SFB 637 Autonomous Logistics

Becker, M.; Weerawardane, T.L.; Li, X.; Görg, C.: Extending OPNET Modeler with External Pseudo Random Number Generators and Statistical Evaluation by the Limited Relative Error Algorithm. In: Ince, N.; Bragg, A. (eds.): Recent Advances in Modeling and Simulation Tools for Communication Networks and Services. Springer, 2007

Extending OPNET Modeler with External Pseudo Random Number Generators and Statistical Evaluation by the Limited Relative Error Algorithm

Markus Becker, Thushara Lanka Weerawardane, Xi Li, Carmelita Görg
Communication Networks, TZI ikom, University of Bremen
28334 Bremen, Germany
E-mail: {mab|tlw|xili|cg}@comnets.uni-bremen.de

Abstract

Pseudo Random Number Generators (PRNG) are the base for stochastic simulations. The usage of good generators is essential for valid simulation results. OPNET Modeler a well-known tool for simulation of communication networks provides a Pseudo Random Number Generator. The extension of OPNET Modeler with external generators and additional statistical evaluation methods that has been performed for this paper increases the flexibility and options in the simulation studies performed.

Introduction

Simulative analysis of complex systems is generally regarded as a scientific method. Stochastic simulations are based on sources of randomness. The randomness can be truly random [2, 3], but is usually created using algorithms called Pseudo Random Number Generators (PRNG). These algorithms do not create true random numbers. Many algorithms for PRNGs have been proposed and have been implemented [5]. Depending on the properties of the PRNGs [7], some of them are more or less suited for usage in simulations [8]. There are many different tests for PRNGs. Good PRNGs and their correct usage is essential for good and valid simulation results [3]. Every simulation environment provides PRNGs for the usage in simulation. In this paper the focus is on the well-known and often used simulation environment OPNET Modeler for communication networks and its Pseudo Random Number Generators.
In the following sections the OPNET Modeler PRNG and its limitations are introduced, other available PRNG implementations and statistical evaluation libraries are introduced and an integration of those into the OPNET Modeler is described. An application of the PRNGs and statistical evaluation algorithm into an M/M/1 model is evaluated with regard to the simulation time needed to fulfill the requirements of a given certain maximum relative error.

OPNET PRNG

According to the documentation of the OPNET Modeler [15] the PRNGs in use in the tool are based on the operating systems random() implementation as shown in Table 1.

Operating System	**Random Number Generator**
Sun Solaris	random()
Microsoft Windows	random() (an OPNET port from the UNIX BSD source distribution)

Table 1: Random Number Generators

The usage in simulations is provided by the kernel procedures op_dist_uniform() and op_dist_outcome(). The underlying PRNG is initialized using the seed environment variable, which can be changed in the *'Configure/Run DES'* dialog. In order to create new instances of the PRNG the kernel procedure op_prg_random_gen_create() can be used.

Limitations of the provided PRNG

The PRNG provided by the OPNET Modeler has at least four limitations as described in the following paragraphs.

Uncertainty of the PRNG

Although, as stated in the previous paragraph OPNET is using the BSD random() PRNG, it seems to be a customized version of the BSD random() PRNG. Additionally, it is not clear which version of the BSD random() PRNG it is based on as according to [18] there are at least 3 different versions (SunOS4, libc5, glibc2), which differ in the seeding procedure of the PRNG.

The PRNG fails tests

According to [16] the BSD PRNGs fails at least the p-value test, which checks the uniformity of the values the PRNG outputs. PRNGs which fail tests should not be used, as several others are passing the test suites [2].

Limited number of provided PRNGs

The OPNET Modeler provides only one PRNG. However, simulation results are doubtful if created with only one type of PRNG according to [19].

No independence of sequences

If there is a need for complete independence of random number sequences, which is usually the case, an external PRNG needs to be used, according to the tools' documentation [15]. The kernel procedure op_dist_outcome_ext() can be used to get non-uniformly distributed random numbers using an external PRNG. However, the OPNET Modeler's documentation does not specify how to include external PRNGs into the models.

