Distributed Systems

Simulating Distributed Systems

Gianpaolo Cugola

Dipartimento di Elettronica e Informazione

Politecnico di Milano, Italy

`cugola@elet.polimi.it`

`http://home.dei.polimi.it/cugola`

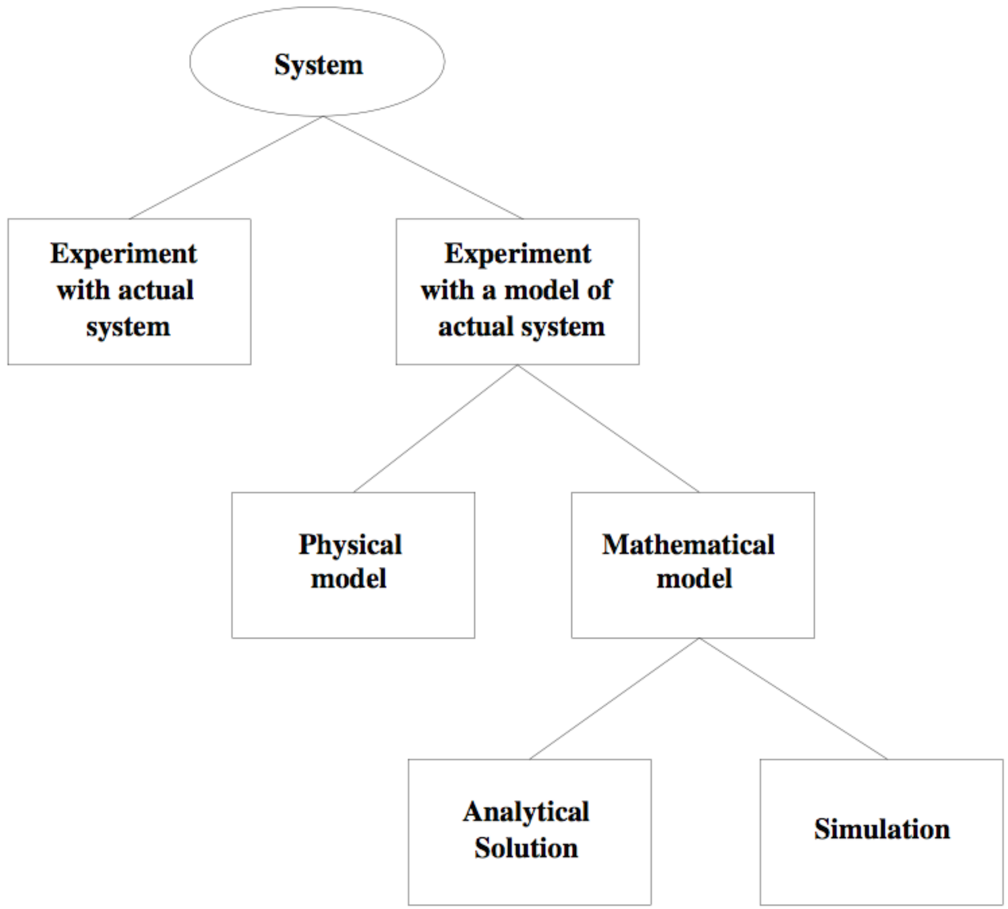
`http://corsi.dei.polimi.it/distsys`

Politecnico di Milano

Contents

- Why to simulate
- Possible form of simulation
- Discrete event simulation in details
- The OMNeT++ simulator
  - The OMNeT++ component model
  - The NED language
  - Simple and compound modules
  - Messages
  - Collecting results
  - The Tic Toc example

Ways To Study A System*



- *Simulation, Modeling & Analysis* (3/e) by Law and Kelton, 2000, p. 4, Figure 1.1

Network emulation

- In between experimenting with actual system and using a model of the system
  - Use the system on a "simulated" environment/network
  - Addresses the difficulty of testing distributed systems in large scale/complex deployments
- Emulation differs from simulation in that a network emulator appears to be a network to the OS

Politecnico di Milano

Network emulation

- It can be accomplished by introducing a component/device that alters packet flow in a way that imitates the behavior of the environment being emulated (e.g., a WAN or a wireless network)
  - Often coupled with a node emulation software (i.e., a virtual machine like UML, VMWare, VirtualBox)
- Network emulation in pratice
  - FreeBSD Dummynet (used in Emulab), Linux NetEm
    - May introduce packet delay, loss, duplication, reordering
  - Other more user friendly tools like marionnet, netkit,...

