A simple OMNeT++ queuing experiment using different random number generators

Bernhard Hechenleitner and Karl Entacher

December 5, 2002

Abstract

We apply a simple queuing-experiment using parallel streams of random numbers to exhibit shortcomings of the OMNeT++ random number generator. As an improvement we implement more modern generators.

1 Introduction

OMNeT++ [20] is an object-oriented discrete event simulation system based on C++. It is primarily designed to simulate computer networks, multi-processors and other distributed systems. The basic development of OMNeT++ began at the Technical University of Budapest (BME) in 1992. Currently, OMNeT++ is being used by dozens of universities and companies as a research tool, for validating hardware and protocol designs, and for performance evaluations. OMNeT++ is a non-commercial, open-source project. It is easy to integrate new components or alter current implementations of components within its object-oriented architecture. OMNeT++ has been extended to support parallel stochastic discrete event simulation. Several synchronization mechanisms can be used. One suitable synchronization mechanism is the statistical synchronization, for which OMNeT++ provides explicit support. Further details on this issue can be found in the OMNeT++ on-line manual [20].

In the present paper we apply a simple queuing-experiment using parallel streams of random numbers to exhibit important shortcomings of the OMNeT++ random number generator (RNG). As an improvement we implement more modern RNGs, one of them supports well tested parallel streams. The following two sections contain a description and the properties of these RNGs. The simulation experiments and the corresponding results are described in Sections 4 and 5.

2 OMNeT++ Random Number Generator

OMNeT++ implements the well-known “minimal standard” generator, which was originally suggested for the IBM System/360 by Lewis, Goodman and Miller in 1969 [15]. It was examined in more detail in Park and Miller [19] and further on in several other studies on random number generation. We will denote this RNG as `ran0`.

This generator is a multiplicative linear congruential type [6, 8, 10, 11, 18] which produces pseudorandom integers via the recursion

$$x_n \equiv a \cdot x_{n-1} \pmod{m}, \quad n \geq 1, \tag{1}$$

with multiplier $a = 7^5 = 16807$, modulus $m = 2^{31} - 1 = 2147483647$, and seed $1 \leq x_0 < m$. The period length of this recursion equals $p = m - 1$. Uniform pseudorandom numbers in $[0, 1)$ are derived by transformation $u_n = x_n/m$, non-uniform distributions by different transformation methods [3].

This particular generator has widely been used and actual implementations are available from the Internet. See [1, 6, 8, 10, 11, 12, 13, 14, 19] for references, empirical tests and implementations in free and commercial software. The following online resources contain related material: Resampling Stats (`www.resample.com`), Numerical Recipes (`www.nr.com`), the mathematical software MATLAB (`www.mathworks.com`), the IMSL Libraries, or the simulation software ACSL (`www.acslsim.com`), SIMAN/Arena, Slam II, AweSim (`www.pritsker.com`) and the network simulation software ns-2 (`www.isi.edu/nsnam/`).

A first problem of `ran0` is the period length $p = 2^{31} - 2$ which is far too short for actual simulations, especially when several parallel streams of random numbers are applied.

The standard OMNeT++ implementation provides 32 RNG objects, each stream with an initial seed $x_0 = 1$ by default. The user is asked to previously initialize the streams with certain seeds, *otherwise equal random number streams will be generated* from each object.

One has to be very careful when manually seeding parallel streams. For example, using the seeds $x_{i,0} = i$, $i = 1, 2, \ldots k$ would result in the following k random number streams which are heavily correlated

$$x_{i,n} \equiv a^n \cdot x_{i,0} \pmod{m} \quad n \geq 0, \quad 1 \leq i \leq k. \tag{2}$$

These correlations can easily be shown from the vectors

$$\vec{x}_n := (x_{1,n}, x_{2,n}, \ldots, x_{j,n}) \equiv a^n \cdot (1, 2, \ldots, j) \pmod{m}, \quad j \leq k, \ n \geq 0. \tag{3}$$

Since $a^n \pmod{m}, n \geq 1$ cycles all numbers in $\{1, 2, \ldots m - 1\}$, the vectors above are contained in the set $\{n \cdot (1, 2, \ldots, j) \pmod{m} \ : \ 1 \leq n \leq m - 1\}$. Therefore the