Available PRNG implementations

Implementations of PRNG algorithms can be found in many locations of the Internet. Two sources of implementations of PRNGs are introduced in more detail: the Communication Networks Class Library (CNCL) [12] and the GNU Scientific Library (GSL) [13].

Communication Networks Class Library

The CNCL is a C++ library created by Communication Networks, Aachen University of Technology, Germany. It provides universal classes, event driven simulation classes, Pseudo Random Number Generators and distribution classes.
The PRNGs have the common super class CNRNG. Currently the CNCL provides the following PRNGs: additive random number generator (CNACG), Fibonacci random number generator (CNFiboG), data file random number generator (CNFileG), linear congruential generator (CNLCG), multiple linear congruential generator (CNMLCG), Tausworthe generator (CNTausG).
Additionally the CNCL provides statistical evaluation classes, which are derived from CNStatistics: batch means, various limited relative error (LRE) algorithms, histograms and moments.
It has been compiled using the GNU compiler collection (gcc) [20] on UNIX operating systems.

GNU Scientific Library

The GNU Scientific Library (GSL) is a powerful and extensive numerical library for C and C++ programmers. Included in the library are algorithms for complex numbers, vectors and matrices, fast Fourier transforms, histograms, statistics and differential equations among many others. Additionally there are also algorithms for the generation of pseudo random numbers and distributions. The random number generators included are Mersenne Twister, RANLUX, CMRG, MRG, Tausworthe, GFSR, the UNIX PRNGs and many others. A Windows version of the GSL is available at [22].

Approaches to the Integration of external PRNGs and statistical evaluation

In order to diminish the drawbacks of the OPNET Modeler's PRNG and to add functionality with respect to the statistical evaluation, the coupling of external libraries and OPNET Modeler is investigated in this paper.
As a first step the integration of the library CNCL and the GSL into OPNET Modeler has been performed. There are different possibilities of integrating external PRNGs and statistical evaluation into OPNET Modeler. In the following we are detailing four approaches.

Approach 1:

The first approach is to use the source code of the CNCL or GSL and integrate it into OPNET process models. For this approach the code needs to be cumbersomely copied and adopted. When integrated into the process models it is compiled and linked in conjunction with the process models. This approach is viable for OPNET on Linux as well as for Windows operating system.

Approach 2:

OPNET Modeler provides a way to use other compilers than the usual Microsoft Visual Studio tools. This second approach uses OPNET External Compiler Interface (ECI) as described in [22] and GCC/mingw. ECI is a specification for interfacing OPNET with third-party compilers and linker programs. This approach is viable for OPNET on Linux as well as for Windows operating system.

Approach 3:

The Linux version of the OPNET simulator could be used. PRNGs and statistical distributions of CNCL could be integrated into the OPNET simulator as a link library. In this case, the task is rather direct because, CNCL, GSL and OPNET are compiling with the GCC compiler. However, this approach is fixed to the operating system Linux.

Approach 4:

The compilation and linking of the libraries is done using Microsoft Visual Studio tools on the Windows platform in the fourth approach. In order to do so, the CNCL library needs to be slightly adapted as until now it has been compiled using the GNU compiler tools. The GSL is available as a Windows library [22].

Becker, M.; Weerawardane, T.L.; Li, X.; Görg, C.: Extending OPNET Modeler with External Pseudo Random Number Generators and Statistical Evaluation by the Limited Relative Error Algorithm. In: Ince, N.; Bragg, A. (eds.): Recent Advances in Modeling and Simulation Tools for Communication Networks and Services. Springer, 2007

Integration of external PRNGs and statistical Evaluation

As a first way to get independent PRNGs into OPNET approach 1 was chosen. During this research progress was made with regard to approach 4. In the following details on the inclusion of PRNGs using approach 4 are given. The advantages of approach 4 are that it does not involve adaptation of source code by hand, a different operating system and the creation of an external compiler interface. The following steps need to be performed to integrate CNCL/GSL and OPNET:

1. Compilation of the CNCL with the Visual C++ compiler and creation of a link library of the CNCL

In order to create a link library, the Configuration Type needs to be set to Static Library (.lib), in the general project properties. Additionally the usage of the Microsoft Foundation Class needs to be specified as a shared library in the properties. The definitions -DCNCL_CC_GCC32 -DCNCL_HAS_BOOL need to be set. A binary library of the CNCL has been made available at [24].

