MPEG@ITEC

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

# An Introduction to OMNeT++

based on Documention from http://www.omnetpp.org/

Christian Timmerer and Mathias Lux

Klagenfurt University, Dept. of Information Technology (ITEC)

October 2007

# Organisatorisches

- Informationstechnik $\wedge\neg$ Betriebssysteme
  - Einführung in die Welt der Prozesse & Threads
  - Mittwoch (10.10. 2007) HS 4, 10-12 Uhr

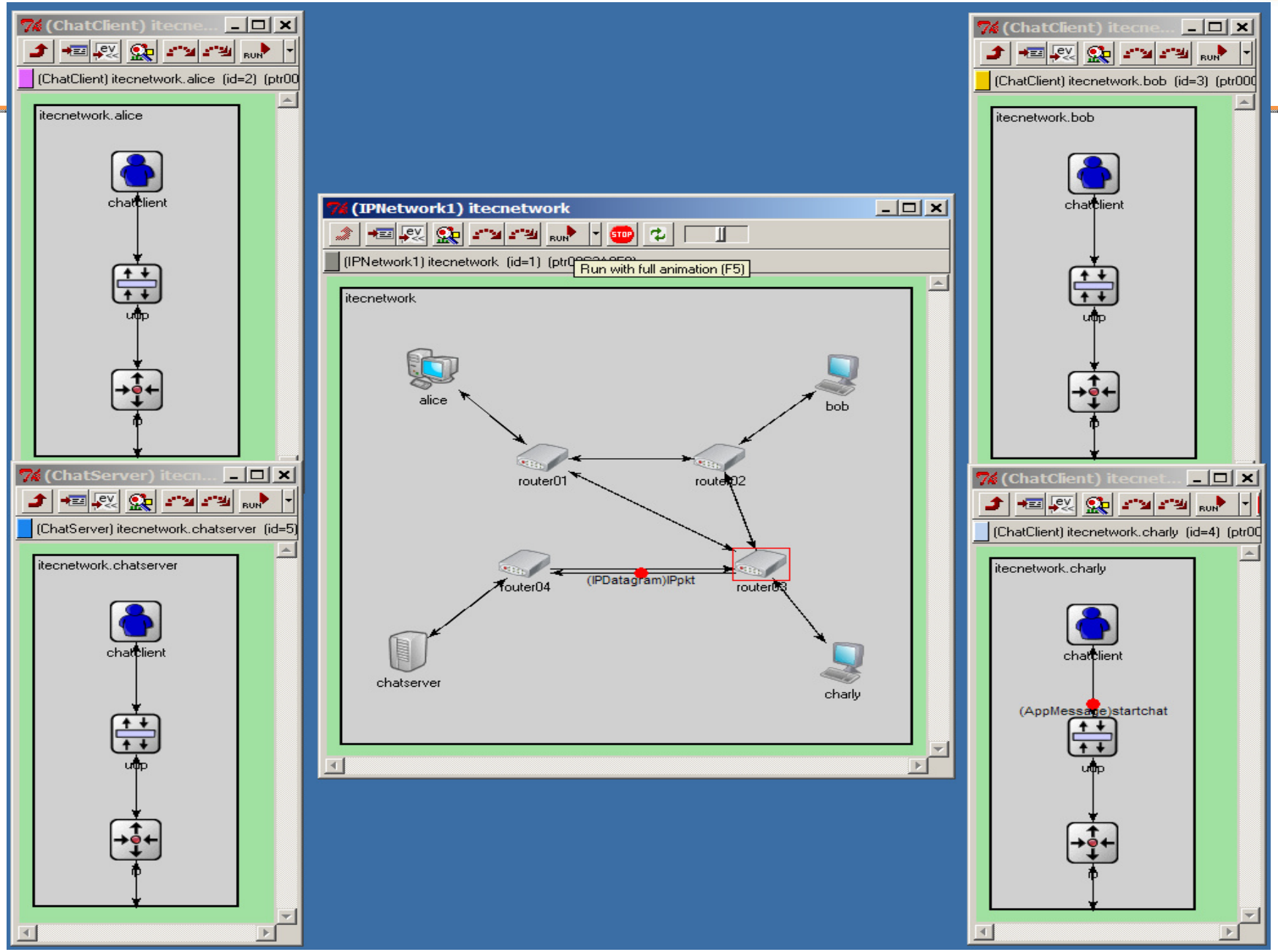
ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

# Outline

- Introduction
- OMNet++ in a Nutshell
  - What does OMNeT++ provide then?
  - OK. What does an OMNeT++ simulation model look like?
  - How do I run this thing?
  - OK! Now, how do I program a model in C++?
- Working with OMNet++: Flow Chart
- TicToc Tutorial
  - Getting started
  - Enhancing the 2-node TicToc
  - Turning into a real network
  - Adding statistics collection
  - Visualizing the results with Plove and Scalars
- Conclusions

# Introduction

- Discrete event simulator
  - Hierarchically nested modules
  - Modules communicate using messages through channels
- Written in C++
  - Source code publicly available
  - Simulation model for Internet, IPv6, Mobility, etc. available
- Free for academic use
  - Commercial version of OMNEST™
- Pros
  - Well structured, highly modular, not limited to network protocol simulations (e.g., like ns2)
- Cons
  - Relatively young and only few simulation models


(ChatClient) itecne...
(ChatClient) itecnetwork.alice (id=2) (ptr00
itecnetwork.alice
chatClient
udp
ip
(ChatClient) itecne...
(ChatClient) itecnetwork.bob (id=3) (ptr000
itecnetwork.bob
chatClient
udp
ip
(IPNetwork1) itecnetwork
(IPNetwork1) itecnetwork (id=1) (ptr
Run with full animation (F5)
itecnetwork
alice
bob
router01
router02
router04
(IPDatagram)IPpkt
router03
chatserver
charly
(ChatServer) itecn...
(ChatServer) itecnetwork.chatserver (id=5)
itecnetwork.chatserver
chatClient
udp
ip
(ChatClient) itecnet...
(ChatClient) itecnetwork.charly (id=4) (ptr00
itecnetwork.charly
chatClient
(AppMessage)startchat
udp
ip

# OMNet++ in a Nutshell:
# What does OMNeT++ provide then?

- C++ class library
  - Simulation kernel
  - Utility classes (for random number generation, statistics collection, topology discovery etc.)
  - ⇨ use to create simulation components (simple modules and channels)
- 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
- Tools to document large simulation models
  - opp_neddoc
- Extension interfaces for real-time simulation, emulation, MRIP, parallel distributed simulation, database connectivity and so on

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

OMNet++ in a Nutshell:

# OK. What does an OMNeT++ simulation model look like?

- OMNet++ provides component-based architecture
- Models assembled from reusable components = modules (can be combined in various way like LEGO blocks)
- Modules can be connected with each other through gates, and combined to form compound modules

```
// Ethernet CSMA/CD MAC
simple EtherMAC
  parameters:
    address : string; // others omitted for brevity
  gates:
    in: phyIn; // to physical layer or the network
    out: phyOut; // to physical layer or the network
    in: llcIn; // to EtherLLC or higher layer
    out: llcOut; // to EtherLLC or higher layer
endsimple
```

```
// Host with an Ethernet interface
module EtherStation
 parameters: ...
  gates: ...
    in: in; // for connecting to switch/hub, etc
    out: out;
  submodules:
    app: EtherTrafficGen;
    llc: EtherLLC;
    mac: EtherMAC;
  connections:
    app.out --> llc.hlIn;
    app.in <-- llc.hlOut;
    llc.macIn <-- mac.llcOut;
    llc.macOout --> mac.llcIn;
    mac.phyIn <-- in;
    mac.phyOut --> out;
endmodule
```

# OMNet++ in a Nutshell:
# How do I run this thing?

- Building the simulation program
  - `opp_makemake -N; make`
  - `-N` does not compile NED files into C++; are loaded dynamically
  - `-I` required in case subdirectories are used
- Run executable
  - Edit `omnetpp.ini` and start `./<exec>`
  - ./<exec> -f <name1>.ini –f <name2.ini>
- By default, graphical user interface, Tkenv
  - `-u Tkenv` oder `-u Cmdenv`

```
[General]
 preload-ned-files = *.ned
 network = etherLAN

[Parameters]
 *.numStations = 20
 **.frameLength = normal(200,1400)
 **.station[0].numFramesToSend = 5000
 **.station[1-5].numFramesToSend = 1000
 **.station[*].numFramesToSend = 0
```

# OMNet++ in a Nutshell:
# OK! Now, how do I program a model in C++?

- Simple modules are C++ classes
  - Subclass from cSimpleModule
  - Redefine a few virtual member functions
  - Register the new class with OMNet++ via Define_Module() macro
- Modules communicate via messages (also timers=timeout are self-messages)
  - cMessage or subclass thereof
  - Messages are delivered to handleMessage(cMessage *msg)
  - Send messages to other modules: send(cMessage *msg, const char *outGateName)
  - Self-messages (i.e., timers can be scheduled): scheduleAt(simtime_t time, cMessage *msg) and cancelEvent(cMessage *msg) to cancel the timer

# OMNet++ in a Nutshell:
# OK! Now, how do I program a model in C++? (cont'd)

- cMessage data members
  - name, length, kind, etc.
  - encapsulate(cMessage *msg), decapsulate() to facilitate protocol simulations
- .msg file for user-defined message types
  - OMNet++ opp_msgc to generate C++ class: _m.h and _m.cc
  - Support inheritance, composition, array members, etc.

```
message NetworkPacket {
 fields:
   int srcAddr;
   int destAddr;
}
```

OMNet++ in a Nutshell:

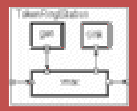
# OK! Now, how do I program a model in C++? (cont'd)

- To simulate network protocol stacks, control_info may carry auxiliary information to facilitate communication between protocol layers
  - cMessage's setControlInfo(cPolymorpic *ctrl) and removeControlInfo()
- Other cSimpleModule virtual member functions
  - initialize(), finish(), handleParameterChange(const char *paramName)
- Reading NED parameters
  - par(const char *paramName)
  - Typically done in initialize() and values are stored in data members of the module class
  - Returns cPar object which can be casted to the appropriate type (long, double, int, etc.) or by calling the object's doubleValue() etc. methods
  - Support for XML: DOM-like object tree

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

# Working with OMNet++: Flow Chart

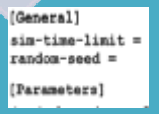
1. An OMNeT++ model is build from components (modules) which communicate by exchanging messages. Modules can be nested, that is, several modules can be grouped together to form a compound module. When creating the model, you need to map your system into a hierarchy of communicating modules.



