MONASH UNIVERSITY
DEPARTMENT OF ELECTRICAL & COMPUTER SYSTEMS ENGINEERING

**Performance of Telecommunication Networks**

# Experiment II: Introduction to Discrete-Event Simulation and OMNeT++ Framework

Y. Ahmet Şekercioğlu

## Contents



## 1 Introduction

Discrete-event simulation complements analytical (mathematical) tools and experimental methods for performance analysis and estimation of communication networks. A well-designed simulation model can provide an extremely important insight into the structural weaknesses of a computer network or a communication protocol.

Because of their versatility, in addition to modeling communication networks, discrete-event simulation methods are indeed widely used in a wide number of areas such as computer system performance analysis, manufacturing processes, database systems etc. and there is a large number of discrete-event simulation tools available.

In this unit, we will be using OMNeT++ [OMN06] for building our simulation models and doing our experiments. OMNeT++ is a very well designed, modular, widely-used system, source code is available and free for teaching and research purposes.

## 2 What is OMNeT++?

Firstly, I think it is beneficial to understand what OMNeT++ discrete-event simulation framework is and is not. OMNeT++ is basically *a collection of software tools and libraries* which you can use to build your own simulation models. The simulation framework provides the following:

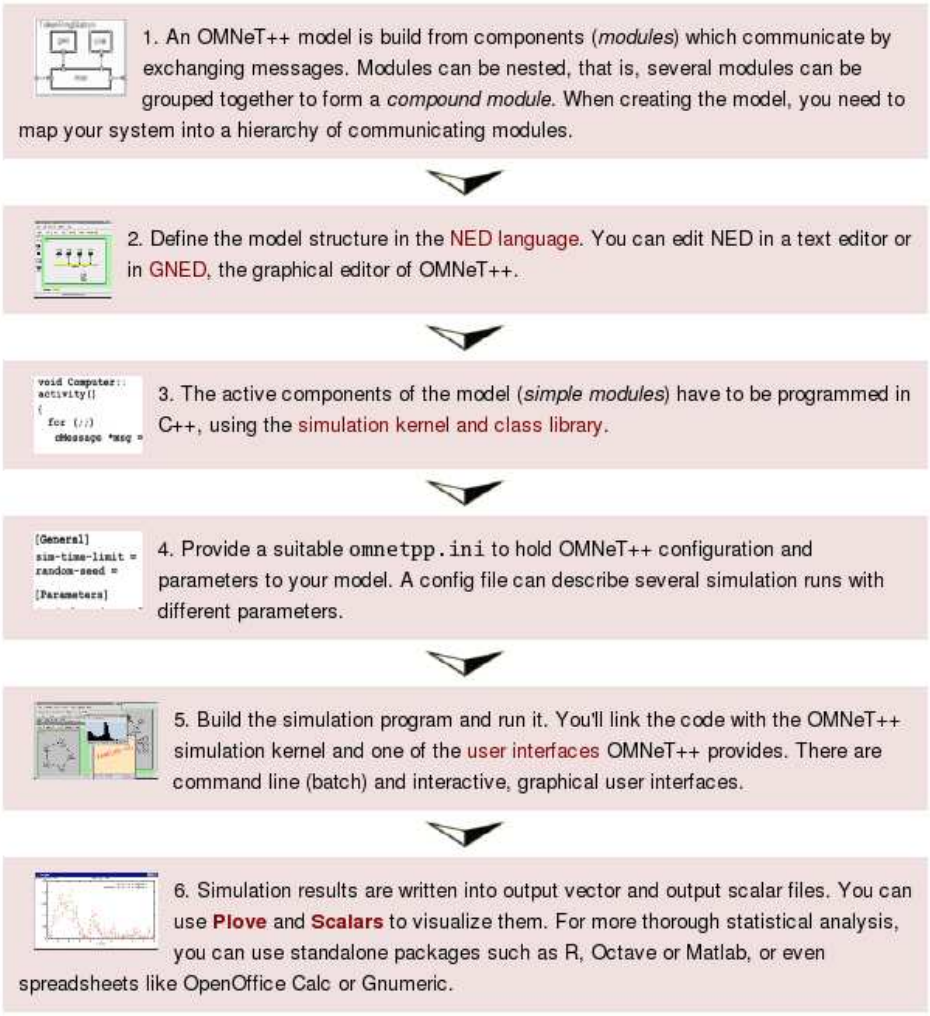


Figure 1: The process of modeling and simulation with OMNeT++ (copied from [OMN06]). We will learn the process by doing the experiments. Note that the actual simulation and data collection process is not linear as shown here, but involves loops since you will need to modify models and re-run the experiments to collect new data.