2. Inclusion of the header files and the library into the OPNET simulation models

Include the following in the source code header block of the process model to indicate to OPNET to use the C++ compiler:

```
OPC_COMPILE_CPP
```

Furthermore, the inclusion of the PRNGs header file and the definition of the variables are needed. Here is an example for the CNCL and GSL PRNG and distributions:

```
#include <CNCL/TausG.h>
#include <CNCL/NegExp.h>

#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>

CNTausG* t_rng;
CNNegExp* cncl_negexp;

gsl_rng* gsl_opnet_rng;
```

In the init enter execs the pseudo random number generators and distributions need to be created.

```
t_rng = new CNTausG();
t_rng->seed(seed);
cncl_negexp = new CNNegExp(1, t_rng);

gsl_opnet_rng = gsl_rng_alloc (gsl_rng_taus2);
gsl_rng_set(gsl_opnet_rng, seed);
```

The following code block shows the usage of the different PRNGs:

```
if (rng_type == OPNET) {
      next_intarr_time = oms_dist_outcome (interarrival_dist_ptr);
} else if (rng_type == CNCL) {
      next_intarr_time = (*cncl_negexp)();
} else { //GSL
      next_intarr_time = gsl_ran_exponential (gsl_opnet_rng, 1);
}
```

For the usage of the LRE algorithms the following additions to the header code block need to be made:

```
OPC_COMPILE_CPP

#include <CNCL/DLREF.h>

CNDLREF* delay_lre = new CNDLREF(0.0, 100.0, 1, 0.05);
CNDLREF* size_lre  = new CNDLREF(0.0, 100.0, 1, 0.05);
```

To use the LRE, the values need to be put into the object:

```
size_lre->put(op_subq_stat (0, OPC_QSTAT_PKSIZE));
```

3. Changes to the preferences need to be made to enable compilation and binding of the simulator

Edit the preferences to meet the settings listed in Table 2 adapted to the paths on your system. After changing the preferences all files need to be saved and compiled by executing 'DES -> Configure/Run Discrete Event Simulation'. At the advanced tab the model recompilation needs to be forced.

comp_flags_common	-IC:\Docume~1\mab\MyDocu~1\src\cncl-2.8\include -IC:\Progra~1\GnuWin32\include -DCNCL_CC_GCC32 -DCNCL_HAS_BOOL
bind_shobj_libs	C:\Docume~1\mab\MyDocu~1\Visual~1\CNCL_M~1\Debug\CNCL_M~1.lib C:\Progra~1\GnuWin32\lib\libgsl.a
bind_static_flag	/LIBPATH:C:\PROGRA~1\OPNET\10.0.A\sys\pc_intel_win32\lib /NODEFAULTLIB:library
repositories	(Empty) not (stdmod)

