Introduction to Network Simulation Using OMNeT++

Ahmet Şekercioğlu*

Contents

1 Introduction

This *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.

1.1 Prerequisites

I assume you have downloaded OMNeT++ from its Web site [OMN96], and successfully installed. I recommend that, to be sure that the installation is error-free, run at least a couple of the examples bundled with the distribution. Also, I assume that you are going to use the linux version of OMNeT++ for the experiments of this tutorial.

2 Discrete Event Simulation with OMNeT++

Here are the steps you take to implement your first simulation:

1. Create a working directory called `tictoc`, and `cd` to this directory.

2. Describe your example network by creating a topology file. A topology file is a text file that identifies the network's nodes and the links between them. Let's call this file `tictoc.ned`:

*Many thanks to András Varga for suggestions.

```
// $Id: tictoc.ned,v 1.2 2003/12/01 02:07:02 ahmet Exp $

simple Txc
    gates:
        in: in;
        out: out;
endsimple

module Tictoc
    submodules:
        tic: Txc;
            display: "p=42,188;b=40,24";
        toc: Txc;
            display: "p=218,188;b=40,24";
    connections:
        tic.out --> toc.in;
        tic.in <-- toc.out;
endmodule

network tictoc : Tictoc
endnetwork

```

In this file, we define a network called `tictoc`, which consists of a *compound* module `Tictoc`. The compound module, in turn, consists of submodules `tic` and `toc`. `tic` and `toc` are instances of the same *simple* module type called `Txc`[1]. `Txc` has one input gate (named `in`), and one output gate (named `out`). We connect `tic`'s output gate to `toc`'s input gate, and vice versa.

3. We now need to implement the functionality of the simple module `Txc`. This is achieved by writing two C++ files: `txc.h` and `txc.cc`:

```
// $Id: txc.h,v 1.3 2003/12/01 02:26:29 ahmet Exp $

#include "omnetpp.h"

// Derive the Txc class cSimpleModule.
class Txc : public cSimpleModule
{
  // This is a macro; it expands to constructor definition etc.
  // 16384 is the size for the coroutine stack (in bytes).
```

[1] It would be a good idea to adopt `C++`-like naming conventions (type names with uppercase, variables with lowercase), and begin the submodule names with lowercase (i.e., `tic` and `toc`).

```
  Module_Class_Members(Txc, cSimpleModule, 16384);

  // This redefined virtual function holds the algorithm.
  virtual void activity();
};
```

```
// $Id: txc.cc,v 1.4 2003/12/01 02:26:29 ahmet Exp $

#include <stdio.h>
#include <string.h>
#include "omnetpp.h"
#include "txc.h"

// register player module types
Define_Module(Txc);

void Txc::activity()
{
  ev << "Hello World! I'm " << name() << ".\n";

  // Am I Tic or Toc?
  if (strcmp("tic", name()) == 0)
    {
      // Tic sends initial message and then waits for Toc's response.
      // (Toc starts waiting for Tic's message straightaway.)
      cMessage *msg = new cMessage(name());
      ev << name() << " sending 1st msg: "<< msg->name() << ".\n";
      send(msg, "out");
    }

  // Infinite loop to process events.
  for (;;)
    {
      cMessage *msgin = receive();
      ev << name() << " got msg: " << msgin->name() << ".\n";
      delete msgin;
      wait(1.0);
      cMessage *msg = new cMessage(name());
      ev << name() << " sending msg: " << msg->name() << ".\n";
      send(msg, "out");
    }
}
```

4. Then, we need to write the file `omnetpp.ini` which will tell the simulation tool what to do:

```
# $Id: tictoc-omnetpp.ini,v 1.2 2003/12/01 02:07:02 ahmet Exp ahmet $

[General]
ini-warnings = no
network = tictoc

[Cmdenv]
module-messages = yes
verbose-simulation = no

[Tkenv]
default-run=1
```

5. We now create the `Makefile` which will help us to compile and link our program to create the executable `tictoc`:
   `opp_makemake`
   This command should have now created a `Makefile` in the working directory `tictoc`.

6. Add dependencies to the `Makefile`:
   `make depend`
   (ignore the generated errors.)

7. Let's now compile and link our very first simulation by issuing the `make` command:
   `make`
   If there are compilation errors, you need to rectify those and repeat the `make` until you get an error-free compilation and linking.

8. Once you complete the above steps, you launch the simulation by issuing this command:
   `./tictoc`
   and, hopefully you should now get the OMNeT++ simulation window similar to the one shown in Figure 1.

9. Press the "run" button to start the simulation. What you should see is that `tic` and `toc` are exchanging messages with each other. This is the result of the `send()` and `receive()` calls in the C++ code.

