

A simple OMNeT++ queuing experiment using parallel streams

Karl Entacher^{1,*}
Bernhard Hechenleitner¹
Stefan Wegenkittl²

¹ School of Telecommunications Engineering,
Salzburg University of Applied Sciences and Technologies
Schillerstraße 30, 5020 Salzburg, Austria

² Department of Mathematics,
University of Salzburg
Hellbrunnerstraße 34, 5020 Salzburg, Austria

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 a more modern generator which supports parallel streams. In addition we tested the behavior of parallel streams of the generators with stringent statistical tests for random numbers.

1 Introduction

OMNeT++ [1] is an object-oriented discrete event simulation system based on C++. It is primarily designed to simulate computer networks, multiprocessors 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

*Corresponding author. E-mail: karl.entacher@fh-sbg.ac.at

statistical synchronization, for which OMNeT++ provides explicit support. Further details on this issue can be found in the OMNeT++ on-line manual [1].

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 a more modern RNG which 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. In Section 6 we have additionally tested the RNGs using strong statistical tests. Such random number tests provide powerful methods to protect against unwanted behavior in simulations.

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 [2]. It was examined in more detail in Park and Miller [3] 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 [4, 5, 6, 7, 8] which produces pseudorandom integers via the recursion

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

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 [9].

This particular generator has widely been used and actual implementations are available from the Internet. See [10, 4, 5, 6, 7, 11, 12, 13, 3] 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 Simulator 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.

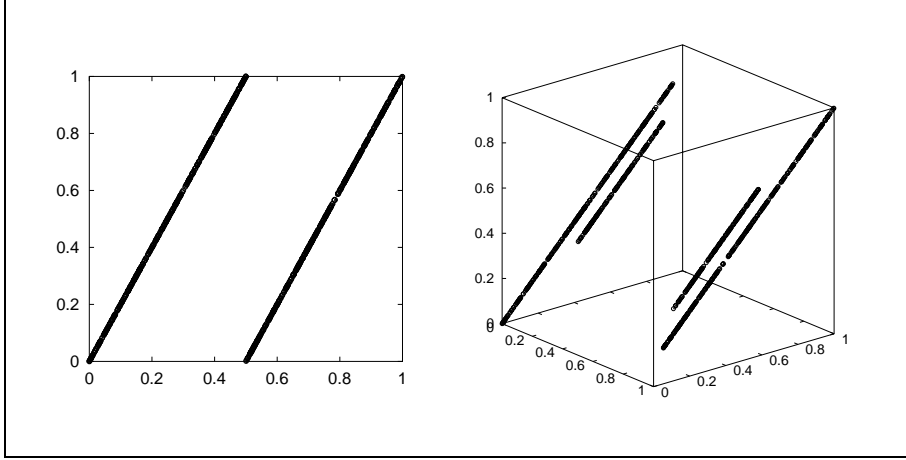


Figure 1: Correlations between two and three different streams of random numbers generated by the OMNeT++ RNG.

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 $x_{i,0} = i$, $i = 1, 2, \dots, k$ would result in the following k random number streams which are heavily correlated

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

These correlations can easily be shown from the vectors

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

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

In [14] an analysis of so-called long range correlations between parallel streams from the minimal standard RNG was performed. 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 correlations¹ of linear random numbers, see [15, 16, 14] 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 [4, 17, 7, 11, 8].

3 A Modern Generator

A modern object-oriented RNG which supports well-tested parallel streams of random numbers was implemented by L'Ecuyer et al. [12]. 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`.

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

¹The 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.

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.

