PERFORMANCE EVALUATION FOR DISCRETE EVENT SIMULATORS: OSSIM VS. OMNET++

Elena ULEIA[1]

Obiectul lucrării de faţă este prezentarea performanţelor de rulare ale simulatorului OSSim (Open Source Simulator), comparativ cu simulatorul OMNeT++ (Object-Oriented Modular Discrete Event Network Simulator), cât şi evaluarea acestora. Pentru a adresa această problemă, s-au efectuat o serie de simulări de viteză mare, aplicate asupra unui sistem simplu de cozi. Prezenta evaluare include colectarea datelor de performanţă pentru ambele pachete de program, verificarea corectitudinii acestora, cât şi raportarea rezultatelor.

The objective of this paper is to present the runtime performance of the OSSim (Open Source Simulator) tool evaluation against a similar class simulation tool, the OMNeT++ tool (Object-Oriented Modular Discrete Event Network Simulator). In order to address this question, a number of performance evaluations for the high speed simulation runs have been conducted, by simulating a simple queueing system. The evaluation includes collecting performance data for both tools, checking the correctness of the analyzed performance data, and reporting the results.

Keywords: simulation, queuing systems, performance analysis, hierarchical models, performance analysis, distributed computing

1. Introduction

The present paper introduces the performance evaluation results for OSSim and OMNeT++ simulation tools. The OSSim simulator tool has been entirely designed and developed by the author during her PhD research.

The evaluation is performed using the same execution environment for both tools, which consists of an Ubuntu 6.10 desktop i386 operating system (a Linux-like OS) running in a virtual machine environment over Windows XP 2002, SP2. The virtual machine runs with VMware Server 1.0.2, configured as local host. The underlying host machine is a Dell Laptop 6400, Core Duo T7200, 2.0GHz, 2Gb RAM memory, and Level 2 cache with 4Mb.

[1] PhD student, Computer Science Faculty, University POLITEHNICA of Bucharest, Romania

2. Tools Description

Methodologies for establishing model credibility, statistical analysis and experimental designs are currently largely unsupported by simulation software. More guidance could be given, particularly in interactive environments. A small body of well-established techniques is available, although mostly scattered through journals and conference proceedings. It is to be hoped that some of the more robust of these will be integrated with new simulation systems [1][2].

2.1 OSSim Tool

The OSSim (Open Source Simulator) software package is designed to provide a comprehensive work environment for the network modeller. It can be used in diverse applications areas of communications networks such as:

- to measure the performances of existing or future conditions networks under a wide range of conditions;
- to analyse and simulate queuing systems to design;
- to debug and fine-tune discrete-event system models.

OSSim is a discrete event simulator package that allows for development of network simulation and analysis which makes use of technologies from the area of distributed computing, client-server architectures and object oriented programming. One of the key issues of the design is high performance of the overall system, making possible the modelling and the simulation of complex networks.

OSsim is also suitable for simulation of any system with a hierarchical structure which admits a discrete time modelling.

2.2 OMNeT++ Tool

Objective Modular Network Test-bed (OMNeT++) is a public-source, component-based, modular simulation framework. It is practically an object-oriented modular discrete event simulator. The simulator can be used for modelling: communication protocols, computer networks (traffic modelling etc.), multi-processors and distributed systems. Or any other system where the discrete event approach is suitable [3].

OMNeT++ provides the simulation library with statistical classes. It has execution environments that support interactive simulation including visualization of collected data. There's a gnuplot-based GUI tool for analysing and plotting simulation results. OMNeT++ also helps specifying parameter values, managing multiple runs, selecting seed values, and supports parallel execution [4].

3. Simulation Model

The model chosen for comparison is a simple FIFO queue. The *M/M/1* FIFO queue is a basic model in the queuing system. It is a very popular queue model, used in modelling client-server system. *M* represents the notation for a Markov process, whereby the transition probabilities depend only on current state, that is, memoryless process. Poisson/Exponential is special case of Markov.