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++.

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


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.

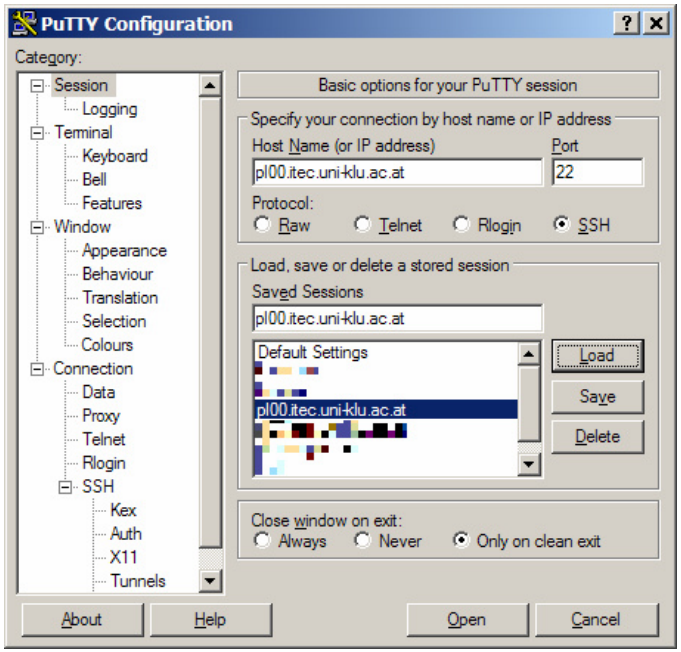
ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

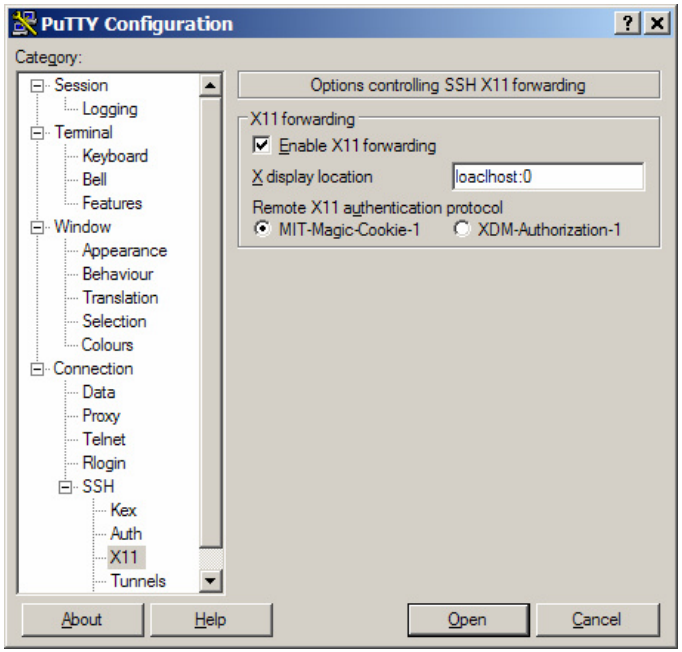
# Using OMNeT++ in our course

- OMNeT++ is installed at pl[00-03].itec.uni-klu.ac.at.
  - Connect there using **SSH**
    - PuTTY on Windows: http://www.chiark.greenend.org.uk/~sgtatham/putty/
  - Use your Uni-Klu username & password
  - Copy files using **SCP**
    - WinSCP on Windows: http://winscp.net

# Using OMNeT++ in our course

- Displaying OMNeT++ Tkenv locally
  - Install XMing on Windows and run it
    - http://sourceforge.net/projects/xming
  - Configure PuTTY the 'right' way



ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

# Remote X11 using Linux ...

- Enable X11 forwarding with
  - xhost +
- Connect with
  - ssh –X –u <user> pl00.itec.uni-klu.ac.at
- Retrieve and edit files with Nautilus or Konqueror
  - <ctrl>-L and “ssh://pl00.itec.uni-klu.ac.at”
  - enter fish://pl00.itec.uni-klu.ac.at

# TicToc Tutorial

- Short tutorial to OMNeT++ guides you through an example of modeling and simulation
- Showing you along the way some of the commonly used OMNeT++ features
- Based on the Tictoc example simulation: samples/tictoc – much more useful if you actually carry out at least the first steps described here
- http://www.omnetpp.org/doc/tictoc-tutorial/

# Getting Started

- Starting point
  - "network" that consists of two nodes (cf. simulation of telecommunications network)
  - One node will create a message and the two nodes will keep passing the same packet back and forth
  - Nodes are called “tic” and “toc”
- Here are the steps you take to implement your first simulation from scratch:
  1. Create a working directory called tictoc, and cd to this directory
  2. Describe your example network by creating a topology file
  3. Implement the functionality
  4. Create the Makefile
  5. Compile and link the simulation
  6. Create the omnetpp.ini file
  7. Start the executable using graphical user interface
  8. Press the Run button on the toolbar to start the simulation
  9. You can play with slowing down the animation or making it faster
  10. You can exit the simulation program …

# Describe your Example Network by creating a Topology File

```
// This file is part of an OMNeT++/OMNEST simulation example.
// Copyright (C) 2003 Ahmet Sekercioglu
// Copyright (C) 2003-2005 Andras Varga
// This file is distributed WITHOUT ANY WARRANTY. See the file
// `license' for details on this and other legal matters.

simple Txc1
  gates:
    in: in;
    out: out;
endsimple

// Two instances (tic and toc) of Txc1 connected both ways.
// Tic and toc will pass messages to one another.

module Tictoc1
  submodules:
    tic: Txc1;
    toc: Txc1;
  connections:
    tic.out --> delay 100ms --> toc.in;
    tic.in <-- delay 100ms <-- toc.out;
endmodule

network tictoc1 : Tictoc1
endnetwork
```

Txc1 is a simple module type, i.e., atomic on NED level and implemented in C++.
Txc1 has one input gate named in, and one output gate named out .

Tictoc1 is a compound module assembled from two sub-modules tic and toc (instances from a module type called Txc1).
Connection: tic's output gate (out) to toc's input gate and vice version w/ propagation delay 100ms.

Define a network tictoc1 which is an instance of the module type TicToc1.

# Implement the Functionality

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

class Txc1 : public cSimpleModule {
  protected:  // The following redefined virtual function holds the algorithm.
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc1); // The module class needs to be registered with OMNeT++

void Txc1::initialize() {
  // Initialize is called at the beginning of the simulation. To bootstrap the tic-toc-tic-toc process, one of the modules needs
  // to send the first message. Let this be `tic'. Am I Tic or Toc?
  if (strcmp("tic", name()) == 0) {
    // create and send first message on gate "out". "tictocMsg" is an arbitrary string which will be the name of the message
object.
    cMessage *msg = new cMessage("tictocMsg");
    send(msg, "out");
  }
}
void Txc1::handleMessage(cMessage *msg) {
  // The handleMessage() method is called whenever a message arrives at the module.
  // Here, we just send it to the other module, through gate `out'.
  // Because both `tic' and `toc' does the same, the message will bounce between the two.
  send(msg, "out");
}
```

Txc1 simple module represented as C++ class Txc1; subclass from cSimpleModule.

Need to redefine two methods from cSimpleModule: initialize() and handleMessage().

Create cMessage and send it to out on gate `out`.

Send it back.

# Create the Makefile, Compile, and Link the Simulation

- Create Makefile: `opp_makemake`
  - This command should have now created a Makefile in the working directory tictoc.
- Compile and link the simulation: `make`

If you start the executable now, it will complain that it cannot find the file `omnetpp.ini`

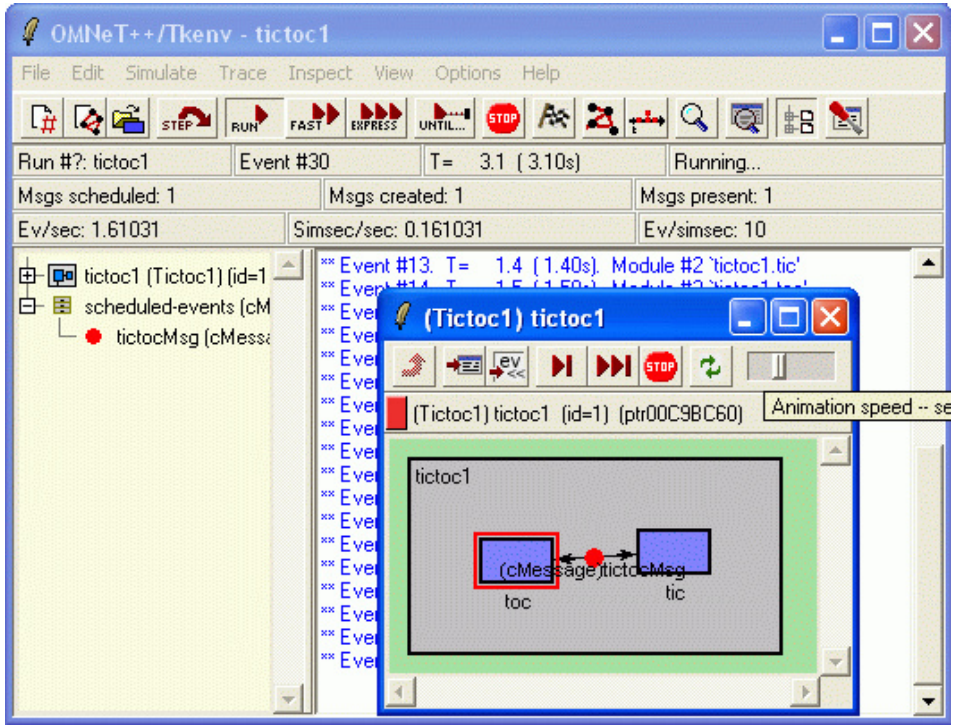
# Create the omnetpp.ini File

- … tells the simulation program which network you want to simulate
- … you can pass parameters to the model
- … explicitly specify seeds for the random number generators etc.