Politecnico di Milano

Analyzing before building

- Often important to “analyze before build”
- Analysis requires a model of the system
- Various classes of models
  - Analytical vs. operational
  - Discrete vs. continuous
  - Deterministic vs. stochastic
- Simulation is a special form of analysis in which a history of system execution is obtained and analyzed

Politecnico di Milano

Discrete event simulation

- Simple method to analyze the performance of a system using a discrete, deterministic, operational model
- Involves different elements
  - A list of events (timestamped objects)
  - A simulation clock
  - A set of state variable and performance indicators
  - An event processing function
    - Takes an event as a parameter and process it by updating the value of state variables and performance indicators and creating new events
- Operates as follow:

forever

- keep the first event from the list and remove it

- advance the clock to the timestamp of the event

- pass the event to the event processing function

Politecnico di Milano

Discrete event simulator

- A software that implements the loop above by simplifying the job of writing discrete event simulations
- Usually provides a library of basic, general purpose elements
  - E.g., random number generators, containers, etc.
- Plus a library of existing models
  - E.g., hosts, routers, switch, etc.
- Examples (relevant for networks/distributed systems)
  - OpNet, QualNet, JiST/Swans, Parsec/Glomosim, J-Sim, Ns2, OMNeT++

Politecnico di Milano

OMNeT++

- An event simulator widely adopted to simulate distributed systems and networks
- Provides
  - A component model to easily and effectively structure complex simulations through reusable components
  - A C++ class library including the simulation kernel and utility classes (for random number generation, statistics collection, topology discovery etc) to build such components
  - An infrastructure to assemble simulations from these components and configure them (NED language, ini files)
  - Runtime environments for simulations (Tkenv, Cmdenv)
  - An Eclipse-based simulation IDE for designing, running and evaluating simulations
- Separately, several simulation frameworks have been developed, which include OMNeT++ components to simulate distributed systems
  - INET, Mobility Framework, Mixim, Castalia, …

The OMNeT++ component model

- Modules
  - *Simple modules* and *compound modules*
  - Each module has *gates* through which messages can be sent to other modules
  - Modules are *connected* together to build a hierarchy of modules
    - The NED language defines modules and wiring
  - For simple modules, the NED definition just introduce the interface, while the implementation is given in C++
  - Modules have parameters whose values can be given into the omnetpp.ini file and change at each run
- The entire simulation is just an instance of the highest level module

A simple module (NED interface)

```
//
// Ethernet CSMA/CD MAC
//
simple EtherMAC {
  parameters:
    string address; // others omitted for brevity
  gates:
    input phyIn;     // to physical layer
    output phyOut;   // to physical layer
    input llcIn;     // to EtherLLC or higher layer
    output llcOut;   // to EtherLLC or higher layer
}
```

A compound module (in NED)

```
//
// Host with an Ethernet interface
//
module EtherStation {
  parameters: ...
  gates: ...
    input in;    // for connecting to switch/hub, etc
    output 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;
}
```

Politecnico di Milano

Implementing simple modules

- For each simple module you have to write a C++ class with the module's name, which extends the library class `cSimpleModule`
  - You can redefine several methods, main ones are `initialize()`, `finish()`, and `handleMessage(cMessage *msg)`
- You have also to register the class via the `Define_Module(module_name)` macro
- NED parameters can be read using the `par(const char *paramName)` method
- You can send messages to other (connected) modules using the `send(cMessage *msg, char *outGateName)` method

Politecnico di Milano

Message classes

- They are subclasses of the cMessage library class
- You can define them using a special, very simple language. E.g.,