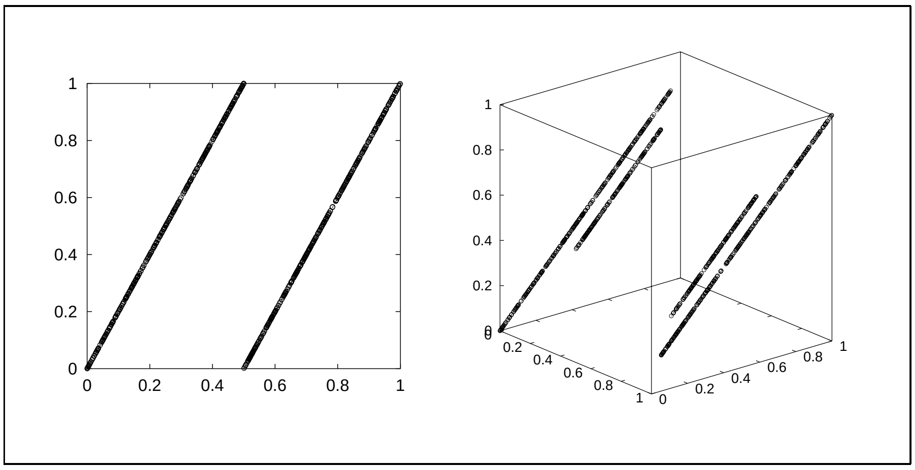


Figure 1: Correlations between two and three different streams of random numbers generated by the OMNeT++ RNG.

normalized vectors $\vec{u}_n := \vec{x}_n/m, n \geq 1$ are situated in a lattice structure in the unit square $[0,1)^2$ consisting of a few lines only, see Fig. 1.

In the following section we will show that such correlations can result in completely biased results even in simple OMNeT++ simulation examples.

Unfortunately not only simple seeding procedures produce strongly correlated streams of random numbers. There are many possible seed combinations where the corresponding streams are heavily correlated. Further examples are seeds $x_{i,0} = k \cdot i$, $i \geq 1$, $k \in \mathbf{Z}$, or all combinations of seeds consisting of very small numbers.

If special seeds are used, so that the full period of the RNG is divided into large blocks, and if each of these blocks will be used as a single stream of random numbers,then strong correlations will appear as well. This property is well known as long-range correlations[1] of linear random numbers, see [2, 4, 5] and the references given there. As an example, Seed Set 4 in Table 1 divides the period p into 5 large blocks.

The quality of linear random number generators and their parallelization has theoretically and empirically been studied in detail. For an overview and references see the surveys contained in [6, 7, 11, 12, 18].

[1]The term long-range correlations may be slightly misleading since it also appears in the theory of stochastic processes. In our context it refers to a geometric property of linear random number generators.

3 Modern Generators

A modern object-oriented RNG which supports well-tested parallel streams of random numbers was implemented by L'Ecuyer et al. [13]. The source-code of this RNG can easily be included in OMNeT++ simulations. The provided C++ files, which implement the class `RngStream`, have to be included in the same directory where all the other files for the simulations are contained. For our simulations, the RNG objects were invoked by the generator objects. Therefore when executing the OMNeT++ scripts `opp_makemake -f` and `make` in the simulation directory, the RNG is compiled together with all the other simulation components. As exponentially distributed random numbers were needed, but the offered RNG package does not provide them, a respective function had to be added to the class `RngStream`. In the following sections, this RNG is denoted as `RandU01`.

The Mersenne Twister (`MT`) [16] is another up-to-date random number generator. This generator is very fast, has a huge period ($2^{19937}-1$) and is known to be theoretically and empirically well tested. Source code of `MT` for several programming languages is freely available at the Mersenne Twister home page [17]. For the integration of `MT` as an OMNeT++ component, the standard `MT` `c`-codes (`mt19937int.c` and `mt19937-2.c`) were used.

4 Simulation Topology

Although there could be many possible topologies and situations where correlations between parallel streams of random numbers could lead to wrong simulation results, a very simple example was chosen for examinations of bad effects due to correlations. The intention of this was to highlight that bad effects already manifest in simple simulation scenarios.

Figure 2 shows the topology which was chosen for the simulations. Job streams of 5 exponential generators (`Expo 1` to `Expo 5`) are aggregated at `FIFO`, a simple First In First Out buffer with a buffer size of 1000 jobs. `FIFO` is connected to `Sink`, which does nothing else than absorbing the jobs, which are handed over by the service entity of `FIFO`. The 5 exponential generators produce streams of jobs with exponential inter-arrival times and `FIFO` offers an exponentially distributed service time for each job. Therefore, the resulting simulation topology constitutes a M/M/1/1001 queuing system.

For the simulations done, each of the exponential traffic generators `Expo 1` to `Expo 5` had the same parameter settings. The mean of the exponentially distributed time between the generation of two successive jobs (inter-arrival time) was set to $41\,ms$. Each of the 5 job streams produced by the generators `Expo 1` to `Expo 5`