```
# This file is shared by all tictoc simulations.
# Lines beginning with `#' are comments

[General]
  preload-ned-files = *.ned
  network = tictoc1 # this line is for Cmdenv, Tkenv will still let you choose from a dialog

[Parameters]
  tictoc4.toc.limit = 5
  # argument to exponential() is the mean; truncnormal() returns values from
  # the normal distribution truncated to nonnegative values
  tictoc6.tic.delayTime = exponential(3)
  tictoc6.toc.delayTime = truncnormal(3,1)
```

# Start the Executable, Run, Play, Exit

- `./tictoc` shall start the simulation environment
- Hit Run button to start the actual simulation
- Play around, e.g., fast forward, stop, etc.
- Hit File->Exit to close the simulation

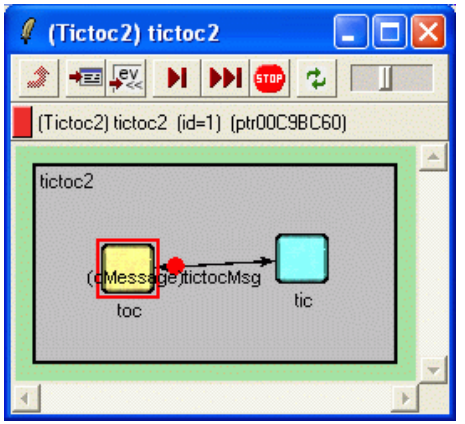


# Enhancing the 2-node TicToc

- Step 2: Refining the graphics, and adding debugging output
- Step 3: Adding state variables
- Step 4: Adding parameters
- Step 5: Modeling processing delay
- Step 6: Random numbers and parameters
- Step 7: Timeout, cancelling timers
- Step 8: Retransmitting the same message

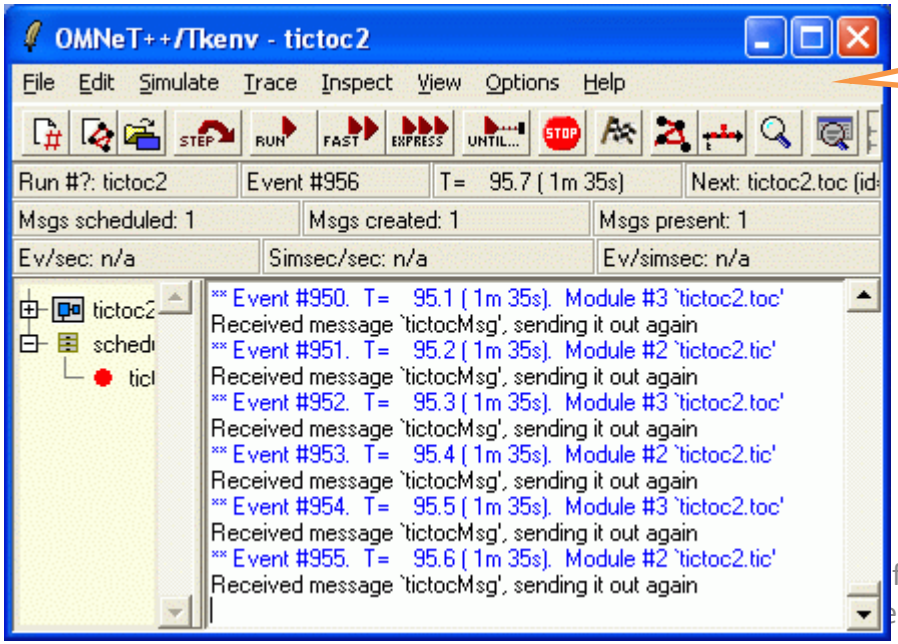
# Step 2: Refining the Graphics, and Adding Debugging Output

```
submodules:
  tic: Txc2;
      display: "i=block/process,cyan"; // Note "i=<icon>,<color>"
  toc: Txc2;
      display: "i=block/process,gold"; // Here too
connections:
```

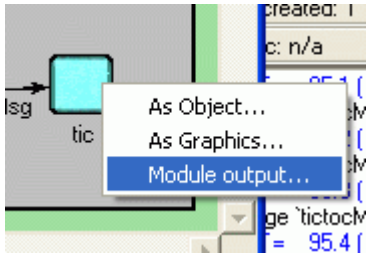



## Debug messages

- ev << "Sending initial message\n";
- ev << "Received message `" << msg->name() << "', sending it out again\n";

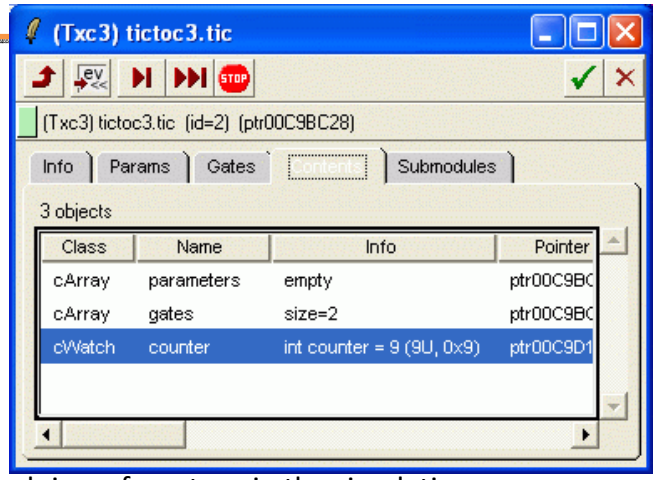



Run the simulation in the OMNeT++ GUI Tkenv.




Open separate output windows for tic and toc by right-clicking on their icons and choosing Module output from the menu – useful for large models.

# Step 3: Adding State Variables




```
#include <stdio.h> #include <string.h> #include <omnetpp.h>
class Txc3 : public cSimpleModule {
 private:
  int counter; // Note the counter here
 protected:
  virtual void initialize();
  virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc3);
void Txc3::initialize() {
 counter = 10;
 /* The WATCH() statement below will let you examine the variable under Tkenv. After doing a few steps in the simulation,
double-click either `tic' or `toc', select the Contents tab in the dialog that pops up, and you'll find "counter" in the list. */
 WATCH(counter);
 if (strcmp("tic", name()) == 0) {
  ev << "Sending initial message\n“;
  cMessage *msg = new cMessage("tictocMsg");
  send(msg, "out");
 }
}
void Txc3::handleMessage(cMessage *msg) {
 counter--; // Increment counter and check value.
 if (counter==0) { /* If counter is zero, delete message. If you run the model, you'll find that the simulation will stop at this point
with the message "no more events“. */
  ev << name() << "'s counter reached zero, deleting message\n“;
  delete msg;
 } else { ev << name() << "'s counter is " << counter << ", sending back message\n"; send(msg, "out"); }
}
```

*We'll turn the "magic number" 10 into a parameter*

# Step 4: Adding Parameters

- Module parameters have to be declared in the NED file: numeric, string, bool, or xml

```
simple Txc4
  parameters:
    limit: numeric const; // declare the new parameter
  gates:
```

- Modify the C++ code to read the parameter in initialize(), and assign it to the counter.

```
counter = par("limit");
```

- Now, we can assign the parameters in the NED file or from omnetpp.ini. Assignments in the NED file take precedence.