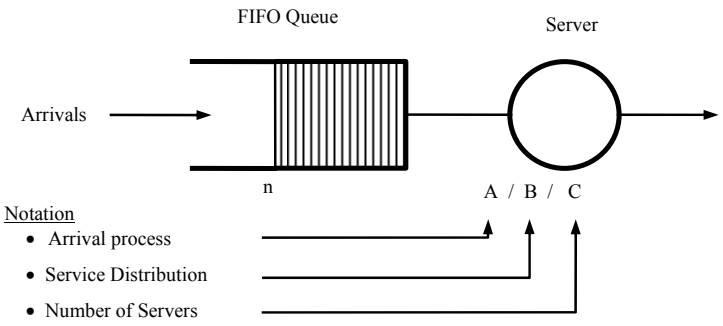

Fig. 1. M/M/1 FIFO queue model

There are two sets of experiments performed. One consists of 6 simulation runs are run for each tool, for a simulation time of 10^5. The other experiment is similar with the first one, but with increased simulation time to 10^6. The pseudo-random numbers generation algorithm is Mersenne Twister RNG [5]. The seed is constant for all of the experiments with value set to 87.

The parameters of the queueing system are: inter-arrival time is an exponential distribution with the mean value of 0.1, message length of 100 bits, and bit rate equal to 1000 bis/sec.

The performance data recorded for each simulation run are: simulation run time (RT), events rate (EvS), average time a packet spends in the queue before service is complete, and maximum time in the queue. The last two parameters are the model output parameters. As the simulation times and the seed values are unchanged, these two parameters are expected to be unchanged for each experiment.

The performance data that form the basis of evaluation are: the average run time for each simulation run (Avg_RT_j), and the average events processed in a unit time during the same simulation run (Avg_EvS_j).

$$Avg\_RT_j = \left(\sum_{i=1}^{n} RT_{ij} \right) \tag{1}$$

$$Avg\_EvS_j = \left(\sum_{i=1}^{n} EvS_{ij} \right) \tag{2}$$

where, n represents the total number of simulation runs within an experiment j;
i represents the index of the simulation runs (replications), with range between $(0,n)$;
j represents the experiment number.

4. Simulation Runs with OMNeT++ Tool

In order to build an executable simulation program, one first needs to translate the NED files and the message files into C++, using the NED compiler (nedtool) and the message compiler (opp_msgc). After this step, the process is the same as building any C/C++ program from source: all C++ sources need to be compiled into object files (.o files on Unix/Linux, and .obj on Windows), and all object files need to be linked with the necessary libraries to get an executable.

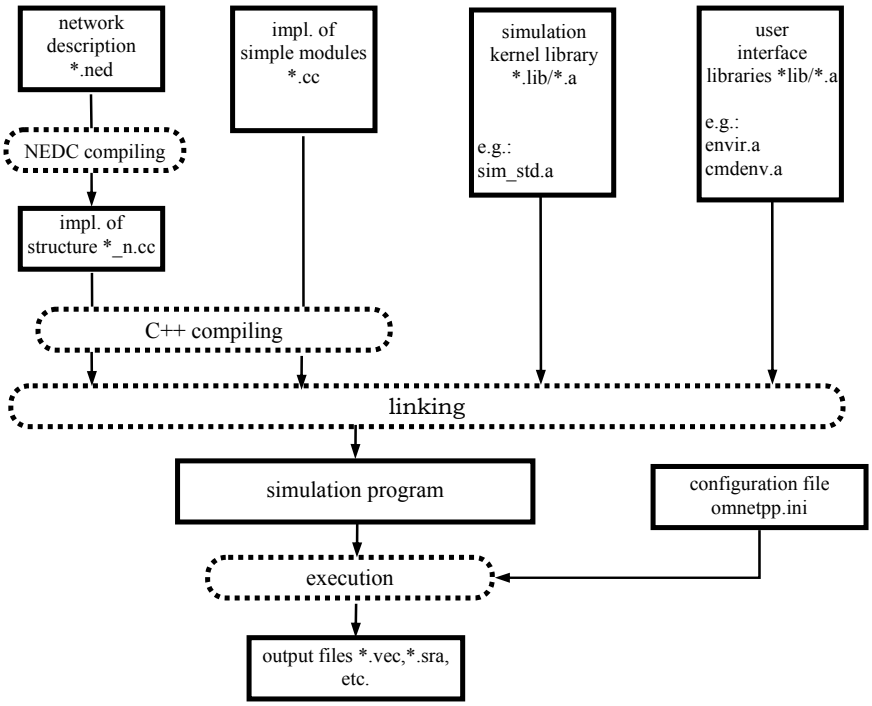

Fig.2. Building and Running an OMNeT++ simulation

A graphics window (Fig. 3) shows the OMNeT++ queueing network model that contains: a generator (*gen*) module, a *FIFO* queue (*fifo*) module, and a sink (*sink*) module.

