

# On Shortcomings of the ns-2 Random Number Generator

Bernhard Hechenleitner<sup>°</sup> and Karl Entacher<sup>°</sup>

<sup>\*</sup>Salzburg Research, 5020 Salzburg, Austria<sup>\*</sup>

<sup>°</sup>University of Applied Sciences and Technologies, 5020 Salzburg, Austria

Bernhard.Hechenleitner@salzburgresearch.at

{Karl.Entacher, Bernhard.Hechenleitner}@fh-sbg.ac.at

**Keywords:** random number generation, correlated random numbers, Mersenne Twister, ns-2, M/D/1

## Abstract

The ns-2 is a widely used network simulation tool which implements a fairly old and weak random number generator (RNG). As the RNG component is used by a variety of other components, it can be seen as one of the most important core components of the ns-2. In this paper we explore weaknesses of this RNG and demonstrate its shortcomings in context with a simple simulation example (M/D/1) which produces severely wrong simulation results when this RNG is used in combination with specific seeds. We incorporate the modern Mersenne Twister RNG into the ns-2 and show how the insensitivity of this RNG regarding to the initial seeds leads to significant improvements in the simulation outputs and that this modern RNG has no bad effects with respect to the performance of random number generation.

## 1 INTRODUCTION

The ns-2 [18] is a very popular network simulation tool. Its development began in 1989 and was funded by DARPA and NFS. Currently, the ns-2 is used by about 600 institutes in 50 countries. As the ns-2 is an open-source project, it is easy to integrate new components or alter current implementations of components within the object-oriented architecture of ns-2.

The ns-2 random number generator (RNG) component implements the well-known “minimal standard” generator, which was originally suggested for the IBM System/360 by Lewis, Goodman and Miller in 1969 [13]. It was examined in more detail in Park and Miller [20] and further on in several other studies on random number generation.

This generator is a multiplicative linear congruential type [7, 11, 19] which produces pseudorandom integers via the re-

ursion

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

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 the transformation  $u_n = x_n/m$ , non-uniform distributions by different transformation methods [4].

Generators of this type have widely been used and actual implementations are available from the Internet. See [1, 7, 10, 11, 14, 20, 21] and the Internet for references, empirical tests and implementations in free and commercial software, as for example Resampling Stats ([www.resample.com](http://www.resample.com)), Numerical Recipes ([www.nr.com](http://www.nr.com)), the mathematical software MATLAB ([www.mathworks.com](http://www.mathworks.com)), the IMSL Libraries, or the simulation software ACSL ([www.mga.com](http://www.mga.com)), SIMAN/Arena, Slam II and AweSim ([www.pritsker.com](http://www.pritsker.com)).

In this paper we demonstrate important shortcomings of this RNG. In the following section some basic properties of this RNG are described. We explore the problem of correlations when wrong seeds are used to produce different random number streams. In Section 3 we use simple simulation examples to exhibit the effects of these correlations in practice and show that the current RNG implementation of ns-2 can lead to im-  
petuously wrong simulation results. Section 4 describes how the integration of a more modern and robust generator into ns-2 can significantly improve the simulation results and shows a performance comparison between the original and the new RNG. Section 5 concludes the paper.

## 2 SHORTCOMINGS OF THE RNG COMPONENT OF NS-2

The RNG component of ns-2 is used by a variety of other components of ns-2, e.g. by components dealing with the generation of random variables of different kinds (uniform, exponential, pareto, normal, lognormal, hyper-exponential, ...), the generation of synthetic traffic (exponential, pareto, web-like, real-audio-like, ...), the dropping/marking mechanisms of a RED queue, the random movement of a wireless node or the

---

<sup>\*</sup>This work is partly funded by the EU IST project AQUILA.

error models of a link. The RNG component can therefore be seen as one of the most important core components of ns-2 which many other components rely on.