```
module Tictoc4
  submodules:
    tic: Txc4;
      parameters:
        limit = 8; // tic's limit is 8
      display: "i=block/process,cyan";
    toc: Txc4; // note that we leave toc's limit unbound here
      display: "i=block/process,gold";
  connections:
```

# Step 4: Adding Parameters (cont'd)

- … and the other in omnetpp.ini:
  - tictoc4.toc.limit = 5
- Note that because omnetpp.ini supports wildcards, and parameters assigned from NED files take precedence over the ones in omnetpp.ini, we could have used
  - tictoc4.t*c.limit = 5
- or
  - tictoc4.*.limit = 5
- or even
  - **.limit = 5
- with the same effect. (The difference between * and ** is that * will not match a dot and ** will.)

# Step 5: Modeling Processing Delay

- Hold messages for some time before sending it back ⇨ timing is achieved by the module sending a message to itself (i.e., self-messages)

```
class Txc5 : public cSimpleModule {
  private:
    // pointer to the event object which we'll use for timing
    cMessage *event;
    // variable to remember the message until we send it back
    cMessage *tictocMsg;
  public:
```

- We "send" the self-messages with the scheduleAt() function

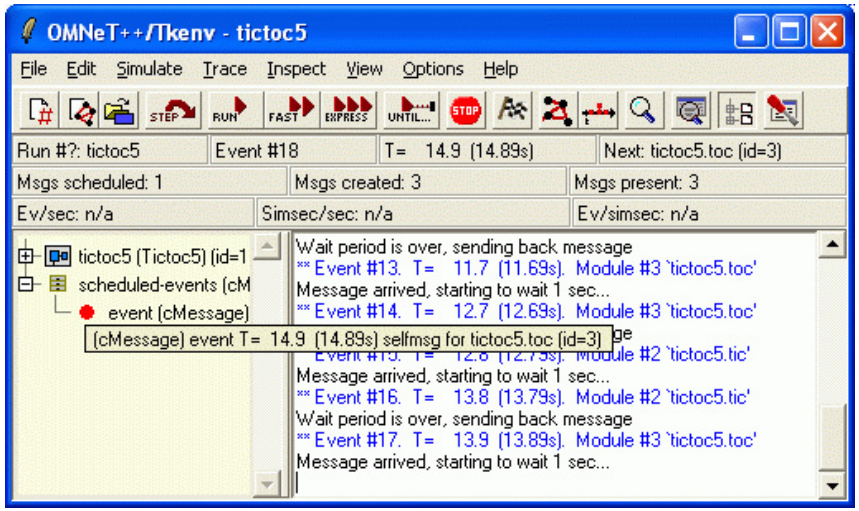
```
scheduleAt(simTime()+1.0, event);
```

- handleMessage(): differentiate whether a new message has arrived via the input gate or the self-message came back (timer expired)

```
if (msg==event) or if (msg->isSelfMessage())
```

# Step 5: Modeling Processing Delay (cont'd)

```
void Txc5::handleMessage(cMessage *msg) {
  // There are several ways of distinguishing messages, for example by message
  // kind (an int attribute of cMessage) or by class using dynamic_cast
  // (provided you subclass from cMessage). In this code we just check if we
  // recognize the pointer, which (if feasible) is the easiest and fastest
  // method.
  if (msg==event) {
    // The self-message arrived, so we can send out tictocMsg and NULL out
    // its pointer so that it doesn't confuse us later.
    ev << "Wait period is over, sending back message\n";
    send(tictocMsg, "out");
    tictocMsg = NULL;
  } else {
    // If the message we received is not our self-message, then it must
    // be the tic-toc message arriving from our partner. We remember its
    // pointer in the tictocMsg variable, then schedule our self-message
    // to come back to us in 1s simulated time.
    ev << "Message arrived, starting to wait 1 sec...\n";
    tictocMsg = msg;
    scheduleAt(simTime()+1.0, event);
  }
}
```



# Step 6: Random Numbers and Parameter

- Change the delay from 1s to a random value :

```
// The "delayTime" module parameter can be set to values like
// "exponential(5)" (tictoc6.ned, omnetpp.ini), and then here
// we'll get a different delay every time.
double delay = par("delayTime");

ev << "Message arrived, starting to wait " << delay << " secs...\n";
tictocMsg = msg;
scheduleAt(simTime()+delay, event);
```

- "Lose" (delete) the packet with a small (hardcoded) probability:

```
if (uniform(0,1) < 0.1) {
    ev << "\"Losing\" message\n";
    delete msg;
}
```

- Assign seed and the parameters in omnetpp.ini:

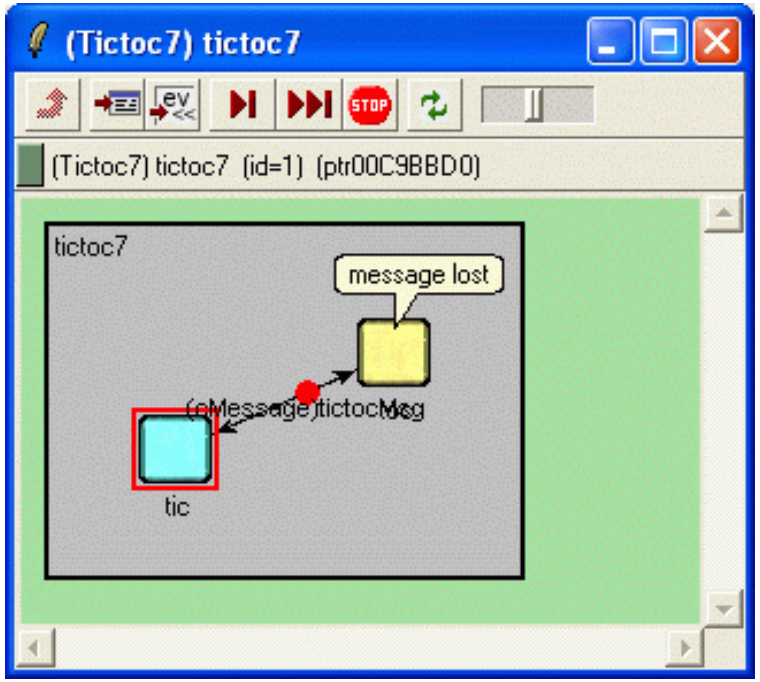
[General] seed-0-mt=532569 # or any other 32-bit value

tictoc6.tic.delayTime = exponential(3)
tictoc6.toc.delayTime = truncnormal(3,1)

# Step 7: Timeout, Cancelling Timers

- Transform our model into a **stop-and-wait simulation**

```
void Tic7::handleMessage(cMessage *msg) {
   if (msg==timeoutEvent) {
      // If we receive the timeout event, that means the packet hasn't
      // arrived in time and we have to re-send it.
      ev << "Timeout expired, resending message and restarting timer\n";
      cMessage *msg = new cMessage("tictocMsg");
      send(msg, "out");
      scheduleAt(simTime()+timeout, timeoutEvent);
   } else { // message arrived
      // Acknowledgement received -- delete the stored message
      // and cancel the timeout event.
      ev << "Timer cancelled.\n";
      cancelEvent(timeoutEvent);

      // Ready to send another one.
      cMessage *msg = new cMessage("tictocMsg");
      send(msg, "out");
      scheduleAt(simTime()+timeout, timeoutEvent);
   }
}
```



# Step 8: Retransmitting the Same Message

- Keep a copy of the original packet so that we can re-send it without the need to build it again:
  - Keep the original packet and send only copies of it
  - Delete the original when toc's acknowledgement arrives
  - Use message sequence number to verify the model
- generateNewMessage() and sendCopyOf()
  - avoid handleMessage() growing too large

# Step 8: Retransmitting the Same Message (cont'd)

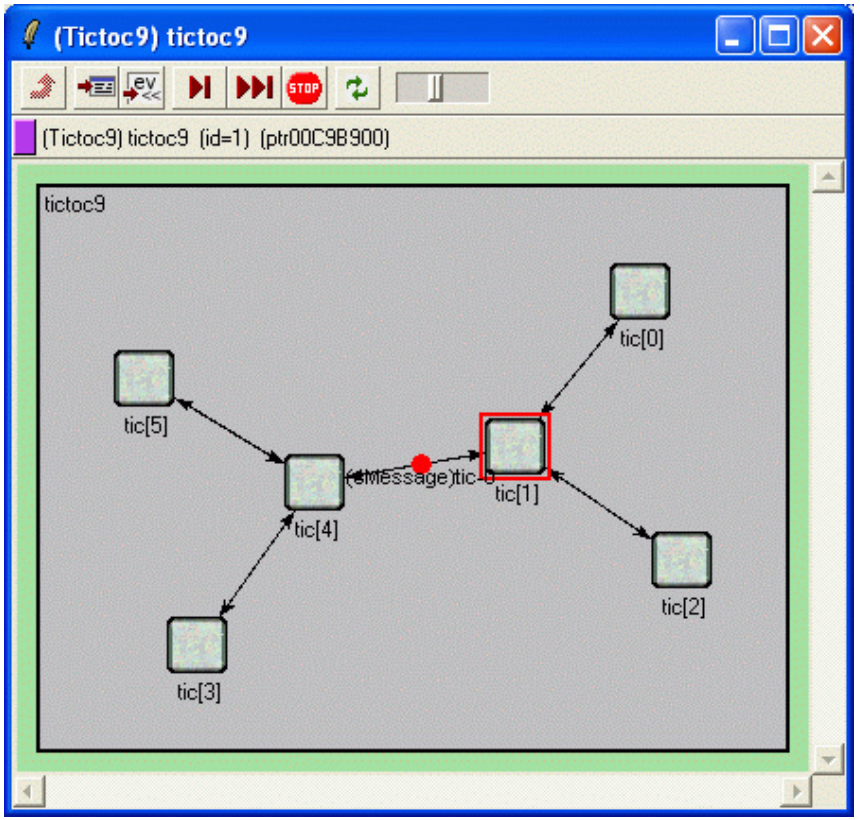
```
cMessage *Tic8::generateNewMessage() {
   // Generate a message with a different name every time.
   char msgname[20];
   sprintf(msgname, "tic-%d", ++seq);
   cMessage *msg = new cMessage(msgname);
   return msg;
}

void Tic8::sendCopyOf(cMessage *msg) {
   // Duplicate message and send the copy.
   cMessage *copy = (cMessage *) msg->dup();
   send(copy, "out");
}
```

# Turning it into a real network

- Step 9: More than two nodes
- Step 10: Defining our message class

# Step 9: More Than Two Nodes