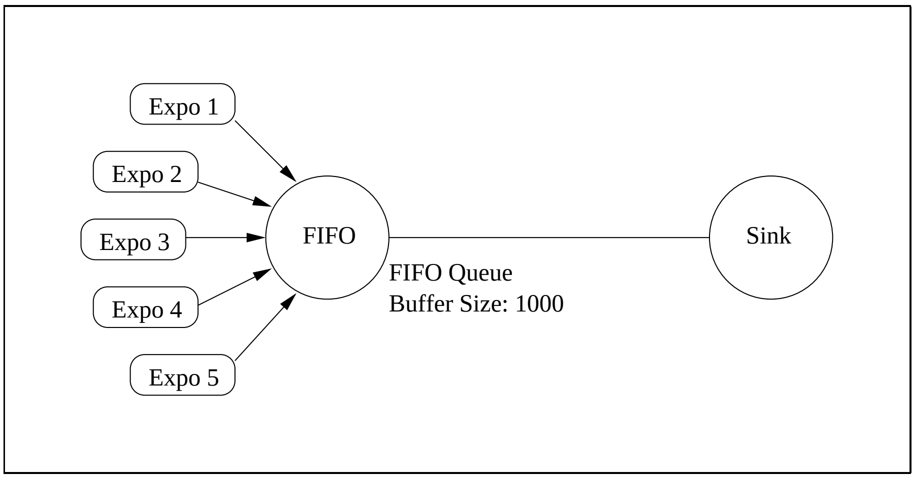


Figure 2: Simulation topology.

can be seen as a Markov Process with an average arrival rate of $\lambda = \frac{1\,job}{41\,ms}$. When aggregating Markov Processes, their average arrival rates may simply be added to get the average arrival rate of the aggregated Markov Process. Therefore the aggregated Markov Process, which arrives at the buffer of `FIFO`, has an average arrival rate of $\lambda_{sum} = 5 \cdot \lambda = \frac{1\,job}{8.2\,ms}$. The service times of `FIFO` for the arriving jobs are exponentially distributed with a mean of $8\,ms$, thus the mean service rate of `FIFO` is $\mu = \frac{1\,job}{8\,ms}$. Summing up, `FIFO` can be seen as a M/M/1/1001 queuing system with a utilization factor of $\rho = \frac{\lambda_{sum}}{\mu} = 0.97561$.

Referring to the theoretical M/M/1 model in [9, 10], the mean number of jobs in the system $\overline{N}$ calculates as $\overline{N} = \rho/(1-\rho)$. Considering a M/M/1 queuing system with a utilization factor of $\rho = 0.97561$, the average number of jobs in such a system would therefore be $\overline{N} = 40.0004\,jobs$. The distribution of the number n of jobs in the system is given by the geometric distribution $p(n) = (1-\rho)\rho^n$.

5 Simulation Results

Regarding the simulations, each of the 5 exponential generators used its own RNG object. Different sets of seeds were used for seeding the 5 RNG objects. For each simulation, the average number of jobs in the system $\overline{N}$ was calculated. Furthermore, the CDF of the number of jobs was determined and plotted for each simulation run. The simulation time was set to $40000\,s$ for each run, which is 975610 times the average inter-arrival time of $41\,ms$ between two successive jobs of each generator.

	Seed Set 1	Seed Set 2	Seed Set 3	Seed Set 4
seed 1	1	934100682	1	1
seed 2	1140279430	558746720	2	634005912
seed 3	1328964554	2081634991	3	634005911
seed 4	310283214	144443207	4	2147483646
seed 5	488838574	513791680	5	151347773
estim. $\bar{N}$	39.66	38.78	44.39	37.73

Table 1: Seed sets used for the OMNeT++ default RNG `ran0`.

5.1 Results for `ran0`

For the simulations using the default RNG of OMNeT++ (`ran0`), the seed sets from Table 1 were used. The first seed set produces consecutive blocks of random numbers of length 2 million. The seeds from Seed Set 2 are taken from the default seeds implemented in the `ns-2` network simulator which implements the same RNG as OMNeT++. Both seed sets ('good' seeds) are expected to not produce correlated streams. The remaining two seed sets ('bad' seeds) produce strongly correlated random number streams, see Sect. 2.

The resulting mean values of the number of jobs in the system are given in Table 1. The values for 'good' seeds are close to the expected value whereas the mean values for 'bad' seeds show stronger deviations from the theoretical $\bar{N}$. Figure 3 shows more detailed information. It compares the theoretical vs. empirical CDFs for the simulations with 'good' seeds and 'bad' seeds. As can be seen easily, the CDFs using 'good' seeds almost exactly match the expected theoretical CDF. On the other hand, the resulting CDFs using 'bad' seeds show significant deviations.

We carried out several simulations with increasing simulation time . All results showed the same behavior as in Figure 3.

5.2 Results for `RandU01`

The seeding procedure for `RandU01`is well defined. The first `RandU01`object, which is used, has to be initialized using an integer vector of length 6 as seed. The initial states of all following `RandU01`RNG objects are generated automatically.

Ten simulations with equal simulation time $t = 40000$ s were carried out using the randomly chosen initial vectors given in Table 2. The average of the mean values of these simulations equals 40.336 with a standard deviation of 0.7419 which results in a 99% confidence interval $[39.5736, 41.0984]$ for $\bar{N}$.