The current RNG component implements a rather old mechanism for the generation of random numbers, hence many basic properties of this RNG are well known. A first shortcoming probably is the period of  $p = 2^{31} - 2$  which is far too short for current applications, especially when parallel streams are used in simulations. Such parallel streams are often needed in ns-2 simulations, e.g. for multiple traffic streams or server queues. The generation of different random number streams is usually realized by applying different seeds  $x_0$ .

A closer look at the seeding segment within the ns-2 RNG source-code `rng.cc` exhibits the following comments:

```
// NEEDSWORK: should be a way to set seed to
// PRDEF_SEED_SOURCE
.
.

// NEEDSWORK: should we throw out known bad seeds?
// (are there any?)
.
.
// Toss away the first few values of heuristic seed.
// In practice this makes sequential heuristic seeds
// generate different first values.
// How many values to throw away should be the subject
// of careful analysis. Until then, I just throw away
// ``a bunch''. --johnh
```

These comments indicate that in the current implementation of the RNG component the seeding procedure is insecurely implemented. Especially the question “are there any bad seeds?” can be answered right away with yes!

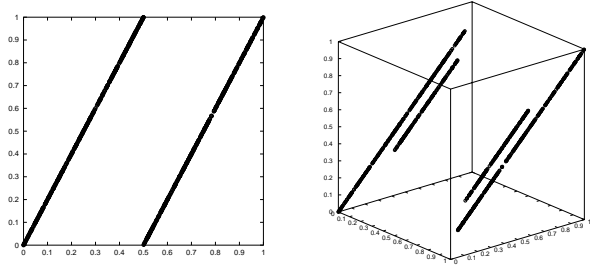
Here is a simple example: suppose two streams of uniformly distributed random numbers should be generated using this RNG. A naive user may simply apply the seeds  $x_{0,1} = 1$  and  $x_{0,2} = 2$  to initialize these two streams  $x_{n,i}$ ,  $n \geq 0$ ,  $i = 1, 2$ . The recursion (1) easily leads to

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

If one wants to analyze correlations between these streams, the vectors  $\vec{x}_n := (x_{n,1}, x_{n,2})$ ,  $n \geq 1$ , which equal

$$\vec{x}_n \equiv a^n \cdot (1, 2) \pmod{m} \quad (3)$$

have to be studied. 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  $\{n \cdot (1, 2) \pmod{m} : 1 \leq n \leq m-1\}$ . 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 two lines only, see Fig. 1. A similar observation can be made with a larger number of different streams generated with seeds  $x_{0,i} = i$ ,  $i \geq 1$ ; e.g. for



**Figure 1.** Correlations between two and three different streams of random numbers generated by the ns-2 RNG

$i = 1, 2, 3$  the right plot in Fig. 1 shows the corresponding structure produced by the vectors  $\vec{x}_n = (x_{n,1}, x_{n,2}, x_{n,3}) \in \{n \cdot (1, 2, 3) \pmod{m} : 1 \leq n \leq m-1\}$ .

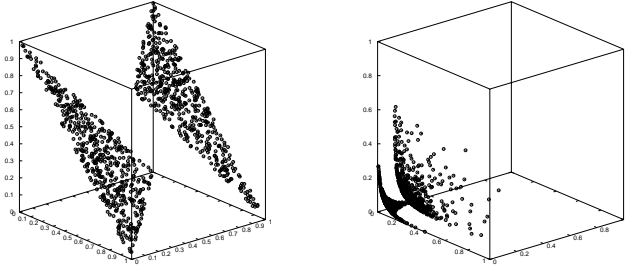
The graphics in Fig. 1 exhibit strong correlations between the streams obtained from these simple seeds. With “real” independent random number streams, such vectors should obviously produce no regular structures. In the next section we will show that such streams result in bad empirical results in a simple simulation example.