The source (*gen*) module generates messages that represent jobs. These job-messages are sent to the *fifo* module. This in turn delays the messages according to the present queue, then services them, and finally sends the messages to the *sink* module. The sink module processes these messages to extract some final statistics. The sink is where the messages leave the queueing network. The sink module releases as well the memory allocated to the job messages [6].

The .ned file specifies the modules gates. Furthermore the modules parameters such as arrival rate, queue size, etc. need be declared in the .ned files too. The definition of the parameters, i.e., giving them a value, takes place when the simulation starts, through the *omnetpp.ini* file.

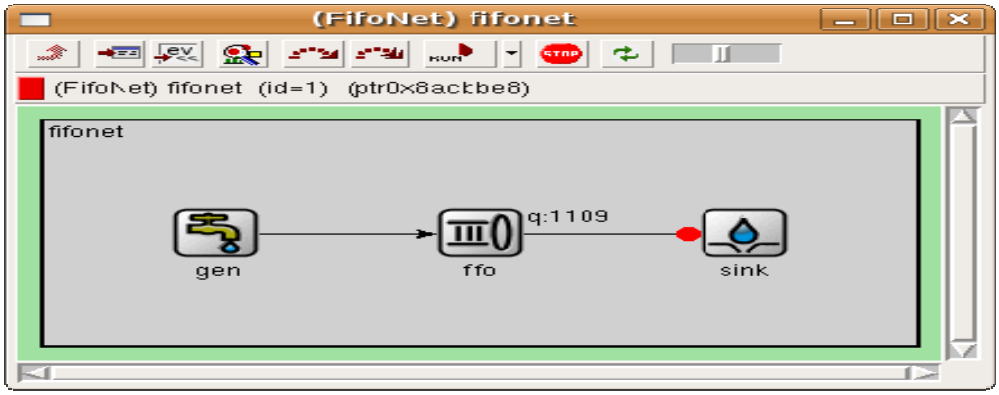

Fig.3. M/M/1 FIFO queue model for OMNeT++ simulation

Thus, the model parameters are specified in the configuration file, *omnetpp.ini*:

```
[General]
network = fifonet
seed-0-mt=87
sim-time-limit = 100000s
[Run 1]
**.gen.sendIaTime = exponential(0.1)
**.gen.msgLength = 100
**.fifo.bitsPerSec = 1000
...
```

As we are only interested in numerical output, we run the simulation straight from the prompt with the *cmdenv* mode. No windows will appear, only the output files will be produced. Due to this reason, the simulation will become quicker as well. This will give maximum speed of the simulation run.

Table 1 below shows the results for the first experiment run, with sim_time = 10^5, for the OMNeT++ tool.

Table 1

OMNeT++ simulation results for sim_time = 10^5

Simulation run no.	Run Time (RT) in sec.	Events/sec (EvS)	Average queue time [sec]	Maximum queue time [sec]
#1	12.979	327427	63.3756	141.828
#2	10.073	413470	63.3756	141.828
#3	10.373	406225	63.3756	141.828
#4	10.325	409883	63.3756	141.828
#5	10.236	406267	63.3756	141.828
#6	10.203	412953	63.3756	141.828

Applying the formulae in equations (1) and (2), the average values for experiment #1, with n = 6, j = 1, i Є (1, 6) become:

$$Avg\_RT_1 = \left(\sum_{i=1}^{6} RT_{i1} \right) = 10.71 \text{ sec.}$$

$$Avg\_EvS_1 = \left(\sum_{i=1}^{6} EvS_{i1} \right) = 396038 \text{ events / sec.}$$

Table 2 below shows the results for the second experiment run, with sim_time = 10^6, for the OMNeT++ tool.

Table 2

OMNeT++ simulation results for sim_time = 10^6

Simulation run no.	Run Time (RT) in sec.	Events/sec (ES)	Average queue time [sec]	Maximum queue time [sec]
#1	127.336	332949	69.1399	241.747
#2	100.364	409227	69.1399	241.747
#3	99.849	412540	69.1399	241.747
#4	98.785	414328	69.1399	241.747
#5	100.522	409809	69.1399	241.747
#6	99.113	414274	69.1399	241.747

By applying again the formulae in equations (1) and (2), the average values for experiment #2, with n = 6, j = 2, i Є (1, 6) become:

$$Avg\_RT_2 = \left(\sum_{i=1}^{6} RT_{i2} \right) = 104.33 \text{ sec.}$$

$$Avg\_EvS_2 = \left(\sum_{i=1}^{6} EvS_{i2} \right) = 398855 \text{ events / sec.}$$

Note that the average run time is increased roughly 10 times, same as the simulation time ratio, and the performance ratio of events processed within a second is approximately the same. Both these results are consistent.