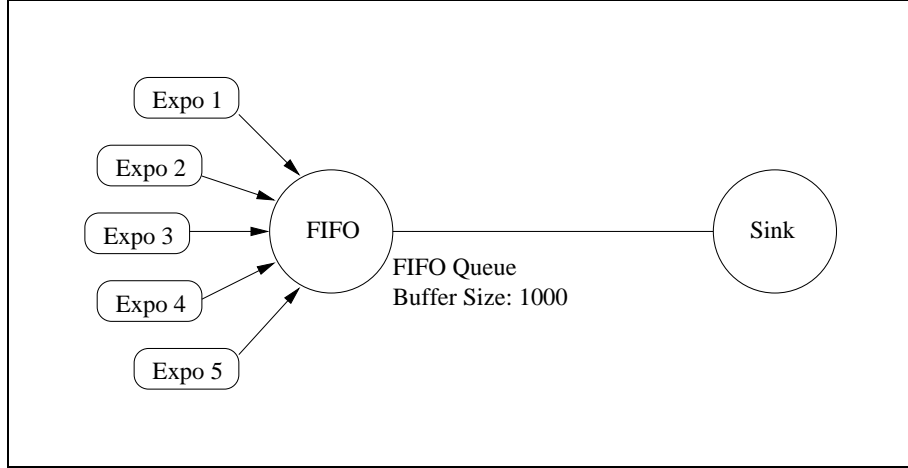


Figure 2: Simulation topology.

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** can be seen as a Markov Process with an average arrival rate of $\lambda = \frac{1\text{ job}}{41\text{ ms}}$. 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 $\lambda_{sum} = 5 \cdot \lambda = \frac{1\text{ job}}{8.2\text{ ms}}$. 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 $\mu = \frac{1\text{ job}}{8\text{ ms}}$. 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 [18, 6], the mean number of jobs in the system \bar{N} calculates as $\bar{N} = \rho/(1 - \rho)$. 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 $\bar{N} = 40.0004\text{ jobs}$. The distribution of the number n of jobs in the system is given by the geometric distribution $p(n) = (1 - \rho)\rho^n$.

	Seed Set 1	Seed Set 2	Seed Set 3	Seed Set 4
seed 1	1	934100682	1	1
seed 2	1808217256	558746720	2	634005912
seed 3	851767375	2081634991	3	634005911
seed 4	1328964554	144443207	4	2147483646
seed 5	1252999968	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 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 \bar{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.

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. The strong influence of the correlations within the streams for the 'bad' seeds can also be seen in Sect. 6.