Table 2: Preferences Settings

Limited Relative Error

The limited relative error has been described in several publications [25], [26], [27], [28]. There are three different versions of the algorithm. LRE I is used for an independent, continuous x-sequence; LRE II for a correlated, continuous x-sequence and the LRE III is for a correlated, discrete x-sequence.
The algorithm uses local mappings to 2-node Markov Chains to calculate the local correlation coefficients and thus the relative error. Shown below is exemplary output of the algorithm. It calculates the cumulative distribution function (CDF), the relative error and correlation coefficient.

```
#LRE RESULT (THIS IS A MAGIC LINE)
#-------------------------------------------------------------------------
#    Discrete Lre --- distribution function
#-------------------------------------------------------------------------
#Name: No name
#Text: No description
#Default values: max.rel. error =    5%  X_min: 0  X_max: 100
#trials: 24983    mean: 10.84 variance: 96.62
#trials < X_min: 0      trials > X_max: 0
#All levels calculated.
#-------------------------------------------------------------------------
#F(x)           x               rel.error       rho             sigrho
0.0000000e+000  0.0000000e+000  0.0000000e+000  0.0000000e+000
3.8426130e-002  0.0000000e+000  4.6821223e-002  3.7277310e-001  1.5821438e-002
3.8426130e-002  1.0000000e+000  4.6821223e-002  3.7277310e-001
1.1431774e-001  1.0000000e+000  3.9807701e-002  6.7266369e-001  8.5854153e-003
1.1431774e-001  2.0000000e+000  3.9807701e-002  6.7266369e-001
1.8196374e-001  2.0000000e+000  3.9791740e-002  7.9590151e-001  5.6872325e-003
1.8196374e-001  3.0000000e+000  3.9791740e-002  7.9590151e-001
2.4692791e-001  3.0000000e+000  3.7231946e-002  8.3812994e-001  4.4035090e-003
2.4692791e-001  4.0000000e+000  3.7231946e-002  8.3812994e-001
```

Application of the external PRNGs and the LRE in Simulations

In order to show the impact of using external PRNGs and external statistical evaluation tools, an M/M/1 model as described in the Tutorial of the OPNET Modeler's documentation [29] is used. The model is depicted in Figure 1 and consists of a source of packets, a queue and a sink, which models the serving process. All settings are according to the tutorial, the changes that where introduced are the usage of different PRNGs, and the evaluation of the number of packets and the delays using the limited relative error (LRE) algorithm. The PRNGs in use are the original OPNET PRNG, the CNCL Tausworthe PRNG and the GSL Tausworthe PRNG.

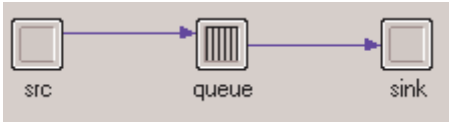
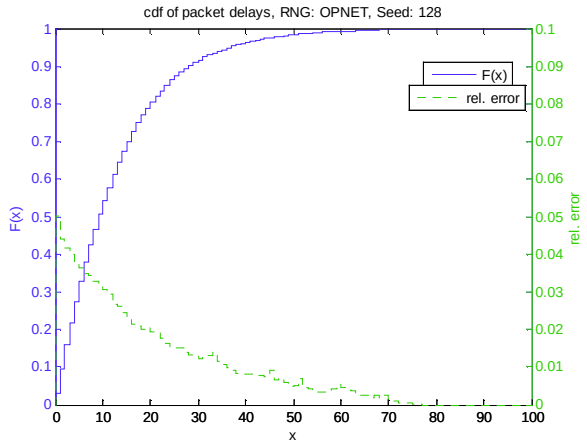


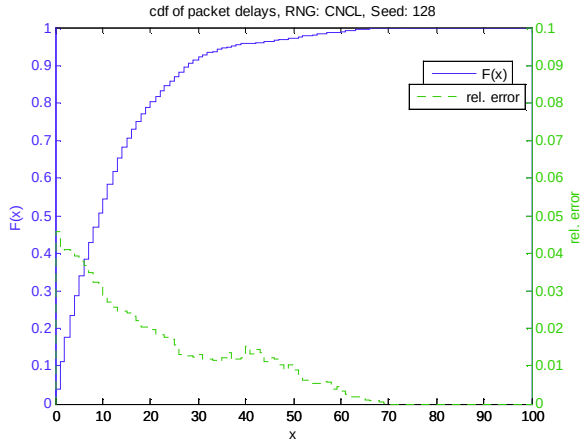
Figure 1: M/M/1 OPNET Model

The cumulative distribution function (CDF) of the packet delay is shown in Figure 2, while the number of packets in the queue is shown in Figure 3. Additionally, the relative error is shown in the Figures. The simulations were run with a fixed seed of 128 for all PRNGs. The model time simulated is 7 hours as described in the tutorial. When comparing Figures 2(a)-(c), it can be seen that the OPNET PRNG has a higher relative error at low packet delay times, where the error is in high regions for all PRNGs. This region