But not only these simple seeds produce strongly correlated streams of random numbers. There are many possible seed combinations where the corresponding streams are strongly correlated. Further simple examples are seeds  $x_{0,i} = k \cdot i$ ,  $i \geq 1$ ,  $k \in \mathbf{Z}$ , 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<sup>1</sup> of linear random numbers, see [2, 3, 5, 6, 15] and the references given in there. As an example, the seeds  $x_{0,1} = 1$ ,  $x_{0,2} = 634005911$ ,  $x_{0,3} = 1513477735$  divide the period of this RNG into three blocks of length  $p/3$ . Fig. 2 visualizes the vectors  $(x_{n,1}, x_{n,2}, x_{n,3})$ ,  $n \geq 1$  which are seeded with the three values just described and again a conspicuous structure is visible. The vectors are situated on two hyperplanes in the three-dimensional unit cube (left graphics). When for example exponentially distributed random number streams instead of uniformly distributed ones are produced, the same vectors exhibit the structure in the right graphics<sup>2</sup> in Fig. 2.

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 [7, 8, 11, 12, 19].

<sup>1</sup>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.

<sup>2</sup>The average of each exponential random number stream was set to 0.1.

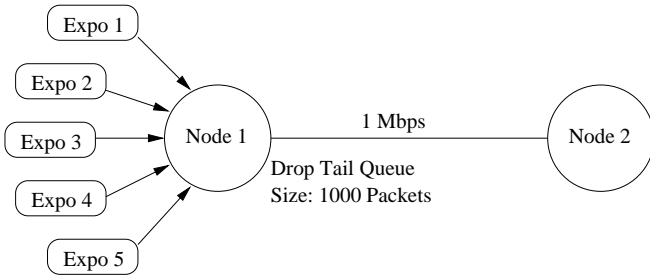


**Figure 2.** Correlations between three uniform streams (left plot) and three exponential streams (right plot) generated by the ns-2 RNG

### 3 SIMPLE SIMULATION EXAMPLES

Although there could be many possible topologies and situations where correlations between different RNG objects could lead to wrong simulation results, a very simple example was chosen for examinations of the ns-2 RNG. The intention of this was to highlight that the bad effects of the current RNG implementation of the ns-2 already manifest in simple simulation scenarios and that complex simulation topologies are not needed to produce wrong simulation results.

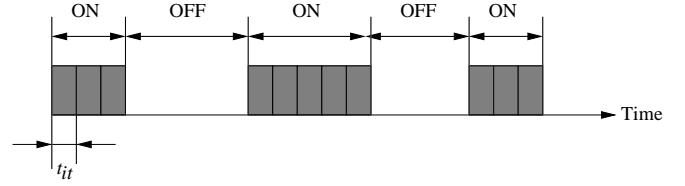
For the examination of shortcomings of the ns-2 RNG the simulation topology shown in Fig. 3 was chosen. Traffic streams of 5 exponential traffic generators (Expo 1 to Expo 5) are aggregated at Node 1. The output interface of Node 1 to Node 2 is configured with a Drop Tail queue of size 1000 packets. The two nodes Node 1 and Node 2 are connected through a link with a capacity of 1 Mbps.



**Figure 3.** Simple simulation topology

Fig. 4 shows the model of the used exponential traffic generator component of ns-2. The implementation of this component can be found in the ns-2 source-code file `expo.cc`. The exponential traffic generator is embedded in the ns-2 architecture as the component `Application/Traffic/Exponential`.

The exponential traffic generator allows to set the mean of an exponentially distributed ON-interval and the mean of an ex-



**Figure 4.** Model of the exponential traffic generator of ns-2

ponentially distributed OFF-interval. During the ON-interval, constant bit rate (CBR) traffic with user definable “internal” rate<sup>3</sup> and packet size is generated. For the following simulations, each of the exponential traffic generators Expo 1 to Expo 5 had the same parameter settings. In order to compare the simulation model to the well-known M/D/1 model, the parameters of the exponential traffic generators were set to the following values.