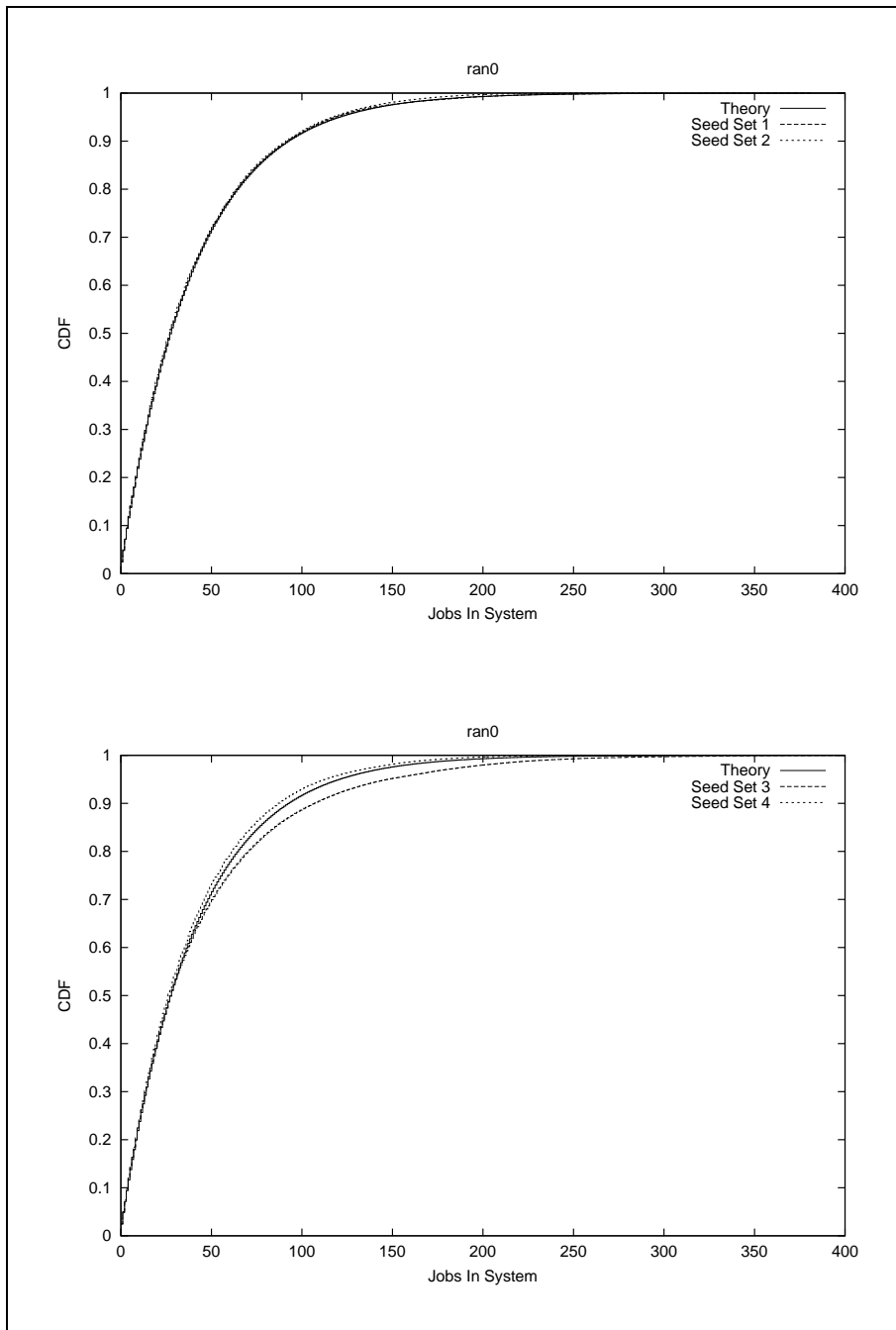


Figure 3: Comparison of the theoretical vs. empirical CDFs for the simulations with 'good' seeds (upper graphics) and 'bad' seeds.