The text below is the log excerpt from the second experiment with OMNet++ tool. This adds results on the standard deviation for the *fifo* queue time, too.

```
Calling finish() at end of Run #1...
Total jobs processed: 9992576
Avg queueing time:    69.1399
Max queueing time:    241.747
Standard deviation:   59.2123
End run of OMNeT++
```

5. Simulation Runs with OSSim Tool

In order to build an executable simulation program, one first needs to translate the GUI user interface modules files into C++, using the internal parser. After this step, the process is the same as building any C/C++ program from source: all C++ sources need to be compiled into object files (.o files on Unix/Linux), and all object files need to be linked with the necessary libraries to get an executable.

The attributes file is loaded dynamically at execution time.

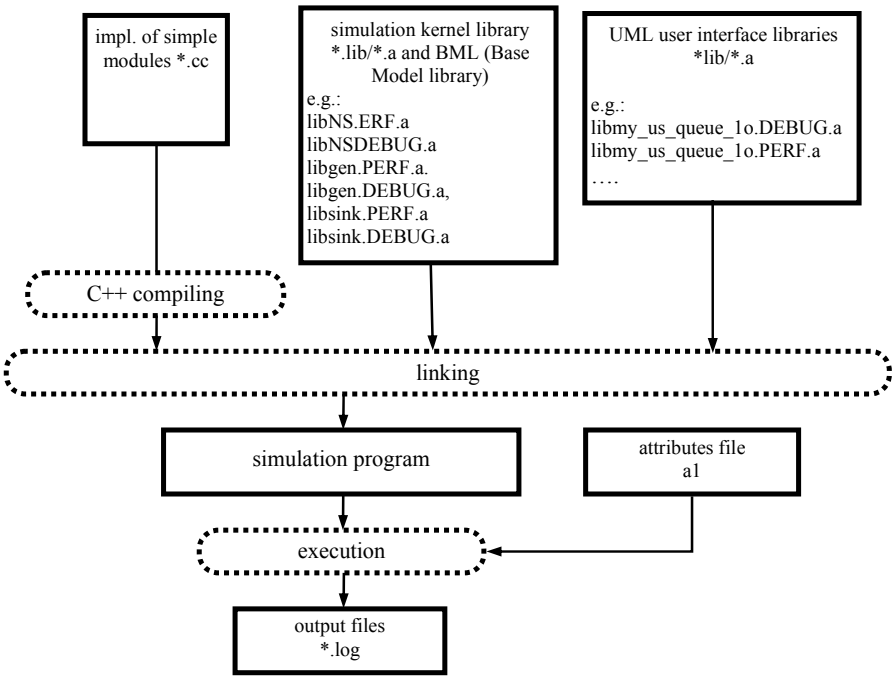

Fig. 4. Building and Running an OSSim simulation

The graphics window (Fig. 5) shows the OSSim queueing network model that contains: a generator (*gen1*) module, a *FIFO* queue (*myq*) module, and a sink (*sink*) module.

The source (*gen1*) module generates packets that are sent to the *myq* module, with rate following the exponential distribution, and length following a constant distribution as with the OMNeT++ case. The *myq* in turn delays the messages according to the present queue. It then sends the packets to the *sink* module. The sink module consumes these packets, by de-allocating the memory.

The *sink* is where the packets leave the queueing network. The *sink* module releases as well the memory allocated to the job messages.