```
simple Txc9
  gates:
    in: in[];  // declare in[] and out[] to be vector gates
    out: out[];
endsimple

module Tictoc9
  submodules:
    tic: Txc9[6];   // we'll have 6 Txc modules
      display: "i=block/process";
  connections:
    tic[0].out++ --> delay 100ms --> tic[1].in++;
    tic[0].in++ <-- delay 100ms <-- tic[1].out++;

    tic[1].out++ --> delay 100ms --> tic[2].in++;
    tic[1].in++ <-- delay 100ms <-- tic[2].out++;

    tic[1].out++ --> delay 100ms --> tic[4].in++;
    tic[1].in++ <-- delay 100ms <-- tic[4].out++;

    tic[3].out++ --> delay 100ms --> tic[4].in++;
    tic[3].in++ <-- delay 100ms <-- tic[4].out++;

    tic[4].out++ --> delay 100ms --> tic[5].in++;
    tic[4].in++ <-- delay 100ms <-- tic[5].out++;
endmodule
```



# Step 9: More Than Two Nodes (cont'd)

- tic[0] will generate the message to be sent around
  - done in initialize()
  - index() function which returns the index of the module
- forwardMessage(): invoke from handleMessage() whenever a message arrives

```
void Txc9::forwardMessage(cMessage *msg) {
    // In this example, we just pick a random gate to send it on.
    // We draw a random number between 0 and the size of gate `out[]'.
    int n = gate("out")->size();
    int k = intuniform(0,n-1);

    ev << "Forwarding message " << msg << " on port out[" << k << "]\n";
    send(msg, "out", k);
}
```

- When the message arrives at tic[3], its handleMessage() will delete the message

# Step 10: Defining our Message Class

- Destination address is no longer hardcoded tic[3] – add destination address to message
- Subclass cMessage: tictoc10.msg

```
message TicTocMsg10 {
    fields:
        int source;
        int destination;
        int hopCount = 0;
}
```

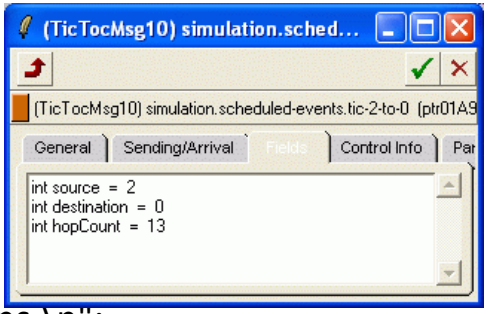
- opp_msgc is invoked and it generates tictoc10_m.h and tictoc10_m.cc
- Include tictoc10_m.h into our C++ code, and we can use TicTocMsg10 as any other class

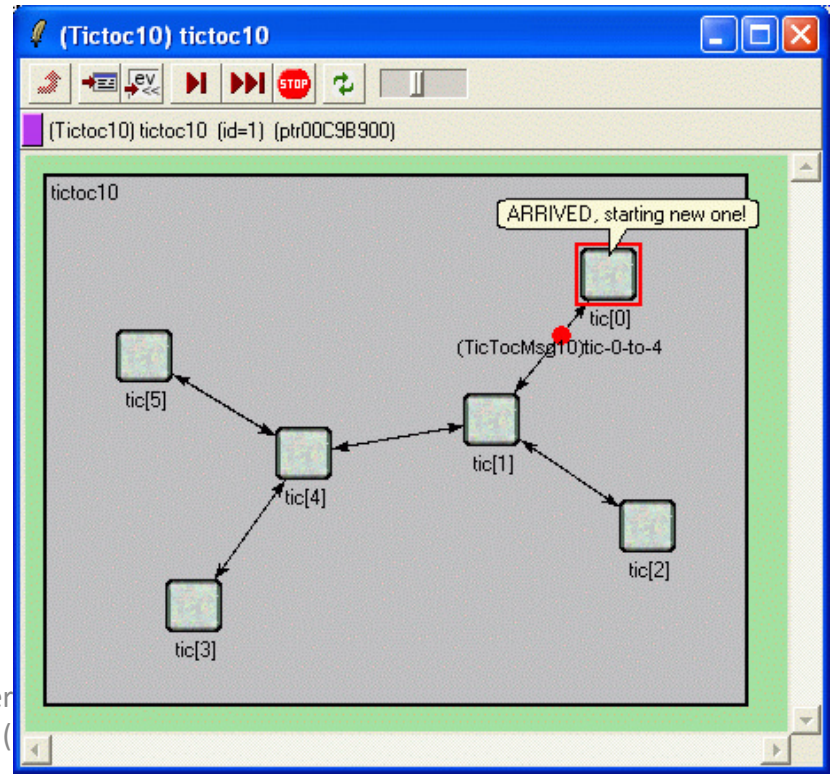
```
#include "tictoc10_m.h"

TicTocMsg10 *msg = new TicTocMsg10(msgname);
msg->setSource(src);
msg->setDestination(dest);
return msg;
```

# Step 10: Defining our Message Class (cont'd)

- Use dynamic cast instead of plain C-style cast ((TicTocMsg10 *)msg) which is not safe

```
void Txc10::handleMessage(cMessage *msg) {
  TicTocMsg10 *ttmsg = check_and_cast<TicTocMsg10 *>(msg);

  if (ttmsg->getDestination()==index()) {
    // Message arrived.
    ev << "Message " << ttmsg << " arrived after " << ttmsg->getHopCount() << " hops.\n";
    bubble("ARRIVED, starting new one!");
    delete ttmsg;

    // Generate another one.
    ev << "Generating another message: ";
    TicTocMsg10 *newmsg = generateMessage();
    ev << newmsg << endl;
    forwardMessage(newmsg);
  } else {
    // We need to forward the message.
    forwardMessage(ttmsg);
  }
}
```



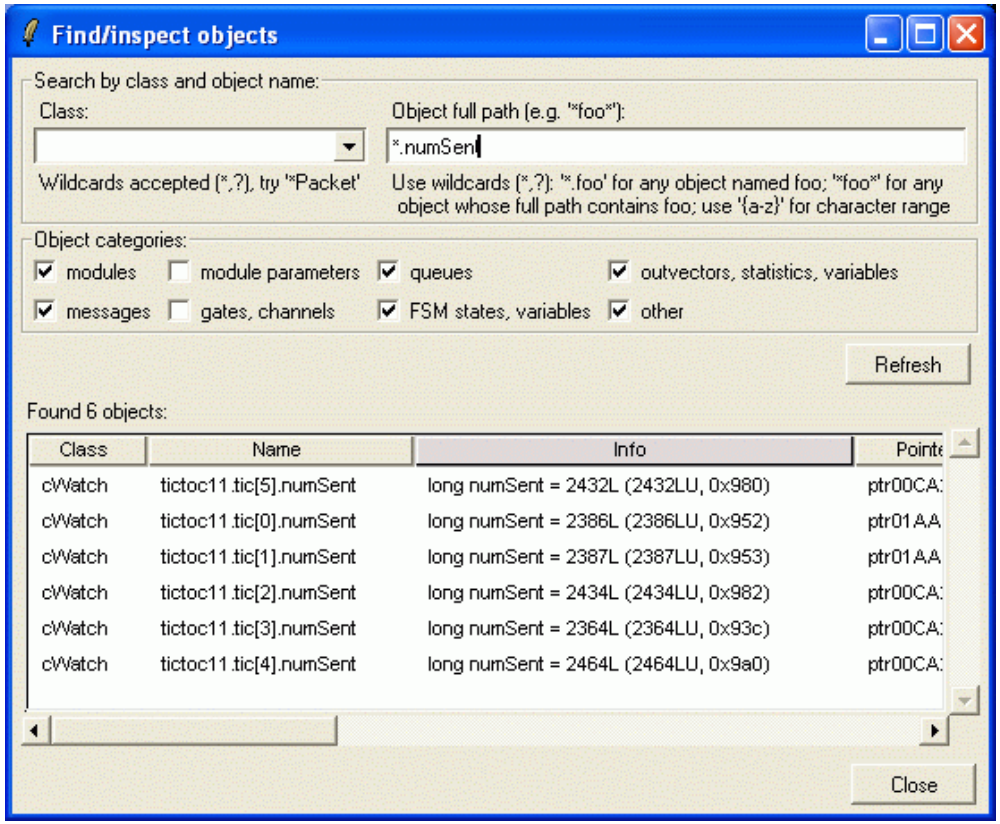



# Adding statistics collection

- Step 11: Displaying the number of packets sent/received
- Step 12: Adding statistics collection

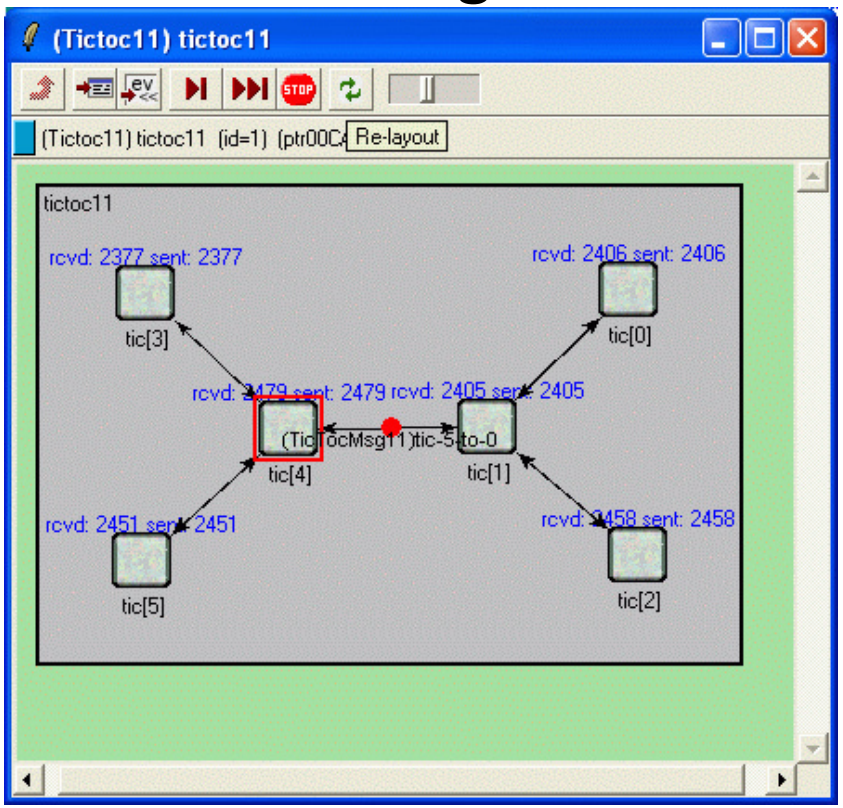
# Step 11: Displaying the Number of Packets Sent/Received

- Add two counters to the module class: numSent and numReceived
- Set to zero and WATCH'ed in the initialize() method
- Use the Find/inspect objects dialog (Inspect menu; it is also on the toolbar)



# Step 11: Displaying the Number of Packets Sent/Received (cont'd)

- This info appears above the module icons using the t= display string tag