During each ON-interval only 1 packet of size 1000 Bytes was generated. The “internal” rate was set to 100 Gbps, therefore the “internal” transmission time for handing over one packet to the node is  $t_{it} = \frac{8000 \text{ Bits}}{100 \text{ Gbps}} = 0.08 \mu\text{s}$ . The mean of the exponentially distributed OFF-interval was set to 41 ms, which is 512500 times the value of  $t_{it}$ . Therefore,  $t_{it}$  can be disregarded<sup>4</sup> for the calculation of the average arrival rate  $\lambda$  of one traffic flow. Considering this simplification, each traffic flow of Expo 1 to Expo 5 can be seen as a Markov Process with an average arrival rate of  $\lambda = \frac{1 \text{ packet}}{41 \text{ ms}} = \frac{8000 \text{ Bits}}{0.041 \text{ s}} = 0.195 \text{ Mbps}$ . 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 has an average arrival rate of  $\lambda_{sum} = 5 \cdot \lambda = \frac{1 \text{ packet}}{8.2 \text{ ms}} = 0.976 \text{ Mbps}$  at the outbound interface of Node 1. The link between Node 1 and Node 2 has a service rate of  $\mu = 1 \text{ Mbps}$ . In combination with packets of fixed size (1000 Bytes) and a queue of size 1000 packets, Node 1 can be seen as a M/D/1/1001 queuing system with a utilization factor of  $\rho = \frac{\lambda_{sum}}{\mu} = \frac{0.976 \text{ Mbps}}{1 \text{ Mbps}} = 0.976$ .

Referring to the theoretical M/D/1 model in [9], the mean queue length  $\bar{q}$  calculates as

$$\bar{q} = \frac{\rho}{1 - \rho} - \frac{\rho^2}{2 \cdot (1 - \rho)} \quad (4)$$

Considering a M/D/1 queuing system with a utilization factor of  $\rho = 0.976$ , the queue of such a system would therefore

<sup>3</sup>This rate is not the rate of a real traffic flow on a link, as the exponential traffic generator is directly attached to a node. It is just a mathematical value to calculate the average number of packets during an ON-interval.

<sup>4</sup>The CDF of exponentially distributed arrivals with  $\lambda = \frac{1 \text{ packet}}{41 \text{ ms}}$  shows a value of  $1.951 \cdot 10^{-6}$  for an inter-arrival time of  $0.08 \mu\text{s}$ , which means that the probability of inter-arrival times  $\leq t_{it}$  is negligible. Therefore the error of not being able to generate such small inter-arrival times because of the chosen packet size may be disregarded.

have an average length of  $\bar{q} = 20.488$  packets.

Regarding the simulations, each exponential traffic generator used its own RNG object. Different sets of seeds were used for seeding the 5 RNG objects. For each simulation, the average queue length  $\bar{q}$  was calculated. The simulation time was set to 7200s, which is 878049 times the average inter-arrival time of 8.2ms between two packets of the aggregated Markov Process. Table 1 shows the results of the simulations using the original RNG component of ns-2. The corresponding seed sets are given in the table as well. Note that those seed sets which result in strong correlations among the different random number streams when using a minimal standard RNG are denoted as “bad” seeds whereas those seed sets which are supposed not to produce correlated streams are denoted as “good” seeds. A Kruskal-Wallis hypothesis test [10] for homogeneity of the distributions of the queue lengths<sup>5</sup> of all four simulations results in a *p-value* of 0.00219 and therefore shows significant differences of the four distributions. The Kruskal-Wallis chi-square statistic is  $T = 14.599$ , which is greater than the critical value  $11.345 = \chi_{3,0.99}^2$ . Therefore, the null hypothesis of homogeneity of the four distributions must be rejected at level  $\alpha = 0.01$ . Fig. 5 shows the CDFs of the queue lengths at the outbound interface of Node 1 for the different simulations using the original RNG component of ns-2. As can be seen, the CDFs obtained by using seed sets 2 and 4 (the “bad” seed sets) differ significantly from the CDFs obtained by using seed sets 1 and 3 (the “good” seed sets) which show a similar progression. It is to mention that the unaware user might not consider using different RNG objects for the different traffic streams. By default, the user doesn’t have to care about random number generation and therefore may set up the simulation topology without configuring any RNG objects. In this case the default RNG object<sup>6</sup> is used to generate all necessary random numbers<sup>7</sup>. Being unaware of different RNG objects, a simulation of the shown topology would result in an average queue length of  $\bar{q} = 20.997$  using the original RNG implementation of ns-2<sup>8</sup>.