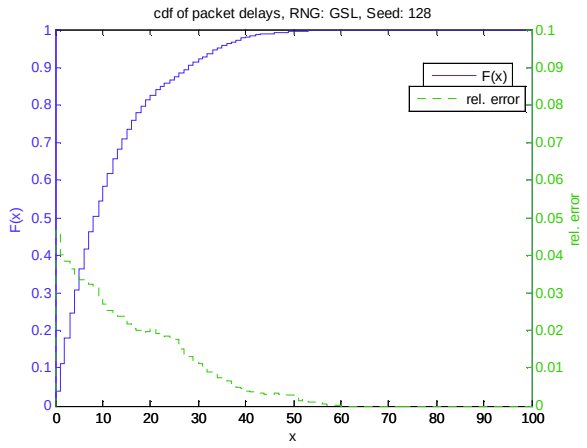
would determine the time needed to simulate with a limited relative error. In the preset time of 7 hours model time, the relative error could not go below a value of 0.05. This means, that the model time has not been long enough to be statistically sound with respect to an error level of 0.05. With respect to the number of packets in the queue, all PRNGs could limit the relative error to 0.05 in the given model time.



(a) OPNET

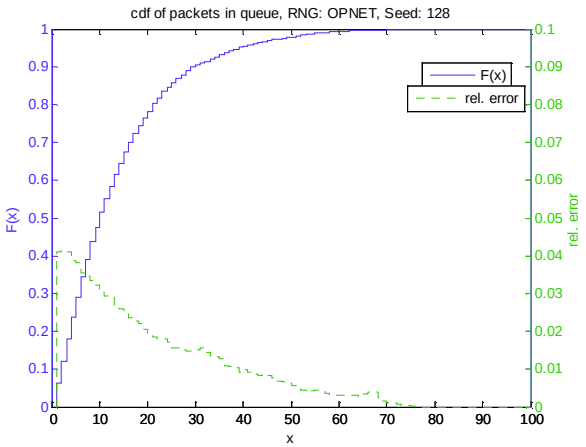


(b) CNCL

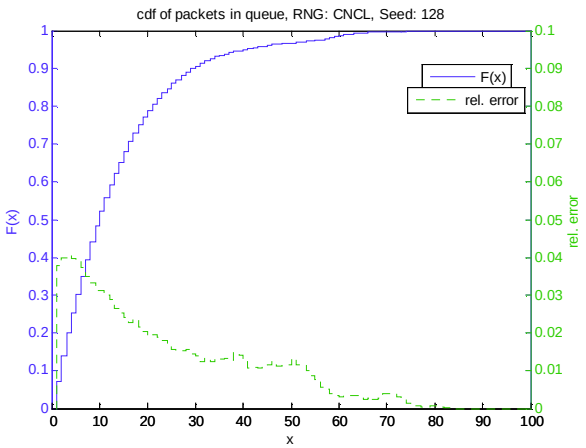


(c) GSL

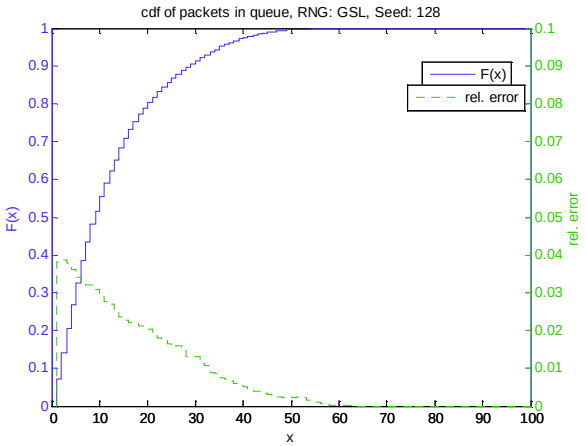
Figure 2: CDF and relative error of packet delays