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

Collecting results

- The simulation results are recorded into output vector (.vec) and output scalar (.sca) files
  - This is done by simple modules in C++
- Output vectors capture behavior over time
  - An output vector file contains several output vectors, each being a named series of (timestamp, value) pairs
  - You can configure output vectors from omnetpp.ini
    - You can enable or disable recording individual output vectors, or limit recording to a certain simulation time interval
- Output scalar files contain summary statistics
  - E.g., number of packets sent, average hop-to-.hop delay
- The OMNeT++ library also includes classes to record statistics and organize them. E.g., `cLongHistogram`

Politecnico di Milano

Random numbers

- Often required to generate data randomly
- OMNeT++ provides a configurable number of RNG instances (that is, streams), which can be freely mapped to individual simple modules in omnetpp.ini.
  - This means that you can set up a simulation model so that all traffic generators use global stream 0, all MACs use global stream 1 for backoff calculation, and physical layer uses global stream 2 and global stream 3 for radio channel modelling
  - Seeding can be automatic or manual, manual seeds also come from the ini file
- Several distributions are supported and they are available from both NED and C++
- Non-const module parameters can be assigned random variates like exponential(0.2), which means that the C++ code will get a different number each time it reads the parameter

OMNeT++ in details: The TicToc simulation

- Two nodes in a network
- One of the nodes create a packet and send it to the other
- The two nodes keep passing the same packet back and forth
- We call the nodes “tic” and “toc”

The main topology file
tictoc1.ned

```
simple Txc1 {
  gates:
    input in;
    output out;
}

// Two instances (tic and toc) of Txc1 connected both ways.

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

Politecnico di Milano

Implementing the simple module

txc1.cc

```
#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");
}
```

The omnetpp.ini file

Politecnico di Milano

```
[General]
# nothing here
[Config Tictoc1]
network = Tictoc1
[Config Tictoc2]
network = Tictoc2
[Config Tictoc3]
network = Tictoc3
[Config Tictoc4]
network = Tictoc4
Tictoc4.toc.limit = 5
[Config Tictoc5]
network = Tictoc5
**.limit = 5
[Config Tictoc6]
network = Tictoc6
Config Tictoc7]
network = Tictoc7
Tictoc7.tic.delayTime = exponential(3s)
Tictoc7.toc.delayTime =
    truncnormal(3s,1s)
[Config Tictoc8]
network = Tictoc8
[Config Tictoc9]
network = Tictoc9
[Config Tictoc10]
network = Tictoc10
[Config Tictoc11]
network = Tictoc11
[Config Tictoc12]
network = Tictoc12
[Config Tictoc13]
network = Tictoc13
[Config Tictoc14]
network = Tictoc14
[Config Tictoc15]
network = Tictoc15
record-eventlog = true
[Config Tictoc16]
network = Tictoc16
**.tic[1].hopCount.result-recording-
    modes = +histogram
**.tic[0..2].hopCount.result-recording-
    modes = -vector
```

Compiling and running

- Building the makefile

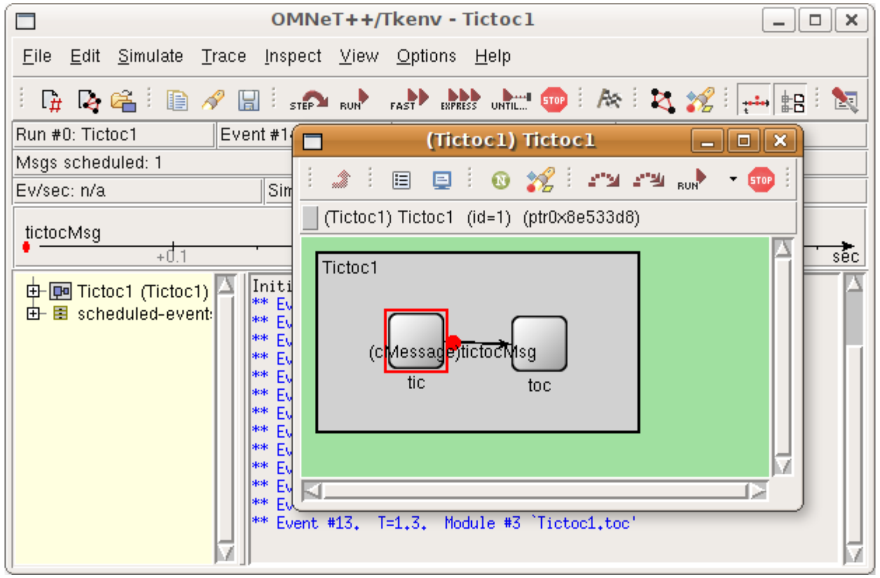
  `opp_makemake`

- Making the simulation

  `make`

- Running the simulation

  `./tictoc`



Enhancing the simulation

tictoc4.ned

Politecnico di Milano