#### 4 A MODERN GENERATOR

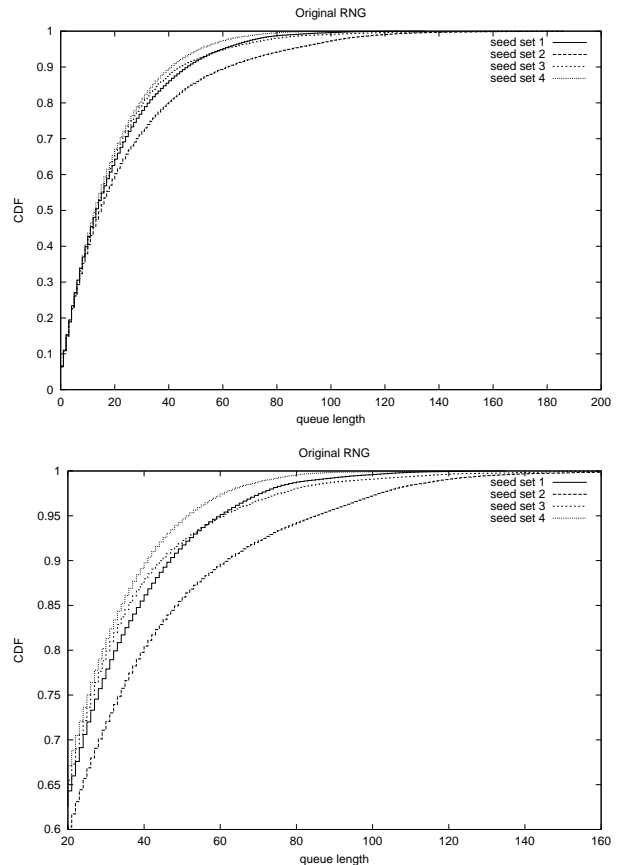
The Mersenne Twister [16] is an up-to-date random number generator. It is to state that there are other modern types of RNGs available, for examples see [12] and the references given in there. The Mersenne Twister (MT) was chosen since this

<sup>5</sup>The distribution of the queue length of a single simulation run was ascertained by calculating the probability of each individual queue length, multiplying this value by the according queue length and scaling the result by a multiplication using the total number of determined queue lengths.

<sup>6</sup>This default RNG object is initiated when ns-2 is started and it is seeded with the value 1.

<sup>7</sup>Hence, as only one random number stream serves all requests for random numbers, no bad effects caused by correlations between several random number streams can appear in this scenario.

<sup>8</sup>Considering that all other simulation parameters are kept unmodified.



**Figure 5.** CDFs of the queue lengths when using the original RNG component of ns-2 in combination with different seed sets (the lower plot shows an enlarged view)

generator is very fast, has a huge period ( $2^{19937} - 1$ ) and is known to be theoretically and empirically well tested. Source code of the MT RNG for several programming languages is freely available at the Mersenne Twister home page [17]. For the integration of the MT RNG into the ns-2 RNG component, the standard MT C-codes (`mt19937int.c` and `mt19937-2.c`) were used. The two ns-2 source-code files `rng.h` and `rng.cc` were altered in such a way that for the calculation of unsigned integers uniformly distributed among  $0$  to  $2^{32} - 1$  as well as for the calculation of real numbers uniformly distributed on the  $[0, 1)$ -interval the new MT code was used. After modifying the `rng.h` and `rng.cc` files, the ns-2 has to be recompiled by using the `make` command<sup>9</sup>. The outputs of the new MT RNG

<sup>9</sup>The authors would like to mention that possibly not all cross-linkages between the new MT RNG component and the other components of the ns-2 have been detected and therefore a tidy integration of the MT RNG component should be done by the ns-2 development team. Concerning the simulation studies in this paper, all necessary modifications to integrate the MT RNG have carefully been examined and are limited to the two source-code files `rng.h` and `rng.cc`.