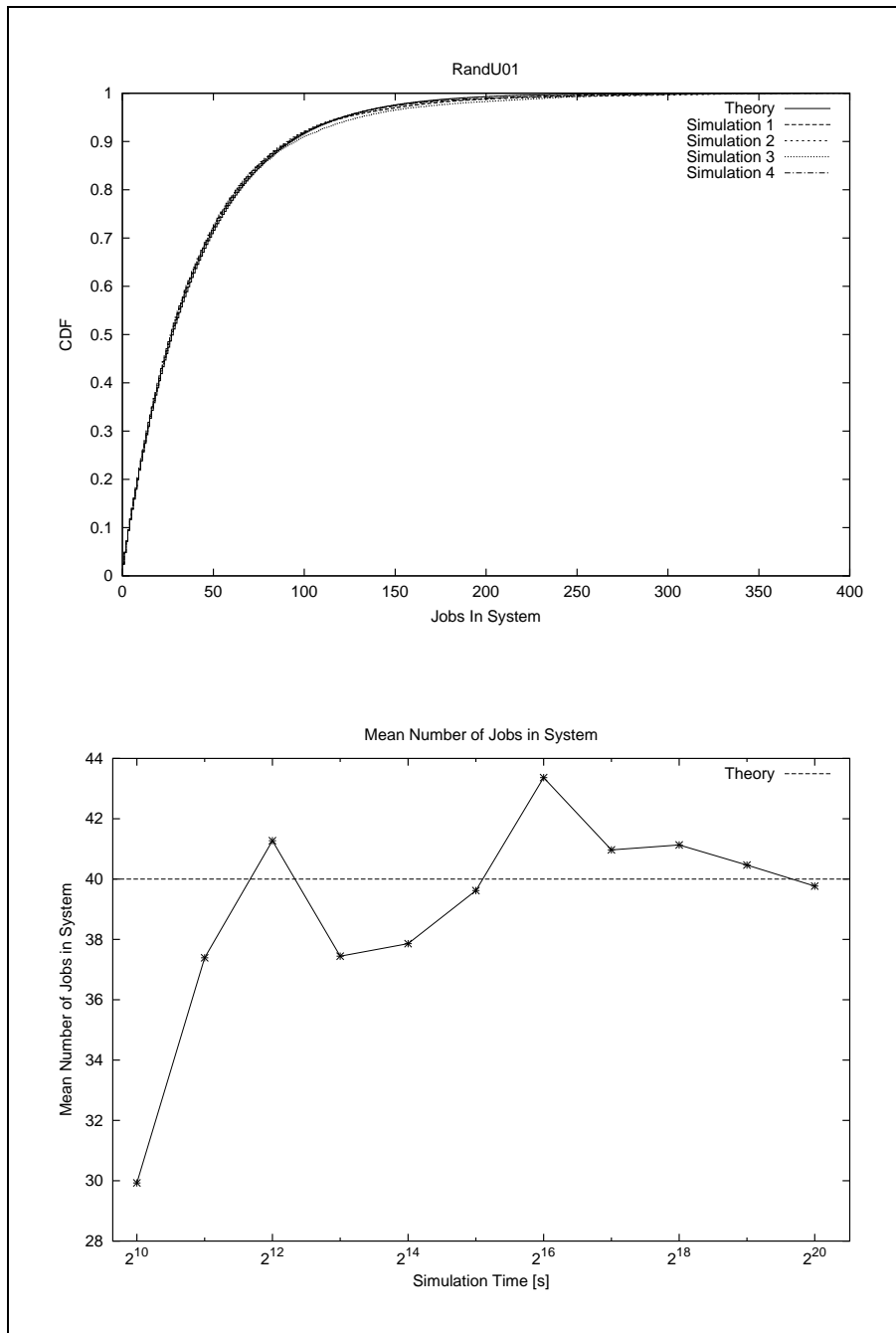


Figure 4: Comparison of the theoretical vs. empirical CDFs for four simulations using RandU01 with automatic initial seeding (upper graphics), and behavior of the mean values for simulations with increasing simulation time.

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 initial vectors $\{i, i, i, i, i, i\}$, $1 \leq i \leq 10$ for the *first* RNG objects. The average of the mean values of these simulations equals 40.1529 with a standard deviation of 0.4863 which results in a 99% confidence interval [39.6531, 40.6527] for \bar{N} .

The upper graphics in Figure 4 shows the empirical CDFs of our first four simulations. The obtained CDFs match the theoretic CDF very well (same for the remaining 6 simulations). In addition we carried out simulation with increasing simulation time $t_i = 2^i$ s, $10 \leq i \leq 20$. The lower graphics in Figure 4 shows the behavior of the mean values for the jobs in the system.

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, we manually applied 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. The simulation results we got from the bad seeding are even worse as those for `ran0` using Seed Set 3.

6 Tests for the Selection of Parallel Generators

Empirical (“black-box”) testing of a random number generator usually complements the preliminary theoretical analysis of its structural properties. The goal is to find evidence against the presumed hypothesis H_0 , that the random numbers are sampled from a sequence of independent random variables distributed uniformly on the unit interval $[0, 1)$. If no such evidence is found, the numbers are said to pass the test.

Applying batteries of statistical tests should help in pointing out applications which may suffer from the deterministic origin and the resulting regularities of the random numbers. For this reason, testing with respect to applications of a generator in parallel stochastic simulation will (in addition to an assessment of the quality of each single stream of random numbers)

stress the quality (i.e., uniformity) of the common empirical distribution of vectors constructed from the i 'th number in each stream.

As has been pointed out above, the situation faced by a user of simulation software is that she/he cannot be sure that the standard generator provided by the package is a modern type of generator that provides for high speed, good empirical quality and stability with respect to generation of parallel streams. In fact, such generators are still rare on the market. We refer the reader to the groups of L'Ecuyer [12] and Matsumoto [19] for public available generators with excellent properties.

If the quality of the generator is questionable however, we advise empirical testing. Here, we aim at doing prototype-simulation with known results and use the (computable) error as an indicator for the quality of the generator.

The following sections review a selection of appropriate test statistics and test designs. We also address the question of how to interpret the results and report on a sample study with respect to the queuing simulation above.