The upper graphics in Figure 4 shows the empirical CDFs of our first five simulations. The obtained CDFs match the theoretic CDF very well (same for the

1412384165	236002145	542159054	521487915	2018087536	414877493
2048888926	655005779	906412827	235849136	1209759501	1676730349
694867568	1899502794	83494906	1475474107	1915987305	2004754650
2074490988	826814765	1119975261	2085427774	1708126621	2029829150
1563149874	1863670447	1639687603	184913654	1409851396	119692977
1305470914	1017386359	1041111727	655198235	699830490	2020995355
982430240	854044306	312611657	750371147	1503044684	744590471
1738868915	1211171448	1524311206	1550825011	554302720	46329467
1244029748	1547483752	1426952987	997595553	631930979	1896589808
1427162620	1470904152	1984561598	1995777718	3457391	1495308727

Table 2: Seeds used for simulation with `RandU01` and `MT`.

remaining 6 simulations). In addition we carried out simulations with increasing simulation time $t_i = 2^i\, s, 11 \leq i \leq 20$.

Note that it is highly recommended to apply the described automatic seeding procedure of `RandU01`. If all used RNG objects are initialized by the user, and if the user chooses bad seed vectors for initializing the RNG objects, the simulation may yield poor simulation results. As an example, one may apply the seed vectors $\{j, j, j, j, j, j\}$, $1 \leq j \leq 5$ for the initialization of our five streams. The parallel random number streams obtained applying these special seeds are highly correlated, similar as Seed Set 3 for `ran0`. The verification for this may be done in a similar way as for `ran0` in Section 2.

5.3 Results for `MT`

The Mersenne Twister has no explicit seeding procedure for parallel streaming. It is often recommended to use random seeds for each stream, since for the huge period of the generator it is unlikely to get overlapping streams.

Again ten simulations with equal simulation time $t = 40000\, s$ were carried out using the first five numbers from each row in Table 2 to initialize the five streams of our simulation setup. The average of the mean values of these simulations equals 39.2224 with a standard deviation of 0.9003 which results in a 99% confidence interval $[38.2971, 40.1477]$ for $\overline{N}$.

The lower graphics in Figure 4 shows the empirical CDFs of our first five simulations. Results from further simulations with increasing simulation time $t_i = 2^i\, s, 11 \leq i \leq 20$ are presented below.

5.4 Increasing Simulation Time

In addition to the examples above, we applied simulations with increasing time. We have varied the simulation time from 2^{11} s to 2^{20} s. For each simulation time, 10 simulations with different seeds were carried to get mean values and error bars for the mean number of jobs in the system.

The seeds for `ran0` have appropriately been chosen to asure nonoverlapping consecutive blocks of random numbers for all simulation times. These same seed values were taken to initialize the simulations with `MT`. Similar as in Subsection 5.2 the initializing seed vectors for `RandU01` were generated randomly using `ran0`.

See Figures 5, 6 and 7 for the results.

Acknowledgments

The authors are supported by the Austrian Science Fund (FWF) Grant S8311-MAT.

References

[1] R.F. Boisvert, M. McClain, and Miller B. GAMS The Guide to Available Mathematical Software. National Institute of Standards and Technology, Gaithersburg, MD, USA, 1998. `http://math.nist.gov/gams/`.

[2] A. DeMatteis and S. Pagnutti. Long-range correlations in linear and non-linear random number generators. *Parallel Comput.*, **14**:207–210, 1990.

[3] L.P. Devroye. *Non-Uniform Random Variate Generation*. Springer, New York, 1986.

[4] M.J. Durst. Using linear congruential generators for parallel random number generation. In E.A. MacNair, K.J. Musselman, and P. Heidelberger, editors, *Proceedings of the 1989 Winter Simulation Conference*, pages 462–466, 1989.

[5] K. Entacher. Parallel streams of linear random numbers in the spectral test. *ACM Transactions on Modeling and ComputerSimulation*, **9**(1):31–44, 1999.

[6] G.S. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*, volume 1 of *Springer Series in Operations Research*. Springer, New York, 1996.

[7] P. Hellekalek and G. Larcher (eds.). *Random and Quasi-Random Point Sets*, volume **138** of *Lecture Notes in Statistics*. Springer, Berlin, 1998.

[8] R. Jain. *The art of computer systems performance analysis*. John Wiley & Sons, New York, 1991.

[9] L. Kleinrock. *Queueing Systems. Volume I: Theory*. John Wiley & Sons, New York, 1975.

[10] A.M. Law and W.D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, 2 edition, 1991.

[11] P. L'Ecuyer. Uniform random number generation. *Ann. Oper. Res.*, **53**:77–120, 1994.

[12] P. L'Ecuyer. Software for uniform random number generation: Distinguishing the good and the bad. In *Proceedings of the 2001 Winter Simulation Conference*, 2001.

[13] P. L'Ecuyer, R. Simard, E.J. Chen, and Kelton W.D. An object-oriented random-number package with many long streams and substreams. Manuscript and source-code from `http://www.iro.umontreal.ca/˜lecuyer/`; to appear in *Operations Research.*, 2002.

[14] H. Leeb and S. Wegenkittl. Inversive and linear congruential pseudorandom number generators in empirical tests. *ACM Trans. Modeling and Computer Simulation*, **7**(2):272–286, 1997.