**Table 1.** Results of simulations using the original RNG component of ns-2

Simulation	RNG seed set 1 ('good' seeds)	RNG seed set 2 ('bad' seeds)	RNG seed set 3 ('good' seeds)	RNG seed set 4 ('bad' seeds)
seed 1	1973272912	1	934100682	1
seed 2	1822174485	2	558746720	634005912
seed 3	1998078925	3	2081634991	634005911
seed 4	678622600	4	144443207	2147483646
seed 5	999157082	5	513791680	151347773
$\bar{q}$	19.451	24.288	19.374	17.573

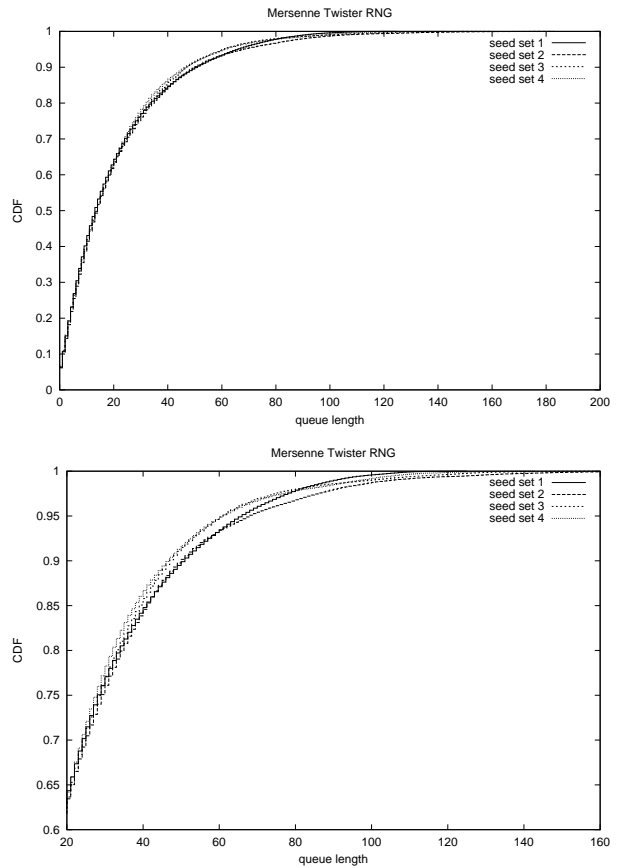
component of ns-2 were verified using the reference outputs published at the Mersenne Twister home page.

Table 2 shows the simulation results when applying the new MT RNG component to the ns-2. The same seed sets as for the original RNG were used. Note that a MT RNG is insensitive to different initial seeds and therefore arbitrary seeds may be applied.

The hypothesis of homogeneity of the different queue length distributions of the four simulations was confirmed by the Kruskal-Wallis test which calculated a *p-value* of 0.76088 and a Kruskal-Wallis chi-square statistic of  $T = 1.167$ .

Fig. 6 shows the CDFs of the queue lengths at the outbound interface of Node 1 for the different simulations using the MT RNG component within ns-2. In contrast to the CDFs obtained with the original RNG component of ns-2 the CDFs which result by using the new MT RNG component all show a very similar progression independent of the used seed sets. Therefore no seed set needs to be denoted as “bad” seeds or “good” seeds.