6.1 Empirical Tests

Among the test batteries available for public download, three prominent candidates are the Diehard Battery (<http://stat.fsu.edu/~geo/diehard.html>), the NIST Battery (<http://csrc.nist.gov/rng/>) and the forthcoming package of L'Ecuyer et. al. (<http://www.iro.umontreal.ca/~lecuyer/>). However, the test statistics and the parameter setting procedures available in these test suites are numerous and the correlation among the tests is not well examined. Moreover, certain tests in these batteries can be shown to be almost identical to each other (e.g. the serial test and the Approximate Entropy test in the NIST battery, see [20]) and besides wasting computing power, the parallel evaluation of these tests leads to statistical problems when trying to combine the significance-values obtained.

In our empirical studies [21, 13, 22, 23] we concentrated on one type of test instead, namely the family of overlapping serial tests. For these tests, the sample space (usually the d -dimensional unit cube $[0, 1)^d$) is partitioned into a set \mathcal{C} of classes and the number of samples in each such class is evaluated for a given total sample size n . Typically, partitioning is done by cutting out some c most significant bits of each pseudorandom number. Comparing the expected number of hits with the actual number of hits by means of a divergence statistic [24] gives an indicator for the divergence of the empirical distribution and the theoretical one presumed by the Null Hypothesis H_0 .

One limitation of the standard overlapping serial test is the computer memory needed to store the array of counters for the number of samples in each class ($2^{c \cdot d}$ integer counters are needed for the standard setup, e.g.) limiting

the practical use to low dimensions (around 7). Several concepts to extend the range of applicability have been proposed including sparse tests [25] and gambling tests [22, 23]. Here we employ the latter concept which is build on a mapping from d -dimensional bit-vectors to 3 distinct states (“win”, “loss”, “skip”, motivated by a gambling strategy) as follows: Choosing d even and $t = d/2$, we count the number of ones in the first $d - 1$ coordinates of the vector. If this number is smaller than t , the vector is mapped to state “skip”. Otherwise, the last bit in the vector determines the state (“win” if the bit is 1, “loss”, otherwise). The number of vectors mapped to each of the 3 states is counted in the sample. A modified divergence statistic, is applied to the resulting 3 dimensional counter vector and gives an indicator for the quality of approximation of the theoretical distribution by the samples from the generator.

6.2 Test Design

Since our aim here is to search for possible correlations among pseudorandom numbers in 5 different parallel streams, we first construct a stream $(y_i)_{i \geq 1}$ of real numbers from the 5 parallel streams under consideration by putting $y_{5(j-1)+r} = x_j^r$, ($1 \leq r \leq 5$), $j \geq 1$ where x_j^r is the j 'th pseudorandom number in the r 'th stream.

For the Load Test, we cut out the $c = 4$ most significant bits in the binary representation of each y_i and concatenate $d = 5$ successive such 4 bit-blocks to a $4 \cdot d$ -bit vector $z_i = (z_i^1, z_i^2, \dots, z_i^{4 \cdot d})$, where the first four bits $(z_i^1, z_i^2, \dots, z_i^4)$ are those of y_i , the next four bits $(z_i^5, z_i^6, \dots, z_i^8)$ are those of y_{i+1} , and so on. Note the overlapping 5-tuples of successive y_i used here. For a sample size n , the Load Test thus uses at most $\lfloor n/5 \rfloor + 1$ pseudorandom numbers from each stream, see [13] for details.

For the Gambling Test, we cut out the $c = 8$ most significant bits in the binary representation of each y_i and concatenate them to a single stream $(z_j)_{j \geq 1}$ of bits. Consequently, z_j is a bit among the 8 most significant bits in $y_{\lfloor (j-1)/8 \rfloor + 1}$, $j \geq 1$. From this bit stream we construct overlapping successive bit vectors of length d , $d \in \{32, 64, 128, 256\}$, and submit them to the Gambling Test. The Gambling Test thus uses at most $\lfloor n/(8 \cdot 5) \rfloor + 1$ pseudorandom numbers from each stream.

The tests (Load Test and Gambling Test) both result in a single real number T , the value of the divergence statistic between the expected theoretical and the empirical number of hits in each class/state. The distribution of this outcome T is known under the Null-Hypothesis: T is distributed chi-square with g degrees of freedom, where $g = 16^d - 16^{d-1}$ for the Load Test and $g = 2$ for the Gambling Test.