```
simple Txc4 {
  parameters:
    bool sendMsgOnInit =
        default(false);
    int limit = default(2);
    @display("i=block/routing");
  gates:
    input in;
    output out;
}
```

```
network Tictoc4 {
  submodules:
    tic: Txc4 {
      parameters:
        sendMsgOnInit = true;
        @display("i=,cyan");
    }
    toc: Txc4 {
      parameters:
        sendMsgOnInit = false;
        @display("i=,gold");
    }
  connections:
    tic.out -->
     {delay = 100ms;} --> toc.in;
    tic.in <--
     {delay = 100ms;} <-- toc.out;
}
```

Enhancing the simulation

txc4.cc

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
class Txc4 : public cSimpleModule {
  private:
    int counter;
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc4);
```

Politecnico di Milano

Enhancing the simulation

txc4.cc

```
void Txc4::initialize() {
  WATCH(counter);
  counter = par("limit");
  if (par("sendMsgOnInit").boolValue() == true) {
    EV << "Sending initial message\n";
    cMessage *msg = new cMessage("tictocMsg");
    send(msg, "out");
  }
}

void Txc4::handleMessage(cMessage *msg) {
  counter--;
  if (counter==0) {
    EV << getName() << "'s counter reached zero, deleting message\n";
    delete msg;
  } else {
    EV << getName() << "'s counter is " << counter <<
          ", sending back message\n";
    send(msg, "out");
  }
}
```

Leveraging inheritance

tictoc5.ned

```
simple Txc5 {
  parameters:
    bool sendMsgOnInit =
      default(false);
    int limit = default(2);
    @display("i=block/routing");
  gates:
    input in;
    output out;
}
simple Tic5 extends Txc5 {
  parameters:
    @display("i=,cyan");
    sendMsgOnInit = true
}
simple Toc5 extends Txc5 {
  parameters:
    @display("i=,gold");
    sendMsgOnInit = false
}
```

```
network Tictoc5 {
  submodules:
    tic: Tic5;
    toc: Toc5;
  connections:
    tic.out -->
     {delay = 100ms;} --> toc.in;
    tic.in <--
     {delay = 100ms;} <-- toc.out;
}
```

Modelling delays and timers

tictoc8.ned

```
simple Tic8 {
  parameters:
    @display("i=block/routing");
  gates:
    input in;
    output out;
}

simple Toc8 {
  parameters:
    @display("i=block/process");
  gates:
    input in;
    output out;
}
```

```
network Tictoc8 {
  submodules:
    tic: Tic8 {
      parameters:
        @display("i=,cyan");
    }
    toc: Toc8 {
      parameters:
        @display("i=,gold");
    }
  connections:
    tic.out -->
     {delay = 100ms;} --> toc.in;
    tic.in <--
     {delay = 100ms;} <-- toc.out;
}
```

Politecnico di Milano

Modelling delays and timers

txc8.cc

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
class Tic8 : public cSimpleModule {
  private:
    simtime_t timeout;  // timeout
    cMessage *timeoutEvent;  // holds pointer to the timeout self-message
  public:
    Tic8();
    virtual ~Tic8();
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Tic8);

Tic8::Tic8() { timeoutEvent = NULL; }
Tic8::~Tic8() { cancelAndDelete(timeoutEvent); }
```

Modelling delays and timers

txc8.cc

Politecnico di Milano