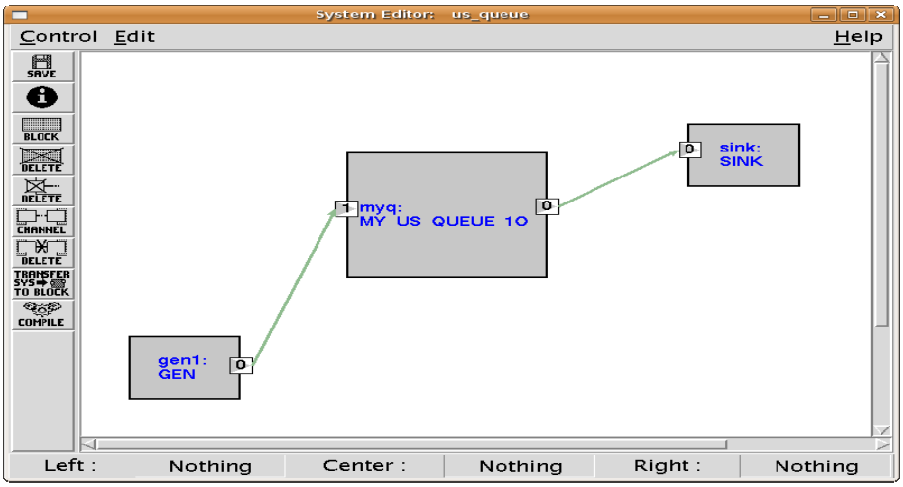

Fig.5. M/M/1 FIFO queue model for OSSim simulation

The modules parameters such as arrival rate, queue size, etc. are declared in the .cc files. The definition of the parameters, i.e., giving them a value, takes place when the simulation starts, through the *a1* attribute file. An excerpt for a1 is given below:

```
# attributes file
.gen1 pklen_dist        constant(100)
.gen1 time_dist         exponential(0.1)
.myq  queue_size        10000
.myq  bitrate           1000
```

Table 3 below shows the results for the first experiment run, with sim_time = 10^5, for the OSSim tool.

Table 3

OSSim simulation results for sim_time = 10^5

Simulation run no.	Run Time (RT) in sec.	Events/sec (EvS)	Average queue time [sec]	Maximum queue time [sec]
#1	0.65120	6,144,712	63.3053	141.828
#2	0.77169	5,185,223	63.3053	141.828
#3	0.72216	5,541,359	63.3053	141.828
#4	0.72535	5,516,973	63.3053	141.828
#5	0.68175	5,869,822	63.3053	141.828
#6	0.75402	5,307,182	63.3053	141.828

Applying the formulae in equations (1) and (2), the average values for experiment #1, with n = 6, j = 1, i Є (1, 6) become:

$$Avg\_RT_1 = \left(\sum_{i=1}^{6} RT_{i1} \right) = 0.717698 \text{ sec.}$$

$$Avg\_EvS_1 = \left(\sum_{i=1}^{6} EvS_{i1} \right) = 5{,}594{,}211 \text{ events / sec.}$$

Table 4 below shows the results for the second experiment run, with sim_time = 10^6, for the OSSim tool.

Table 4

OSSim simulation results for sim_time = 10^6

Simulation run no.	Run Time (RT) in sec.	Events/sec (EvS)	Average queue time [sec]	Maximum queue time [sec]
#1	6.16664	6,184,462	69.1338	241.747
#2	6.39358	6,252,112	69.1338	241.747
#3	6.14285	6,509,386	69.1338	241.747
#4	6.38476	6,262,749	69.1338	241.747
#5	6.39232	6,255,341	69.1338	241.747
#6	6.17144	6,479,228	69.1338	241.747

By applying again the formulae in equations (1) and (2), the average values for experiment #2, with n = 6, j = 2, i Є (1, 6) become:

$$Avg\_RT_2 = \left(\sum_{i=1}^{6} RT_{i2} \right) = 6.275 \text{ sec.}$$

$$Avg\_EvS_2 = \left(\sum_{i=1}^{6} EvS_{i2} \right) = 6{,}323{,}879 \text{ events / sec.}$$

Note that the average run time is increased roughly 10 times, same as the simulation time ratio. These results are consistent.

The performance ratio of events processed per second, though higher in the second replications set, is on average about $6*10^6$, which is the tool benchmark.

The window below (Fig. 6) is the capture from the second experiment, first simulation replication with OSSim tool.

