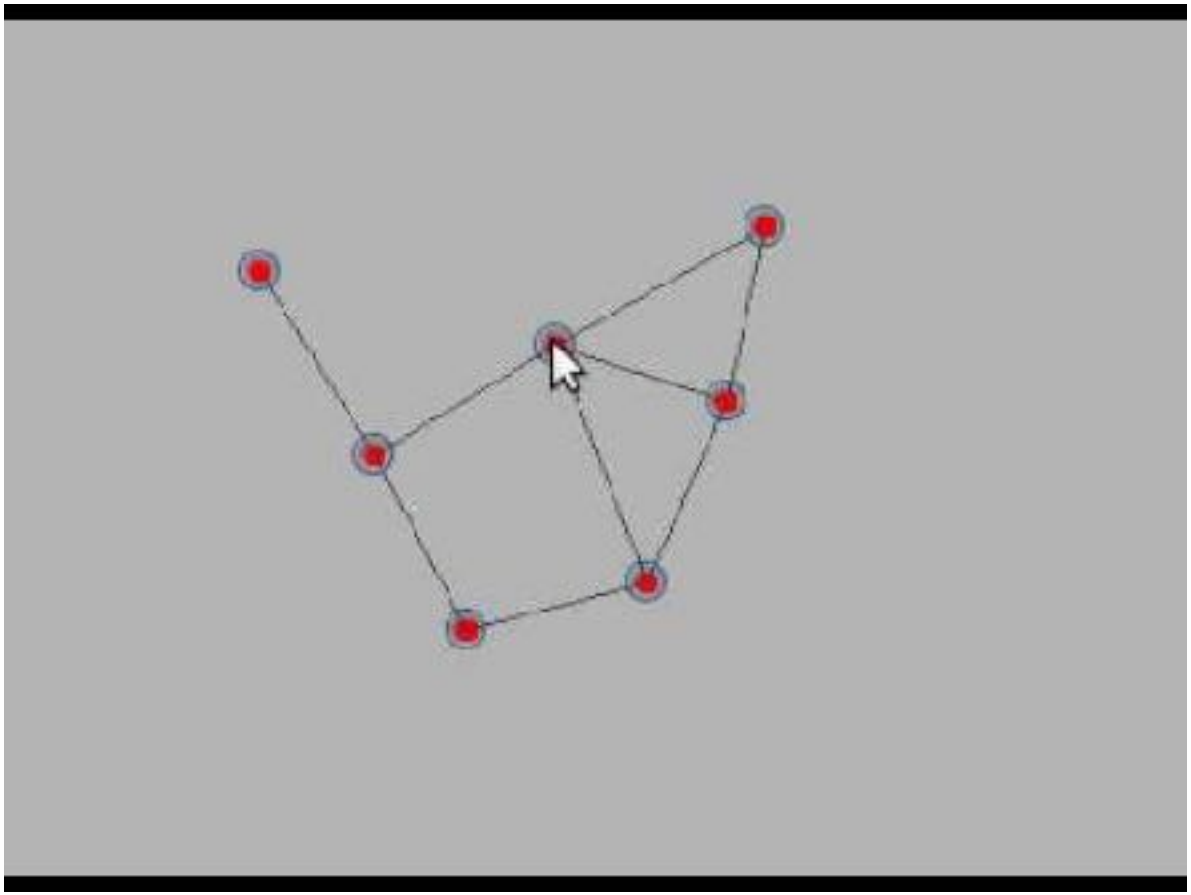
By default, a message sent in round r is delivered at the beginning of round r+1

# JBotSim

```
import jbotsim.Message;
import jbotsim.Node;
import java.awt.*;

public class LonerMessageBased extends Node {
    boolean mayBeAlone = true;
    @Override
    public void onClock(){
        setColor(mayBeAlone? Color.green:Color.red);
        sendAll(new Message());
        mayBeAlone = true;
    }
    @Override
    public void onMessage(Message msg) {
        mayBeAlone = false;
    }
}
```

# JBotSim



# Mobility Models

# Random Waypoint

The movement of nodes is governed in the following manner:

Each node does forever:

Selects a random destination in the simulation area and moves to this destination at constant speed.