```
if (ev.isGUI())
    updateDisplay();
```

```
void Txc11::updateDisplay(){
  char buf[40];
  sprintf(buf, "rcvd: %ld sent: %ld", numReceived, numSent);
  displayString().setTagArg("t",0,buf);
}
```

# Step 12: Adding statistics collection

- Example: average hopCount a message has to travel before reaching its destination
  - Record in the hop count of every message upon arrival into an output vector (a sequence of (time,value) pairs, sort of a time series)
  - Calculate mean, standard deviation, minimum, maximum values per node, and write them into a file at the end of the simulation
  - Use off-line tools to analyze the output files

# Step 12: Adding statistics collection (cont'd)

- Output vector object (which will record the data into omnetpp.vec) and a histogram object (which also calculates mean, etc)

```
class Txc12 : public cSimpleModule {
  private:
    long numSent;
    long numReceived;
    cLongHistogram hopCountStats;
    cOutVector hopCountVector;

  protected:
```

- Upon message arrival, update statistics within handleMessage()

```
hopCountVector.record(hopcount);
hopCountStats.collect(hopcount);
```

- hopCountVector.record() call writes the data into omnetpp.vec (will be deleted each time the simulation is restarted)

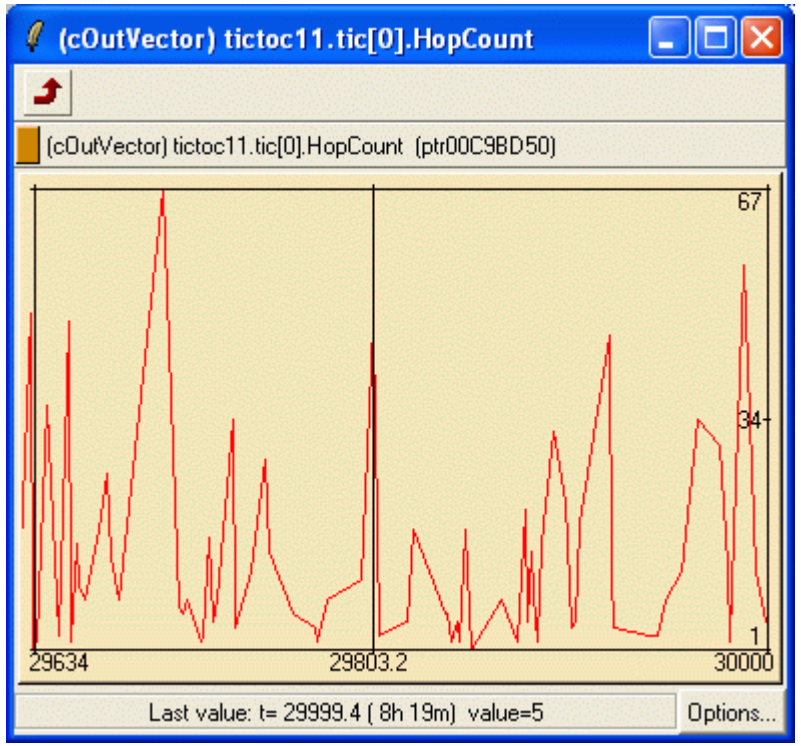
# Step 12: Adding statistics collection (cont'd)

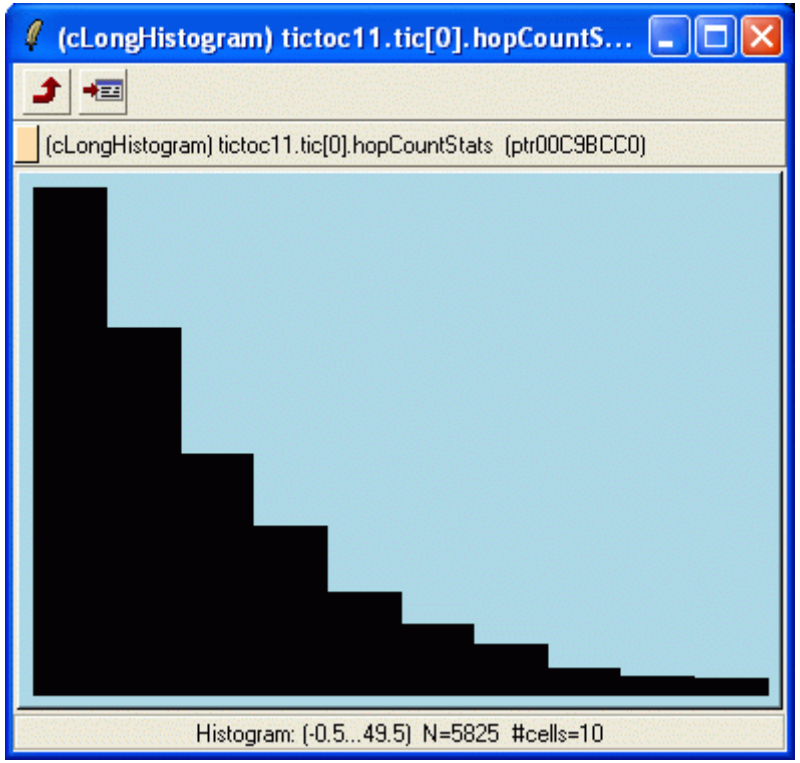
- Scalar data (histogram object) have to be recorded manually, in the finish() function

```
void Txc12::finish() {
    // This function is called by OMNeT++ at the end of the simulation.
    ev << "Sent:     " << numSent << endl;
    ev << "Received: " << numReceived << endl;
    ev << "Hop count, min:    " << hopCountStats.min() << endl;
    ev << "Hop count, max:    " << hopCountStats.max() << endl;
    ev << "Hop count, mean:   " << hopCountStats.mean() << endl;
    ev << "Hop count, stddev: " << hopCountStats.stddev() << endl;

    recordScalar("#sent", numSent);
    recordScalar("#received", numReceived);
    hopCountStats.recordScalar("hop count");
}
```

- recordScalar() calls in the code below write into the omnetpp.sca file
- omnetpp.sca not deleted between simulation runs; new data are just appended – allows to jointly analyze data from several simulation runs

# Step 12: Adding statistics collection (cont'd)

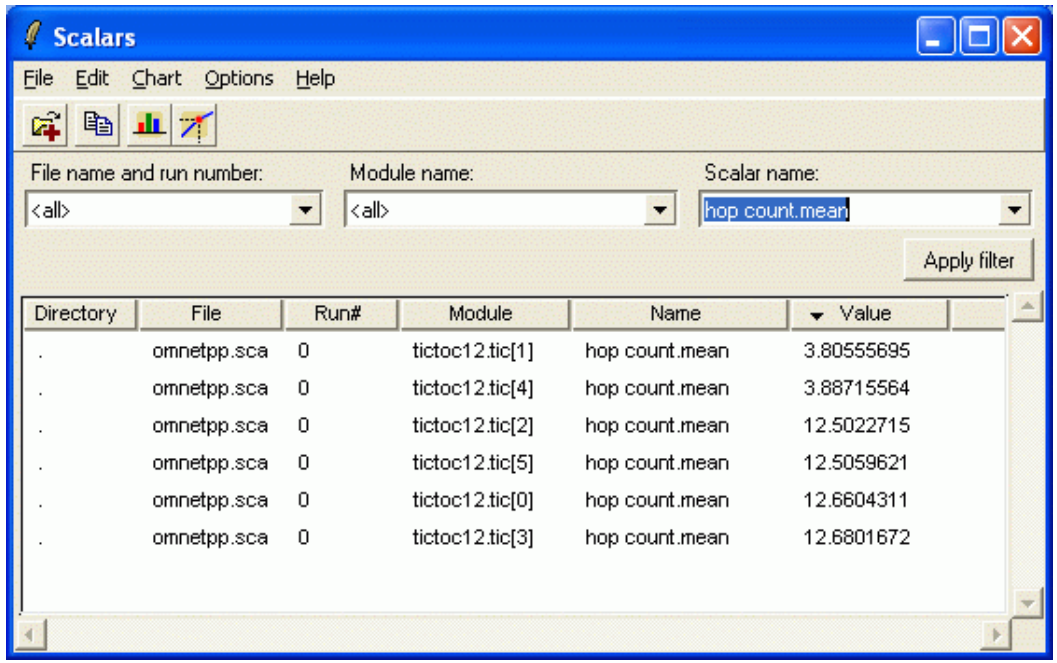


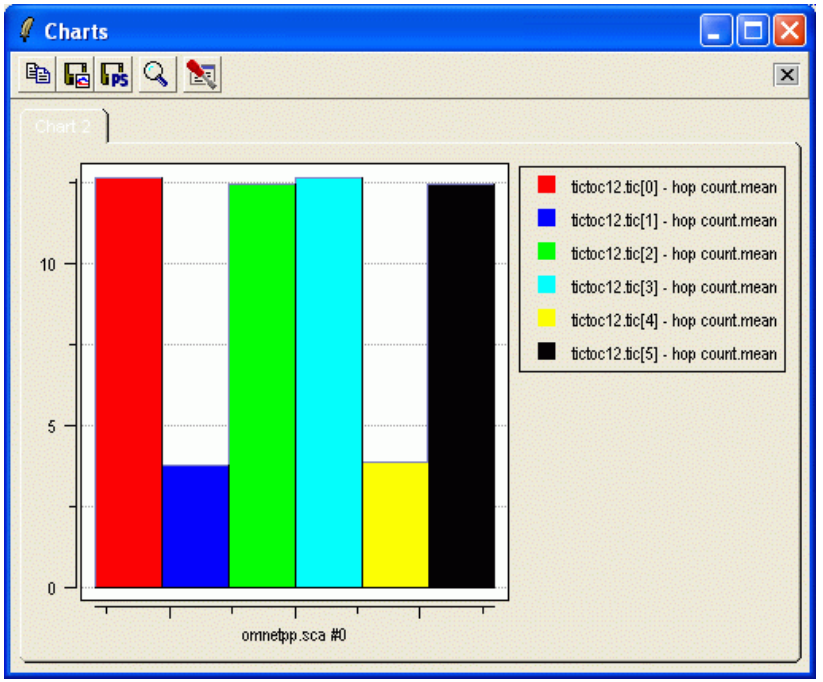



# Visualizing the results with Scalars

- Scalars tool can be used to visualize the contents of the omnetpp.sca file: bar charts, x-y plots (e.g. throughput vs. offered load), or export data