[15] P.A. Lewis, A.S. Goodman, and J.M. Miller. A pseudo-random number generator for the System/360. *IBM Syst. J.*, **8** :136–146, 1969.

[16] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, **7**(1):3–30, 1998.

[17] Makoto Matsumoto. Mersenne twister home page. `http://www.math.keio.ac.jp/˜matumoto/emt.html`, 1998.

[18] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, Philadelphia, 1992.

[19] S.K. Park and K.W. Miller. Random number generators: good ones are hard to find. *Comm. ACM*, **31**:1192–1201, 1988.

[20] A. Varga. OMNeT++ Discrete Event Simulation System. `http://www.hit.bme.hu/phd/vargaa/omnetpp.htm`.

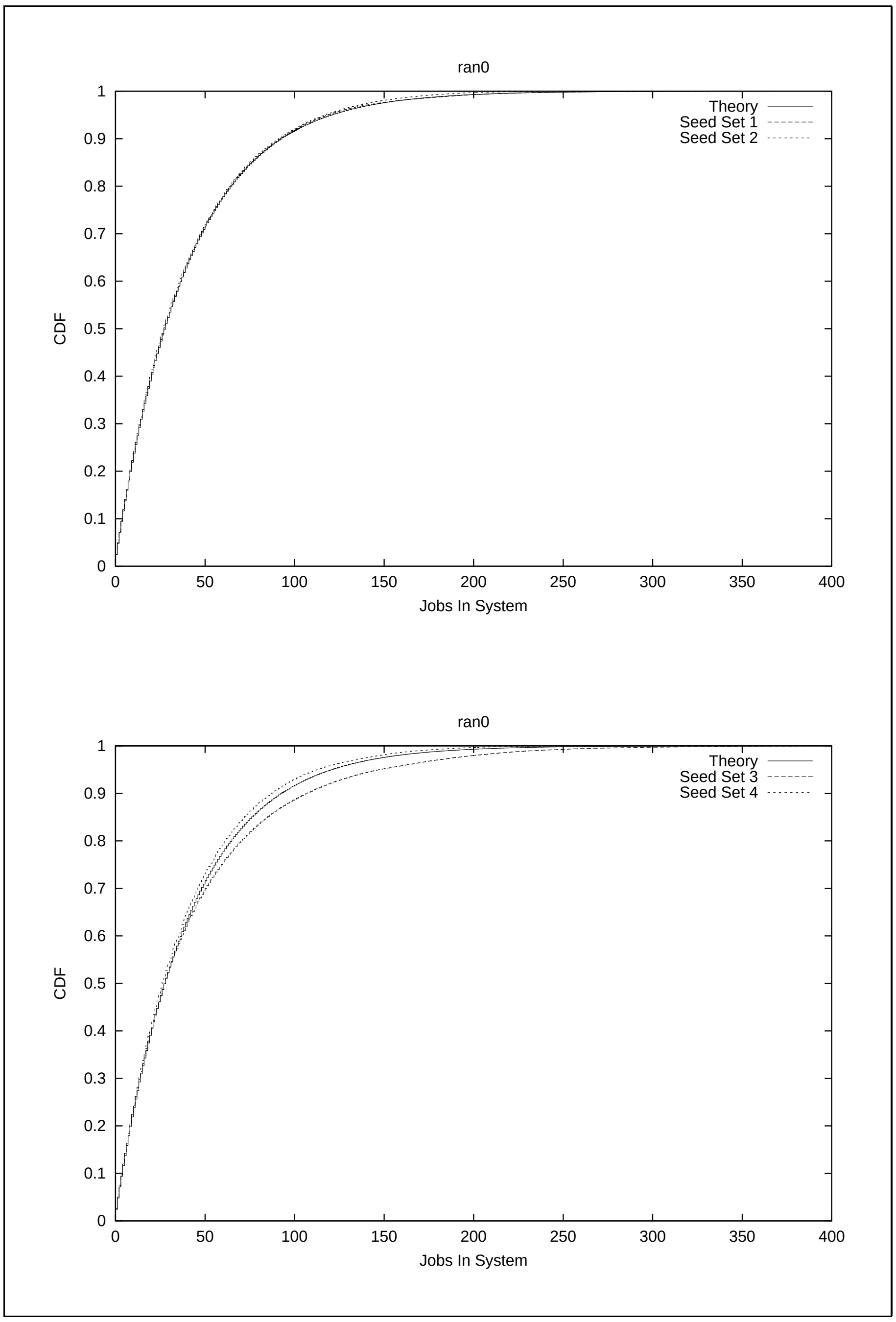


Figure 3: Comparison of the theoretical vs. empirical CDFs for the simulations with 'good' seeds (upper graphics) and 'bad' seeds.