The whole test is now repeated N times ($N = 32$ for the Load Test and

$N = 16$ for the Gambling Test) based on N consecutive samples of pseudo-random numbers. The resulting values T_1, \dots, T_N give a good summary on the performance of the generator: Under H_0 , these values should behave like a sample from a chi-square distribution. This will be assessed using a second level test, namely, a two-sided Kolmogorov-Smirnov statistic giving a single goodness-of-fit value.

6.3 Interpretation of Results

Since most generators under consideration for parallel stochastic simulation will be deterministic (i.e., computer programs simulating randomness), every reasonable statistical test for the "true" random origin of such numbers will reject the generator provided the sample size n is chosen large enough (where the sample size needed will also depend on the dimension d).

With this general result in mind we should interpret results of an empirical assessment procedure as follows: given a test setup which resembles (to some extent) the application scenario (in our case, using parallel streams of random numbers) and sample sizes n used in the stochastic simulation, a rejection by a statistical test indicates that the ability to pretend randomness of the generator is likely to be not sufficient for yielding unbiased simulation results. We do not recommend using such generators.

On the other hand, one can never certify the goodness of a generator using statistical tests. In view of the numerous types of correlations that might be present in the output of a generator we recommend to use a large number of sample sizes n and dimensions d and to try to get the generator to its limits in order to get an impression of its suitability for a given simulation problem. Critical simulations should always be verified using a different type of generator, of course.

6.4 Sample Study

We applied the Load Test with sample sizes $n \in \{2^{18}, 2^{19}, \dots, 2^{24}\}$ and dimension $d = 5$ and the Gambling Test with sample sizes $n \in \{2^{22}, 2^{23}, \dots, 2^{26}\}$ and dimensions $d \in \{32, 64, 128, 256\}$ to the generators given in Table 2.

The results of the Load tests are shown in Figure 5. The bars denote the value of the Kolmogorov-Smirnov (KS) statistic. For 32 repetitions of the Load Test, a one-sided 99% confidence interval for KS is given by $[0, 1.59]$, see e.g. [26]. Bars with higher values are colored almost black and indicate a rejection of the generator for the given sample size and confidence. For smaller values of KS, the bars follow a gray level coding ranging from white (corresponding to a KS value of zero) to light gray (corresponding to a KS

RNG	Seeds
G1	ran0 with 'good' Seed Set 1 from Table 1
G2	ran0 with 'good' Seed Set 2
B1	ran0 with 'bad' Seed Set 3
B2	ran0 with 'bad' Seed Set 4
U01_1	RandU01 with default seeds
U01_2	RandU01 with the same seeds as described in Section 5.2

Table 2: Short names of the generators in the empirical tests.

value of 1.59). The number of pseudorandom numbers taken from each stream are approximately 1677722 (for $n = 2^{18}$) and 107374183 (for $n = 2^{24}$). For some of the generators under examination this will result in reusing portions of the pseudorandom numbers if $n \geq 2^{23}$, a fact which has to be taken into account when interpreting the results.

The results show that U01_1 and U01_2 show inconspicuous behavior for all sample sizes, the well-seeded small linear congruential generators G1 and G2 are working well until reusing of pseudorandom numbers starts. Badly seeded generators B1 and B2 completely fail the test (Note, that the values have been truncated at 2 in the plot).

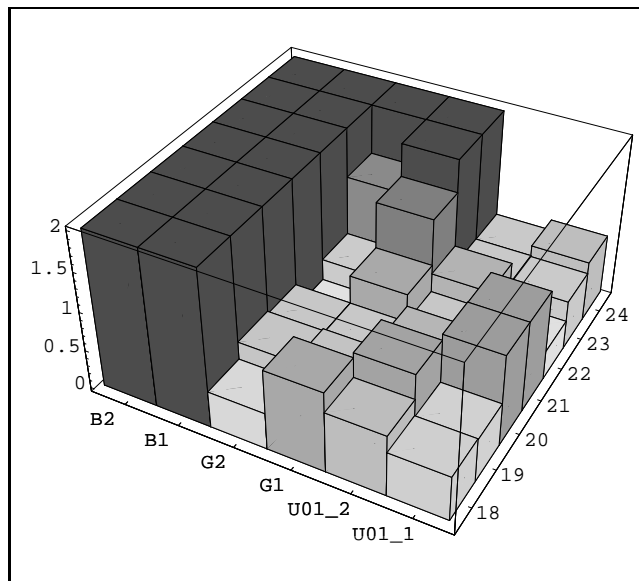


Figure 5: Values of the two-sided Kolmogorov-Smirnov statistic of the Load Test in dimension 5. The y-axis denotes the dual logarithm of the sample size n .