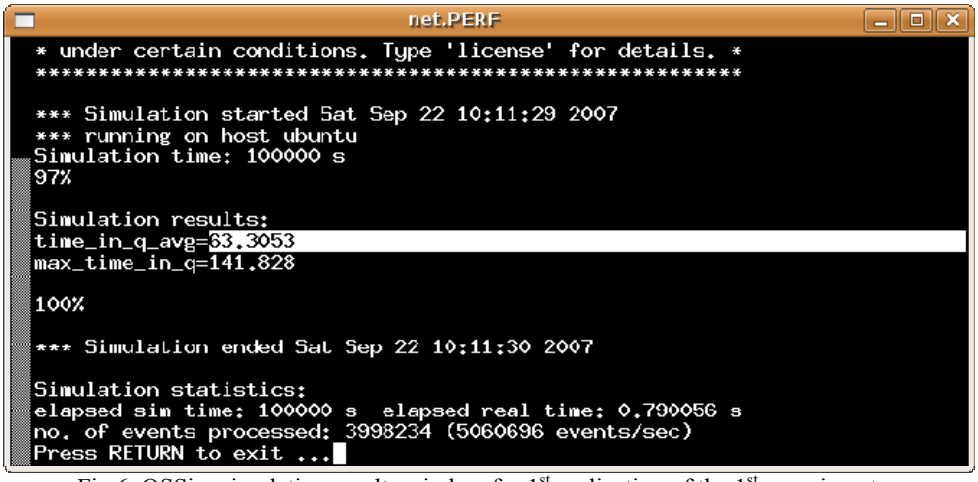

Fig.6. OSSim simulation results window for 1st replication of the 1st experiment

6. Performance discussion and Conclusions

The performance evaluation conducted in this section demonstrates the reliability of the OSSim tool as well as high speed running of a simulation.

The statistical results for the average queue time, and for the maximum queue time are equal in both OSSim and OMNeT++ simulations. This happened actually after choosing the same random number generation algorithm in both tools, which led to implementing the Mersenne Twister (MT) in the OSSim simulation kernel. Before doing this, the results for OSSim were 20%-30% higher than with the OMNeT++.

The generated random number sequence was obviously different between the MT (OMNeT++), and erand48 (OSSim), which caused big differences in the output parameter values.

Nevertheless, by using the same RNG algorithms, the maximum queue time are identical for both experiments. On the other hand, the average queue time in the first experiment with OSSim tool is shorter by 0.07sec, which is 1.1%. For the second experiment, the difference is a lot smaller, only by 0.006 sec, which is 0.01%. The OSSim results being shorted this time, too.

Some further investigation is required for determining the difference of 1.1% in the average values for the first experiment.

The OSSim tool proves to be very fast while running a simulation with any simulation time value. The ratio in the simulation time is preserved almost the same in the run time length. A simulation executable for the comparison model, with a simulation time of 10^8, takes 11min 23sec (683sec). This proves a linear behaviour of the tool.

OSSim tool behaves 15 times faster than the OMNeT++ tool, with both tools running in command line mode. This ensured both tools are run in their fastest mode with minimal interaction with the environment.

The high performance network simulation environment is capable of simulating, on usual hardware, 1.2E10 events within one hour, based on several implementation optimisations [7],[8].

The current OSSim development release is l.0alpha6. The current body of code is relatively small – about 5,000 lines of C++ code for the Simulation Kernel, and about 8,000 lines of Tcl code for the GUI part. However, the size of a program is by no means the right measure for its performance.

Other experiments are planned to be performed on an increased complexity system, for insight performance evaluation. Additionally, very long time simulation experiments are planned for checking further the convergence and linearity of the tool behaviour.

REFERENCES

[1] *W.Kreutzer*, System Simulation: Programming Styles and Languages, Addison-Wesley, 1986

[2] *J. Banks, J.S.Carson II, B.L. Nelson,* Discrete-Event System Simulation, Prentice Hall, 1999 [3]. OMNeT++ object-oriented discrete event simulation system, URL reference: http://www.omnetpp.org, accessed August 2007.

[4] *Varga, Andre*, 'OMNeT++ Object-oriented Discrete Event Simulation System User Manual', ver.3.2, URL reference: http://www.omnetpp.org/doc/manual/usman.html, accessed Aug 2nd, 2007

[5] *M. Matsumoto and T. Nishimura,* 'Mersenne Twister Home Page, A very fast random number generator of period 2^{19937}-1', http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html, accessed May 2nd, 2007

[6] *N. van Foreest*, Simulation Queueing Networks with OMNet++ URL reference http://www.omnetpp.org/doc/queueing-tutorial.pdf, accessed Aug 25th, 2007.

[7] *A. Maranda, Elena Uleia,* PADS: Software protocols hand coding made easy, Institute of Microtechnology, Bucharest, ESPRIT 20287 project, Internal report, June 10, 1997.

[8] *Elena Uleia, Simona Halunga,* Implementation techniques for communications protocols, International Symposium on Signals, Circuits and Systems, 2005, ISSCS 2005, Volume 2, 14-15 July 2005 Page(s):529 - 532 Vol. 2