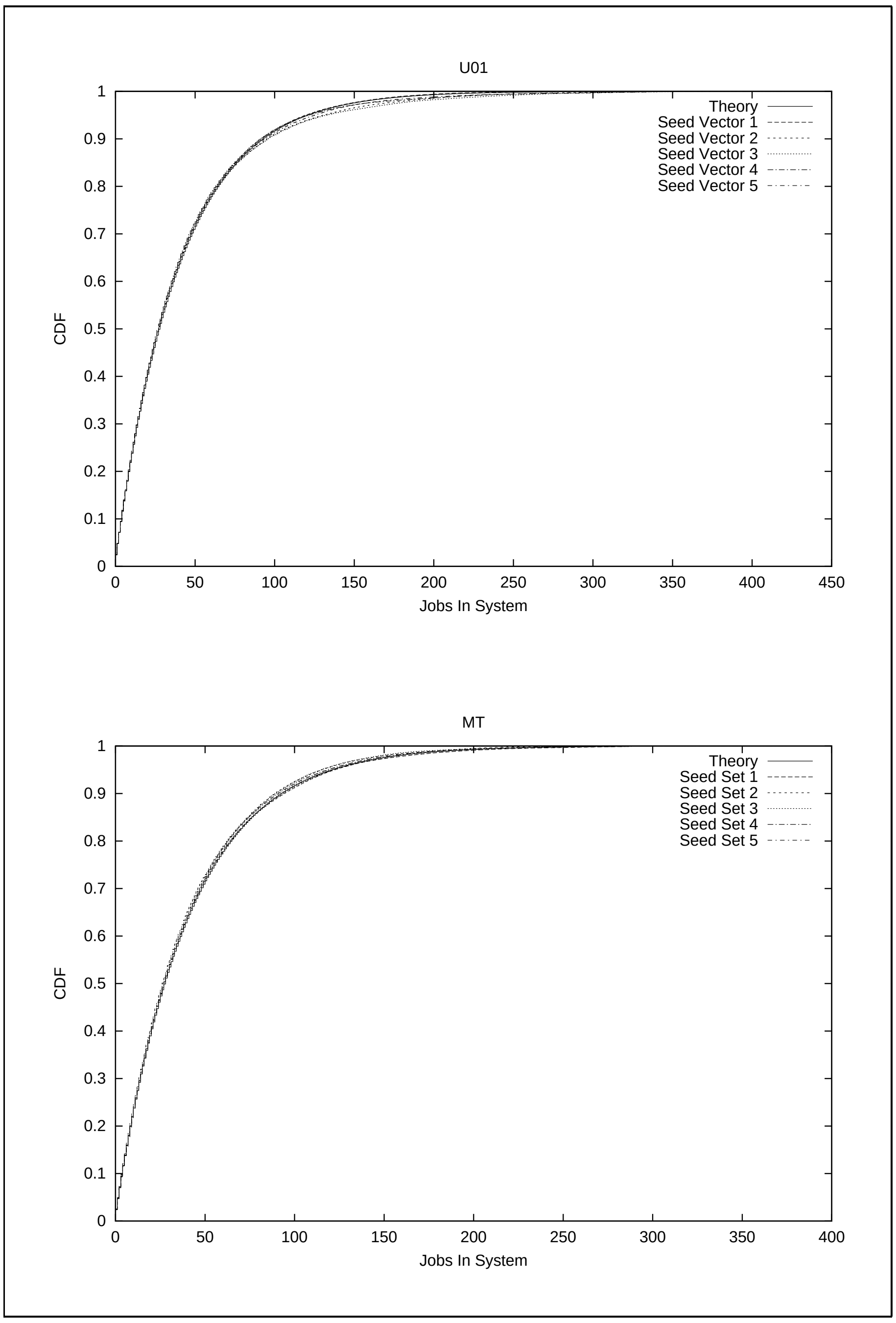


Figure 4: Comparison of the theoretical vs. empirical CDFs for five simulations using `RandU01` with automatic initial seeding (upper graphics), and the same number of simulations using `MT`.