In order to see the influence of the MT RNG with regard to the speed of random number generation, a performance test between the original RNG and the MT RNG was executed. Each RNG was incorporated into ns-2 as the RNG component and then a `tc1`-script was used to produce a certain amount of random numbers on the  $[0, 1)$ -interval using one stream of random numbers. Each test was repeated 8 times and the mean time for the generation of the random numbers was calculated. Table 3 shows the results of the performance test. It can be seen that both RNGs take about the same amount of time to produce their random numbers if they are embedded in ns-2 and if the generation of random numbers is controlled via a `tc1`-script. By increasing the amount of numbers to be generated, the MT RNG performs slightly better. The tests were performed on an unstressed PC working with SuSE 7.1 Linux operating system in console mode, an Intel Celeron 550 MHz CPU and 128 MB of RAM.



**Figure 6.** CDFs of the queue lengths when using the Mersenne Twister RNG component within ns-2 in combination with different seed sets (the lower plot shows an enlarged view)

**Table 2.** Results of simulations using the Mersenne Twister RNG component within ns-2

Simulation	MT RNG seed set 1	MT RNG seed set 2	MT RNG seed set 3	MT RNG seed set 4
seed 1	1973272912	1	934100682	1
seed 2	1822174485	2	558746720	634005912
seed 3	1998078925	3	2081634991	634005911
seed 4	678622600	4	144443207	2147483646
seed 5	999157082	5	513791680	151347773
$\bar{q}$	20.259	21.227	20.120	19.803

**Table 3.** Performance comparison of original RNG and Mersenne Twister RNG when embedded in ns-2

Amount of numbers	Original RNG Time [ms]	MT RNG Time [ms]
$2^{15} = 32768$	2700	2631
$2^{16} = 65536$	5366	5257
$2^{17} = 131072$	10744	10470
$2^{18} = 262144$	21231	20970
$2^{19} = 524288$	43254	42020
$2^{20} = 1048576$	86458	84499
$2^{21} = 2097152$	172681	167480

## 5 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 have examined the RNG of the widely used network simulator ns-2 and showed severe weaknesses. We have shown that the currently implemented RNG can lead to significantly wrong simulation results even when a very simple simulation scenario is used. By integrating the modern Mersenne Twister RNG we showed that the deployment of this RNG produces correct simulation results and therefore it is necessary to incorporate a robust RNG into the important network simulation tool ns-2.

## 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] A. DeMatteis and S. Pagnutti. Long-range correlation analysis of the Wichmann-Hill random number generator. *Statistics and Computing*, **3**:67–70, 1993.
- [4] L.P. Devroye. *Non-Uniform Random Variate Generation*. Springer, New York, 1986.
- [5] 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.
- [6] K. Entacher. Parallel streams of linear random numbers in the spectral test. *ACM Transactions on Modeling and Computer Simulation*, **9**(1):31–44, 1999.
- [7] G.S. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*, volume 1 of *Springer Series in Operations Research*. Springer, New York, 1996.
- [8] P. Hellekalek and G. Larcher (eds.). *Random and Quasi-Random Point Sets*, volume **138** of *Lecture Notes in Statistics*. Springer, Berlin, 1998.
- [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.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.
- [14] G. Marsaglia. The structure of linear congruential sequences. In S. K. Zaremba, editor, *Applications of Number Theory to Numerical Analysis*, pages 248–285. Academic Press, New York, 1972.
- [15] M. Mascagni. Parallel linear congruential generators with prime moduli. *Parallel Computing*, **24**(5–6):923–936, 1998.
- [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] S. McCanne and S. Floyd. UCB/LBNL/VINT Network Simulator - ns (version 2). <http://www.isi.edu/nsnam/ns/>, April 1999.
- [19] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, Philadelphia, 1992.
- [20] S.K. Park and K.W. Miller. Random number generators: good ones are hard to find. *Comm. ACM*, **31**:1192–1201, 1988.
- [21] B.D. Ripley. The lattice structure of pseudo-random number generators. *Proc. Roy. Soc. London Ser. A*, **389**:197–204, 1983.