- A simulation kernel,
- a compiler for the NED topology description language,
- a graphical network editor for NED files,
- two types of user interfaces for simulation execution:
  - a graphical user interface, or
  - a command-line user interface
- tools for plotting data,
- utilities (random number seed generation tool, makefile creation tool, etc.)

- and tools for documentation

The actual models are written by the model designer, or borrowed from someone who has already written these. There are significant number of models available through the OMNeT++ Web site.

Figure 1 summarizes the stages of OMNeT++ based simulation process from beginning to end. We will learn about these stages step-by-step by doing the tutorials provided.

### 2.1 The Simulation Environment

The simulator is already installed on your laboratory PCs (under the Linux operating system). if you would like to experiment with OMNeT++, you can install it on your home PC or laptop (OMNeT++ can be installed under Windows XP as well).

## 3 Exercise 1 - Getting Started

Our first exercise with OMNeT++ aims to teach the basics of the simulation framework. The exercise is accessible through the following Web page: http://omnetpp.org/doc/tictoc-tutorial/part1.html and aims to get you running your very first simulation.

Follow the steps given there and run your first simulation experiment. Afterwards answer the questions given in the following section.

I think we have now familiar with the OMNeT++ modeling and simulation process. Note that whenever you modify the topology (`.ned`) file(s), or files describing the active components of the model (`.cc` files) or add new ones, you need to re-run some commands again. Please have a look at the detailed flow diagram shown in Figure 2 to see what to do whenever you make a *change* in your simulation files.

### 3.1 Questions

1. In which file do we define our network topology?
2. How many nodes and links does our network have?
3. What does `initialize()` function do?
4. What does `handleMessage()` function do?
5. What does `send()` simulation kernel call do in the used in the `initialize()` function?
6. What does the `Define_module` macro do? Can this macro be used in a header file (.h file) and why?
7. In `handleMessage()` function, can we use another `send()` function right after the first `send()` to send out the same message?
8. Following activities and questions are related to *dynamic memory allocation and de-allocation in C++*:

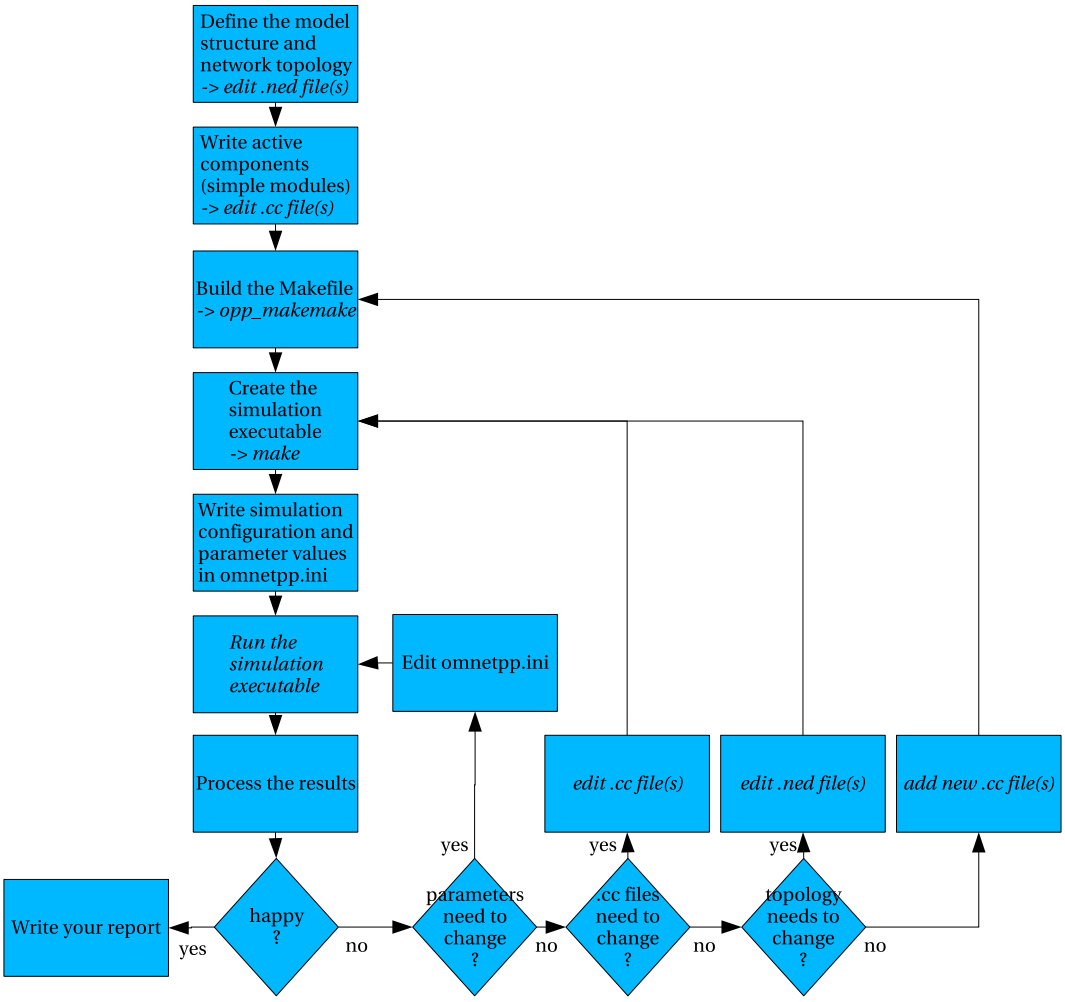


Figure 2: Detailed flowchart of the OMNeT++ simulation process.

(a) Locate the following two statements in the `initialize()` function

```
cMessage *msg = new cMessage("tictocMsg");
send(msg, "out");
```

and comment them out. Recompile the simulation model (`make`) and run again (`./tictoc`). What does happen?

(b) Now, put these lines back in. But insert

```
delete msg;
```

between the `cMessage ...` and `send ...` statements. Does the code compile without errors? If yes, run it. Does it run without any errors? If no, what is the error reported?

(c) Modify the line containing `delete ...` as follows

```
delete msg; msg = NULL
```

recompile and run the code. What does happen?

→ Most of the run-time errors you will encounter in your experiments will be related to dealing with invalid pointer references. If you keep to the habit of assign-

ing `NULL` value to a pointer variable referencing to a de-allocated memory block, you will be spending almost no time on finding the errors. Otherwise frustrating hours may pass without any results!

*Important note:* I may ask similar questions in the final exam!

# 4 Exercise 2 - Towards More Sophisticated Models

Our second exercise aims to teach the fundamental methods required to create more sophisticated simulation models. Through this experiment we learn how to add state variables, parametric simulation, modeling processing delays, dealing with random numbers and implementing timers. The exercise is available in this Web page: http://omnetpp.org/doc/tictoc-tutorial/part2.html, and aims to teach following aspects of the OMNeT++ simulation process:

- refining the graphics, and adding debugging output,
- adding state variables,
- adding parameters,
- modelling processing delay,
- working with random numbers and parameters,
- setting and cancelling timers, and
- retransmitting messages.

Please complete the activities of the tutorial to answer the questions given in the following section.

## 4.1 Questions

1. What does `ev << "Hello\n";` do? If I write `printf("Hello\n");` instead, does it do the same?
2. What does "`WATCH`" do?
3. Refer to the OMNeT++ manual [Var06] for information on `cMessage` object to find out how to define the name and the kind of a message?
4. Modify the `initialize()` function to create two messages with different names and kinds (you decide the name and the kind). Send the two messages in sequence and print out their names and kinds (hint: you might want to use "enum" to do this).
5. Modify the `handleMessage()` function to receive the two messages (see previous item) separately. When a message is received, print out its name and kind.
6. How do we model processing delay? In the example given we choose to model processing time of 1 second. If I change this to 100000 seconds, how much longer the simulation would run?

7. Go through the OMNeT++ manual [Var06] to find out how many different types of random number distributions implemented in the framework. Write them here.

*Important note:* I repeat, I may ask similar questions in the final exam!

## References

[OMN06] OMNeT++ Object-Oriented Discrete Event Simulation System. URL reference http://www.omnetpp.org, 1996–2006.

[Var06] A. Varga. OMNeT++ Object-Oriented Discrete Event Simulation System User Manual. URL reference http://www.omnetpp.org/doc/manual/usman.html, 2006.