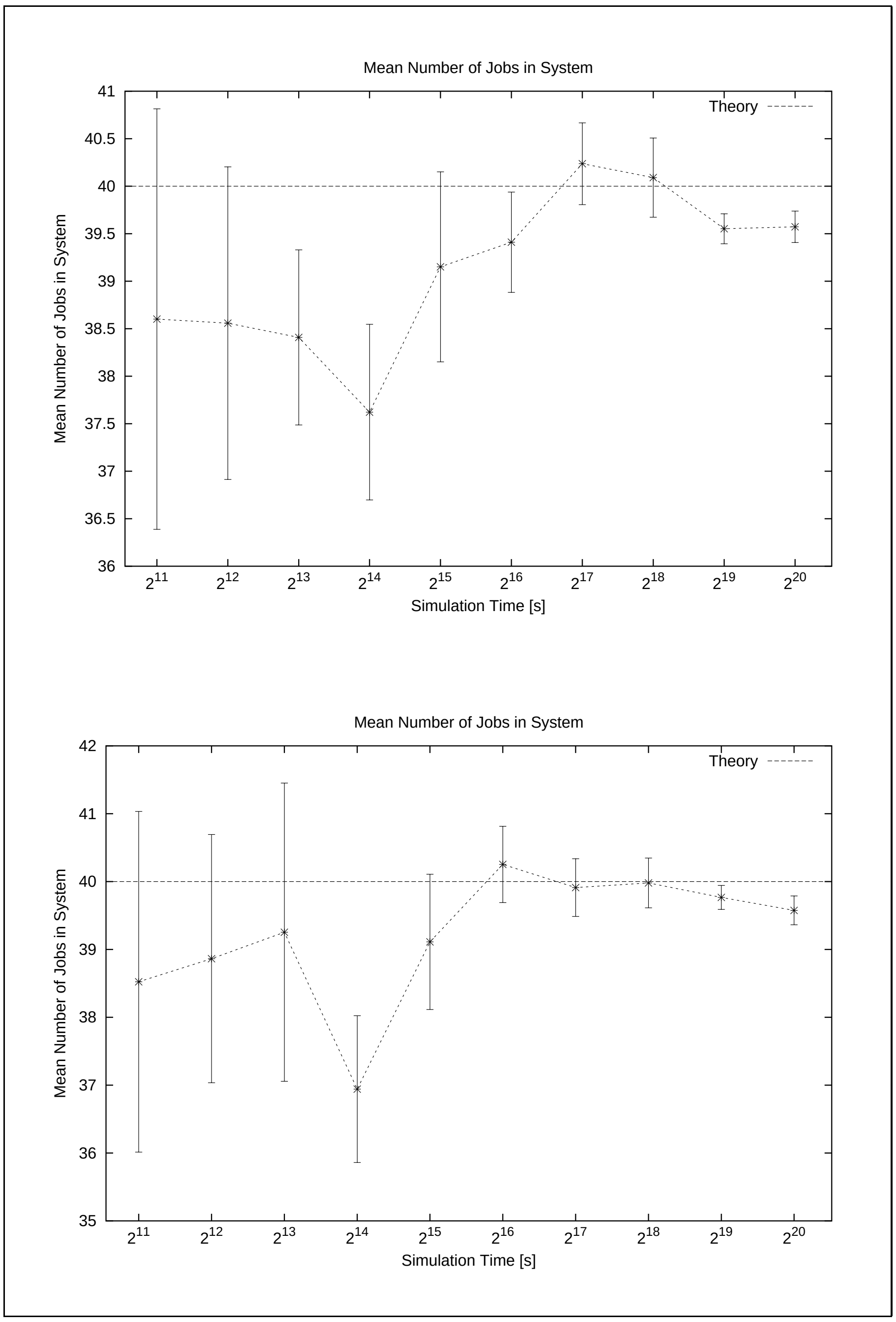
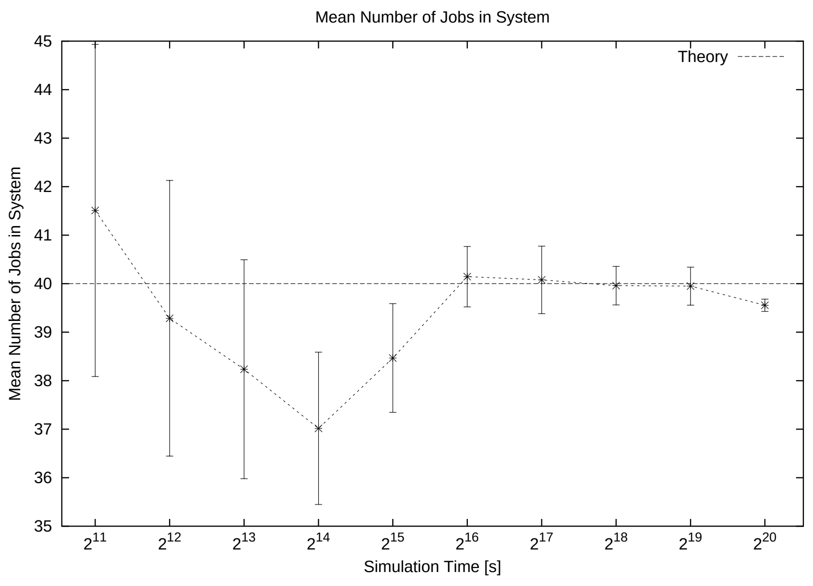


Figure 5: Mean values and 95% confidence intervals for the mean number of jobs in the system using `ran0`(two differnt simulations).