   The main window behind displays text messages generated via the `ev << ...` lines from these modules. Observe that the messages "`Hello World!  I'm tic.`"

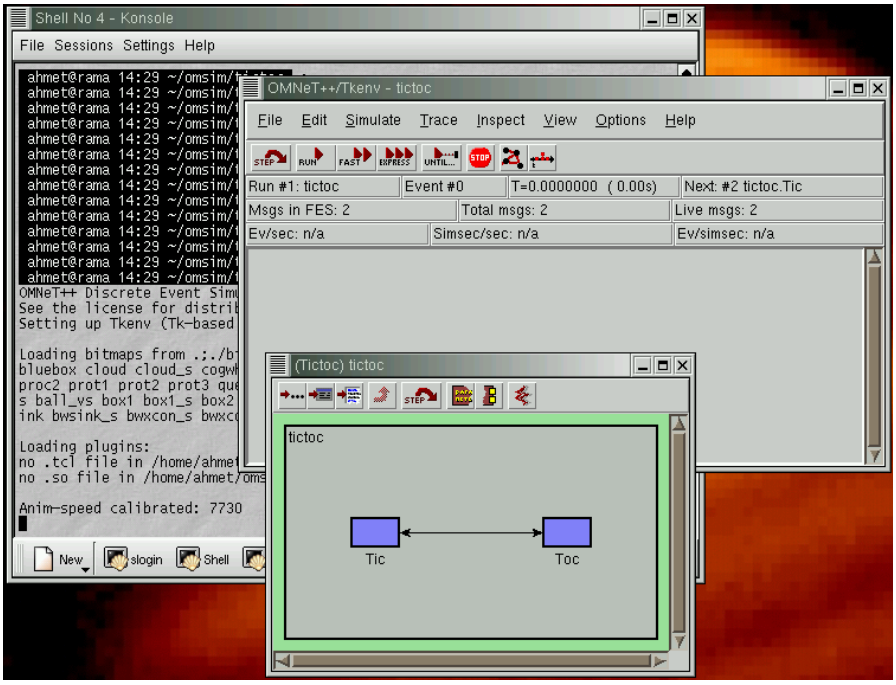

Figure 1: `tictoc` simulation.

and "`Hello World!  I'm toc.`" are only printed *once* at the beginning of the simulation.

The main window toolbar displays the simulated time. This is "virtual" time, it has nothing to do with the actual (or wall-clock) time that the program takes to execute. Actually, how many seconds you can simulate in one real-world second depends highly on the speed of your hardware and even more on the nature and complexity of the simulation model itself (Exercise 2 will emphasize this issue). Note that the toolbar also contains a simsec/sec gauge which displays you this value.

10. In real life communication systems, the links carrying the packets do not transmit a packet from one end to the other instantaneously. Instead, the packets experience "propagation delays". Let's now improve our model by introducing a more realistic link which will delay the messages 0.5 sec. in both directions:

    (a) Edit the `tictoc.ned` to insert the following lines after line 7:

    ```
    channel TicTocLink
       delay 0.5 // sec.
    endchannel
    ```

    (b) Then, modify the lines 16 and 17 of `tictoc.ned` as follows:

```
tic.out --> TicTocLink --> toc.in;
tic.in <-- TicTocLink <-- toc.out;
```

We now have a link connecting modules `tic` and `toc` involving a 0.5 seconds propagation delay.

11. Repeat the `make` command, and then run the `tictoc` to see the effects of the propagation delay. In OMNeT++, communication link models can have quite sophisticated properties. Explore the user manual [Var] to see what they are.

3 Exercises

1. Line 31 of `txc.cc` contains the `wait(1.0);` OMNeT++ kernel call. Find out what does this call do by referring to the OMNeT++ user manual [Var].

2. Now, let a `tictoc` simulation running. Change the line 31 (`wait(1.0);`) of the `txc.cc` to `wait(100.0);`. Then do a "`make`", and run the new `tictoc` in another window. Should we expect that the message exchange between `tic` and `toc` slowed? If no, why?

3. Delete the communication link connecting `tic` and `toc`. Then, insert a third module called `tac` between the modules `tic` and `toc`. This new module will be responsible for relaying messages bouncing between `tic` and `toc` by having direct connections to both of them: `tic` → `tac` → `toc` (i.e., there will not be a direct communication between `tic` and `toc`).

4. I have written the message handling mechanism of `txc.cc` by using the `activity()` call. Rewrite the `txc.cc`, this time by using the `handleMessage()` mechanism of OMNeT++. To do this, you need to study the examples presented in the Section 5.3.2 of the OMNeT++ user manual [Var].

References

[OMN96] OMNeT++ object-oriented discrete event simulation system. URL reference: `http://www.omnetpp.org`, 1996.

[Var] A. Varga. *OMNeT++ Object-oriented Discrete Event Simulation System User Manual*. URL reference: `http://www.omnetpp.org/external/doc/html/usman.php`.