```
$ scalars omnetpp.sca
```



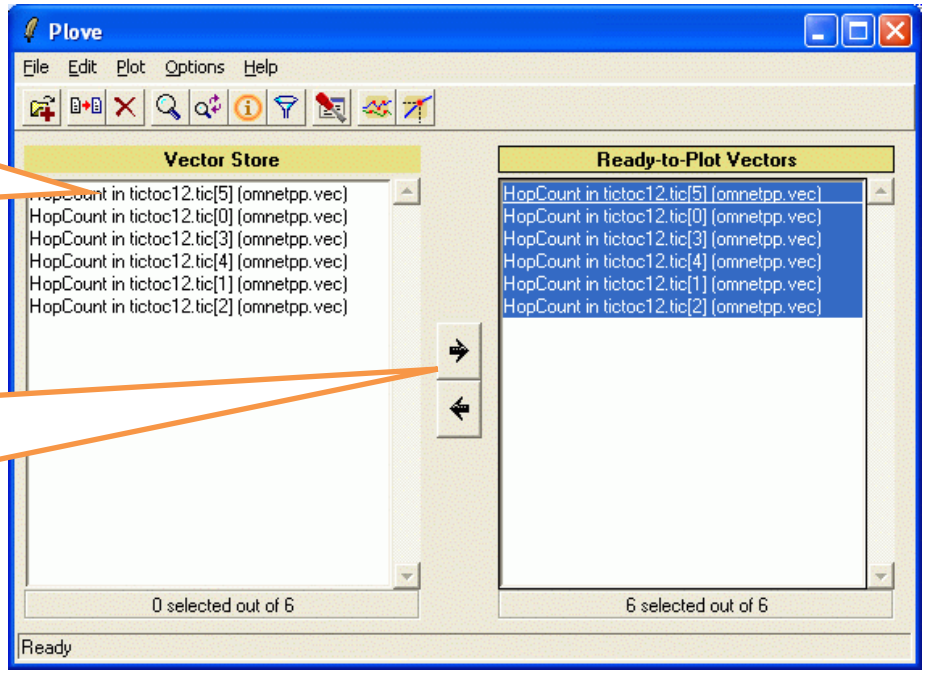



# Visualizing the results with Plove

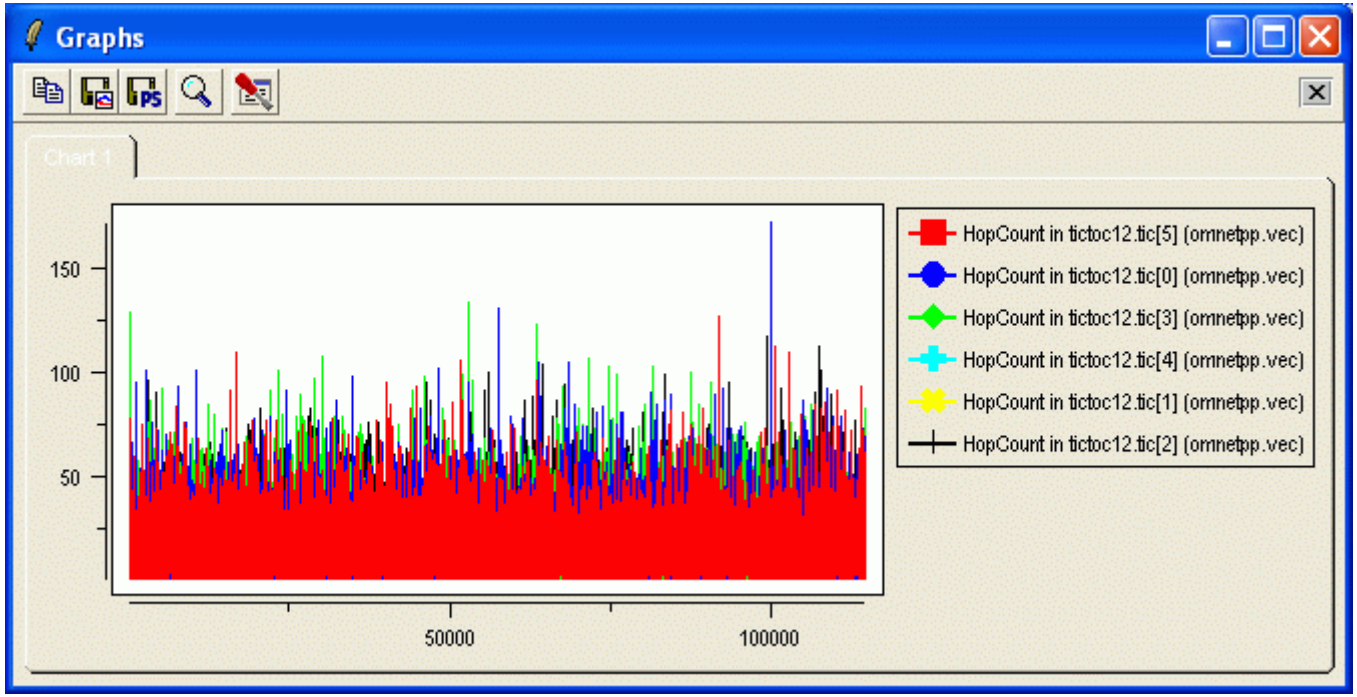
```
$ plove omnetpp.vec
```

The left pane displays vectors that are present in the omnetpp.vec file. (You can load further vector files as well.)

To plot, you have to copy some vectors to the right pane, select one or more of them (shift+click and ctrl+click works), and click the Plot icon on the toolbar.

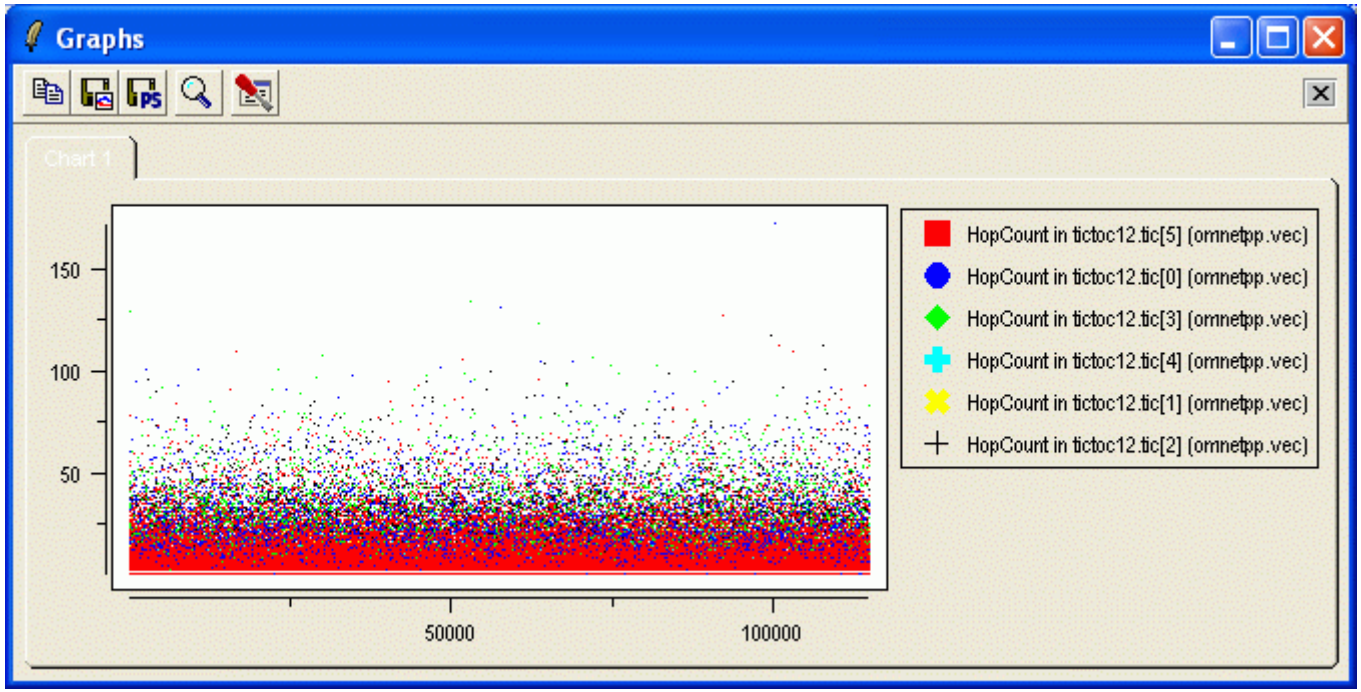


# Visualizing the results with Plove (cont'd)



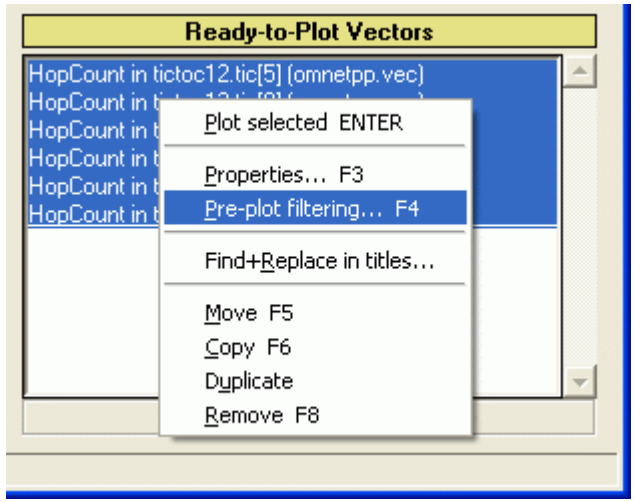