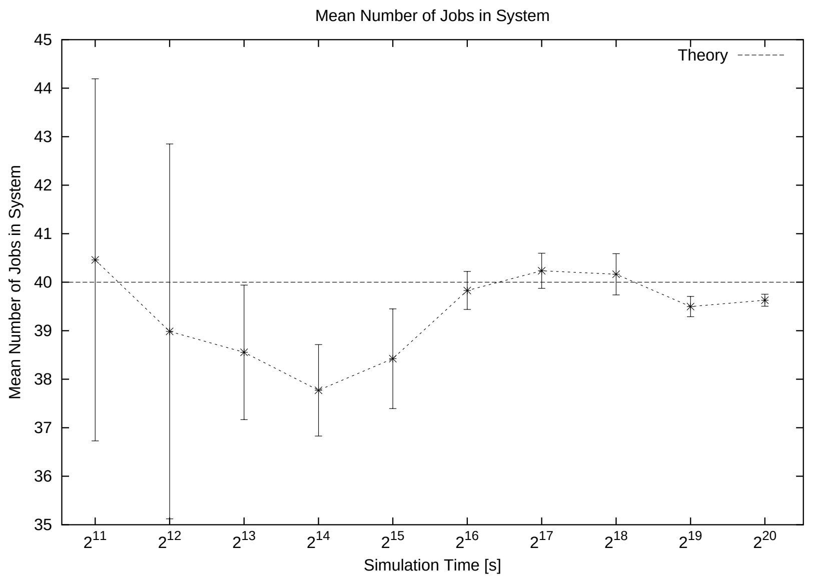


Figure 6: Mean values and 95% confidence intervals for the mean number of jobs in the system using `MT`(two differnt simulations).

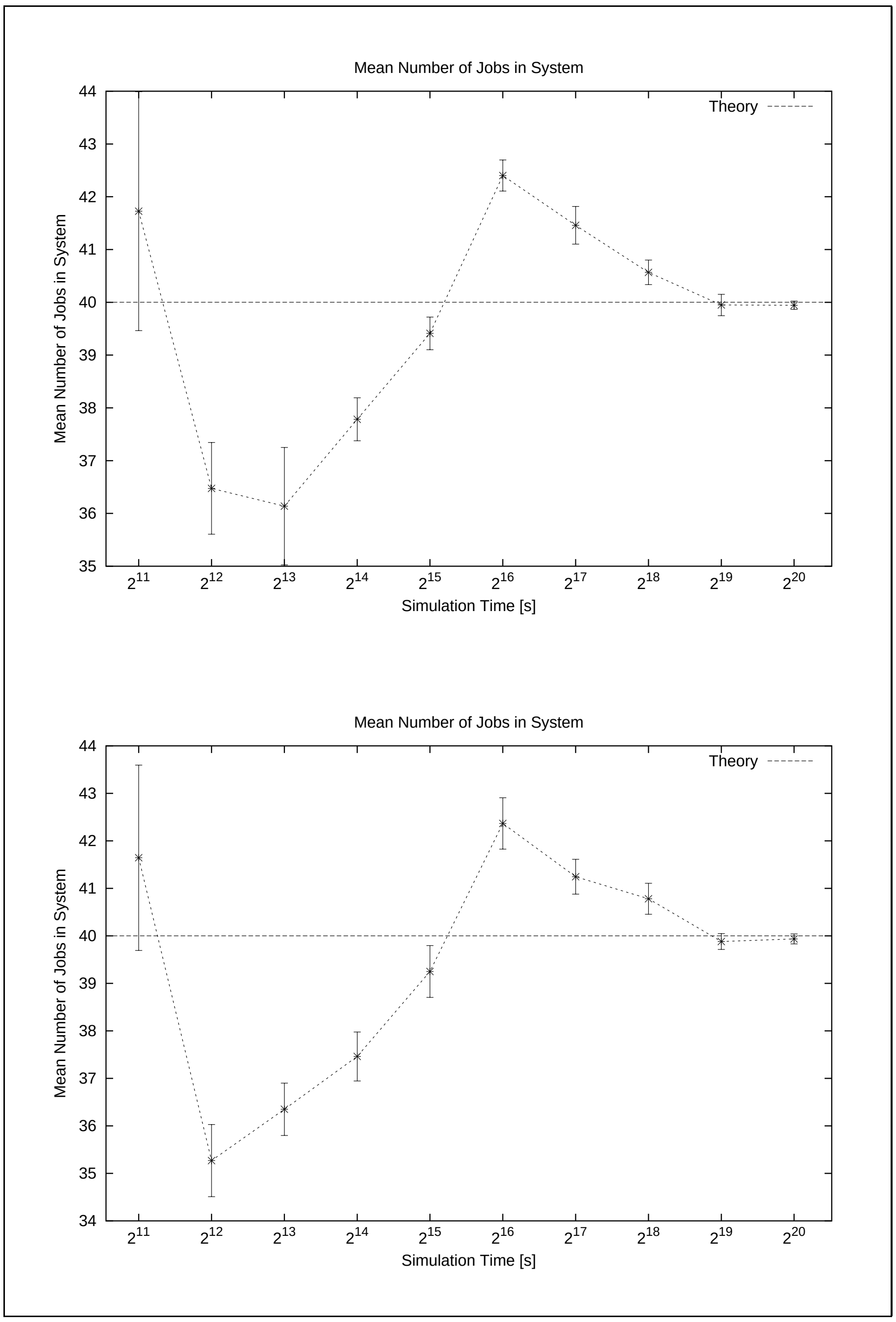


Figure 7: Mean values and 95% confidence intervals for the mean number of jobs in the system using `RandU01`(two differnt simulations).