Table 3 gives the numerical values of the KS statistic for the $N = 16$ repetitions of the Gambling Tests in dimensions $d = 32, 64, 128, 256$ for sample size $n = 2^{22}$, Table 4 shows the corresponding results for $n = 2^{26}$. The corresponding number of pseudorandom numbers used from each stream are 131072 (for $n = 2^{22}$) and 2097152 (for $n = 2^{26}$), so that almost no pseudorandom number is used more than once in the test.

The largest possible value 4 of the KS statistic with $N = 16$ is (almost ever) assumed by the generators B1 and B2 even for the relatively small sample size $n = 2^{22}$ indicating that these generators completely fail to give the desired number (8) of bits in 5 parallel streams. The other generators generally behave well, yielding KS values in the 99% confidence interval $[0, 1.57)$ of the KS statistic with $N = 16$ repetitions.

Generator	$d = 32$	$d = 64$	$d = 128$	$d = 256$
G1	1.320	0.851	0.448	0.888
G2	0.910	0.768	0.669	0.916
B1	4.000	4.000	4.000	4.000
B2	4.000	4.000	4.000	4.000
U01_1	0.887	0.925	1.010	0.669
U01_2	0.902	1.300	0.515	0.976

Table 3: KS values of the Gambling Test with dimension d , $N = 16$ and $n = 2^{22}$.

Generator	$d = 32$	$d = 64$	$d = 128$	$d = 256$
G1	0.390	0.867	0.567	1.550
G2	0.926	0.679	0.945	1.100
B1	4.000	4.000	4.000	4.000
B2	4.000	4.000	4.000	4.000
U01_1	1.190	1.430	1.000	0.446
U01_2	0.837	0.911	0.927	0.680

Table 4: KS values of the Gambling Test with dimension d , $N = 16$ and $n = 2^{26}$.

It can be seen that there is a clear consistency between the performance of a generator in the Load- and Gambling Tests and the performance of the same generator in our simple simulation study. In general, this consistency heavily depends on the similarity of the test design and the application scenario.

The Load Test from the family of overlapping serial tests has the advantage of being able to detect *any* deviation from H_0 provided that dimension d and

sample size n are chosen high enough. Since every deterministic generator will fail in dimensions d higher than some critical dimension d_{crit} because of the regularities induced by the deterministic origin of the numbers, our main goal in empirical testing should be to choose d in a range that is relevant for the desired application. If such a d is too high to be implemented using the Load Test Design, one is left to using tests that distribute their discriminative power among more dimensions at the cost the generality (i.e., the ability to detect any possible correlations). An example is the family of Gambling Tests which have been constructed with the typical limitations of deterministic generators in mind and have proven powerful even for generators with large moduli [23].

7 Conclusions

Today's rapid development of new protocols and mechanisms in the world of networking makes it essential to study their interactions and effects by the means of simulation before implementing them in the real world. Simulation tools can be of great help for a better understanding and for testing those mechanisms in a variety of different possible environments. Of course simulation tools are only helpful if one can trust their outputs. In this paper we used the widely deployed simulation tool OMNeT++ to examine bad effects of correlations of parallel streams of random numbers on simulation outputs.

Our simulations have shown that the outputs of certain simulation scenarios may be wrong, if the default built-in RNG of OMNeT++ (`ran0`) is used in combination with carelessly selected initial seeds for parallel streams of random numbers. Further examinations, which were not presented within the scope of this paper, showed that in fact it is rather hard to find sets of seeds for the parallel streams which produce expected simulation results.

The integration of a modern RNG (`RandU01`) significantly improved the simulation outputs of our simulation scenario. When correctly using this RNG, i.e. when using the recommended automatic seeding procedure, the simulation outputs match the expected output very close, independent from the initial seed vector used for the first RNG objects.