```
void Tic8::initialize() {
  timeout = 1.0;   timeoutEvent = new cMessage("timeoutEvent");
  EV << "Sending initial message\n";
  cMessage *msg = new cMessage("tictocMsg");   send(msg, "out");
  scheduleAt(simTime()+timeout, timeoutEvent);
}

void Tic8::handleMessage(cMessage *msg) {
  if (msg==timeoutEvent) {
    EV << "Timeout expired, resending message and restarting timer\n";
    cMessage *msg = new cMessage("tictocMsg");   send(msg, "out");
    scheduleAt(simTime()+timeout, timeoutEvent);
  } else {
    EV << "Timer cancelled.\n";
    cancelEvent(timeoutEvent);
    cMessage *msg = new cMessage("tictocMsg");   send(msg, "out");
    scheduleAt(simTime()+timeout, timeoutEvent);
  }
}
```

Modelling delays and timers

txc8.cc

```
class Toc8 : public cSimpleModule {
  protected:
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Toc8);

void Toc8::handleMessage(cMessage *msg) {
  if (uniform(0,1) < 0.1) {
    EV << "\"Losing\" message.\n";
    bubble("message lost");  // making animation more informative...
    delete msg;
  } else {
    EV << "Sending back same message as acknowledgement.\n";
    send(msg, "out");
  }
}
```


Politecnico di Milano

Using copies of messages

txc9.cc

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
class Tic9 : public cSimpleModule {
  private:
    simtime_t timeout;  // timeout
    cMessage *timeoutEvent, *message;
    int seq;
  public:
    Tic9();
    virtual ~Tic9();
  protected:
    virtual cMessage *generateNewMessage();
    virtual void sendCopyOf(cMessage *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Tic9);
Tic9::Tic9() { timeoutEvent = message = NULL; }
Tic9::~Tic9() { cancelAndDelete(timeoutEvent); delete message; }
```

Politecnico di Milano

Using copies of messages

txc9.cc

```
void Tic9::initialize() {
  seq = 0;
  timeout = 1.0;   timeoutEvent = new cMessage("timeoutEvent");
  EV << "Sending initial message\n";
  message = generateNewMessage();   sendCopyOf(message);
  scheduleAt(simTime()+timeout, timeoutEvent);
}
void Tic9::handleMessage(cMessage *msg) {
  if (msg==timeoutEvent) {
    EV << "Timeout expired, resending message and restarting timer\n";
    sendCopyOf(message);
    scheduleAt(simTime()+timeout, timeoutEvent);
  } else {
    EV << "Received: " << msg->getName() << "\n";
    delete msg;
    EV << "Timer cancelled.\n";
    cancelEvent(timeoutEvent);
    delete message;
    message = generateNewMessage();   sendCopyOf(message);
    scheduleAt(simTime()+timeout, timeoutEvent);
  }
}
```

Using copies of messages

txc9.cc

```
cMessage *Tic9::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 Tic9::sendCopyOf(cMessage *msg) {
  // Duplicate message and send the copy.
  cMessage *copy = (cMessage *) msg->dup();
  send(copy, "out");
}
```

Politecnico
di Milano

Using copies of messages
txc9.cc

```
class Toc9 : public cSimpleModule {
  protected:
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Toc9);

void Toc9::handleMessage(cMessage *msg) {
  if (uniform(0,1) < 0.1) {
    EV << "\"Losing\" message.\n";
    bubble("message lost");  // making animation more informative...
    delete msg;
  } else {
    EV << msg << " received, sending back an acknowledgement.\n";
    delete msg;
    send(new cMessage("ack"), "out");
  }
}
```

Exercise

Politecnico di Milano

- Use OMNeT++ to simulate a token ring algorithm to control the access to a shared resource
- Modify the example above to take care of lossy channels
  - Use the `ber` or `per` attributes of channels together with the `asBitError()` method of class `cPacket`
  - Add a mechanism to rebuild the token if it gets lost, e.g., letting a single (leader) node to use a (long enough) timeout to build a new token

Politecnico di Milano

More enhancements

- How to collect results of the simulation via output vectors, scalars, and statistics classes
  - See example `tictoc15`
- Same as above but using *signals* to decouple simulation modelling from statistics collection
  - See example `tictoc16`

Exercise

- Modify the token ring example to measure main performance of the protocol like the time to wait before accessing a resource, the number of packets transmitted, etc.