# Visualizing the results with Plove (cont'd)

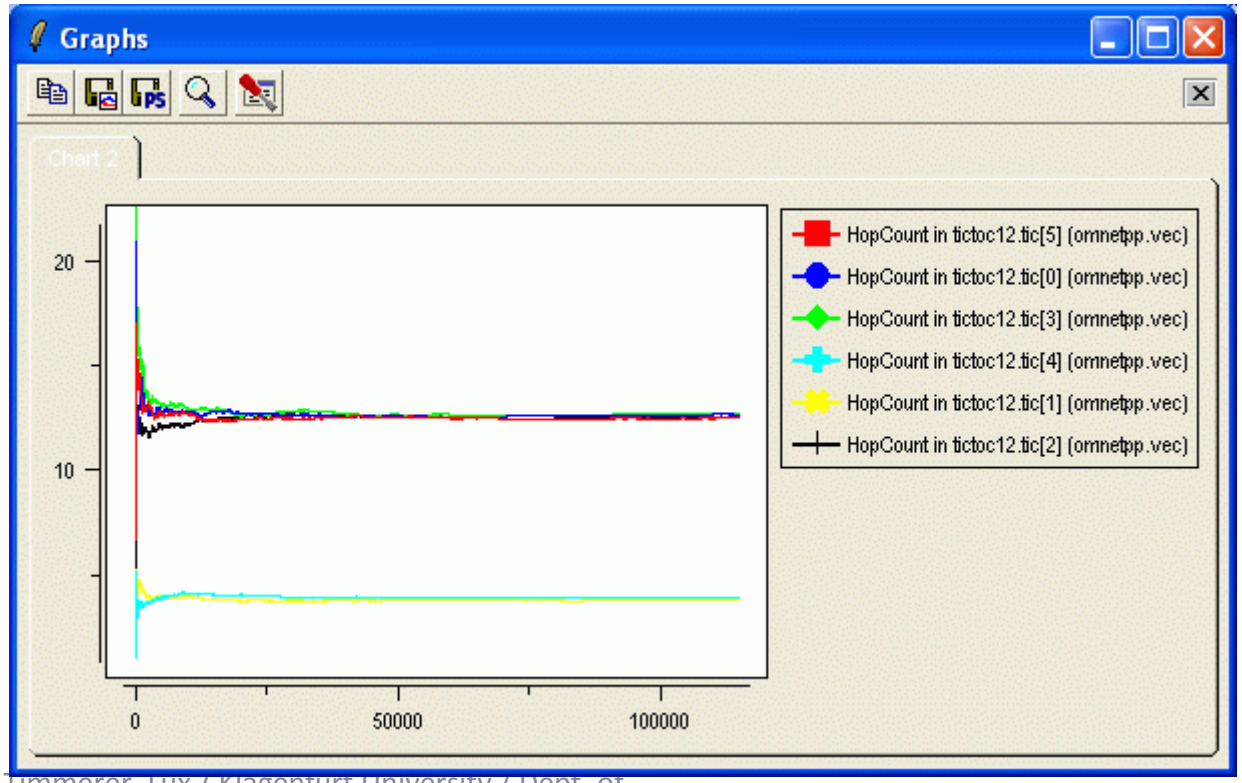
- Turned off connecting the data points



# Visualizing the results with Plove (cont'd)

- Apply a filter which plots mean on [0,t).



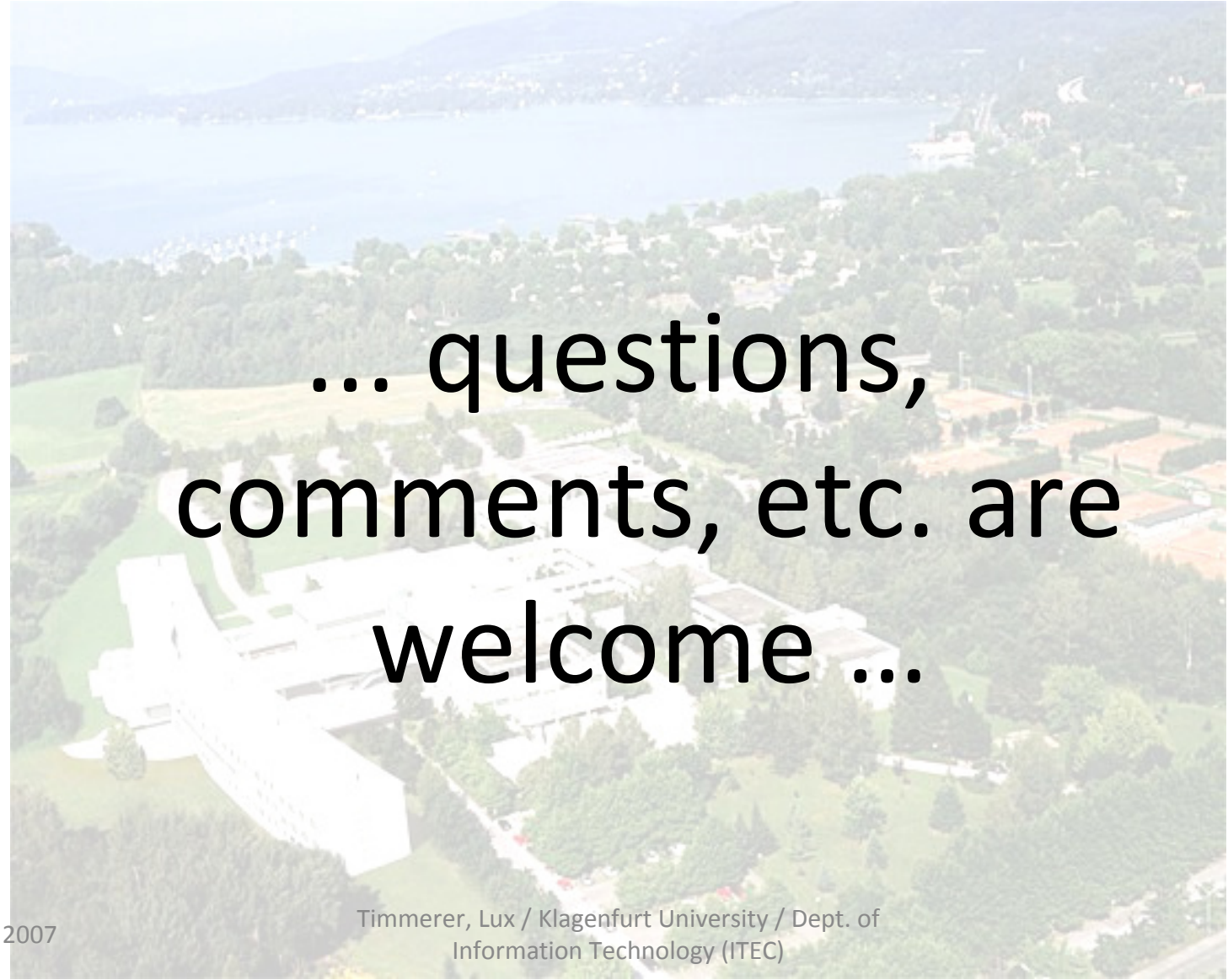



ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

# Conclusions

- OMNeT++: discrete event simulation system
- OMNeT++ is a
  - public-source,
  - component-based,
  - modular and open-architecture simulation environment
  - with strong GUI support and
  - an embeddable simulation kernel

# Thank you for your attention



... questions, comments, etc. are welcome …