Empirical testing was shown to clearly forecast the performance of a generator in the simulation study provided the dimension of the test is similar to that of the simulation problem. Besides the well-known overlapping serial tests which are limited in their practical use because of the curse of dimensionality, a dimension reduction technique called Gambling Tests can be employed if the simulation problem is supposed to be high dimensional.

Acknowledgments

The first two authors were supported by the Austrian Science Fund (FWF) Grant S8311-MAT, the third author by the FWF-Grant S8303-MAT.

References

- [1] A. Varga. OMNeT++ Discrete Event Simulation System. <http://www.hit.bme.hu/phd/vargaa/omnetpp.htm>.
- [2] 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.
- [3] S.K. Park and K.W. Miller. Random number generators: good ones are hard to find. *Comm. ACM*, **31**:1192–1201, 1988.
- [4] G.S. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*, volume 1 of *Springer Series in Operations Research*. Springer, New York, 1996.
- [5] R. Jain. *The art of computer systems performance analysis*. John Wiley & Sons, New York, 1991.
- [6] A.M. Law and W.D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, 2 edition, 1991.
- [7] P. L'Ecuyer. Uniform random number generation. *Ann. Oper. Res.*, **53**:77–120, 1994.
- [8] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, Philadelphia, 1992.
- [9] L.P. Devroye. *Non-Uniform Random Variate Generation*. Springer, New York, 1986.
- [10] 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/>.
- [11] P. L'Ecuyer. Software for uniform random number generation: Distinguishing the good and the bad. In *Proceedings of the 2001 Winter Simulation Conference*, 2001.
- [12] 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.

- [13] 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.
- [14] K. Entacher. Further Remarks on Long-Range Correlations among Linear Congruential Random Numbers. In G. Okša, R. Trobec, A. Uhl, M. Vajteršic, R. Wyrzykowski, and P. Zinterhof, editors, *Proceedings of the international Workshop PARALLEL NUMERICS ParNum 2000*, pages 69–80, Bratislava, Slovakia, 2000.
- [15] A. DeMatteis and S. Pagnutti. Long-range correlations in linear and non-linear random number generators. *Parallel Comput.*, **14**:207–210, 1990.
- [16] 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.
- [17] P. Hellekalek and G. Larcher (eds.). *Random and Quasi-Random Point Sets*, volume **138** of *Lecture Notes in Statistics*. Springer, Berlin, 1998.
- [18] L. Kleinrock. *Queueing Systems. Volume I: Theory*. John Wiley & Sons, New York, 1975.
- [19] Makoto Matsumoto. Mersenne twister home page. <http://www.math.keio.ac.jp/~matumoto/emt.html>, 1998.
- [20] S. Wegenkittl. Entropy estimators and serial tests for ergodic chains. *IEEE Transactions on Information Theory*, **47**(6):2480–2489, Sep 2001.
- [21] S. Wegenkittl. On empirical testing of pseudorandom number generators. In G. De Pietro, A. Giordano, M. Vajteršic, and P. Zinterhof, editors, *Proceedings of the international workshop Parallel Numerics '95*. CEI-PACT Project, WP5.1.2.1.2, 1995.
- [22] S. Wegenkittl. Gambling tests for pseudorandom number generators. *Mathematics and Computers in Simulation*, **55**(1–3):281–288, 2001.
- [23] S. Wegenkittl and M. Matsumoto. Getting rid of correlations among pseudorandom numbers: Discarding versus tempering. *ACM Trans. Modeling and Computer Simulation*, **9**(3):282–294, 1999.
- [24] S. Wegenkittl. A generalized ϕ -divergence for asymptotically multivariate normal models, 2002. To appear in *Journal of Multivariate Analysis*, Vol. 84, No. 1, 2003, available via IDEALFirst.
- [25] P. L'Ecuyer, R. Simard, and S. Wegenkittl. Sparse serial tests of uniformity for random number generators. Accepted for publication in SISC, 2002.
- [26] J. Hartung, B. Elpelt, and K. H. Klösener. *Statistik*. R. Oldenburg, Munich, 9th edition, 1993.