(a) OPNET



(b) CNCL



(c) GSL

Figure 3: CDF and relative error of number of packets in the queue

In order to compare the model time needed to limit the relative error to 0.05, simulations of the earlier mentioned model have been undertaken. The simulations are using the three different PRNG algorithms and each PRNG is run 88 times with randomly chosen seeds. The seeds were created using the Linux /dev/random device that creates random numbers within the estimated number of bits of noise in its entropy pool [23].The model time was again limited to 7 hours (25200s).
Figure 4 depicts the number of simulations with a maximum relative error below 0.05 with respect to the simulated model time. In the first 10000s no simulation has a relative error below 0.05. At the end of the simulated model time 10% of the simulations with the OPNET PRNG have not reached the desired error level, while for the CNCL and GSL PRNG algorithms there are less simulations remaining with a higher error level than 0.05. Furthermore, it can be seen that both external PRNGs have a higher amount of simulations which have the desired error level when compared with the original OPNET PRNG.
With all PRNGs the area below the curves could be employed for reducing the time needed to simulate the model, as the desired error level has been reached. The LRE algorithm in the CNCL supports the stopping of simulations that have reached the desired error level.

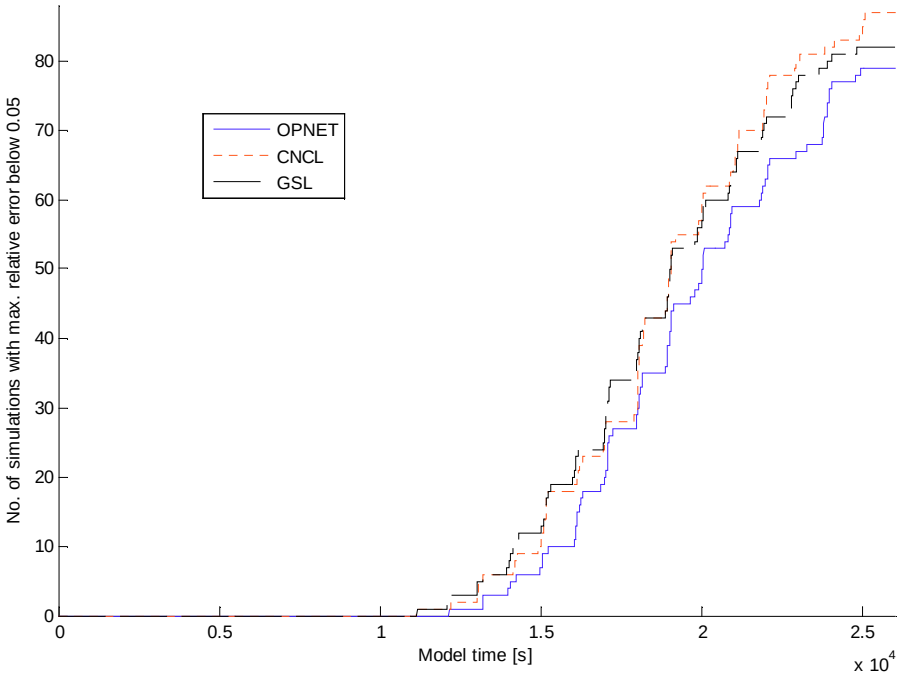


Figure 4: Number of simulation runs with a maximum relative error of 0.05

Conclusion & Outlook

In this paper the PRNG used in the well-known simulation tool OPNET Modeler was presented. The limitations of using only one PRNG in this tool were discussed. As a way to overcome these limitations 4 different approaches are proposed in this paper and detailed information is provided on how to integrate external algorithms, including PRNGs and the LRE statistical evaluation, into the tool. The results by the different PRNGs are evaluated using the LRE algorithm. It was shown that the usage of external PRNGs in combination with the LRE algorithm can reduce the computation time for simulations.
In future the external PRNG and additional statistical evaluation methods will be used in simulations of UMTS radio network simulations as well as simulations that evaluate the usage of mobile radio communications for logistical processes.

Acknowledgements

This research was supported by the German Research Foundation (DFG) as part of the Collaborative Research Centre 637 “Autonomous Cooperating Logistic Processes”.

References

[1] Krzysztof Pawlikowski, Hae-Duck Joshua Jeong, Jong-Suk Ruth Lee: On credibility of simulation studies of telecommunication networks. IEEE Communications Magazine, vol. 40, no. 1, Jan 2002 pp. 132-139.
[2] P. L'Ecuyer: Random Number Generation. Chapter 4 of the Handbook on Simulation, Jerry Banks Ed., Wiley, 1998, 93—137.
[3] P. L'Ecuyer: Software for Uniform Random Number Generation: Distinguishing the Good and the Bad. Proceedings of the 2001 Winter Simulation Conference, IEEE Press, Dec. 2001, 95-105.
[4] P. Hellenek: Vorsicht: Zufallszahlen!, International Mathematische Nachrichten, Nr. 180, April 1999.
[5] M. Matsumoto and T. Nishimura: Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 (1998).
[6] Webpage: Theory and Practice of Random Number Generation. http://random.mat.sbg.ac.at/

[7] Webpage: Random Number Generation and Testing. http://csrc.nist.gov/rng/
[8] Webpage: True Random Numbers. http://www.random.org/
[9] Webpage: True Deterministic Numbers. http://www.noentropy.net/
[10] M. Junius, M. Steppler, M. Büter, D. Pesch, and others, Communication Networks Class Library, Edition 1.10, 1998.
[11] Webpage: GNU Compiler Collection (gcc). http://gcc.gnu.org/
[12] Webpage: Communication Networks Class Library (CNCL). https://www.comnets.uni-bremen.de/typo3site/index.php?id=31
[13] Webpage: GSL - GNU Scientific Library http://www.gnu.org/software/gsl/
[14] OPNET technologies Inc., OPNET Modeler Accelerating Networks R&D, www.opnet.com
[15] OPNET Modeler Product Documentation Release 10.0, pages MC-11-26 – MC-11-27.
[16] Pierre L'Ecuyer, Richard Simard: TestU01: A C Library for Empirical Testing of Random Number Generators. May 2006, Revised November 2006, ACM Transactions on Mathematical Software, to appear.
[17] GNU Scientific Library Reference Manual - Revised Second Edition, http://www.gnu.org/
[18] M. Galassi et al: GNU Scientific Library Reference Manual. Revised Second Edition, Section 17.10: Unix random number generators, ISBN 0954161734.
[19] P. Hellekalek: Good random number generators are (not so) easy to find. In Mathematics and Computers in Simulation (1998).
[20] Webpage: GNU compiler collection: http://gcc.gnu.org
[21] OPNET Modeler Product Documentation Release 10.0, pages MC-11-26 – MC-11-27. OPNET Modeler Product Documentation Release 10.0, external reference manual, pages E-6-1 to E-6-5.
[22] Webpage: Windows Version of the GSL: http://gnuwin32.sourceforge.net/packages/gsl.htm
[23] Linux Programmer's Manual RANDOM(4), 2003-10-25.
[24] Webpage: Windows binary of the Communication Networks Class Library: http://www.comnets.uni-bremen.de/~mab/opnet/cncl/
[25] F. Schreiber, C. Görg: Rare Event Simulation: A Modified RESTART-Method using the LRE-Algorithm. Teletraffic and Datatraffic, Proceedings 14th ITC, Antibes, Juan-Les-Pins, France, June 6-10, 1994, 787–796, 1994.
[26] F. Schreiber, C. Görg: Stochastic Simulation: a Simplified LRE-Algorithm for Discrete Random Sequences. AEU, 50:233–239, 1996.
[27] C. Görg, F. Schreiber: Der LRE -Algorithmus für die statistische Auswertung korrelierter Zufallsdaten. In F. Breitenecker, I. Troch, P. Kopacek, (Eds.), Simulationstechnik, 170–174, Wiesbaden, 1990. 6. ASIM - Symposium, Wien 1990, Vieweg.
[28] C. Görg, F. Schreiber: The RESTART/LRE Method for Rare Event Simulation. In 1996 Winter Simulation Conference, 390–397, Coronado, California, USA, December 1996.
[29] OPNET Modeler Product Documentation Release 10.0, General Tutorials, M/M/1 Queue, pages 1 – 42.